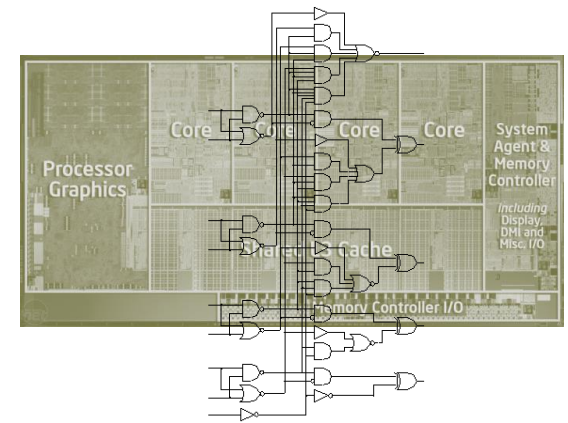




고려대학교
KOREA UNIVERSITY



COSE221, COMP211 Logic Design

Lecture 6. Adders

Prof. Taeweon Suh

Computer Science & Engineering

Korea University

Introduction

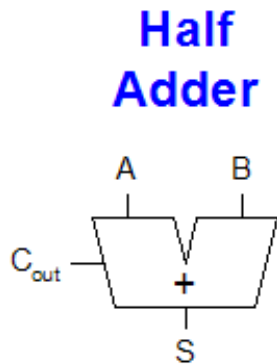
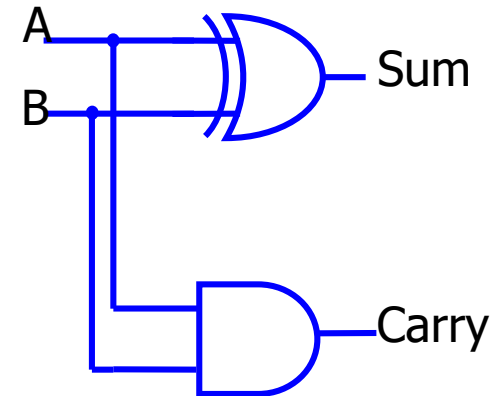
- So far, we have studied the basic skills of designing combinational and sequential logic using schematic and Verilog-HDL
- Now, we are going to study some interesting combinational and sequential blocks, including
 - Arithmetic circuits: Adders and Subtractor
 - Counters
 - Shift Registers
- Also, we are going to discuss how computer represents floating-point numbers
 - `float` and `double` in C language
- The last topics to discuss are Memory and Logic Arrays

Arithmetic Circuits

- Computers are able to perform various arithmetic operations such as addition, subtraction, comparison, shift, multiplication, and division
 - Arithmetic circuits are the central building blocks of computers (CPUs)
- We are going to study hardware implementations of these operations
- Let's start with adder
 - Addition is one of most common operations in computer

1-bit Half Adder

- Let's first consider how to implement an 1-bit adder
- Half adder**
 - 2 inputs: A and B
 - 2 outputs: S (Sum) and C_{out} (Carry)



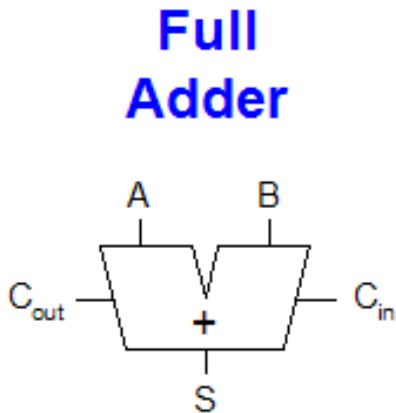
A	B	S(um)	C(array)
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

$$S = \bar{A}B + A\bar{B} = A \oplus B$$

$$C = AB$$

1-bit Full Adder

- Half adder lacks a C_{in} input to accept C_{out} of the previous column
- **Full adder**
 - 3 inputs: A, B, C_{in}
 - 2 outputs: S, C_{out}



Cin	A	B	S(um)	C _{out}
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

1-bit Full Adder

Cin	A	B	S(um)	C _{out}
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

Sum

Cin \ AB	AB			
	00	01	11	10
0	0	1	0	1
1	1	0	1	0

$$\begin{aligned}
 S &= \text{Cin}\bar{A}\bar{B} + \bar{\text{Cin}}\bar{A}B + \text{Cin}A\bar{B} + \bar{\text{Cin}}AB \\
 &= \text{Cin}(\bar{A}\bar{B} + AB) + \bar{\text{Cin}}(\bar{A}B + A\bar{B}) \\
 &= \text{Cin}(\bar{A} \oplus B) + \bar{\text{Cin}}(A \oplus B) \\
 &= \text{Cin} \oplus A \oplus B
 \end{aligned}$$

C_{out}

Cin \ AB	AB			
	00	01	11	10
0	0	0	1	0
1	0	1	1	1

or

Cin \ AB	AB			
	00	01	11	10
0	0	0	1	0
1	0	1	1	1

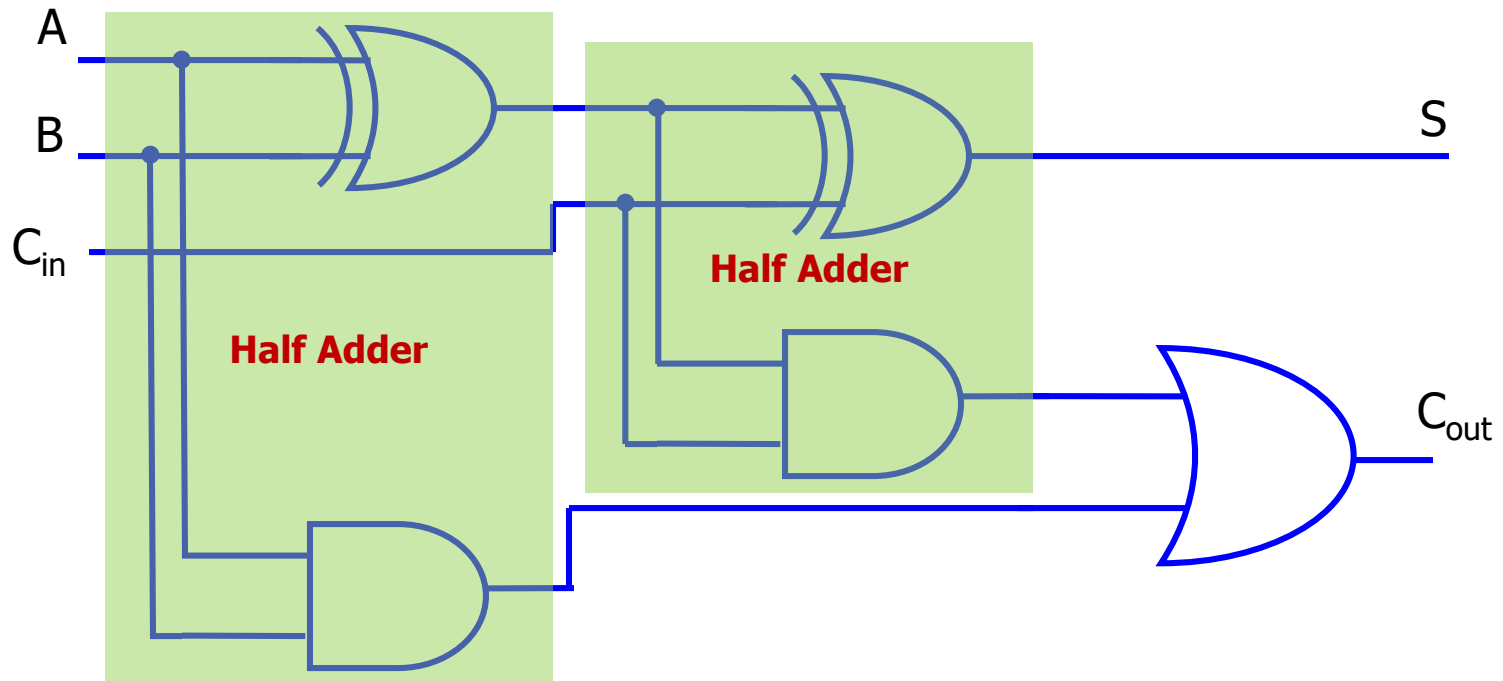
$$C_{out} = \text{Cin}B + \text{Cin}A + AB = AB + \text{Cin}(A + B)$$

