EECE 2322: Fundamentals of Digital Design and Computer Organization

Lecture 4_3: ALU

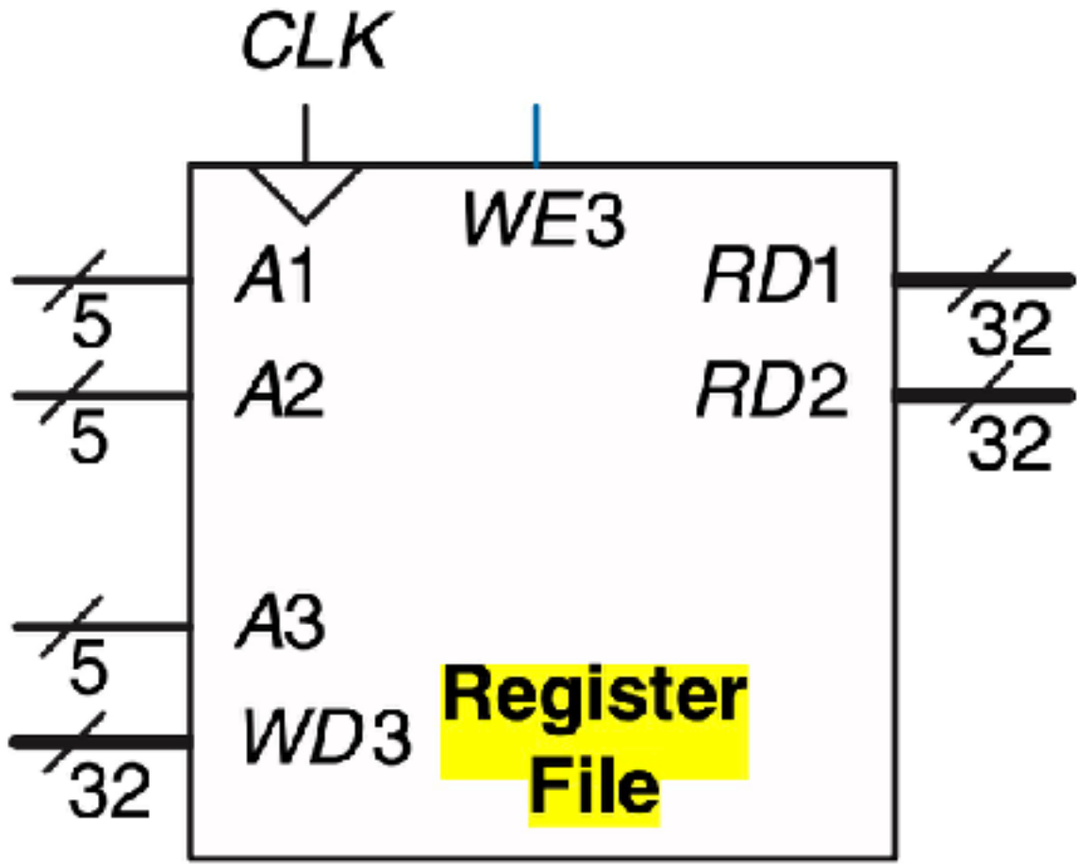
Xiaolin Xu
Department of ECE
Northeastern University

Mid-term Exam1

- When
  - Thursday, Feb. 17, 2022, 1:35—2:40PM
- Where
  - Online, as a regular assignment in Canvas
  - So, no lecture that day!
- What
  - Slides, Week1—Week4
  - HWs

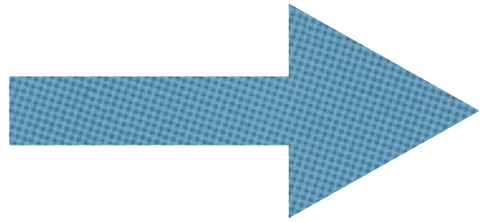
Register File

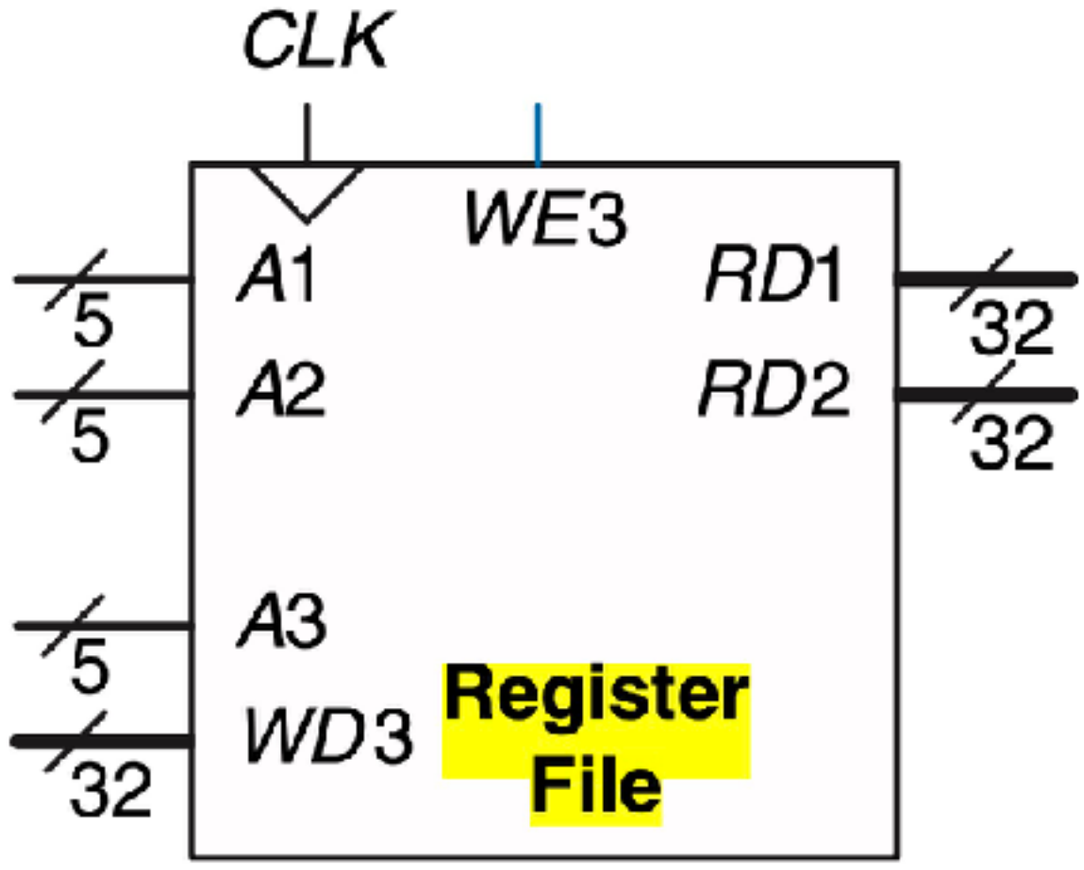
- Register file or Regfile: a number of registers are used to store variables
  - Could be a number of memory cells, e.g., SRAM array
- ***32 × 32 register file with two read ports and one write port***



Register File

- ***32 × 32 register file with two read ports and one write port***

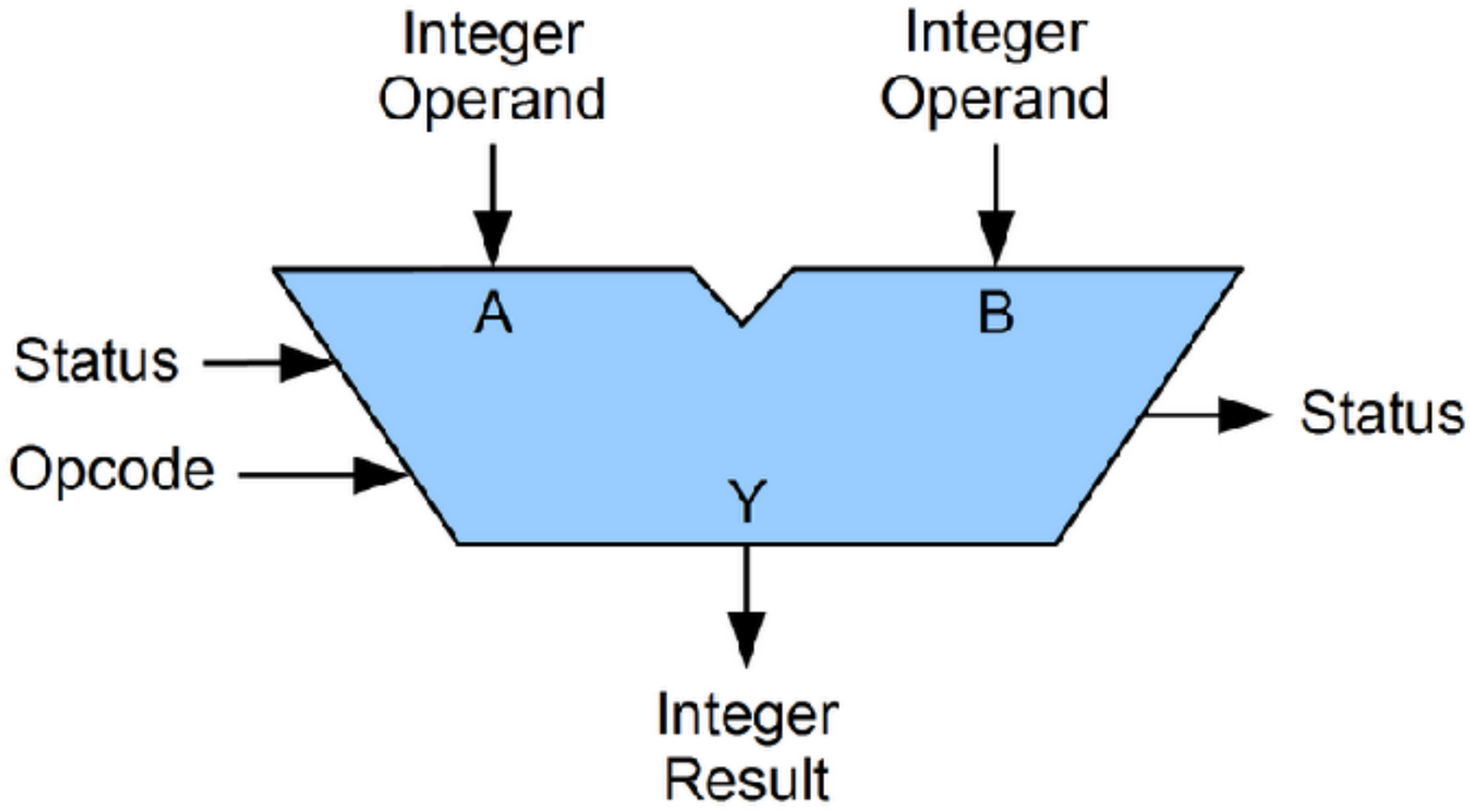




- Two read ports (A1/RD1 and A2/RD2)
- One write port (A3/ WD3).
- 5-bit addresses(A1, A2, and A3) can each access all 32 registers.
- ***Two registers can be read and one register written simultaneously.***

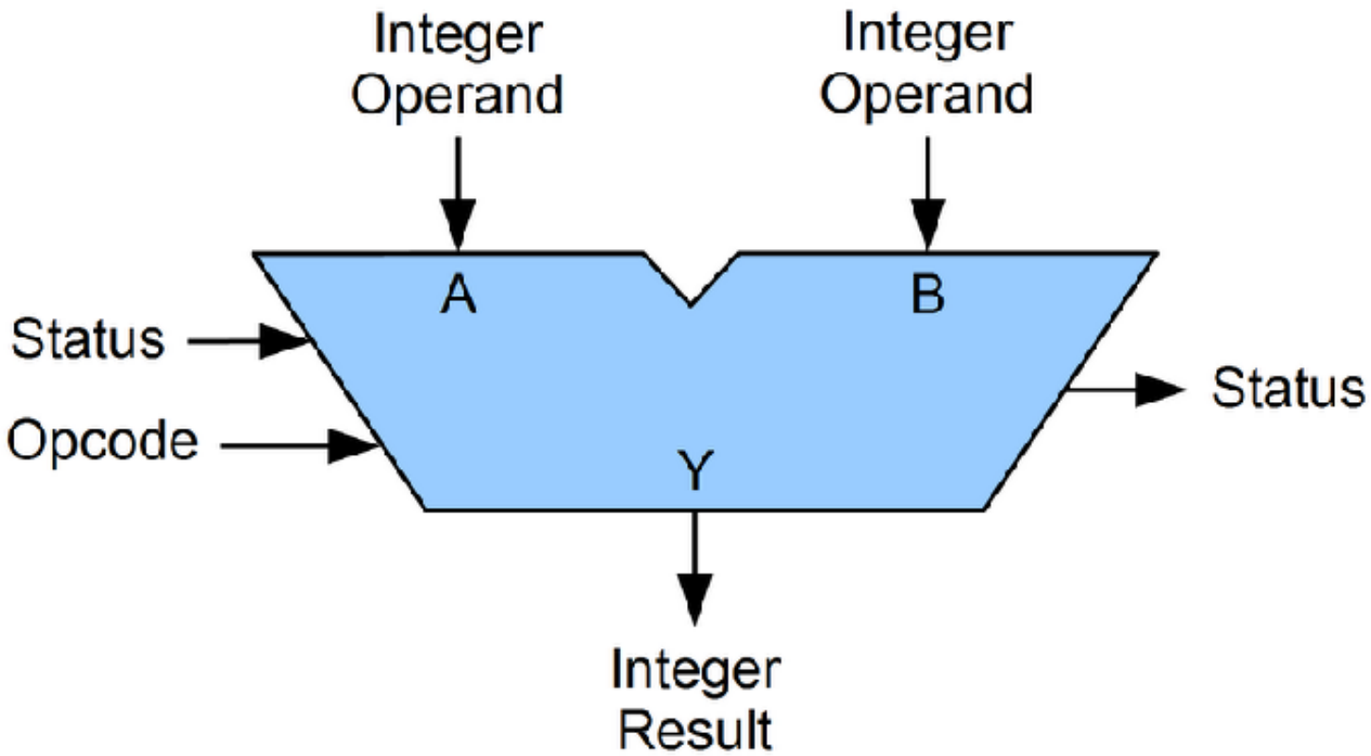
ALU: Arithmetic Logic Unit

- Digital circuit that performs arithmetic and bitwise operations



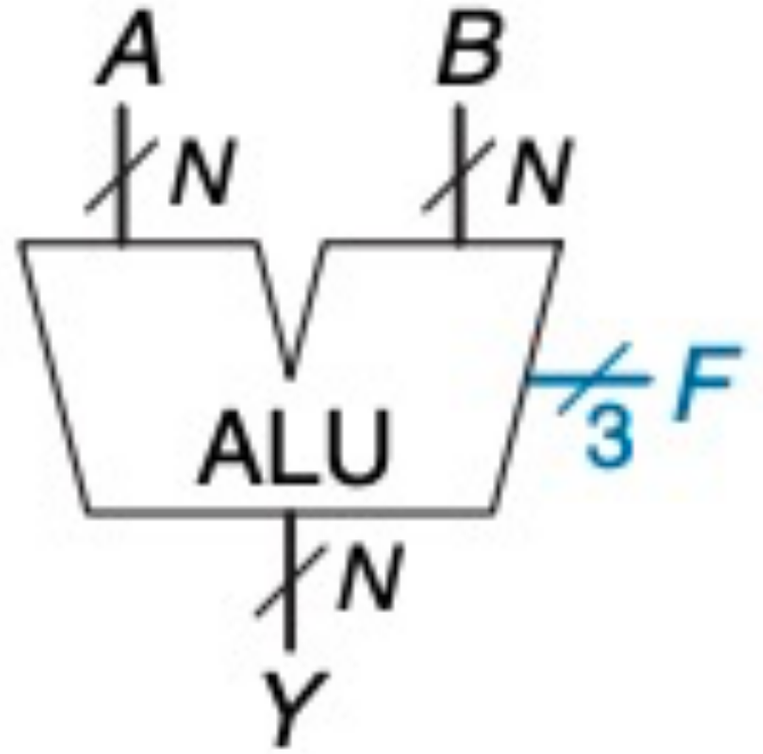
ALU: Arithmetic Logic Unit

- Digital circuit that performs arithmetic and bitwise operations
  - Data, Opcode, Status
- Arithmetic
  - Add, Add with carry, Subtract, Subtract with borrow …
- Bite-wise logic
  - AND, OR, XOR …



ALU Schematic

- Abstraction of ALU, three components
  - Two inputs (A, B)
  - One output (Y)
  - Control signal F (3-bit)

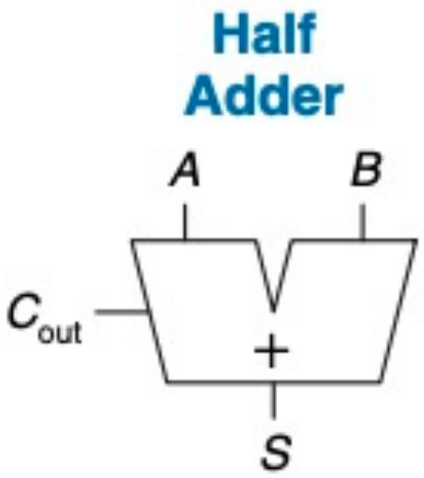


$F_{2:0}$	Function
000	A AND B
001	A OR B
010	A + B
011	not used
100	A AND $\overline{B}$
101	A OR $\overline{B}$
110	A – B
111	SLT

SLT: Set less than (later)

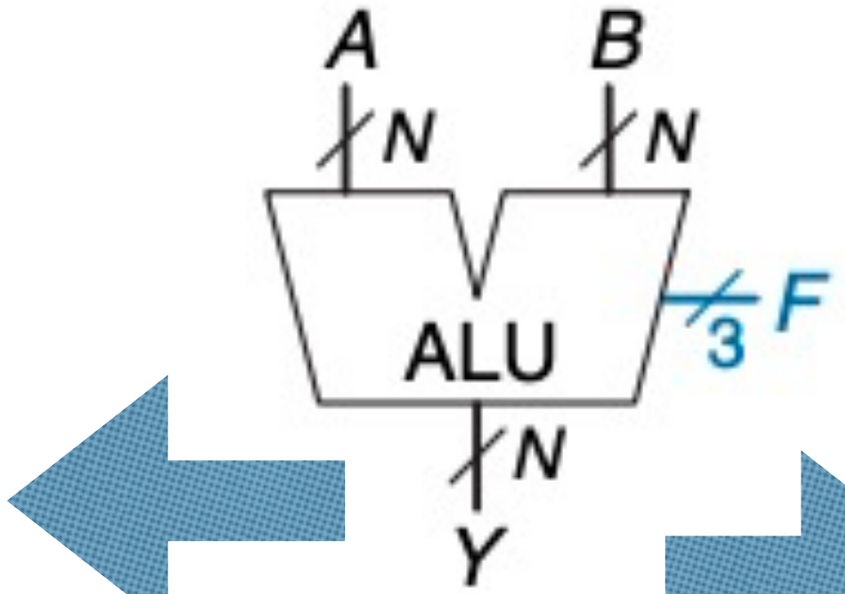
ALU Operations: Addition

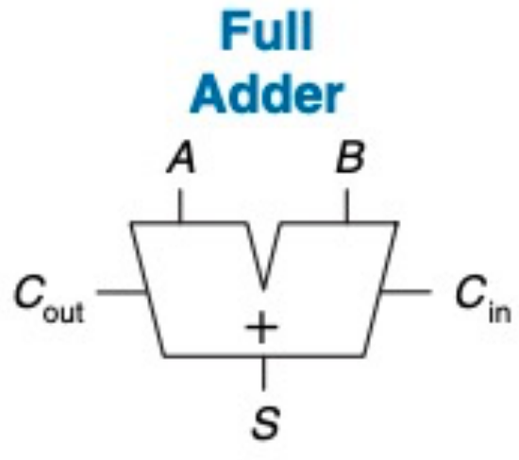
- Adder, already introduced
- Replace the ALU with +



