Durham
University

Digital Electronics

Timing and Advanced Adders

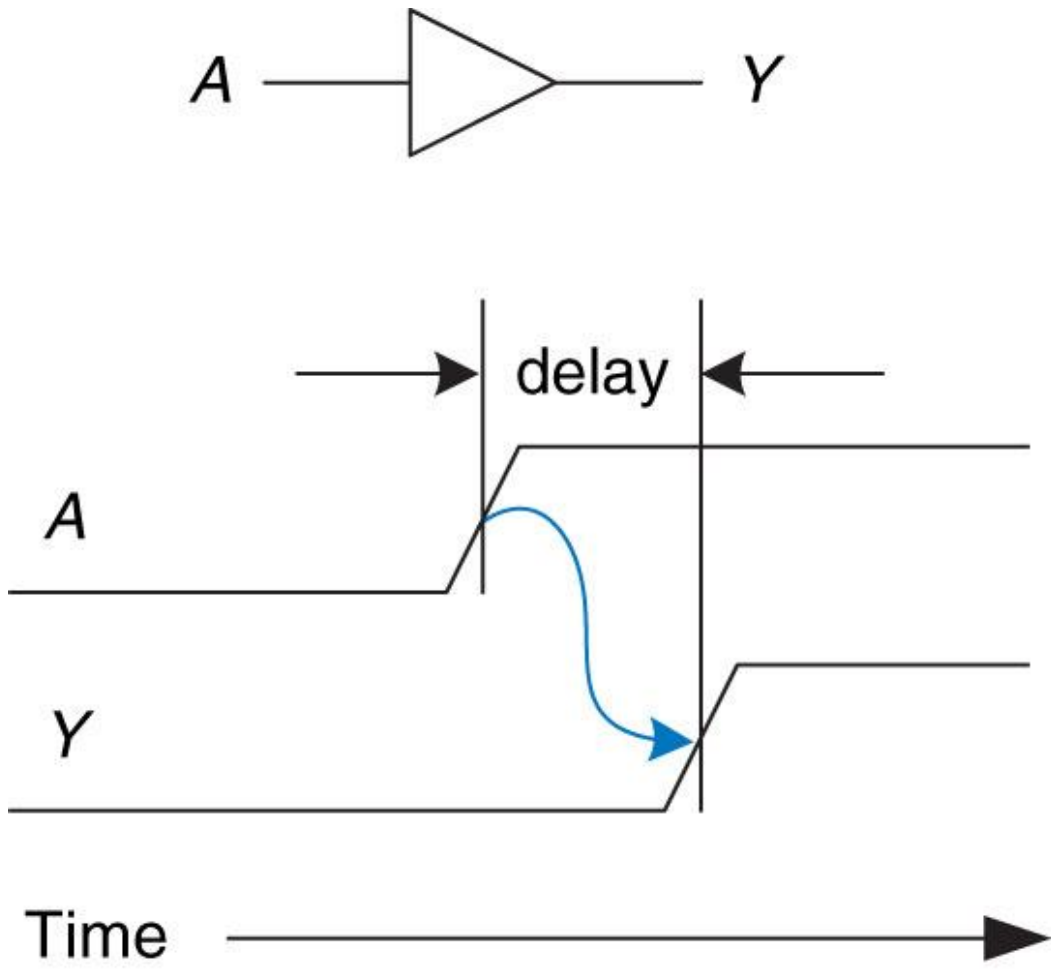
Dr. Eleni Akrida
eleni.akrida@durham.ac.uk

Overview of today’s lecture

- Timing
- Critical Paths
- Glitches
- Ideas for faster adders

Timing

In any physical gate or circuit there is a delay between the input changing and the output adjusting appropriately.

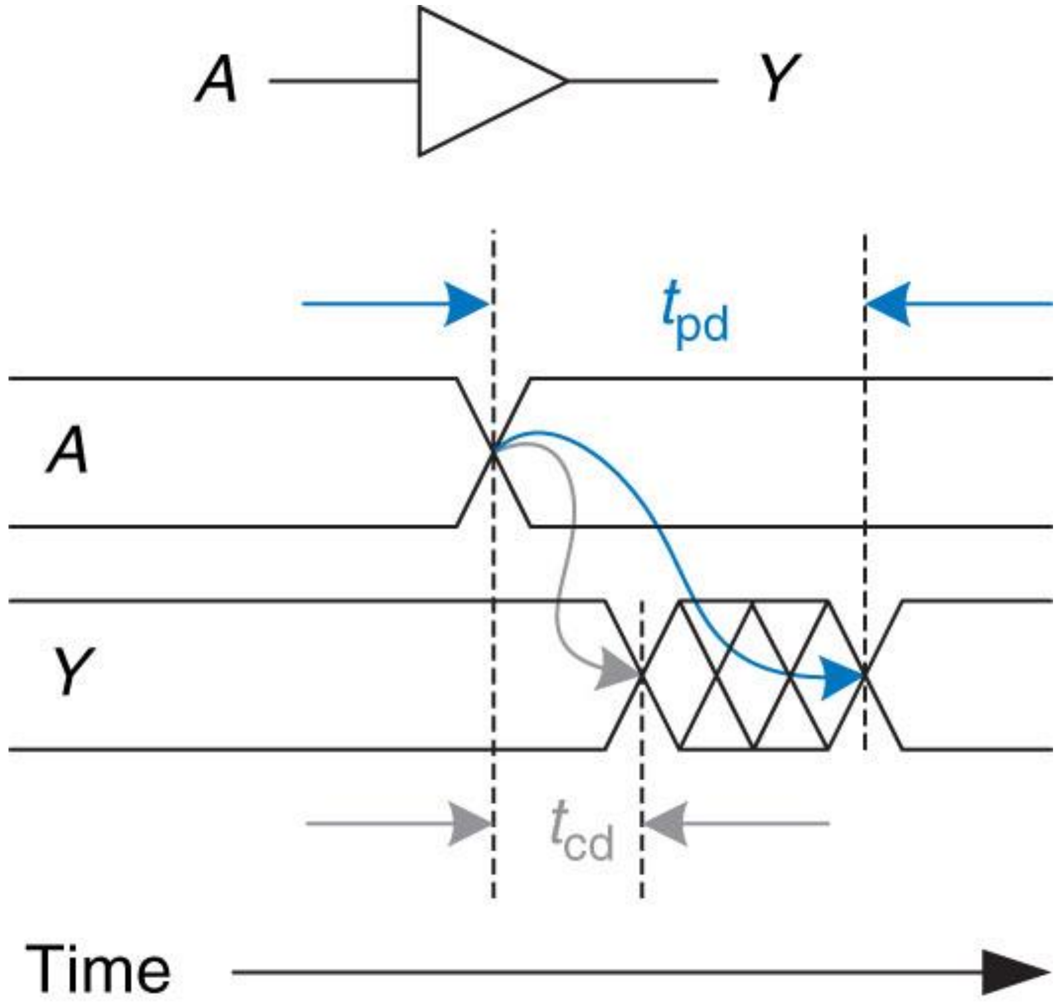


Timing

In any physical gate or circuit there is a delay between the input changing and the output adjusting appropriately.

