Lecture Outline

Reminders to self:

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

- Last Lecture
  – Foundations for K-Maps (Minterms & Maxterms)
  – Started K-Maps(2-variable introduction so far)
- Today’s Lecture
  – Continue K-Maps
    - 3-variable
    - 4-variable
  – Start binary adders and subtracters

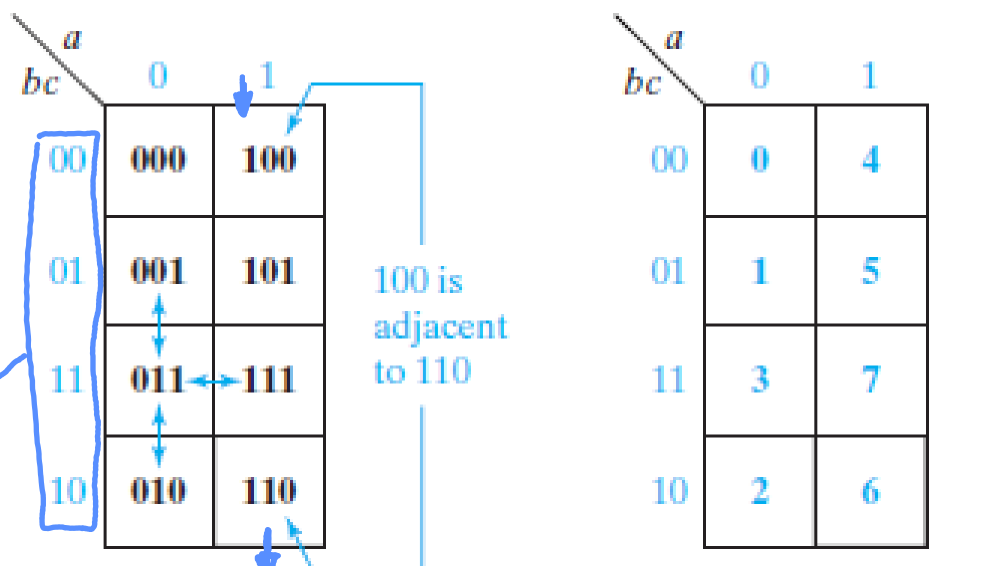
Handouts and Announcements

- Announcements
  - Homework Problems 5-1, 5-2
    - Posted on Carmen yesterday
    - Due in Carmen 11:25am, Monday 2/6
  - Homework Problems 2-4 and 4-1 reminder
    - Both due: 11:59pm Thursday 2/2
  - Participation Quiz 3
    - Based on this lecture. 15min limit after you start.
    - Available 12:25pm today, due 12:25pm tomorrow, but also available 24hrs more with late penalty
  - Read for Wednesday: No new reading assignment

3-Variable Karnaugh Maps

- Three-variable K-Map plotted in a similar manner
- The value of one variable, *a*, is listed on the top and the values of the other two, *b* and *c*, are listed on the side
  - Order of variables in the minterms shown here is *abc*
  - Note: alternative layout with one variable on the side and two on the top is equally valid

FIGURE 5-3 Location of Minterms on a Three-Variable Karnaugh Map



(a) Binary notation

(b) Decimal notation

- Note "Gray-code-like" order
- Only one variable changes from row-to-row

3-Variable Karnaugh Maps

- Minterms in adjacent squares of map differ in only one variable and therefore can be combined using uniting theorem $XY + XY' = X$
- Do this as an example. Formal algorithm next lecture

From Truth Table: $F(A,B,C) = A'BC' + A'BC + AB'C' + ABC'$

From K-map: $F = A'B + AC'$

Values of F come from first equation *just because*
- equation can be changed to whatever we want

A B C	F
0 0 0	0
0 0 1	0
0 1 0	1
0 1 1	1
1 0 0	1
1 0 1	0
1 1 0	1
1 1 1	0

(a)

$ABC = 001, F = 0$

BC \ A	0	1
00	0	1
01	0	0
11	1	0
10	1	1

F

$ABC = 110, F = 1$

$A = 1,1 \to A$
$B = 1,0 \to 0$
$C = 0,0 \to C'$

$A = 0,0 \to A'$
$B = 1,1 \to B$
$C = 1,0 \to 0$

(b)