A	B	C_{out}	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

$$S = A \oplus B$$
$$C_{out} = AB$$



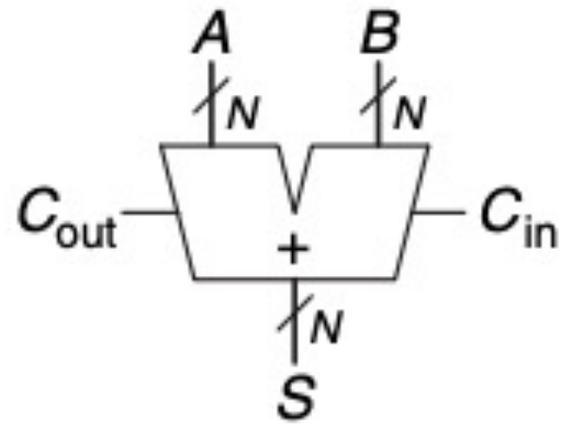


C_{in}	A	B	C_{out}	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

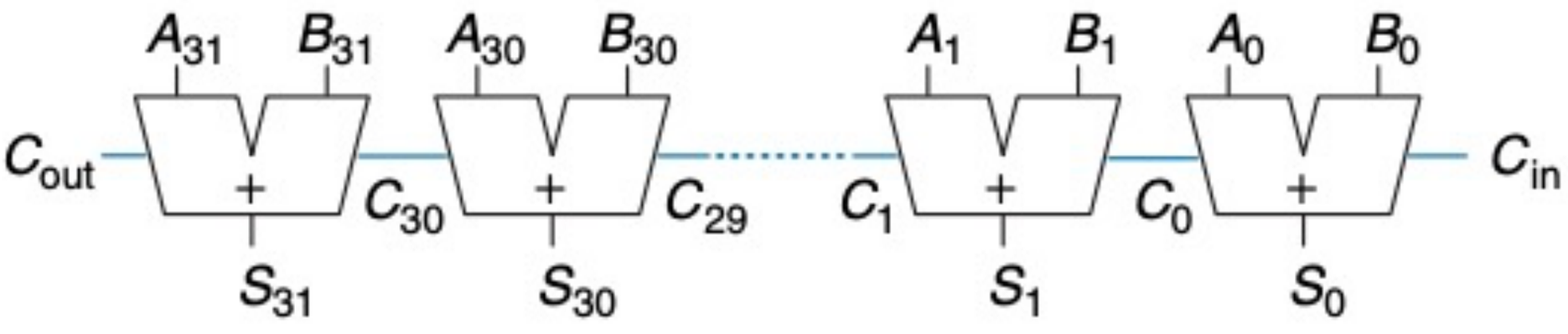
$$S = A \oplus B \oplus C_{in}$$
$$C_{out} = AB + AC_{in} + BC_{in}$$

N-bit Adder

- Carry Propagate Adder (CPA)
  - Full adder symbol, A, B, and S are busses rather than single bits



- Ripple-Carry Adder
  - 32-bit example



  - ***Performance issue?***

Ripple-Carry Adder

- RCA

Ripple-Carry Adder

- RCA
  - The Cout of LSB is cascaded/used as the Cin of its LSB+1

Ripple-Carry Adder

- RCA
  - The Cout of LSB is cascaded/used as the Cin of its LSB+1
  - Pros

Ripple-Carry Adder

- RCA
  - The Cout of LSB is cascaded/used as the Cin of its LSB+1
  - Pros
    - Straightforward, easy topology

Ripple-Carry Adder

- RCA
  - The Cout of LSB is cascaded/used as the Cin of its LSB+1
  - Pros
    - Straightforward, easy topology
  - Cons

Ripple-Carry Adder

- RCA
  - The Cout of LSB is cascaded/used as the Cin of its LSB+1
  - Pros
    - Straightforward, easy topology
  - Cons
    - Long-latency

Ripple-Carry Adder

- RCA
  - The Cout of LSB is cascaded/used as the Cin of its LSB+1
  - Pros
    - Straightforward, easy topology
  - Cons
    - Long-latency
    - MSB will have to wait for the results from LSB!

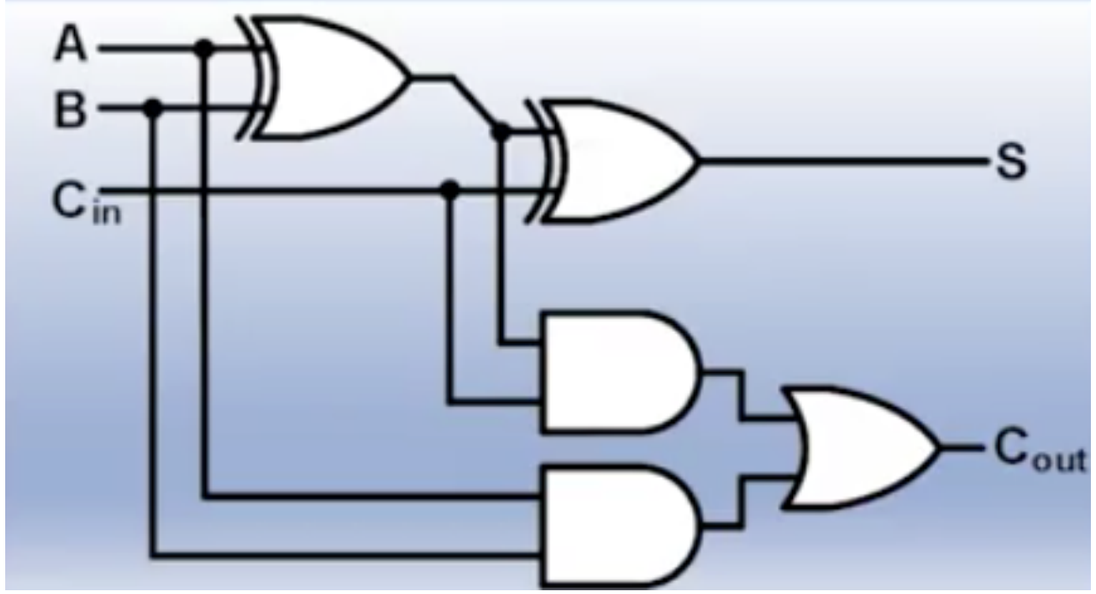
Delay Analysis

Delay Analysis

- Basic component

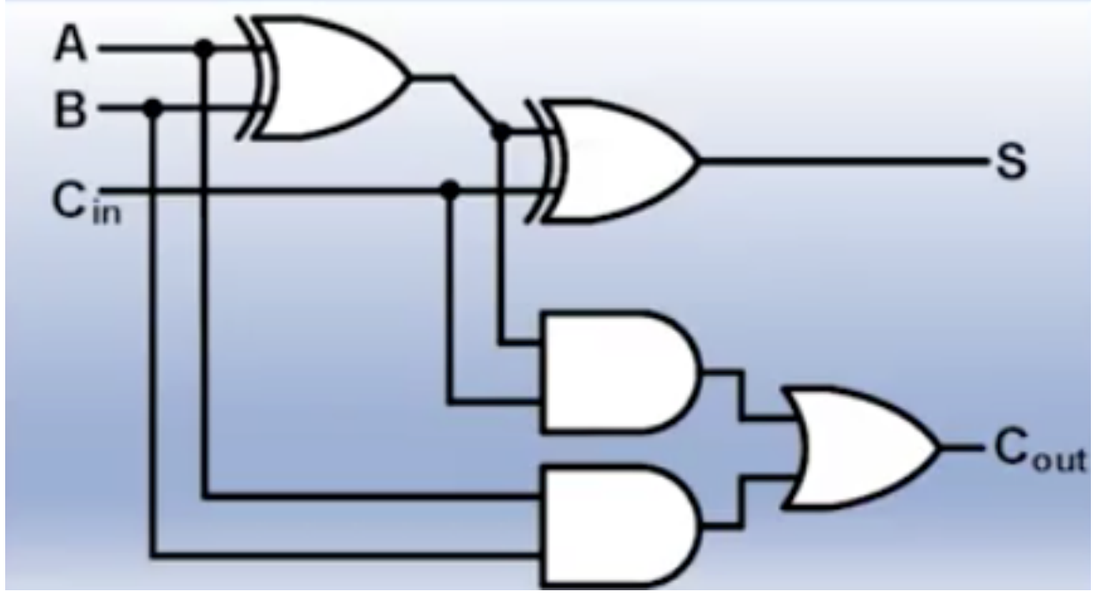
Delay Analysis

- Basic component



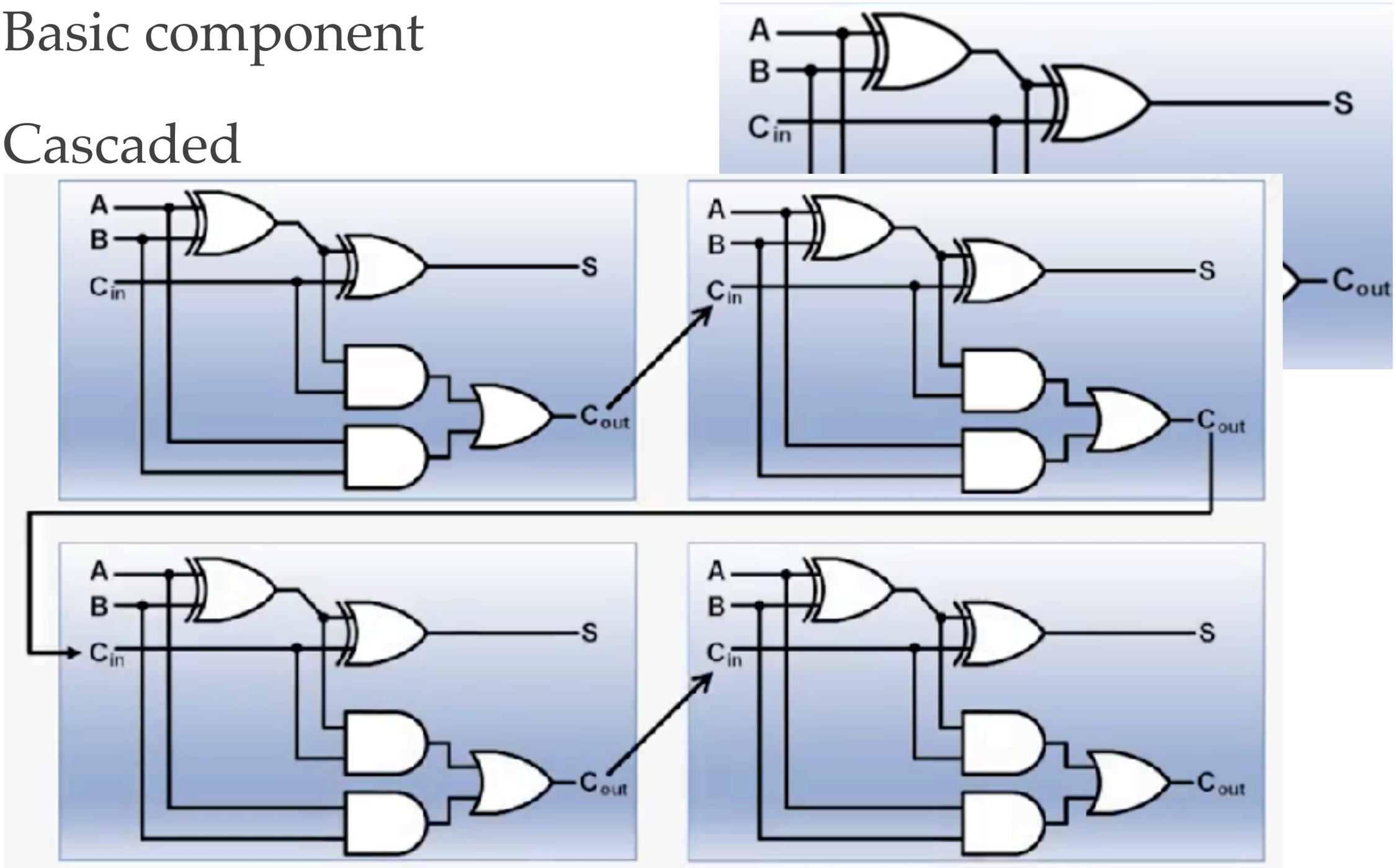
Delay Analysis

- Basic component
- Cascaded



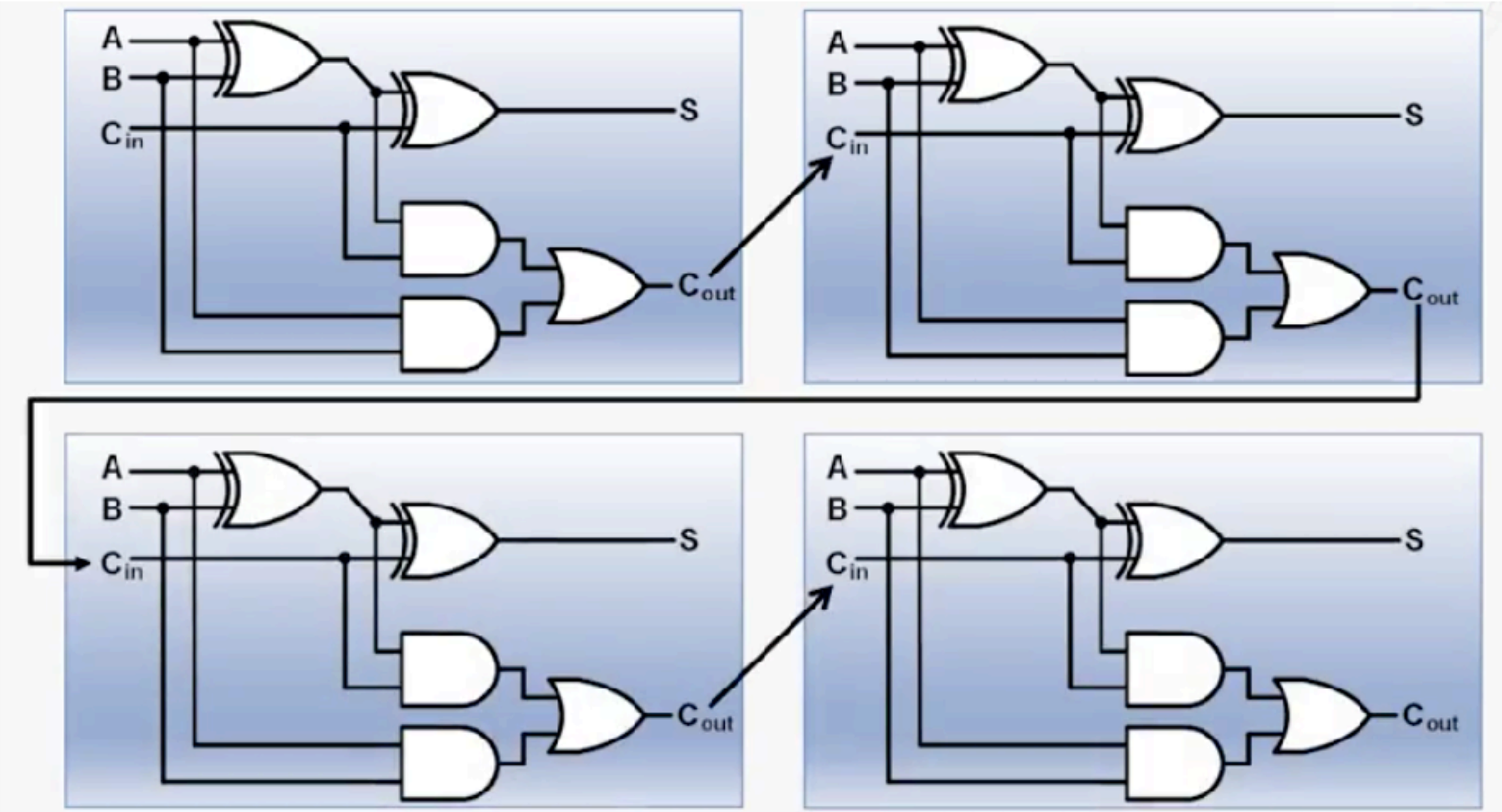
Delay Analysis

- Basic component
- Cascaded



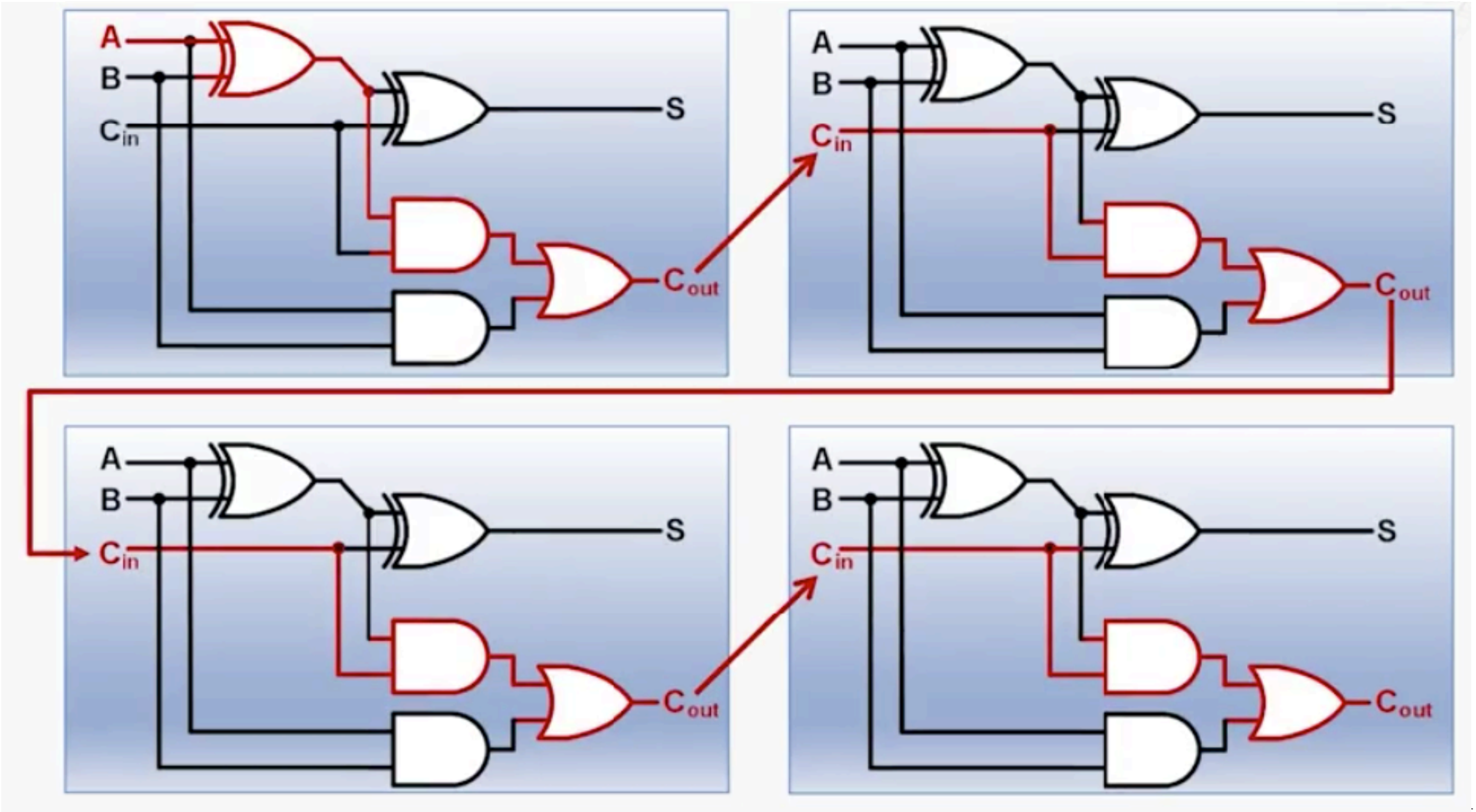
Critical Path Analysis

- Where is the critical path of the proposed schematic?



