# DIGITAL SYSTEM DESIGN

Lecture 4

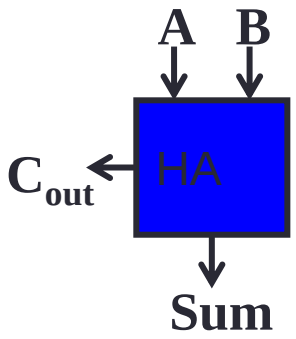
Adders

Waqar Ahmad

Faculty of Computer Science and Engineering

# Adding Two One-bit Operands

- One-bit Half Adder:

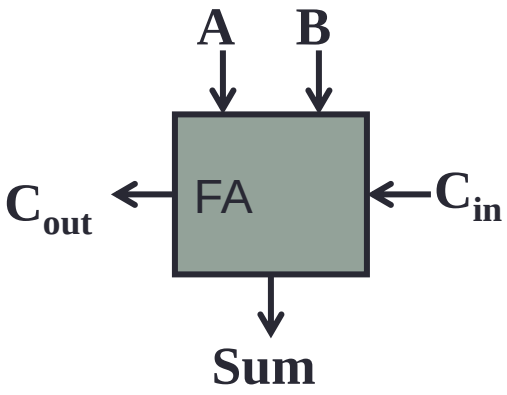



$$Sum = A \oplus B$$

$$Cout = A.B$$

| A | B | Sum | Cout |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 |

- One-bit Full Adder:



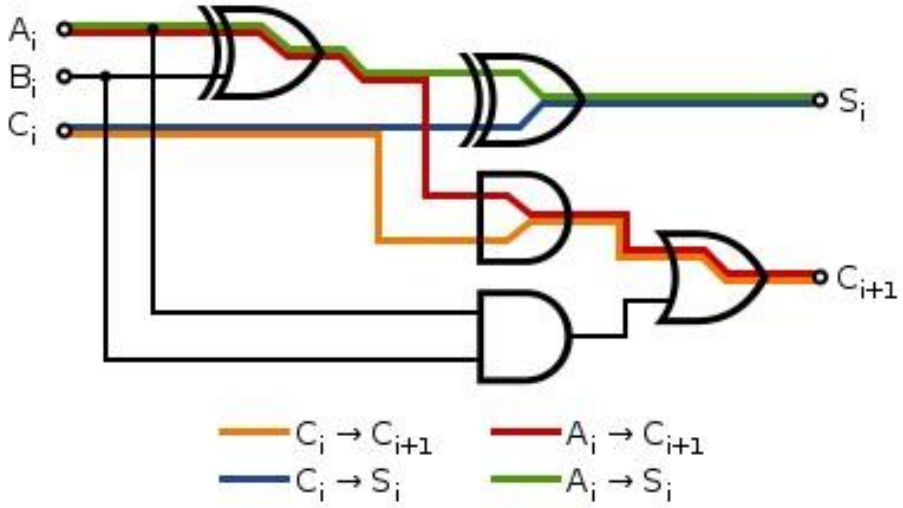

$$Sum = A \oplus B \oplus Cin$$

$$Cout = A.B + B.Cin + A.Cin$$

$$= A.B + (A \oplus B).Cin$$

| $C_{in}$ | A | B | Sum | Cout |
|---|---|---|---|---|
| 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 |

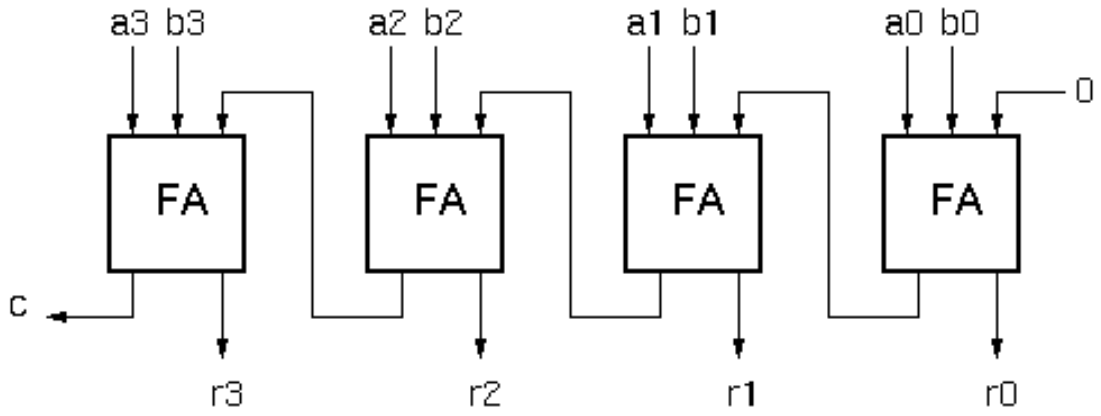
# Propagation Delay in Full Adders




- From $A_i$ or $B_i$ to $C_{i+1}$: 4 gate-delays (XOR → AND → OR)
- From $A_i$ or $B_i$ to $S_i$: 4 gate-delays (XOR → XOR)
- From $C_i$ to $C_{i+1}$: 2 gate-delays (AND → OR)
- From $C_i$ to $S_i$: 2 gate-delays (XOR)

# Ripple Carry Adder

- Constructed by cascading full adders (FA) blocks in series
- A 4-bit carry ripple adder is shown below



# Delay in Ripple Carry Adders

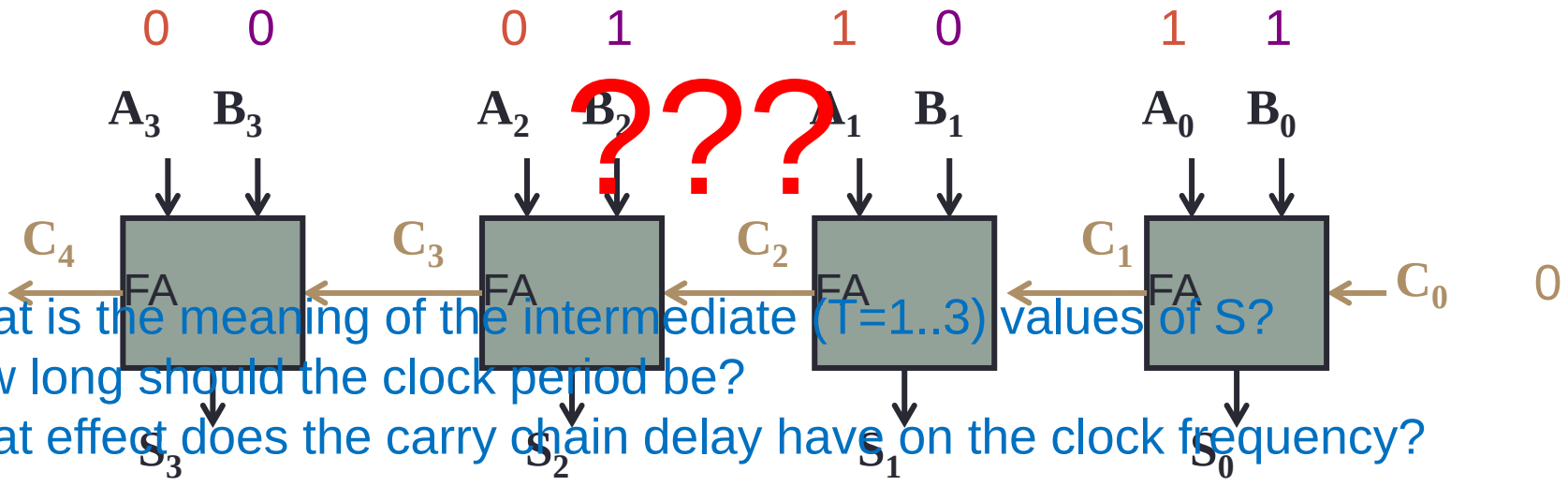
- Recall: For a full-adder
  - From $A_i$ or $B_i$ to $C_{i+1}$: 4 gate-delays (XOR → AND → OR)
  - From $A_i$ or $B_i$ to $S_i$: 4 gate-delays (XOR → XOR)
  - From $C_i$ to $C_{i+1}$: 2 gate-delays (AND → OR)
  - From $C_i$ to $S_i$: 2 gate-delays (XOR)

- In ripple carry adder, the carry-out of one FA is the carry-in of next FA, the worst-case propagation delay is then:
  - 4 gate-delays from generating the first carry signal ($A_0/B_0 \rightarrow C_1$).
  - 2 gate-delays per intermediate stage ($C_i \rightarrow C_{i+1}$).
  - 2 gate-delays at the last stage to produce both the sum and carry-out outputs ($C_{n-1} \rightarrow C_n$ and $S_{n-1}$).

# 4-bit Ripple Carry Addition: Example

A=0011

B=0101




What is the meaning of the intermediate (T=1..3) values of S?

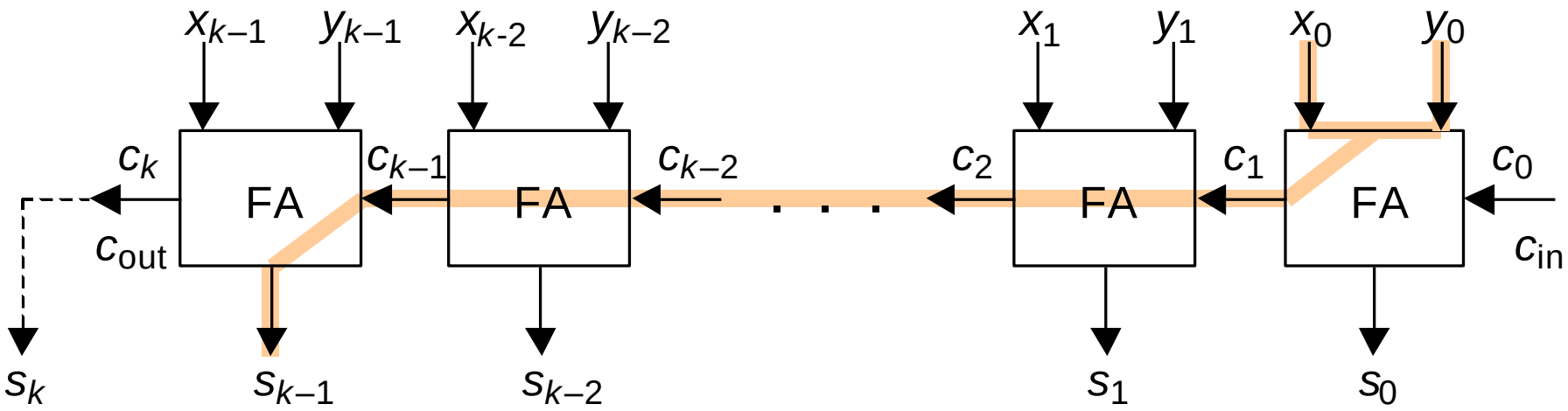
How long should the clock period be?

What effect does the carry chain delay have on the clock frequency?

