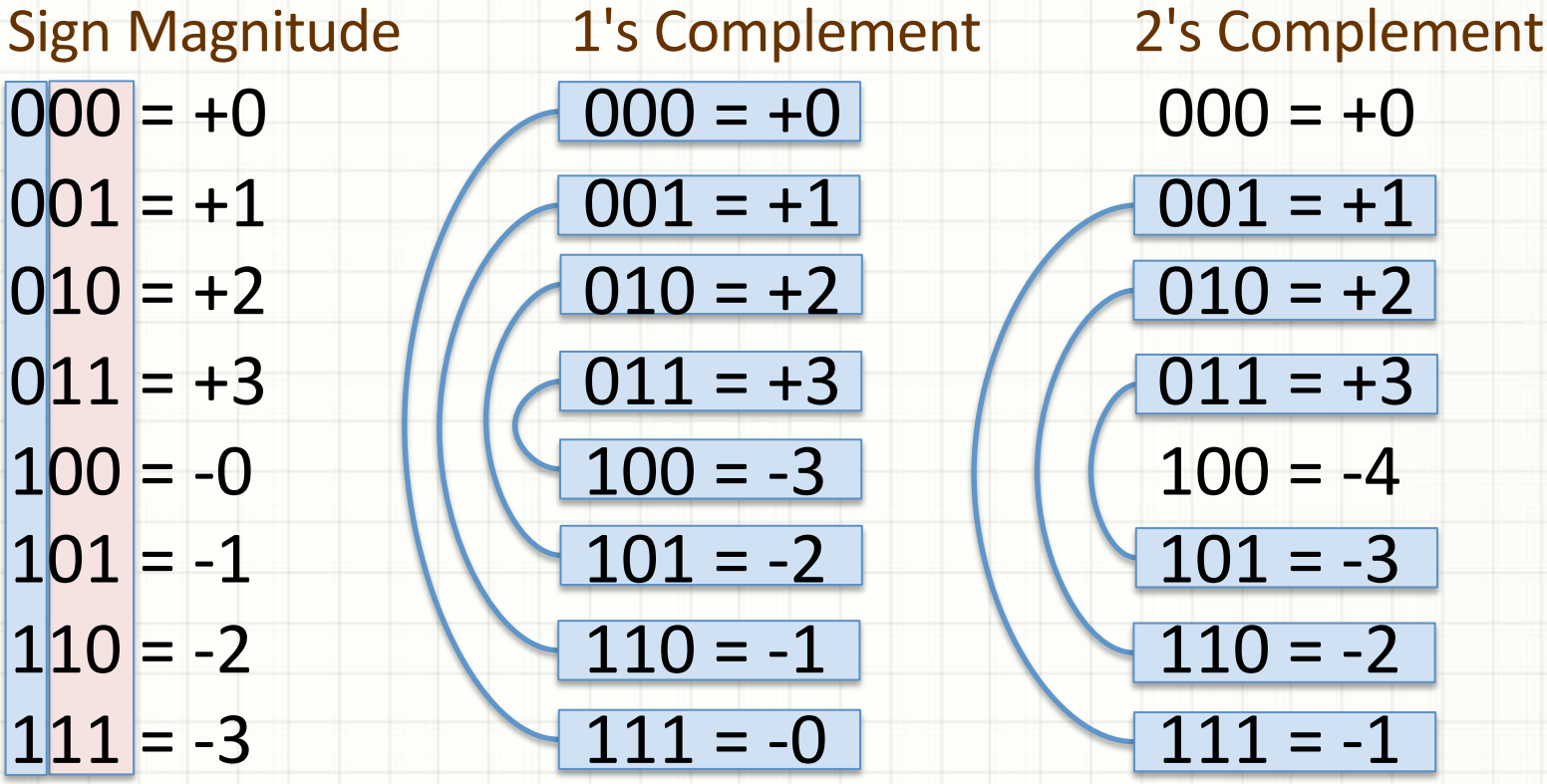
COL215 Digital Logic and System Design

Signed Numbers,

Fast addition

06 September 2017

Representing signed integers



	Sign Magnitude	1's Complement	2's Complement
Sign	leftmost bit	leftmost bit	leftmost bit
Balance	Yes	Yes	No
Zeroes	Not unique	Not unique	Unique

16 bit signed integers

$0000\ 0000\ 0000\ 0000_2 = \quad 0_{10}$

$0000\ 0000\ 0000\ 0001_2 = +\ 1_{10}$

$0000\ 0000\ 0000\ 0010_2 = +\ 2_{10}$

.....

maxint = 32,767

$0111\ 1111\ 1111\ 1110_2 = +\ (2^{15}\text{-}2)_{10}$

$0111\ 1111\ 1111\ 1111_2 = +\ (2^{15}\text{-}1)_{10}$

$1000\ 0000\ 0000\ 0000_2 = -\ (2^{15})_{10}$

$1000\ 0000\ 0000\ 0001_2 = -\ (2^{15}\text{-}1)_{10}$

$1000\ 0000\ 0000\ 0010_2 = -\ (2^{15}\text{-}2)_{10}$

minint = -32,768

.....

$1111\ 1111\ 1111\ 1101_2 = -\ 3_{10}$

$1111\ 1111\ 1111\ 1110_2 = -\ 2_{10}$

$1111\ 1111\ 1111\ 1111_2 = -\ 1_{10}$

32 bit signed integers

$0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000_2 = 0_{10}$

$0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0001_2 = +1_{10}$

$0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0010_2 = +2_{10}$

.....

maxint = 2,147,483,647

$0111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1110_2 = +(2^{31}-2)_{10}$

$0111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111_2 = +(2^{31}-1)_{10}$

$1000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000_2 = -(2^{31})_{10}$

$1000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0001_2 = -(2^{31}-1)_{10}$

$1000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0010_2 = -(2^{31}-2)_{10}$

minint = -2,147,483,648

.....

$1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1101_2 = -3_{10}$

$1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1110_2 = -2_{10}$

$1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111_2 = -1_{10}$

Add/Subtract signed integers

- 2's complement representation makes it easy - add/subtract ignoring sign!

0111 [7]	0101 [5]	0010 [2]
+ 1110 [-2]	- 1110 [-2]	- 0101 [5]
0101 [5]	0111 [7]	1101 [-3]

Why?