4-Variable Karnaugh Maps

A	B	C	D	F
0	0	0	0	m_0
0	0	0	1	m_1
0	0	1	0	m_2
0	0	1	1	m_3
0	1	0	0	m_4
0	1	0	1	m_5
0	1	1	0	m_6
0	1	1	1	m_7
1	0	0	0	m_8
1	0	0	1	m_9
1	0	1	0	m_{10}
1	0	1	1	m_{11}
1	1	0	0	m_{12}
1	1	0	1	m_{13}
1	1	1	0	m_{14}
1	1	1	1	m_{15}

Map with AB on top
and CD on side

Remember that the map wraps

- Left edge is adjacent to right edge (Only one bit changes)
- Top edge is adjacent to bottom edge (only one bit changes)

$B = 1$

$CD \backslash AB$	00	01	11	10
00	m_0	m_4	m_{12}	m_8
01	m_1	m_5	m_{13}	m_9
11	m_3	m_7	m_{15}	m_{11}
10	m_2	m_6	m_{14}	m_{10}

$D = 1$

$C = 1$

$A = 1$

$AB \backslash CD$	00	01	11	10
00	m_0	m_1	m_3	m_2
01	m_4	m_5	m_7	m_6
11	m_{12}	m_{13}	m_{15}	m_{14}
10	m_8	m_9	m_{11}	m_{10}

Map with AB on side
and CD on top

K-Map Minimal SOP Algorithm

1. Draw the largest rectangular box (squares are a special case of rectangular) that:
   A. Does not include any 0s
   B. Has height (# of rows) that is a power of 2 (1, 2, 4, 8, ...)
   C. Has width (# of columns) that is a power of 2
2. Make sure that every 1 on the map is in the **largest possible box** Larger boxes do more reduction
3. Reduce the products by looking at boxes
   A. The reduced expression will be SOP
   B. The result may not be unique, but will be maximally reduced

K-Map Example

a	b	c	f
0	0	0	**0**
0	0	1	**1**
0	1	0	**0**
0	1	1	**1**
1	0	0	**1**
1	0	1	**1**
1	1	0	**0**
1	1	1	**1**

$$f(a,b,c) = \sum m(1,3,4,5,7)$$

$$f = a'b'c + a'bc + ab'c' + ab'c + abc$$

$bc \backslash a$	0	1
00	0	1
01	1	1
11	1	1
10	0	0

$c = 1$

$b = 1$

A=1, B=0
C changes

C=1
Because A and B are changing

F= AB' + C

K-Map Example

w	y	z	f
0	0	0	**0**
0	0	1	**1**
0	1	0	**0**
0	1	1	**1**
1	0	0	**1**
1	0	1	**1**
1	1	0	**X**
1	1	1	**1**

$$f(w,y,z) = \sum m(1,3,4,5,7) + \sum d(6)$$

$yz \backslash w$	0	1
00	0	1
01	1	1
11	1	1
10	0	X

W=1

$z = 1$

z=1

$y = 1$

F=W+Z

Don't Cares in K-maps

- All 1s must be covered
- Xs are used only if they will simplify the resulting expression

Note: Still SOP form

K-Map Amount of Reduction

K-map Boxes that are Groups of:

- 2 → reduce minterm by 1 variable
- 4 → reduce minterm by 2 variables
- 8 → reduce minterm by 3 variables
- 1 → reduce minterm by 0 variables (full-length minterm)

K-Map Example

$$f(a,b,c) = \sum m(0,2,4)$$

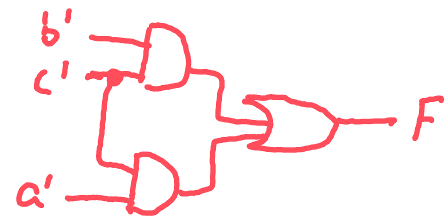
$a \backslash bc$	00	01	11	10
0	1	0	0	1
1	1	0	0	0

a=0
c=0

b=0, c=0

F=A'C'+B'C'

Sketch circuit



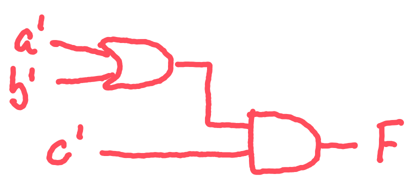
Later in semester will implement logic using hardware that assume 2-level AND-OR → SOP

Note: $f = c'(a' + b')$ is algebraically correct, but it is POS.