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| T=0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | S=0000 |
| T=1 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | S=0110 |
| T=2 | 0 | 0 | 0 | 1 | 1 | 0 | 1 | 0 | S=0100 |
| T=3 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | S=0000 |
| T=4 | 0 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | S=1000 |

# Critical Path Through a Ripple-Carry Adder

$$T_{\text{ripple-add}} = T_{\text{FA}}(X \rightarrow s) + (k-2) \times T_{\text{FA}}(c_{\text{in}} \rightarrow c_{\text{out}}) + T_{\text{FA}}(c_{\text{in}} \rightarrow s)$$



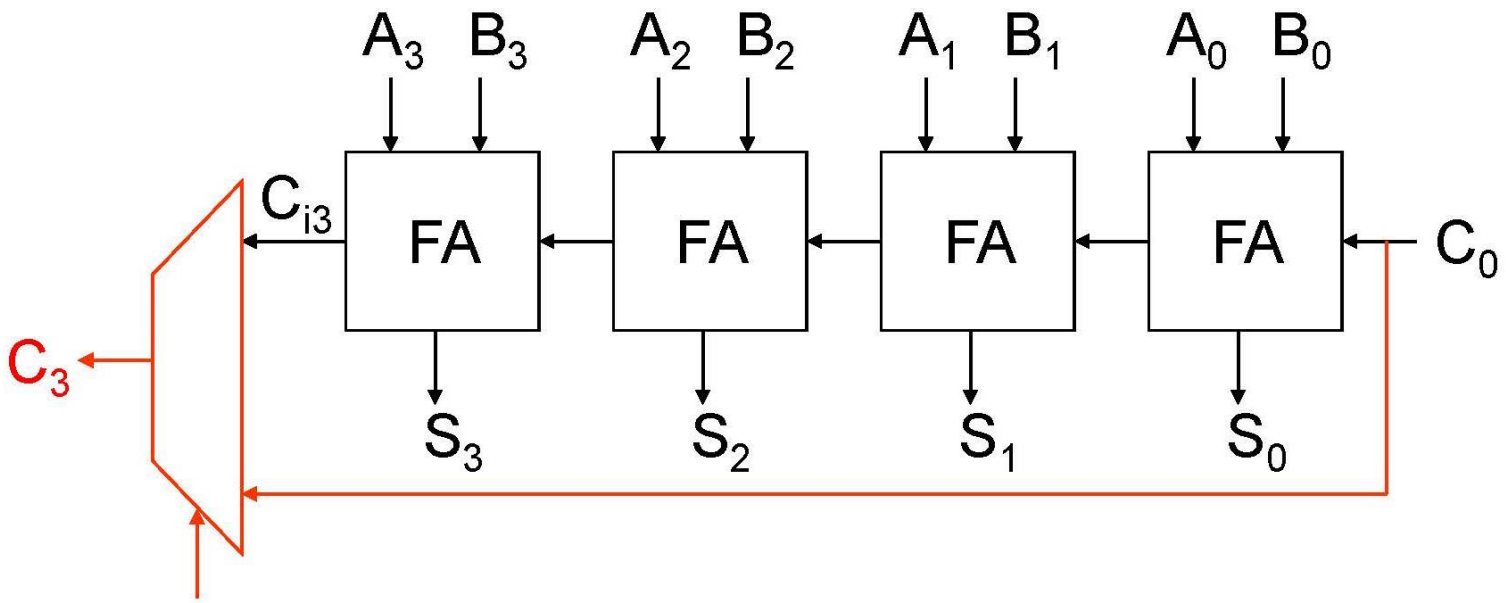

Critical path in a *k*-bit ripple-carry adder.

# Pros and Cons

- Advantages
  - Compact layout giving smaller chip area.
  - Lower power consumption

- Disadvantages
  - Delay increases linearly with increasing bit-length
    - At least 2 gate delays for every additional bit

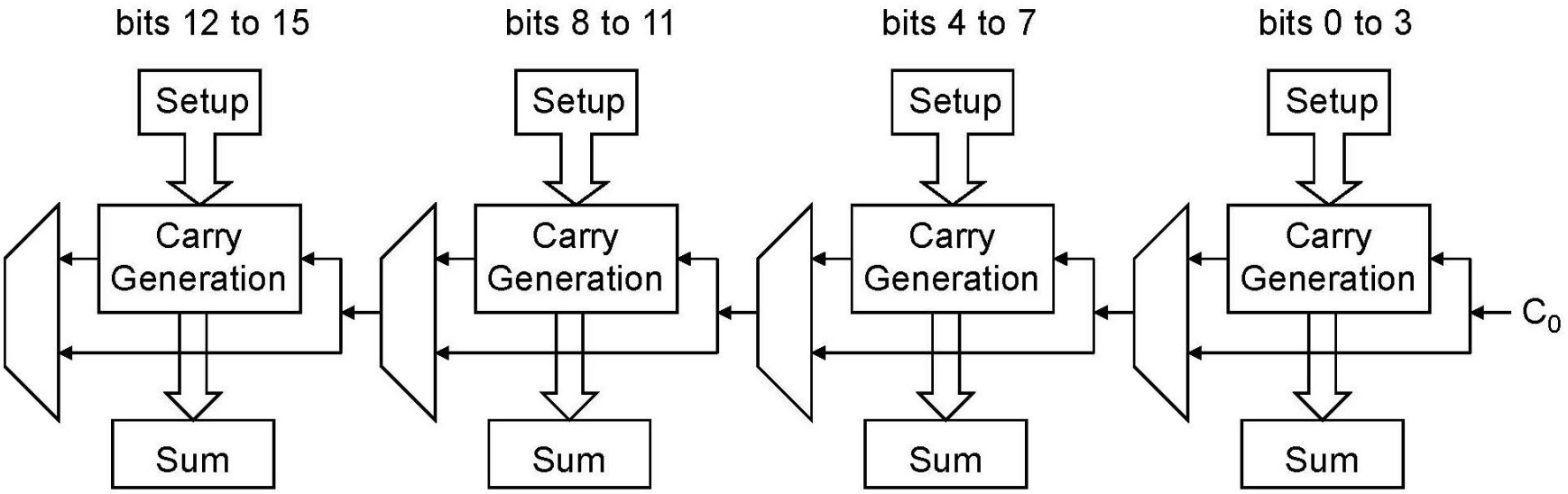
# Carry Skip (Bypass) Adder



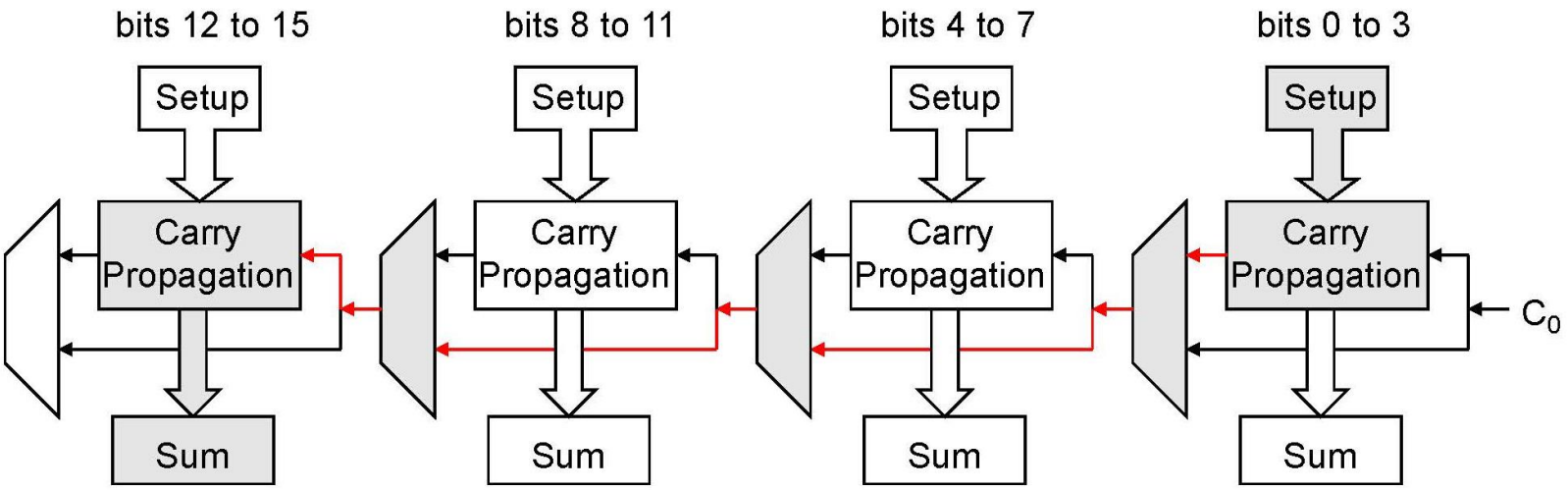

BP = $P_0 P_1 P_2 P_3$ (Block Propagate)

if ($P_0 P_1 P_2 P_3$) $C_3 = C_0$
else $C_3 = C_{i3}$

# 16-bit Carry Skip Adder using 4-bit blocks



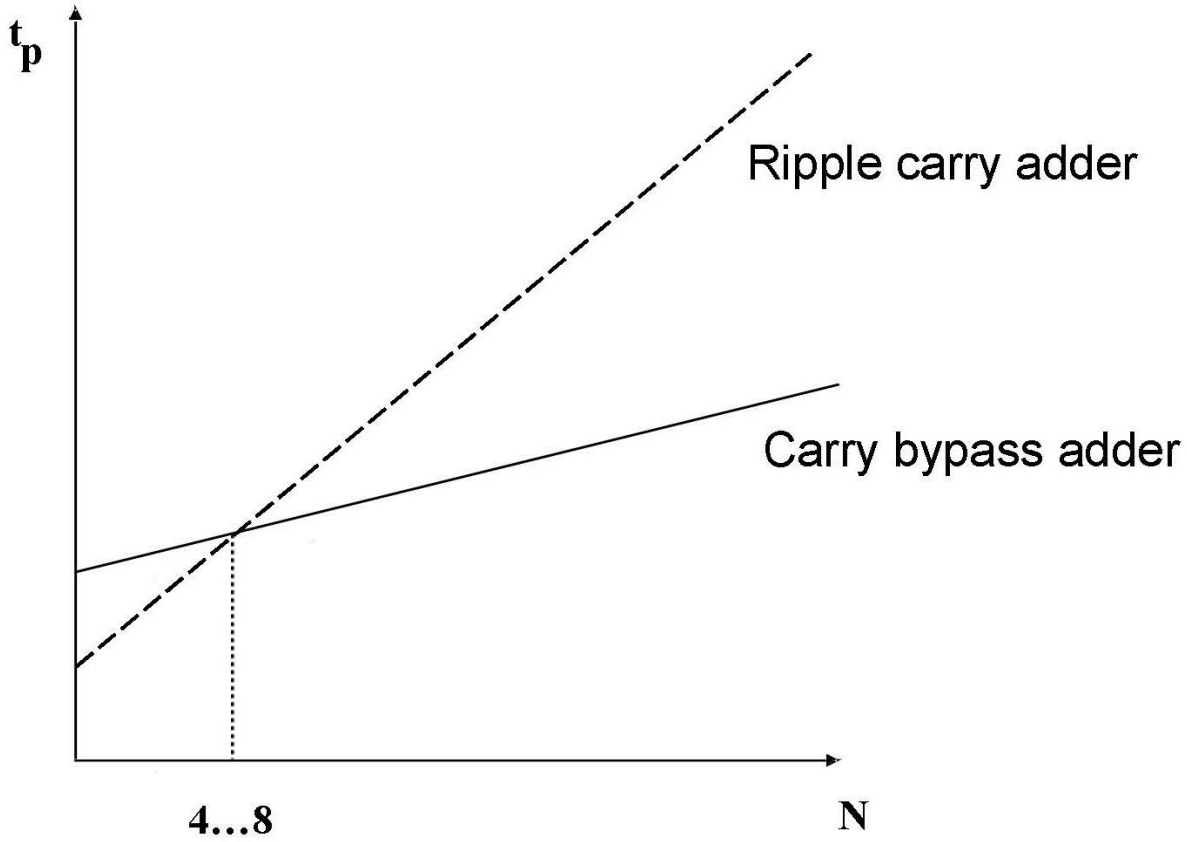
# Carry Skip Adder – Delay




worst-case delay = carry from bit 0 to bit 15 = carry generated in bit 0, ripples through bits 1, 2, and 3, skips the middle two groups (M is the group size in bits), ripples in the last group from bit 12 to bit 15

$$t_{add} = t_{setup} + M\, t_{carry} + ((N/M) - 1)\, t_{skip} + (M-1)\, t_{carry} + t_{sum}$$