Representation of -X is nothing but 2^n - X

Two's complement representation

- Represent "- 3"

	10000 [2^4]	alternatively	1100 [invert 3]
-	0011 [3]		+ 0001 [1]
	1101 [-3]		1101 [-3]

- Negating a two's complement number (+ve or -ve): invert all bits and add 1
- Subtraction using addition: X-Y = X+Y' +1

Converting n bit numbers into numbers with more than n bits

Required when operands are of mixed size

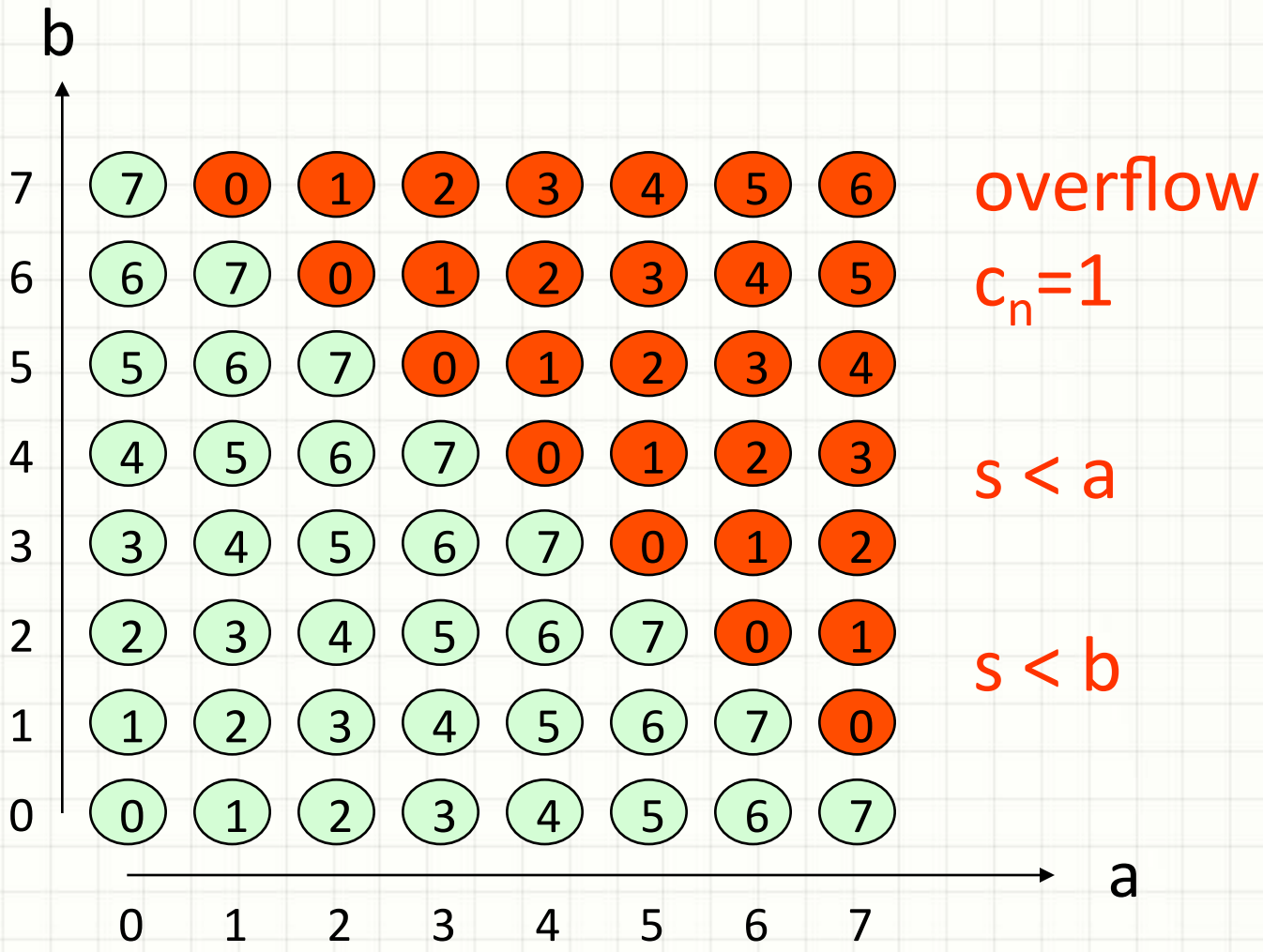
For example, adding a signed half word to a signed full word

