Arithmetic Circuits

Adders

Virendra Singh

Professor

Computer Architecture and Dependable Systems Lab

Department of Electrical Engineering

Indian Institute of Technology Bombay

http://www.ee.iitb.ac.in/~viren/

E-mail: viren@ee.iitb.ac.in

CS-226: Digital Logic Design

Lecture 17-B: 08 March 2021

CADSL

Half-Adder Adds two Bits

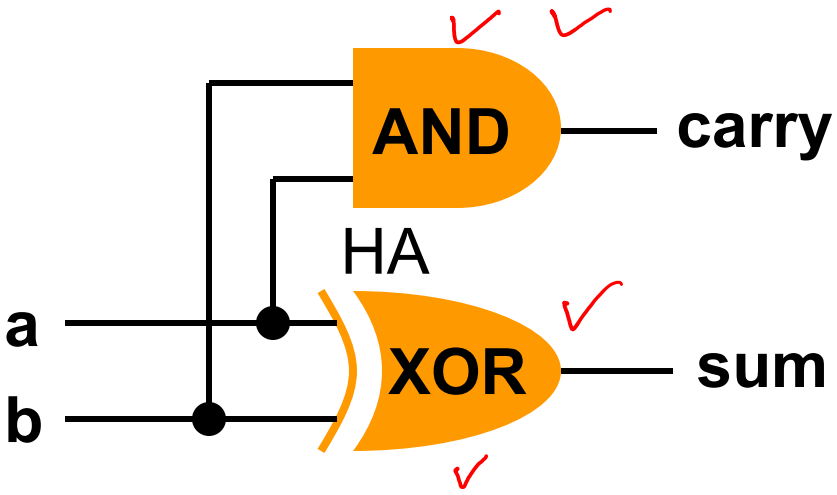
"half" because it has no carry input

- Adding two bits:

a	b	a + b
0	0	00
0	1	01
1	0	01
1	1	10

carry sum

$a \cdot b$ $\bar{a} \cdot b + a \cdot \bar{b}$ $a \oplus b$



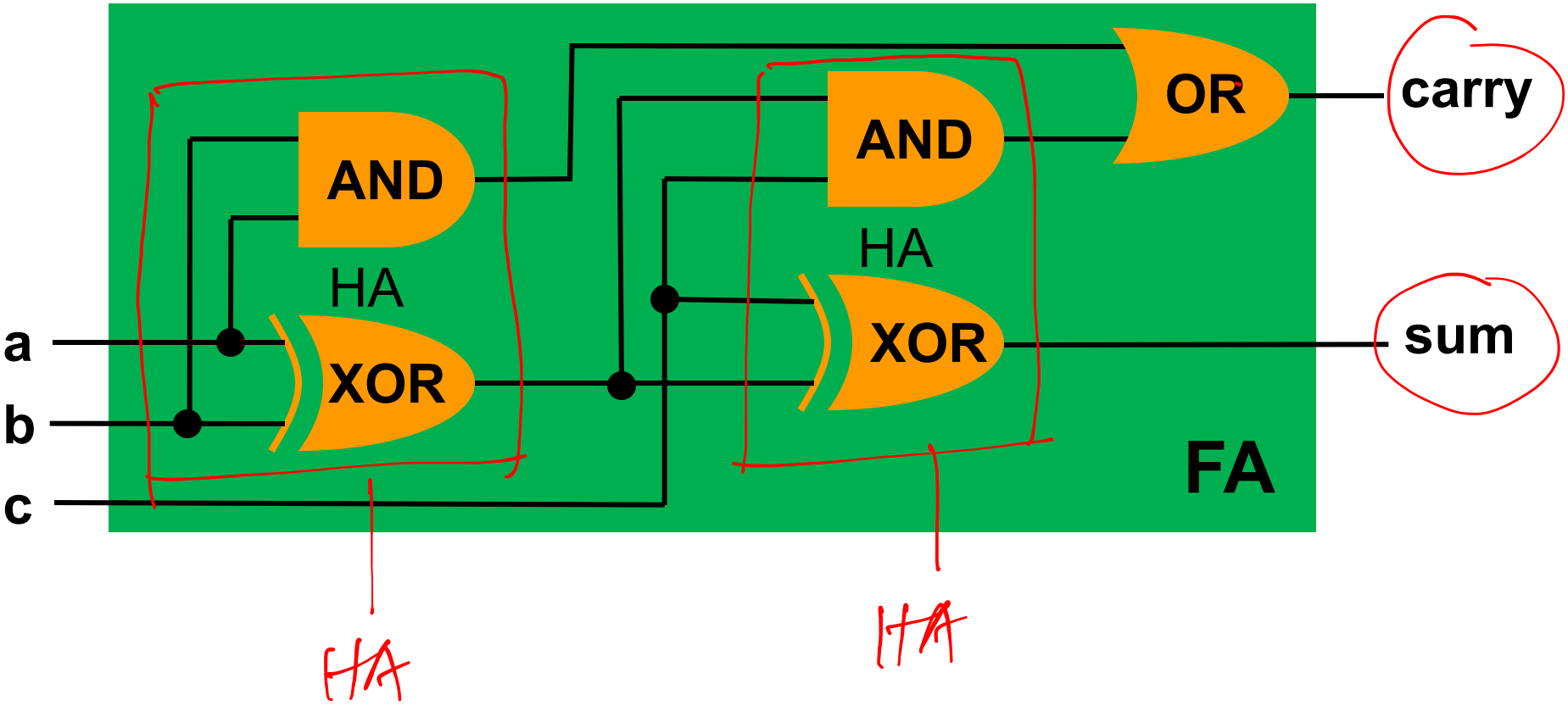
Full Adder: Include Carry Input

a	b	c	s = a + b + c	
			Decimal value	Binary value
0	0	0	0	00
0	0	1	1	01
0	1	0	1	01
0	1	1	2	10
1	0	0	1	01
1	0	1	2	10
1	1	0	2	10
1	1	1	3	11