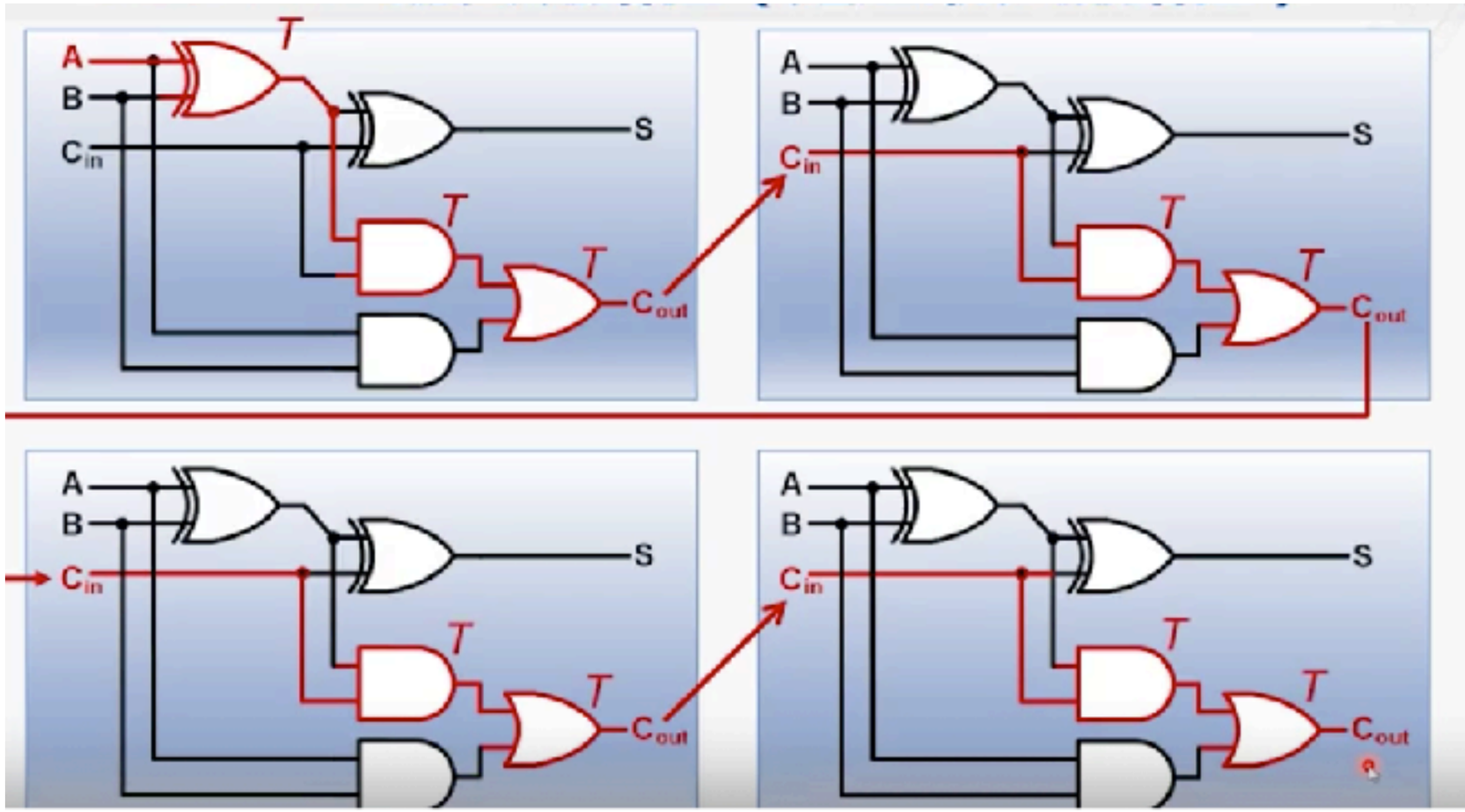
Critical Path Analysis

- Where is the critical path of the proposed schematic?



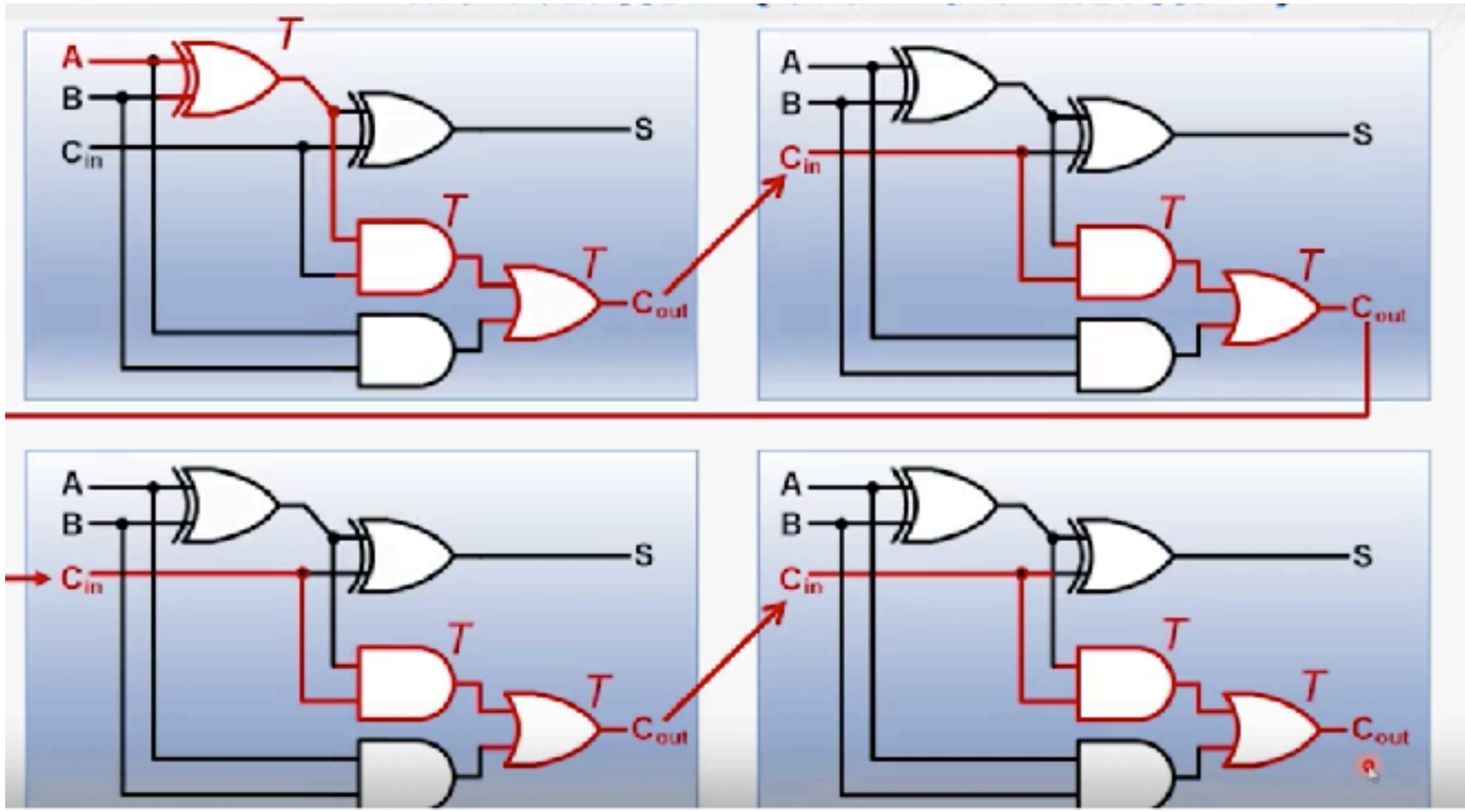
Delay Formulation

- Each gate has a delay of T (ideal)
- How much delay in total?
  - ***hand calculation***



Delay Formulation

- Each gate has a delay of T (ideal)
- How much delay in total?
  - ***hand calculation***
  - $2T^*N+T$

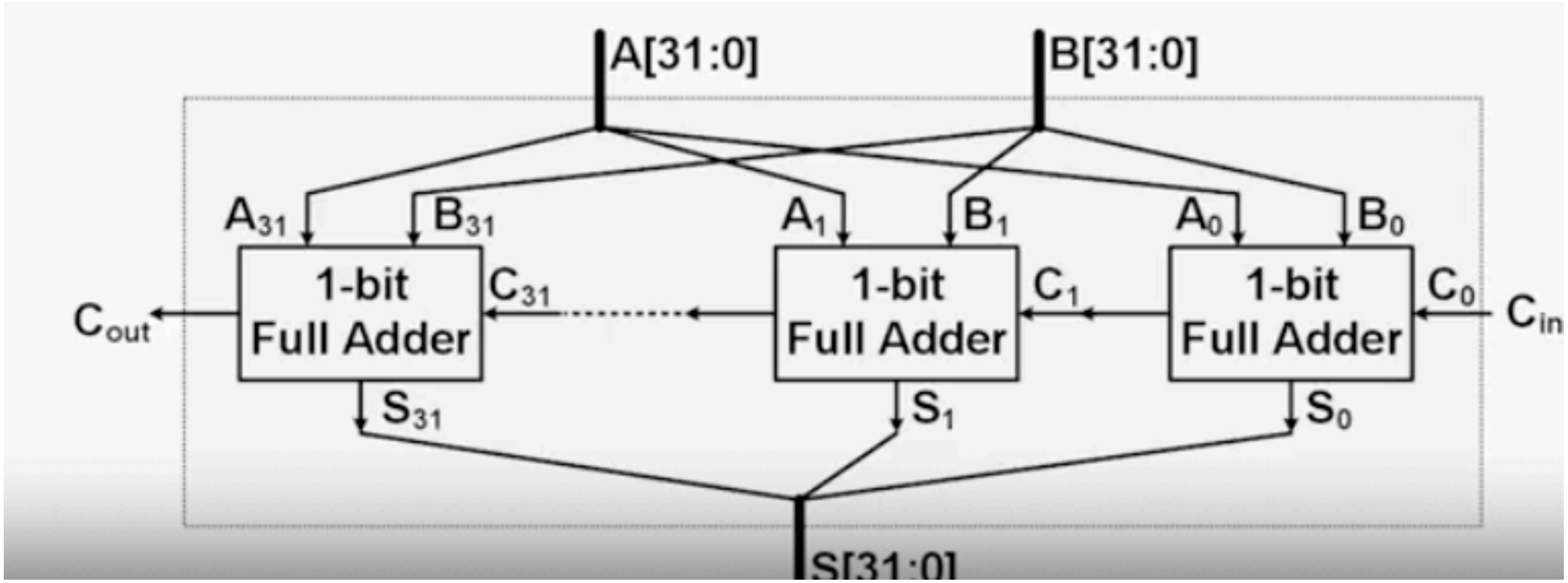


Delay Formulation

- How much delay for a 32-bit full adder?
- (T+T)*N+T where N=32 = 65T

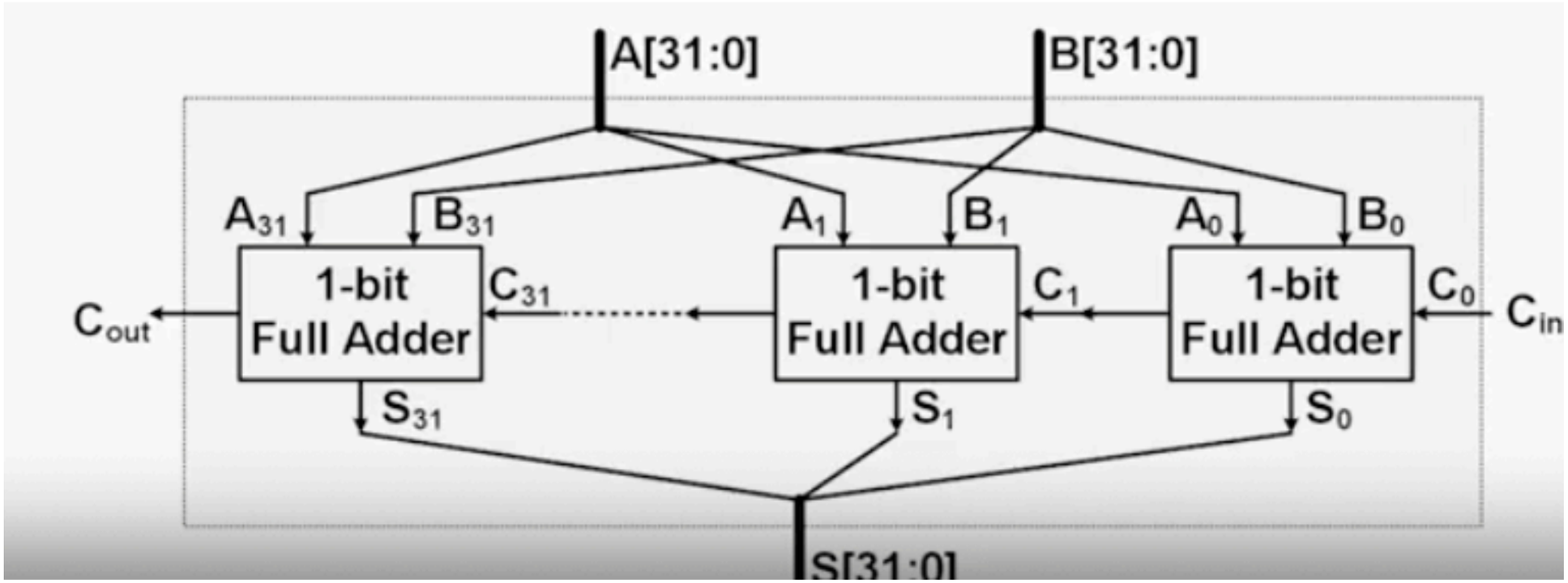
Delay Formulation

- How much delay for a 32-bit full adder?
- (T+T)*N+T where N=32 = 65T



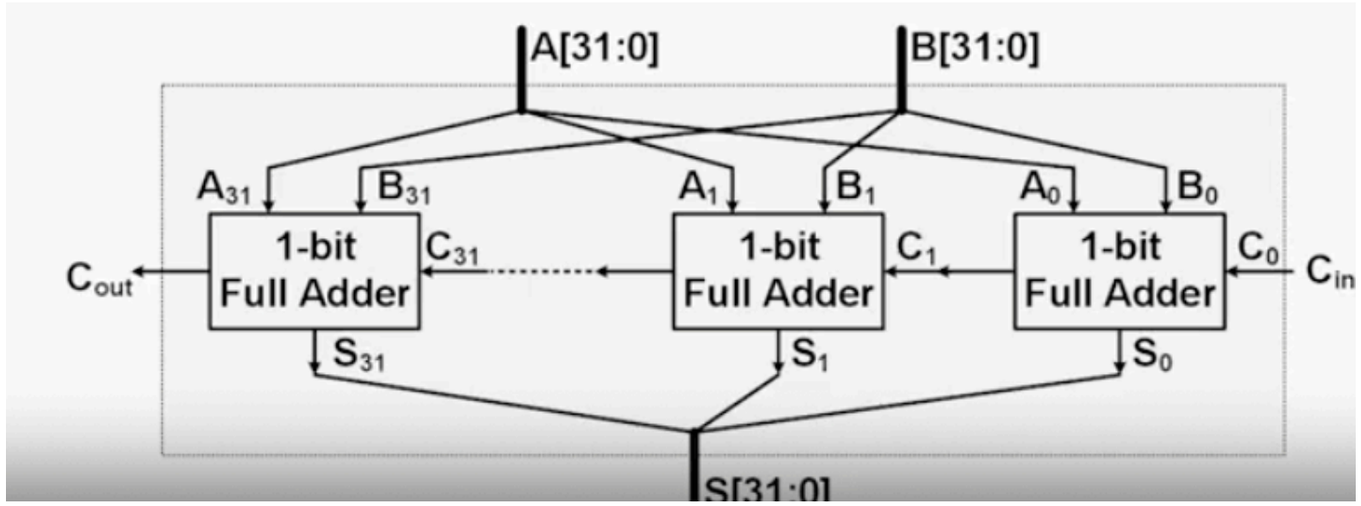
Delay Formulation

- How much delay for a 32-bit full adder?
- (T+T)*N+T where N=32 = 65T
- Good? Bad? What this number means?