- copy the most significant bit (the sign bit) into the other bits

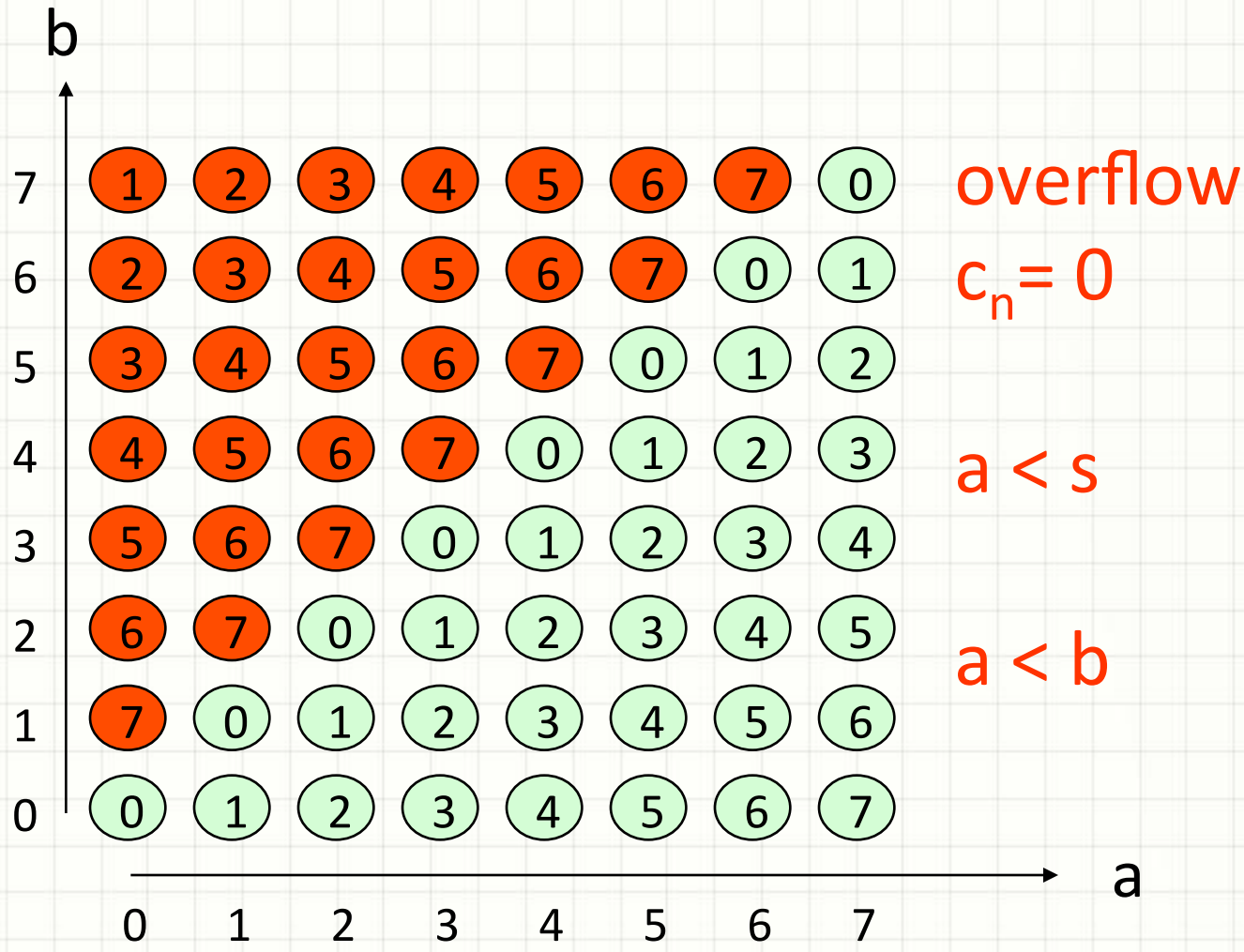
0010 $\Rightarrow$ 0000 0010

1010 $\Rightarrow$ 1111 1010

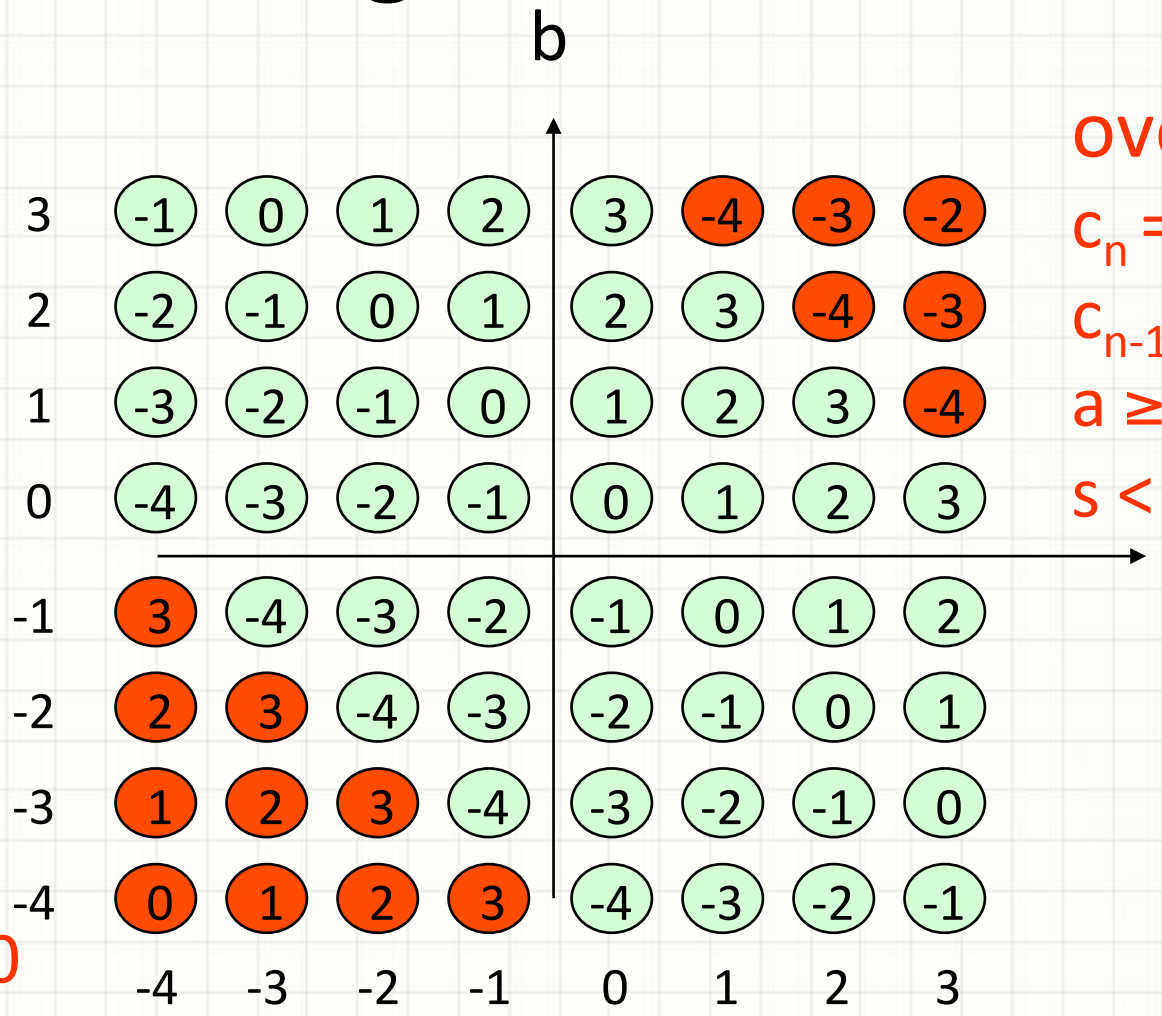
Overflow in unsigned s = a + b



Overflow in unsigned s = a + b’ + 1



Overflow in signed s = a + b



overflow

$c_n = 0$

$c_{n-1} = 1$

$a \geq 0, b \geq 0$

$s < 0$

overflow

$c_n = 1$

$c_{n-1} = 0$

$a < 0, b < 0$

$s \geq 0$

general : $c_n \neq c_{n-1}$, sign(a) = sign(b) ≠ sign(s)