Not correct for SOP

But uses fewer gates

Sketch circuit

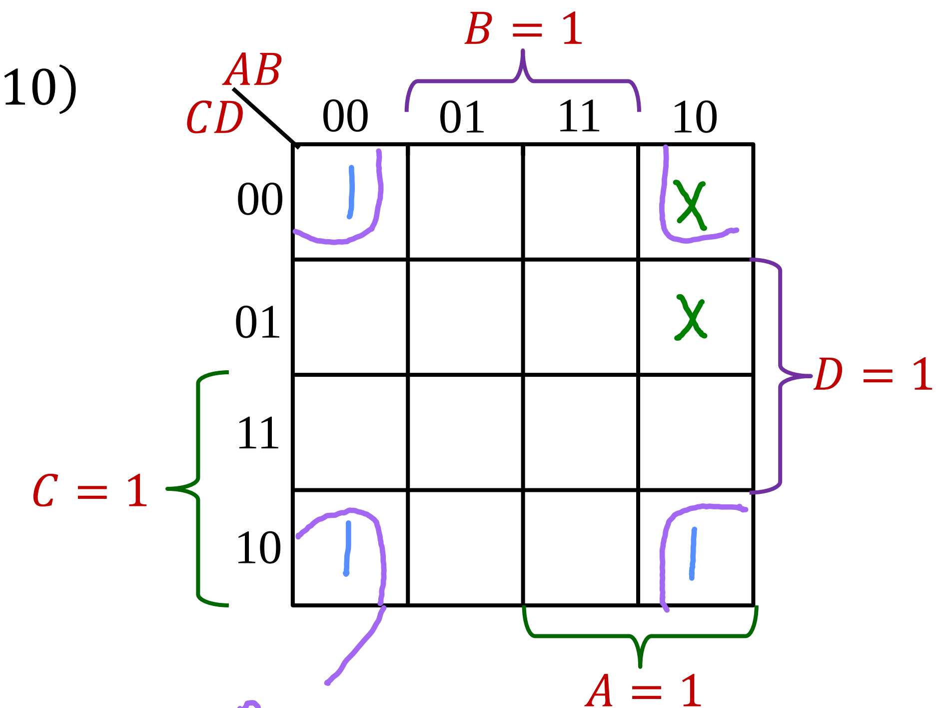


K-Map Example

A	*B*	*C*	*D*	*F*
0	0	0	0	1
0	0	0	1	0
0	0	1	0	1
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	X
1	0	0	1	X
1	0	1	0	1
1	0	1	1	0
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	0

$$f(A,B,C,D) = \sum m(0,2,10)$$

$$+\sum d(8,9)$$



B = 0
D = 0

F = B'D'

K-Map Example

A	*B*	*C*	*D*	*F*
0	0	0	0	**1**
0	0	0	1	**0**
0	0	1	0	**0**
0	0	1	1	**0**
0	1	0	0	**0**
0	1	0	1	**1**
0	1	1	0	**0**
0	1	1	1	**0**
1	0	0	0	**0**
1	0	0	1	**0**
1	0	1	0	**1**
1	0	1	1	**0**
1	1	0	0	**0**
1	1	0	1	**0**
1	1	1	0	**0**
1	1	1	1	**1**

$$f(A,B,C,D) = \sum m(0,5,10,15)$$

I copied the truth table from the previous slide and pasted it here. I was supposed to edit it for the function of this example, but apparently neglected to do so before lecture. I corrected it after lecture.

