EE-421: Digital System Design

Doing MATHS on an FPGA

Arithmetic Circuit:
(Revisiting) Adder for Performance
Carry Look-Ahead Adder

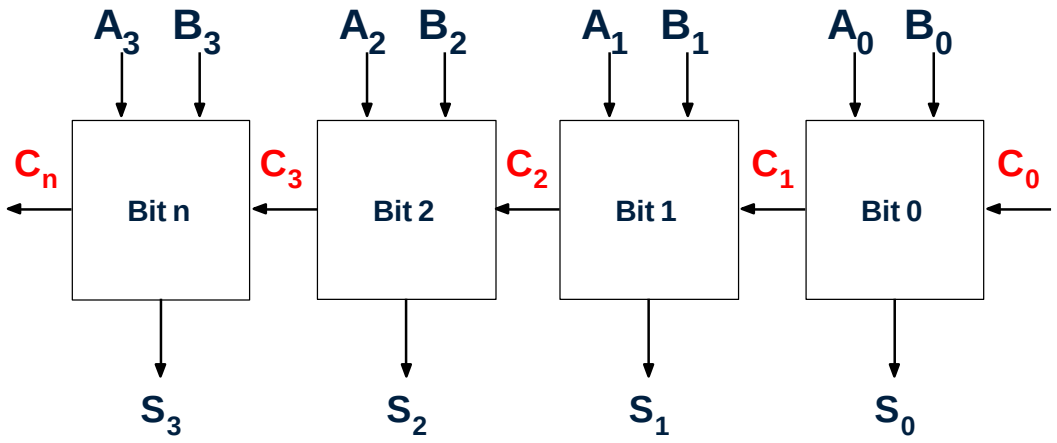
Dr. Rehan Ahmed [rehan.ahmed@seecs.edu.pk]

NUST
SCHOOL OF ELECTRICAL ENGINEERING &
COMPUTER SCIENCE (SEECS)

Carry Look-Ahead Adder (CLA)

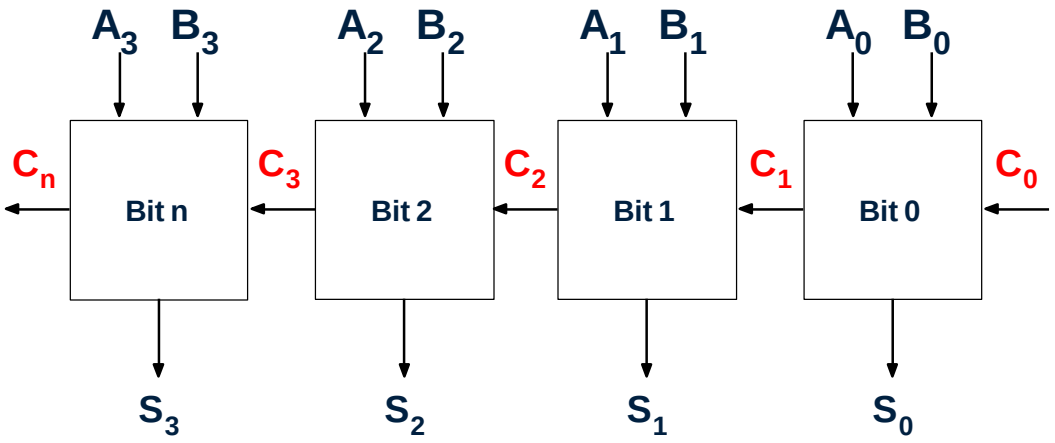
Mr. Carry

Approach



- Attempt (look-ahead) to evaluate quickly for each stage whether the carry-in from the previous stage will have a value 0 or 1:
  - If a correct evaluation can be made in a relatively short time, then the performance of the complete adder will be improved.

Approach – how?



$$c_{i+1} = a_i b_i + a_i c_i + b_i c_i$$

$$c_{i+1} = a_i b_i + (a_i + b_i) c_i$$

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

G = Generate

P = Propagate

Generate Term

A_i, B_i would generate a carry if both are 1

A_i	B_i	G_i
0	0	0
0	1	0
1	0	0
1	1	1

$$G_i = A_i \; and \; B_i$$

Propagate Term

A_i, B_i would propagate a carry if either is 1

A_i	B_i	P_i
0	0	0
0	1	1
1	0	1
1	1	1

$$P_i = A_i \; or \; B_i$$

Approach

$$c_{i+1} = a_i b_i + a_i c_i + b_i c_i$$

$$c_{i+1} = a_i b_i + (a_i + b_i) c_i$$

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

- The function $\mathbf{g_i}$ is equal to 1 when both inputs a_i and b_i are equal to 1,
  - *regardless of the value of the incoming carry to this stage, c_i*
  - Since in this case stage i is guaranteed to generate a carry-out, **g** is called the ***generate*** function.

- The function $\mathbf{p_i}$ is equal to 1 when at least one of the inputs a_i and b_i is equal to 1,
  - In this case a carry-out is produced if $c_i = 1$.
  - The effect is that the carry-in of 1 is propagated through stage i; hence p_i is called the ***propagate*** function.

Approach Takeaway

Pre-compute parts of carry logic

For each bit of the addition, independently calculate two terms:

- **Generate term**

$$G_i = f(A_i, B_i)$$

- **Propagate term**

$$P_i = f(A_i, B_i)$$

G_i and P_i are independent of Carry terms C_i

Then, by using the various Generate and Propagate terms, we can compute the carry terms at each bit without the ripple effect.