Overflow in signed s = a + b' + 1

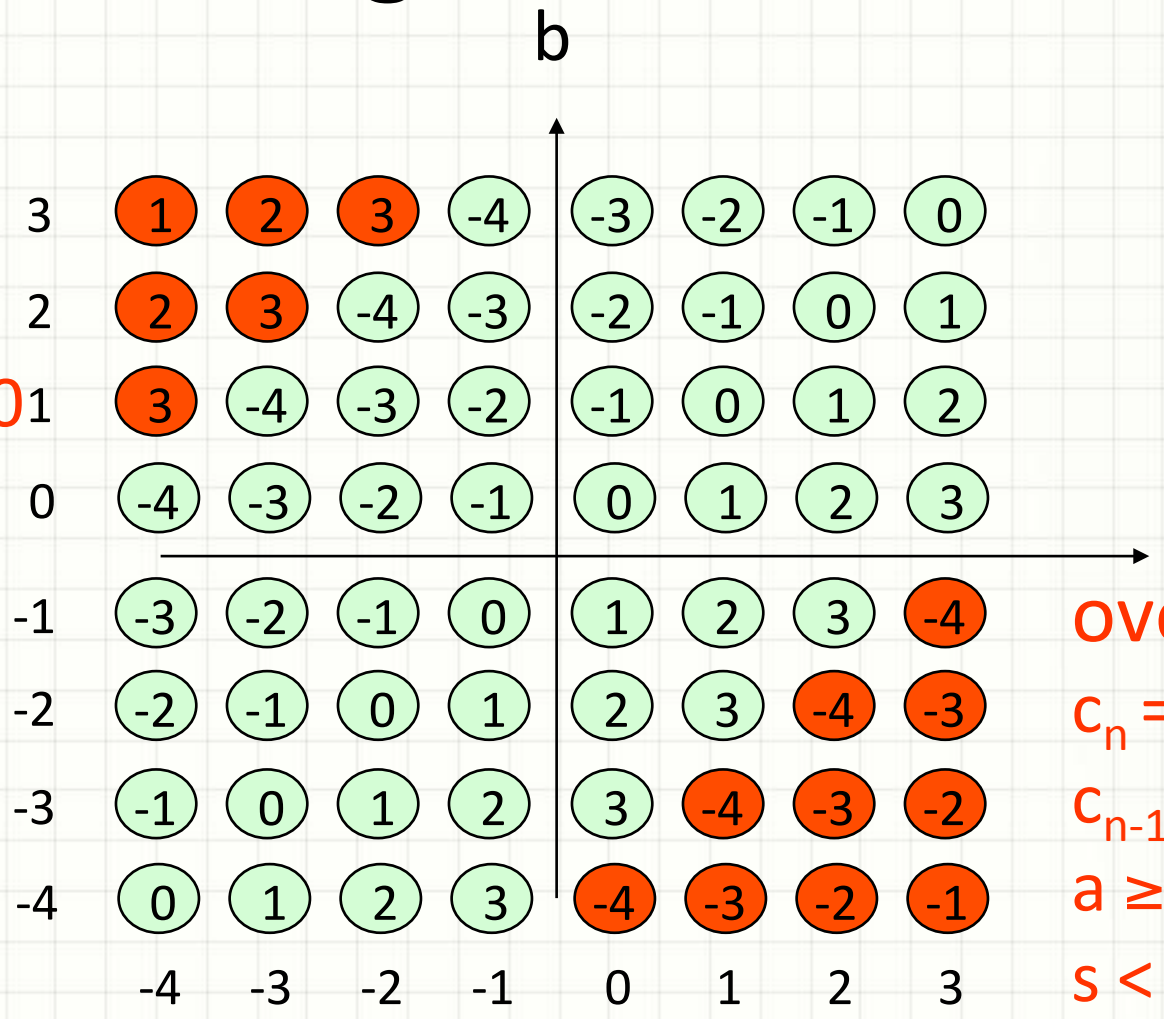
overflow

$c_n = 1$

$c_{n-1} = 0$

$a < 0, b \geq 0$

$s \geq 0$



overflow

$c_n = 0$

$c_{n-1} = 1$

$a \geq 0, b < 0$

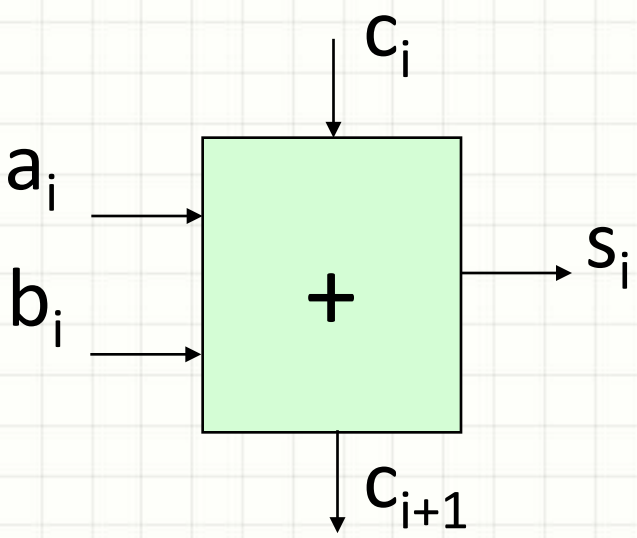
$s < 0$

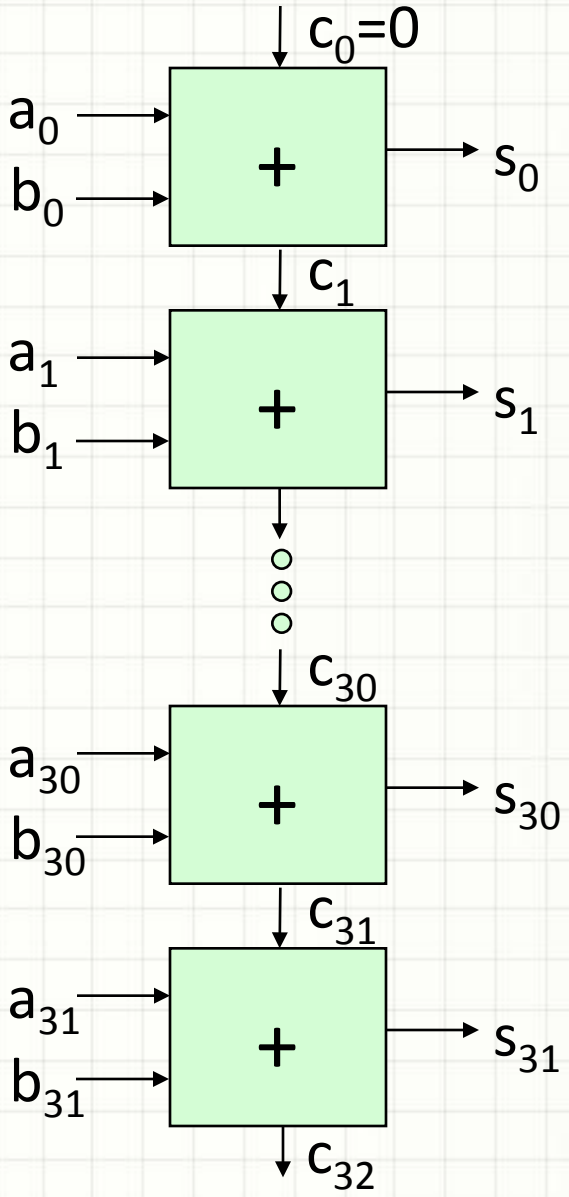
general : $c_n \neq c_{n-1}$, sign(a) ≠ sign(b) = sign(s)

Adder circuit

n bit adder =

array of n full adders





One bit adder implementation

