# CPEG 422/622
# EMBEDDED SYSTEMS DESIGN

Chengmo Yang

chengmo@udel.edu

Evans 201C

# LECTURE 4
# CLA, OVERFLOW

# OUTLINE

- Last lecture:
  Addition/Subtraction

- This lecture:
  Project 1
  More on overflow, adder, CLA

# PROJECT 1

1. 24-bit carry ripple adder/subtractor 25%
2. 24-bit CLA adder/subtractor 25%
3. 24-bit adder/subtractor testbench 25%
4. Final report 25%

- Due dates: 3/6 at midnight (VHDL design code and testbench)
  3/9 at midnight (Final report)

# PROJECT 1

- Entity definition:
  You are **required** to use the given top-level entity for 24-bit adders.

```
entity addsub_24 is
  Port (
     A :       in STD_LOGIC_VECTOR (23 downto 0);
     B :       in STD_LOGIC_VECTOR (23 downto 0);
     M:        in STD_LOGIC;
     S :       out STD_LOGIC_VECTOR (23 downto 0);
     Overflow : out STD_LOGIC
     );
end addsub_24;
```

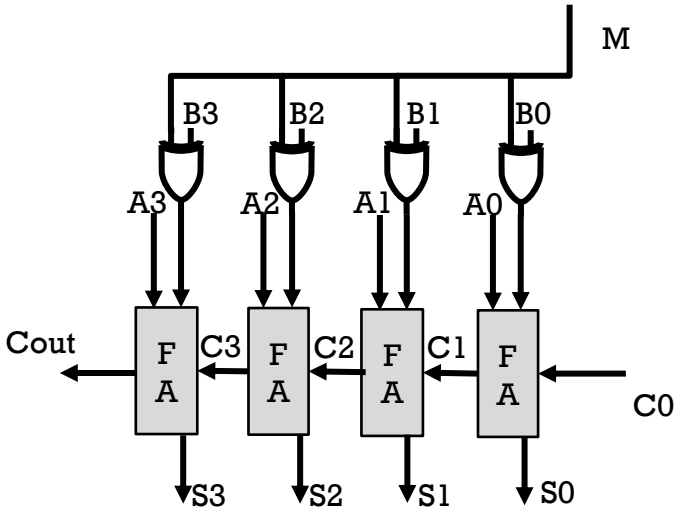
# 4-BIT CARRY-RIPPLE ADDER/SUBTRACTOR

- Adder/subtractor design:

Full adder code with XOR gate

- Control by M

M=1 subtraction; M=0 addition.




4-bit ripple adder /subtractor

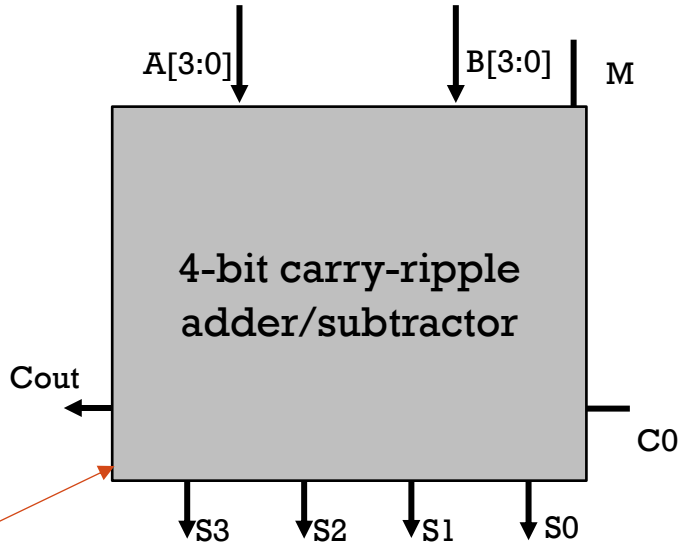
# 4-BIT CARRY-RIPPLE ADDER/SUBTRACTOR

- Adder/subtractor design:

Full adder code with XOR gate

- Control by M

M=1 subtraction; M=0 addition.




4-bit ripple adder /subtractor

*The 4-bit carry-ripple adder/subtractor is the component of the 24-bit carry-ripple adder/subtractor.*

