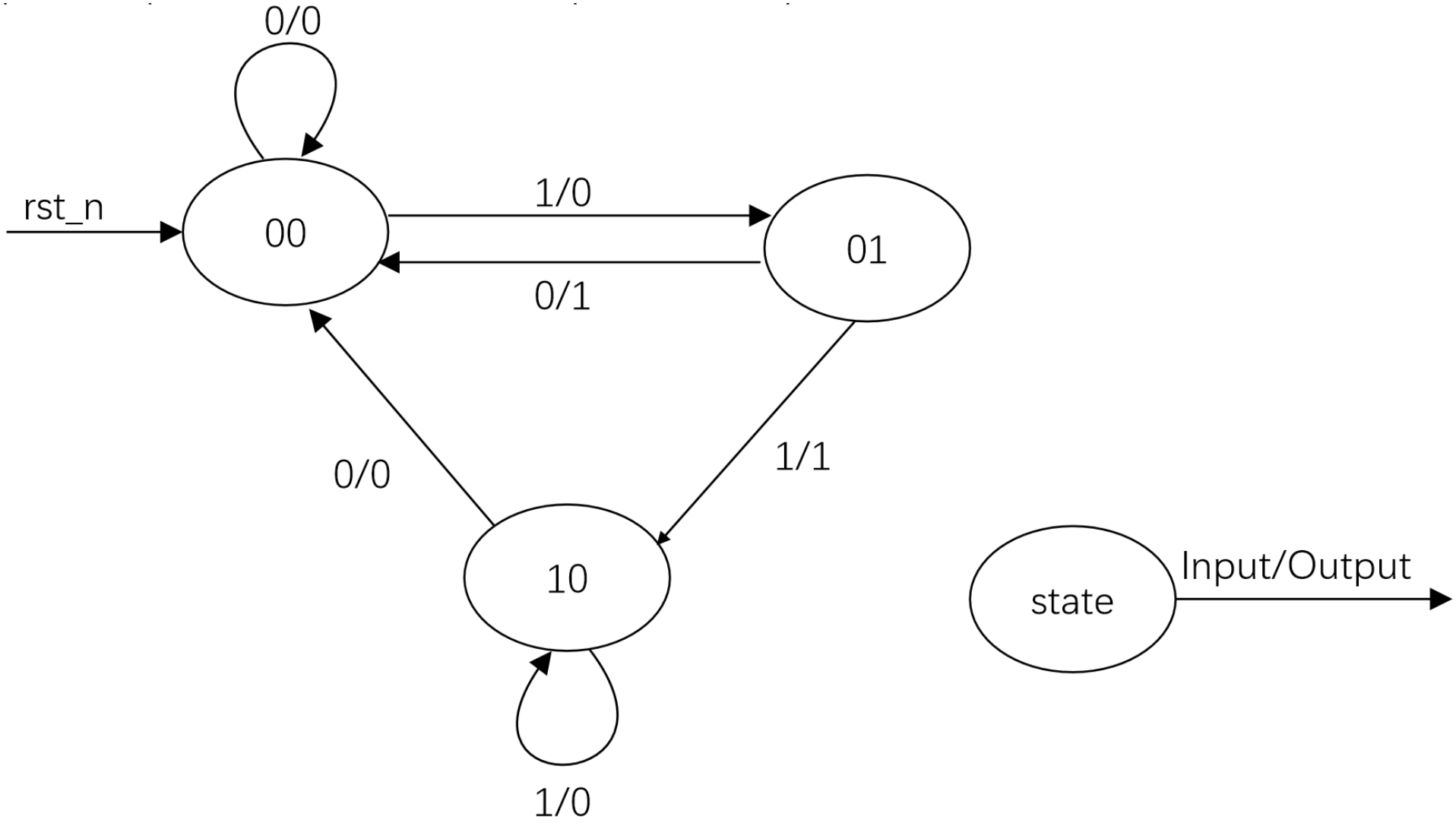
Computer Architecture

Discussion 6

CB

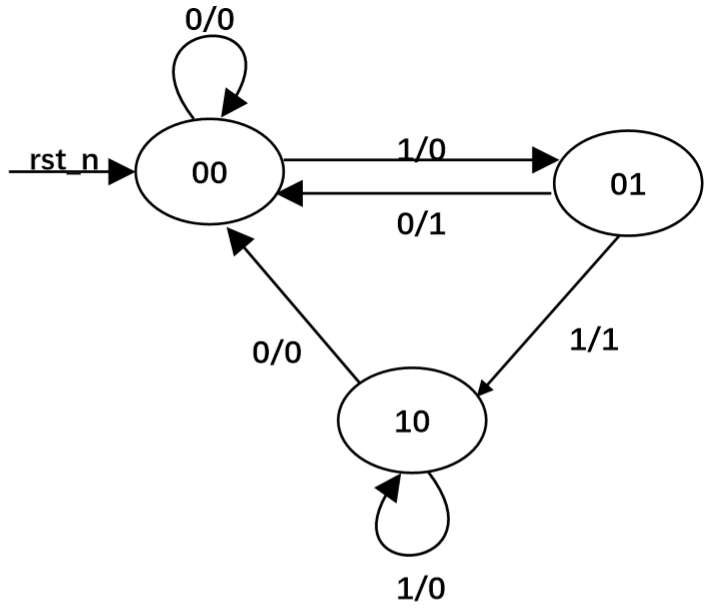
The FSM Diagram



From FSM to Truth Table

List the input, current state, next state and output

Current State		Input	Next State		Output
S_1	S_0	I	S_1'	S_0'	Y
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	0	0	1
0	1	1	1	0	1
1	0	0	0	0	0
1	0	1	1	0	0
1	1	0	X	X	X
1	1	1	X	X	X