# Delay – Carry Skip Adder vs Ripple Carry Adder



# Carry Select Adder

- Do we have to wait for the previous-carry to show up to compute next sum and carry?
  - We do have to know the carry to get the right answer.
  - But, it can only take on two values

- Idea: Compute both possible values and select correct result when we know the actual input

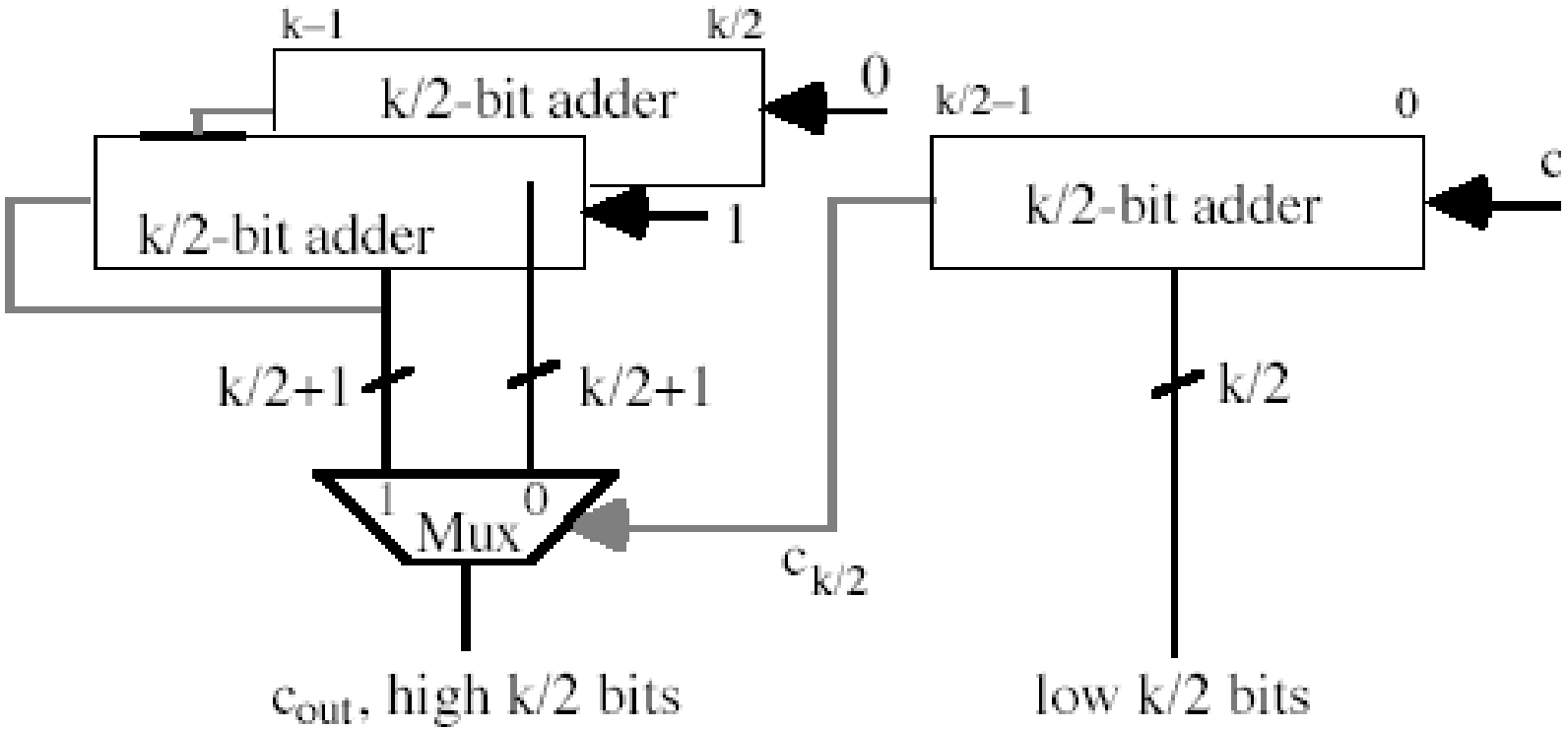
- Principle:
  - The adder is partitioned into M groups
  - Two values of sum with cin (1 and 0) are pre computed for each adder group
  - Actual sum is selected using a 2-to-1 MUX by the carry of the previous group

  - Compute possible results in parallel
  - Select when actual carry-in available
  - Requires internal carry for blocks, e.g. ripple effected by block sizing

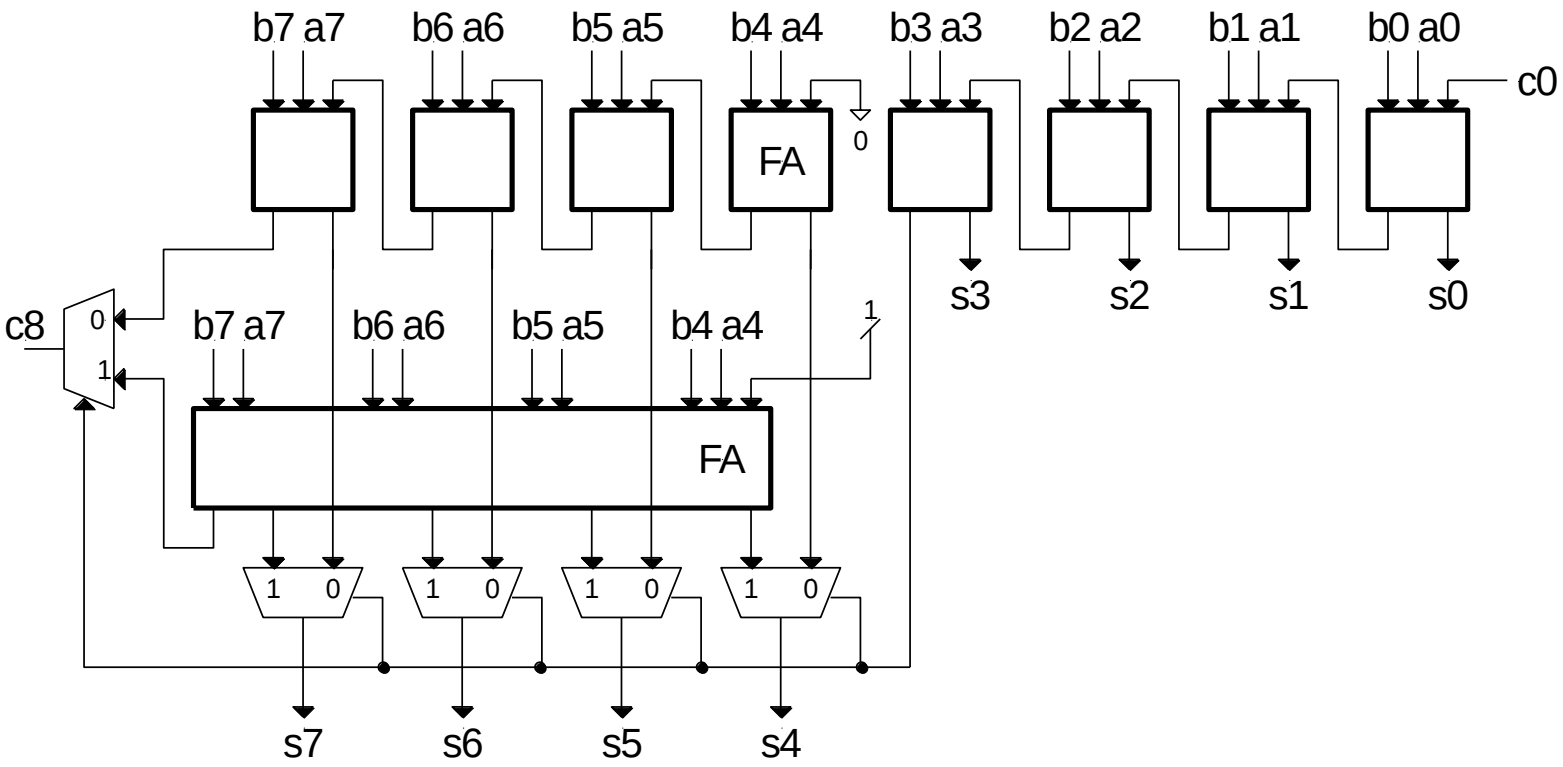
# Ways of Implementation

- Levels
  - 1 – Level
  - N – Level
- Grouping
  - Uniform Group
  - Non-Uniform Group

# One-level k-bit Carry-Select Adder



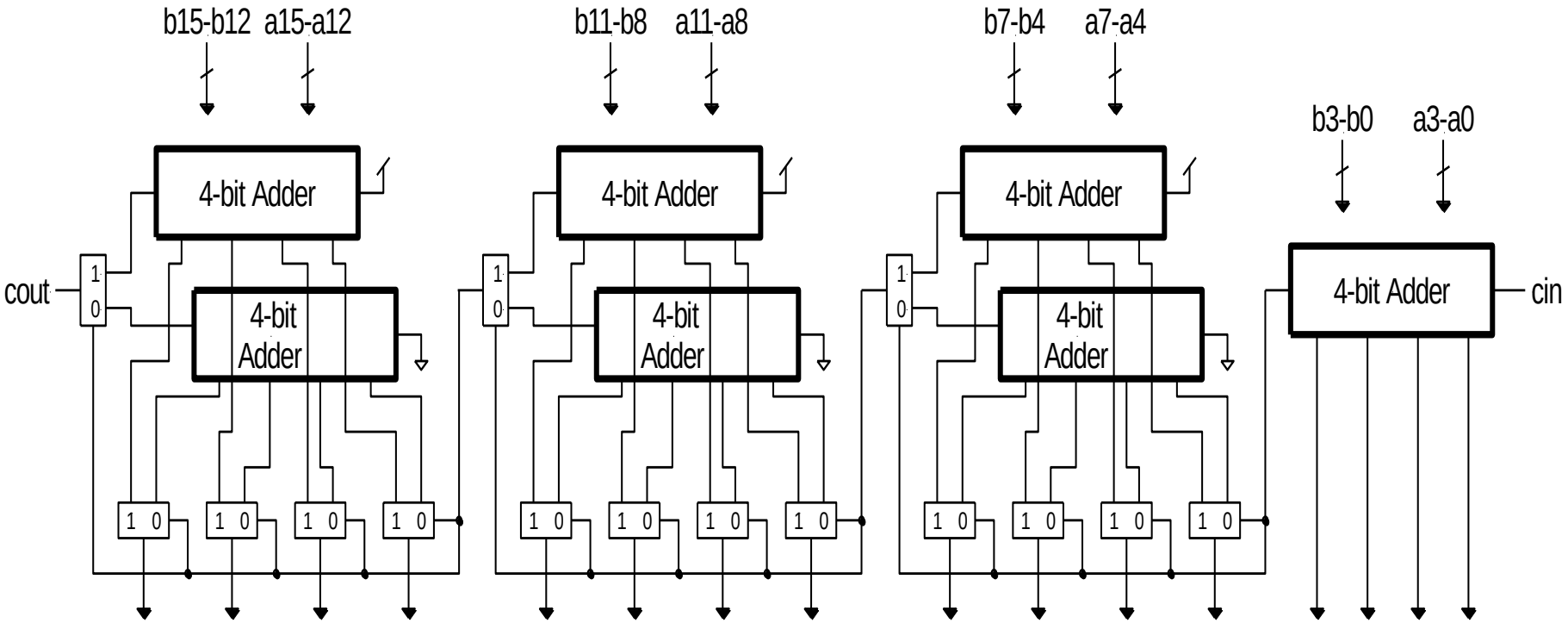
# 1-Level 8-bit Uniform Carry Select Adder