Approach Takeaway – (2)

$$Cout_{i+1} = G_i \; or \; (P_i \; and \; Cin_i)$$

Example: Generate Term

Does adding the *i-th* bits of A, B generate a carry?

Example: A = 011, B = 010

Look at each bit-position **INDEPENDANTLY**

Does adding bit 0 generate a Carry?

A + B = 011 + 010

NO, 1 + 0 = 01 (carry NOT generated)

$G_0 = 0$

Does adding bit 1 generate a Carry?

A + B = 011 + 010

YES, 1 + 1 = 10 (carry generated)

$G_1 = 1$

Does adding bit 2 generate a Carry?

A + B = 011 + 010

NO, 0 + 0 = 00

$G_2 = 0$

Example: Propagate Term

If there was a carry-in at the *i-th* bit, would it propagate to the next stage?

Example: A = 011, B = 010

Look at each bit-position **INDEPENDANTLY**

Would bit 0 propagate a Carry?

A + B = 011 + 010

YES, 1 + 0 + 1 = 10 (carry propagated)

P_0=1

hypothetical carry-in

Would bit 1 propagate a Carry?

A + B = 011 + 010

YES, 1 + 1 + 1 = 11 (carry propagated)

P_1= 1

hypothetical carry-in

Would bit 2 propagate a Carry?

A + B = 011 + 010

NO, 0 + 0 + 1 = 01

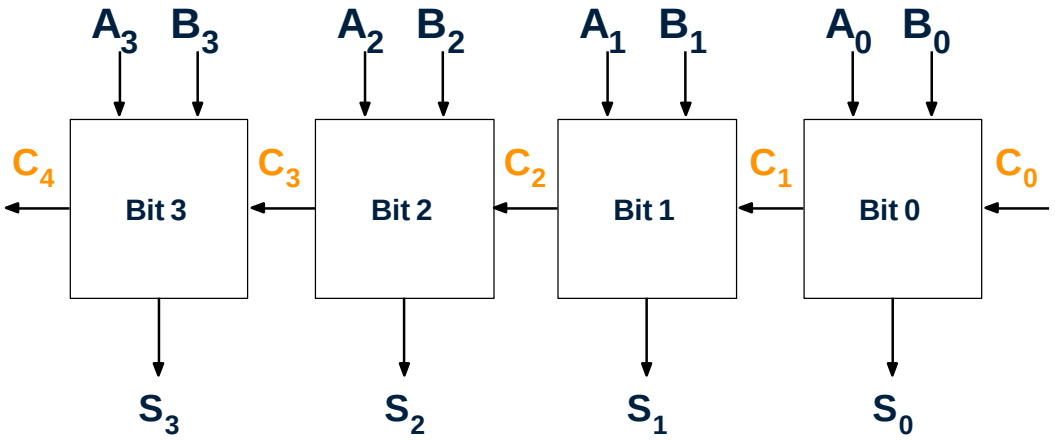
P_2= 0

hypothetical carry-in

4-bit CLA Example

Let C_i denote the carry-in of stage i

this means that it is also the carry-out of the previous stage



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

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

$$C_3 = G_2 + P_2 \cdot C_2$$

$$C_4 = G_3 + P_3 \cdot C_3$$

Note: We are using logical operators here:

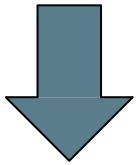
+ means OR

. means AND

Carry Look-Ahead Logic

$$C_1 = G_0 + P_0 \cdot C_0$$
$$C_2 = G_1 + P_1 \cdot C_1$$
$$C_3 = G_2 + P_2 \cdot C_2$$
$$C_4 = G_3 + P_3 \cdot C_3$$

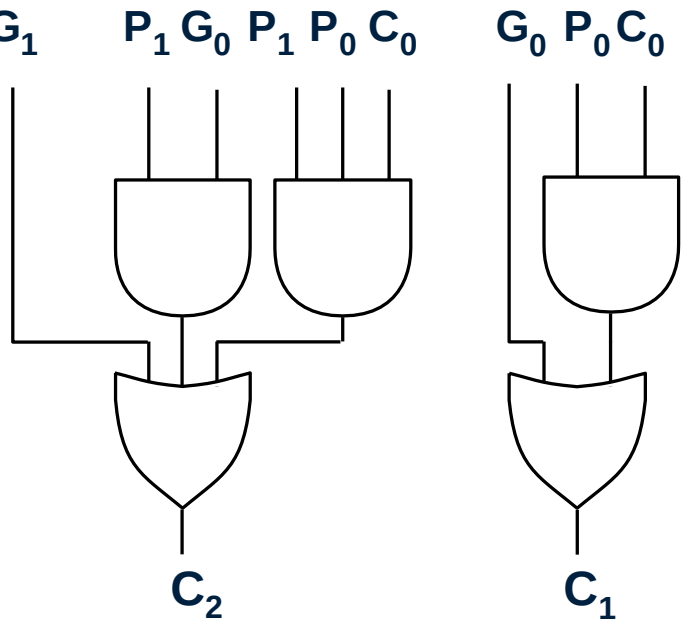
Perform Forward Substitution



$$C_1 = G_0 + P_0 \cdot C_0$$
$$C_2 = G_1 + P_1 \cdot G_0 + P_1 \cdot P_0 \cdot C_0$$
$$C_3 = G_2 + P_2 \cdot G_1 + P_2 \cdot P_1 \cdot G_0 + P_2 \cdot P_1 \cdot P_0 \cdot C_0$$
$$C_4 = G_3 + P_3 \cdot G_2 + P_3 \cdot P_2 \cdot G_1 + P_3 \cdot P_2 \cdot P_1 \cdot G_0 + P_3 \cdot P_2 \cdot P_1 \cdot P_0 \cdot C_0$$