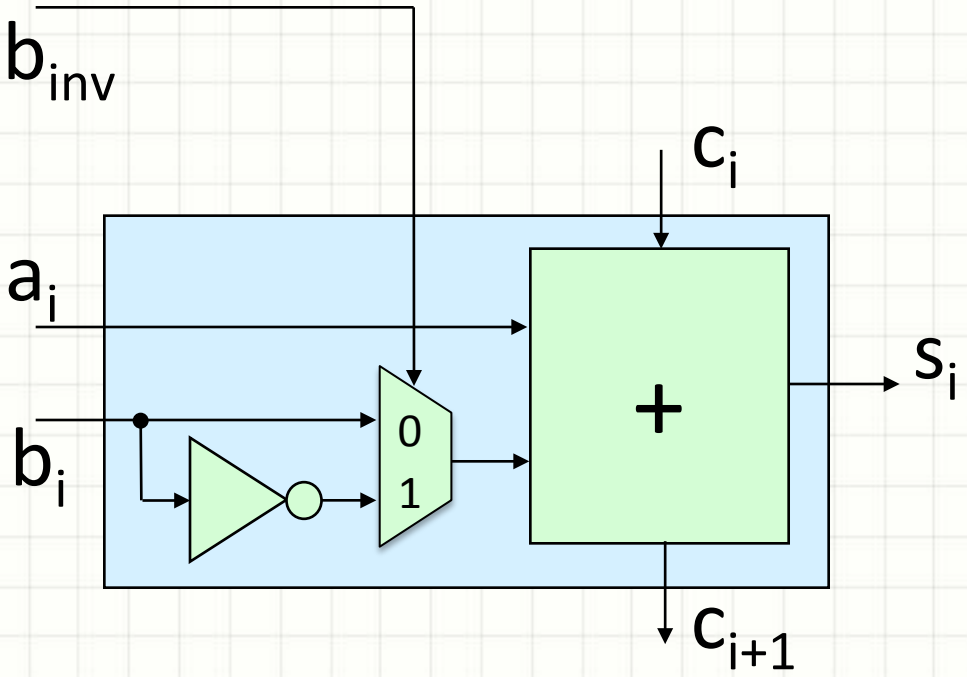
a_i	b_i	c_i	c_{i+1}	s_i
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

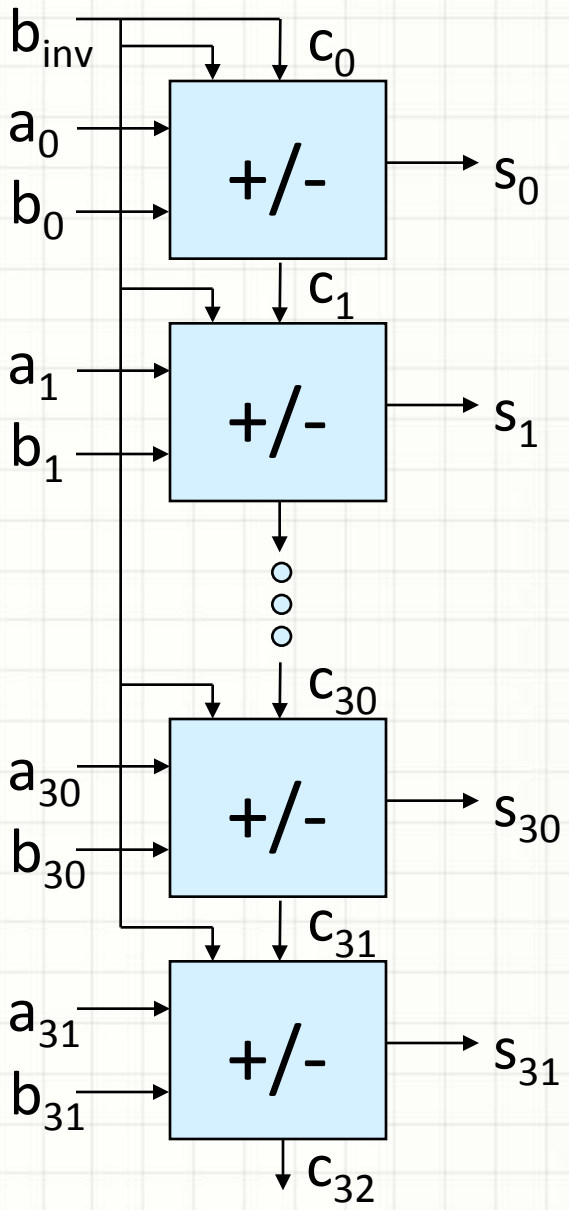
$$s_i = a_i \oplus b_i \oplus c_i$$

$$c_{i+1} = a_i b_i + a_i c_i + b_i c_i$$

Combining addition and subtraction

Use a multiplexer circuit and initial carry





Boolean expressions for adder

$s_i = a_i' \; b_i' \; c_i + a_i' \; b_i \; c_i' + a_i \; b_i' \; c_i' + a_i \; b_i \; c_i$