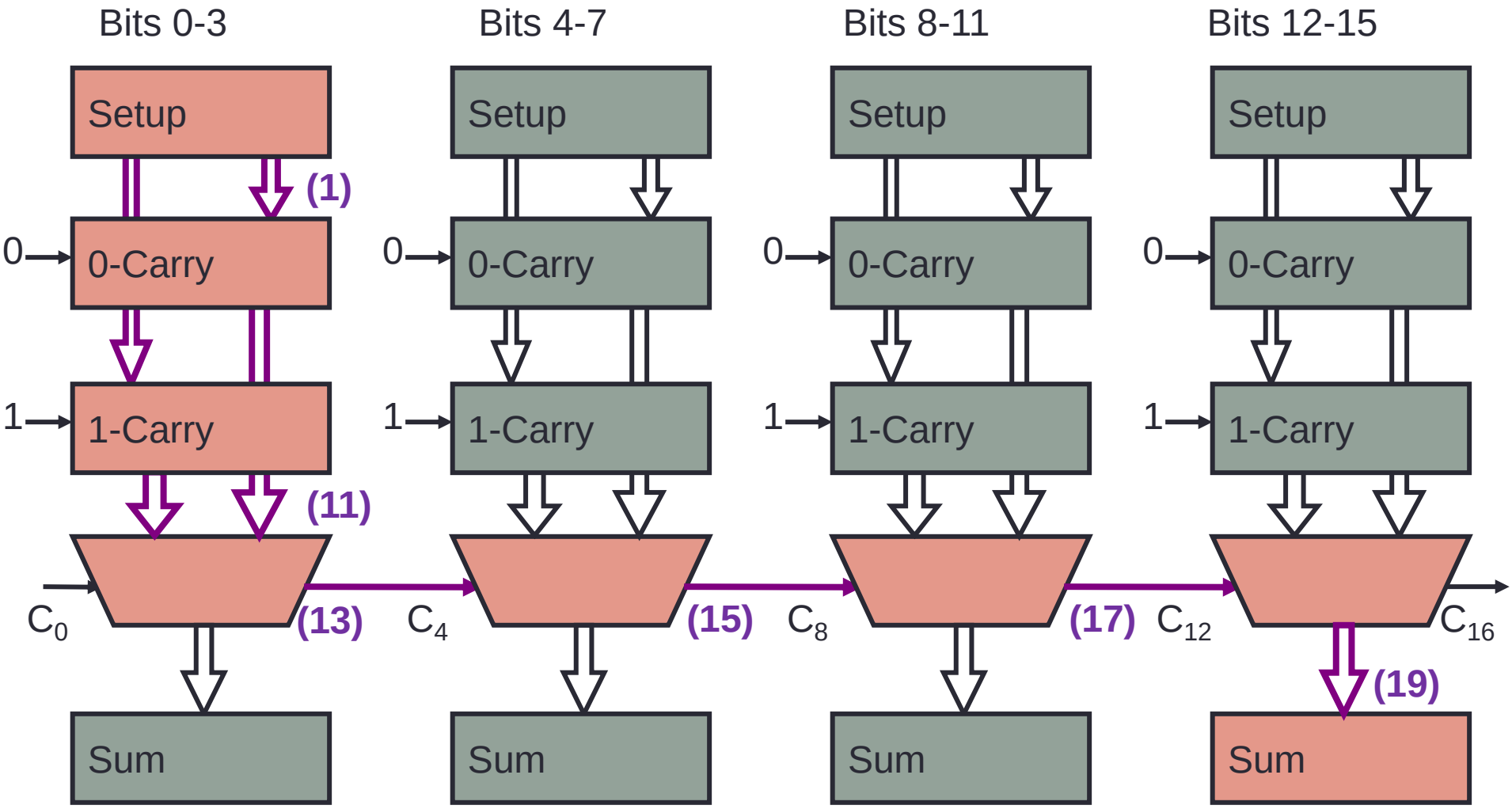

$$T = T_{4bit\_ripple\_adder} + T_{MUX}$$

# 1-Level 16-bit Uniform Carry Select Adder

- Extending Carry-select to multiple blocks
- What is the optimal # of blocks and # of bits/block?
  - If # blocks too large delay dominated by total mux delay
  - If # blocks too small delay dominated by adder delay



# Uniform Carry Select Adder: Timing



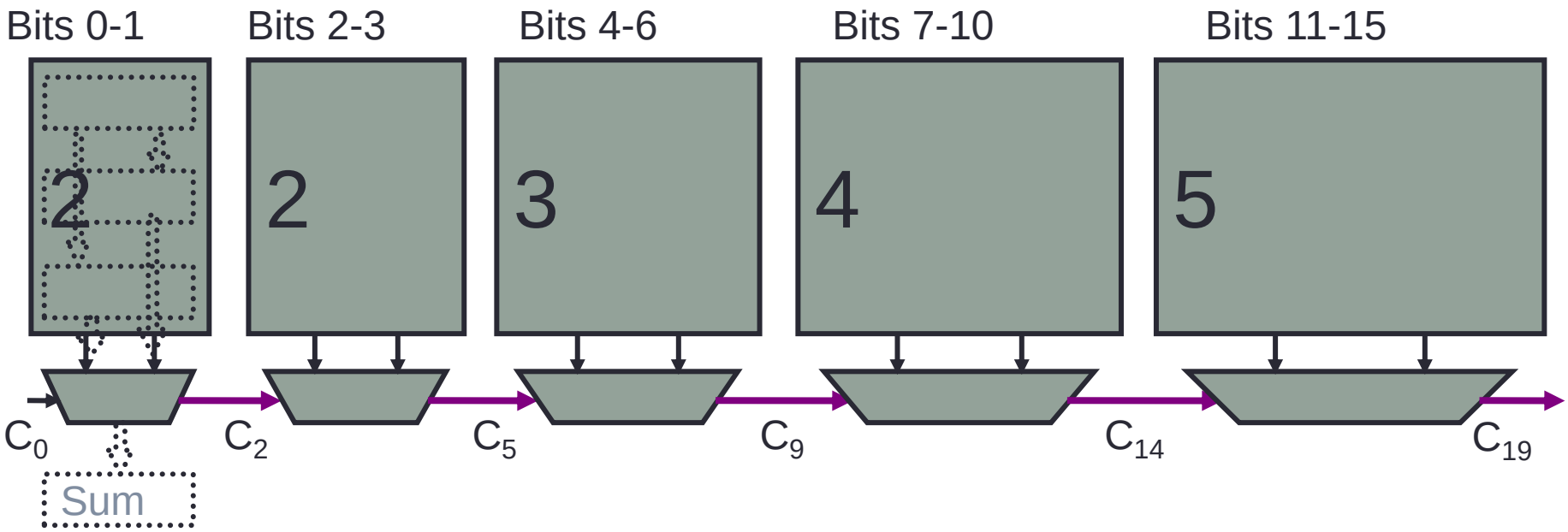

Delay = 1+10 + 2 + 2 + 2 + 2 = 19 (16 bits)

# Non-Uniform Carry Select Adder

- Later stages have to wait for the multiplexers in the earlier stages

- Why not give them bigger chunks of data to compute?
  - Balances the delay paths
  - Sub-linear delay (we will see why)

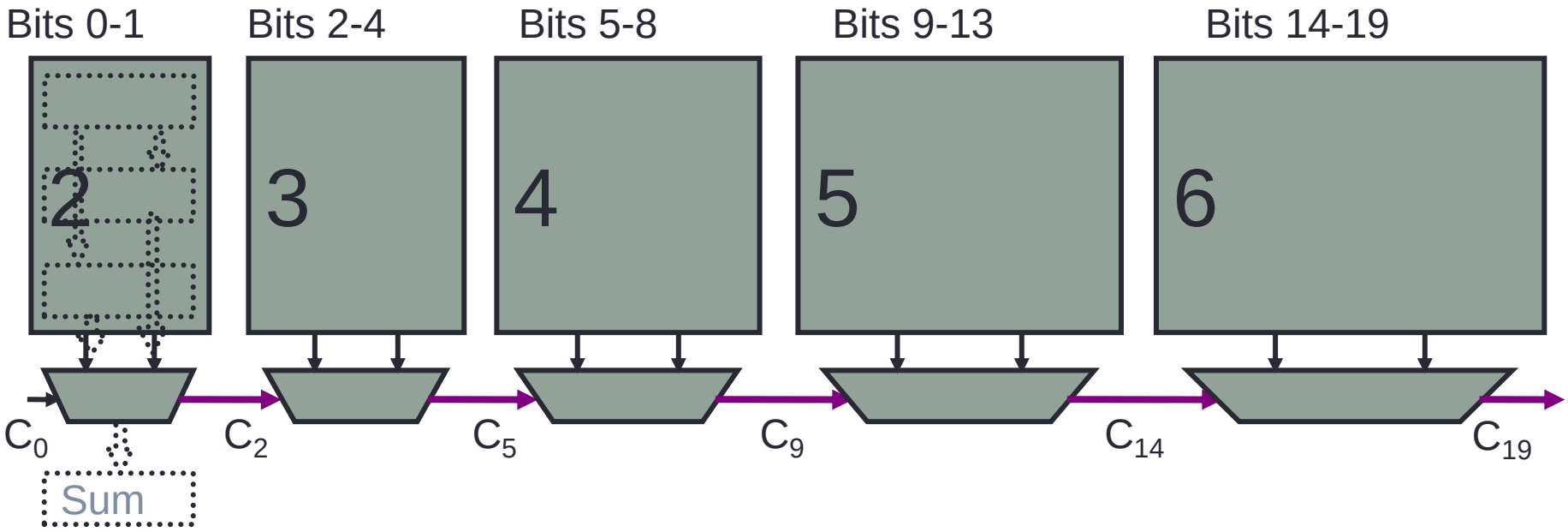
# Non-Uniform Carry Select Adder: The Structure




Delay = 1+6+2+2+2+2+2 = 17 (16 bits)

The numbers in the block indicate number of bits processed in that block.