$B = 1$ (columns 01, 11)

$CD \backslash AB$	00	01	11	10
00	1			
01		1		
11			1	
10				1

$D = 1$ (rows 01, 11)

$C = 1$ (rows 11, 10)

$A = 1$ (columns 11, 10)

No adjacent 1s
cannot be reduced

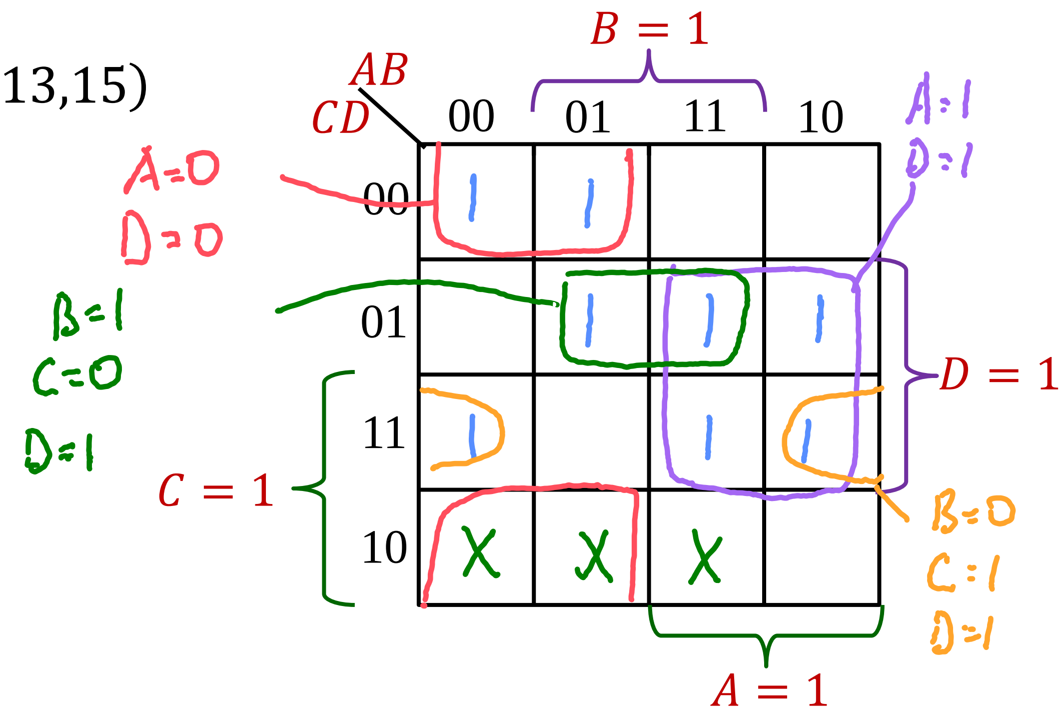
$$f(A,B,C,D) = A'B'C'D' + A'BC'D + ABCD + AB'CD'$$

equation already minimal

K-Map Example

$$f(A,B,C,D) = \sum m(0,3,4,5,9,11,13,15)$$

$$+\sum d(2,6,14)$$



$f(A,B,C,D) = AD + A'D' + BC'D + B'CD$

Note that this solution is not unique. There are other ways to pick up those last two ones:

- $A'BC'$ for the first one
- $A'B'C$ for the second one

Summary of K-Maps

- Method to reduce SOP expression into another SOP expression:
  - Minimal in # of gates
  - Minimal in # of inputs/gate
- Map minterms next to each other such that they vary by only one bit
- Group in rectangles whose dimensions are power of 2
  - Variables whose values change within rectangles can be eliminated
  - $AB'C + A'B'C = B'C(A + A') = B'C$
  - Group of 1, no reduction
  - Group of 2, eliminate 1 variable
  - Group of 4, eliminate 2 variables
  - Group of 8, eliminate 3 variables
- K-Map edges are connectable (wrap around) $\leftrightarrow$ $\updownarrow$ ⊠ corners too
- Five variable K-map (think 3-D: layers) section not assigned reading
  - You are not responsible for (not on HW or exams)
  - But we will do an example that uses them late in semester