Propagation delay: t_{pd}
the max delay before the output is stable.

Contamination delay: t_{cd}
the min delay before the output changes.

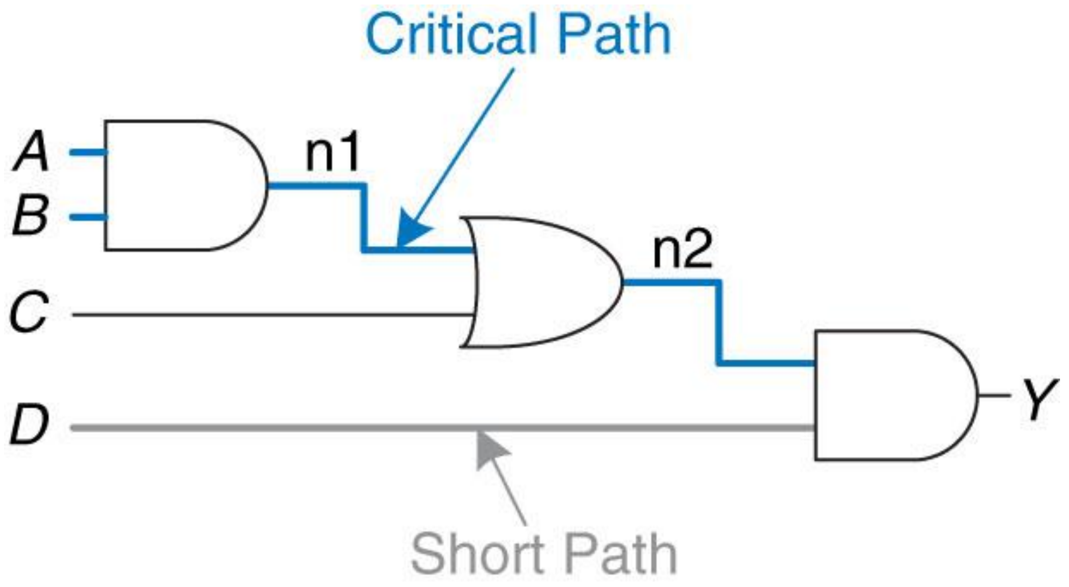


Timing

- **Delay is caused by**
  - Capacitance and resistance in a circuit
  - The speed of light limitation

- Reasons why t_{pd} and t_{cd} may be **different**:
  - Different rising and falling delays
  - A circuit may have multiple inputs and outputs, some of which are faster than others
  - Circuits slow down when hot and speed up when cold

Critical paths

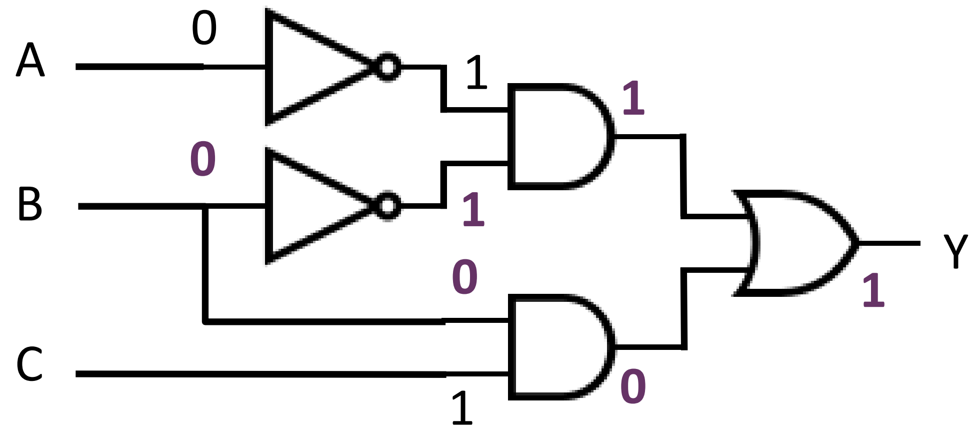


In a circuit the **critical path** is the path determining the propagation delay of the circuit – i.e. the **longest path** in the circuit: $t_{pd} = 2t_{pd\_\mathrm{AND}} + t_{pd\_\mathrm{OR}}$

The **short path** is the path determining the contamination delay of the circuit – i.e. the **shortest path** in the circuit: $t_{cd} = t_{cd\_\mathrm{AND}}$

Glitches

Sometimes the output can **temporarily** move to an incorrect value before stabilising. This is called a **glitch**.



A=0, B=1, C=1
What happens when B falls to 0?

Mux delays

4-line multiplexor circuits.

