Lecture Outline

Reminders to self:

- ❑ Turn on lecture recording to Cloud
- ❑ Turn on Zoom microphone

- Last Lecture
  - Continued K-Maps (3- and 4-variable & don't cares in)
  - Started binary adders (Half-adder, part of full-adder)
- Today's Lecture
  - Continue binary adders and subtracters
    - Finish full adder
    - Ripple carry adder and subtracter
    - Carry look-ahead enhancements

Handouts and Announcements

- Announcements
  - Homework Problems – no new assignments
  - Homework Reminders
    - Problems 2-4 and 4-1 due: 11:59pm Thursday 2/2
    - Problems 5-1 and 5-2 due: 11:25am Monday 2/6
  - Participation Quiz 4
    - Based on Lecture 9 (last lecture). 15min limit after you start.
    - Available 11:15am today, due 11:15am tomorrow, but also available 24hrs more with late penalty
  - Read for Friday: pages 193, 199-203

Handouts and Announcements

- Announcements
  - Mini-Exam 2 Reminder
    - Available 5pm Monday 2/6 through 5:00pm Tuesday 2/7
    - Due in Carmen PROMPTLY at 5:00pm on 2/7
    - Designed to be completed in ~36 min, but you may use more
    - When planning your schedule:
      - I recommend building in 10-15 min extra
      - To allow for downloading exam, signing and dating honor pledge, saving solution as pdf, and uploading to Carmen
    - I also recommend not procrastinating
  - Exam review topics available on Carmen
  - Sample Mini-Exams 1 and 2 from Au20 also available

Full Adder

- Add two binary bits, X_i and Y_i; and carry bit C_i
- $X_i + Y_i + C_i = S_i, C_{i+1}$ (regular addition here)
- $S_i = \sum m(1,2,4,7)$
- $C_{i+1} = \sum m(3,5,6,7)$
- K-map for S_i

X_i	Y_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

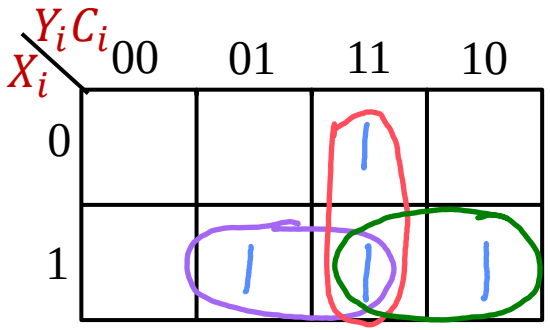
X_i \ Y_iC_i	00	01	11	10
0	0	1	0	1
1	1	0	1	0

- Four groups of 1 - the four minterms are already minimal
- For S_i each minterm has an odd number of 1s
- $S_i = X_i \oplus Y_i \oplus C_i$

Ran out of time here.
Review before continuing.

Full Adder

- Add two binary bits, X_i and Y_i; and carry bit C_i
- $X_i + Y_i + C_i = S_i, C_{i+1}$ (regular addition here)
- $S_i = \sum m(1,2,4,7)$
- $C_{i+1} = \sum m(3,5,6,7)$
- K-map for C_{i+1}



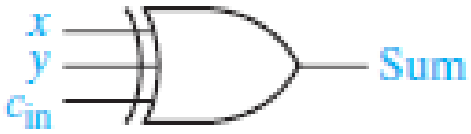
X_i	Y_i	C_i	S_i	C_{i+1}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

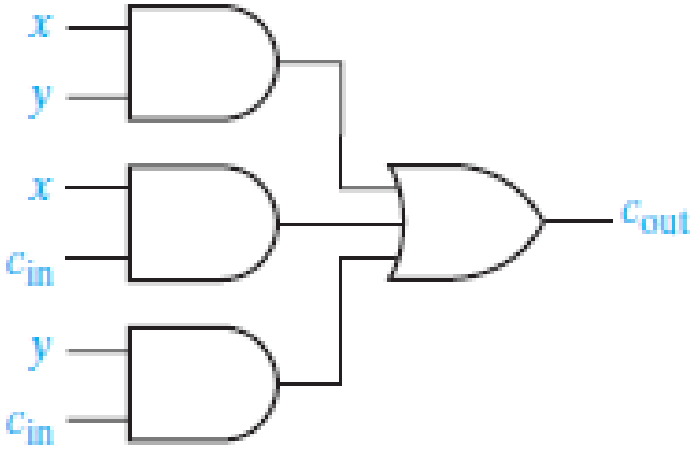
- Groups of 2, 1 variable is eliminated
- $C_{i+1} = X_iY_i + X_iC_i + Y_iC_i$

Sheet of paper (it's in the wind)

Ripple Carry Adder

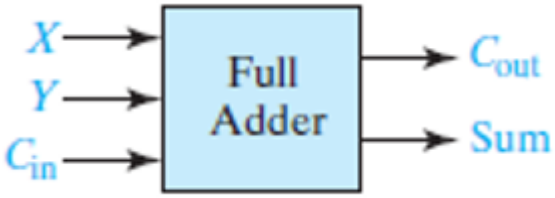
- $S_i = X_i \oplus Y_i \oplus C_i$
- $C_{i+1} = X_iY_i + X_iC_i + Y_iC_i$
- The circuits for each of the i bits are:





Not unique: other full adders exist

- The Carry-out circuit has two levels of logic
  - Two gate-delays for the C_{out} signal to be stable after x, y and c_{in} are applied
- In block diagram form, each Full Adder block contains both the Sum and Carry-out circuits