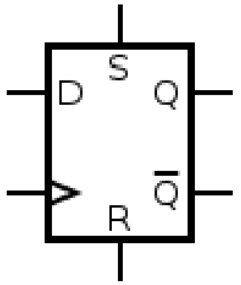


Represent the states

Represent a state by the output of several flip-flops

Store the state value

Can be changed at the rising edge of the clock

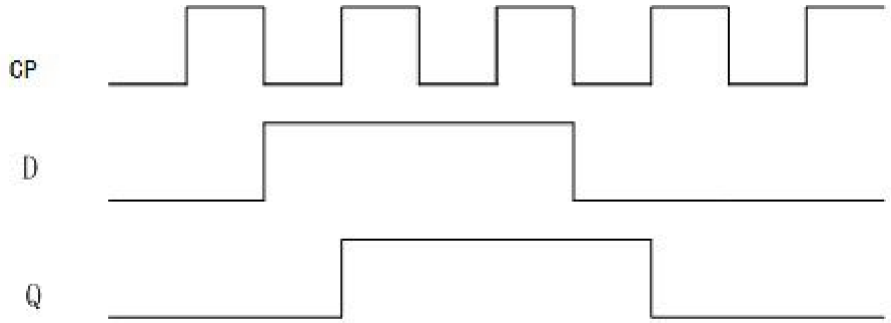


A D-Flip Flop

Output = Input at rising edge

Set S=R=0 to use it as expected

If S=R=1, output will always be 0



Simplifying the logical representations

- **Simplify the circuit with a Karnaugh Map**
- Useful for 3 to 4 input variables (e.g., your next experiment)
- Hamming Distance of neighbor columns/rows = 1 (e.g., 01 and 11)
- Group 1's and X's together (1x2, 2x1, 2x2, 1x4, 4x1, etc.) and simplify
- **S_0,S_1 are inputs of the 2 D-Flip flops, S_0'S_1' are the outputs**

S_0' Cancel S_0

S_0 \ S_1I	00	01	11	10
0	0	0	1	0
1	0	1	X	X

Cancel S_1

S_1'

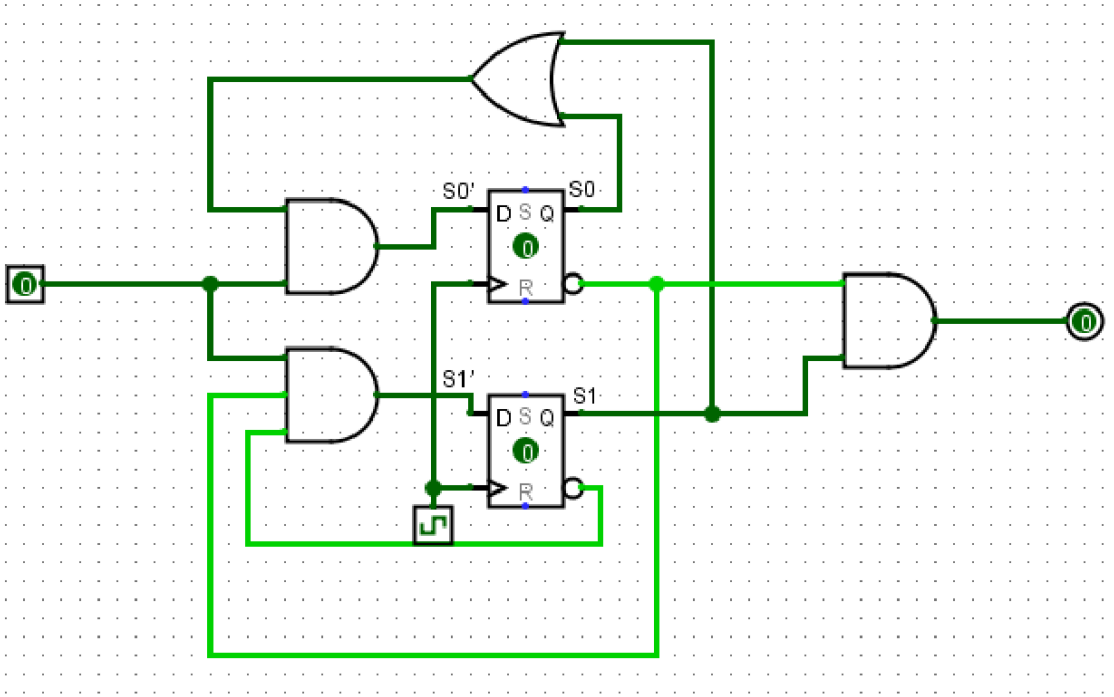
S_0 \ S_1I	00	01	11	10
0	0	1	0	0
1	0	0	X	X

Ignore redundant X

$$S_0' = S_0I + S_1I = (S_0 + S_1)I, S_1' = \overline{S_0}\,\overline{S_1}I, Y = \overline{S_0}S_1$$

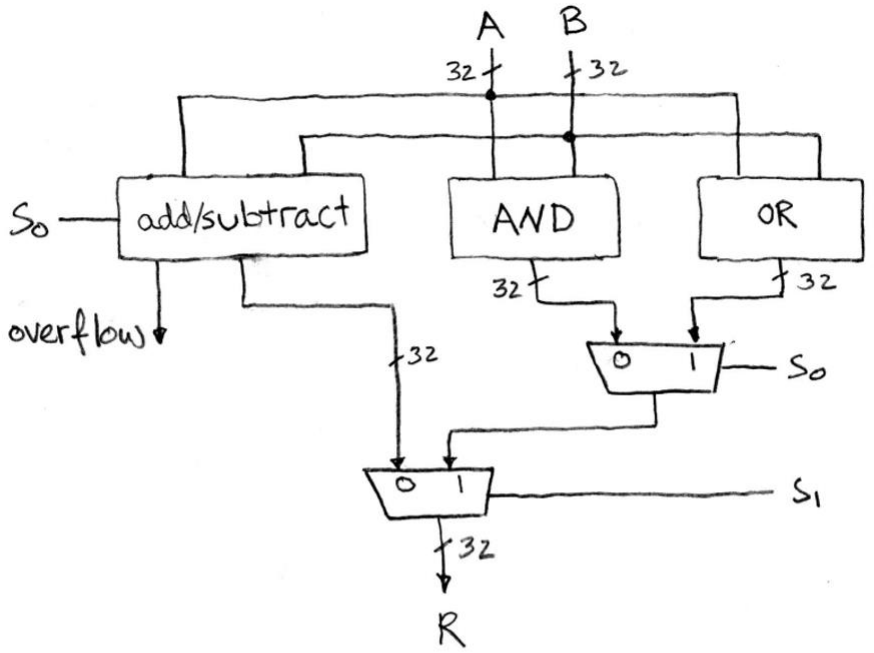
Connect the circuit

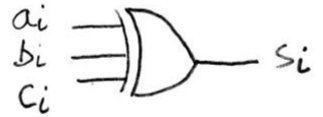
$$S_0' = S_0 I + S_1 I = (S_0 + S_1) I, S_1' = \overline{S_0}\,\overline{S_1} I, Y = \overline{S_0} S_1$$