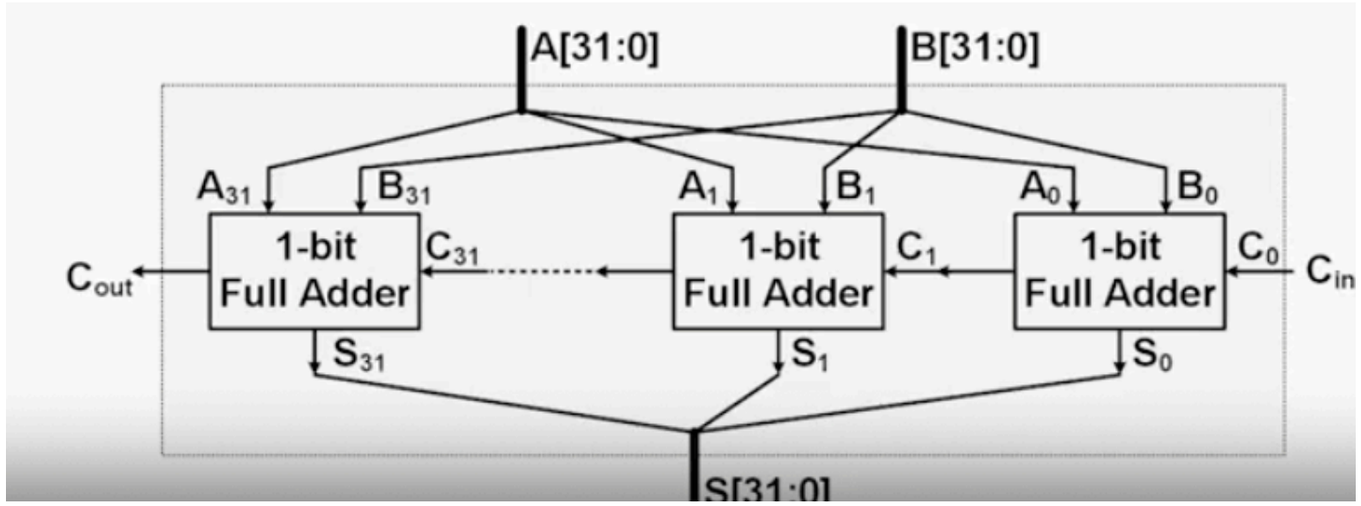
A Real-World Example

- 65T
- A7 SoC of iPhone 5s
  - 28nm CMOS
  - 1.3GHz (~0.66ns)



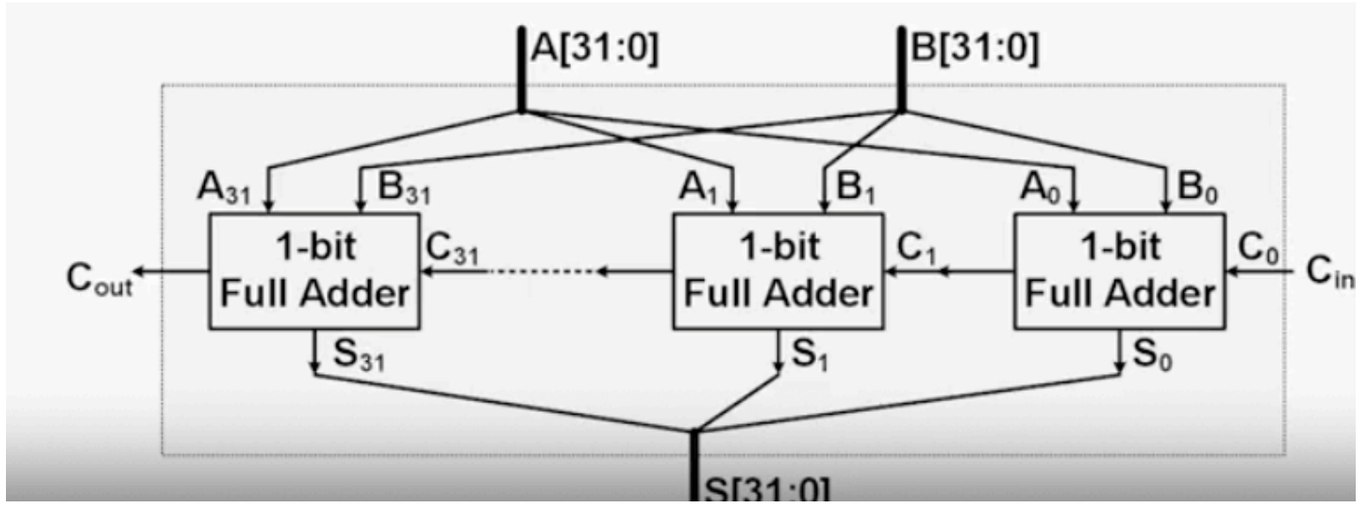
A Real-World Example

- 65T
- A7 SoC of iPhone 5s
  - 28nm CMOS
  - 1.3GHz (~0.66ns)



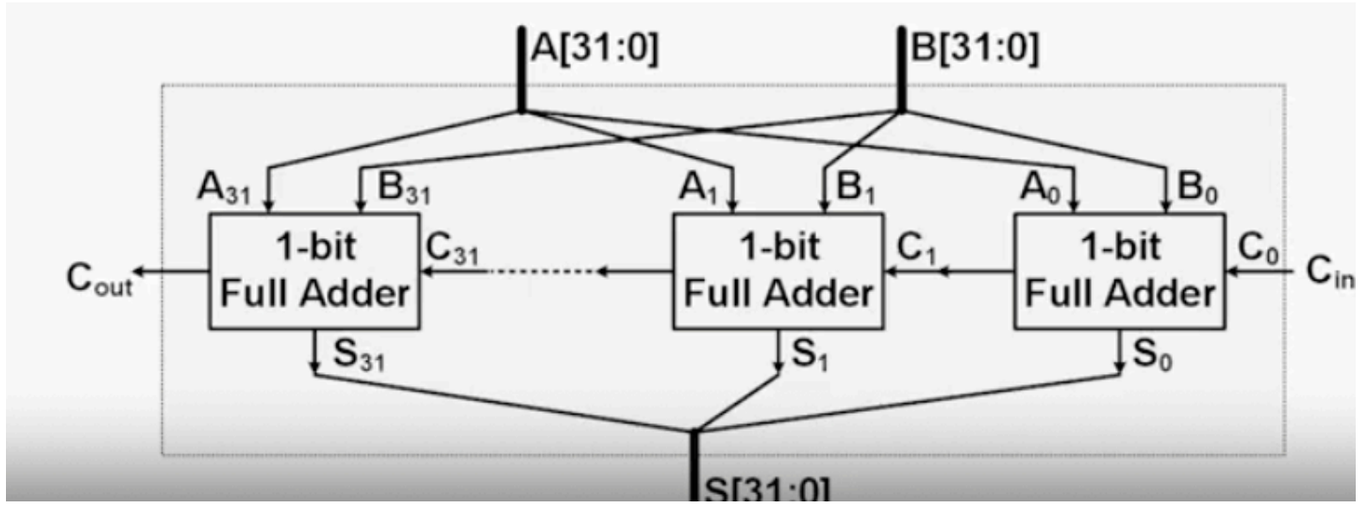
A Real-World Example

- 65T
- A7 SoC of iPhone 5s
  - 28nm CMOS
  - 1.3GHz (~0.66ns)
- Satisfying??
  - T=0.02ns for 28nm
  - No! Not even considering setup/hold!



A Real-World Example

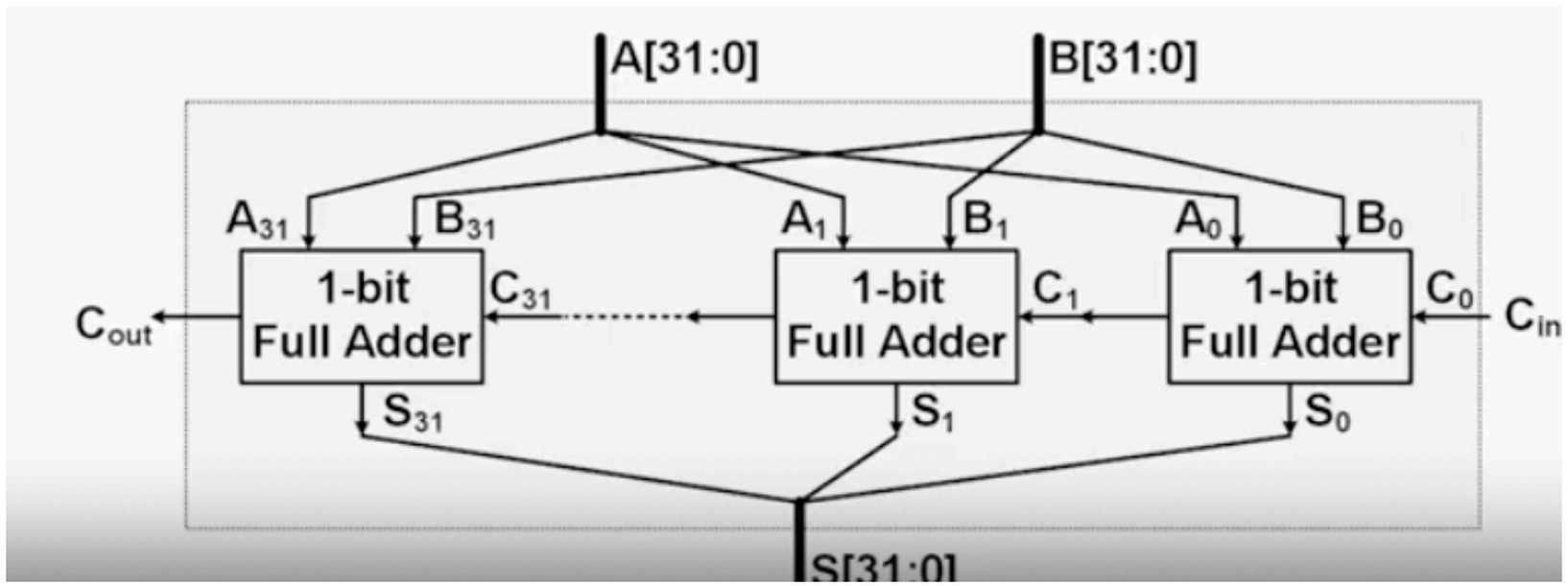
- 65T
- A7 SoC of iPhone 5s
  - 28nm CMOS
  - 1.3GHz (~0.66ns)
- Satisfying??
  - T=0.02ns for 28nm
  - No! Not even considering setup/hold!



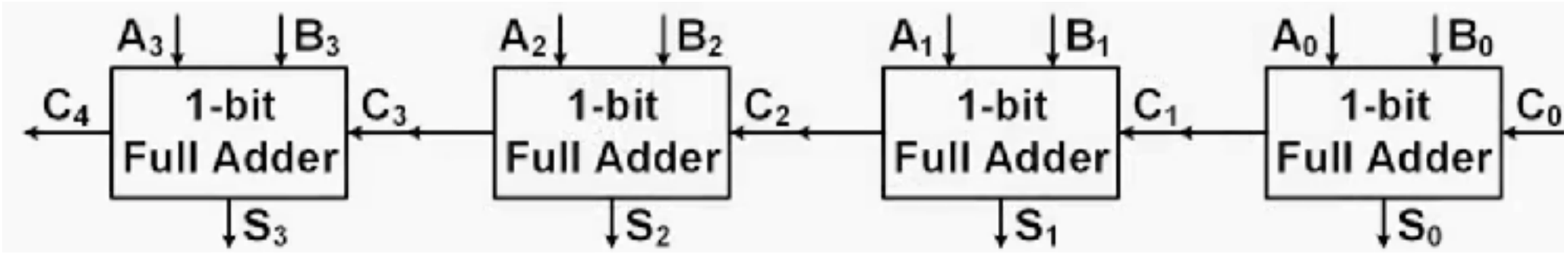
4-bit RCA	0.18ns	5.56GHz
32-bit RCA	1.3ns	769MHz

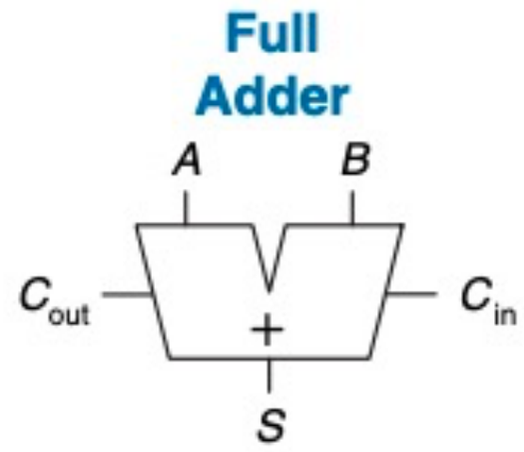
Need to Optimize the Adder

- Key issue
  - MSBs has to wait for Cout from LSBs
- How to optimize?
  - What if we can predict the Cout?



Analysis of Cout



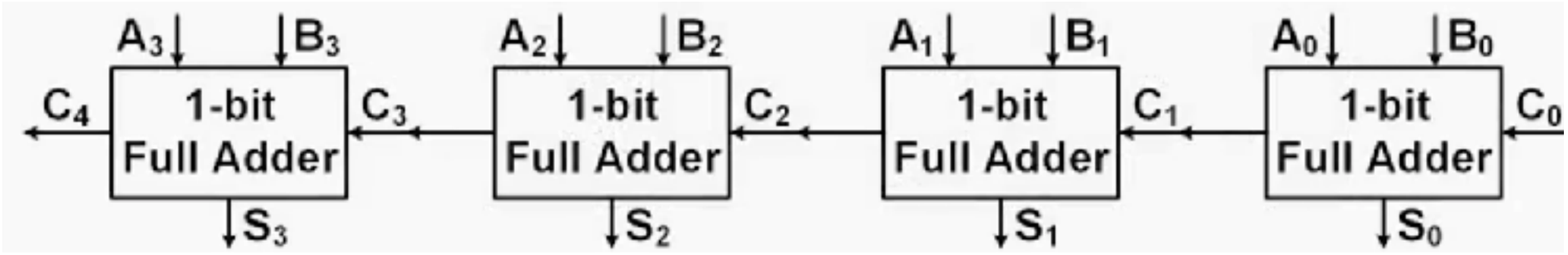


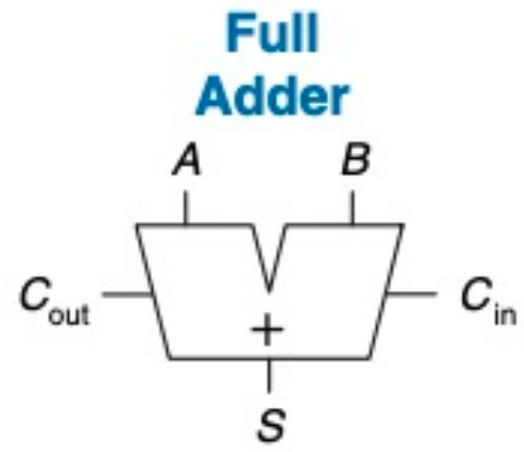
C_{in}	A	B	C_{out}	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

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

$$C_{out} = AB + AC_{in} + BC_{in}$$

Analysis of Cout





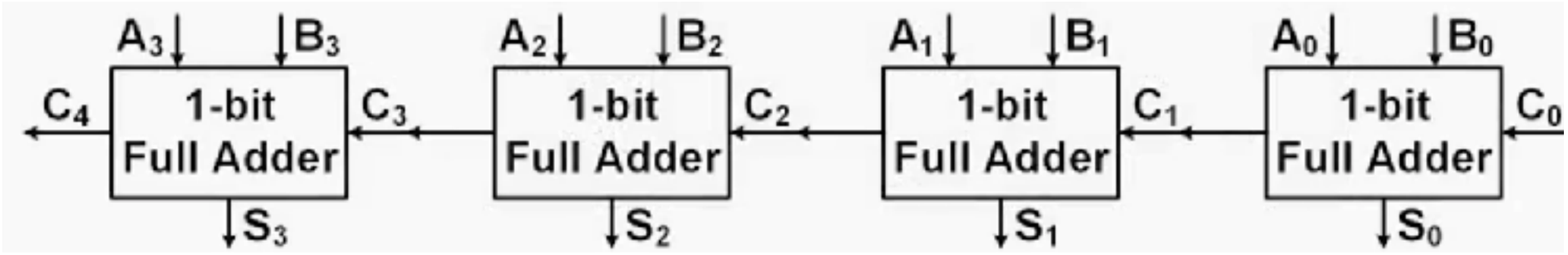
$C_{i+1} =$

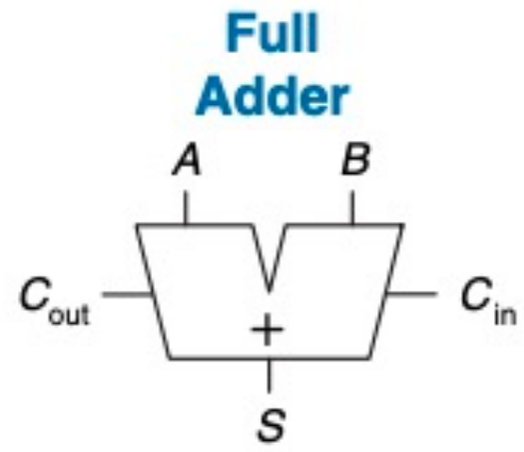
C_{in}	A	B	C_{out}	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

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

$$C_{out} = AB + AC_{in} + BC_{in}$$

Analysis of Cout





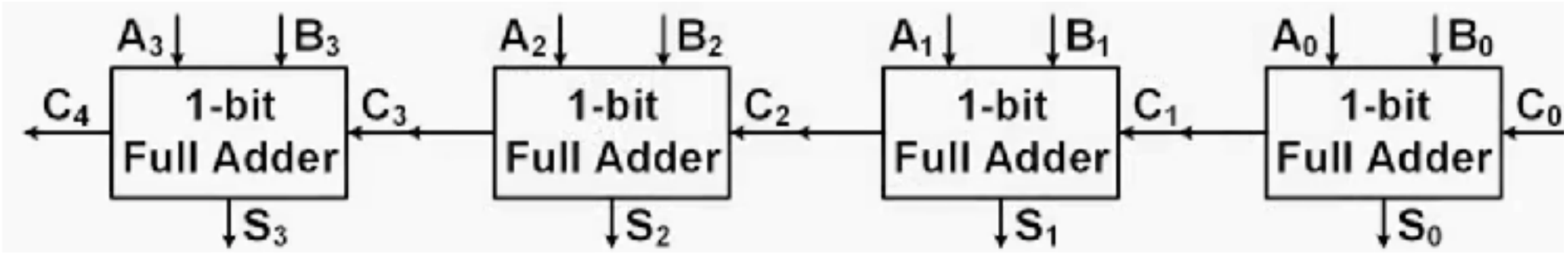
$$C_{i+1} = (A_i \cdot B_i) + (A_i \cdot C_i) + (B_i \cdot C_i)$$

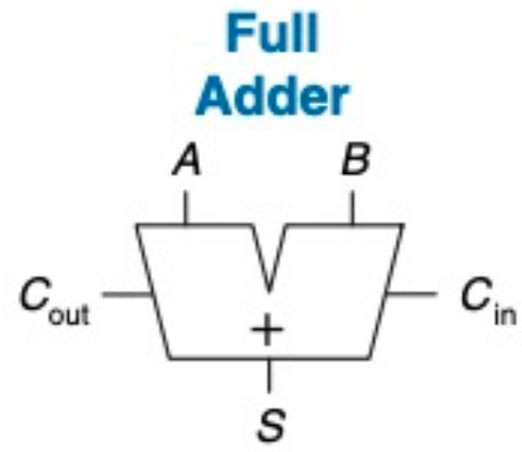
C_{in}	A	B	C_{out}	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

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

$$C_{out} = AB + AC_{in} + BC_{in}$$

Analysis of Cout





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

C_{in}	A	B	C_{out}	S
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

$$S = A \oplus B \oplus C_{in}$$
$$C_{out} = AB + AC_{in} + BC_{in}$$

Carry-Lookahead Adder

- Proposed for addressing the low-performance of ripple-carry adder
- How?