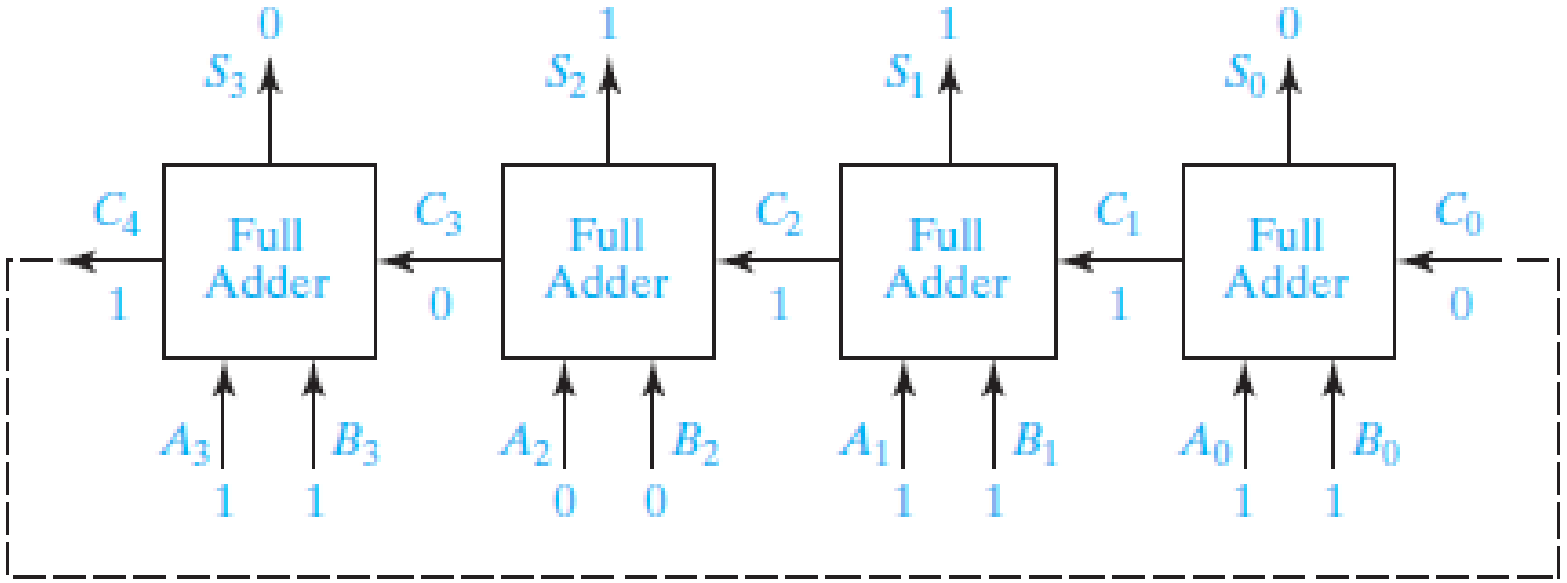
Four-bit Adder

- In principle, one could make a four-bit adder directly from a truth table using a brute force approach
  - Add $B_3B_2B_1B_0$ to $A_3A_2A_1A_0$
  - Need to allow for a carry-in bit, C_0 Why?
    - Nine inputs
    - Five outputs: $S_3S_2S_1S_0$ and C_4 (Carry out)
  - Truth table will have $2^9 = 512$ rows
  - This approach very difficult, and logic circuit to implement very complex
- Alternate approach
  - Use multiple instances of the full-adder block
  - This type of approach - using multiple instances of smaller functional blocks - is standard in digital design
  - Allows reliable design of complex systems to perform complex tasks

Ripple Carry Adder

- To make a four-bit adder use four full adders
  - Add $B_3B_2B_1B_0$ to $A_3A_2A_1A_0$, bit-by-bit
  - With the carry-out from the i-th bit becoming the carry-in of the $i+1$ bit
  - Figure shows addition of 1011 and 1011
  - Result is 0110 with a carry of 1

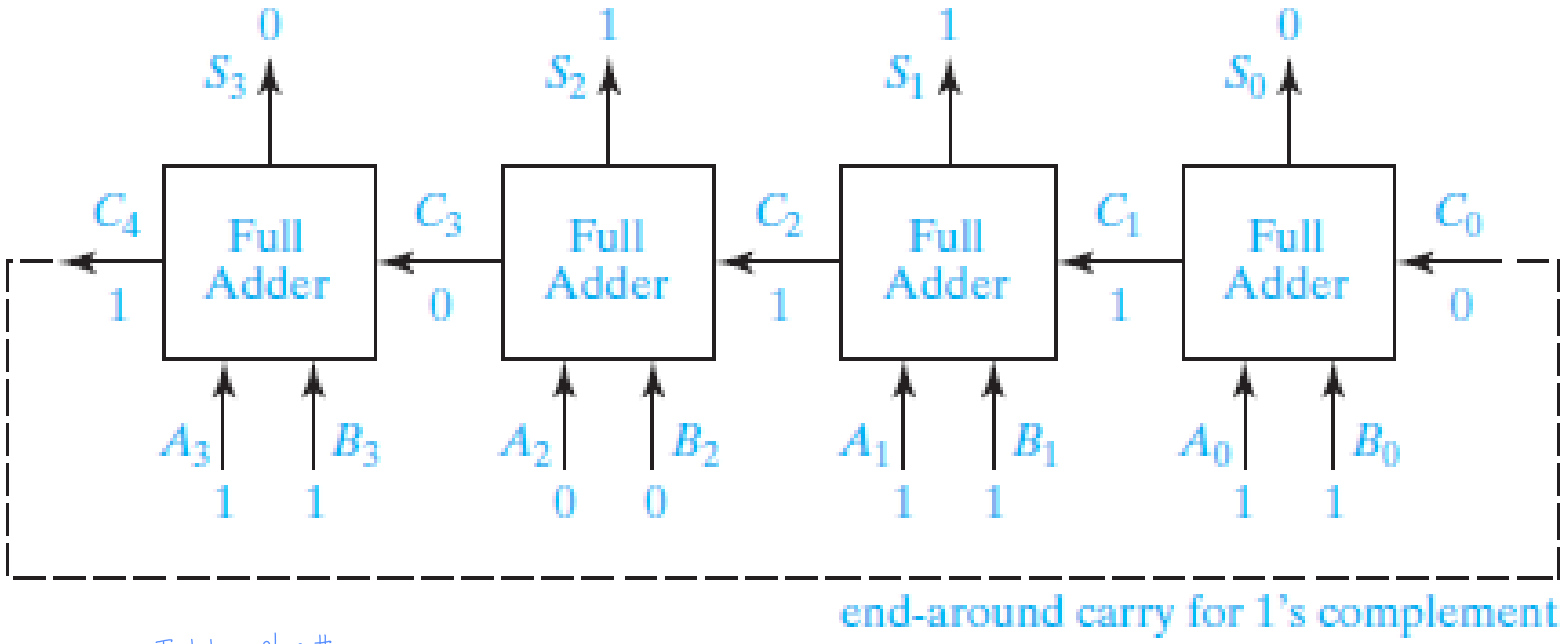
Question: What can you say about the result if this is 1's or 2's complement addition?



- Dashed line wire is only needed for: end-around carry for 1's complement
- For 2's complement the carry out of bit 4 is discarded, so wire not needed