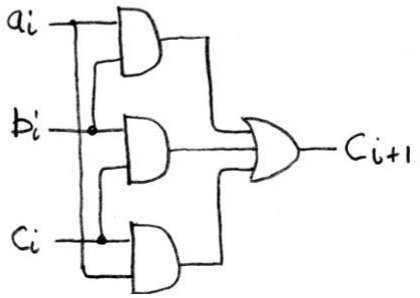


Designing an Adder/Subtractor

- An adder/subtractor is a circuit that is:
  - An adder when S_0=0
  - An subtractor when S_0=1
  - Required in ALU (top figure)
- Can be a cascade of 1-bit adders
- 1-bit adders(bottom figure):
  - Input: a, b, c0 (carry of last adder)
  - Output: s, c1 (sum and carry)
  - Design by truth table. S = a xor b xor c



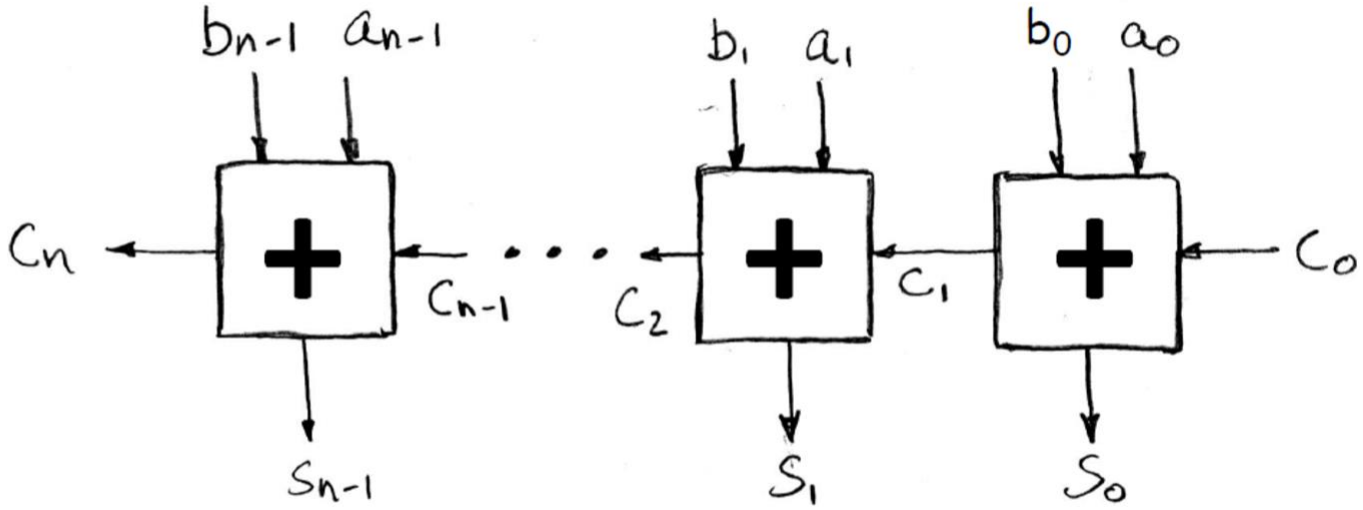




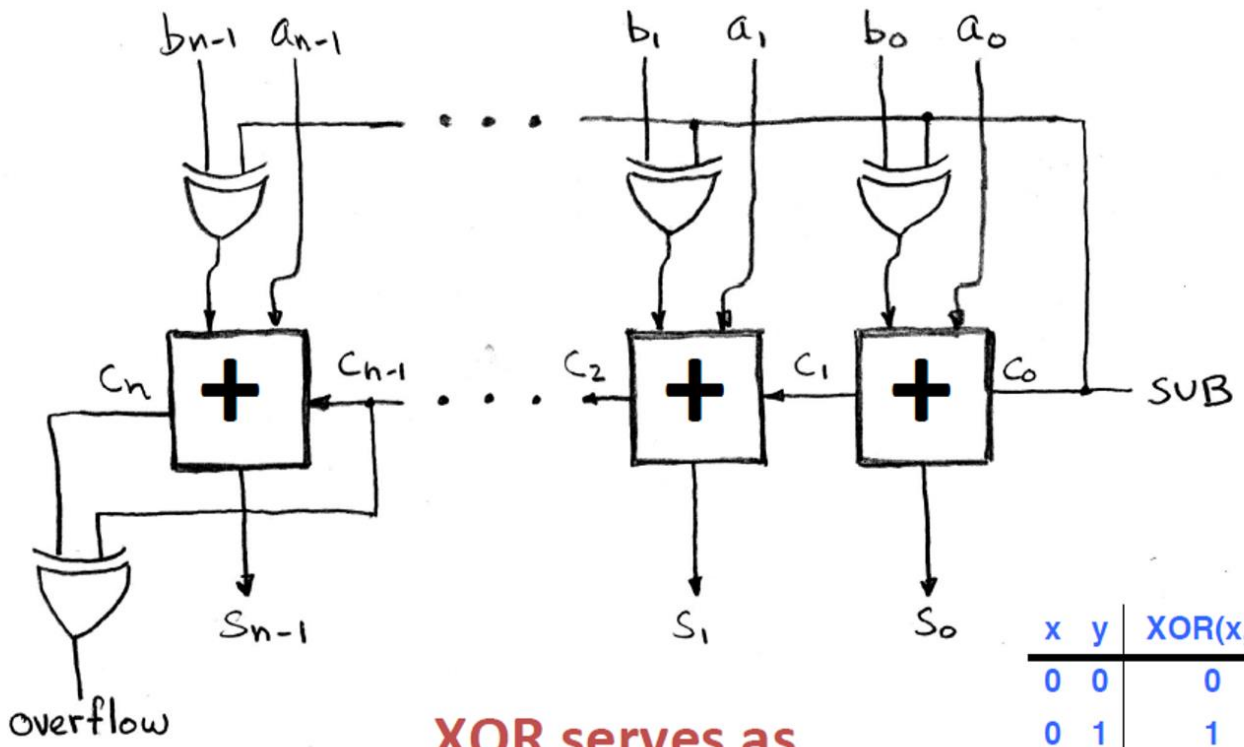
Cascade of 1-bit adders into an adder

- Rather intuitive
- C_n to represent overflow

	a_3	a_2	a_1	a_0
+	b_3	b_2	b_1	b_0
	s_3	s_2	s_1	s_0



Cascade of 1-bit adders into a subtractor



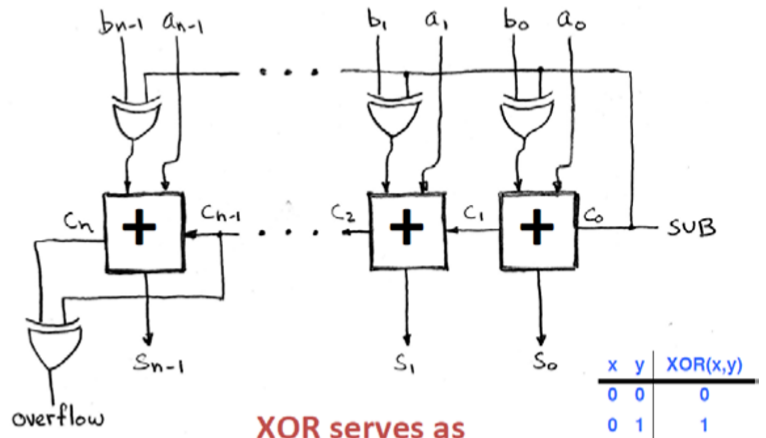
x	y	XOR(x,y)
0	0	0
0	1	1
1	0	1
1	1	0

XOR serves as conditional inverter!

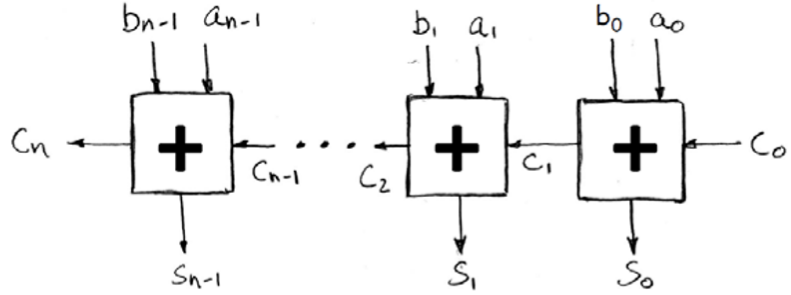
Cascade of 1-bit adders into a subtractor

- A, B are n-bit numbers with signs
- $\tilde{A}$ is the 2's complement, $\bar{A}$ is 1's complement
- Setting $SUB = 1$ to get $\bar{B}$
  - b_i xor $SUB = b_i$ when $SUB = 0$
  - b_i xor $SUB = \bar{b_i}$ when $SUB = 1$
- Subtract to addition: $A - B = A + \tilde{B}$
  - $-B = \tilde{B}$ considering only the lowest (n-1) bits:
    $(-B)_{n-1} = (2^{n-1} + (-B))_{n-1} = (2^{n-1} - 1 - B) + 1$

  Where $(\cdot)_{n-1}$ represents taking the lowest (n-1) bits



Overflow of n-bit adder



- Let A, B be the equivalent inputs to an n-bit adder
- If $A > 0, B > 0$
  - $a_{n-1} = b_{n-1} = 0 \Rightarrow c_n = 0$