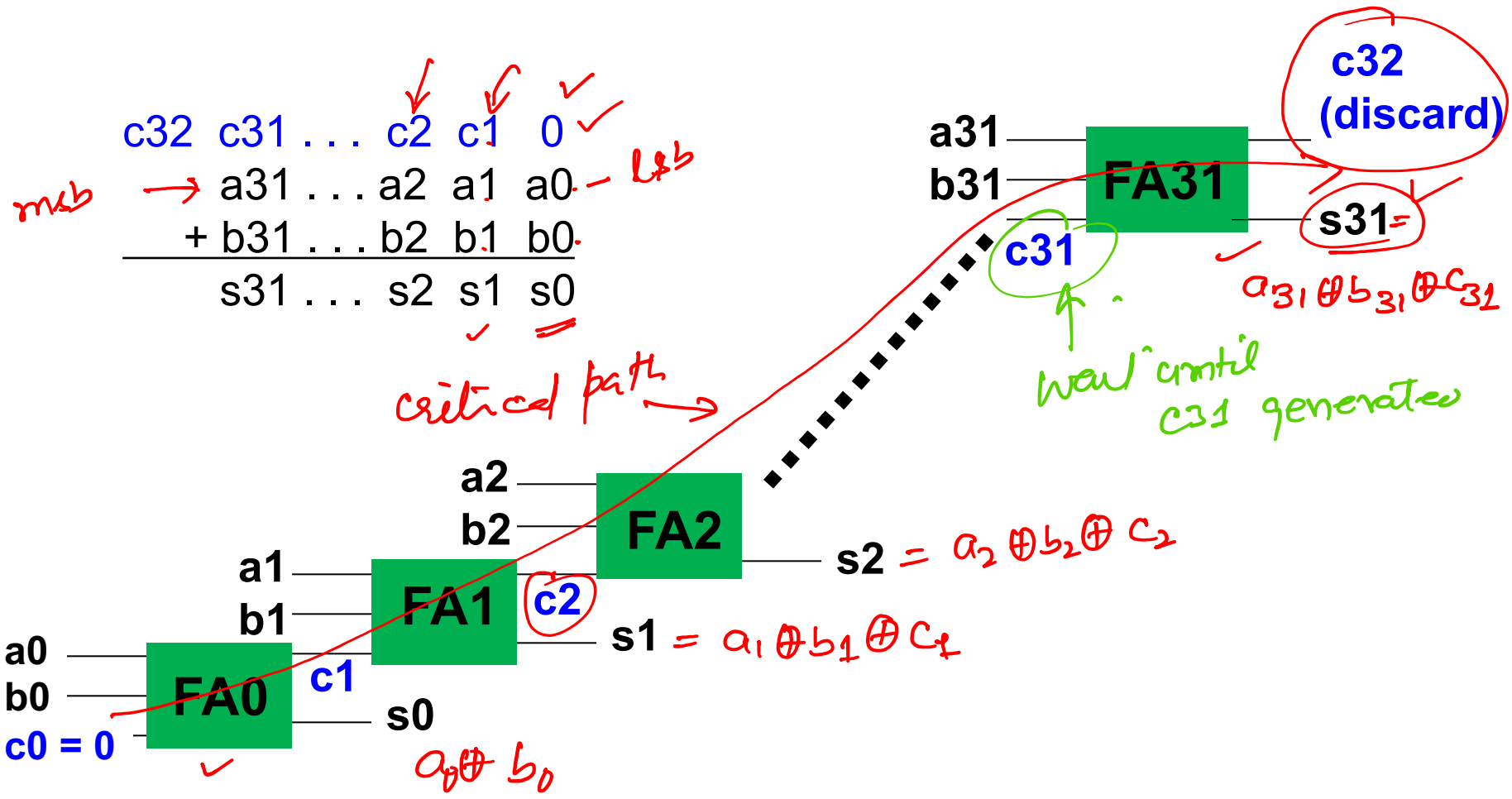
CARRY

SUM

Full-Adder Adds Three Bits

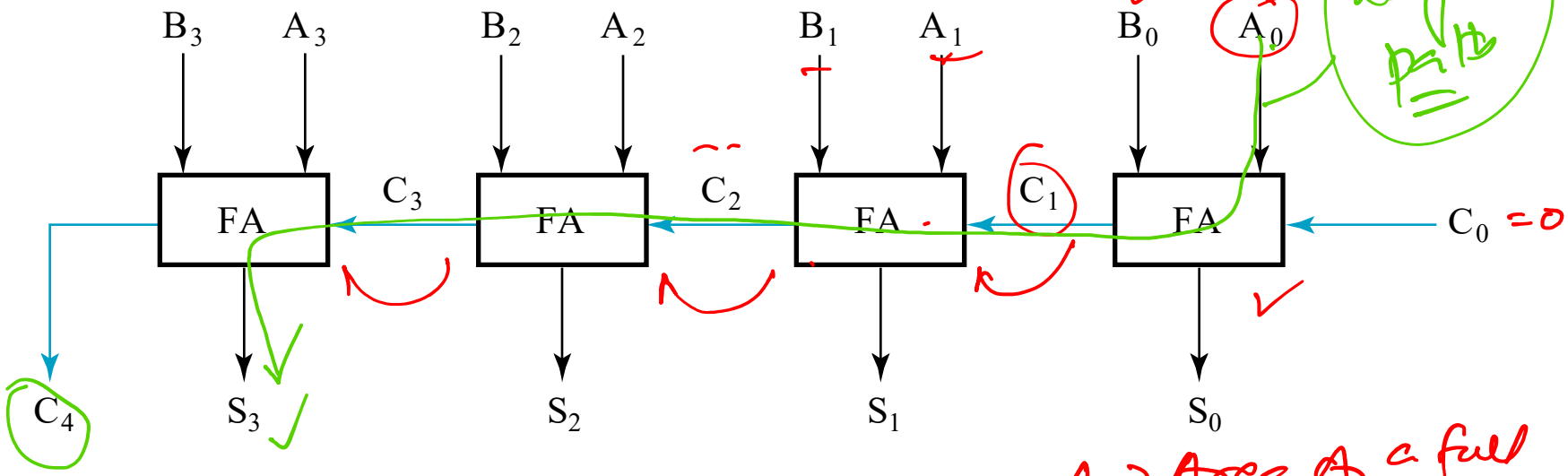


32-bit Ripple-Carry Adder



4-bit Ripple-Carry Binary Adder

- A four-bit Ripple Carry Adder made from four 1-bit Full Adders:



How Fast is Ripple-Carry Adder?

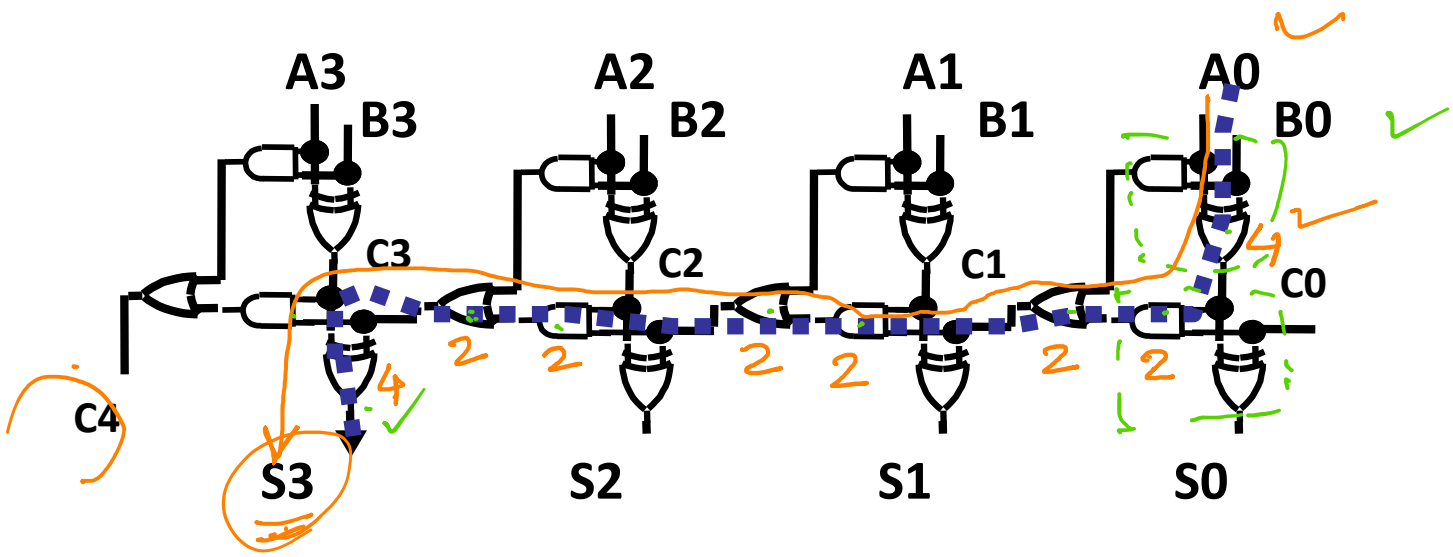
- Longest delay path (critical path) runs from (a0, b0) to sum31/C32
- Suppose delay of full-adder is 100ps
- Critical path delay = 3,200ps

32 x 100 ps

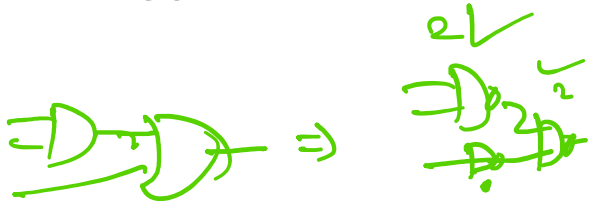
- Must use more efficient ways to handle carry

Carry Propagation & Delay

- Propagation delay:
  - Carry must ripple from LSB to MSB.
- The gate-level propagation path for a 4-bit ripple carry adder of the last example:



- Too slow for many bits (e.g. 32 or 64)



Carry Propagation & Delay

delay

CMOS → AND = N, OR = N, XOR = 2N

2 input AND = 2

2 I/P OR = 2

2 i/p XOR = 4

XOR → 4 + 2+2 + 2+2 + 2+2 + 4

AND OR

XOR.