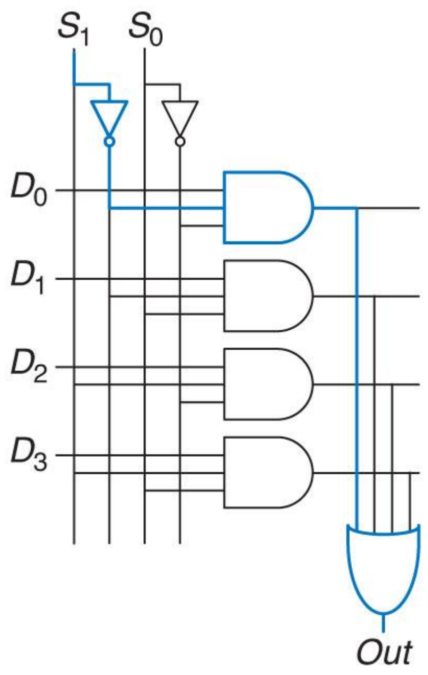
Different characteristics

for **selector change**

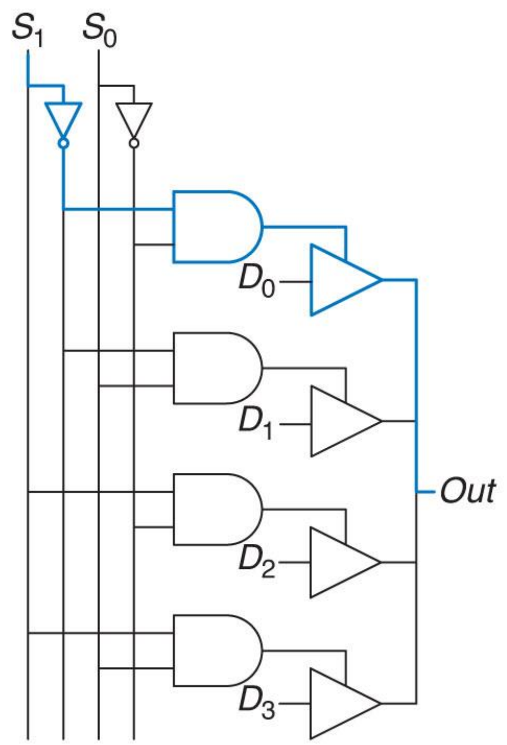
and **data change**

Gate	t_{pd} (ps)
NOT	30
AND	60
3-AND	80
4-OR	90
Tristate (D)	50
Tristate (S)	35

“Canonical” Design:



Tristate gates design:



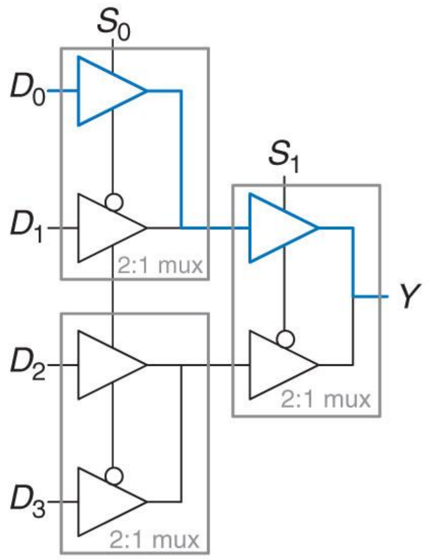
Mux delays

4-line multiplexor circuits.

Different characteristics for **selector change** and **data change**

Gate	t_{pd} (ps)
NOT	30
AND	60
3-AND	80
4-OR	90
Tristate (D)	50
Tristate (S)	35

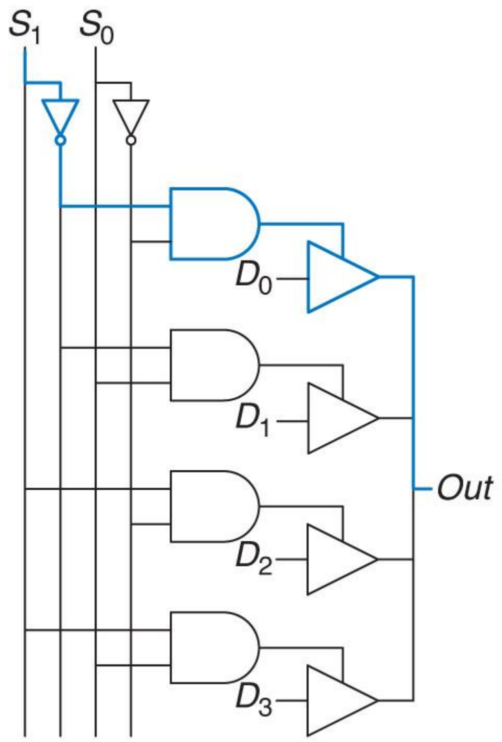
Hierarchical Design:



Prefer (b) when data changes last,

but prefer (c) when control changes last

Tristate gates design:

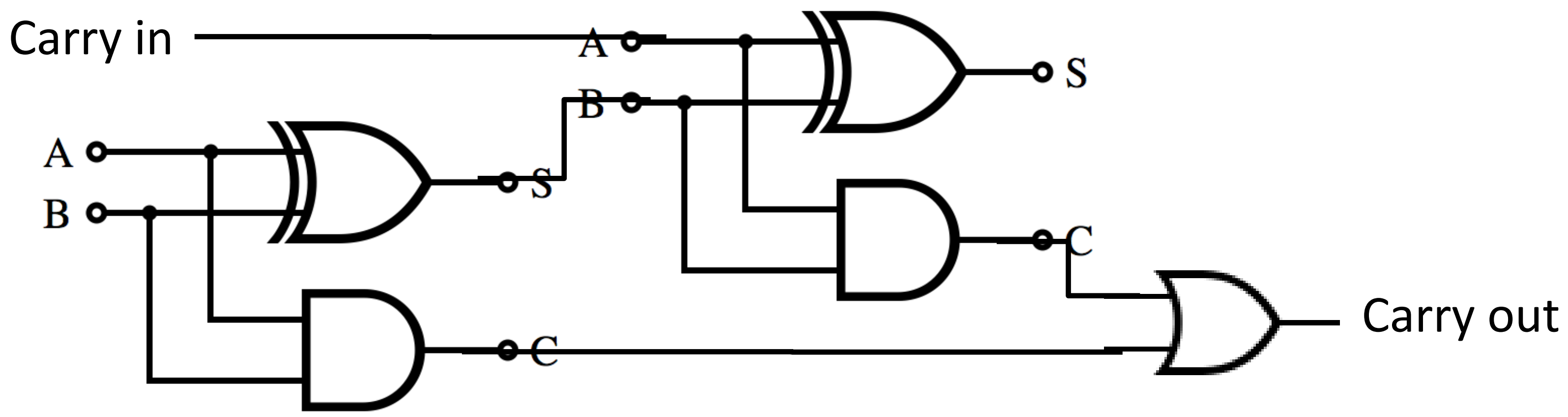


(b)

$$t_{pd\_sy} = t_{pd\_INV} + t_{pd\_AND2} + t_{pd\_TRI\_sy}$$
$$= 30 \text{ ps} + 60 \text{ ps} + 35 \text{ ps}$$
$$= \mathbf{125 \text{ ps}}$$
$$t_{pd\_dy} = t_{pd\_TRI\_ay}$$
$$= \mathbf{50 \text{ ps}}$$