  - If overflow, $c_{n-1} = 1$
- If $AB < 0$
  - Overflow won't happen since $|A + B| \leq |A|$

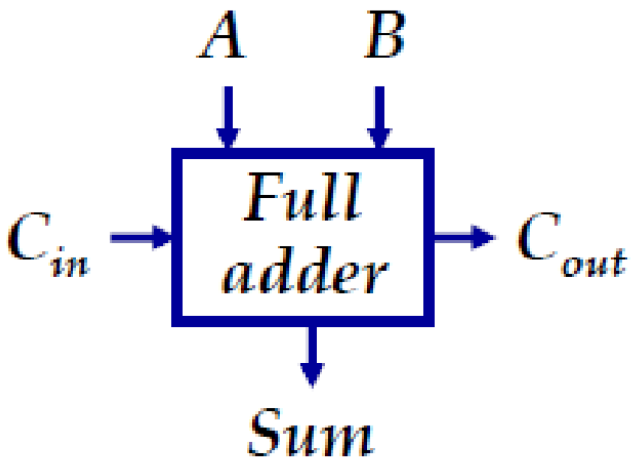
Overflow of n-bit **adder**

- If $A < 0, B < 0$, we actually add their complements $\bar{A}, \bar{B}$
  - $a_{n-1} = b_{n-1} = 1 \Rightarrow c_n = 1$
  - If overflow, $c_{n-1} = 0$ (a bit tricky here)
    - If $(A + B)_{n-1} = 2^{n-1}$, then $A + B = -2^n$, represented by $\mathbf{0b100\ldots0}$, not overflow
    - Overflows when $(A + B)_{n-1} > 2^{n-1}$

    $\Rightarrow 2^{n-1} - \tilde{A}_{n-1} + 2^{n-1} - \tilde{B}_{n-1} > 2^{n-1} \Rightarrow \tilde{A}_{n-1} + \tilde{B}_{n-1} < 2^{n-1} \Rightarrow c_{n-1} = 0$
- To sum up, the addition overflows if and only if c_n xor c_{n-1}=true

- **Sum function**
  - S_0 = 1 when Odd input 1's
  - S_0 = XOR (A,B,C_i)
  - $S_0 = A \times B \times C_i$
- **Carry function**
  - C_0 = 1 when 2 or more input 1's
  - $C_0 = AB + BC_i + AC_i$

A	B	C_i	S_o	C_o
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



Binary Adder

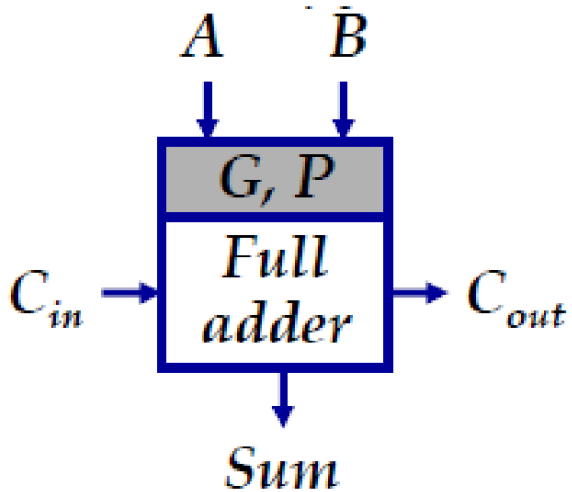
$$S = A \oplus B \oplus C_i$$

$$= A\bar{B}\bar{C}_i + \bar{A}B\bar{C}_i + \bar{A}\bar{B}C_i + ABC_i$$

$$C_o = AB + BC_i + AC_i$$

Full Adder – Another Look

- It's all about carry, so redefine terms for more efficient design
- Carry status
  - Delete (D): $D = \bar{A} \cdot \bar{B}$
  - Propagate (P): $P = A \oplus B$
  - Generate (G): $G = A \cdot B$
- Full Adder
  - Internally generates P, G, (D)