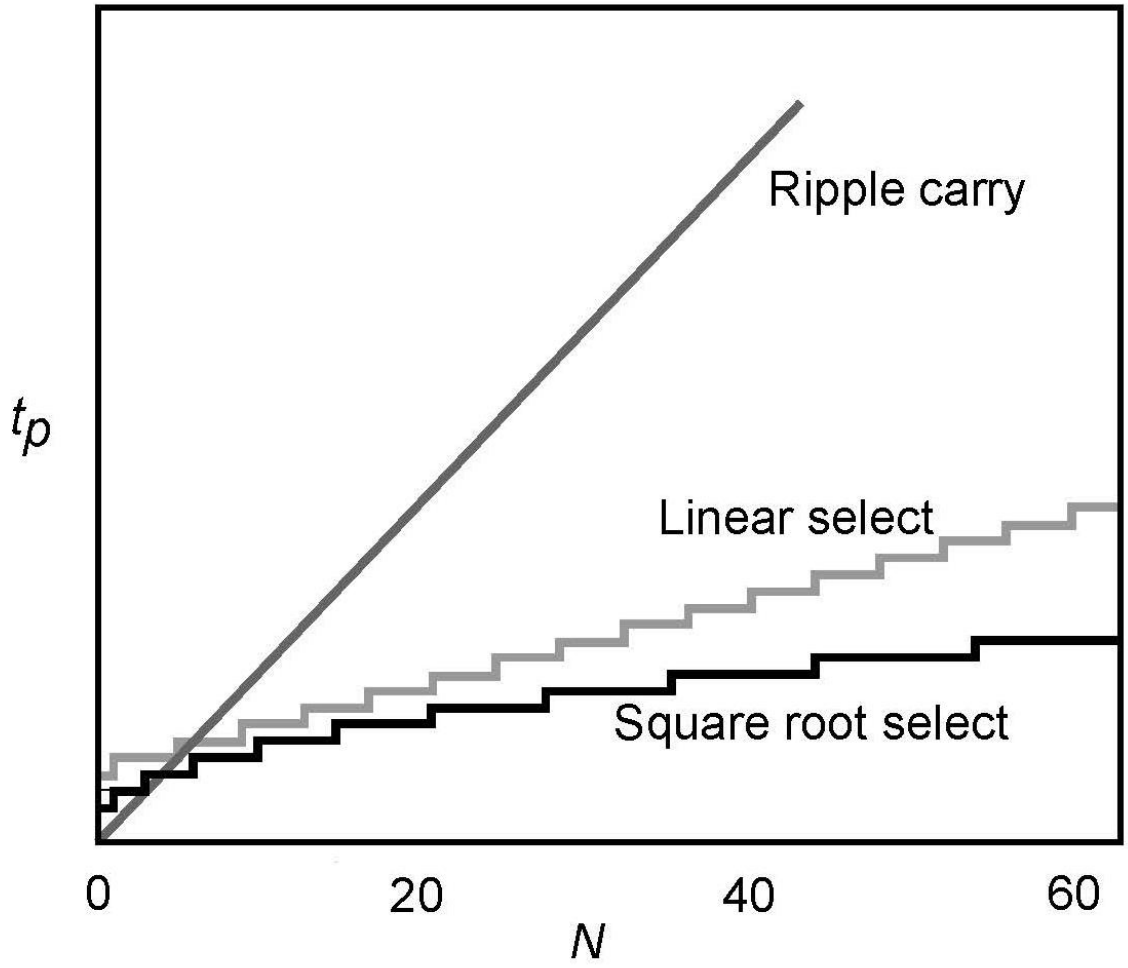
# Non-Uniform Carry Select Adder: 20-bits



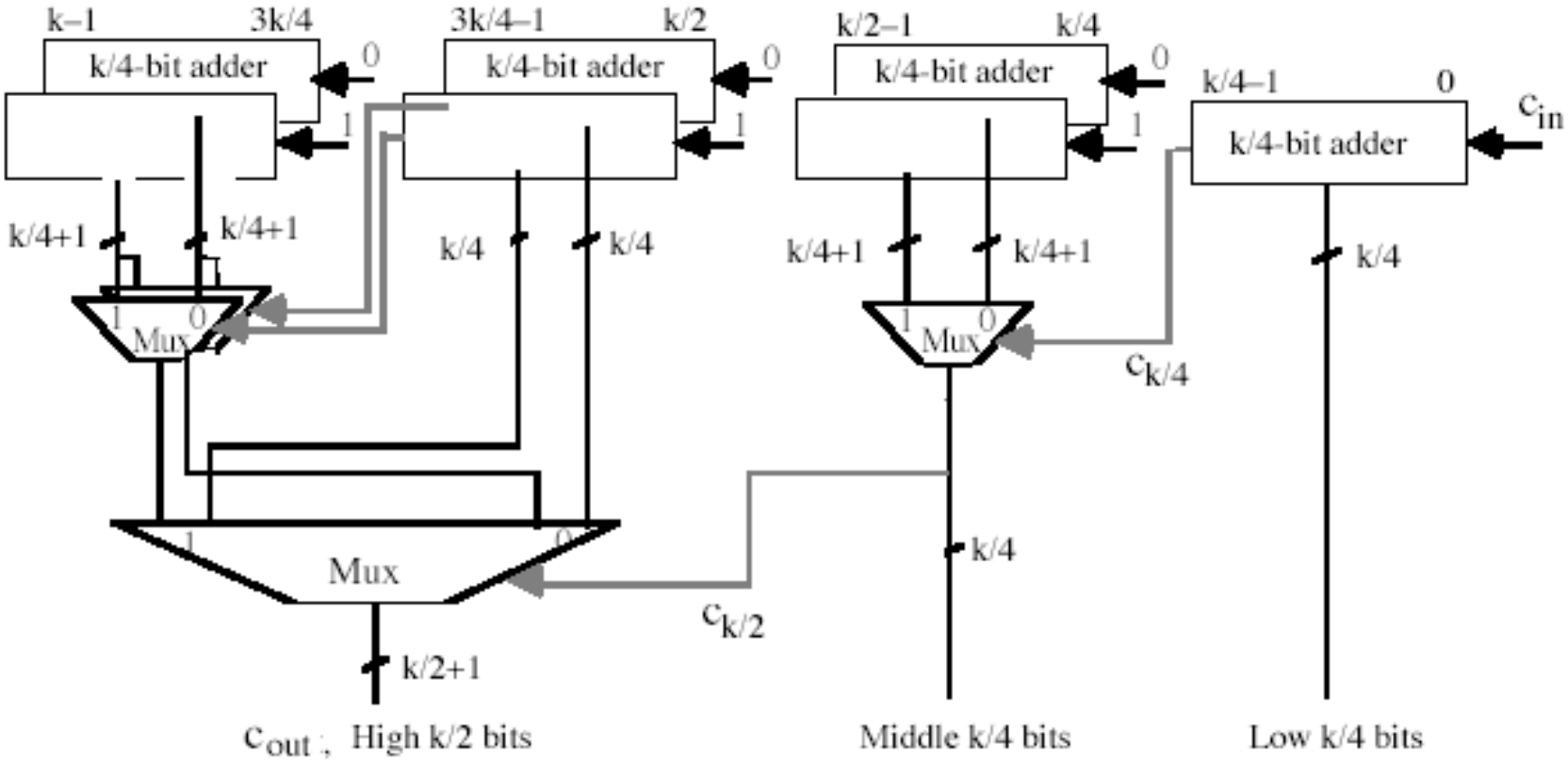

Delay = 1+6+2+2+2+2+2 = 17 (20 bits)

Note how the numbers of bits pre-computed by each stage progressively increase.