Adder

The full adder circuit we have looked at takes **3 gate delays** for the carry out to be computed:



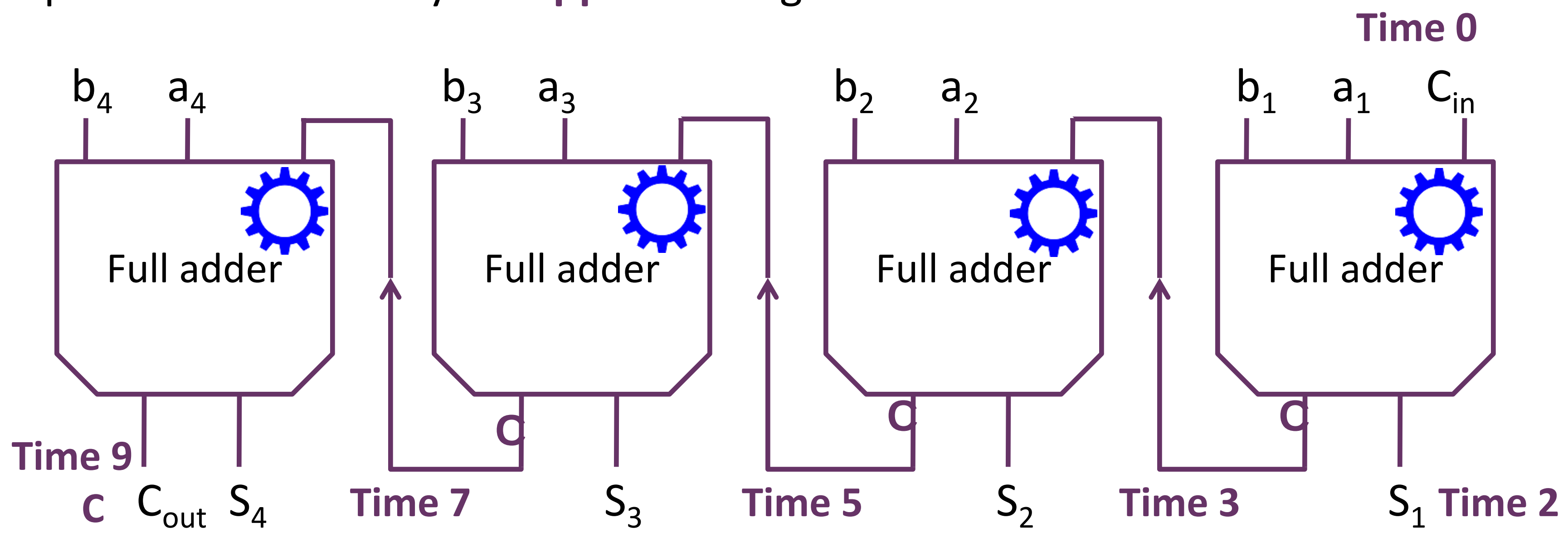
Time 0 **Time 1** **Time 2** **Time 3**

If we can **pre-compute the first level**, there are only 2 gate delays once the carry in arrives.

N.B. I am counting all gates as 1 time step. In reality some take longer than others.

Ripple adder

The computation of the carry bit **ripples** through the chained adders:

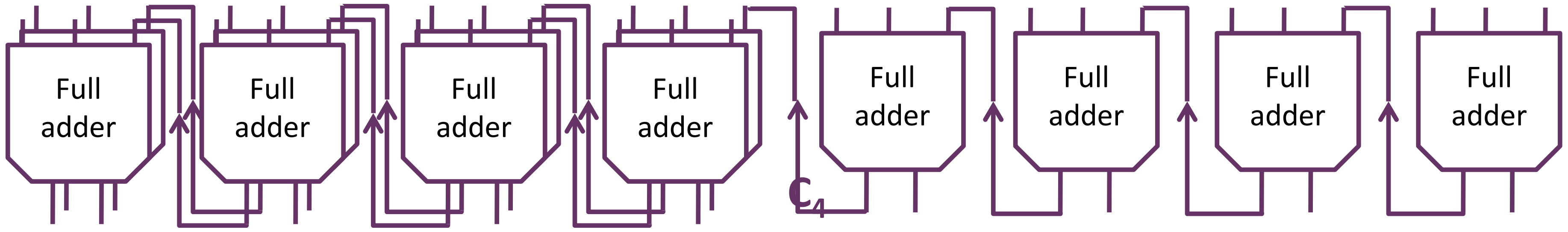


A k-bit ripple adder will take 3+2(k-1)=**2k+1** gate delays to compute C_{out}.

So a 32-bit adder takes 65 gate delays!

Faster adders - idea

8-bit ripple adder takes 17 gate delays.



Time 9: Answer if C_4=0 and if C_4=1

Time 9: Answer and C_4

Time 11: Correct Answer

Key idea: for later bits pre-compute the outcome with both carry in and no carry in, then quickly select the right answer.

Faster addition at the expense of more circuitry.

Carry-Lookahead Adder

The 8 rules of binary addition:

0 + 0 = 0	C_{in} + 0 + 0 = 1
0 + 1 = 1	C_{in} + 0 + 1 = 0 with C_{out}
1 + 0 = 1	C_{in} + 1 + 0 = 0 with C_{out}
1 + 1 = 0 with C_{out}	C_{in} + 1 + 1 = 1 with C_{out}

Define two functions.

Generate: G(A,B) = 1 if A and B would cause C_{out} even if C_{in} = 0.