$$C_{out} = AB + \text{Cin}(\bar{A}B + A\bar{B}) = AB + \text{Cin}(A \oplus B)$$

1-bit Full Adder Schematic

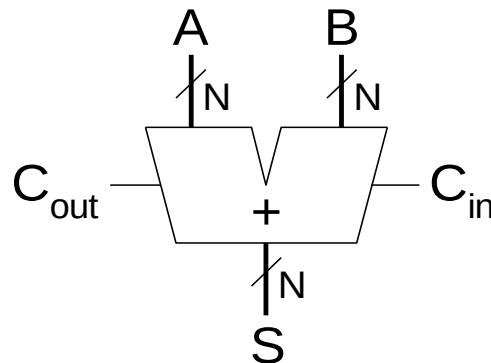
$$S = C_{in} \oplus A \oplus B$$

$$C_{out} = AB + C_{in}(A \oplus B)$$



Multi-bit Adder

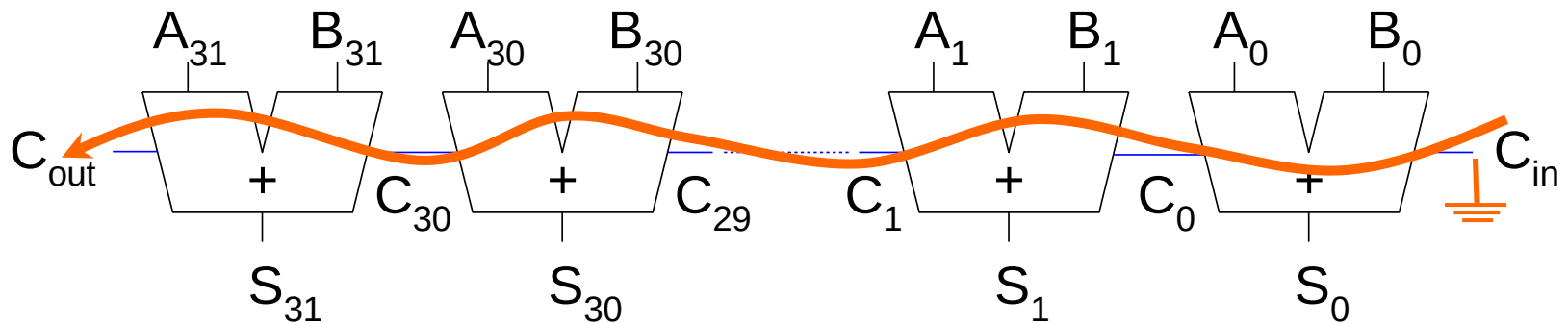
- It seems that an 1-bit adder is doing not much of work
- How to build a multi-bit adder?
 - N-bit adder sums two N-bit inputs (A and B), and C_{in} (carry-in)
- Three common CPA implementations
 - Ripple-carry adders (slow)
 - Carry-lookahead adders (fast)
 - Prefix adders (faster)
- It is commonly called **carry propagate adders (CPAs)** because the carry-out of one bit propagates into the next bit



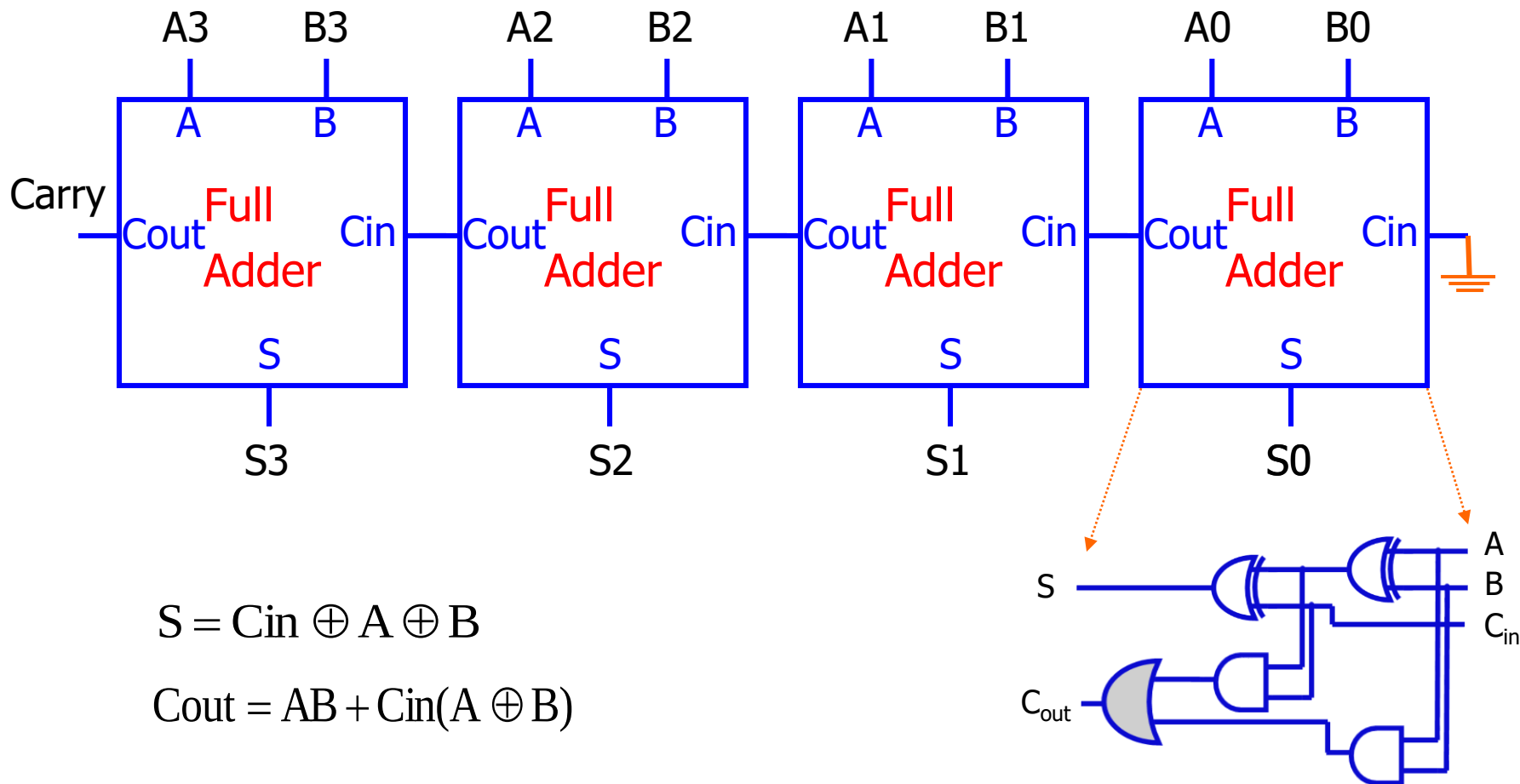
Ripple-Carry Adder

- The simplest way to build an N-bit CPA is to chain 1-bit adders together
 - Carry ripples through entire chain