$= a_i \oplus b_i \oplus c_i$

alternatively,

$s_i = (a_i' \; b_i' + a_i \; b_i) \; c_i + (a_i \; b_i' + a_i' \; b_i) \; c_i'$

$= t_i' \; c_i + t_i \; c_i'$ where $t_i = a_i \; b_i' + a_i' \; b_i$

$= (a_i \oplus b_i) \oplus c_i$

$c_{i+1} = a_i \; b_i + a_i \; c_i + b_i \; c_i = a_i \; b_i + (a_i + b_i) \; c_i$

Performance considerations

- Delay along a path depends on
  - the number of gates in the path
- Delay of an individual gate depends on
  - the number of inputs to the gate
- Path(s) with maximum delay is(are) called "critical path(s)"
- Current design: simple and slow
  - clever changes can improve performance

Speed of ripple carry adder (carry propagate adder)

- Ripple is caused because c_{i+1} is generated from c_i

$$c_{i+1} = b_i c_i + a_i c_i + a_i b_i$$

$$c_1 = b_0 c_0 + a_0 c_0 + a_0 b_0$$
$$c_2 = b_1 c_1 + a_1 c_1 + a_1 b_1$$
$$c_3 = b_2 c_2 + a_2 c_2 + a_2 b_2$$
$$c_4 = b_3 c_3 + a_3 c_3 + a_3 b_3$$

- Can c_{i+1} be generated directly from $a_0 .. a_i$, $b_0 .. b_i$, and c_0?

c_{i+1} in terms of $a_0 .. a_i$, $b_0 .. b_i$, and c_0

$c_1 = b_0 c_0 + a_0 c_0 + a_0 b_0$

$c_2 = b_1 c_1 + a_1 c_1 + a_1 b_1 =$

$b_0b_1c_0+a_0b_1c_0+a_0b_0b_1+a_1b_0c_0+a_0a_1c_0+a_0a_1b_0+a_1b_1$

$c_3 = b_2 c_2 + a_2 c_2 + a_2 b_2 =$

$b_0b_1b_2c_0+a_0b_1b_2c_0+a_0b_0b_1b_2+a_1b_0b_2c_0+a_0a_1b_2c_0+a_0a_1b_0b_2+$

$a_1b_1b_2 +a_2b_0b_1c_0+a_0a_2b_1c_0+a_0a_2b_0b_1+a_1a_2b_0c_0+a_0a_1a_2c_0+$

$a_0a_1a_2b_0+a_1a_2b_1 +a_2b_2$

$c_4 = b_3 c_3 + a_3 c_3 + a_3 b_3 =$ .....

c_{32} will have 4 billion terms !!

Effect of large fanin and fanout ??

Carry-lookahead adder

- An approach in-between our two extremes
- Idea:
  - If we didn't know the value of carry-in, what could we do?
  - When would we always generate a carry?

    $g_i = a_i b_i$
  - When would we propagate the carry?

    $p_i = a_i + b_i$

Carry-lookahead adder

Express carries using p's and g's

$c_1 = p_0 c_0 + g_0$

$c_2 = p_1 c_1 + g_1 = p_1 p_0 c_0 + p_1 g_0 + g_1$

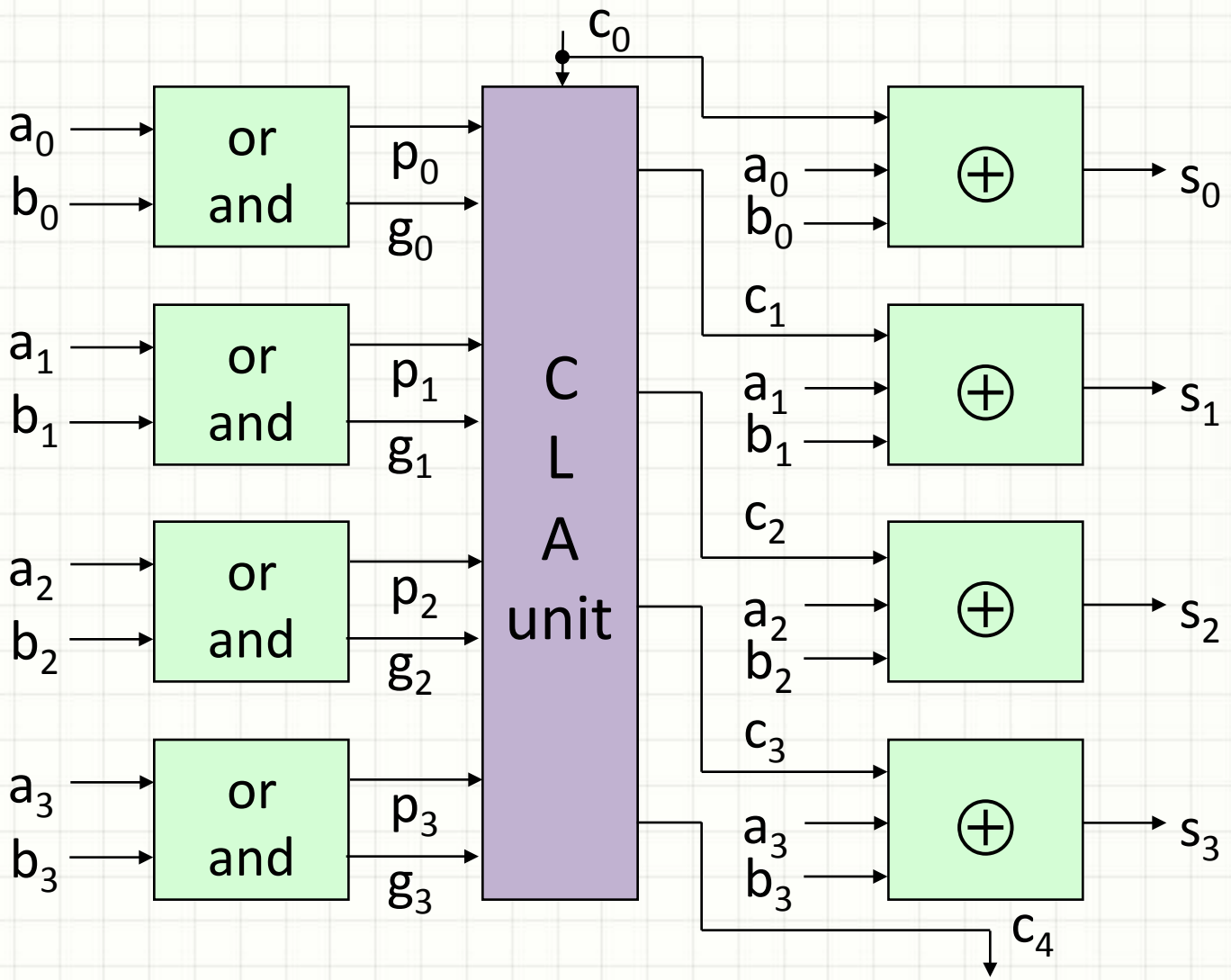
$c_3 = p_2 c_2 + g_2 = p_2 p_1 p_0 c_0 + p_2 p_1 g_0 + p_2 g_1 + g_2$