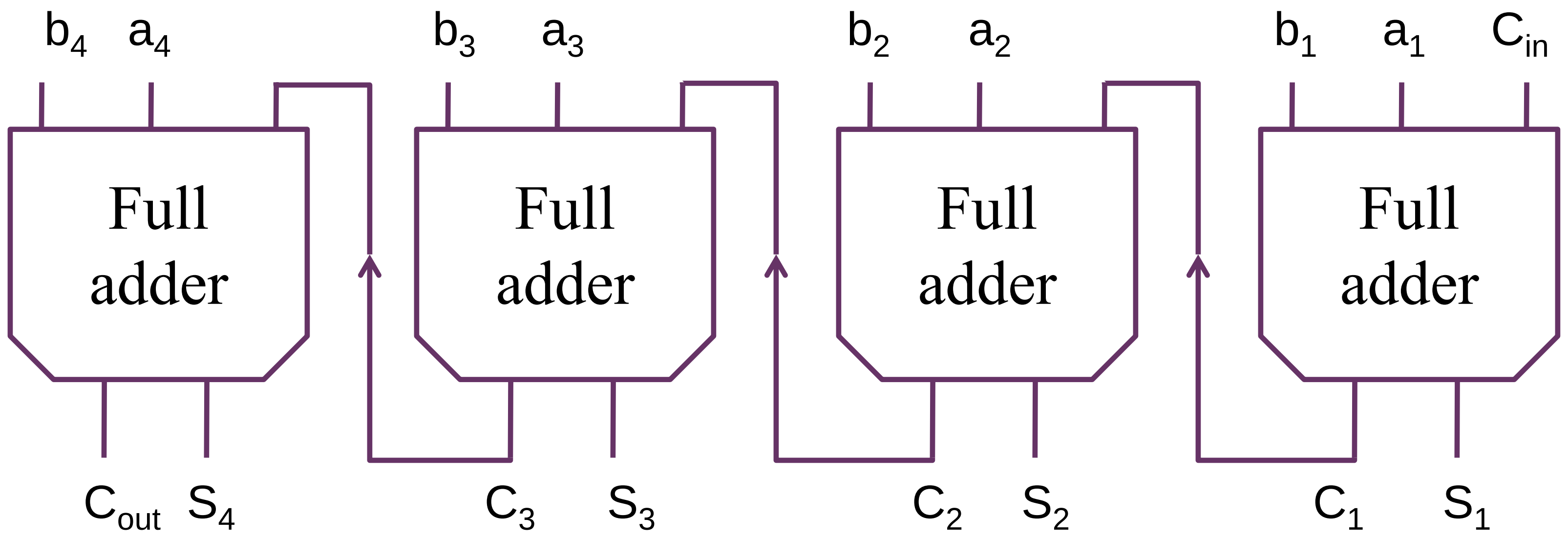
Propagate: P(A,B) = 1 if A and B would cause C_{out} if C_{in} = 1.

G(A,B) = A AND B

P(A,B) = A OR B

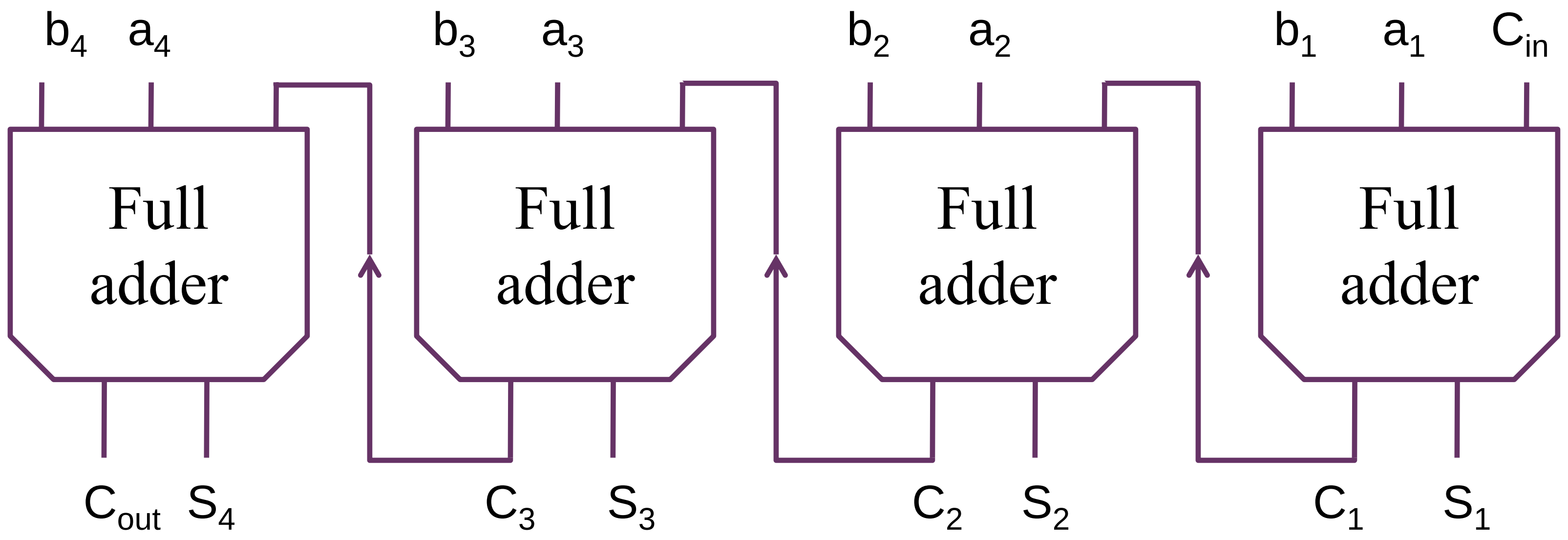
Carry occurs if it is **generated** or if there is carry in and it is **propagated**: $C_{out} = G(A,B) + P(A,B) \cdot C_{in}$

Carry-Lookahead Adder



$$
\begin{aligned}
C_{out} &= G_4 + P_4 \cdot C_3 \\
&= G_4 + P_4 \cdot (G_3 + P_3 \cdot C_2) \\
&= G_4 + P_4 \cdot (G_3 + P_3 \cdot (G_2 + P_2 \cdot C_1)) \\
&= G_4 + P_4 \cdot (G_3 + P_3 \cdot (G_2 + P_2 \cdot (G_1 + P_1 \cdot C_{in}))) \\
&= G_4 + P_4G_3 + P_4P_3G_2 + P_4P_3P_2G_1 + P_4P_3P_2P_1 \cdot C_{in}
\end{aligned}
$$