Ripple Carry Adder

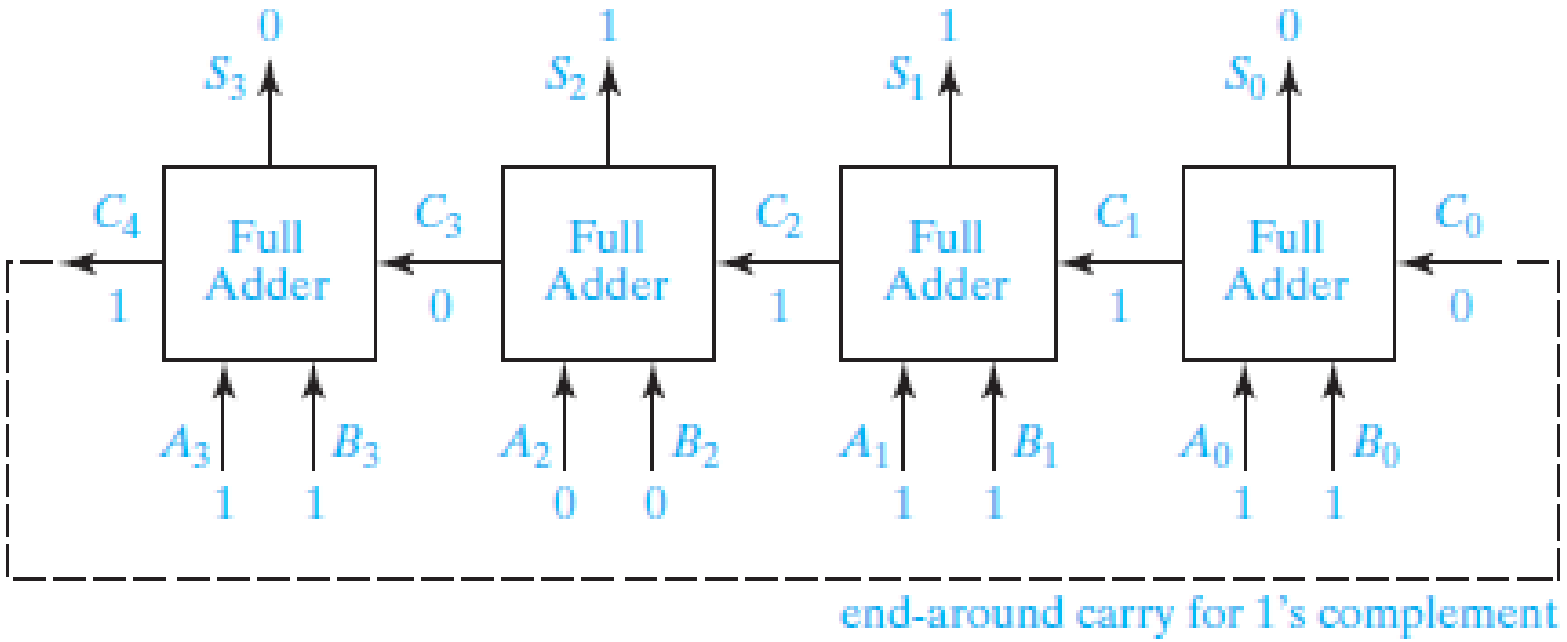
- For 2s complement addition
  - Since each there is a delay of 2 gate-delays to generate each carry, and
  - the carry from one bit ripples into the next bit (hence the name)
  - there is a total delay of 8 gate-delays to generate the final carry



I lost my phone #
can I have yours?
(I need a study partner)

Ripple Carry Adder

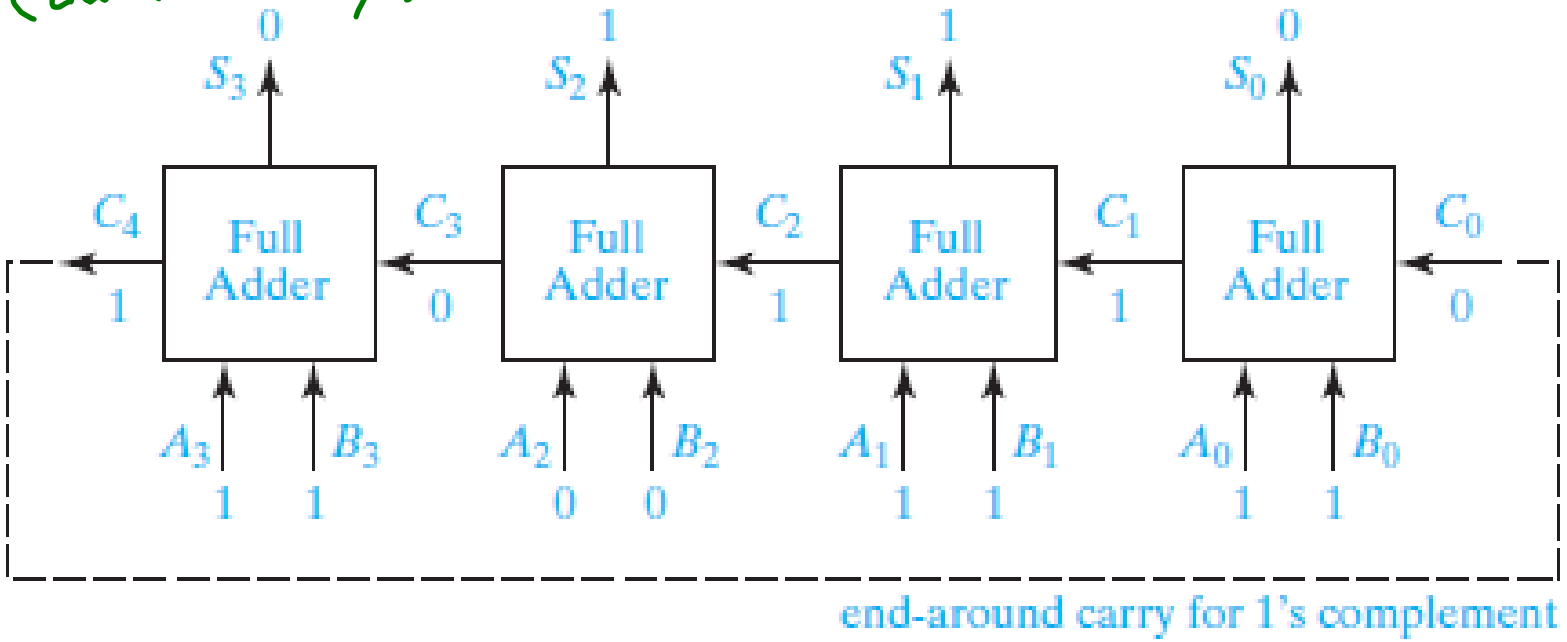
- For 2s complement addition
  - The final carry is discarded (for the result)
  - Note that $C_0 = 0$, giving simplifications $S_0 = A_0 \oplus B_0$ and $C_1 = A_0 B_0$