Carry Look-Ahead Logic

$$C_1 = G_0 + P_0 \cdot C_0$$
$$C_2 = G_1 + P_1 \cdot G_0 + P_1 \cdot P_0 \cdot C_0$$
$$C_3 = G_2 + P_2 \cdot G_1 + P_2 \cdot P_1 \cdot G_0 + P_2 \cdot P_1 \cdot P_0 \cdot C_0$$
$$C_4 = G_3 + P_3 \cdot G_2 + P_3 \cdot P_2 \cdot G_1 + P_3 \cdot P_2 \cdot P_1 \cdot G_0 + P_3 \cdot P_2 \cdot P_1 \cdot P_0 \cdot C_0$$



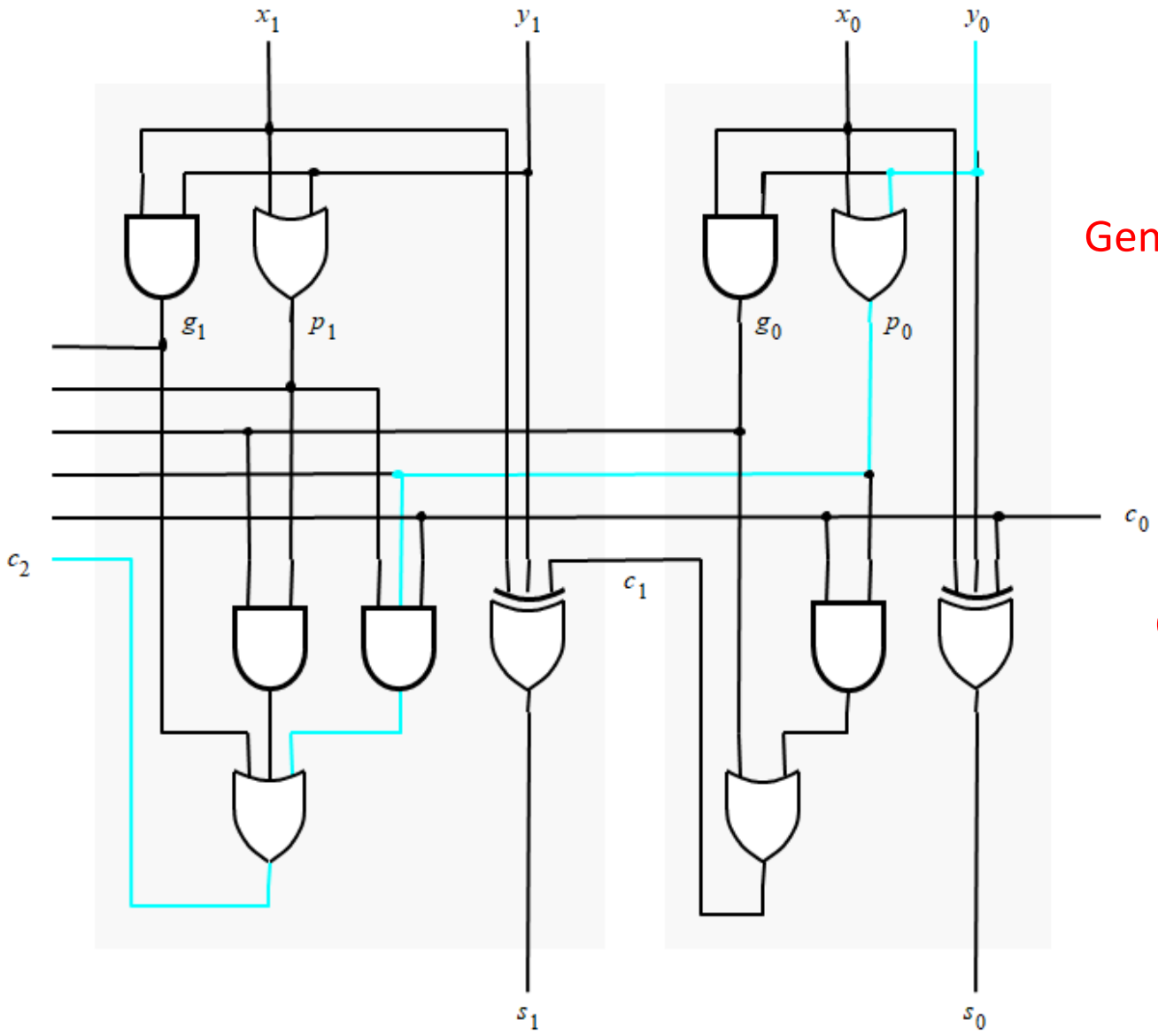
AND-OR networks

If C_0 and all G_i and P_i terms are available at the same time,

ALL C_i terms will be ready after **2 gate delays**

No Ripple Effect!