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

- The i-th adder will **generate** a carry-out, if both A and B are 1, ***independent of carry_in!***
- The i-th adder will **propagate** a carry-out, if either A or B is 1, ***and there is a of carry_in!***

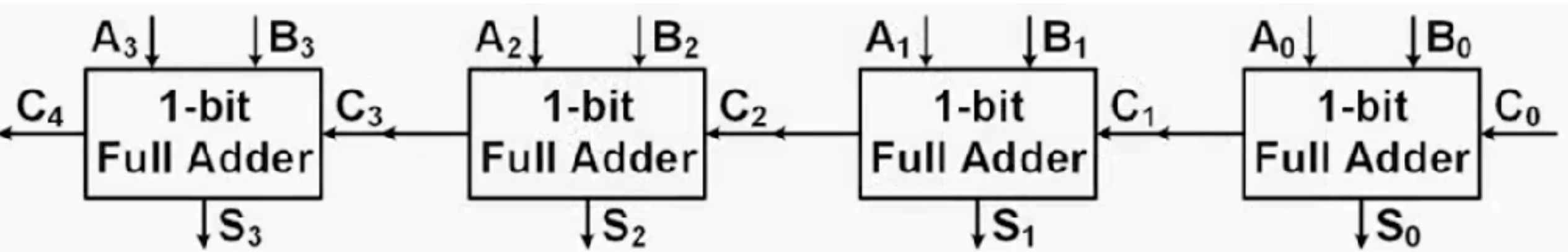
Analysis of Cout

- Define two new parameters:
  - Generate: Gi = Ai AND Bi
  - Propagate: Pi = Ai + Bi

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

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

$$= (A_i \cdot B_i) + (A_i + B_i) \cdot C_i$$



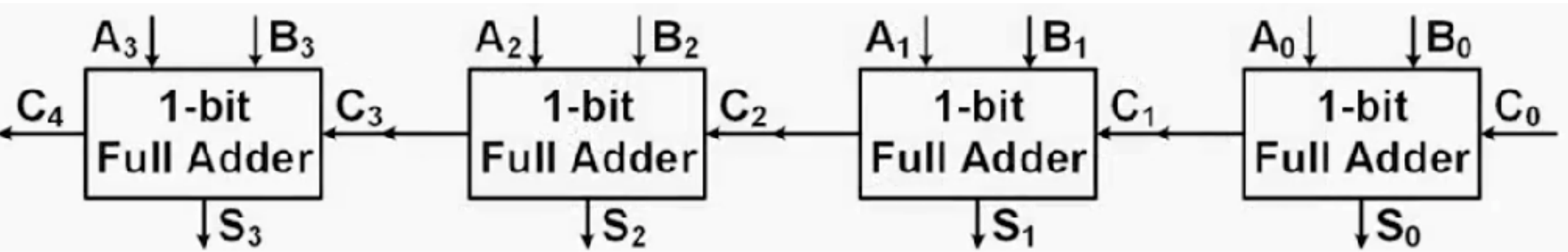
Analysis of Cout

- Define two new parameters:
  - Generate: Gi = Ai AND Bi
  - Propagate: Pi = Ai + Bi

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

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

$$= (A_i \cdot B_i) + (A_i + B_i) \cdot C_i$$



Know the Values in Advance

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

Know the Values in Advance

$$C_1 = G_0 + P_0 \cdot C_0$$

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

Know the Values in Advance

$$C_1 = G_0 + P_0 \cdot C_0$$

$$C_2 = G_1 + P_1 \cdot C_1$$

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

Know the Values in Advance

$$C_1 = G_0 + P_0 \cdot C_0$$

$$C_2 = G_1 + P_1 \cdot C_1$$

$$= G_1 + P_1 \cdot (G_0 + P_0 \cdot C_0)$$

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

Know the Values in Advance

$$C_1 = G_0 + P_0 \cdot C_0$$

$$\begin{aligned} C_2 &= G_1 + P_1 \cdot C_1 \\ &= G_1 + P_1 \cdot (G_0 + P_0 \cdot C_0) \\ &= G_1 + P_1 \cdot G_0 + P_1 \cdot P_0 \cdot C_0 \end{aligned}$$

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

Know the Values in Advance

$$C_1 = G_0 + P_0 \cdot C_0$$

$$\begin{aligned} C_2 &= G_1 + P_1 \cdot C_1 \\ &= G_1 + P_1 \cdot (G_0 + P_0 \cdot C_0) \\ &= G_1 + P_1 \cdot G_0 + P_1 \cdot P_0 \cdot C_0 \end{aligned}$$

$$\begin{aligned} C_3 &= G_2 + P_2 \cdot C_2 \\ &= G_2 + P_2 \cdot (G_1 + P_1 \cdot G_0 + P_1 \cdot P_0 \cdot C_0) \\ &= G_2 + P_2 \cdot G_1 + P_2 \cdot P_1 \cdot G_0 + P_2 \cdot P_1 \cdot P_0 \cdot C_0 \end{aligned}$$

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

Know the Values in Advance

$$C_1 = G_0 + P_0 \cdot C_0$$

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

$$\begin{aligned} C_2 &= G_1 + P_1 \cdot C_1 \\ &= G_1 + P_1 \cdot (G_0 + P_0 \cdot C_0) \\ &= G_1 + P_1 \cdot G_0 + P_1 \cdot P_0 \cdot C_0 \end{aligned}$$

$$\begin{aligned} C_3 &= G_2 + P_2 \cdot C_2 \\ &= G_2 + P_2 \cdot (G_1 + P_1 \cdot G_0 + P_1 \cdot P_0 \cdot C_0) \\ &= G_2 + P_2 \cdot G_1 + P_2 \cdot P_1 \cdot G_0 + P_2 \cdot P_1 \cdot P_0 \cdot C_0 \end{aligned}$$

$$\begin{aligned} C_4 &= G_3 + P_3 \cdot C_3 \\ &= G_3 + P_3 \cdot (G_2 + P_2 \cdot G_1 + P_2 \cdot P_1 \cdot G_0 + P_2 \cdot P_1 \cdot P_0 \cdot C_0) \\ &= G_3 + P_3 \cdot G_2 + P_3 \cdot P_2 \cdot G_1 + P_3 \cdot P_2 \cdot P_1 \cdot G_0 + P_3 \cdot P_2 \cdot P_1 \cdot P_0 \cdot C_0 \end{aligned}$$

Detailed Analysis of C4

$$\mathbf{C_4} = G_3 + P_3 \cdot C_3$$
$$= G_3 + P_3 \cdot (G_2 + P_2 \cdot G_1 + P_2 \cdot P_1 \cdot G_0 + P_2 \cdot P_1 \cdot P_0 \cdot C_0)$$
$$\mathbf{= G_3 + P_3 \cdot G_2 + P_3 \cdot P_2 \cdot G_1 + P_3 \cdot P_2 \cdot P_1 \cdot G_0 + P_3 \cdot P_2 \cdot P_1 \cdot P_0 \cdot C_0}$$

Detailed Analysis of C4

$$C_4 = G_3 + P_3 \cdot C_3$$
$$= G_3 + P_3 \cdot (G_2 + P_2 \cdot G_1 + P_2 \cdot P_1 \cdot G_0 + P_2 \cdot P_1 \cdot P_0 \cdot C_0)$$
$$= \mathbf{G_3 + P_3 \cdot G_2 + P_3 \cdot P_2 \cdot G_1 + P_3 \cdot P_2 \cdot P_1 \cdot G_0 + P_3 \cdot P_2 \cdot P_1 \cdot P_0 \cdot C_0}$$

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

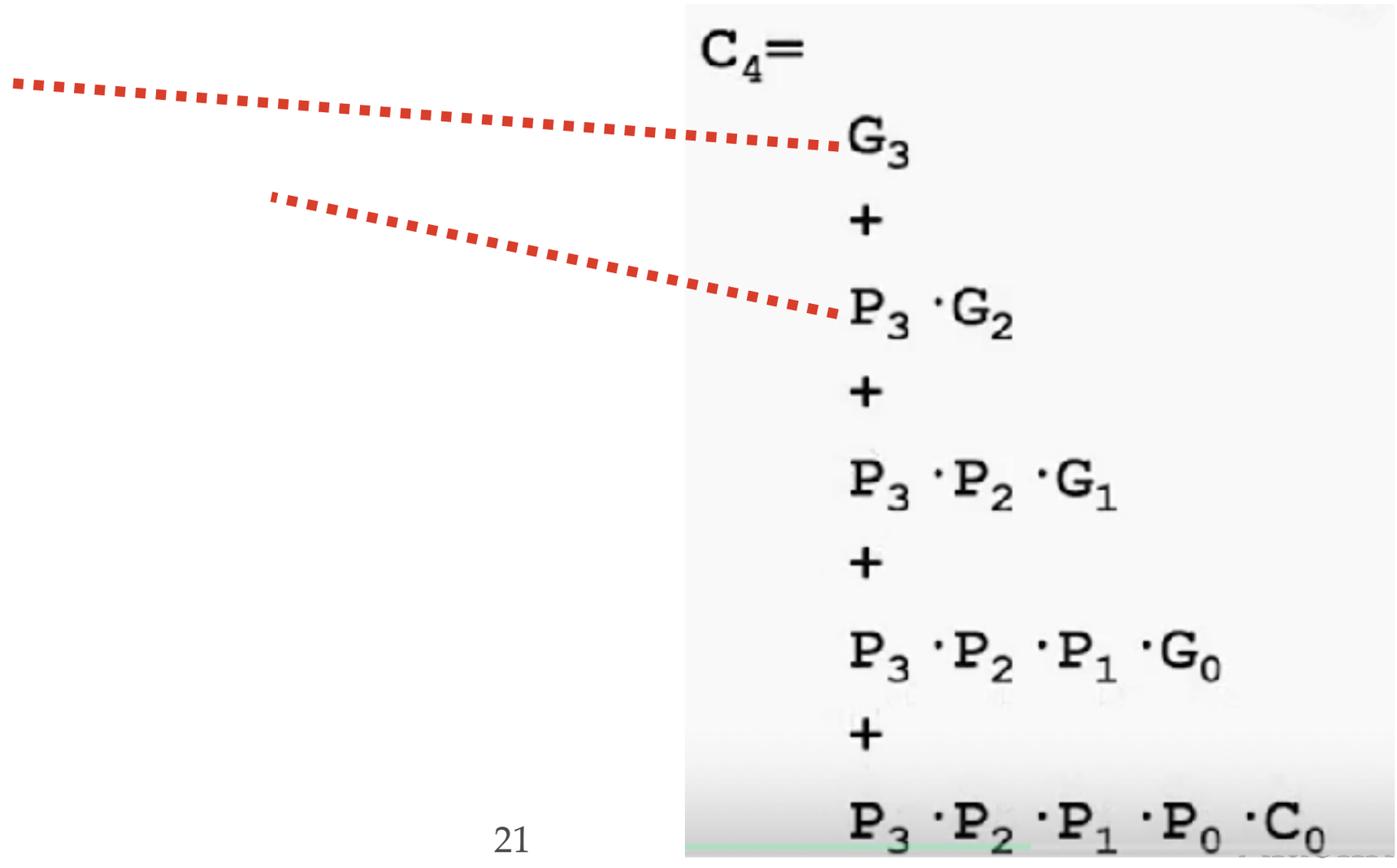
Detailed Analysis of C4

$$C_4 = G_3 + P_3 \cdot C_3$$
$$= G_3 + P_3 \cdot (G_2 + P_2 \cdot G_1 + P_2 \cdot P_1 \cdot G_0 + P_2 \cdot P_1 \cdot P_0 \cdot C_0)$$
$$= \mathbf{G_3 + P_3 \cdot G_2 + P_3 \cdot P_2 \cdot G_1 + P_3 \cdot P_2 \cdot P_1 \cdot G_0 + P_3 \cdot P_2 \cdot P_1 \cdot P_0 \cdot C_0}$$

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

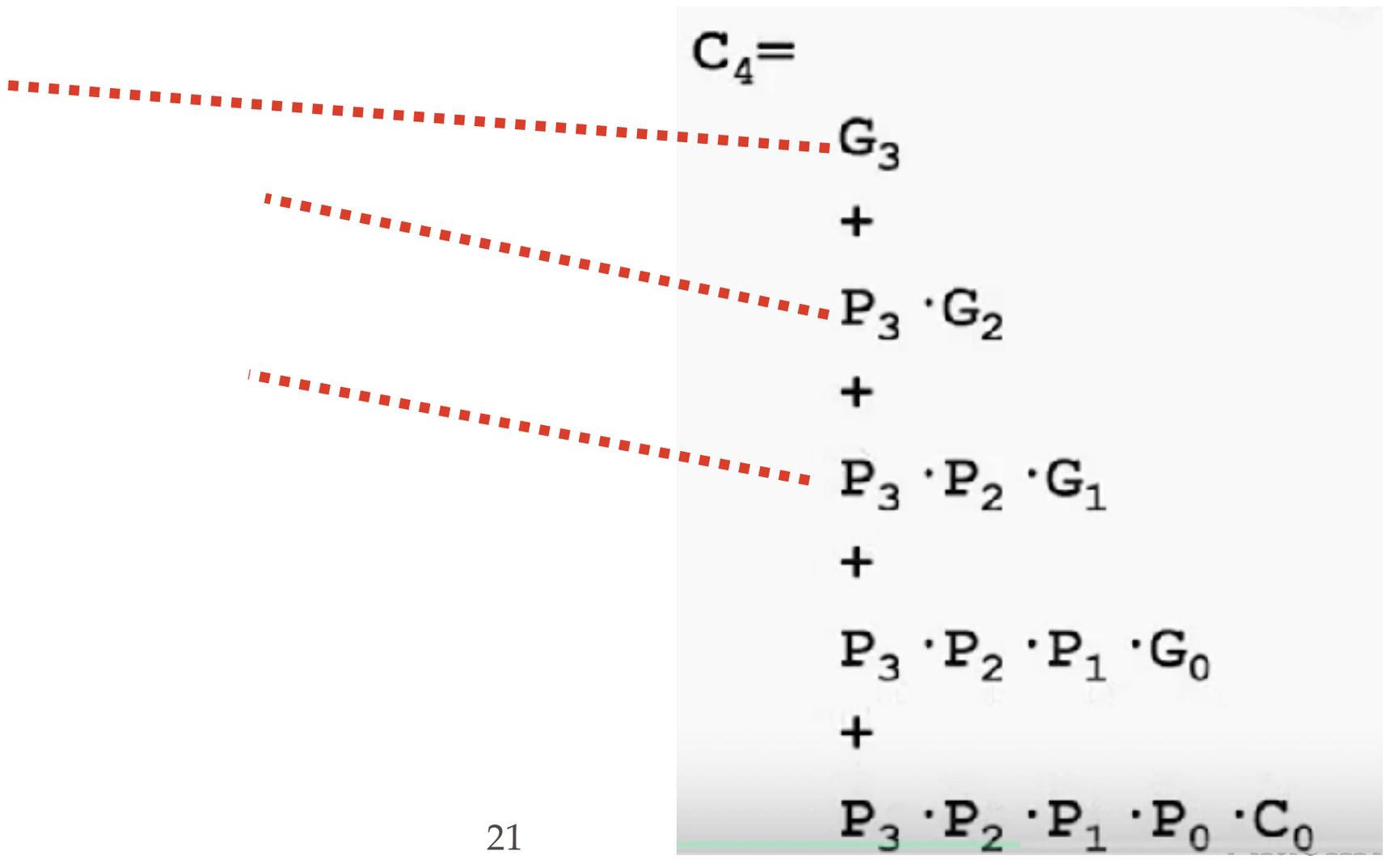
Detailed Analysis of C4

$$
\begin{aligned}
\mathbf{C_4} &= G_3 + P_3 \cdot C_3 \\
&= G_3 + P_3 \cdot (G_2 + P_2 \cdot G_1 + P_2 \cdot P_1 \cdot G_0 + P_2 \cdot P_1 \cdot P_0 \cdot C_0) \\
&= \mathbf{G_3 + P_3 \cdot G_2 + P_3 \cdot P_2 \cdot G_1 + P_3 \cdot P_2 \cdot P_1 \cdot G_0 + P_3 \cdot P_2 \cdot P_1 \cdot P_0 \cdot C_0}
\end{aligned}
$$



Detailed Analysis of C4

$$
\begin{aligned}
\mathbf{C_4} &= G_3 + P_3 \cdot C_3 \\
&= G_3 + P_3 \cdot (G_2 + P_2 \cdot G_1 + P_2 \cdot P_1 \cdot G_0 + P_2 \cdot P_1 \cdot P_0 \cdot C_0) \\
&= \mathbf{G_3 + P_3 \cdot G_2 + P_3 \cdot P_2 \cdot G_1 + P_3 \cdot P_2 \cdot P_1 \cdot G_0 + P_3 \cdot P_2 \cdot P_1 \cdot P_0 \cdot C_0}
\end{aligned}
$$



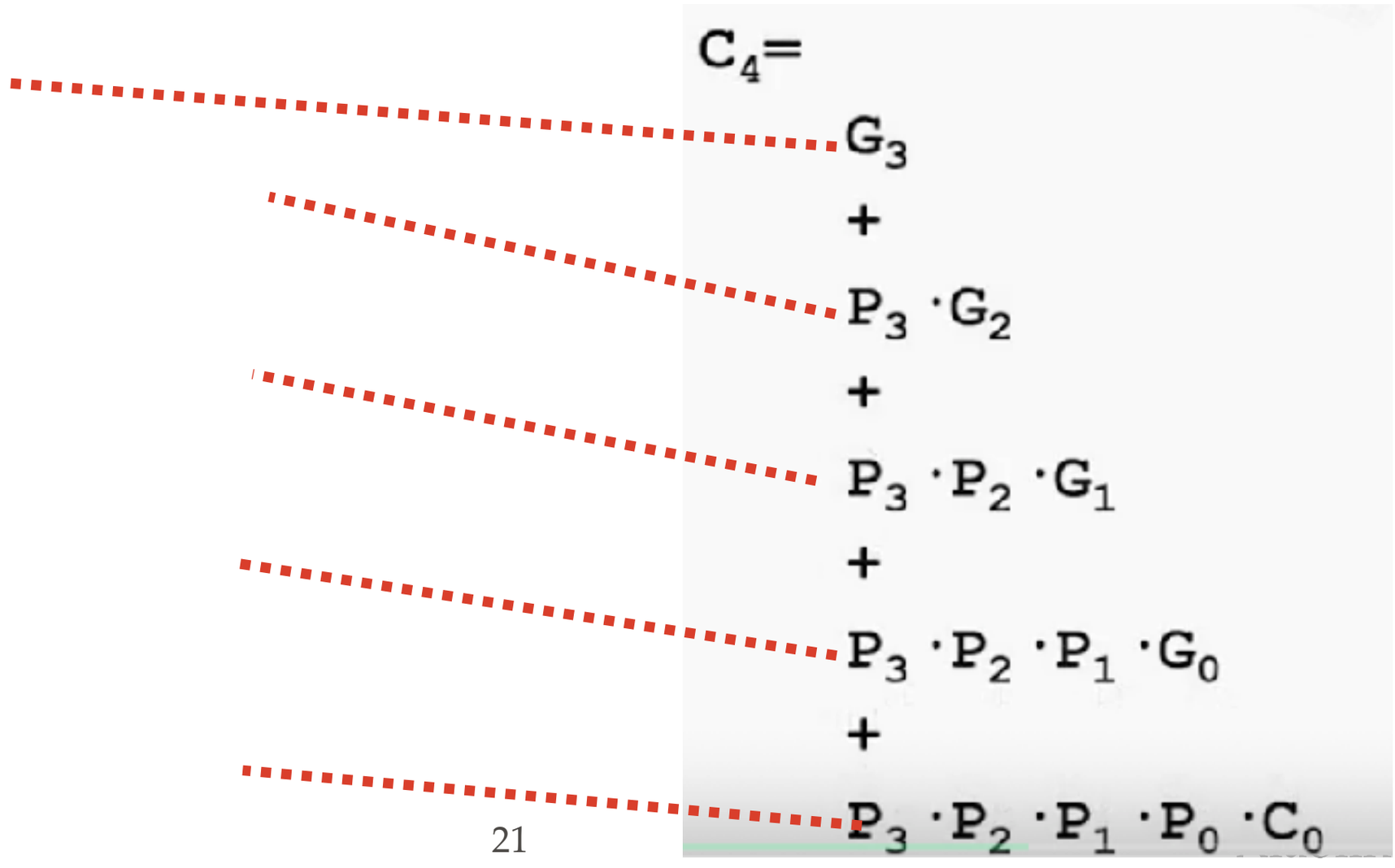
Detailed Analysis of C4

$$
\begin{aligned}
\mathbf{C_4} &= G_3 + P_3 \cdot C_3 \\
&= G_3 + P_3 \cdot (G_2 + P_2 \cdot G_1 + P_2 \cdot P_1 \cdot G_0 + P_2 \cdot P_1 \cdot P_0 \cdot C_0) \\
&= \mathbf{G_3 + P_3 \cdot G_2 + P_3 \cdot P_2 \cdot G_1 + P_3 \cdot P_2 \cdot P_1 \cdot G_0 + P_3 \cdot P_2 \cdot P_1 \cdot P_0 \cdot C_0}
\end{aligned}
$$

$$
\begin{aligned}
\mathbf{C_4} = \\
& \mathbf{G_3} \\
+ \\
& \mathbf{P_3 \cdot G_2} \\
+ \\
& \mathbf{P_3 \cdot P_2 \cdot G_1} \\
+ \\
& \mathbf{P_3 \cdot P_2 \cdot P_1 \cdot G_0} \\
+ \\
& \mathbf{P_3 \cdot P_2 \cdot P_1 \cdot P_0 \cdot C_0}
\end{aligned}
$$