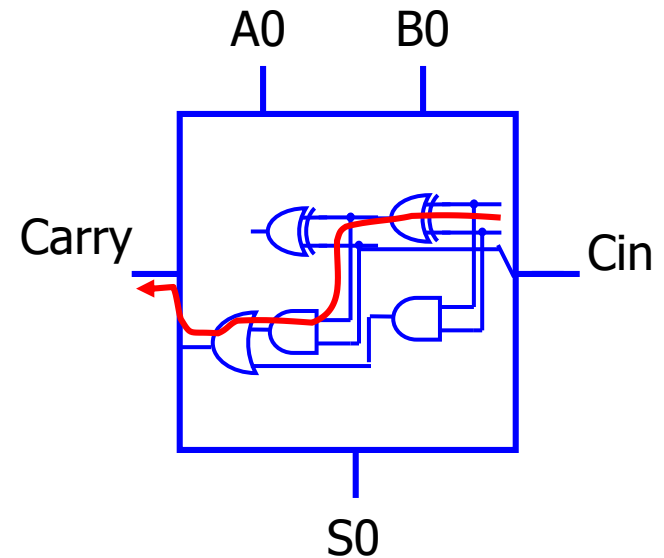
Example: 32-bit Ripple Carry Adder



4-bit Ripple-Carry Adder



Delay of Ripple Carry Adder

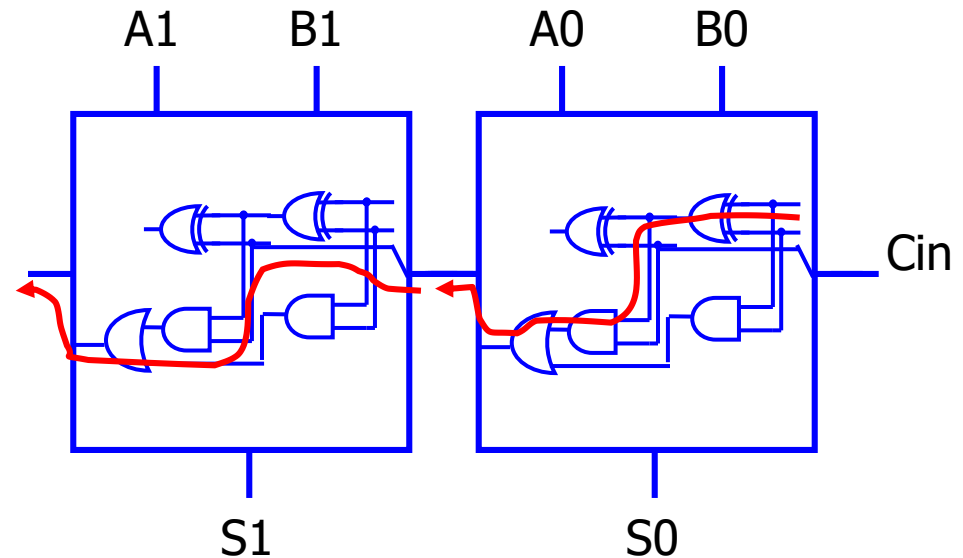


$$S = C_{in} \oplus A \oplus B$$

$$C_{out} = AB + C_{in}(A \oplus B)$$

1st Stage Critical Path
= 3 gate delays
= $D_{XOR} + D_{AND} + D_{OR}$

Delay of Ripple Carry Adder



$$S = C_{in} \oplus A \oplus B$$

$$C_{out} = AB + C_{in}(A \oplus B)$$

2nd Stage Critical Path

= 2-gate delay

= $D_{AND} + D_{OR}$

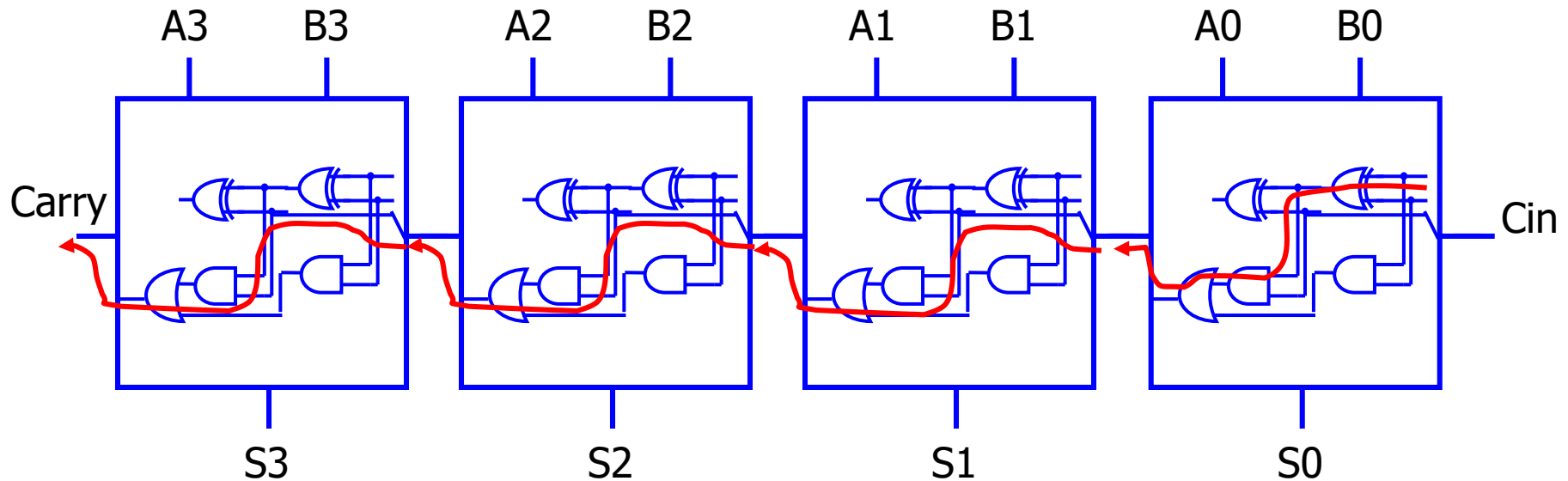
(Assume that inputs
are applied at the
same time)