2-bit Carry Lookahead Adder

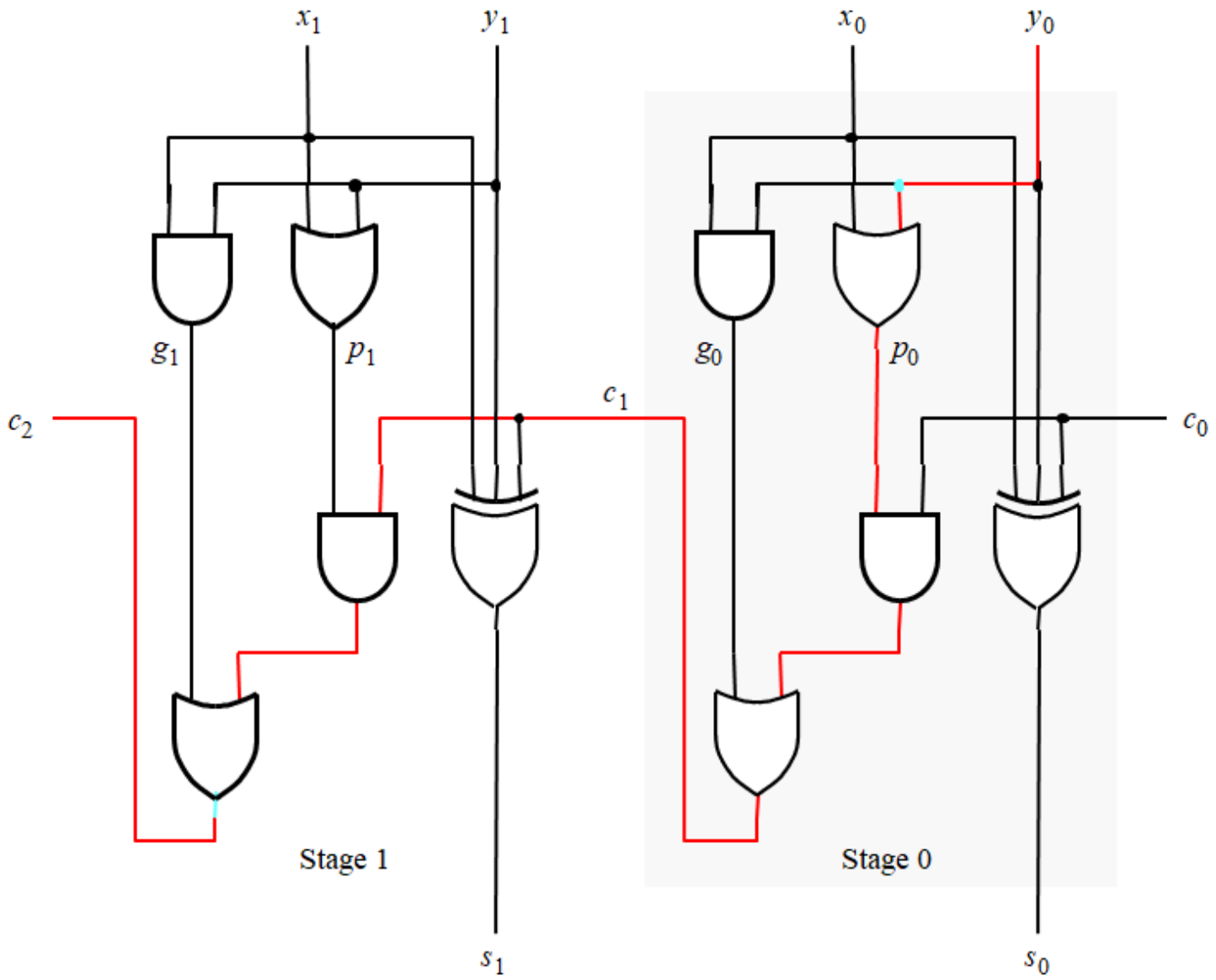


Generate/Propagate: 1 t_{pd}

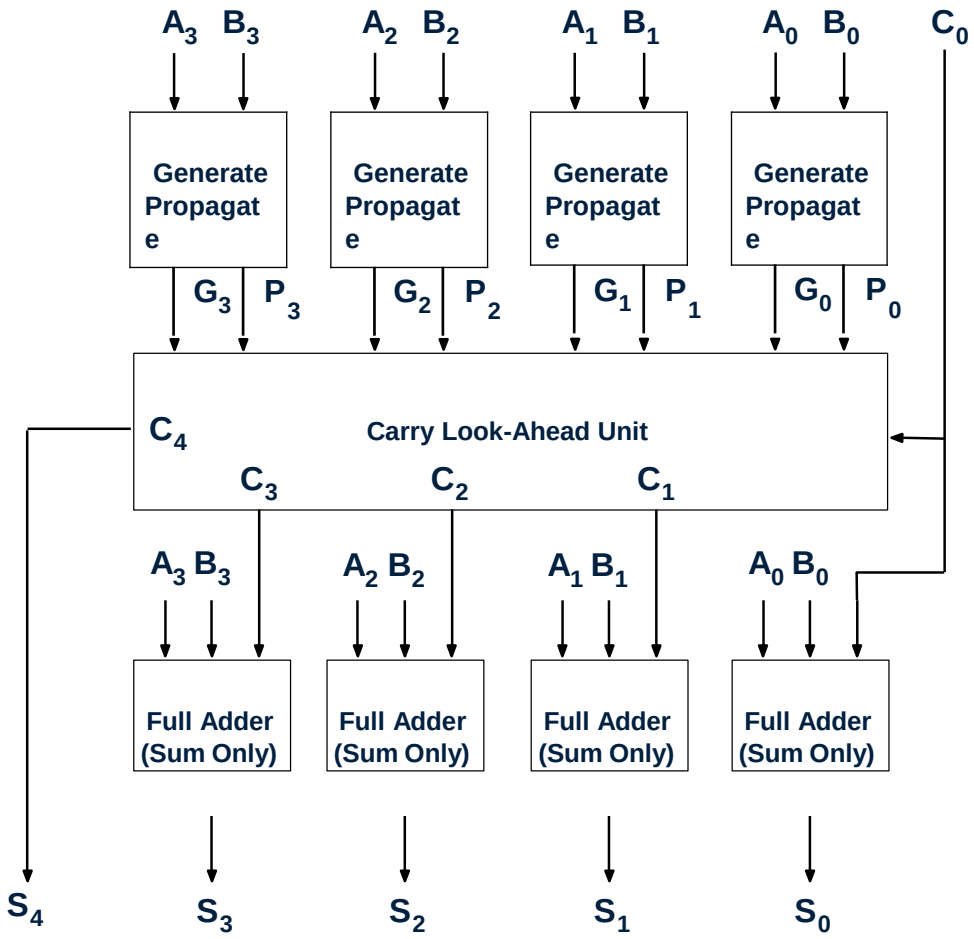
CLA Logic: 2 t_{pd}

Sum: 1 t_{pd}

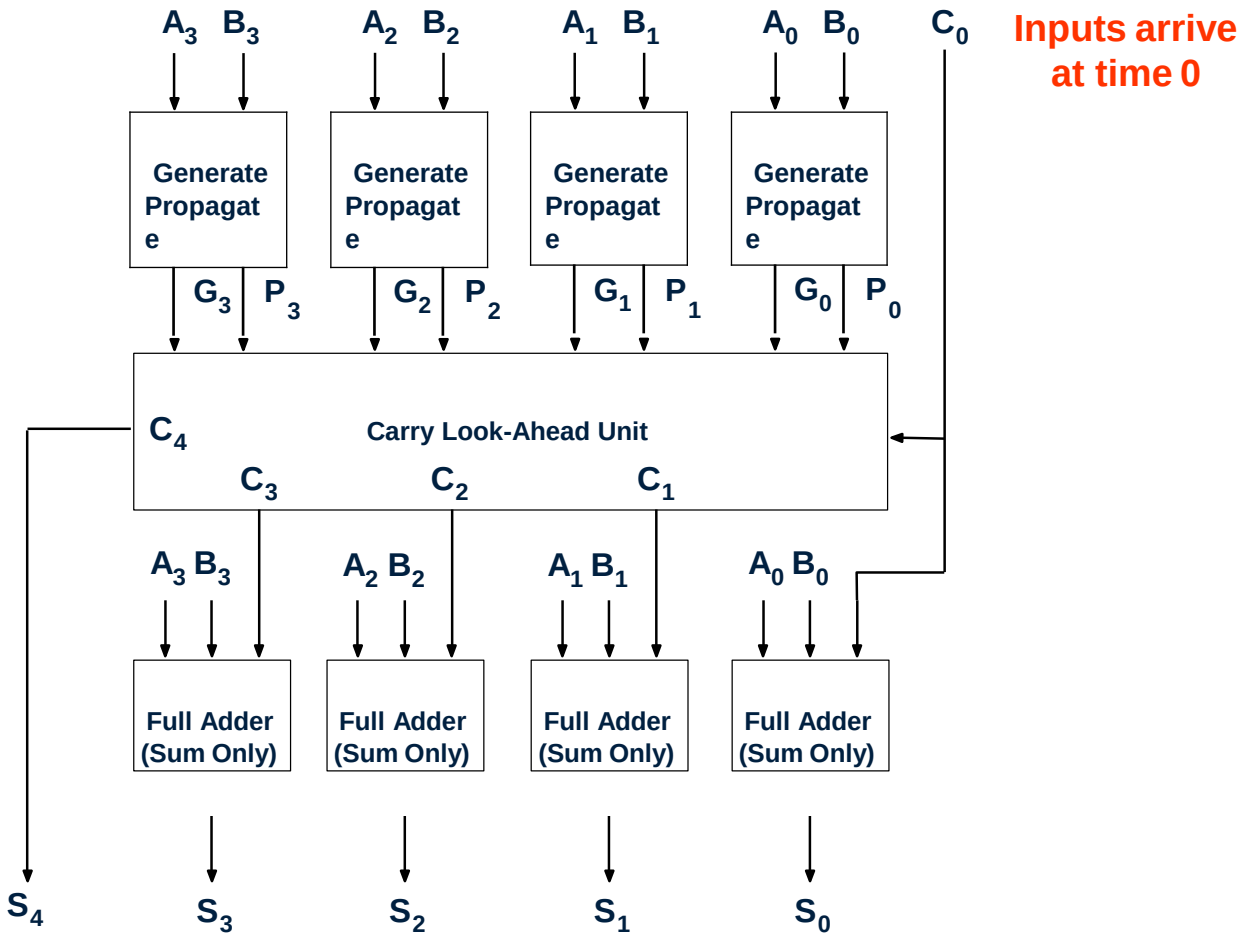
Compare: A ripple-carry adder



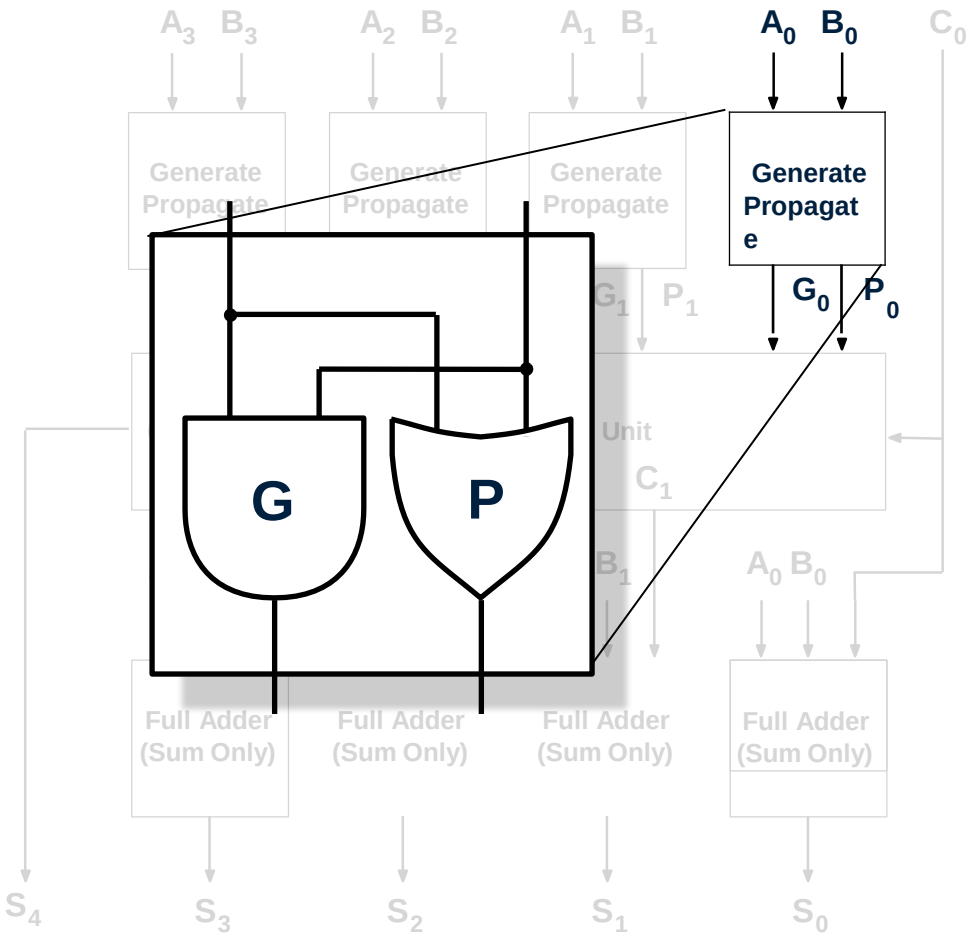
4-bit Carry Lookahead Adder



Overall Critical Path Delay



Overall Critical Path Delay