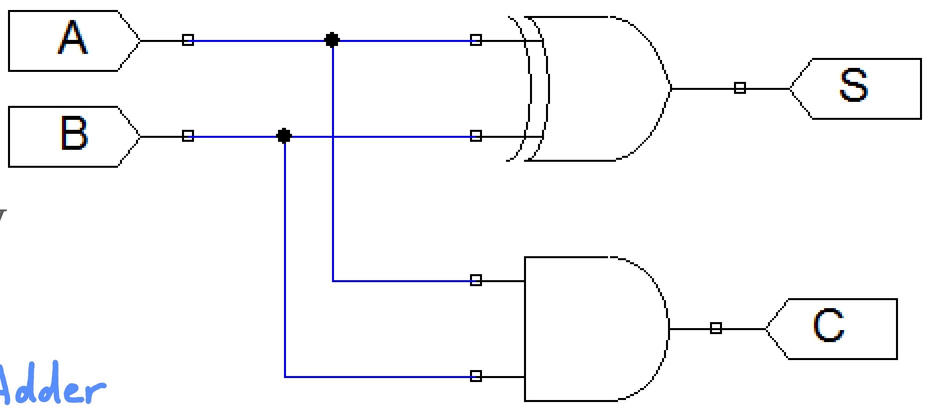
Half Adder

- Add two binary bits, A and B Need two outputs
  - A sum bit, S
  - A carry bit, C
- $A + B = S, C$ (regular addition here, not an OR)
- $S = A'B + AB' = A \oplus B$
- $C = AB$
- This works for adding two 1-bit numbers, but what if each number has more than one bit?
- Need to be able to add in the carry from the previous bit
- Circuit for that is known as a Full Adder

A	B	S	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

XOR

AND



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_iC_i	00	01	11	10
0		1		1
1	1		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

- No reduction possible
- For S_i each minterm has an odd number of 1s
- $S_i = X_i \oplus Y_i \oplus C_i$

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}

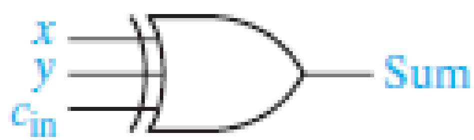
$X_i \backslash Y_iC_i$	00	01	11	10
0				
1				

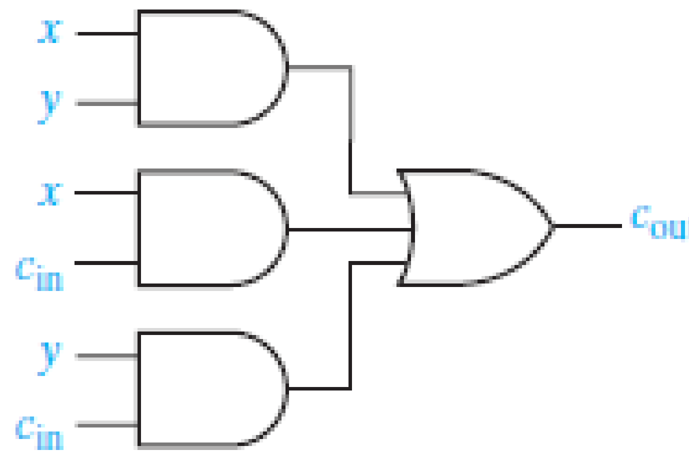
X_i	Y_i	C_i	S_i	C_{i+1}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