1st Stage Critical Path

= 3-gate delay

= $D_{XOR} + D_{AND} + D_{OR}$

Delay of Ripple Carry Adder



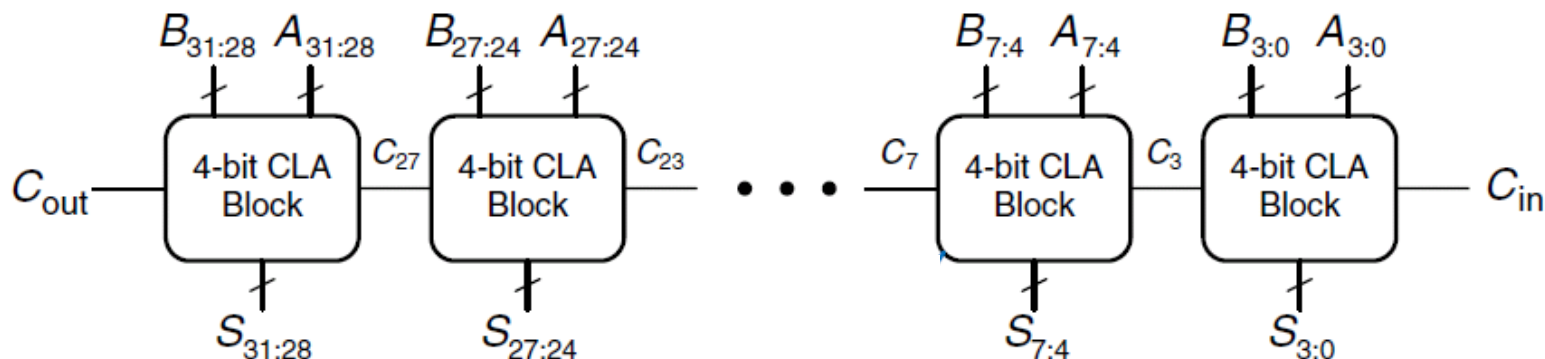
- **Critical path delay** of a 4-bit ripple carry adder
 - $D_{XOR} + 4 (D_{AND} + D_{OR})$: 9-gate delay
- **Critical path delay** of an N-bit ripple carry adder
 - $2(N-1)+3 = (2N+1)$ - gate delay

Ripple-Carry Adder Delay

- Ripple-carry adder has disadvantage of being **slow when N is large**
 - The delay of an N-bit ripple-carry adder is roughly $t_{\text{ripple}} = N \cdot t_{\text{FA}}$ (t_{FA} is the delay of a full adder)
- A faster adder needs to address the serial propagation of the carry bit

Carry-Lookahead Adder

- The fundamental reason that large ripple-carry adders are slow is that the carry signals must propagate through every bit in the adder
- A carry-lookahead adder (CLA) is another type of CPA that solves this problem.
 - It divides the adder into blocks and provides circuitry to quickly determine the carry-out of a block as soon as the carry-in is known



Carry-Lookahead Adder

$$C_{out} = C_{in}B + C_{in}A + AB = AB + C_{in}(A + B)$$

- Compute the carry-out (C_{out}) for an N-bit block
 - Compute generate (G) and propagate (P) signals for columns and then an N-bit block
 - A column (bit i) can produce a carry-out by **either generating** a carry-out **or propagating** a carry-in to the carry-out
- Generate (G_i) and Propagate (P_i) signals for each column
 - A column will **generate** a carry-out if **both** A_i **and** B_i are 1
$$G_i = A_i B_i$$
 - A column will **propagate** a carry-in to the carry-out if **either** A_i **or** B_i is 1
$$P_i = A_i + B_i$$
- Express the carry-out of a column (C_i) in terms of P_i and G_i
$$C_i = A_i B_i + (A_i + B_i) C_{i-1} = G_i + P_i C_{i-1}$$

Carry Generate & Propagate

$$g_i = A_i B_i$$

$$p_i = A_i + B_i$$

$$C_i = A_i B_i + (A_i + B_i) C_{i-1}$$

$$C_i = g_i + p_i C_{i-1}$$

$$C_0 = g_0 + p_0 C_{-1}$$

$$C_1 = g_1 + p_1 C_0 = g_1 + p_1 g_0 + p_1 p_0 C_{-1}$$

$$C_2 = g_2 + p_2 C_1 = g_2 + p_2 g_1 + p_2 p_1 g_0 + p_2 p_1 p_0 C_{-1}$$

$$C_3 = g_3 + p_3 C_2 = g_3 + p_3 g_2 + p_3 p_2 g_1 + p_3 p_2 p_1 g_0 + p_3 p_2 p_1 p_0 C_{-1}$$

What do these equations mean? Let's think about these equations for a moment

Carry Generate & Propagate

- A 4-bit block will **generate** a carry-out if column 3 generates (g_3) a carry or if column 3 propagates (p_3) a carry that was generated or propagated in a previous column

$$\begin{aligned} G_{3:0} &= g_3 + p_3 (g_2 + p_2 (g_1 + p_1 g_0)) \\ &= g_3 + p_3 g_2 + p_3 p_2 g_1 + p_3 p_2 p_1 g_0 \end{aligned}$$

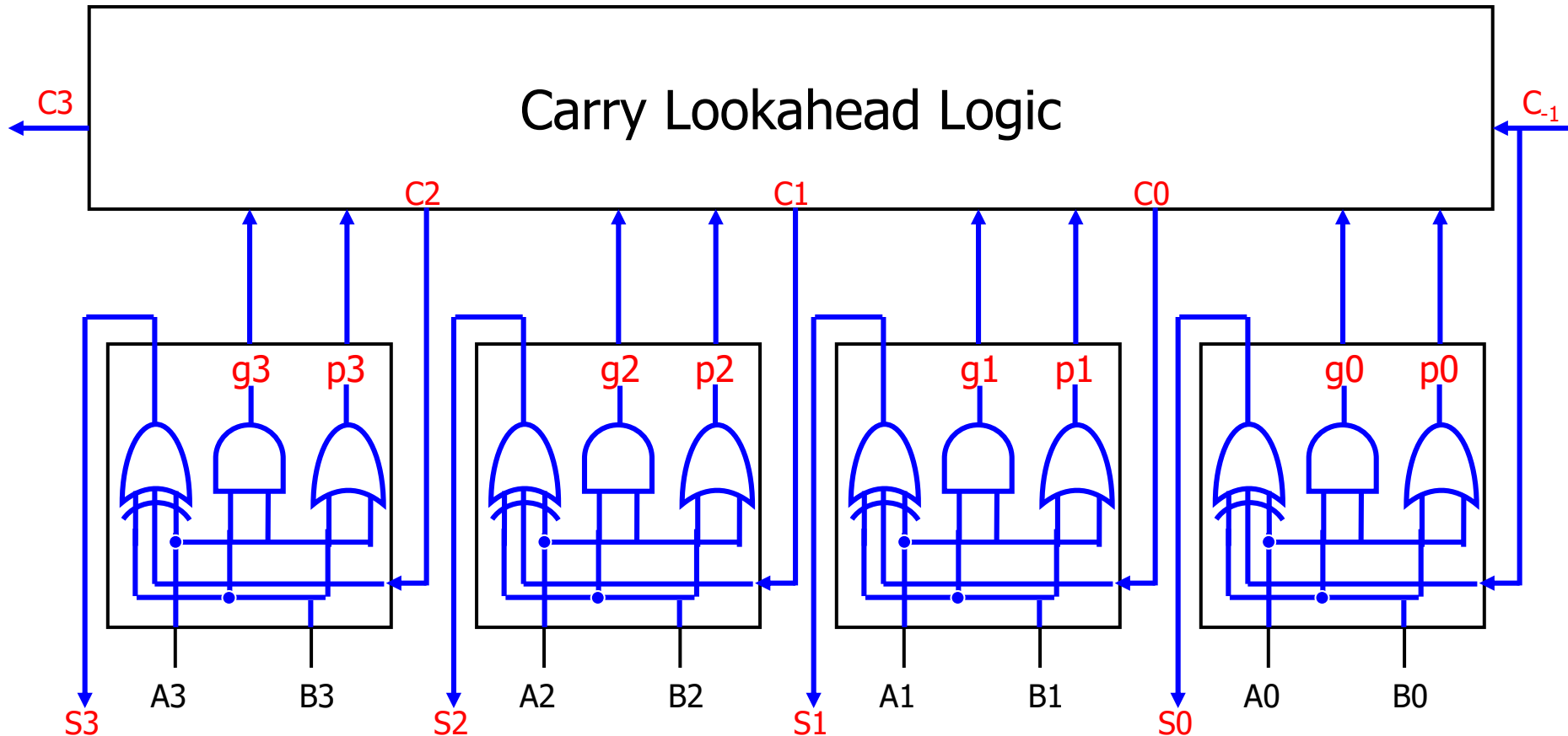
- A 4-bit block will **propagate** a carry-in to the carry-out if all of the columns propagate the carry