Carry-Lookahead Adder



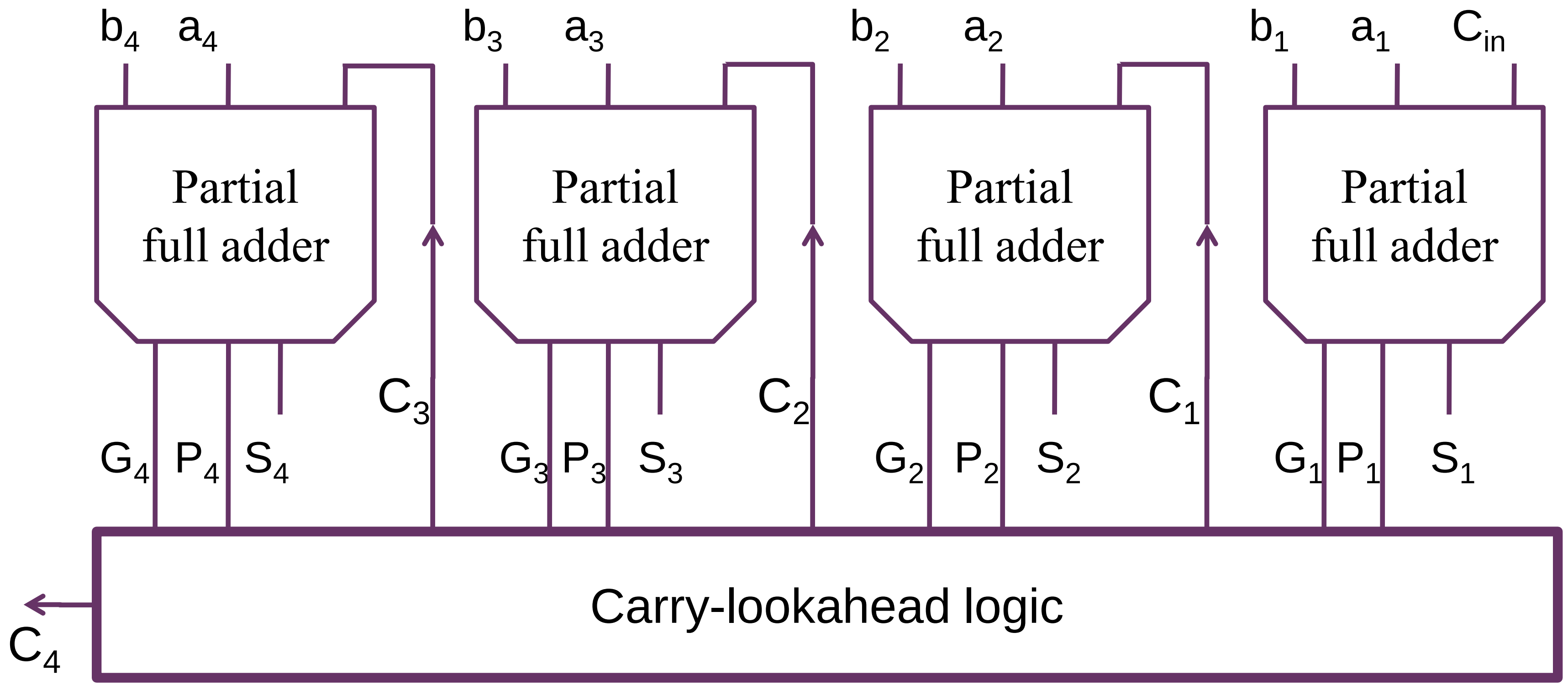
$C_{out} = G_4 + P_4G_3 + P_4P_3G_2 + P_4P_3P_2G_1 + P_4P_3P_2P_1 \cdot C_{in}$

We can compute every P_i and G_i in one gate delay

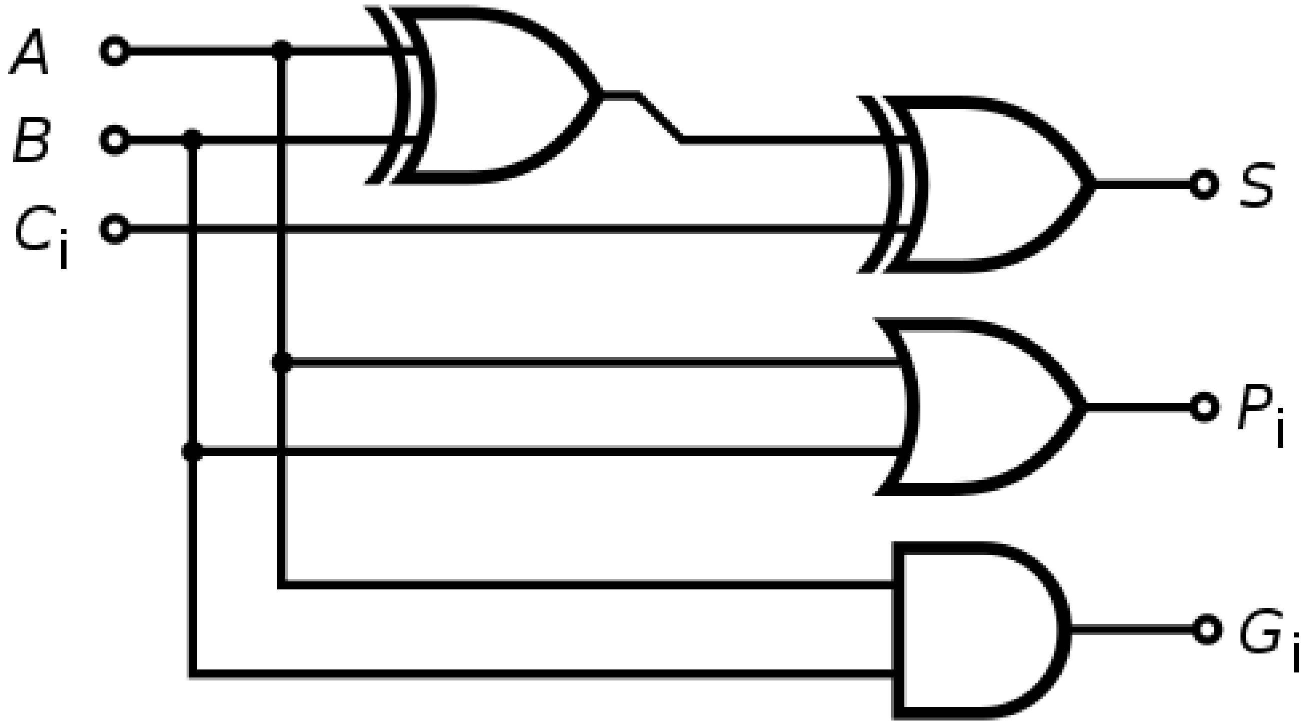
So we can compute C_{out} (and all the intermediate carries) in 3 gate delays (1 for P_i,G_i, an AND layer and an OR layer).