2 × 4

8 + 4 × 3 = 20

Generalize.

8 + 4(n-1)

32 bit 8 + 4 × 31 = 8 + 124 = 132 delay

1 unit = 10 ps ⇒ 132 × 10 ps = 1.32 ns

Carry Propagation & Delay

$8 + 4(n-1)$

64 bit

$8 + 4 \times 63 = 8 + 252 = 260$ units

delay n bit adder $\propto n$

If I want to compute addition in 1 ns

then

Can I compute 32 bit addition?

NO — 1.32 ns ✗

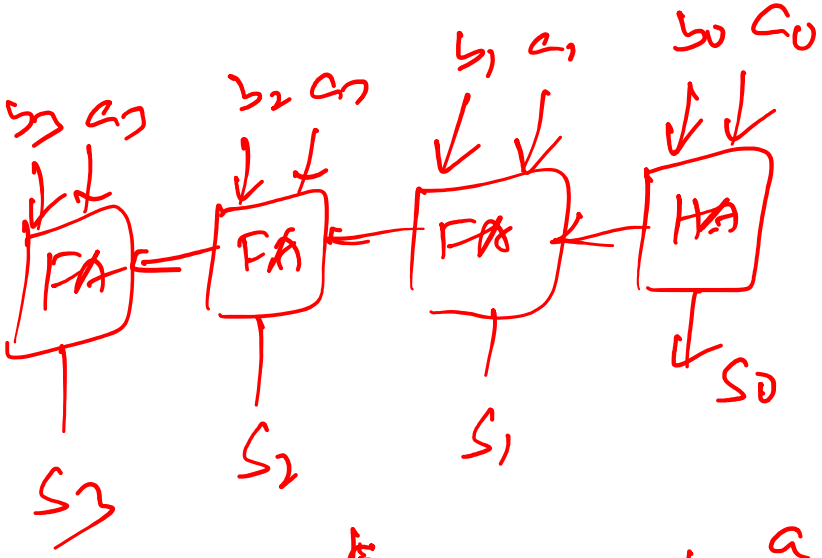
16 bit

64 bit bit Adders — Common