Ripple Carry Adder

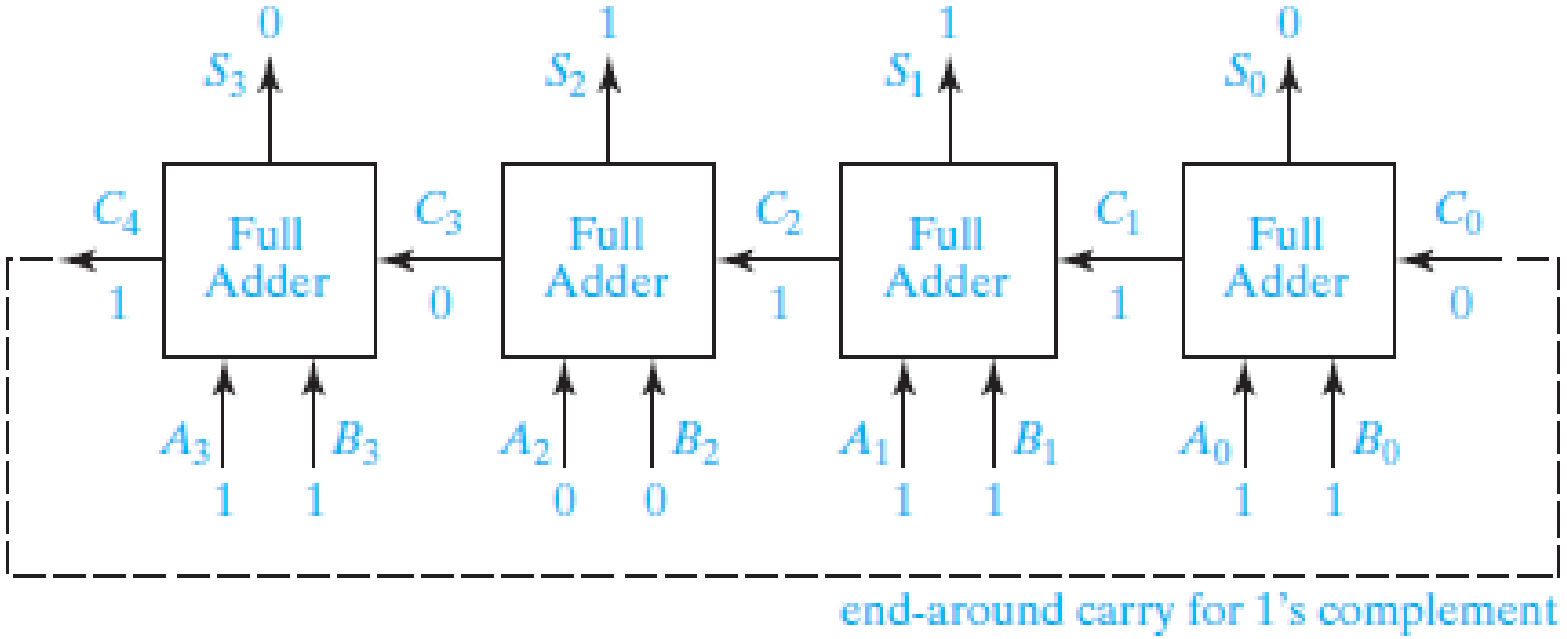
- For 1s complement addition
  - The final carry is used for the end-around carry
  - Fed back as C_0, as shown by the dashed line
  - Have to wait an additional 7 gate delays for certainty its effects rippled to S_3

(Write in results after end-around carry)



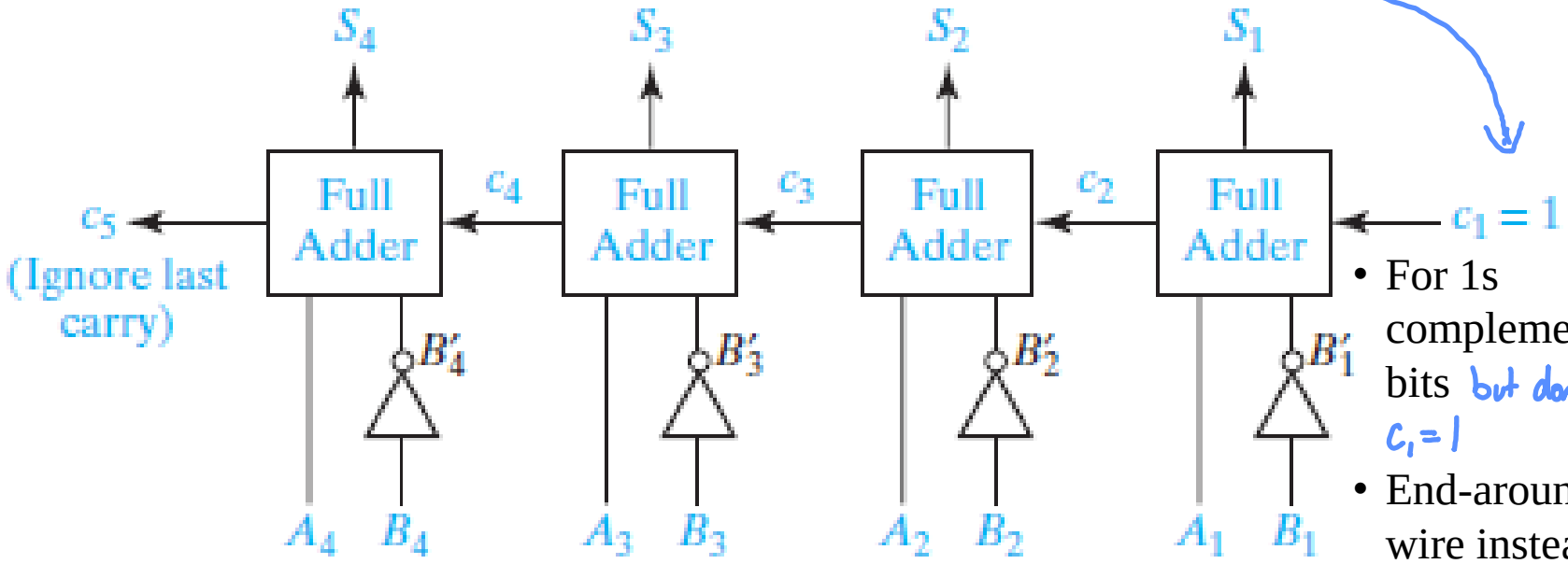
Ripple Carry Adder

- Overflow detection
  - Adding two positive numbers and getting a negative result
  - Adding two negative numbers and getting a positive result
  - Use the sign bits of A, B and S
  - Overflow $V = A_3' B_3' S_3 + A_3 B_3 S_3'$



Subtractor (v1)

- Can be done using the ripple carry adder already developed
- Add the complement of the number to be subtracted
- Can be done with either 1s complement or 2s complement
- 2s complement version shown here (flip bits and add 1)
  - Inverters flip bits B_i
  - Add 1 by setting Carry-in = 1

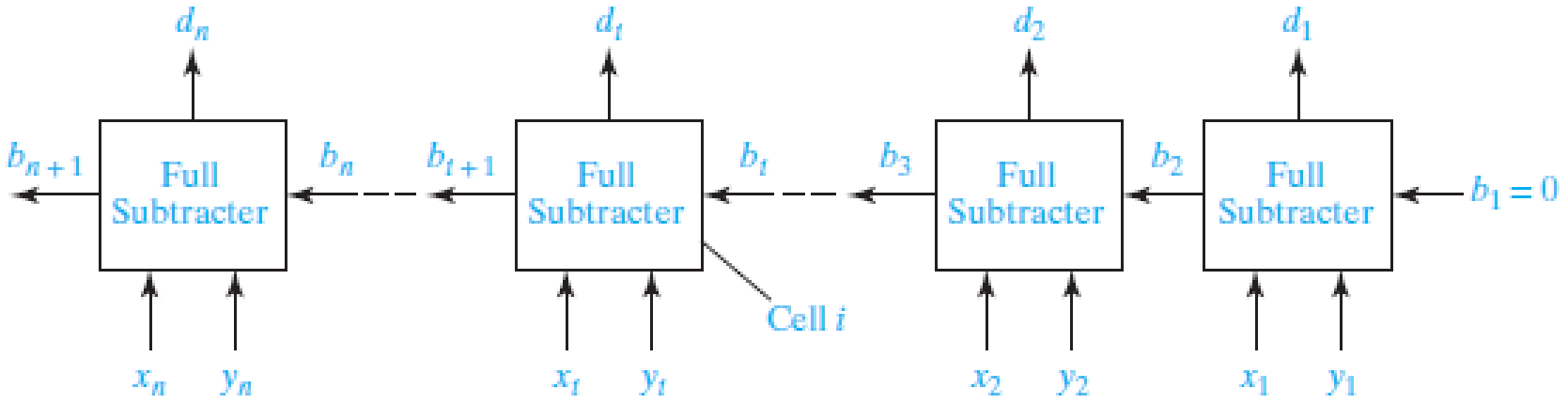


- For 1s complement flip bits but don't set $c_1 = 1$
- End-around carry wire instead