# Delay – Carry Select Adder vs Ripple Carry Adder



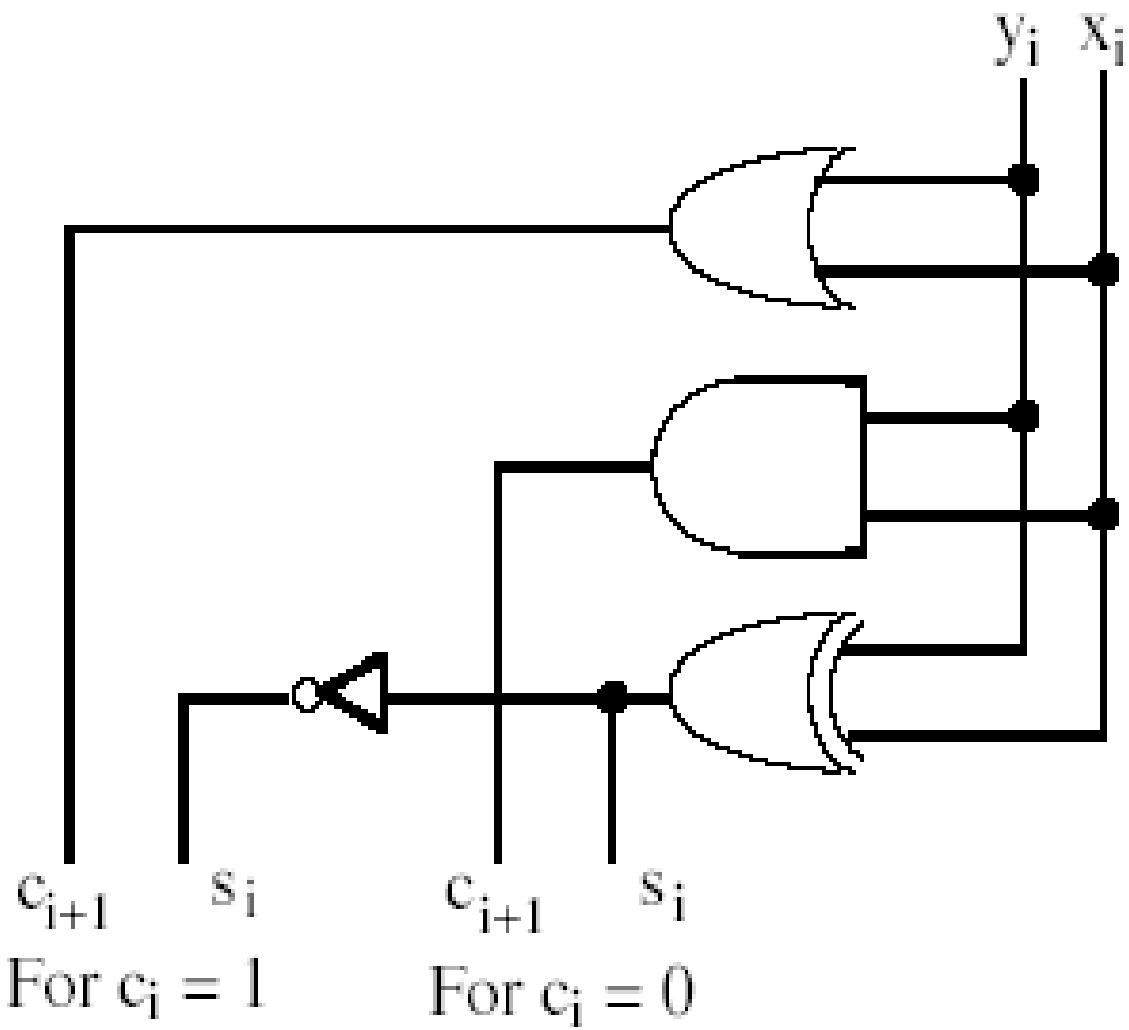
# Two-level k-bit Carry Select Adder



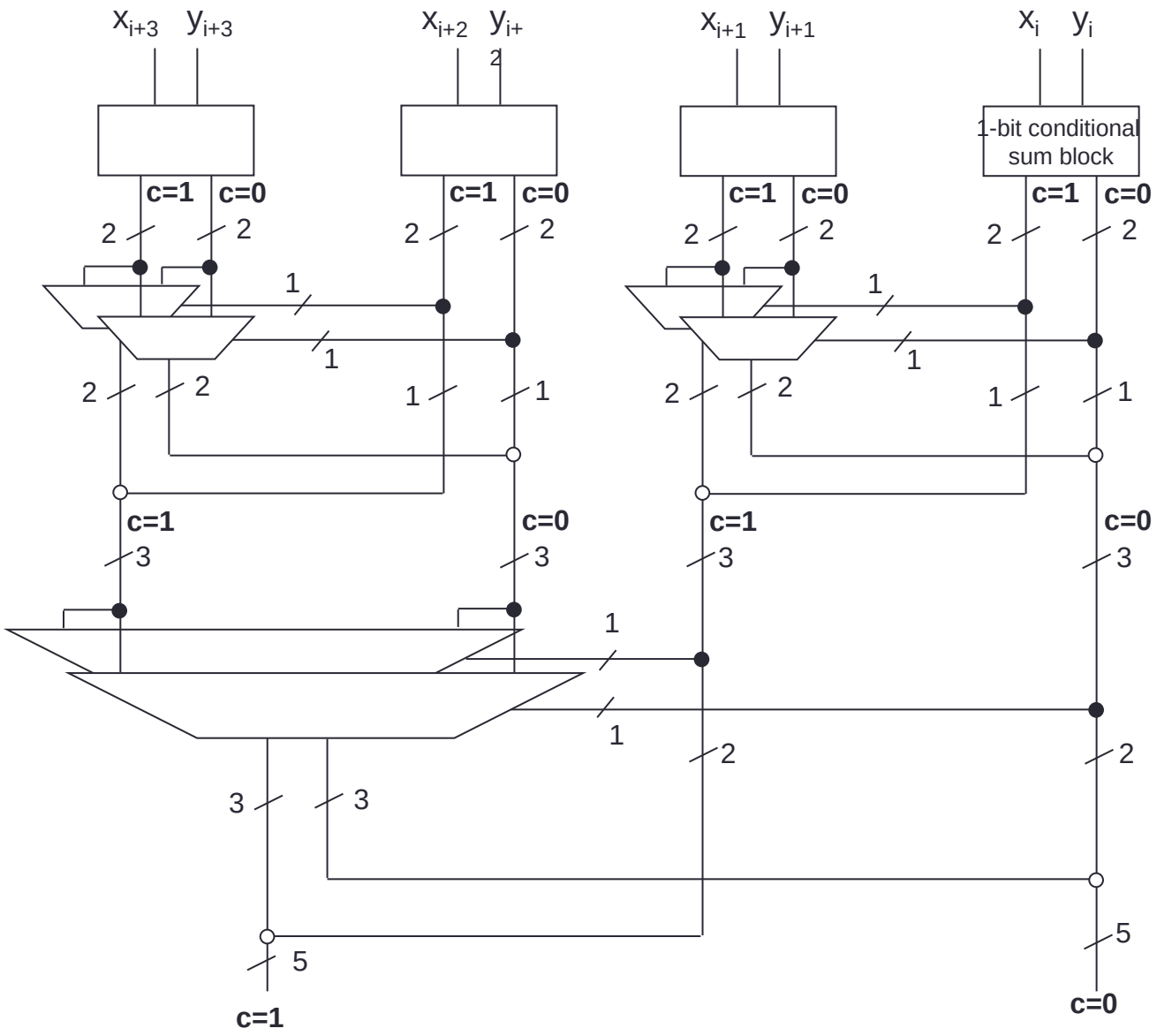
# Conditional Sum Adder

- Extension of carry-select adder
- Carry select adder
  - One-level using k/2-bit adders
  - Two-level using k/4-bit adders
  - Three-level using k/8-bit adders
  - Etc.
- Assuming k is a power of two, eventually have an extreme where there are $\log_2 k$-levels using 1-bit adders
  - This is a conditional sum adder

# Top-Level Block for One Bit Position



# Three Levels of a Conditional Sum Adder

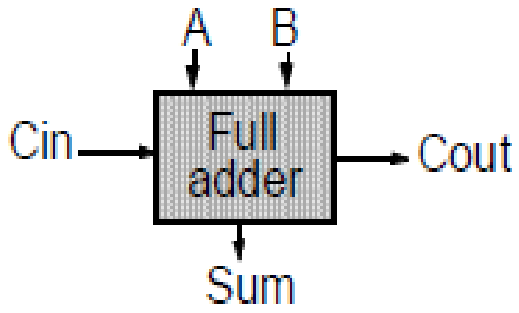



- branch point
- concatenation

block carry-in determines selection

# Full Adder - Revisited

Reformulate basic adder stage:




| $a_i$ $b_i$ | $c_i$ | $S_i$ | $c_{i+1}$ | Carry Status |
|---|---|---|---|---|
| 0 0 | 0 | 0 | 0 | Delete |
| | 1 | 1 | 0 | (Carry) |
| 0 1 | 0 | 1 | 0 | Propagate |
| | 1 | 0 | 1 | (Carry-in) |
| 1 0 | 0 | 1 | 0 | |
| | 1 | 0 | 1 | |
| 1 1 | 0 | 0 | 1 | Generate |
| | 1 | 1 | 1 | (Carry-out) |

$$Sum = A \oplus B \oplus Cin$$

$$Cout = A.B + (A \oplus B).Cin$$

$$c_{i+1} = g_i + p_i c_i$$
$$s_i = p_i \oplus c_i$$

carry "propagate"

$$p_i = a_i \oplus b_i$$

carry "generate"

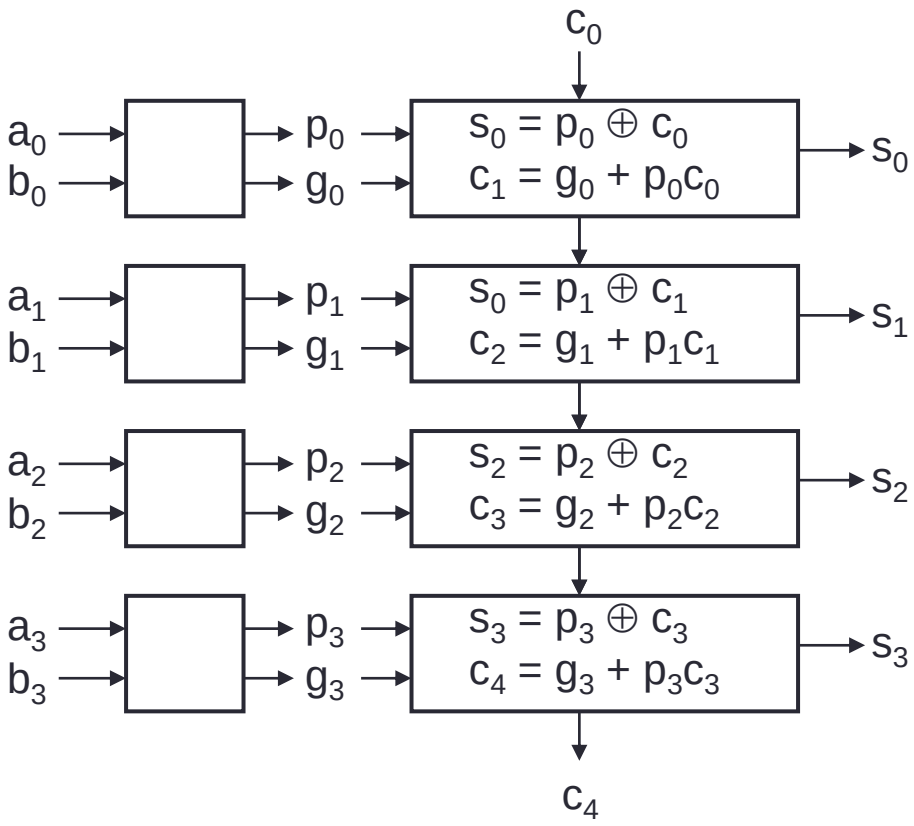
$$g_i = a_i\, b_i$$

# Carry Look Ahead Adder

- Calculates the carry signals in advance, based on the input signals
- Define two additional signals $g_i$ and $p_i$
- $g_i$ is the *generate* bit, which is 1 only if the carry out is to be generated irrespective of the input carry. This is the case, when both inputs are high
- $p_i$ is the *propagate* bit, which is 1 only if the output carry is to be same as input carry, i.e. $c_i = c_{i+1}$. This is the case, when only one of the two inputs is high.
- Thus the carry out of ***i***th stage can be expressed as $c_{i+1} = g_i + p_i c_i$

# Carry Look-ahead Adders

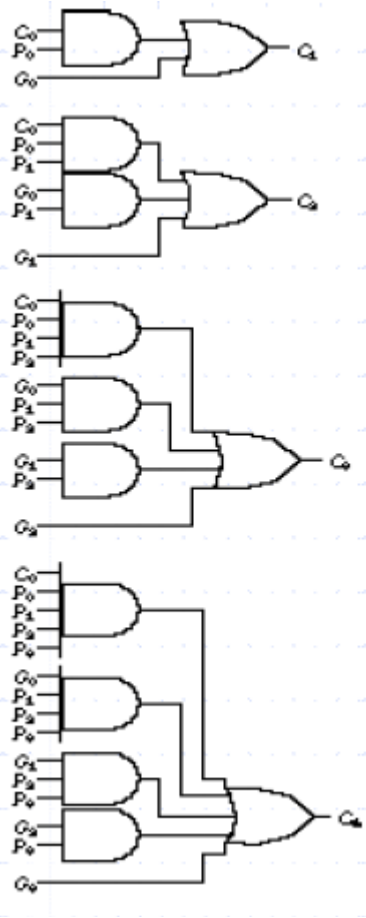
- Ripple adder using p and g signals:




- So far, no advantage over ripple adder: $T \alpha N$

# Carry Look-ahead Adders

- If $C_4$=1, then either:
  - $g_3$ generated at bit pos 3
  - $g_2.p_3$ generated at bit pos 2, propagated 3
  - $g_1.p_2.p_3$ generated at bit pos 1, propagated 2,3
  - $g_0.p_1.p_2.p_3$ generated at bit pos 0, propagated 1,2,3
  - $C_{in}.p_0.p_1.p_2.p_3$ input carry, propagated 0,1,2,3
- $C_4 = g_3 + g_2.p_3 + g_1.p_2.p_3 + g_0.p_1.p_2.p_3 + C_{in}.p_0.p_1.p_2.p_3$



- Expand carries:

$c_0$

$c_1 = g_0 + p_0 c_0$

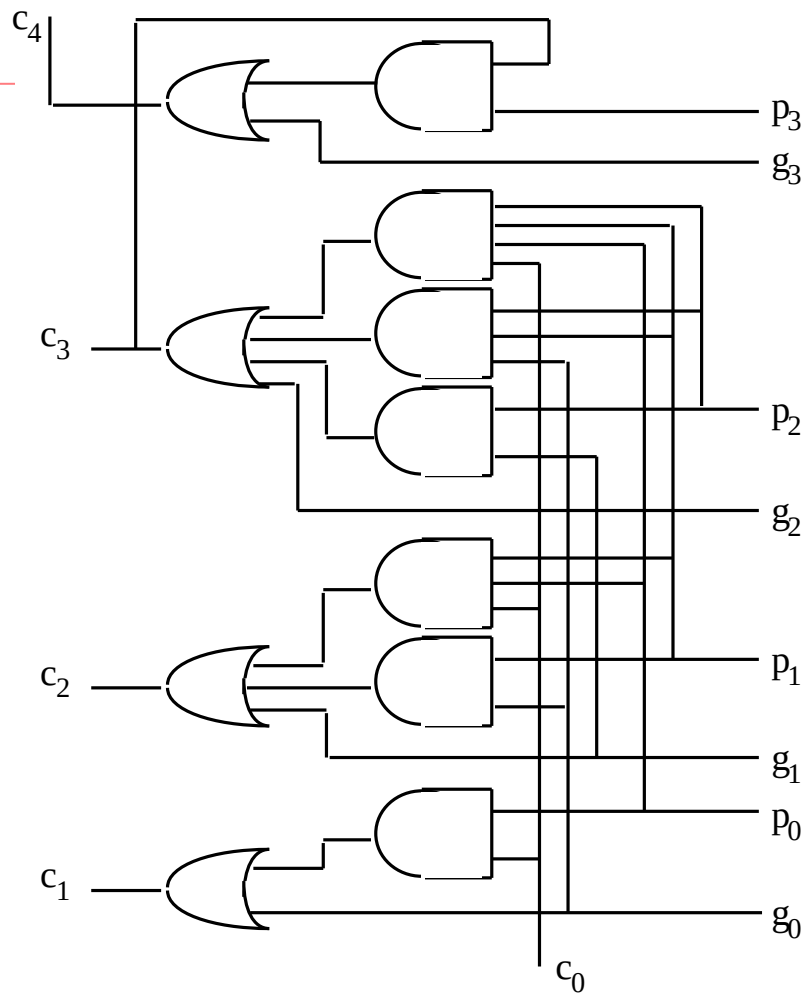
$c_2 = g_1 + p_1c_1 = g_1 + p_1g_0 + p_1p_0c_0$

$c_3 = g_2 + p_2c_2 = g_2 + p_2g_1 + p_1p_2g_0 + p_2p_1p_0c_0$

$c_4 = g_3 + p_3c_3 = g_3 + p_3g_2 + p_3p_2g_1 + p_3p_2p_1g_0 + p_3p_2p_1p_0c_0$

# Four-Bit Carry Look-ahead Adder

Complexity reduced by deriving the carry-out indirectly




Full carry look-ahead is quite practical for a 4-bit adder

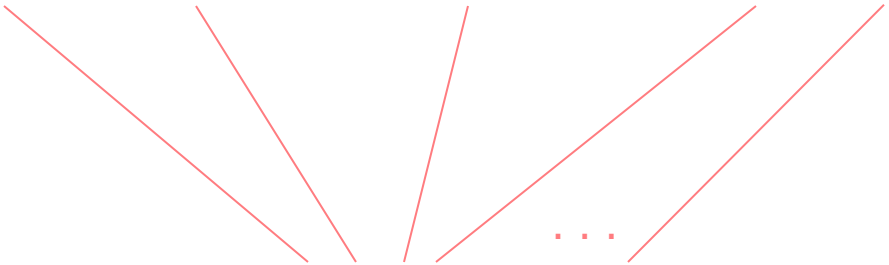
# Carry Look-ahead Beyond 4 Bits

Consider a 32-bit adder

$c_1 = g_0 \mid c_0 p_0$
$c_2 = g_1 \mid g_0 p_1 \mid c_0 p_0 p_1$
$c_3 = g_2 \mid g_1 p_2 \mid g_0 p_1 p_2 \mid c_0 p_0 p_1 p_2$
.
.
.

32-input AND

$c_{31} = g_{30} \mid g_{29} p_{30} \mid g_{28} p_{29} p_{30} \mid g_{27} p_{28} p_{29} p_{30} \mid \ldots \mid c_0 p_0 p_1 p_2 p_3 \ldots p_{29} p_{30}$




32-input OR

High fan-ins necessitate tree-structured circuits

# Two Solutions to the Fan-in Problem

- High-radix addition (i.e., radix $2^h$)

  Increases the latency for generating $g$ and $p$ signals and sum digits, but simplifies the carry network

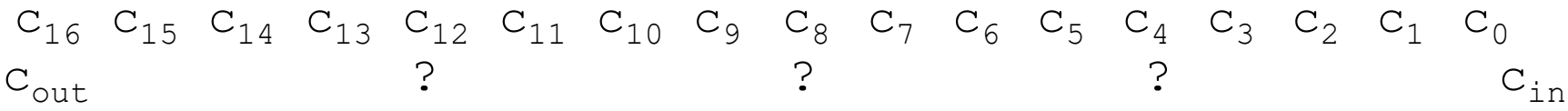
- Multilevel look-ahead

- Example: 16-bit addition

  Radix-16 (four digits)

  Two-level carry look-ahead (four 4-bit blocks)

- Either way, the carries $c_4$, $c_8$, and $c_{12}$ are determined first

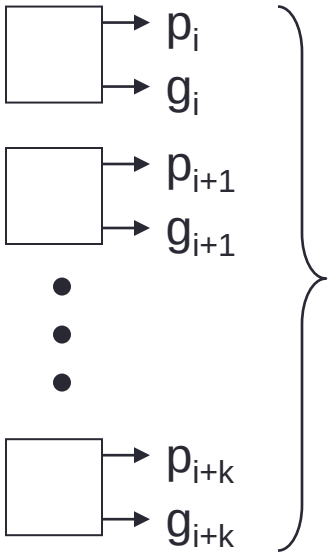


# Multilevel CLA

- “Group” propagate and generate signals:

- P true if the group as a whole propagates a carry to $c_{out}$
- G true if the group as a whole generates a carry

- Group P and G can be generated hierarchically.