$$P_{3:0} = p_3 p_2 p_1 p_0$$

- We compute the carry-out of the 4-bit block (C_i) as

$$C_i = G_{i:j} + P_{i:j} C_{j-1}$$

4-bit CLA

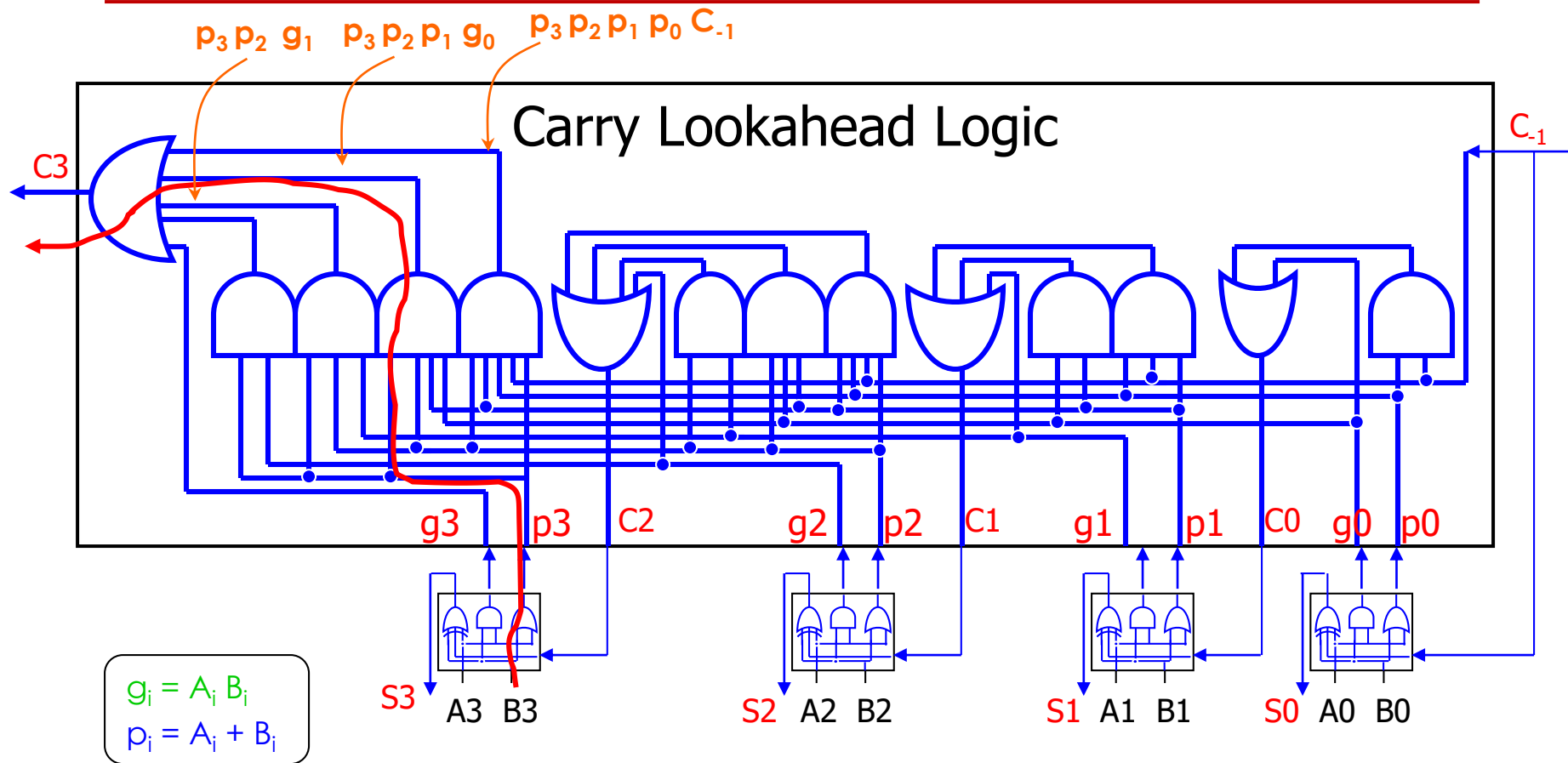


$$S_i = C_i \oplus A_i \oplus B_i$$

$$g_i = A_i B_i \quad (\text{generate})$$

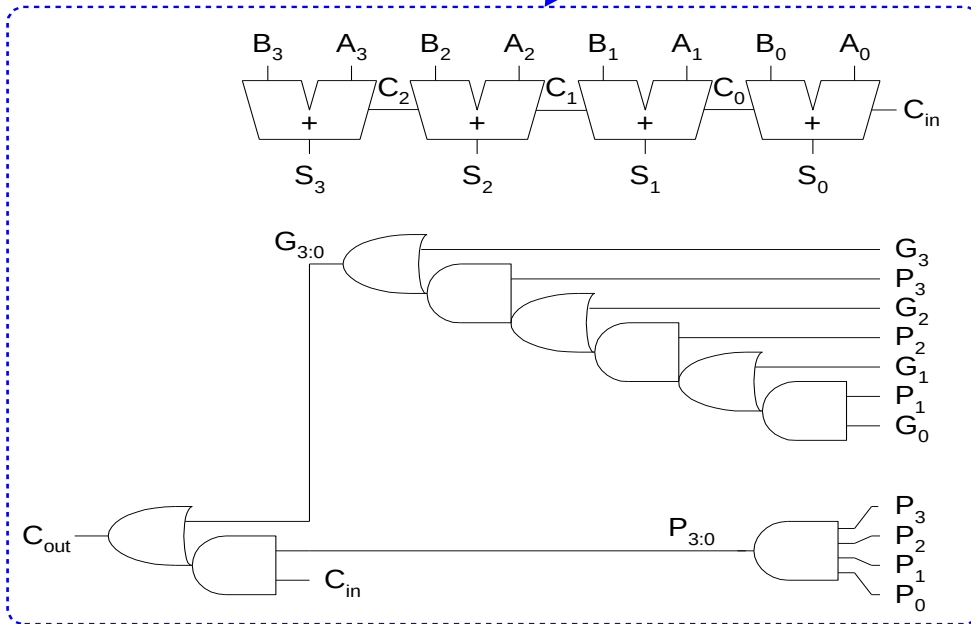
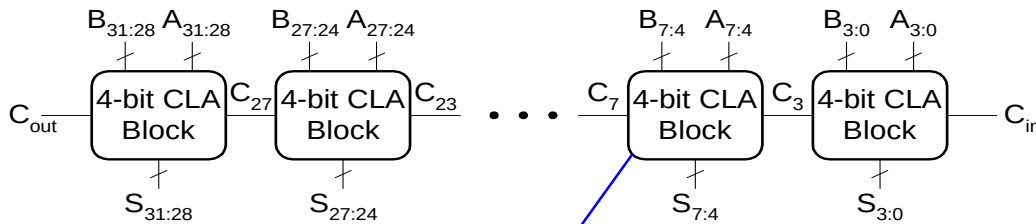
$$p_i = A_i + B_i \quad (\text{propagate})$$