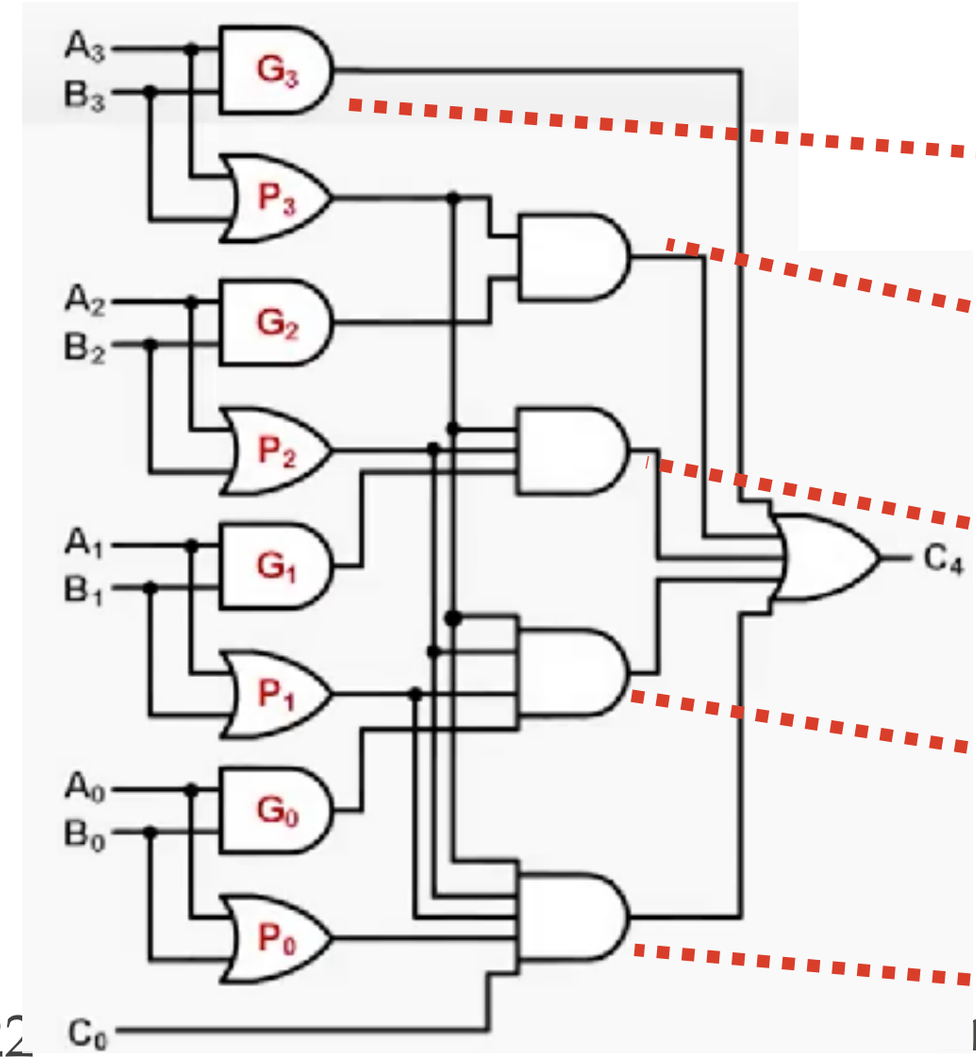
Detailed Analysis of C4

$$
\begin{aligned}
\mathbf{C_4} &= G_3 + P_3 \cdot C_3 \\
&= G_3 + P_3 \cdot (G_2 + P_2 \cdot G_1 + P_2 \cdot P_1 \cdot G_0 + P_2 \cdot P_1 \cdot P_0 \cdot C_0) \\
&= \mathbf{G_3 + P_3 \cdot G_2 + P_3 \cdot P_2 \cdot G_1 + P_3 \cdot P_2 \cdot P_1 \cdot G_0 + P_3 \cdot P_2 \cdot P_1 \cdot P_0 \cdot C_0}
\end{aligned}
$$



Detailed Analysis of C4

$$C_4 = G_3 + P_3 \cdot C_3$$
$$= G_3 + P_3 \cdot (G_2 + P_2 \cdot G_1 + P_2 \cdot P_1 \cdot G_0 + P_2 \cdot P_1 \cdot P_0 \cdot C_0)$$
$$= \mathbf{G_3 + P_3 \cdot G_2 + P_3 \cdot P_2 \cdot G_1 + P_3 \cdot P_2 \cdot P_1 \cdot G_0 + P_3 \cdot P_2 \cdot P_1 \cdot P_0 \cdot C_0}$$



$C_4 =$

G_3

$+$

$P_3 \cdot G_2$

$+$

$P_3 \cdot P_2 \cdot G_1$

$+$

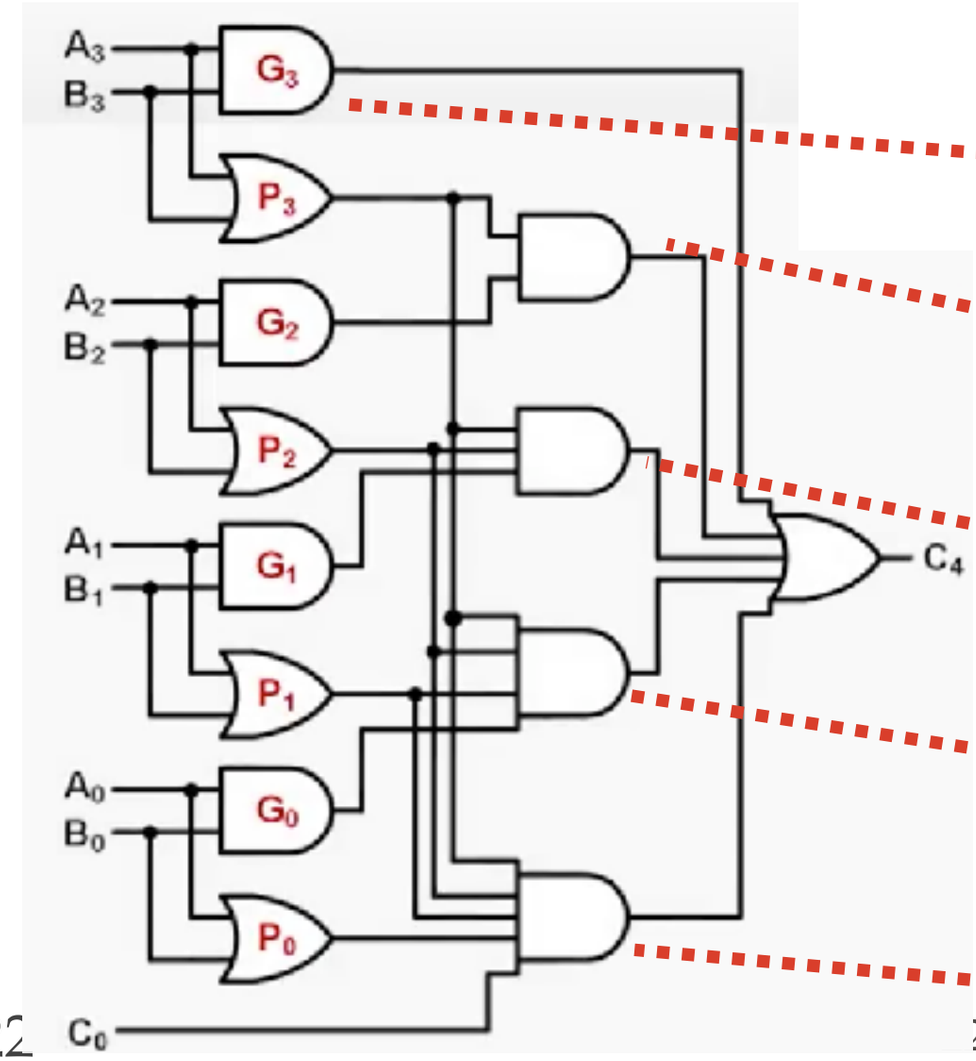
$P_3 \cdot P_2 \cdot P_1 \cdot G_0$

$+$

$P_3 \cdot P_2 \cdot P_1 \cdot P_0 \cdot C_0$

Detailed Analysis of C4

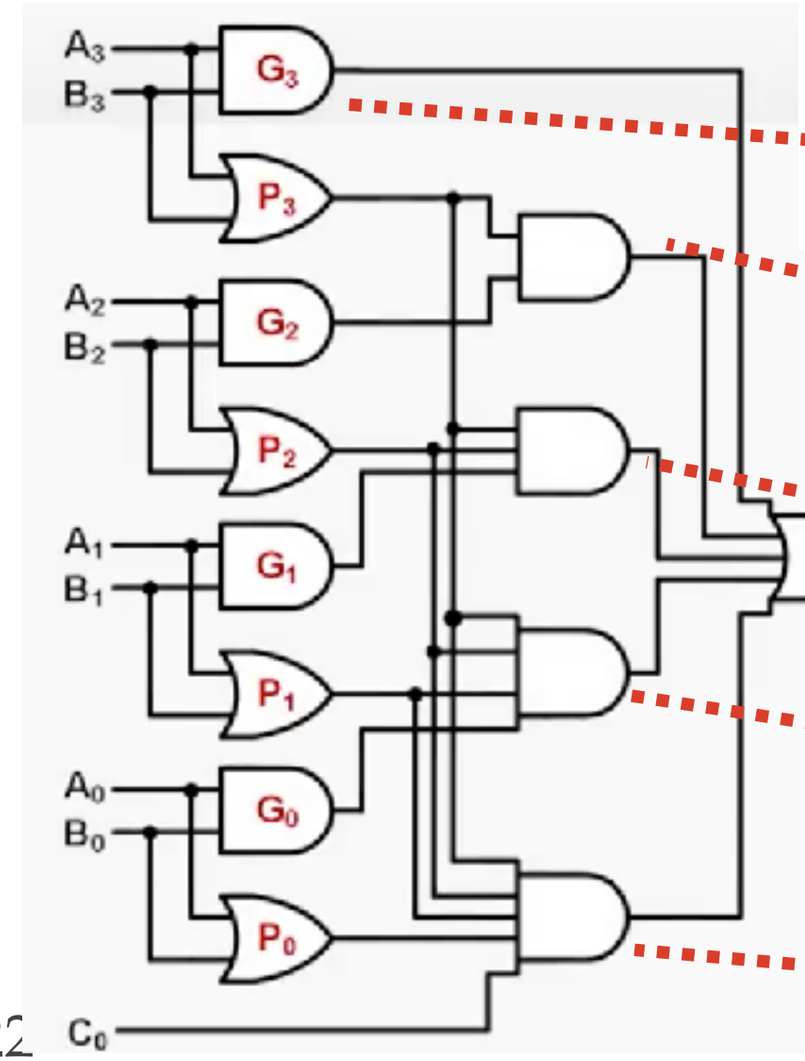
- Advantages?
  - The latency of Ci+1 is constant! Not dependent on N anymore



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

Detailed Analysis of C4

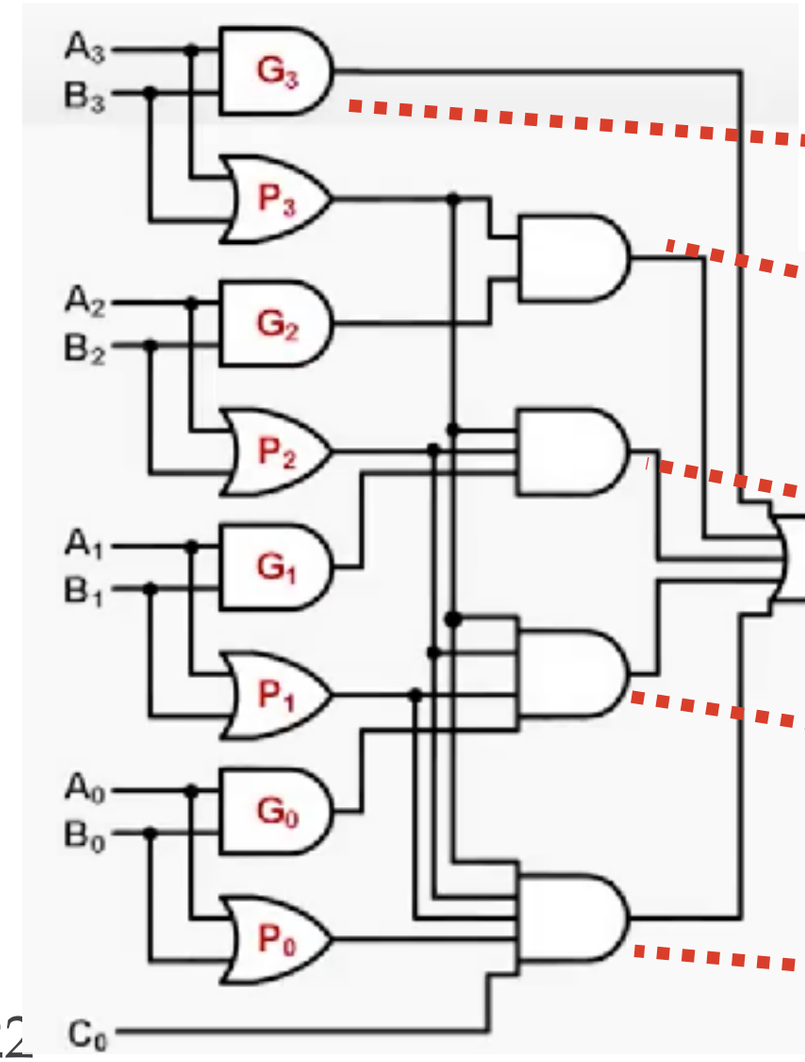
- Any disadvantages?



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

Detailed Analysis of C4

- Any disadvantages?
  - With the adder becomes wider, the circuit will be very complicated!



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

Further Optimization: Block of Adders

- Applying the concepts of "generate" and "propagate" to multiple-bit blocks/adders
  - A block ***generates*** a carry_out independent of the carry_in
  - A block ***propagates*** a carry_out if there is a carry_in

- Two new variables:
  - Gi:j and Pi:j

$$G_{3:0} = G_3 + P_3(G_2 + P_2(G_1 + P_1G_0))$$

$$P_{3:0} = P_3P_2P_1P_0$$

Further Optimization: Block of Adders

- Applying the concepts of "generate" and "propagate" to multiple-bit blocks/adders
  - A block ***generates*** a carry_out independent of the carry_in
  - A block ***propagates*** a carry_out if there is a carry_in