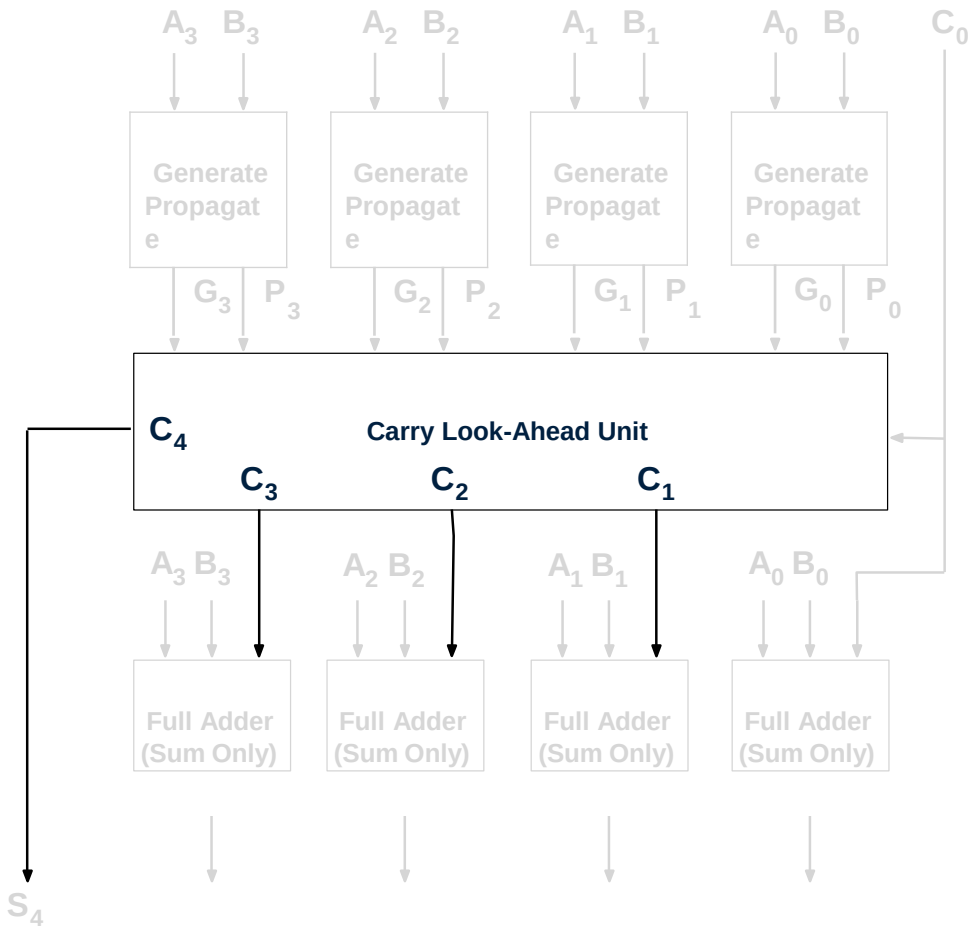


Inputs arrive at time 0

t_{PD}

All G, P terms available after single gate delay

Overall Critical Path Delay



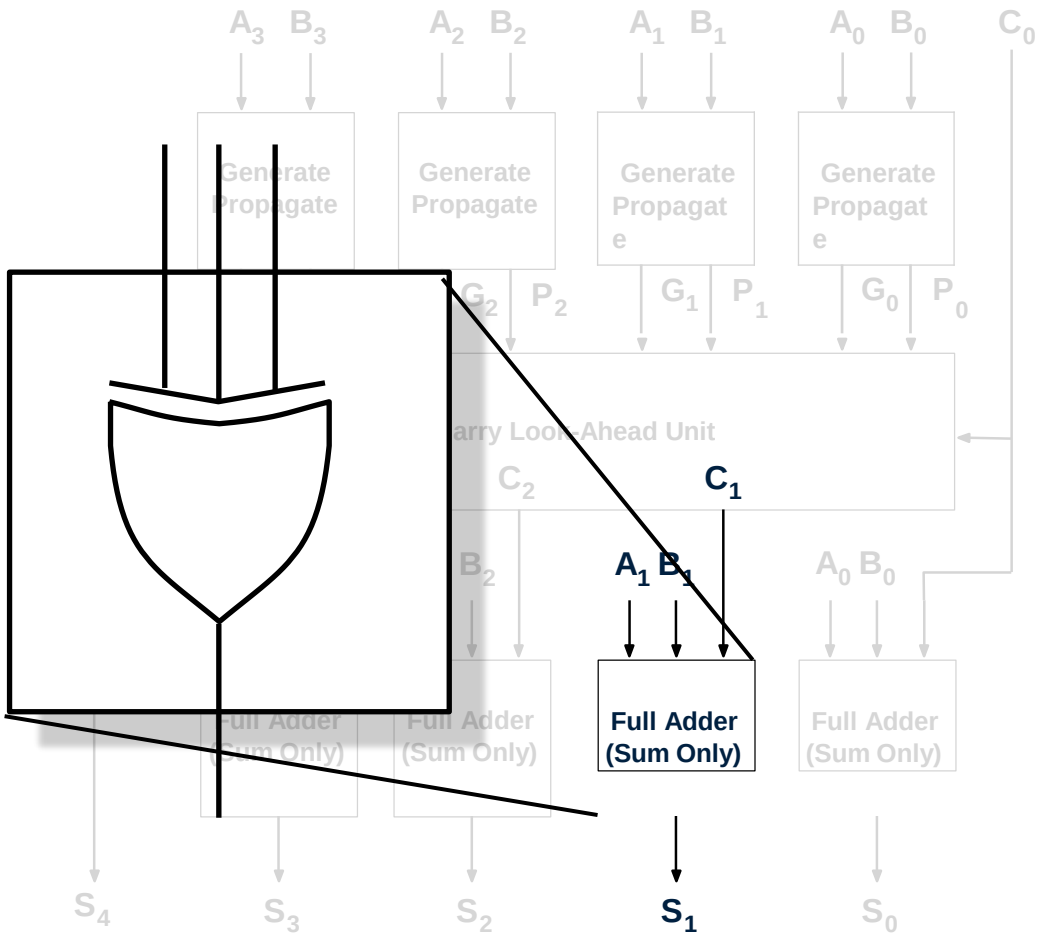
Inputs arrive at time 0

t_{PD}

+

$2\ t_{PD}$

Overall Critical Path Delay



Inputs arrive at time 0

t_{PD}

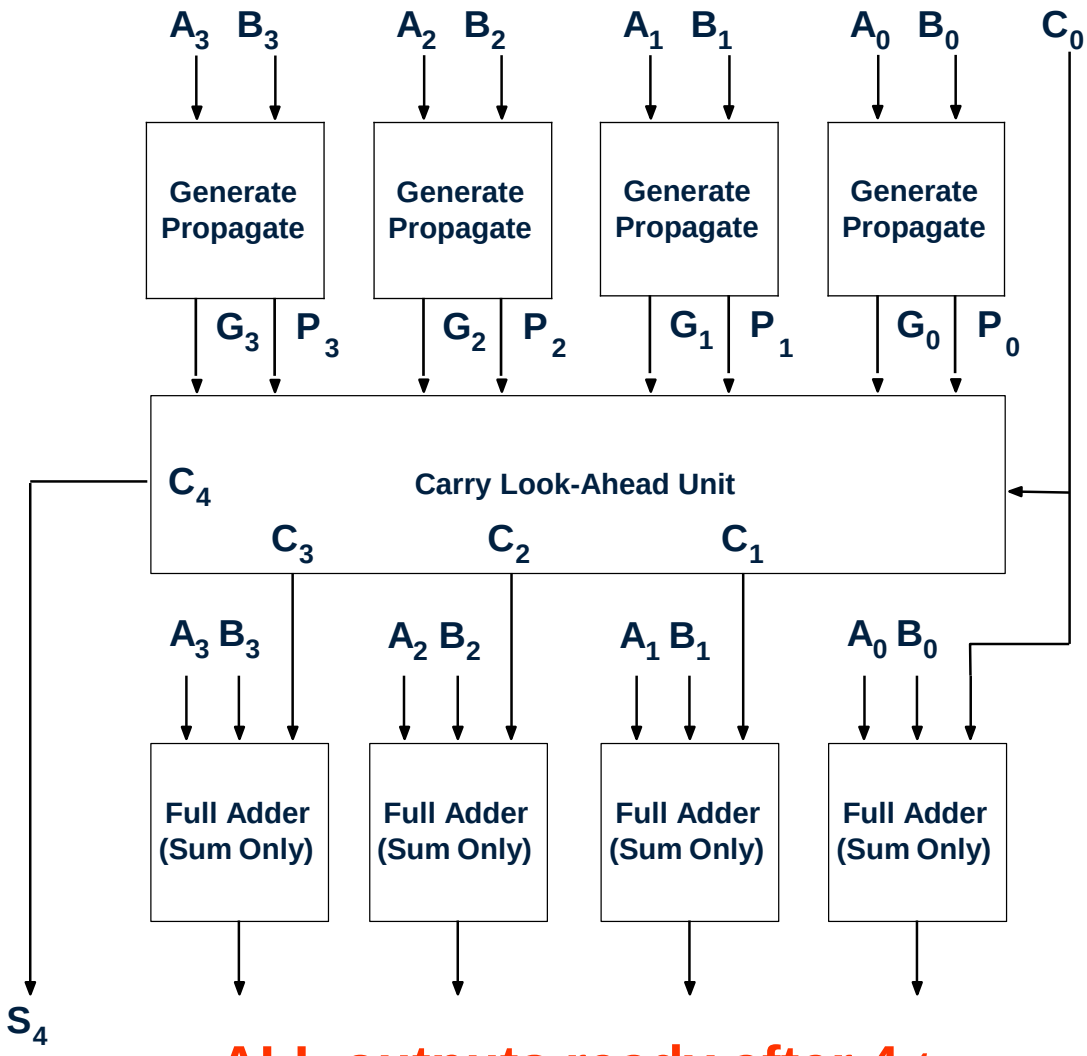
+

$2\ t_{PD}$

+

t_{PD}

CLA: Overall Critical Path Delay