$c_4 = p_3 c_3 + g_3 =$

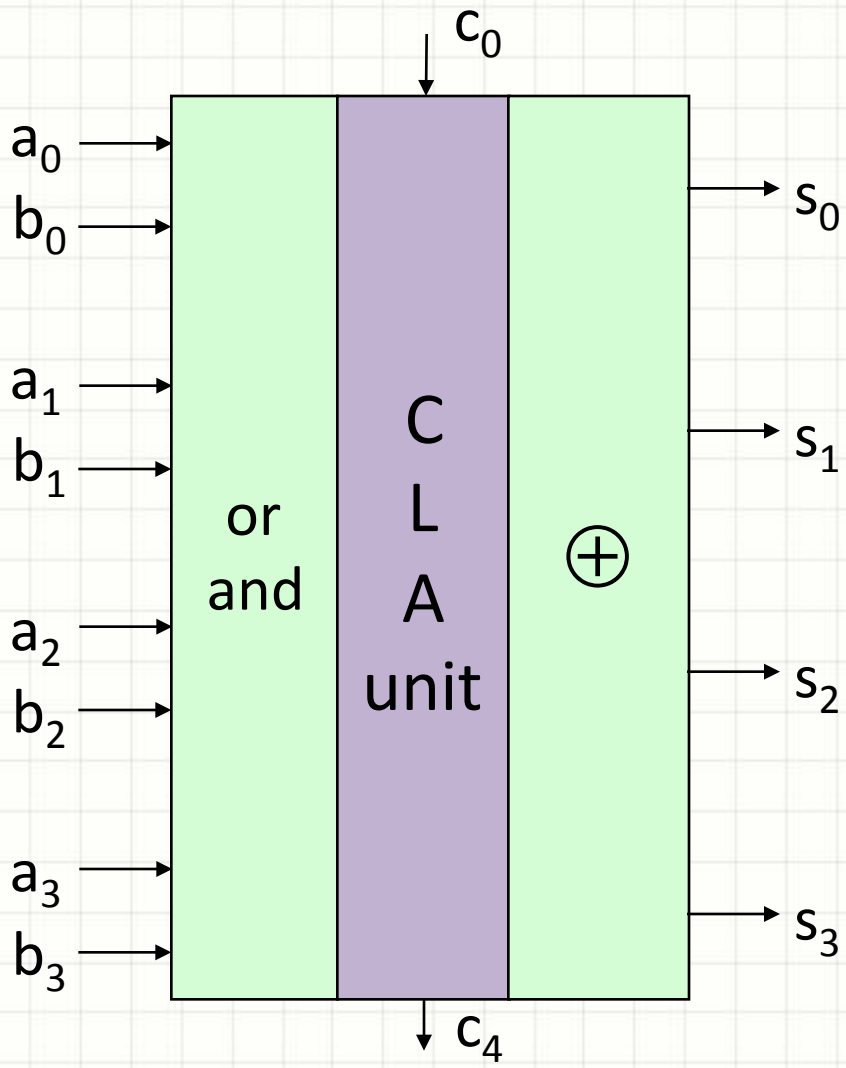
$p_3 p_2 p_1 p_0 c_0 + p_3 p_2 p_1 g_0 + p_3 p_2 g_1 + p_3 g_2 + g_3$

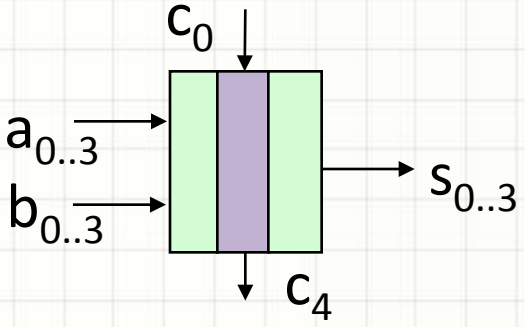
Feasible! To what extent?