-
- $C_{i+1} = X_iY_i + X_iC_i + Y_iC_i$

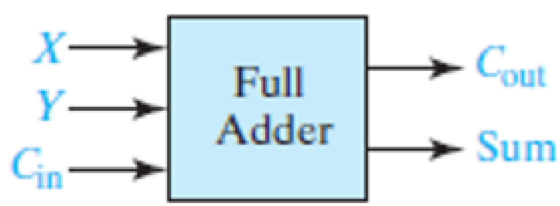
Ripple Carry Adder

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





- 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



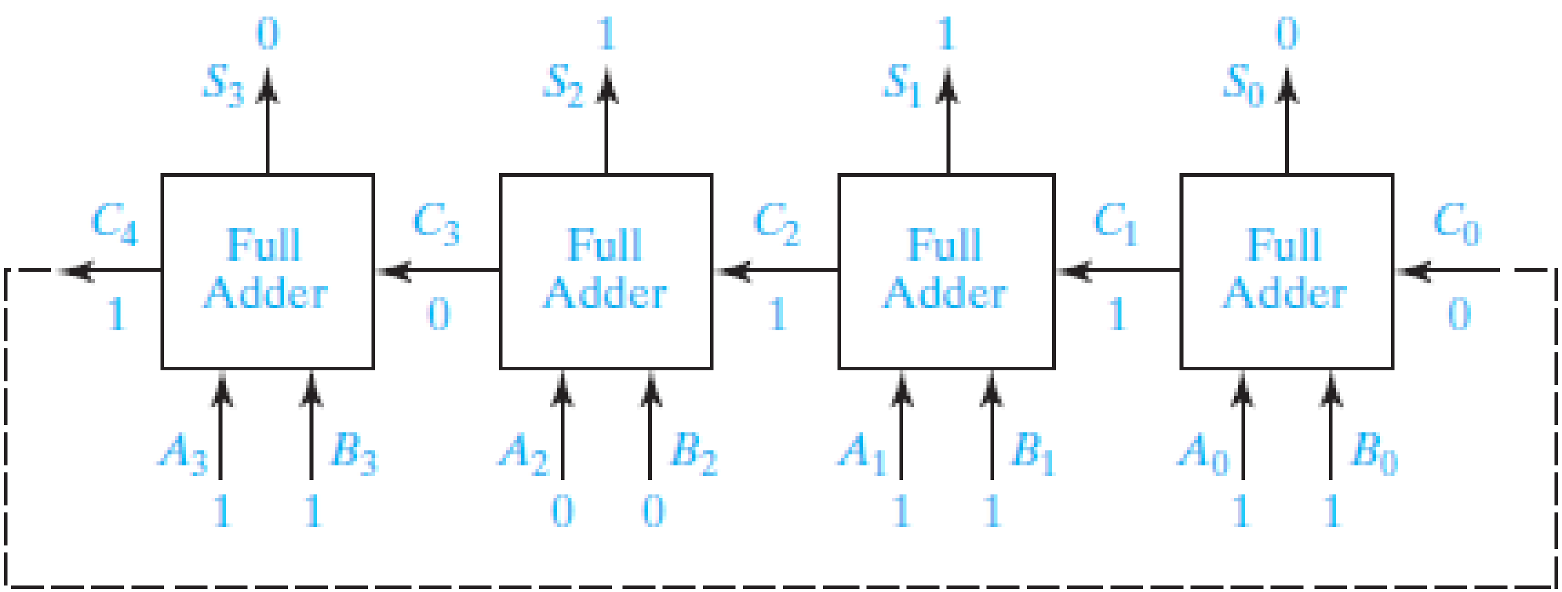
Four-bit Adder

- 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