A CLA Implementation



Only 3 gate delay for each Carry $C_i = D_{AND} + 2 \cdot D_{OR}$

32-bit CLA with 4-bit blocks



An implementation in our book:
each block contains a 4-bit **RCA**
and **carry-lookahead logic**

$$C_0 = g_0 + p_0 C_{-1}$$

$$C_1 = g_1 + p_1 C_0 = g_1 + p_1 g_0 + p_1 p_0 C_{-1}$$

$$C_2 = g_2 + p_2 C_1 = g_2 + p_2 g_1 + p_2 p_1 g_0 + p_2 p_1 p_0 C_{-1}$$

$$C_3 = g_3 + p_3 C_2 = g_3 + p_3 g_2 + p_3 p_2 g_1 + p_3 p_2 p_1 g_0 + p_3 p_2 p_1 p_0 C_{-1}$$

$$C_i = G_{i:j} + P_{i:j} C_{j-1}$$

$$G_{3:0} = g_3 + p_3 (g_2 + p_2 (g_1 + p_1 g_0))$$

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

$$P_{3:0} = p_3 p_2 p_1 p_0$$

It shows a path to C3 only

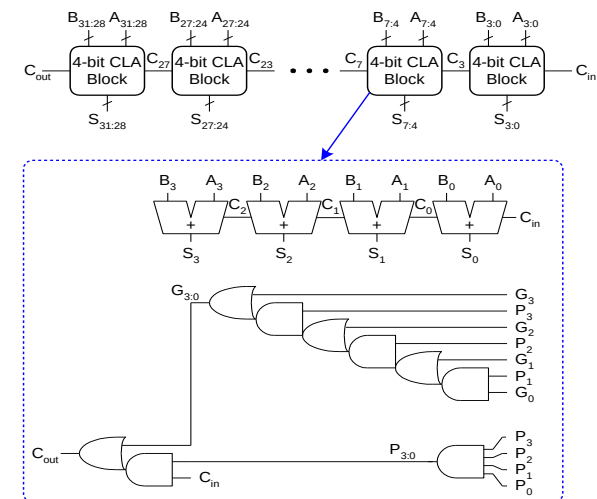
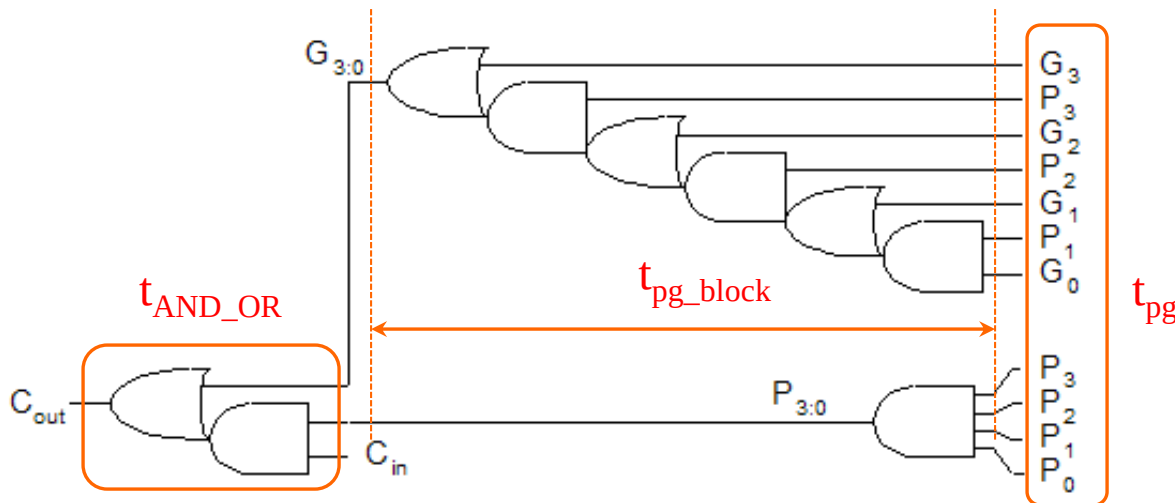
CLA Delay

- The delay of an N-bit CLA with k-bit blocks is roughly:

$$t_{CLA} = t_{pg} + t_{pg_block} + (N/k - 1)t_{AND_OR} + kt_{FA}$$

where

- t_{pg} is the delay of the column generate and propagate gates
- t_{pg_block} is the delay of the block generate and propagate gates
- t_{AND_OR} is the delay from C_{in} to C_{out} of the final AND/OR gate

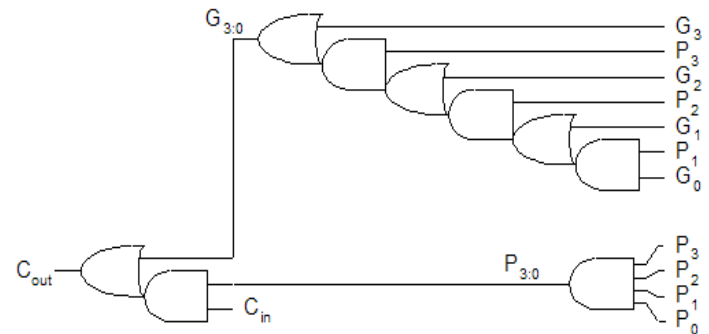


Adder Delay Comparisons

- Compare the delay of 32-bit ripple-carry adder and CLA
 - The CLA has 4-bit blocks
 - Assume that each two-input gate delay is 100 ps
 - Assume that a full adder delay is 300 ps

$$\begin{aligned}t_{\text{ripple}} &= Nt_{\text{FA}} \\&= 32(300 \text{ ps}) \\&= 9.6 \text{ ns}\end{aligned}$$

$$\begin{aligned}t_{\text{CLA}} &= t_{\text{pg}} + t_{\text{pg_block}} + (N/k - 1)t_{\text{AND_OR}} + kt_{\text{FA}} \\&= [100 + 600 + (7)200 + 4(300)] \text{ ps} \\&= 3.3 \text{ ns}\end{aligned}$$