$c_{in}$

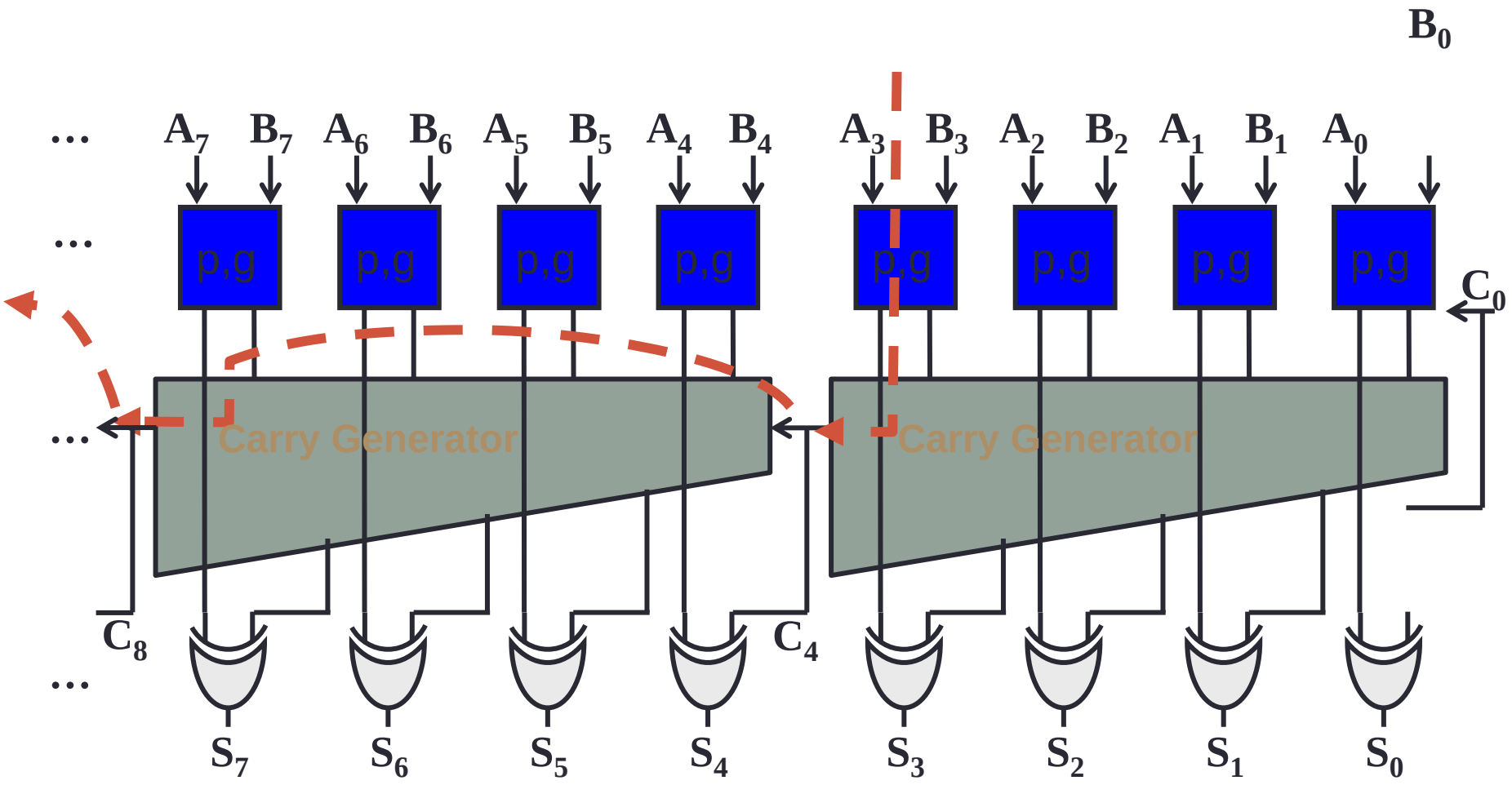
$P = p_i\, p_{i+1} \ldots p_{i+k}$

$G = g_{i+k} + p_{i+k}g_{i+k-1} + \ldots + (p_{i+1}p_{i+2} \ldots p_{i+k})g_i$

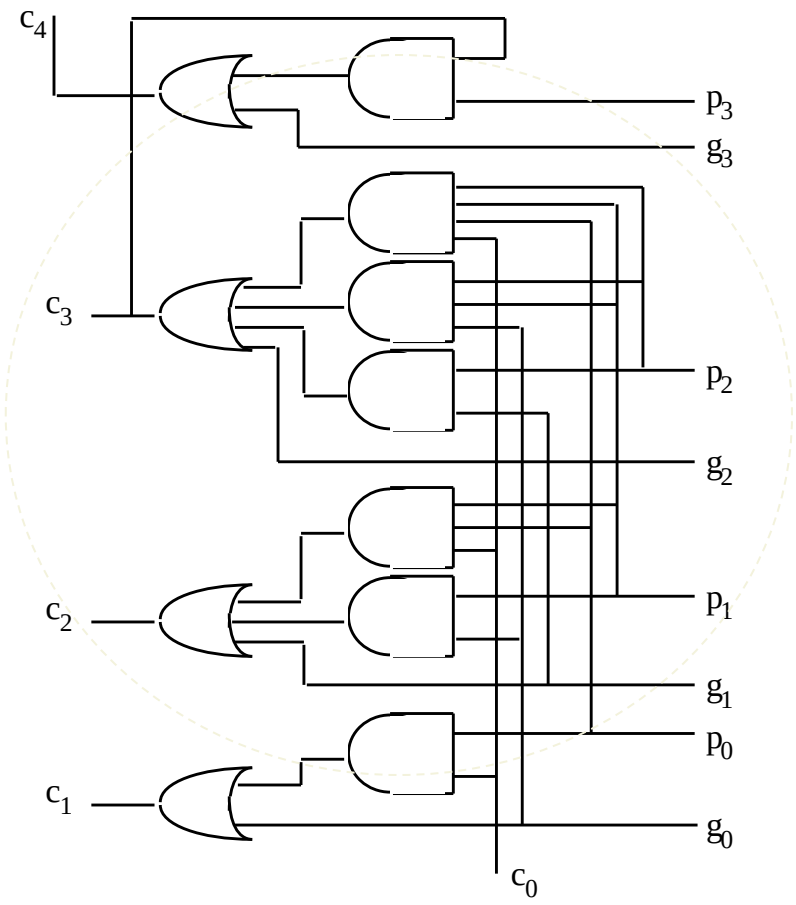
$c_{out}$

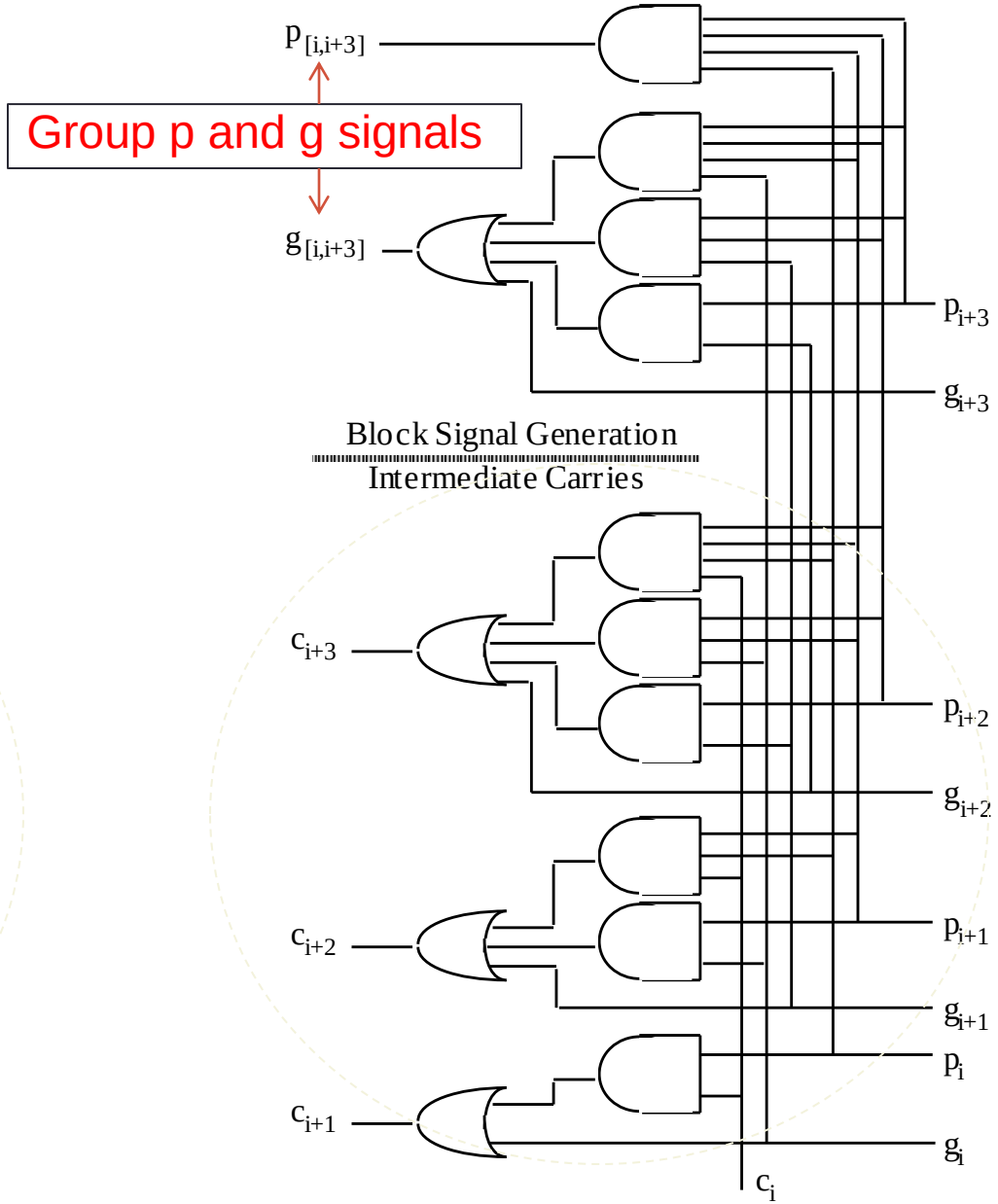
$$C_{out} = G + PC_{in}$$

# CLA: N-Bit Architecture



# A Building Block for Multilevel CLA



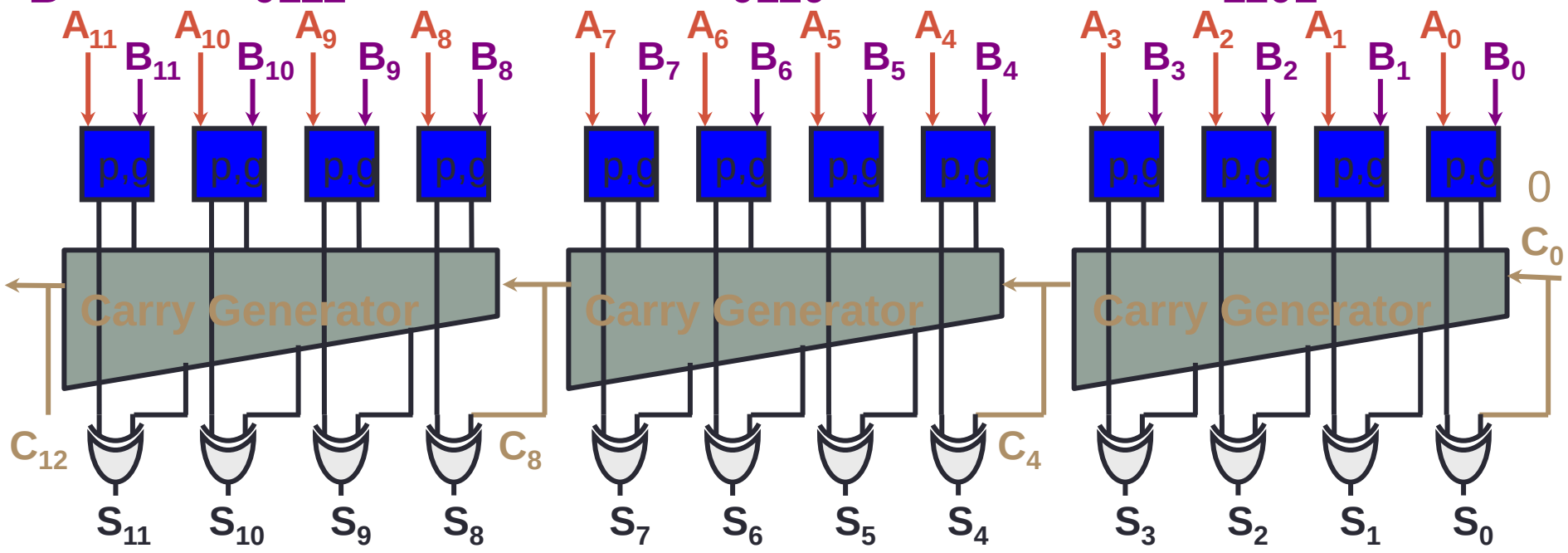




A 4-bit CLA group network

# 12-Bit CLA

A= 1101 1001 1010

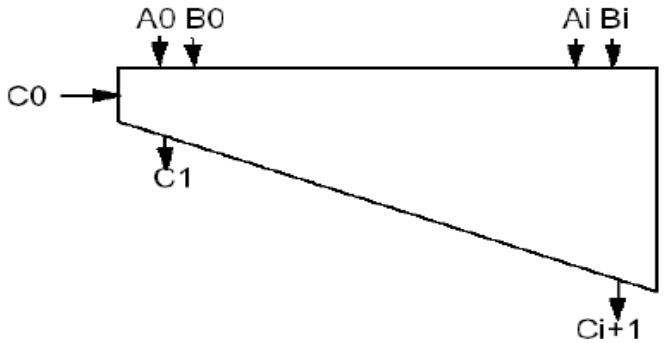
B= 0111 0110 1101




| | | | | | | | |
|---|---|---|---|---|---|---|---|
| T=0 | 0 | 0000 | | 0 | 0000 | | 0 | 0000 |
| T=2 | 1 | 0100 | | 0 | 1111 | | 1 | 0111 |
| T=3 | 1 | 0100 | | 1 | 0000 | | 1 | 0111 |
| T=4 | 1 | 0101 | | 1 | 0000 | | 1 | 0111 |

# Summary: Carry Look-ahead Adder

- CLA compared to ripple-carry adder:
  - Faster
    but delay still linear (w.r.t. # of bits)
  - Larger area
    - P, G signal generation
    - Carry generation circuits
    - Carry generation circuit for each bit position (no re-use)
- Limitation: cannot go beyond 4 bits of look-ahead
  - Large p,g fan-out slows down carry generation
- Non-Uniform Lay-out



# References

- The contents of this lecture are courtesy of
  - Kia Bazargan
  - Behrooz Parhami
  - Kris Gaj
  - Sukumar Ghosh
  - Narjis Fatima

Write down Verilog Code to swap three numbers on falling clock edge, (none of them retains its previous value). The circuit should have a synchronous reset.

Give most optimum level implementation of a 25-bit Non-uniform Carry Select Adder. Compare the results with uniform Carry Select Adder.

Point out the errors in the following Verilog code

```
always @ (*) begin
if (sel == 2'b00) begin
out_a = 2'b01;
out_b = 2'b10; end
else out_a = 2'b01; end
```

Code the logic of following equations in Verilog. All variables are two bit wide

```
acc0 = acc1 + in1;
acc1= acc0 + in2;
out = acc0 + acc1;
```

Write down Verilog Code of a circuit that produces a series of Fibonacci numbers (0,1,2,3,5,8….). The only inputs to the circuit are clock and reset. The output is 5-bits. Assume the reset is asserted at start.

Give most optimum level implementation of an 18-bit Non-uniform Carry Select Adder. Compare the results with uniform Carry Select Adder.

Point out the mistake in the following Verilog code

```
always@(a or b or c)
 case(a)
    2'b11 : e = b;
    2'b10 : e = ~c;
 end case
```

Write a Verilog description for a circuit that accepts a 4-bit input and outputs true if the input is a Fibonacci number