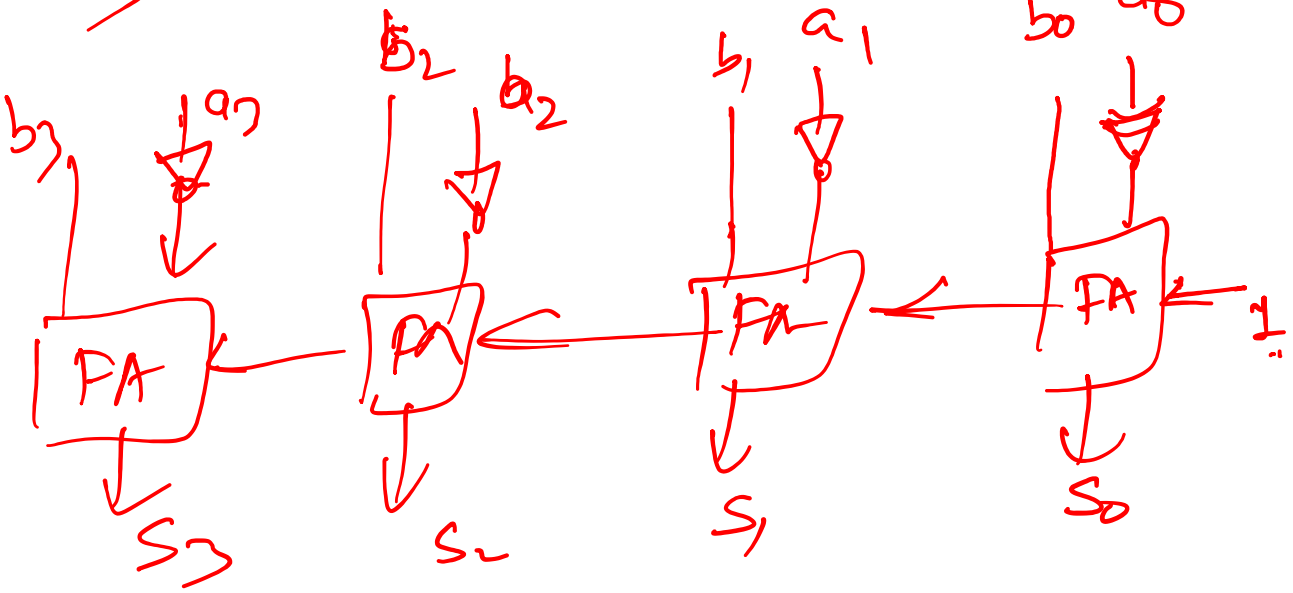
Ripple Carry Adder

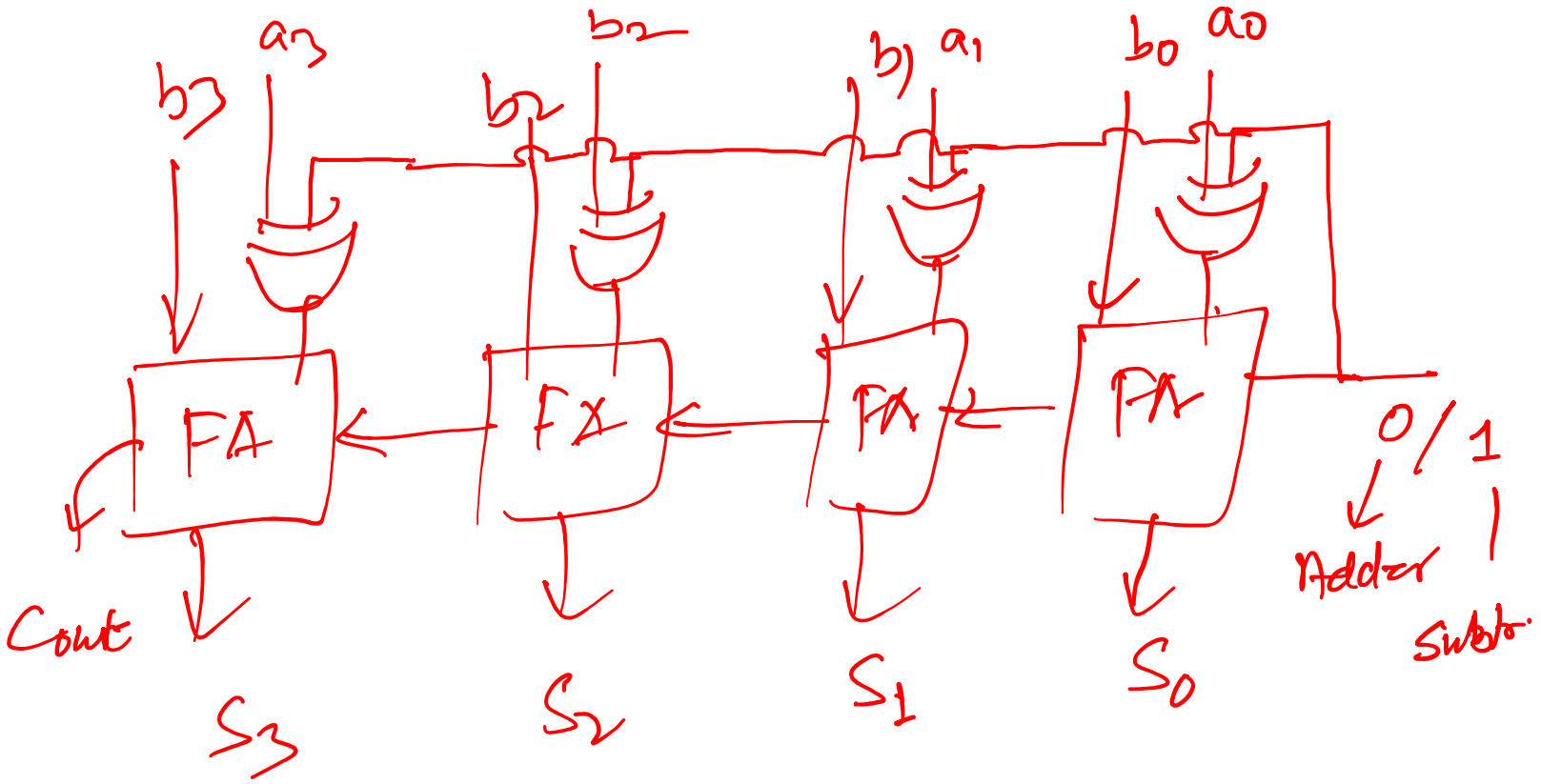
- Easy to create
  - Good hierarchy
  - Tileable structures
- Small hardware — n Full Adders
- Slow! — delay $\propto$ n (# bits)
  - Design is limited by the delays in propagating carry through all of the bitwise additions
  - The output bit at position m is not valid until after the carry out of the m-1 position is ready