Verilog-HDL Representation

```
module adder #(parameter N = 8)
    (input  [N-1:0] a, b,
     input                cin,
     output [N-1:0] s,
     output                cout);

    assign {cout, s} = a + b + cin;

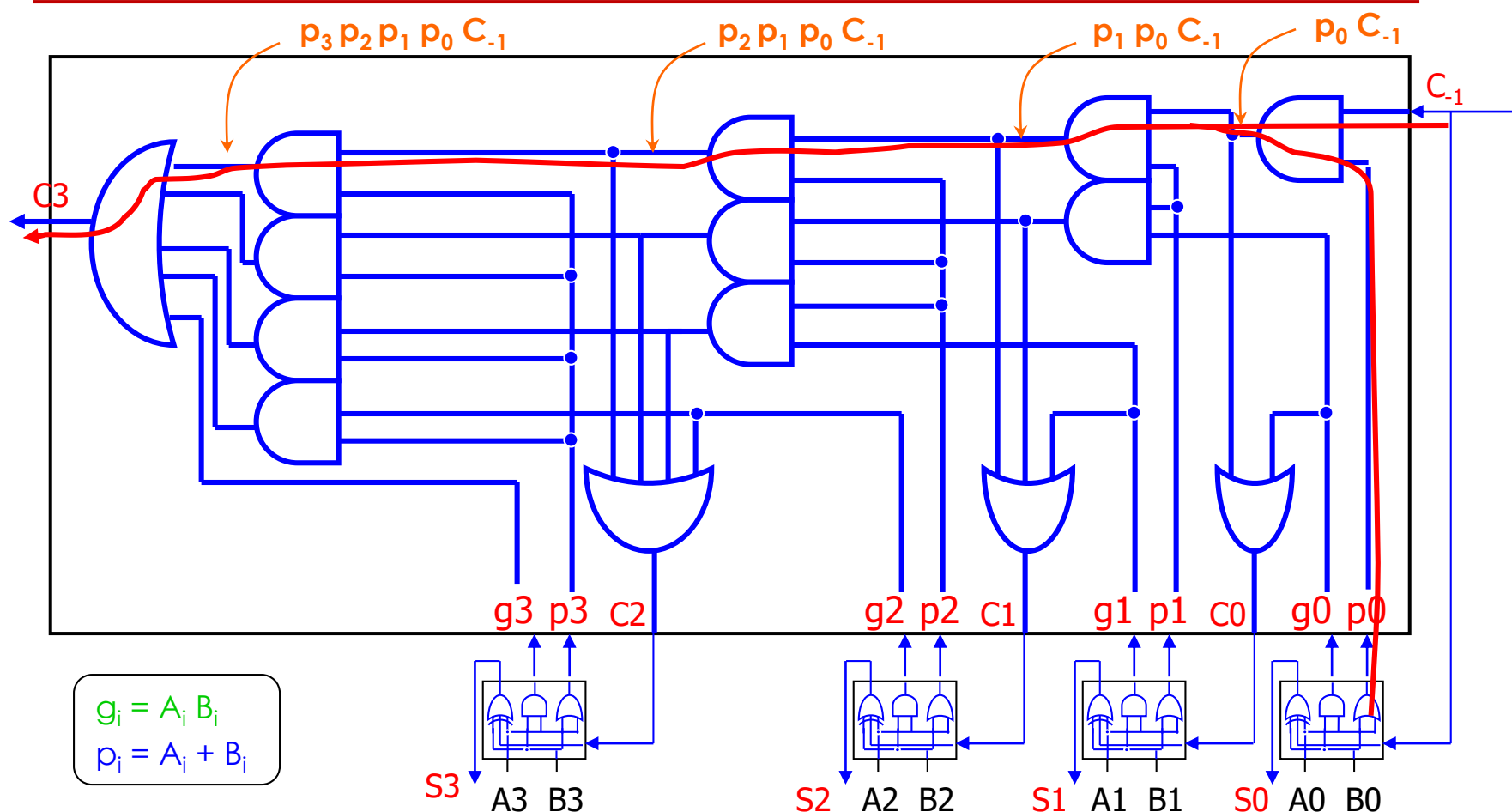
endmodule
```

Then, When to Use What?

- We have discussed 3 kinds of CPA
 - Ripple-carry adder
 - Carry-lookahead adder
 - Prefix adder (see backup slides)
- Faster adders require more hardware and therefore they are more expensive and power-hungry
- So, depending on your speed requirement, you can choose the right one
- If you use HDL to describe an adder, the CAD tools will generate appropriate logic considering your speed requirement

Backup Slides

An Implementation of CLA



$$\begin{aligned}
 C_0 &= g_0 + p_0 C_{-1} \\
 C_1 &= g_1 + p_1 C_0 = g_1 + p_1 g_0 + p_1 p_0 C_{-1} \\
 C_2 &= g_2 + p_2 C_1 = g_2 + p_2 g_1 + p_2 p_1 g_0 + p_2 p_1 p_0 C_{-1} \\
 C_3 &= g_3 + p_3 C_2 = g_3 + p_3 g_2 + p_3 p_2 g_1 + p_3 p_2 p_1 g_0 + p_3 p_2 p_1 p_0 C_{-1}
 \end{aligned}$$

- Carry delay is $4 \cdot D_{\text{AND}} + 2 \cdot D_{\text{OR}}$ for C_3
- Reuse some gate output results in little improvement

Prefix Adder

- Computes generate and propagate signals for all of the columns (!) to perform addition even faster
- Computes G and P for 2-bit blocks, then 4-bit blocks, then 8-bit blocks, etc. until the generate and propagate signals are known for each column
 - Then, the prefix adder has $\log_2 N$ stages
- The strategy is to compute the carry in (C_{i-1}) for each of the columns as fast as possible and then to compute the sum:

$$S_i = (A_i \oplus B_i) \oplus C_{i-1}$$

Prefix Adder

- A carry is generated by being either generated in a column or propagated from a previous column

- Define column -1 to hold C_{in} , so

$$G_{-1} = C_{in}, \quad P_{-1} = 0$$

Then,

$$C_{i-1} = G_{i-1:-1}$$

because there will be a carry out of column $i-1$ if the block spanning columns $i-1$ through -1 generates a carry