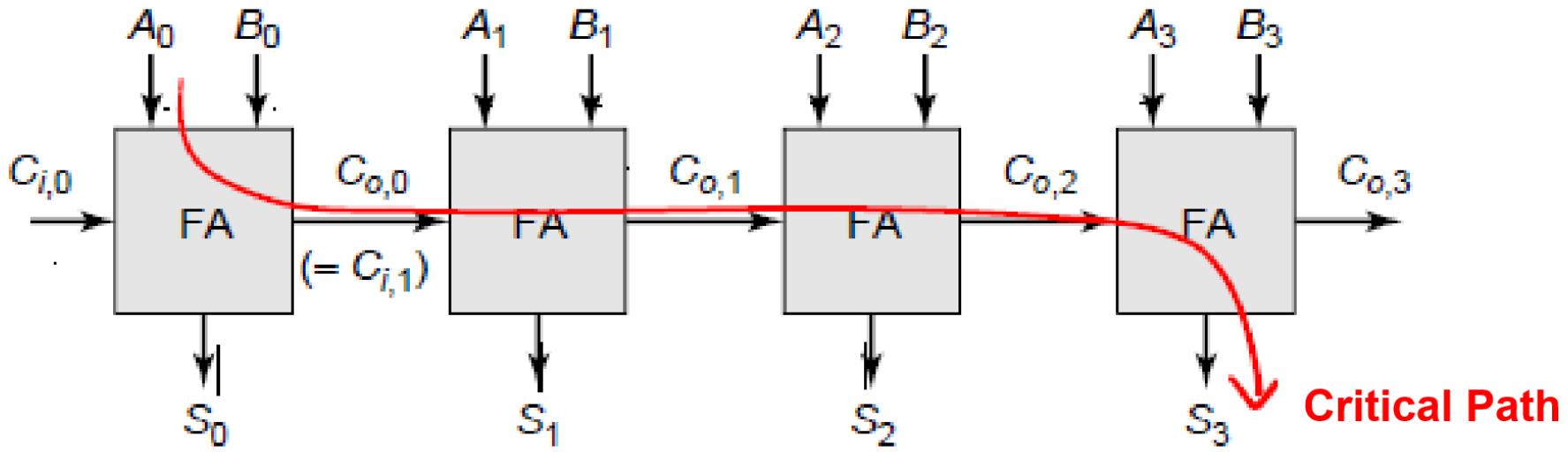
Binary Adder

C_o status	A	B	C_i	C_o
Delete	0	0	0	0
Delete	0	0	1	0
Propagate	0	1	0	0
Propagate	0	1	1	1
Propagate	1	0	0	0
Propagate	1	0	1	1
Generate	1	1	0	1
Generate	1	1	1	1

$$C_o(G,P) = G + P \cdot C_i$$

$$S(G,P) = P \oplus C_i$$

The Ripple-Carry Adder



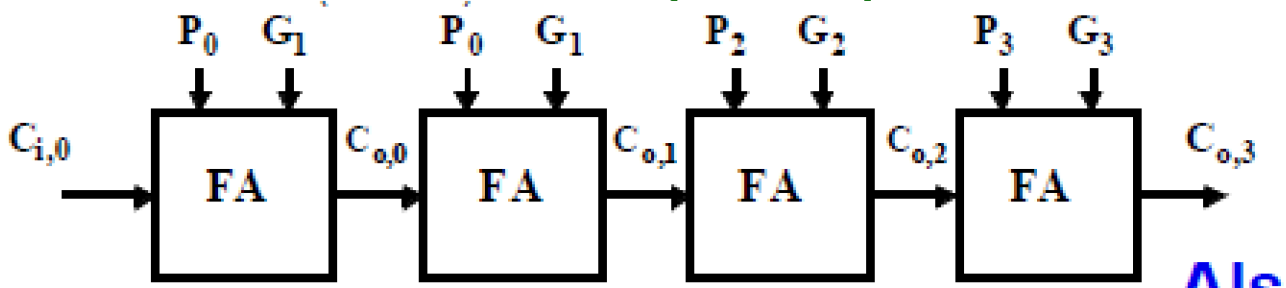
- Worst case delay is proportional to the number of bits

$$t_d = (N-1)t_{carry} + t_{sum} = O(N)$$

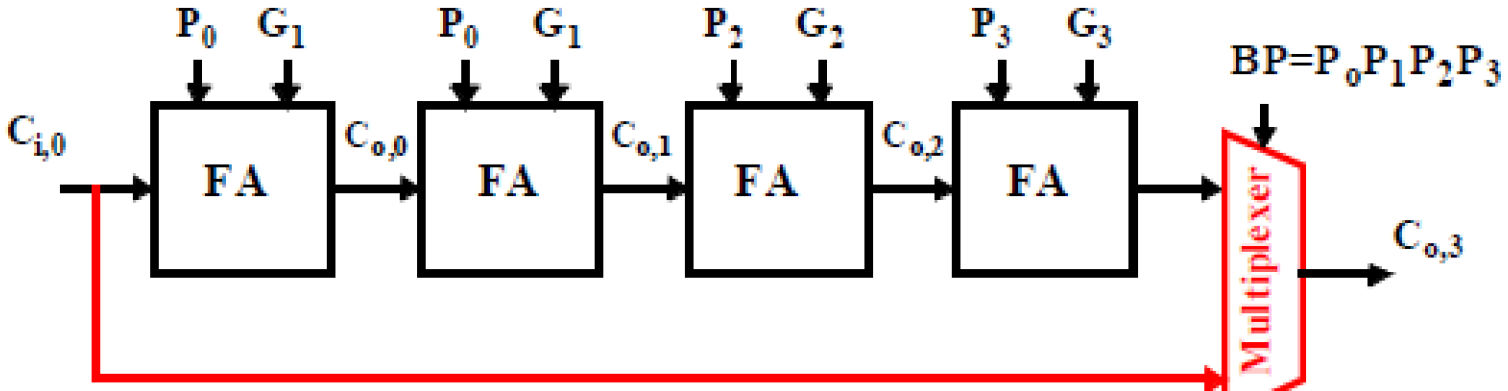
***GOAL*: Make the fastest possible carry path circuit**