4 bit CLA adder



4 bit CLA adder abstraction

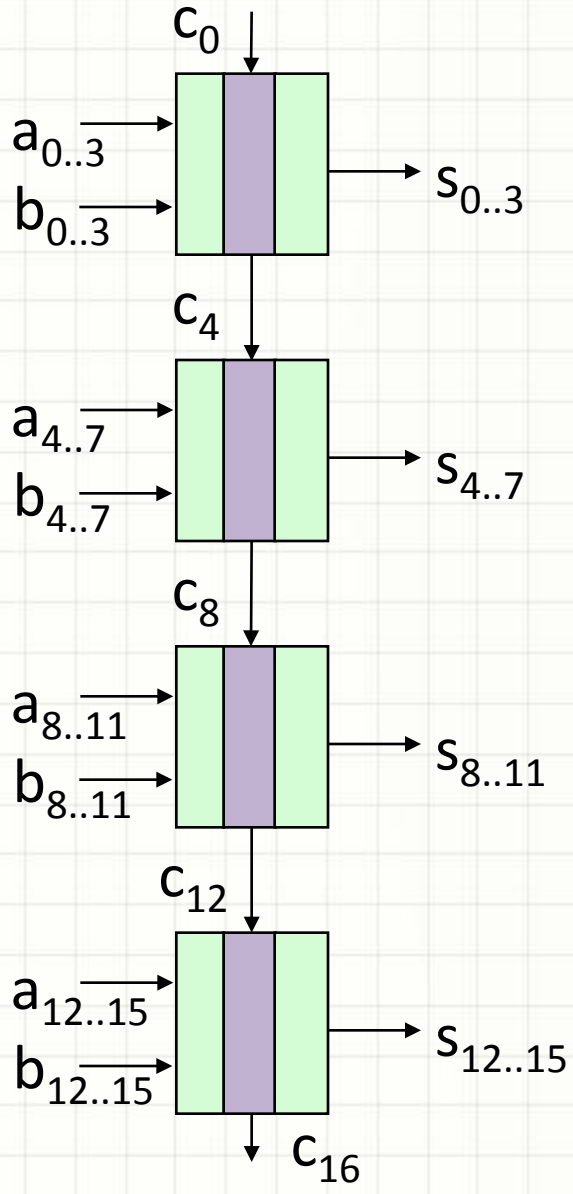




abstraction

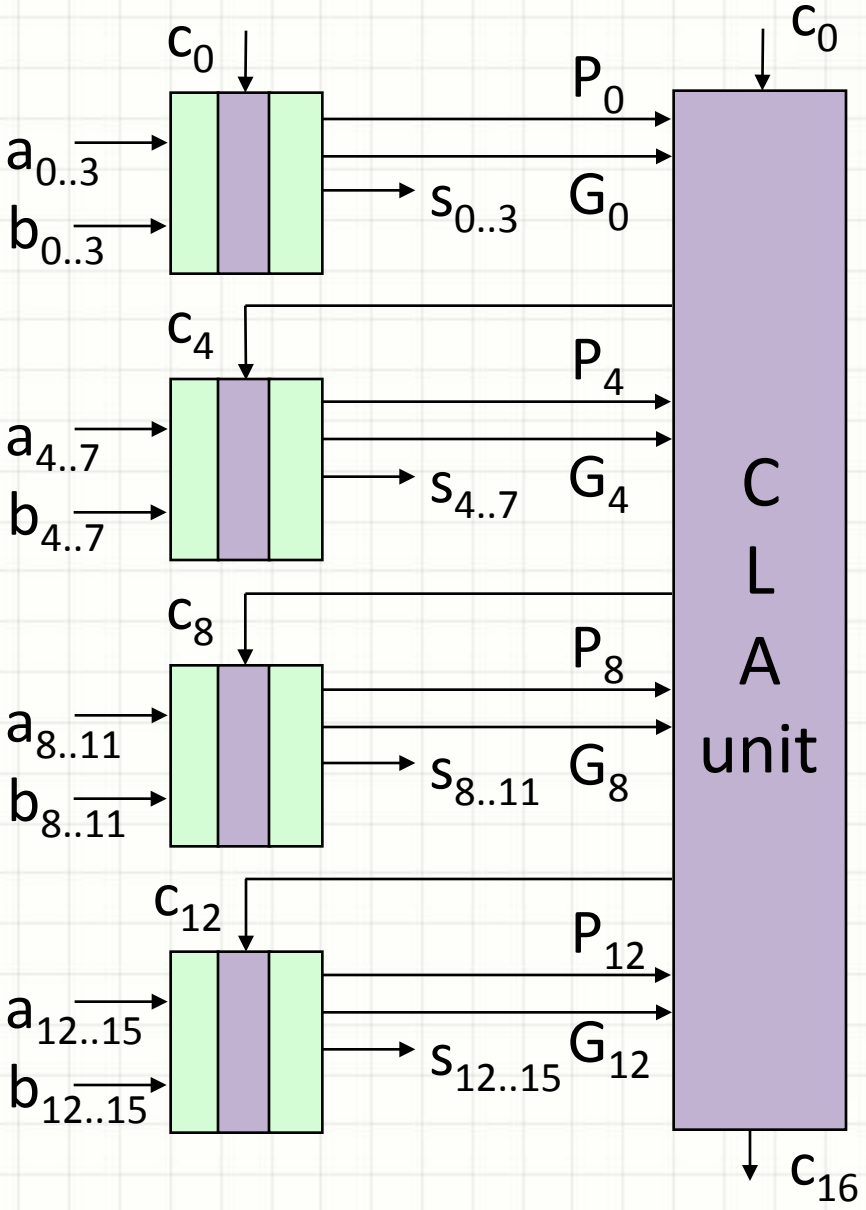
16 bit addition with 4 bit CLAs

partial rippling of carry



2 levels of look ahead

no rippling of carry



Group propagate & generate

$c_1 = p_0 c_0 + g_0$

$c_2 = p_1 c_1 + g_1 = p_1 p_0 c_0 + p_1 g_0 + g_1$

$c_3 = p_2 c_2 + g_2 = p_2 p_1 p_0 c_0 + p_2 p_1 g_0 + p_2 g_1 + g_2$

$c_4 = p_3 c_3 + g_3 =$

$p_3 p_2 p_1 p_0 c_0 + p_3 p_2 p_1 g_0 + p_3 p_2 g_1 + p_3 g_2 + g_3$

$P_0 = p_3 p_2 p_1 p_0$

$G_0 = p_3 p_2 p_1 g_0 + p_3 p_2 g_1 + p_3 g_2 + g_3$

$c_4 = P_0 c_0 + G_0$

Group propagate & generate

$$P_i = p_{i+3}\, p_{i+2}\, p_{i+1}\, p_i$$

$$G_i = p_{i+3}\, p_{i+2}\, p_{i+1}\, g_i + p_{i+3}\, p_{i+2}\, g_{i+1} + p_{i+3}\, g_{i+2} + g_{i+3}$$

$$c_4 = P_0\, c_0 + G_0$$

$$c_8 = P_4\, P_0\, c_0 + P_4\, G_0 + G_4$$

$$c_{12} = P_8\, P_4\, P_0\, c_0 + P_8\, P_4\, G_0 + P_8\, G_4 + G_8$$

$$c_{16} = P_{12}\, P_8\, P_4\, P_0\, c_0 + P_{12}\, P_8\, P_4\, G_0 + P_{12}\, P_8\, G_4 + P_{12}\, G_8 + G_{12}$$

THANKS