    - Nine inputs
    - Five outputs: $S_3S_2S_1S_0$ and C_4 ( )
  - Truth table will have
  - 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 and
  -

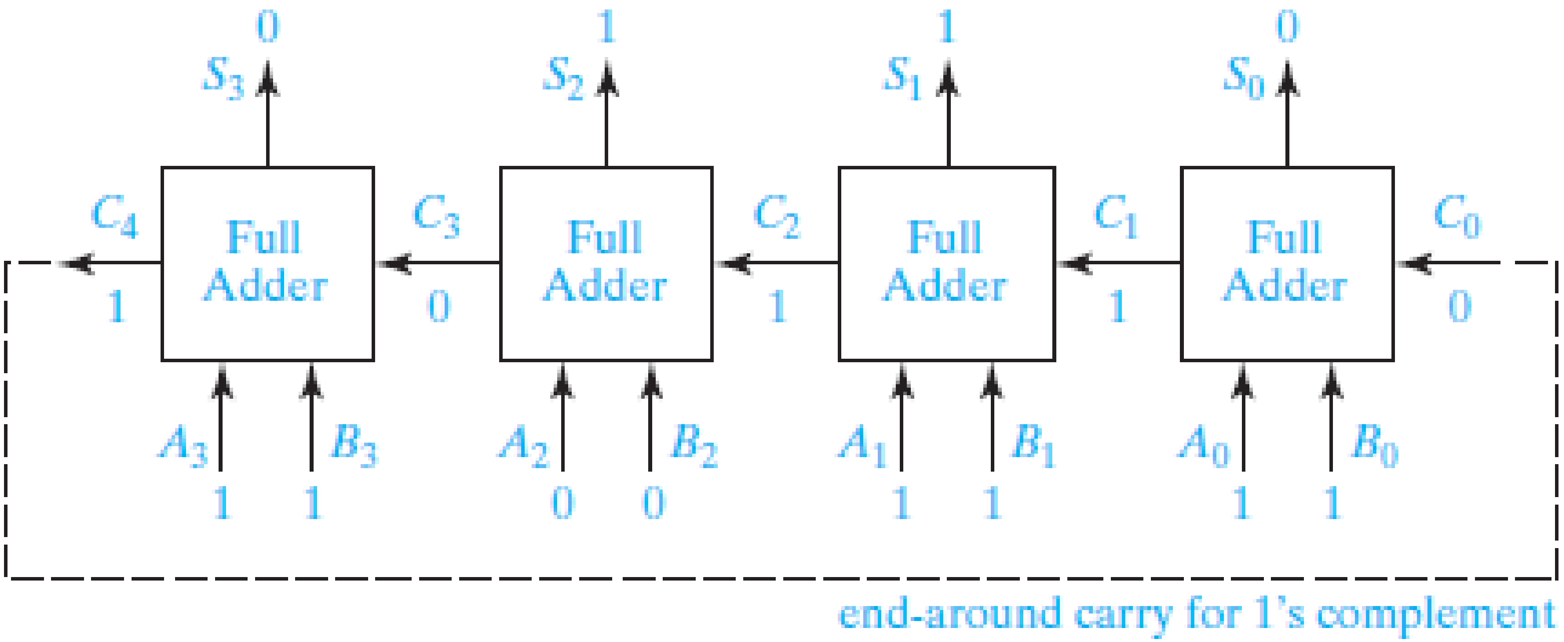
Question: What can you say about the result if this is 1's or 2's complement addition?