Carry-Bypass Adder

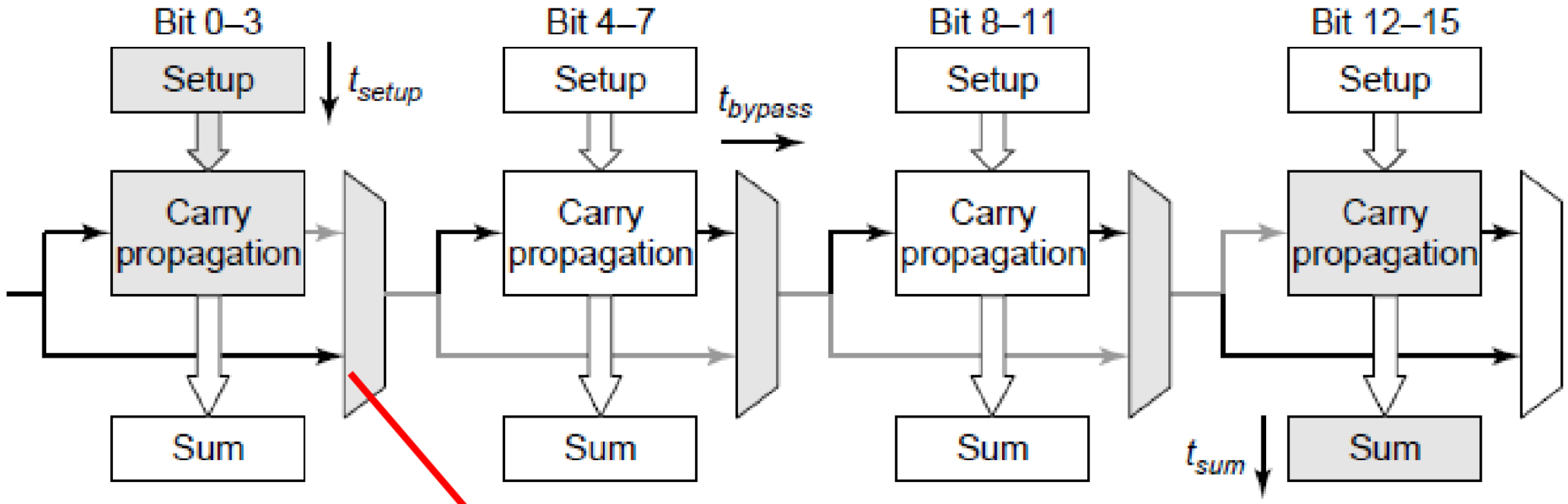
- **Simple adders ripple the carry, faster ones bypass it**
  - Calculate the carry several bits at a time
  - Good for small adders (n < 16)

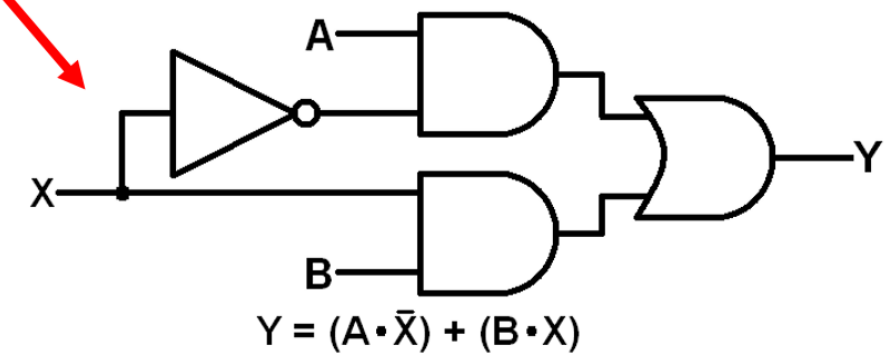


Also called Carry-Skip



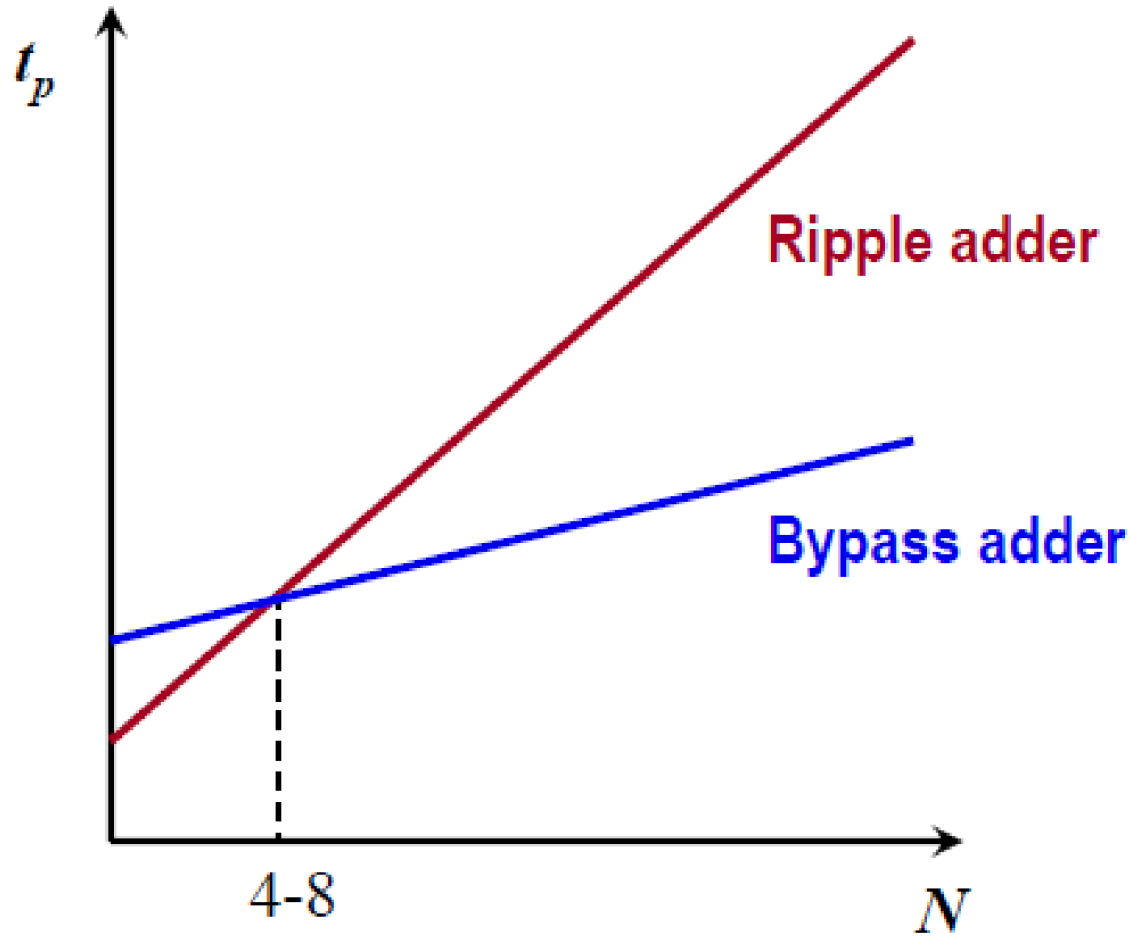
Idea: If (P_0 and P_1 and P_2 and P_3 = 1), then $C_{o,3} = C_o$, else "kill" or "generate"