Adder



Subtraction



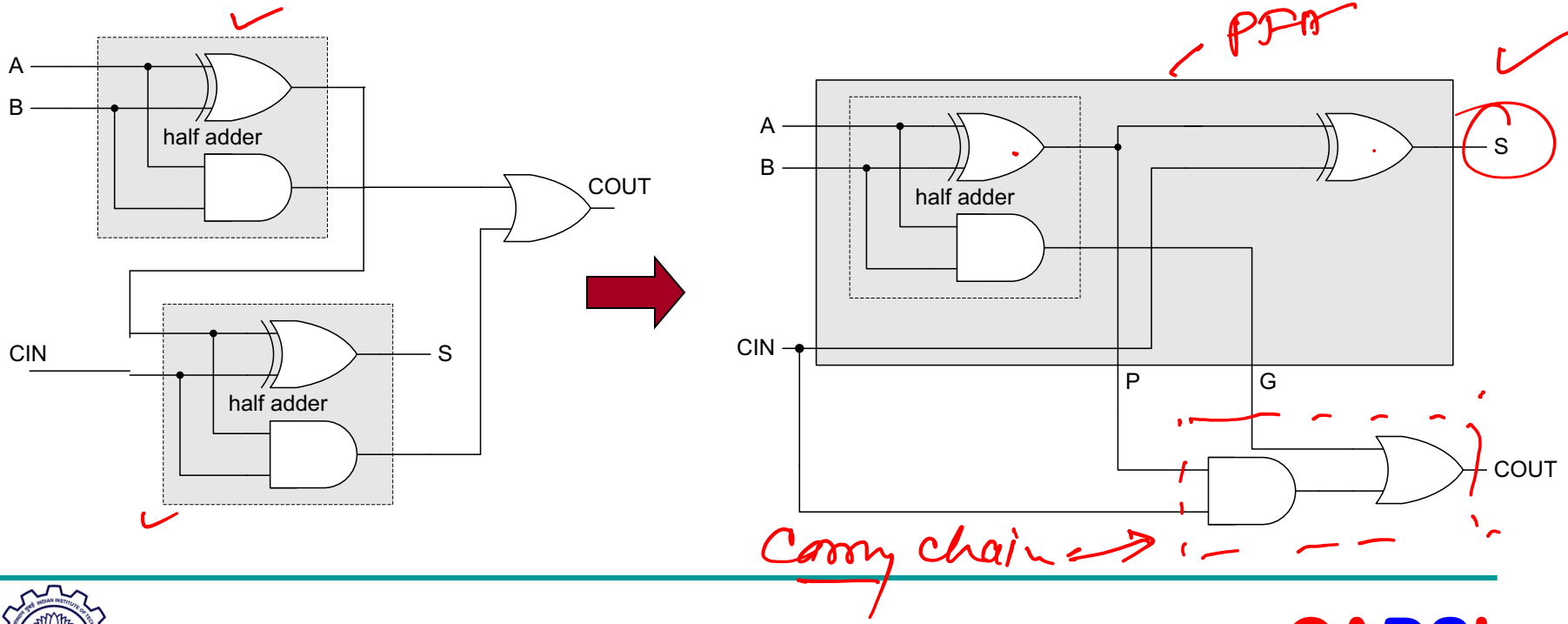


Adder/subtractor circuit

FAST ADDERS

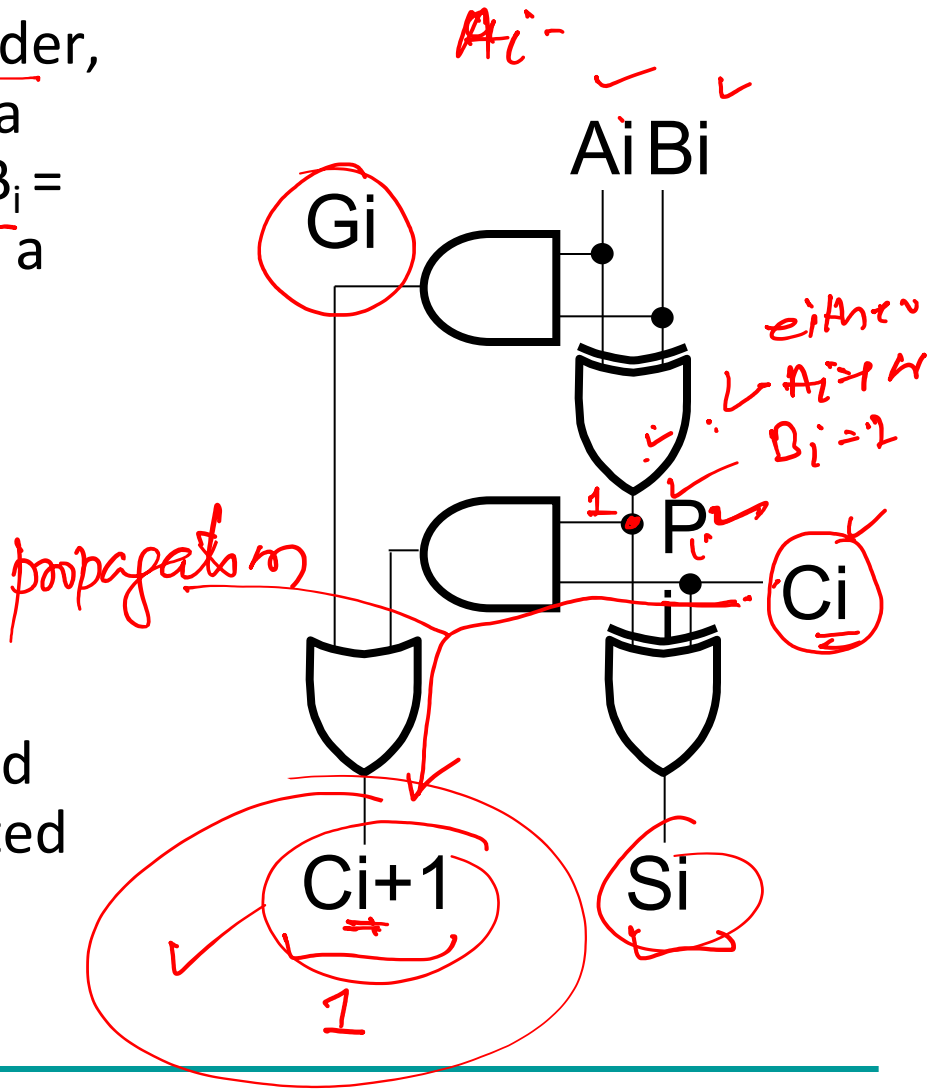
Faster Addition

- For big adders, the carry-chain is very long
- Separate the carry chain and sum logic
- Partial Full Adders
  - Contain only the sum part of a FA