Inputs arrive at time 0

t_{PD}

+

$2\ t_{PD}$

+

t_{PD}

ALL outputs ready after 4 t_{PD}

The total delay in the n-bit carry-lookahead adder is four gate delays

CLA: Overall Critical Path Delay

ALL outputs ready after 4 t_{PD}

The total delay in the n-bit carry-lookahead adder is four gate delays

But there are still few wrinkles to iron out…

- What about CLA Scalability?

In theory, we could build Carry Look-Ahead Adders of any size N

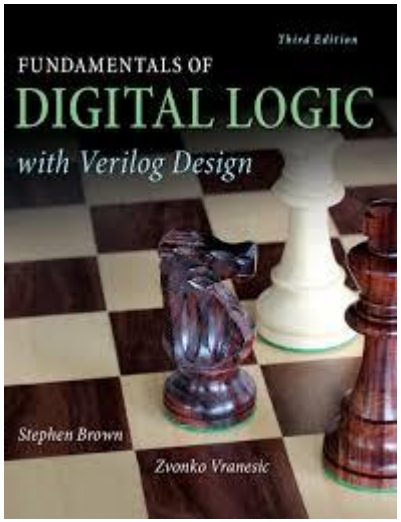
However, equations get more complex very quickly, and we need wider and wider gates (slow) in the carry logic.

$$C_1 = G_0 + P_0 \cdot C_0$$
$$C_2 = G_1 + P_1 \cdot G_0 + P_1 \cdot P_0 \cdot C_0$$
$$C_3 = G_2 + P_2 \cdot G_1 + P_2 \cdot P_1 \cdot G_0 + P_2 \cdot P_1 \cdot P_0 \cdot C_0$$
$$C_4 = G_3 + P_3 \cdot G_2 + P_3 \cdot P_2 \cdot G_1 + P_3 \cdot P_2 \cdot P_1 \cdot G_0 + P_3 \cdot P_2 \cdot P_1 \cdot P_0 \cdot C_0$$

Typically do not extend beyond 4-bits due to fan-in constraints

Recommended Reading

- Digital System Design with Verilog HDL, 3/e, b **S**tephen Brown and **Z**vonko Vranesic. [**S&Z**]
  - S&Z,
    - Chapter-3
      - 3.4



THANK YOU