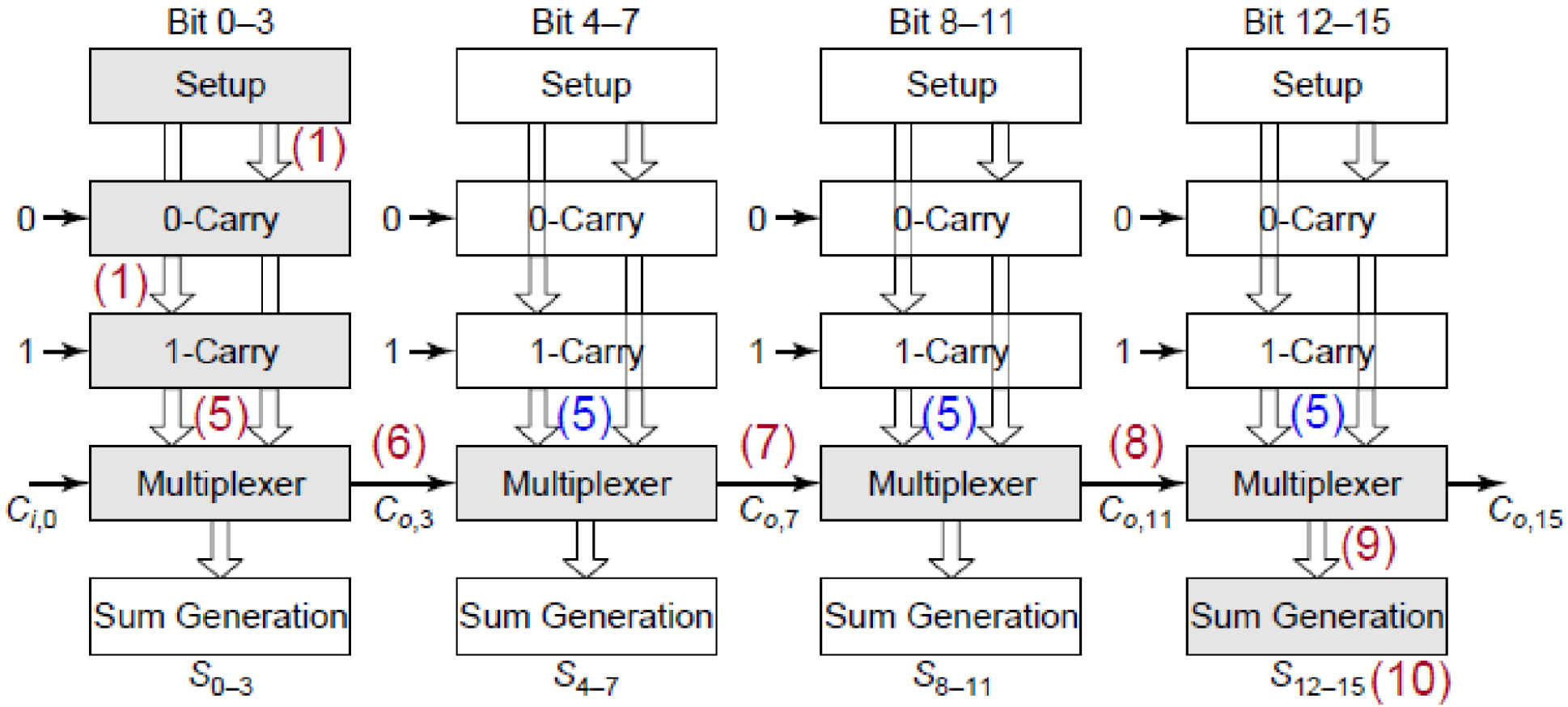


$$t_d = t_{setup} + M \cdot t_{carry} + (N/M - 1) \cdot t_{bypass} + (M - 1) \cdot t_{carry} + t_{sum}$$