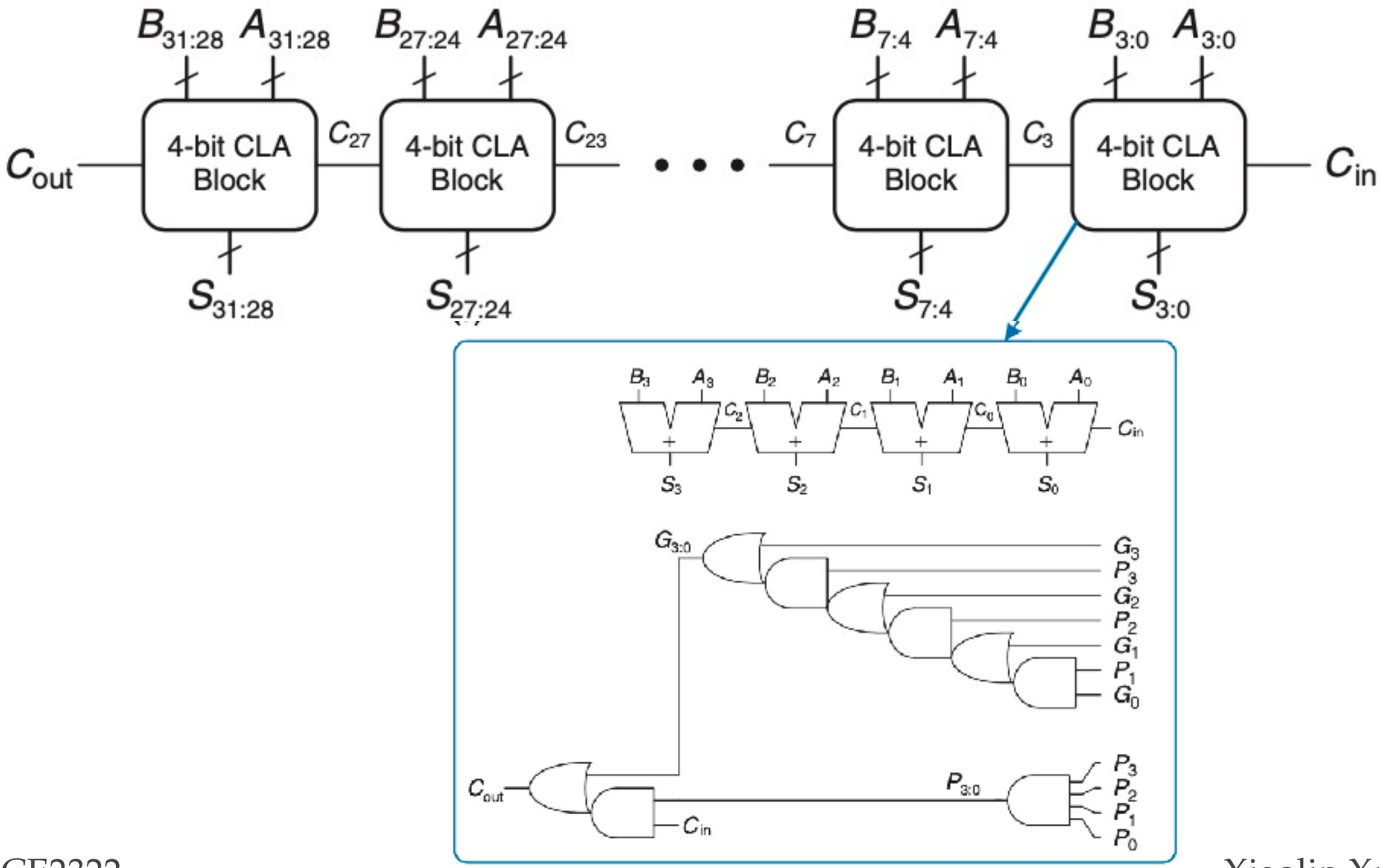
- Two new variables:
  - Gi:j and Pi:j

$$G_{3:0} = G_3 + P_3(G_2 + P_2(G_1 + P_1 G_0))$$

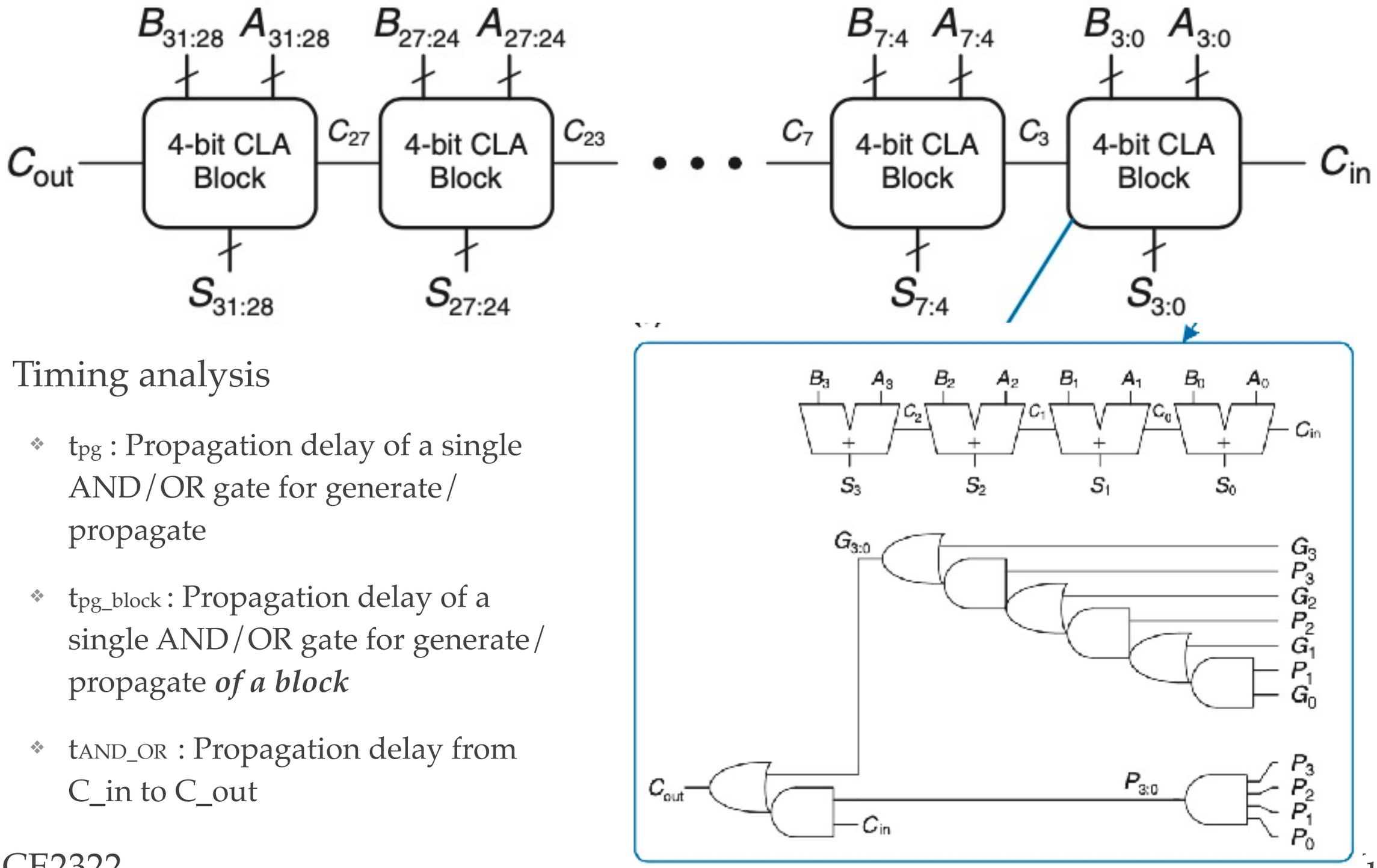
$$P_{3:0} = P_3 P_2 P_1 P_0$$

$$C_i = G_{i:j} + P_{i:j} C_j$$

32-bit CLA Example

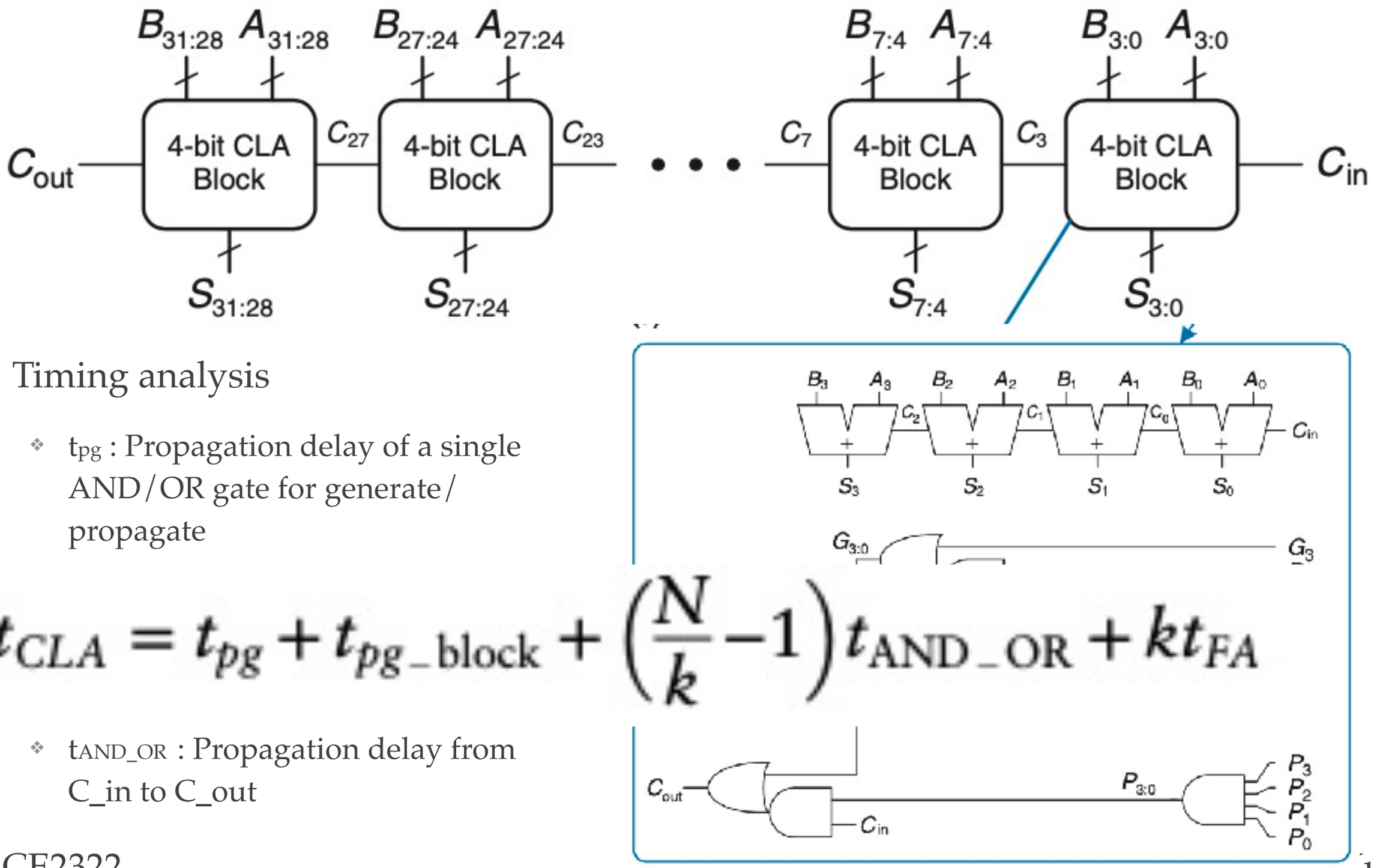


32-bit CLA Example



- Timing analysis
  - t_{pg} : Propagation delay of a single AND/OR gate for generate/ propagate
  - t_{pg_block} : Propagation delay of a single AND/OR gate for generate/ propagate ***of a block***
  - t_{AND_OR} : Propagation delay from C_in to C_out

32-bit CLA Example



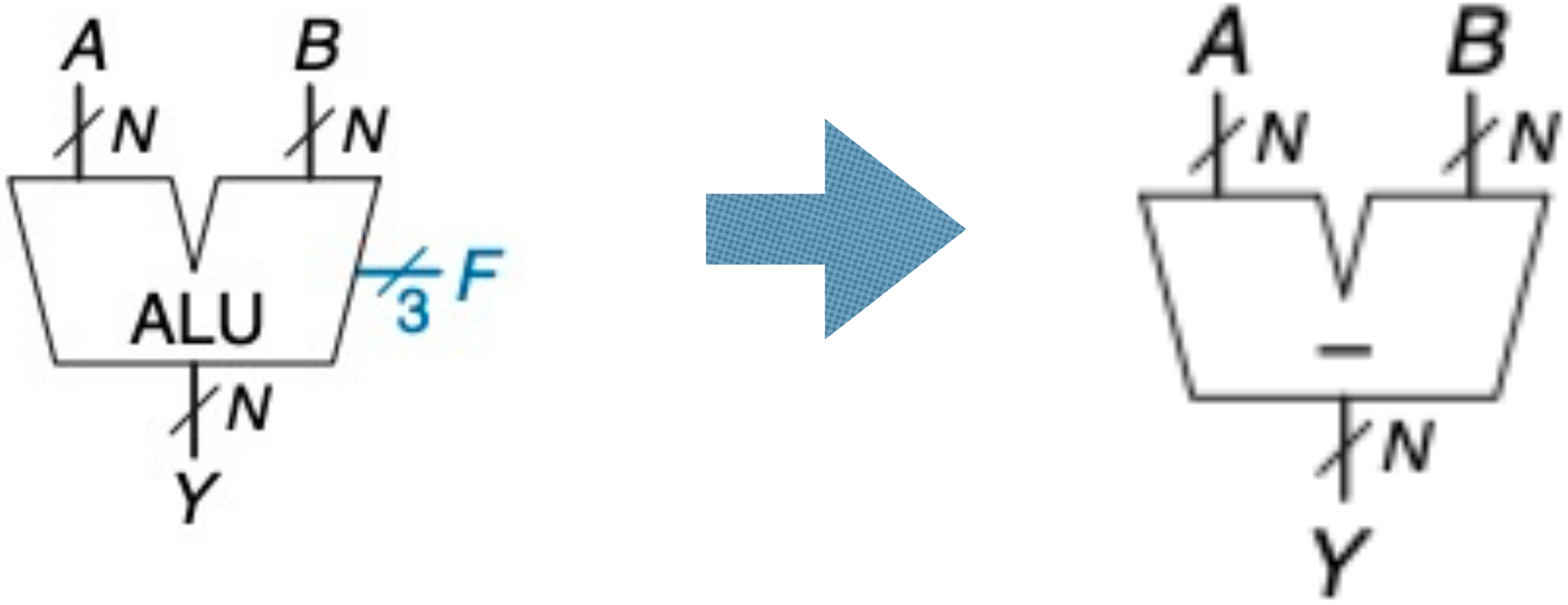
- Timing analysis
  - t_{pg} : Propagation delay of a single AND/OR gate for generate/ propagate

$$t_{CLA} = t_{pg} + t_{pg\_block} + \left(\frac{N}{k} - 1\right) t_{AND\_OR} + k t_{FA}$$

  - t_{AND_OR} : Propagation delay from C_in to C_out

ALU Operations: Subtraction

- Replace the ALU with — **(easy!)**
- **How to implement? Do we need a new circuit? (easy?)**



- **Recall the two's complement number representation**

Two's Complement Number

- $17_{10} = 11111111_2 - 00010001_2 + 1$
  - $= 11101110_2 + 1$
  - $= 11101111_2$
  - $= -17_{10}$
  - The MSB has a negative weight -2^(n-1)
- Hardware mapping?

$$Y = A - B$$

Two's Complement Number

- $17_{10} = 11111111_2 - 00010001_2 + 1$
  - $= 11101110_2 + 1$
  - $= 11101111_2$
  - $= -17_{10}$
  - The MSB has a negative weight -2^(n-1)
- Hardware mapping?

$$Y = A - B = A + \overline{B} + 1$$

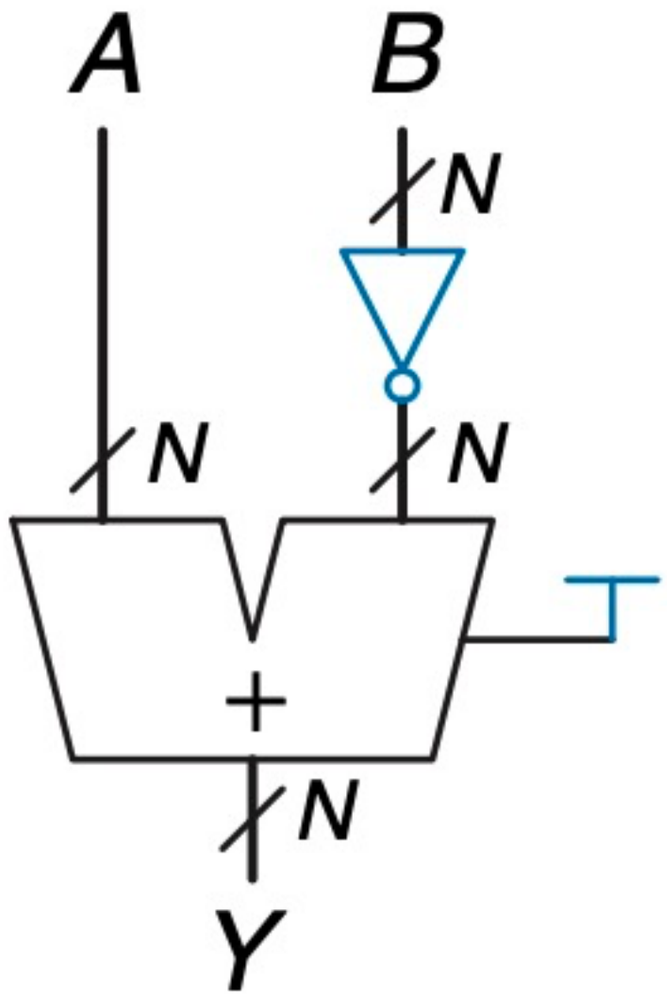
Two's Complement Number

- Hardware mapping?
  - Step1: inverting all bit in B
  - Step2: add "1" to ~B
  - Step3: adding two variables

$$Y = A - B = A + \overline{B} + 1$$

Two's Complement Number

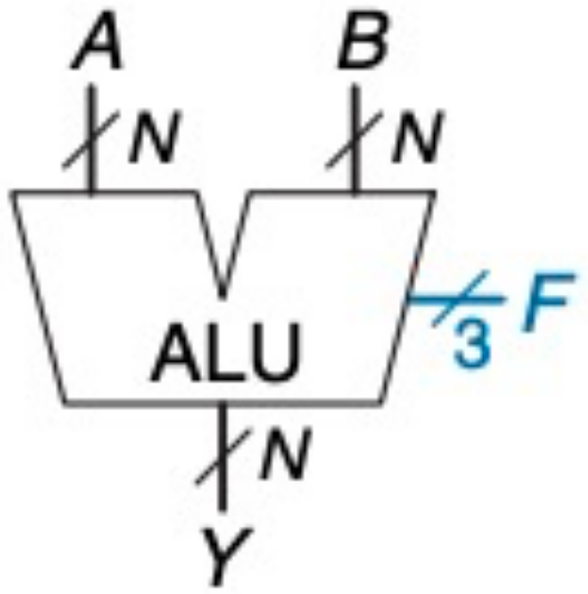
- Hardware mapping?
  - Step1: inverting all bit in B
  - Step2: add "1" to ~B
  - Step3: adding two variables



$$Y = A - B = A + \overline{B} + 1$$

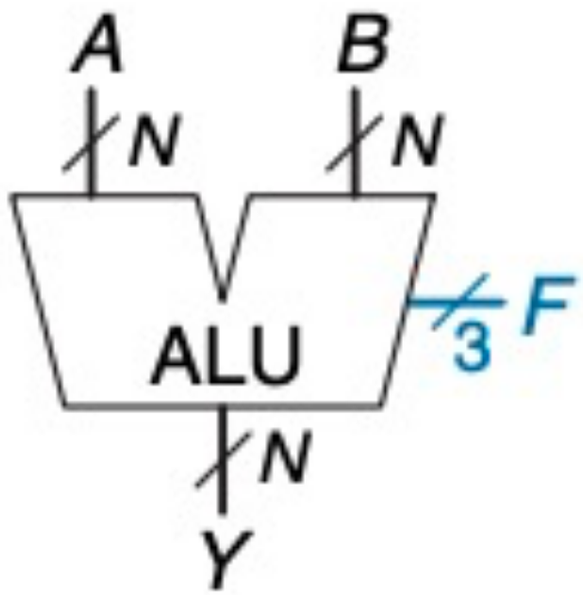
ALU Operations: Comparator

- Determines whether two binary numbers are equal or if one is greater or less than the other
- How to implement?



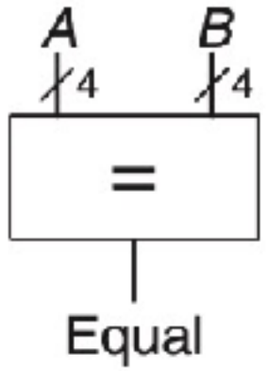
ALU Operations: Comparator

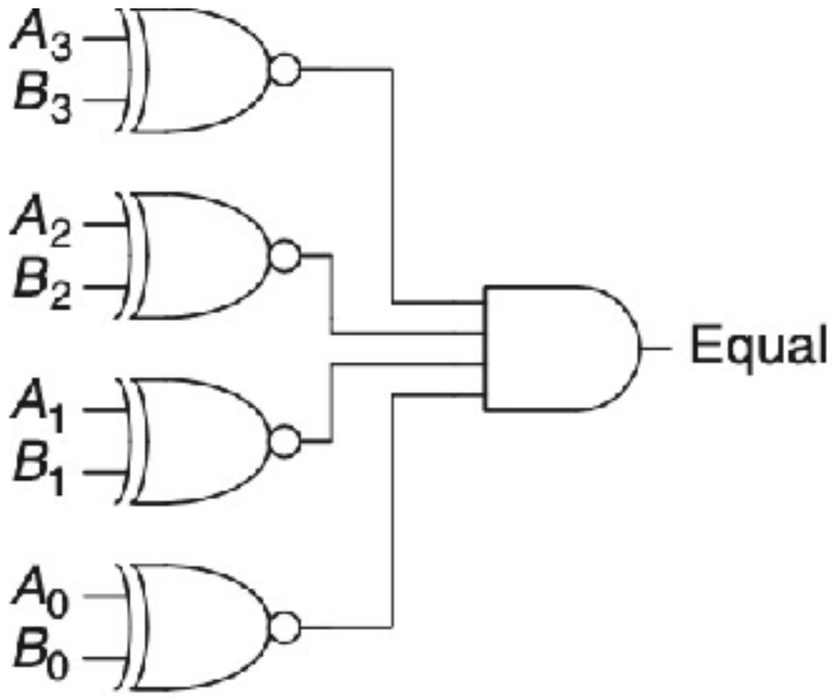
- Determines whether two binary numbers are equal or if one is greater or less than the other
- How to implement?
  - Equal

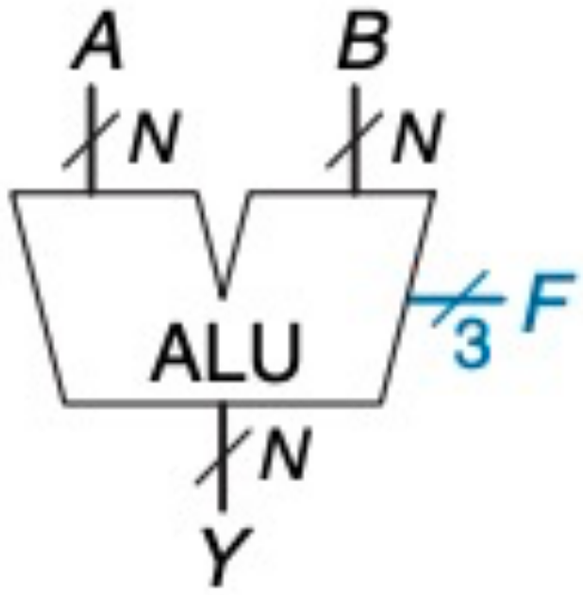


ALU Operations: Comparator

- Determines whether two binary numbers are equal or if one is greater or less than the other
- How to implement?
  - Equal