Subtractor (v2)

- Alternative: perform direct subtraction using full subtractor blocks
- Develop using steps similar to that for full adder, but with difference and borrow truth table

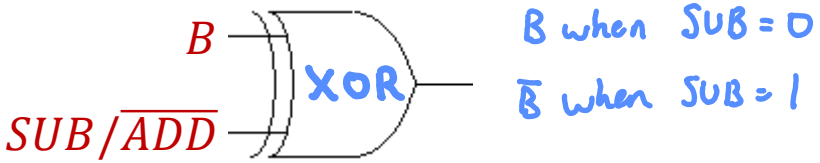
x_i	y_i	b_i	b_{i+1}	d_i
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	1	0
1	0	0	0	1
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1



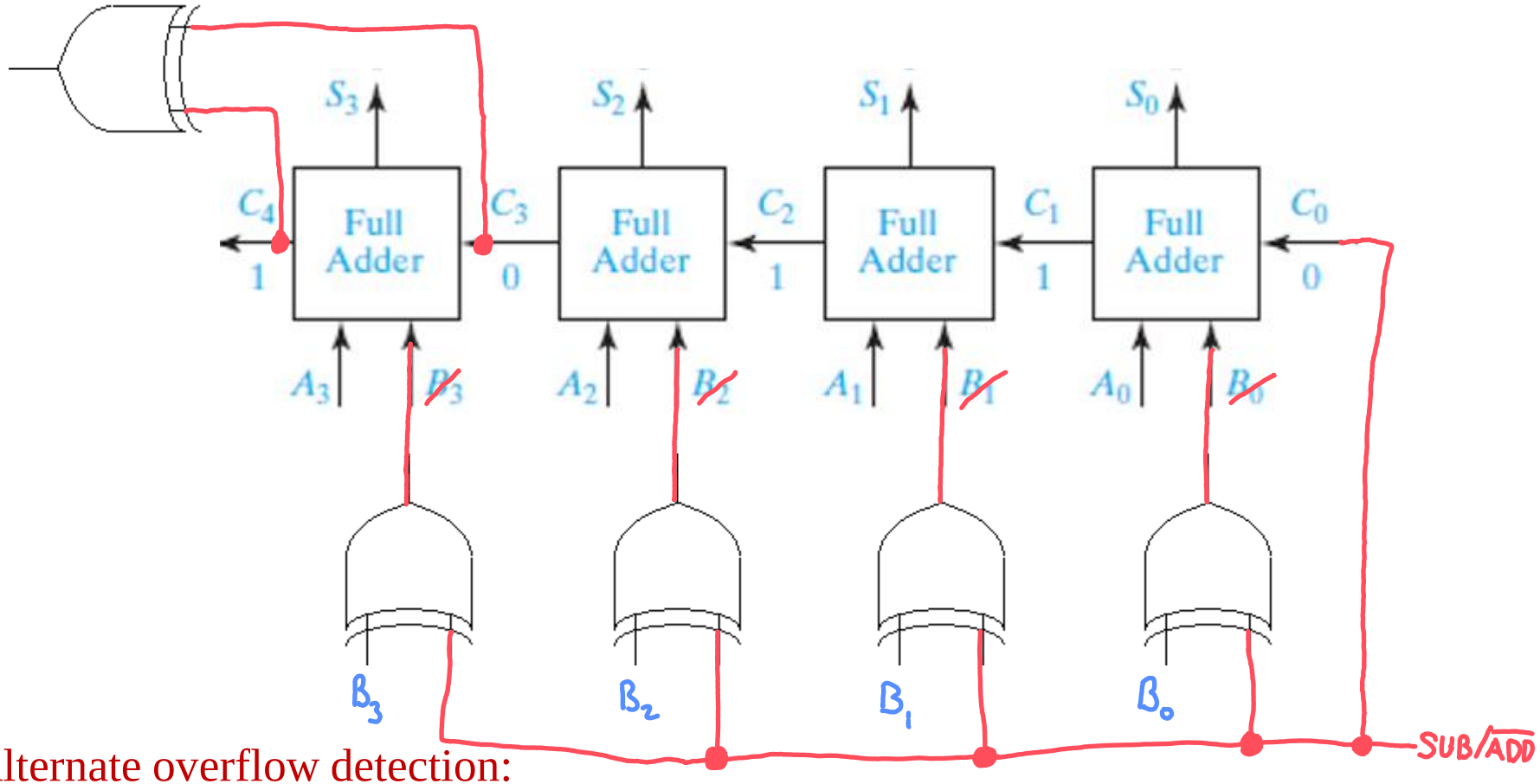
Adder/Subtractor

- Both of those versions of a subtractor are separate circuits
  - Need an adder circuit
  - Need a separate subtractor circuit
- Possible to implement a single adder/subtractor circuit
  - With control bit that determines if addition or subtraction is performed
  - Control bit: $SUB/\overline{ADD}$ (1→ subtract, 0→add)

B	$SUB/\overline{ADD}$	OUT
0	0	0
0	1	1
1	0	1
1	1	0



Adder/Subtractor



Alternate overflow detection:

- Chapter 1: An overflow occurs iff the carry out of the sign position is not equal to the carry into the sign position
- Caution: For 1's complement may falsely flag overflow before end-around carry ripples through