- 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

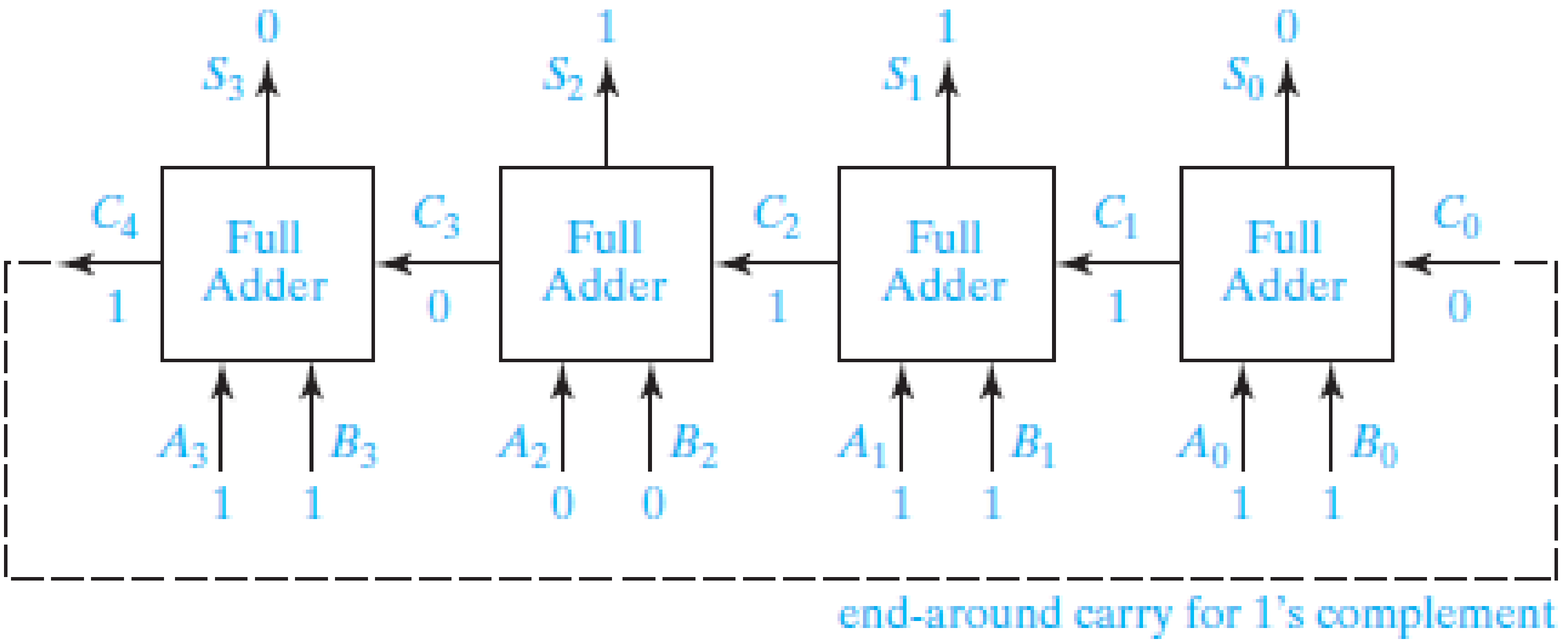
Ripple Carry Adder

- 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 ( )
  - there is a total delay of 8 gate-delays to generate the final carry



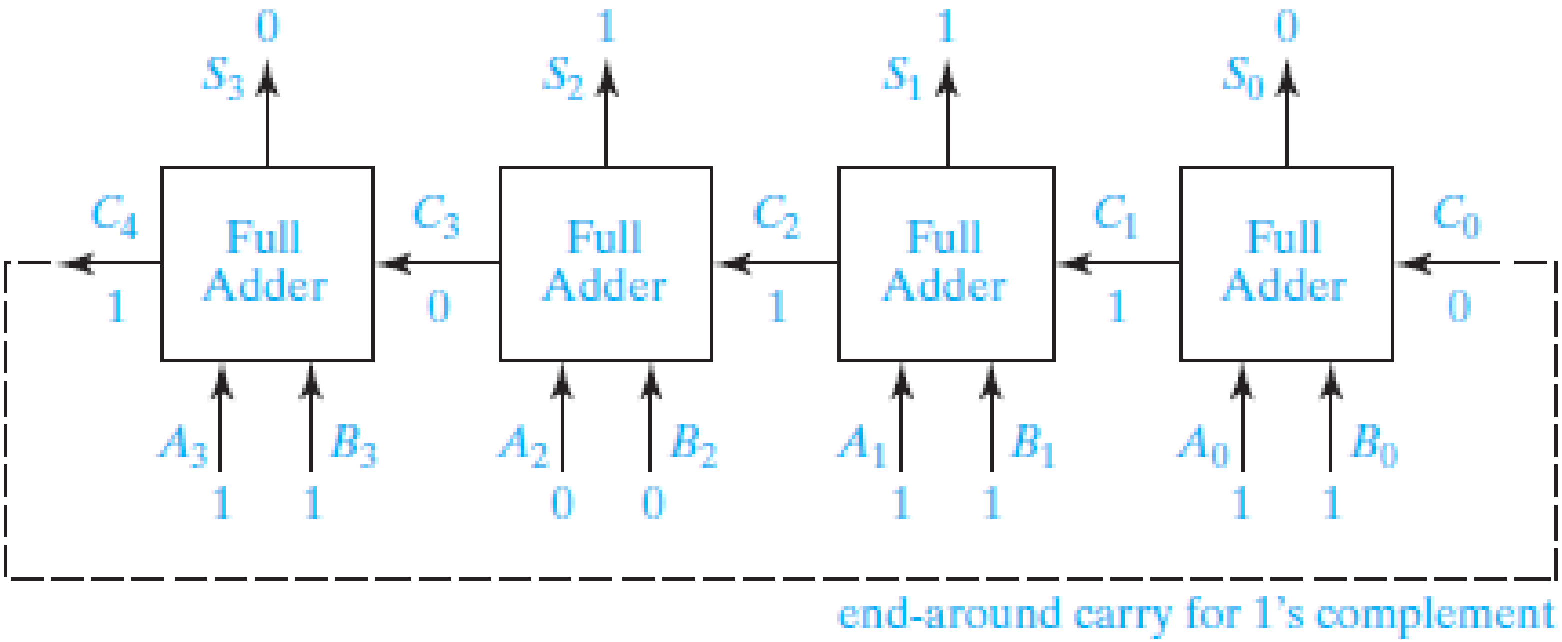
Ripple Carry Adder

- For 2s complement addition
  - The final carry is discarded (   )
  - 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 <u>additional</u> 7 gate delays for certainty its effects rippled to S_3



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 $V = A_3'B_3'S_3 + A_3B_3S_3'$