- Thus, we can rewrite the sum equation as:

$$S_i = (A_i \oplus B_i) \oplus G_{i-1:-1}$$

Prefix Adder

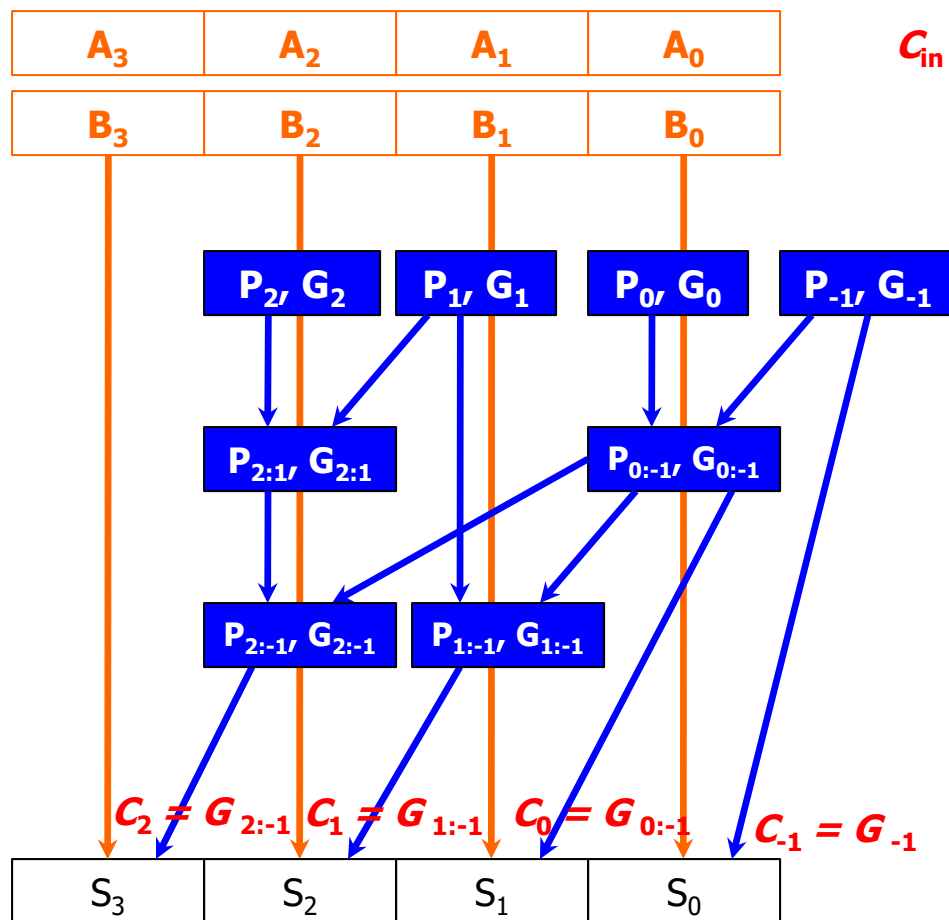
- The generate and propagate signals for a block spanning bits $i:j$ are

$$G_{i:j} = G_{i:k} + P_{i:k} G_{k-1:j}$$

$$P_{i:j} = P_{i:k} P_{k-1:j}$$

- These signals are called the *prefixes* because they must be *precomputed* before the final sum computation can complete
- In words, these prefixes describe that
 - A block will **generate** a carry if the upper part ($i:k$) generates a carry or the upper part propagates a carry generated in the lower part ($k-1:j$)
 - A block will **propagate** a carry if both the upper and lower parts propagate the carry.

4-bit Prefix Adder



Remember that $P_{2:-1}$ is always "0" since $P_{-1} = 0$, but intermediate propagate signals ($P_{1:-1}, P_{0:-1}, P_{2:1}$) are used for calculating subsequent generate signals

$$G_{-1} = C_{in}, \quad P_{-1} = 0$$

$$P_i = A_i B_i, \quad G_i = A_i + B_i$$

$$P_{2:1} = P_2 P_1, \quad G_{2:1} = G_2 + P_2 G_1$$

$$P_{0:-1} = P_0 P_{-1}, \quad G_{0:-1} = G_0 + P_0 G_{-1}$$

$$P_{2:-1} = P_{2:1} P_{0:-1},$$

$$G_{2:-1} = G_{2:1} + P_{2:1} G_{0:-1}$$

$$P_{1:-1} = P_1 P_{0:-1},$$

$$G_{1:-1} = G_1 + P_1 G_{0:-1}$$

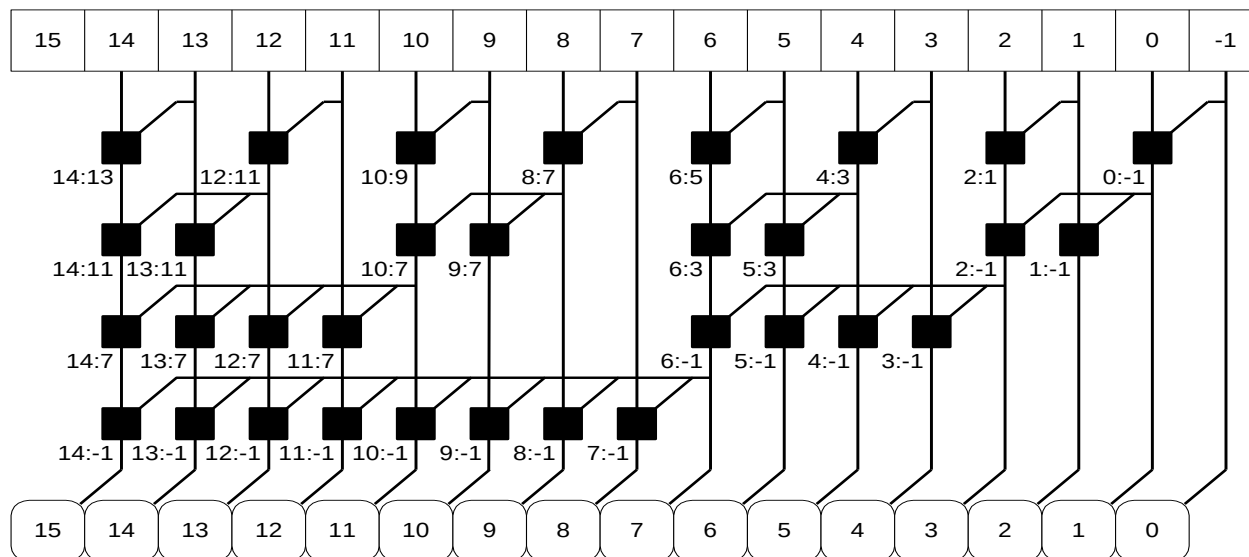
$$S_3 = A_3 \oplus B_3 \oplus G_{2:-1}$$

$$S_2 = A_2 \oplus B_2 \oplus G_{1:-1}$$

$$S_1 = A_1 \oplus B_1 \oplus G_{0:-1}$$

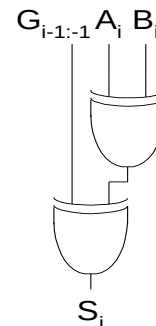
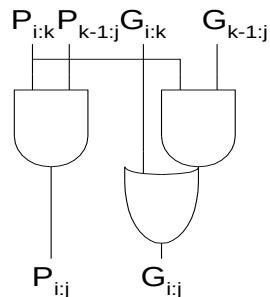
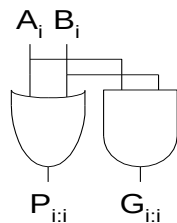
$$S_0 = A_0 \oplus B_0 \oplus G_{-1}$$

16-bit Prefix Adder



$$G_{-1} = C_{in}, \quad P_{-1} = 0$$

Legend



Prefix Adder Delay

- The delay of an N -bit prefix adder is:

$$t_{PA} = t_{pg} + \log_2 N (t_{pg_prefix}) + t_{XOR}$$

where

- t_{pg} is the delay of the column generate and propagate gates
- t_{pg_prefix} is the delay of the black prefix cell