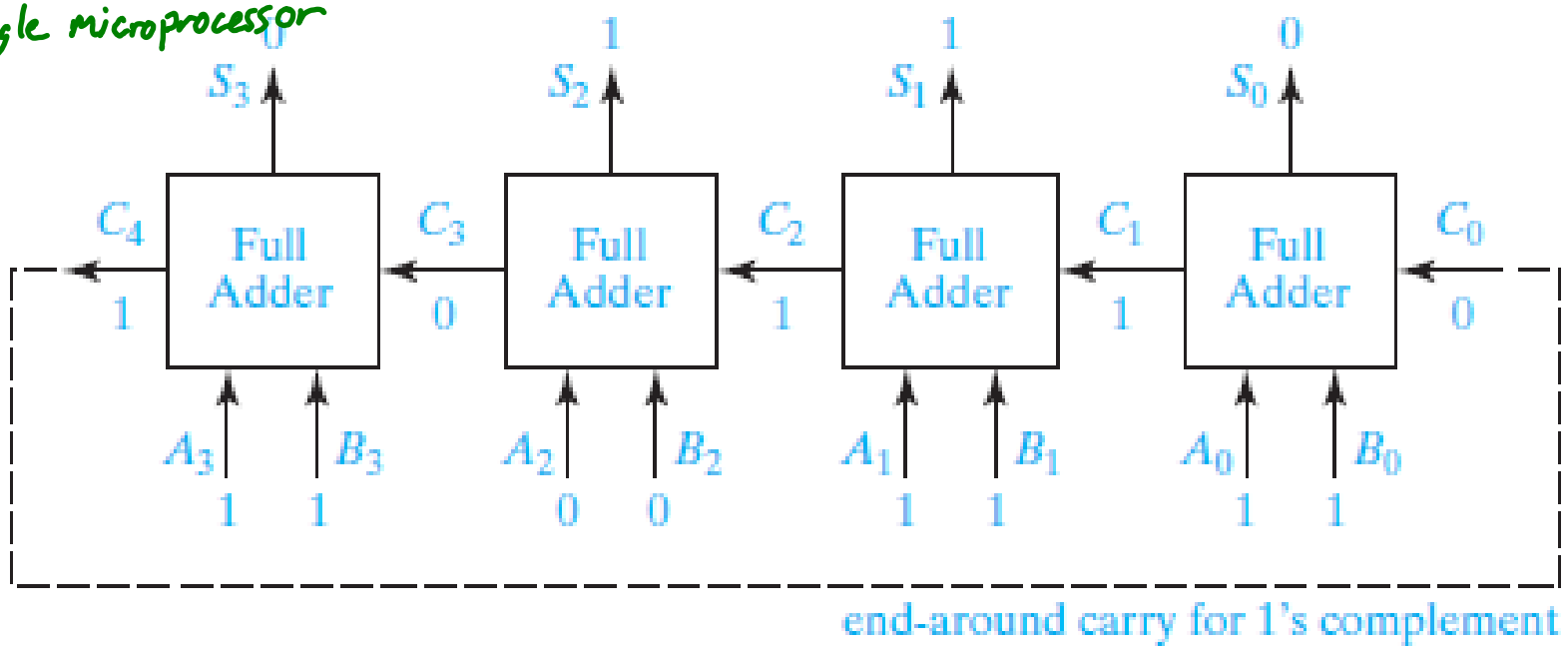
Ripple Carry Adder

From earlier in lecture

- Since each there is a delay of 2 gate-delays to generate each carry, and
- the carry from one bit ripples into the next bit (*hence name "ripple carry"*)
- **there is a total delay of 8 gate-delays to generate the final carry**

Intel 4004
first CPU to have
single microprocessor

Speed can be improved



Carry-Lookahead Adder

- In a parallel adder, carry out of i^{th} stage can be written as
  $C_{i+1} = A_iB_i + C_i(A_i + B_i) = A_iB_i + C_i(A \oplus B) = G_i + P_i C_i$

OR can be replaced by XOR since A_iB_i term already covers the 11 case (look at truth table from earlier)

- Where $G_i = A_iB_i$ is condition for Generation of carry at that stage, &
- $P_i = A_i \oplus B_i$ is condition for Propagation of carry-in to carry-out
- Can be used to algebraically express each carry in terms of
  - The values of the i^{th} bit of A and B, through G_i, and
  - The carry into the first stage, C_0

$$C_1 = G_0 + P_0C_0$$
$$C_2 = G_1 + P_1G_0 + P_1P_0C_0$$
$$C_3 = G_2 + P_2G_1 + P_2P_1G_0 + P_2P_1P_0C_0$$
$$C_4 = G_3 + P_3G_2 + P_3P_2G_1 + P_3P_2P_1G_0 + P_3P_2P_1P_0C_0$$

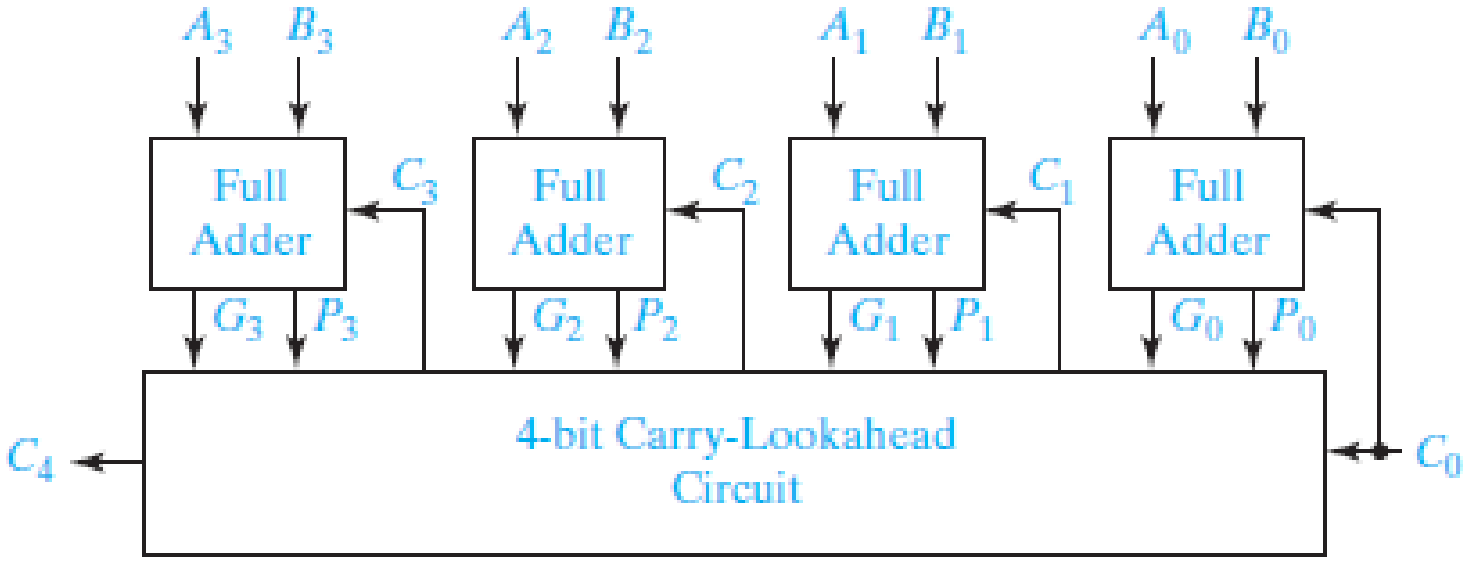
For a 4-bit lookahead carry adder

Carry-Lookahead Adder

$$C_1 = G_0 + P_0C_0$$
$$C_2 = G_1 + P_1G_0 + P_1P_0C_0$$
$$C_3 = G_2 + P_2G_1 + P_2P_1G_0 + P_2P_1P_0C_0$$
$$C_4 = G_3 + P_3G_2 + P_3P_2G_1 + P_3P_2P_1G_0 + P_3P_2P_1P_0C_0$$

Implement in a 2-stage combinational logic circuit:

Two-gate delays



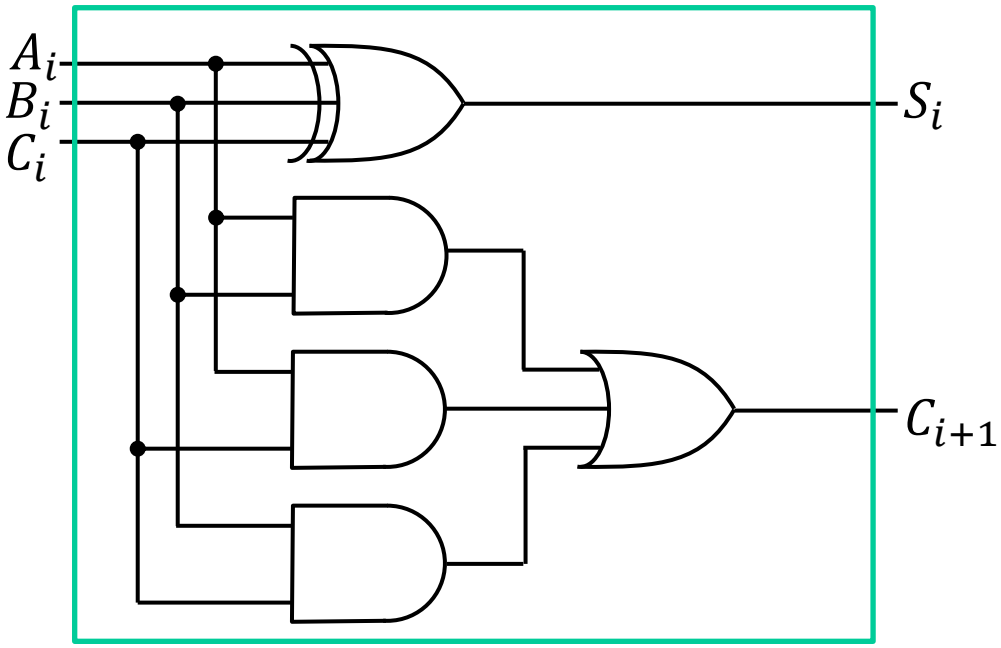
Requires a 5-input AND gate

Requires a 5-input OR gate

Modified full adder circuit on next slide: (G_i & P_i but no C_{i+1})

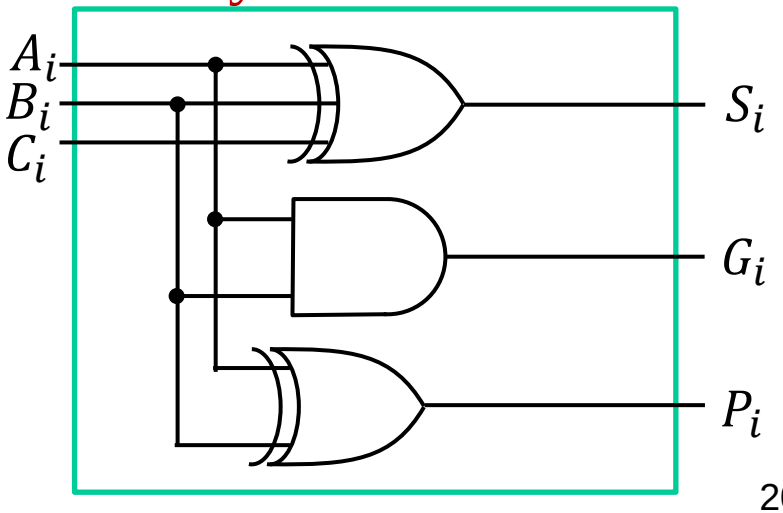
- After the G_i and P_i of the full adders are stable, a change in C_0 propagates to all C_i $(i = 1,2,3,4)$ in 2 gate-delays
- Ripple carry adder: change in C_0 took 8 gate-delays to reach C_4