Ripple Adder VS Bypass Adder



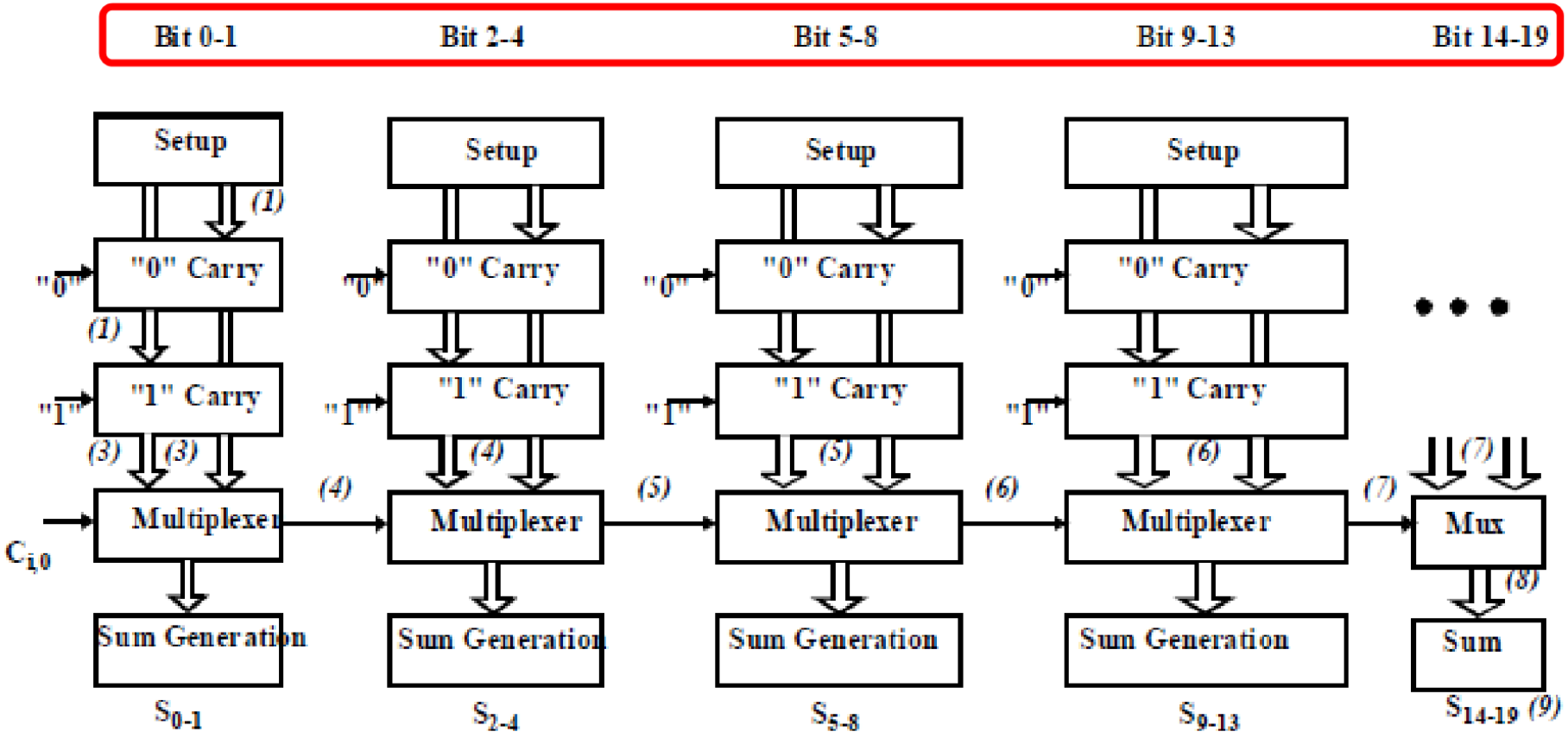
Critical Path in Linear Carry Select Adder



$$t_d = t_{setup} + M \cdot t_{carry} + N/M \cdot t_{MUX} + t_{sum}$$

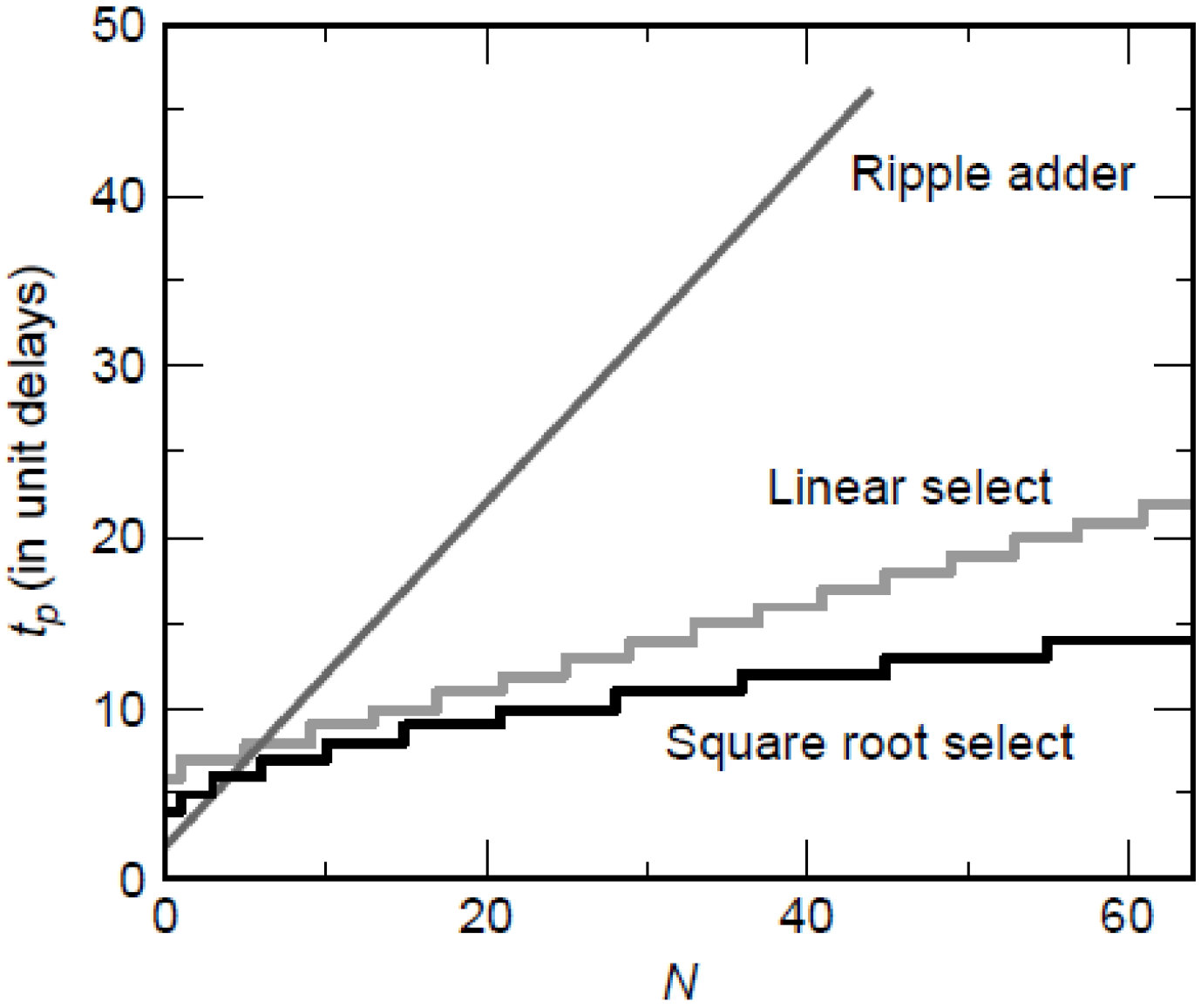
Square Root Carry Select

- Increase group size toward MSBs to fix the slack



$$t_d = t_{setup} + M \cdot t_{carry} + \sqrt{2N} \cdot t_{MUX} + t_{sum}$$

Adder Delay - Comparison



THX >.<