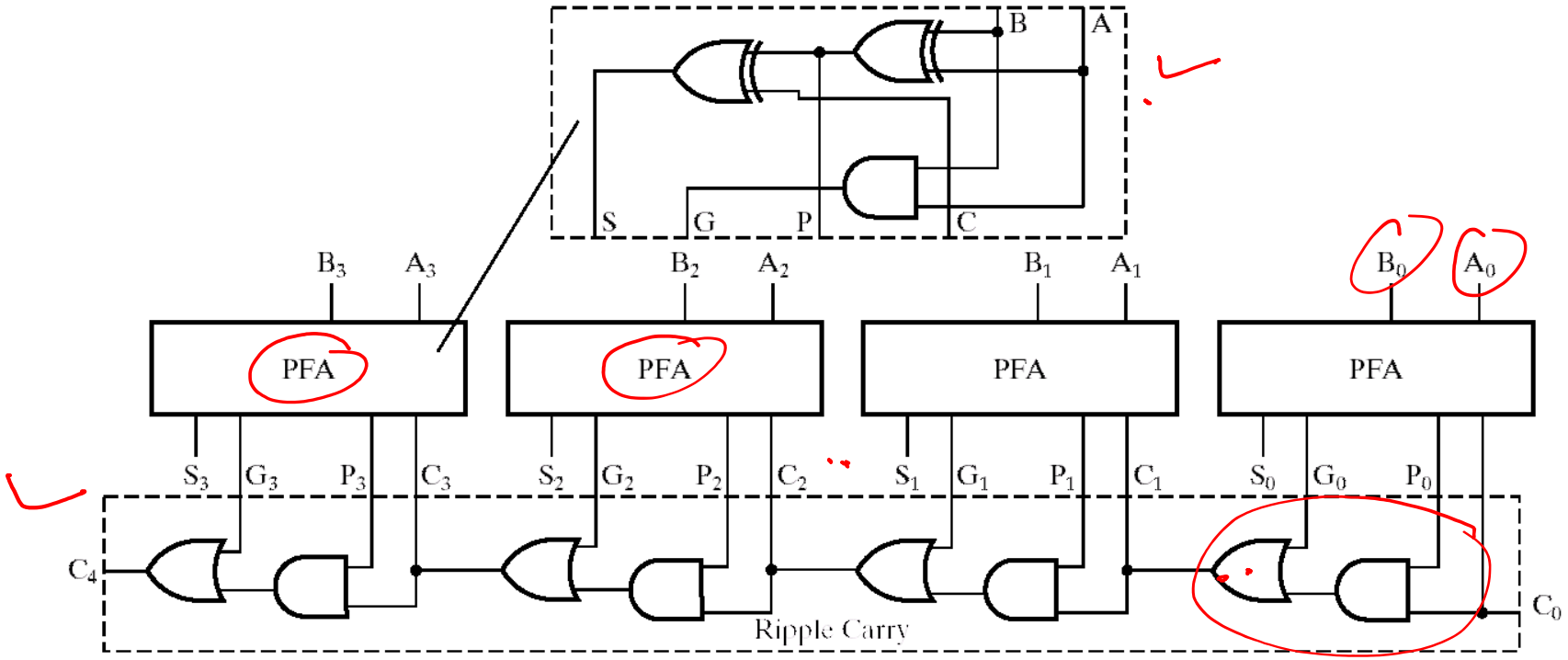
Carry Lookahead

- Given Stage i from a Full Adder, we know that there will be a carry generated when $A_i = B_i =$ "1", whether or not there is a carry-in.
- Alternately, there will be a carry propagated if the "half-sum" is "1" and a carry-in, C_i occurs.
- These two signal conditions are called *generate*, denoted as G_i, and *propagate*, denoted as P_i respectively and are identified in the circuit:



Reassembling RCA Using PFAs

- Can create ripple-carry adder with PFAs
  - G: **Generates** a carry at this position
  - P: **Propagates** a carry through this position



Carry Lookahead (continued)

- In the ripple carry adder:
  - G_i, P_i, and S_i are local to each cell of the adder
  - C_i is also local each cell
- In the carry lookahead adder, in order to reduce the length of the carry chain, C_i is changed to a more global function spanning multiple cells
- Defining the equations for the Full Adder in term of the P_i and G_i:

$$P_i = A_i \oplus B_i \qquad G_i = A_i B_i$$

$$S_i = P_i \oplus C_i \qquad C_{i+1} = G_i + P_i C_i$$

local carry generation

carry propagation

Carry Lookahead Development

- Flatten equations for carry using G_i and P_i terms for less significant bits
- Beginning at the cell 0 with carry in C_0:

$$C1 = G0 + P0\ C0$$

$G_0 = A_0 B_0$

$P_0 = A_0 + B_0$

$P_0 = A_0 \oplus B_0$

$C_1 = G_0 + P_0 C_0$

$C_{i+1} = G_i + P_i \cdot C_i$

$C_2 = G_1 + P_1 C_1$

$$C2 = G1 + P1\ C1 = G1 + P1(G0 + P0\ C0)$$

$G_1 = A_1 \cdot B_1$

$P_1 = A_1 + B_1$

$A_1 \oplus B_1$

$$= G1 + P1G0 + P1P0\ C0$$

Carry Lookahead Development

$G_2 = A_2 \cdot B_2$ $P_2 = A_2 \oplus B_2$ or $A_2 + B_2$

$$C3 = G2 + P2\,C2 = G2 + P2(G1 + P1G0 + P1P0\,C0)$$

$$= G2 + P2G1 + P2P1G0 + P2P1P0\,C0$$

2 logic level

$$C4 = G3 + P3\,C3 = G3 + P3G2 + P3P2G1 + P3P2P1G0 + P3P2P1P0\,C0$$

3 logic level

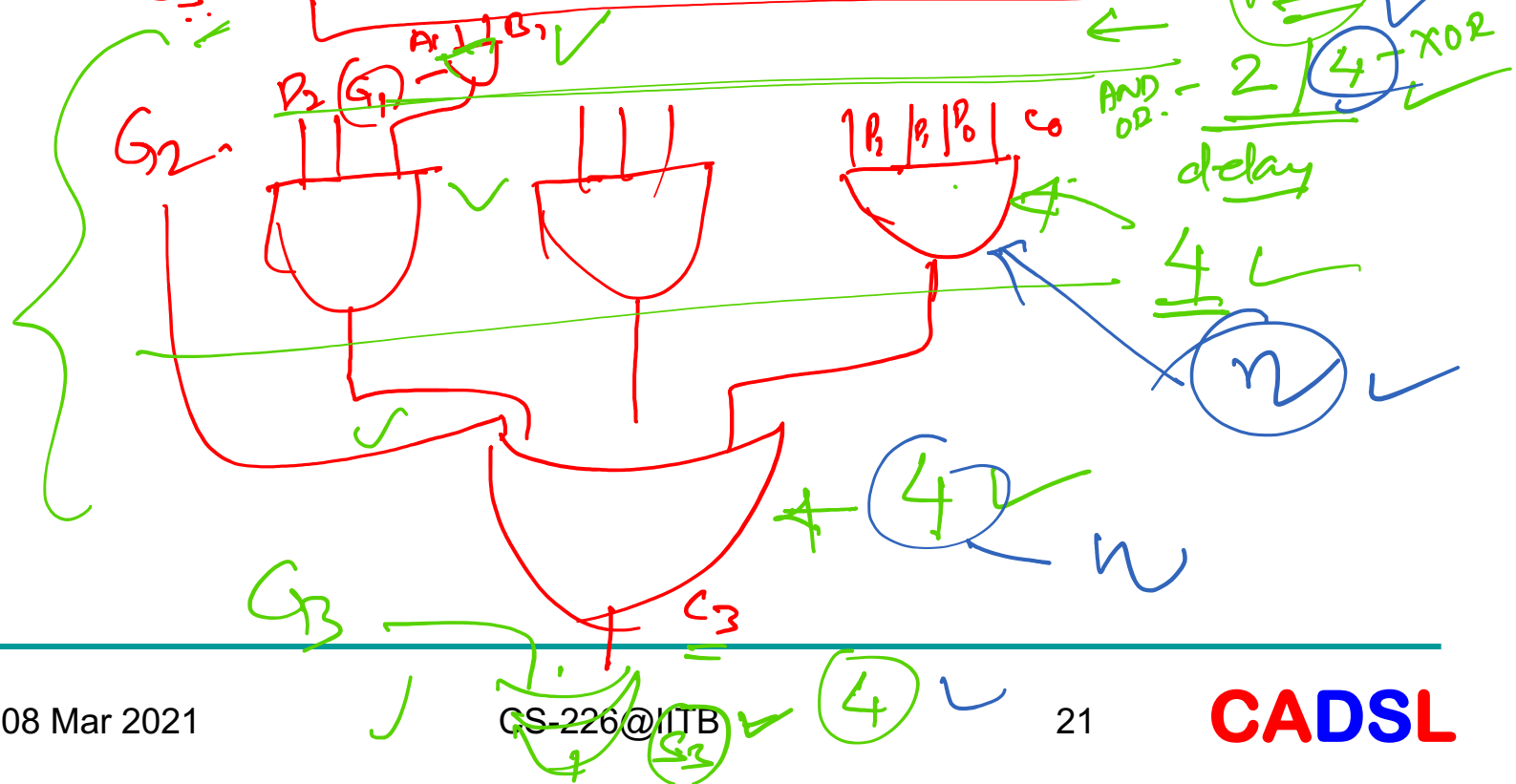
Carry Lookahead Adder

$C_4 = G_3 + P_3 G_2 + P_3 P_2 G_1 + P_3 P_2 P_1 G_0 + P_3 P_2 P_1 P_0 C_0$

$S_3 = G_3 \oplus C_3$

$C_3 = G_2 + P_2 G_1 + P_2 P_1 G_0 + P_2 P_1 P_0 C_0$

$G_2 = G_1 A_0 \cdot B_3$



Carry Lookahead Adder

$4 + 4 + 4 + 4 = 16$ unit

First XOR

XOR

4 bit

n bit

$4 + n + n + 4$

$= 8 + 2n$

$n = 32$

$= 8 + 2 \times 32 = 8 + 64 = 72$

1 unit = 10 ps

delay $= 72 \times 10 = 720$ ps $= .72$ ns

delay = $\delta + 2n$ ✓ (Carry logic ahead) CLA

delay = $\delta + 4(n-1)$ (Ripple Carry Adder)

delay $\propto \delta n$

n = # bits ✓

Scalability

Old TTL (BJT) — simplistic assumption delay = Const

delay = 4 X delay of a level
K.

$\leq 4K$ constant

Carry look ahead

Very expensive in area

little efficient in performance

delay

CMOS

Carry Look-Ahead Adder

- As usual, can trade area/power for speed
  - Multi-level logic (ripple carry) reduces area
  - Flattening carry logic increases speed
- Sometimes called CLA

Thank You