Carry-Lookahead Adder



Original Full Adder

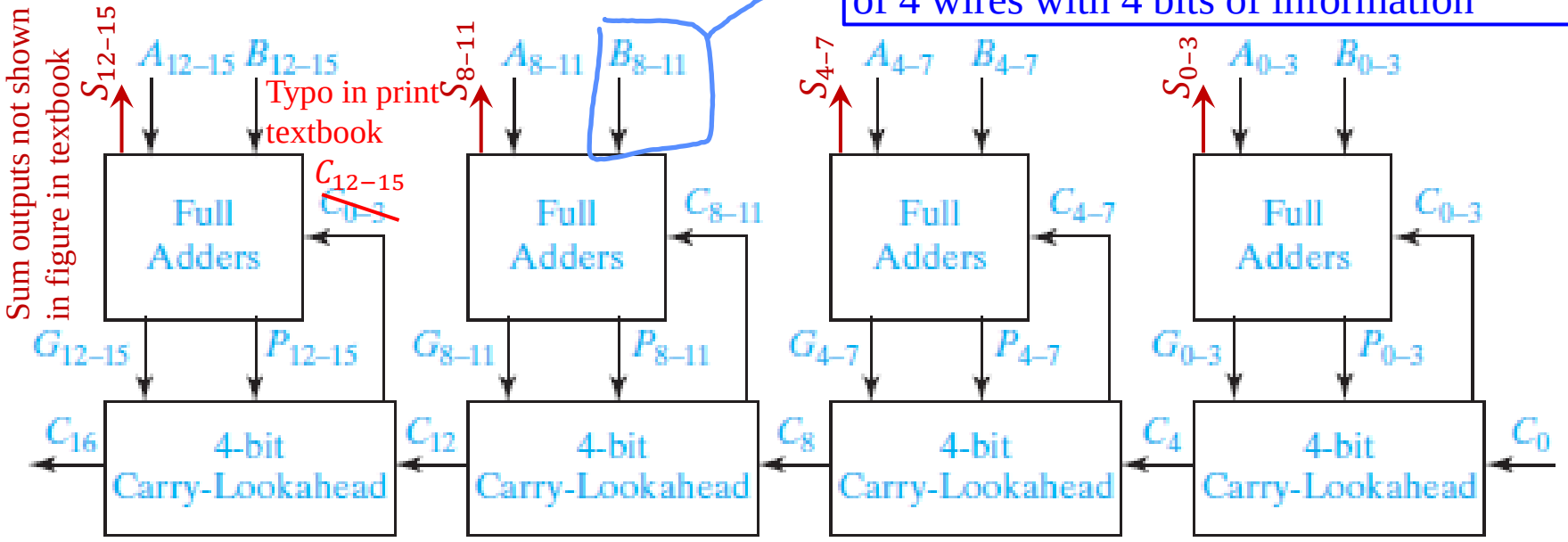
Full Adder for
Carry Lookahead



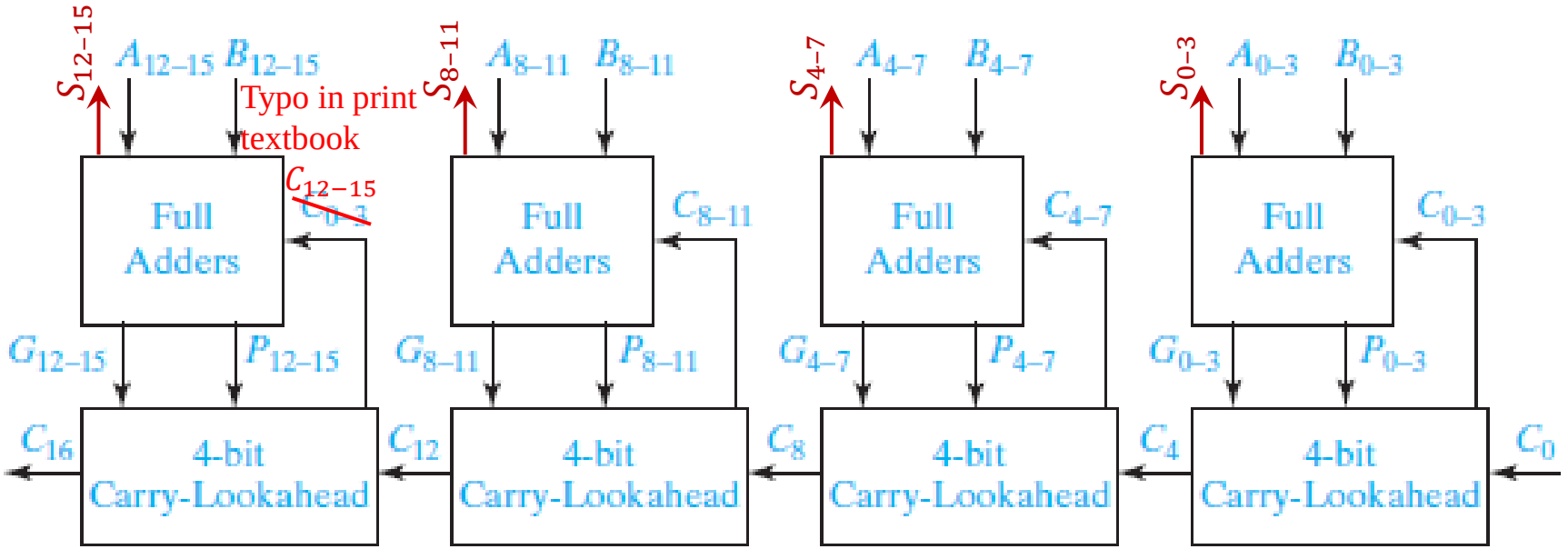
16-bit Carry-Lookahead Adder

- If more than four bits need to be added
  - In principle the carry-lookahead circuit can be expanded
  - But each additional bit means number of inputs to AND and OR gates increases
    - Number of inputs is known as "fan-in"
    - Practical limits (from transistor-level design of gates) set a maximum fan-in
  - In practice, can cascade the 4-bit carry-lookahead circuits
  - A 16-bit adder shown here

Observe notation: Now represents a "bus" of 4 wires with 4 bits of information



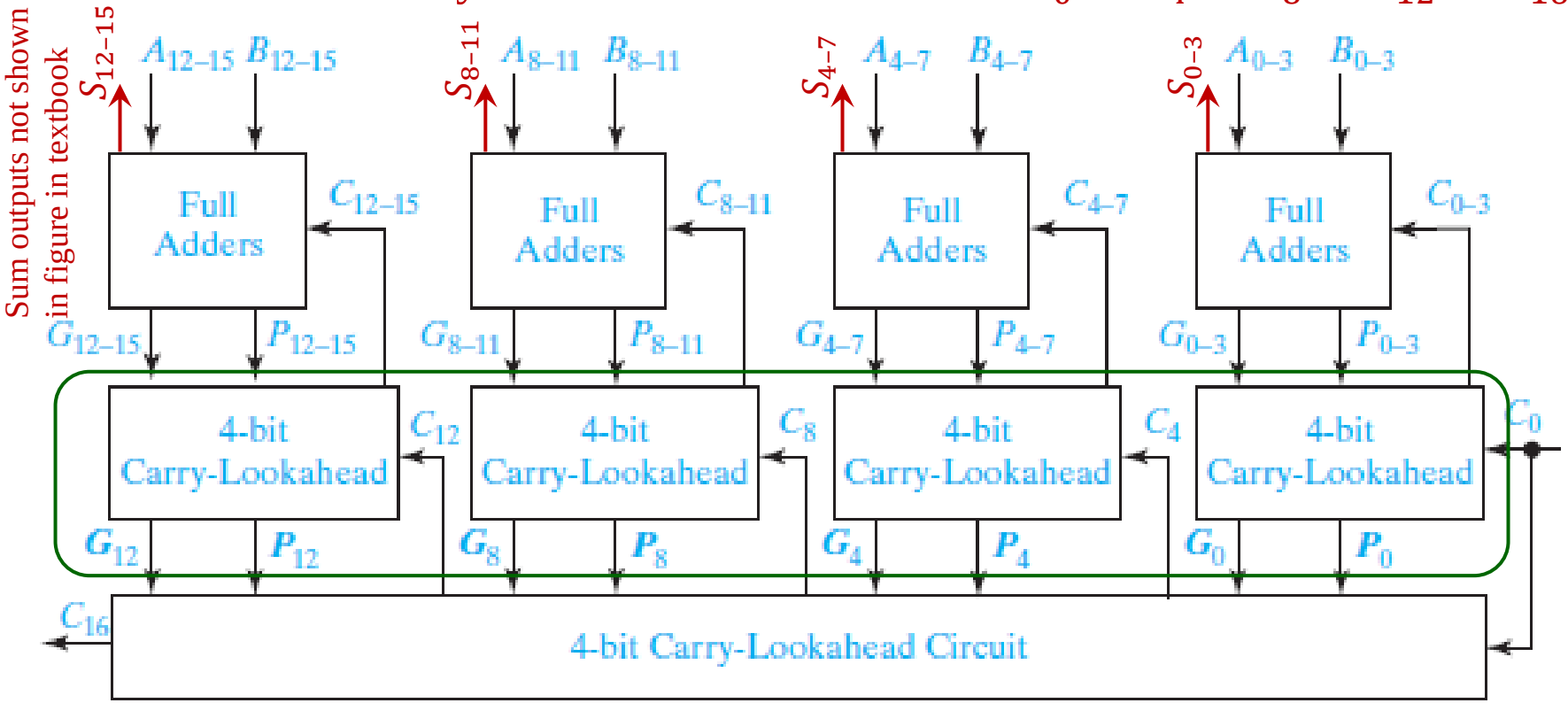
16-bit Carry-Lookahead Adder



- But now there is ripple delay cascading through the Carry-Lookahead circuits: $C_0 \Rightarrow C_4 \Rightarrow C_8 \Rightarrow C_{12} \Rightarrow C_{16}$
- It is still an improvement over brute force 16 bit ripple adder
  - Brute force 16 bit ripple adder: gate delays for C_{16} to settle
  - 16 bit adder shown here: gate delays
- Additional improvement possible: additional level of Carry-Lookahead

16-bit Carry-Lookahead Adder

Additional level of Carry-Lookahead added to handle $C_0 \Rightarrow C_4 \Rightarrow C_8 \Rightarrow C_{12} \Rightarrow C_{16}$



This is same Carry-Lookahead circuit as before

But first level of Carry-Lookahead modified to output G and P as shown, but not the carries to pass forward