We could compute the full sum in 4 gate delays!

Carry-Lookahead Adder

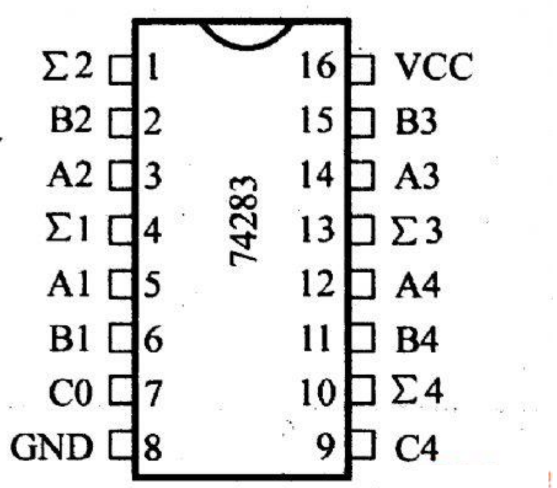


Partial full Adder

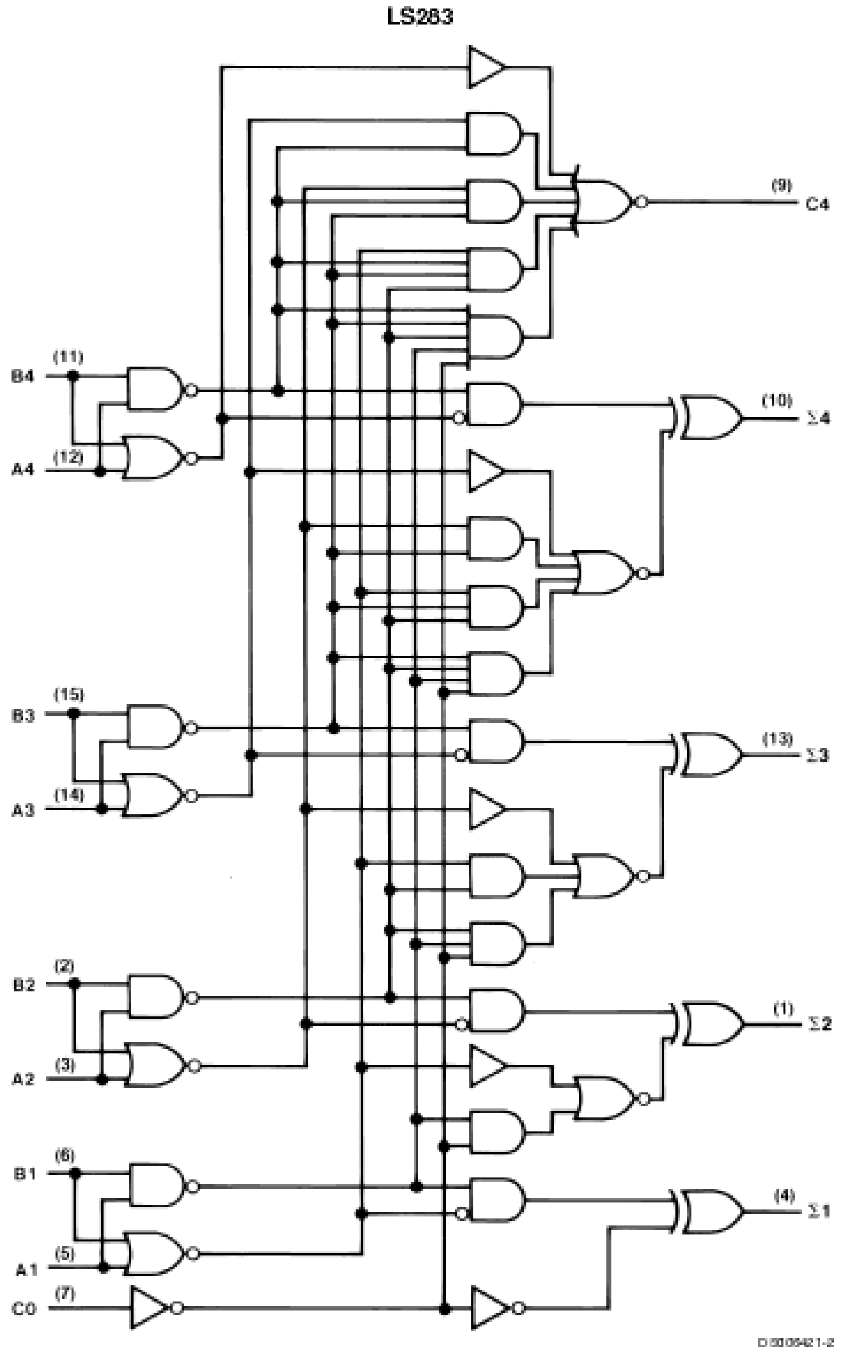


MSI chip 74x283

- 4-bit CLA adder



- **Notes:**
- Gate depth 4
- Requires gates with many inputs



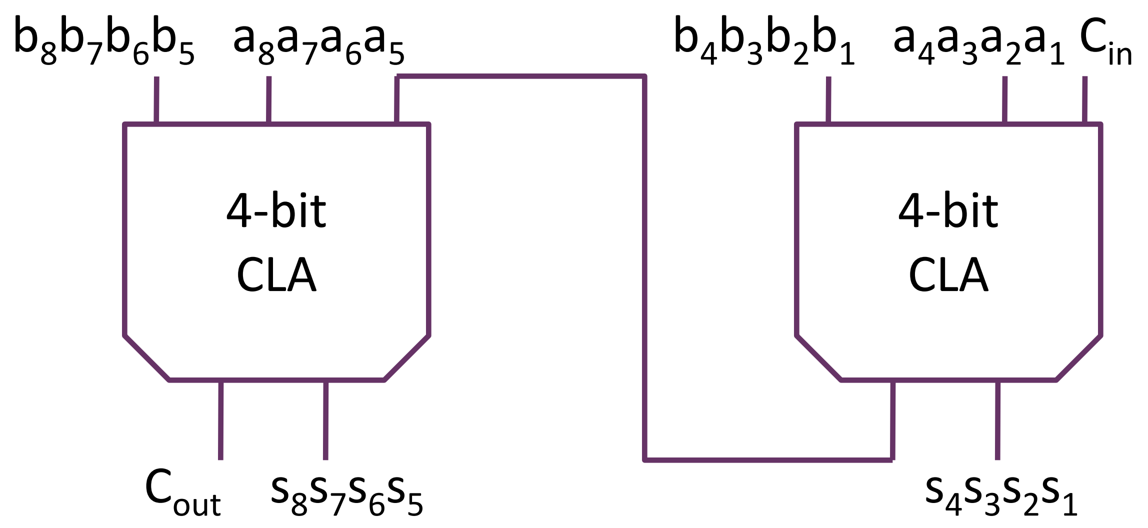
Carry-Lookahead Adder

We could create a CLA which adds n-bit numbers in constant gate delay.

It would require order n^2 gates and gates taking order n inputs.

This is impractical for large n.

First solution: chain 4-bit CLAs

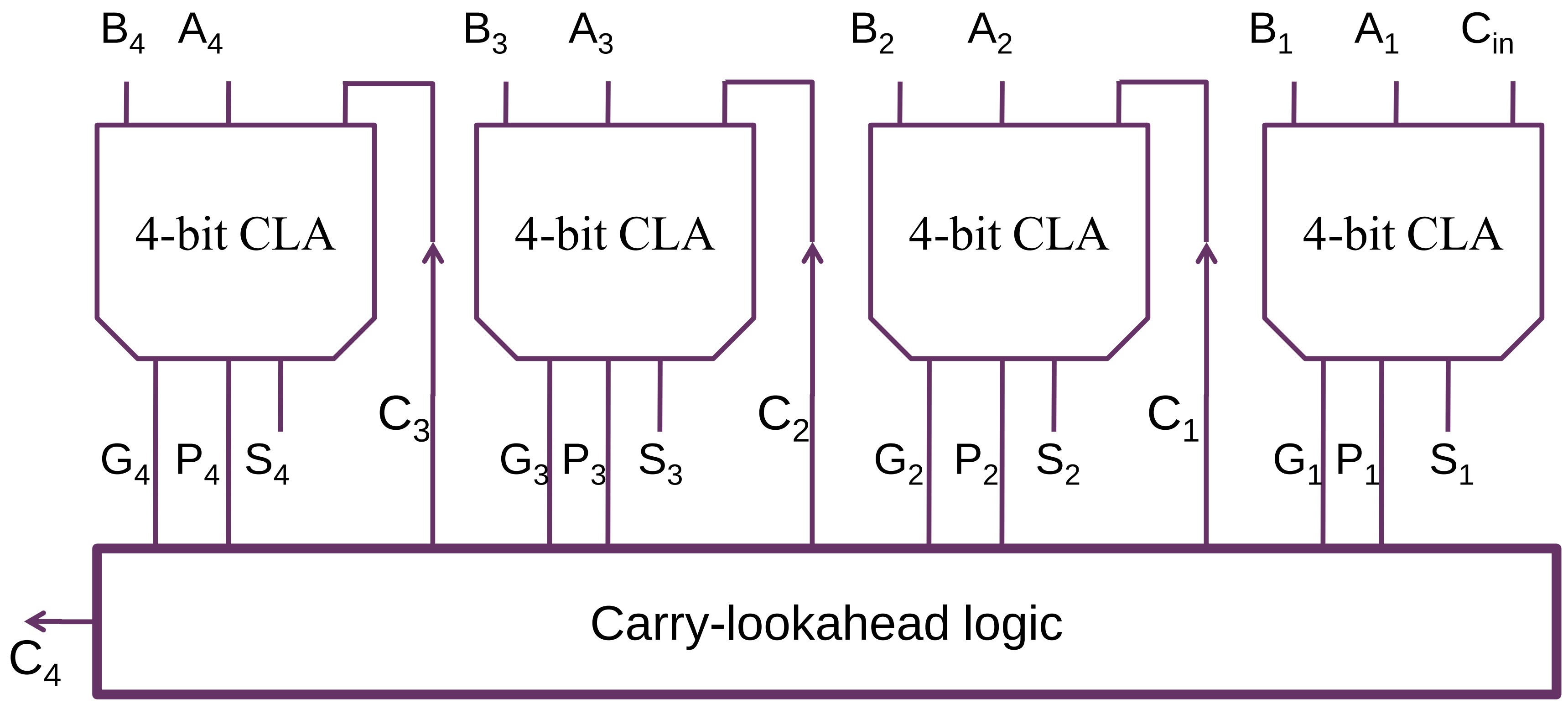


$b_8b_7b_6b_5$ $a_8a_7a_6a_5$
$b_4b_3b_2b_1$ $a_4a_3a_2a_1$ C_{in}
4-bit CLA
4-bit CLA
C_{out} $s_8s_7s_6s_5$
$s_4s_3s_2s_1$

The carry now ripples along the chain 4-times as fast as originally

2-level Carry-Lookahead Adder

Rather than letting the carry ripple through the CLAs, we can pre-compute whether it would be generated or propagated by each CLA.



A_i, B_i, S_i each represent 4 bits.

2-level Carry-Lookahead Adder

We now have a 16-bit 2-level CLA.

Gate delays:

- 1 gate delay to compute first (bit) level Gs, Ps
- 2 gate delays to compute second (block) level Gs, Ps
- 2 gate delays to compute carries across the whole system
- 1 gate delay to compute sums across the whole system

Total **6 gate delays.**

A standard 16-bit ripple carry adder would take 33 gate delays.

These 16-bit adders could be chained to give an n-bit adder with gate delay 2(n/16)+4.

Alternatively, a 3 (or more) level CLA could be used to create an n-bit adder with delay $O(\log n)$ and not too much circuitry.