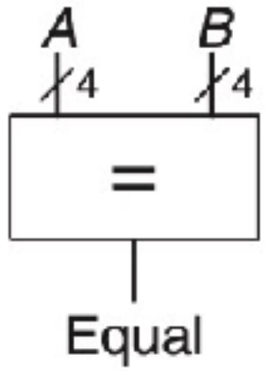


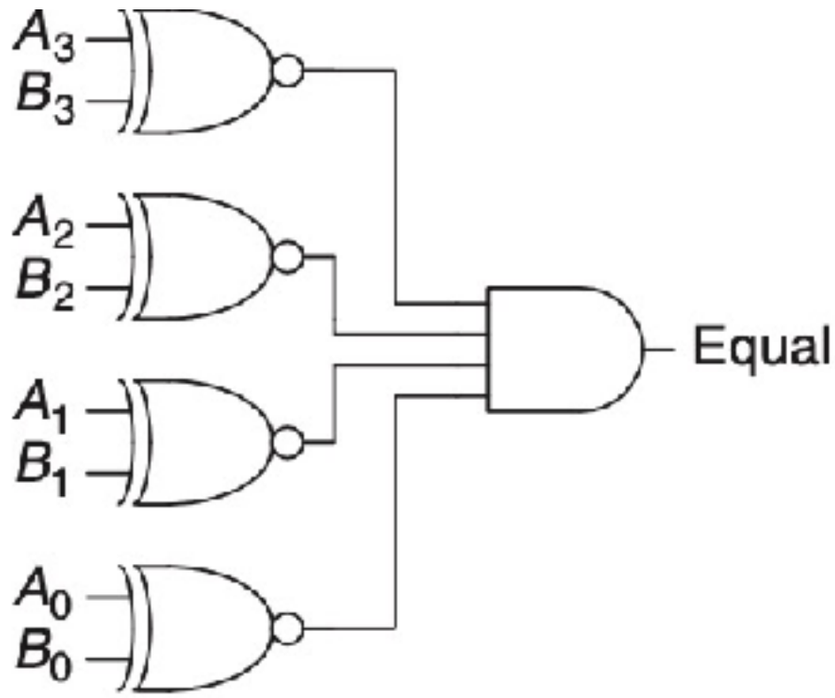


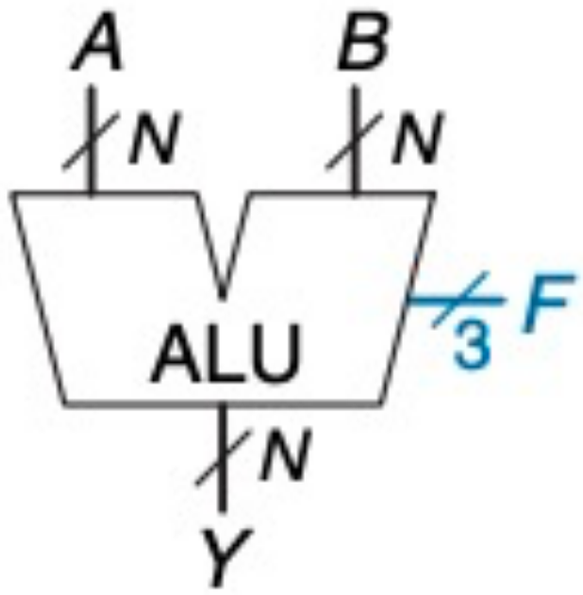


ALU Operations: Comparator

- Determines whether two binary numbers are equal or if one is greater or less than the other
- How to implement?
  - Equal
  - Unequal

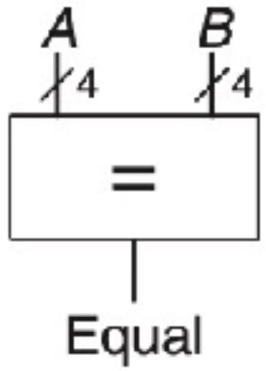


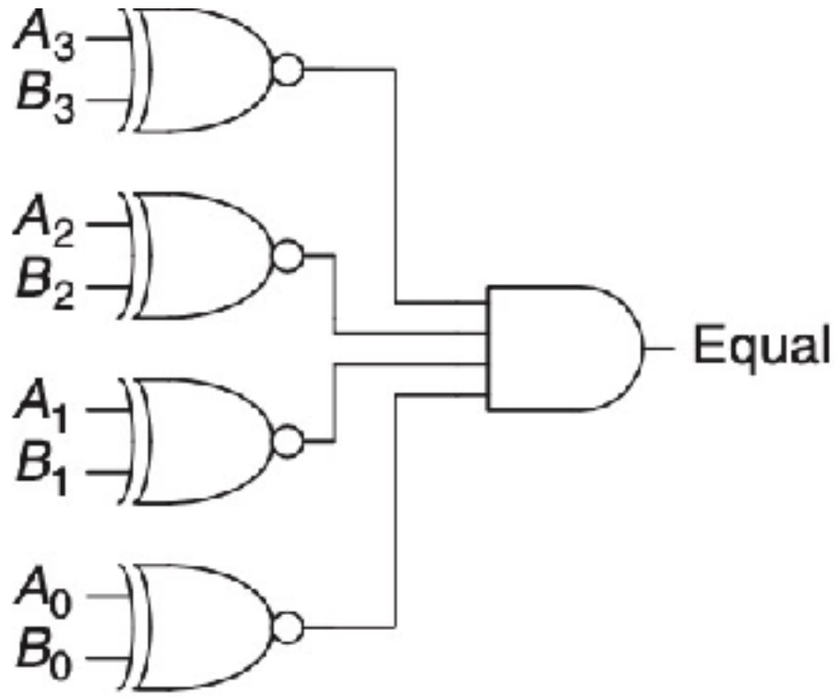


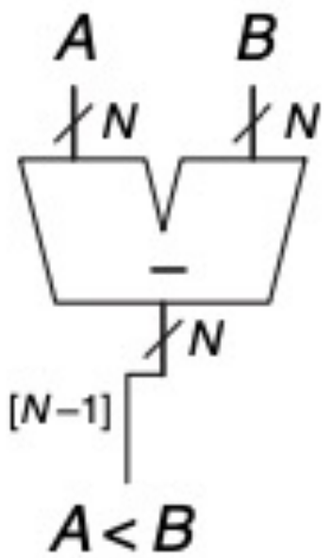


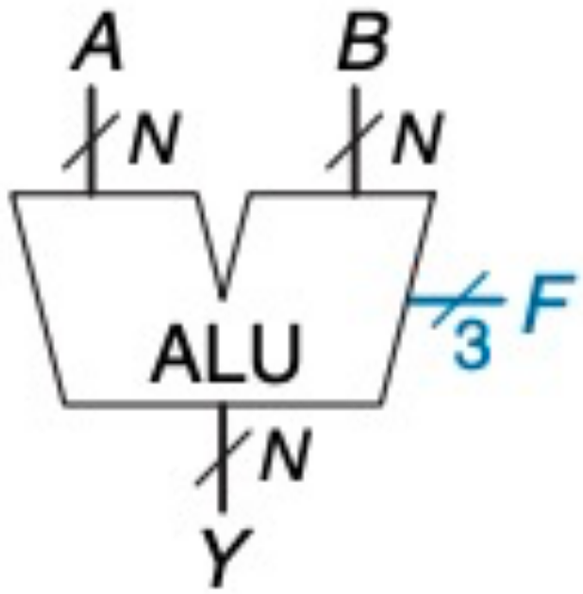
ALU Operations: Comparator

- Determines whether two binary numbers are equal or if one is greater or less than the other
- How to implement?
  - Equal
  - Unequal



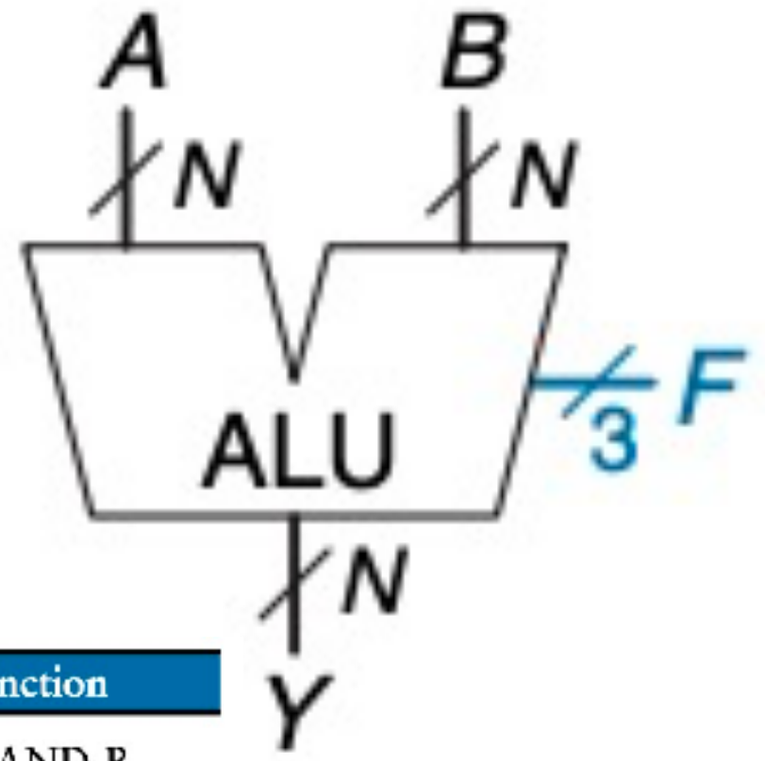






Recall the ALU Schematic

- Abstraction of ALU, three components
  - Two inputs (A, B)
  - One output (Y)
  - Control signal F (3-bit)
- **Most operations done!**



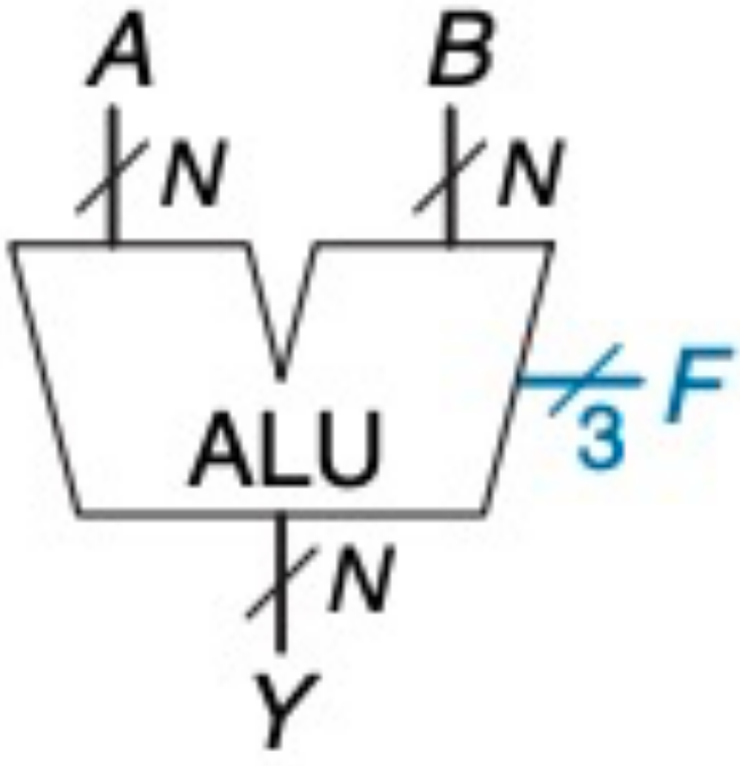
$F_{2:0}$	Function
000	A AND B
001	A OR B
010	A + B
011	not used
100	A AND $\overline{B}$
101	A OR $\overline{B}$
110	A – B
111	SLT

SLT: Set if Less Than

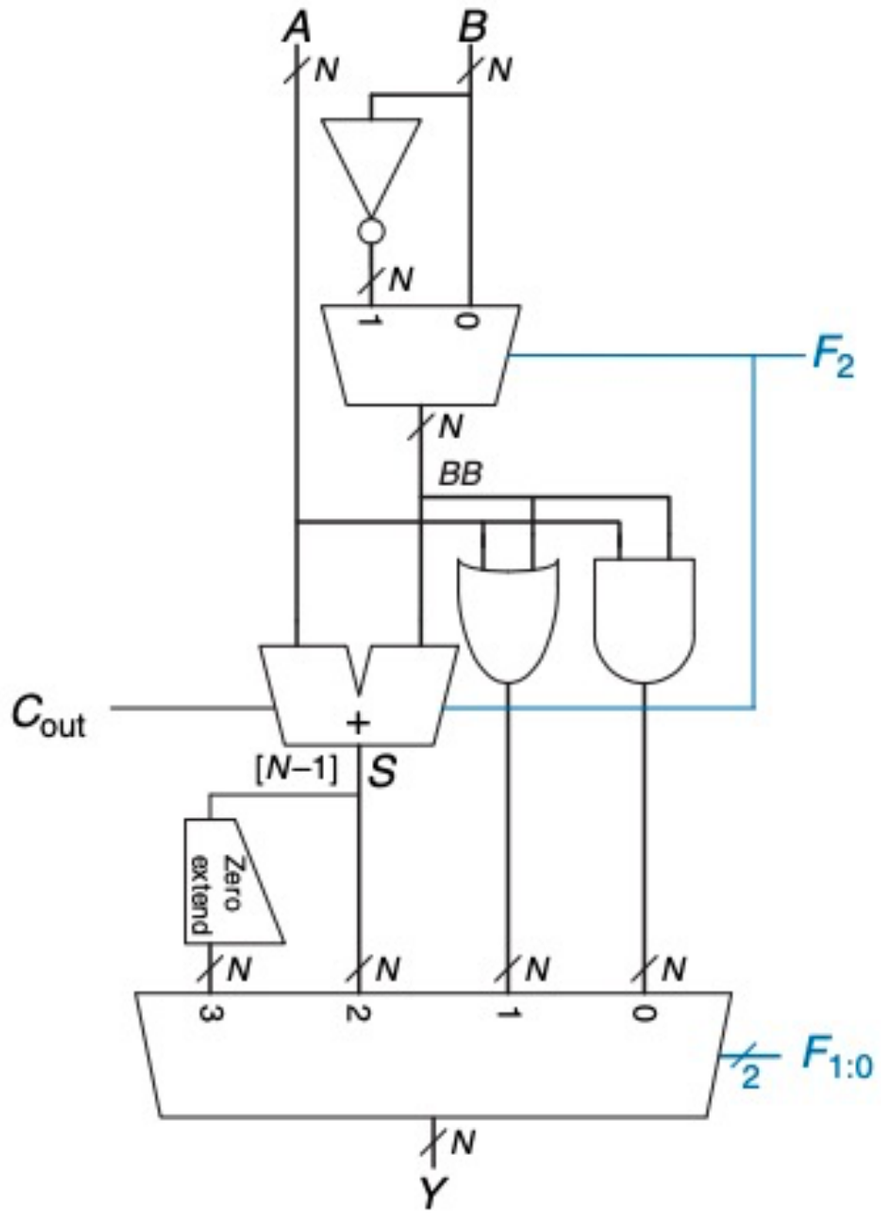
- Y is set to 1 if A is less than B
  - A<B, Y=1
  - Otherwise, Y=0
- Simply computing Y = A - B
  - If negative, then Y=1
  - Otherwise, Y = 0

An ALU Design

- Abstraction of ALU, three components
- **All operations done!**



$F_{2:0}$	Function
000	A AND B
001	A OR B
010	A + B
011	not used
100	A AND $\overline{B}$
101	A OR $\overline{B}$
110	A – B
111	SLT

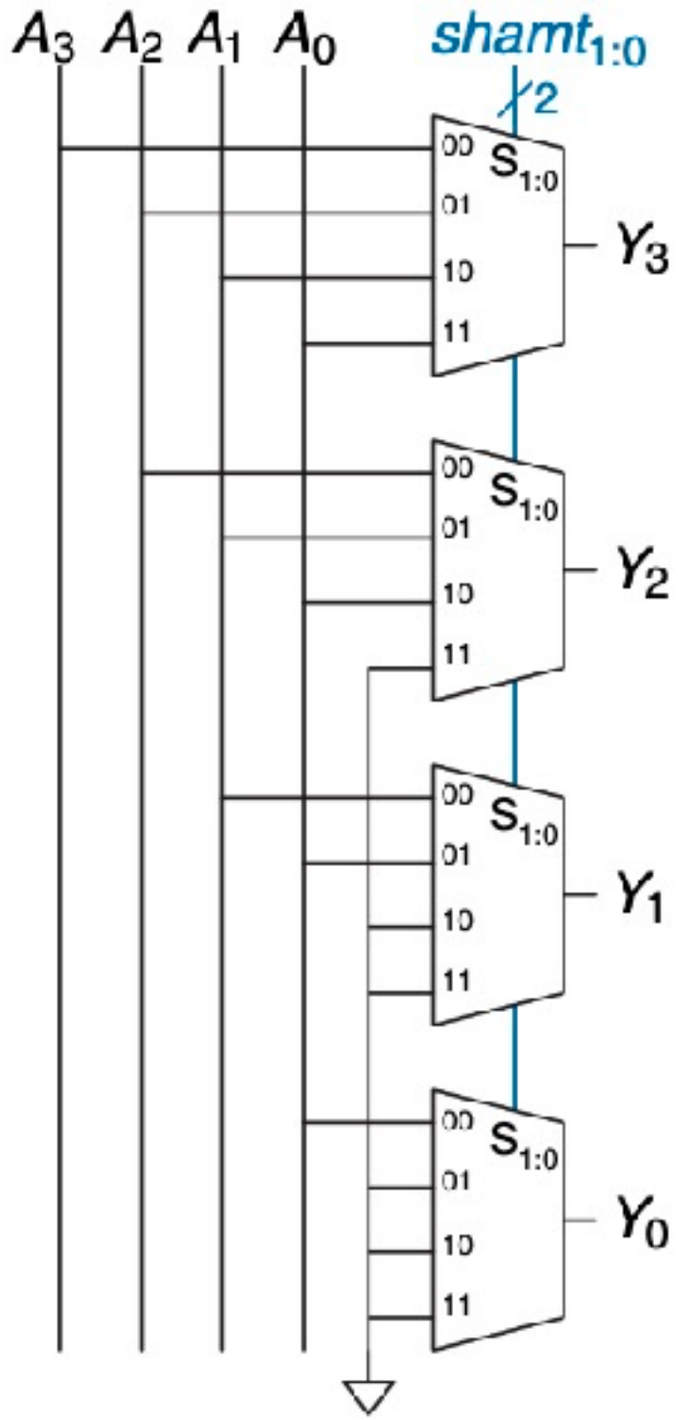


Other Operations: Shifter

- Move bits toward left or right
  - **Logical shifter** — shifts the number to the left (LSL) or right (LSR) and fills empty spots with 0s
    - 11001 LSR 2 =_____; 11001 LSL 2 = _____;
  - Arithmetic shifter
    - Same as a logical shifter
    - Right shifts fills the most significant bits with a copy of the old most significant bit (msb)
    - 11001 ASR 2 = _____; 11001 ASL 2 = _____;

Other Operations: Shifter

- Move bits toward left or right
  - **Logical shifter —** shifts the number to the left (LSL) or right (LSR) and fills empty spots with 0s
    - 11001 LSR 2 =_____; 11001 LSL 2 = _____;
- Arithmetic shifter
  - Same as a logical shifter
  - Right shifts fills the most significant bits with a copy of the old most significant bit (msb)
  - 11001 ASR 2 = _____; 11001 ASL 2 = _____;



Other Operations: Rotator

- Rotator—rotates number in circle such that empty spots are filled with bits shifted off the other end.
  - 11001 ROR 2 = _____; 11001 ROL 2 = _____;