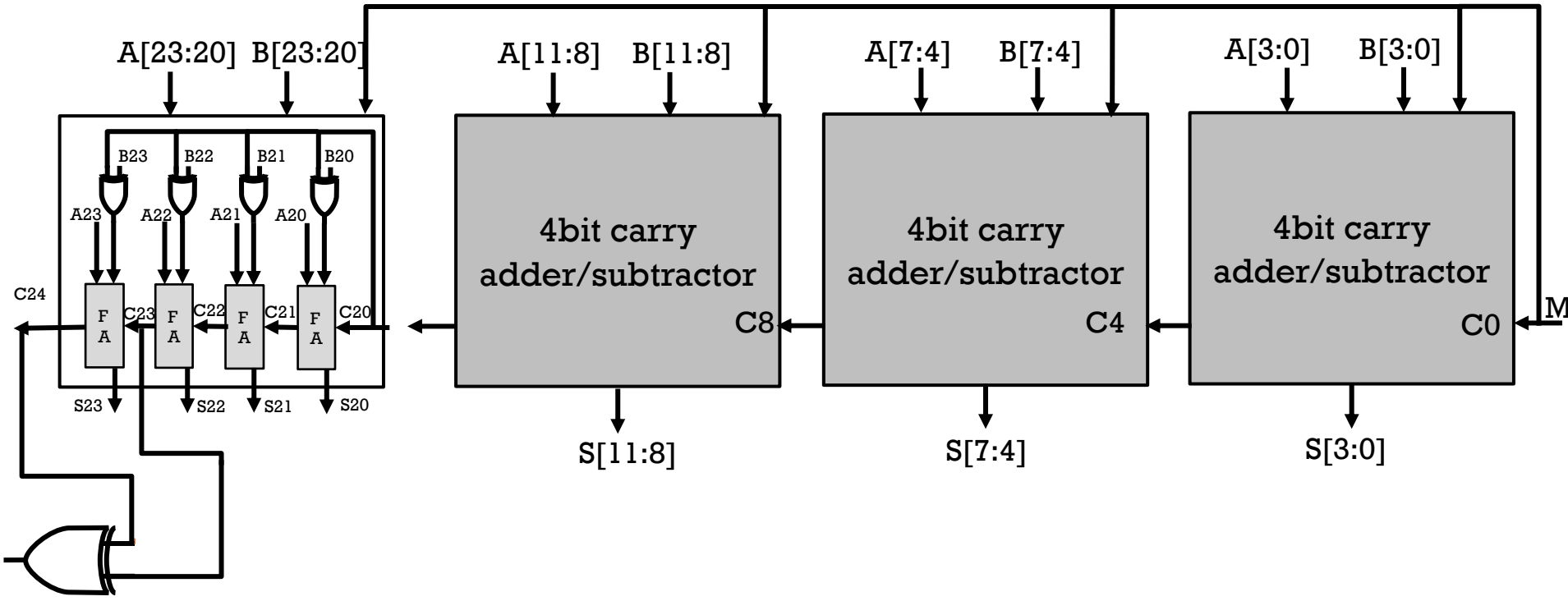
# VHDL PORT NAMES

```
entity ripple24_addsub is
Port
    (
    A, B: in STD_LOGIC_VECTOR (23 downto 0);
    M: in STD_LOGIC;
    S : out STD_LOGIC_VECTOR (23 downto 0);
    Overflow : out STD_LOGIC
    );
end ripple24_addsub;

architecture ripple24_addsub of ripple24_addsub is

component ripple4_addsub
Port
    (
    A, B: in STD_LOGIC_VECTOR (3 downto 0);
    Cin: in STD_LOGIC;
    S : out STD_LOGIC_VECTOR (3 downto 0);
    Cout: out STD_LOGIC
);
end component;

begin
```



- The same input names A and B and same output name S for the ports of the 24-bit adder and 4-bit adder are used.
- This does not pose a problem in VHDL since they refer to different levels.

# 24-BIT CARRY-RIPPLE ADDER/SUBTRACTOR




overflow ← *What is overflow?*

# OVERFLOW

- The fundamental reason of overflow is that the result value exceeds the range of representation,
  - i.e., add two **N**-bit number, and the sum is a **N+1**-bit number.
- The exact overflow condition differs for signed vs. unsigned numbers, as they have different ranges of representation.

| # of bits | # of numbers represented | Unsigned # range | Signed # range |
|---|---|---|---|
| 4 | 16 | 0 ~ 15 | -8 ~ 7 |
| 6 | 64 | 0 ~ 63 | -32 ~ 31 |
| 8 | 256 | 0 ~ 255 | -128 ~ 127 |
| n | $2^n$ | 0 ~ $2^n$ - 1 | - $2^{n-1}$ ~ $2^{n-1}$ - 1 |

# OVERFLOW

- The fundamental reason of overflow is that the result value exceeds the range of representation,
  - i.e., add two **N**-bit number, and the sum is a **N+1**-bit number.
- The exact overflow condition differs for signed vs. unsigned numbers, as they have different ranges of representation.

**Example:**

$$\begin{array}{r} (4)_{10} \\ +\ (5)_{10} \\ \hline (9)_{10} \end{array} \Leftrightarrow \begin{array}{r} 0\,1\,0\,0 \\ +\ 0\,1\,0\,1 \\ \hline 1\,0\,0\,1 \end{array}$$

If unsigned, no overflow

V.S.

$$\begin{array}{r} 0\,1\,0\,0 \\ +\ 0\,1\,0\,1 \\ \hline 1\,0\,0\,1 \end{array}$$

If signed, $(1001)_2$ is -7 not 9, overflow

# OVERFLOW CHECKING FOR SIGNED NUMBER

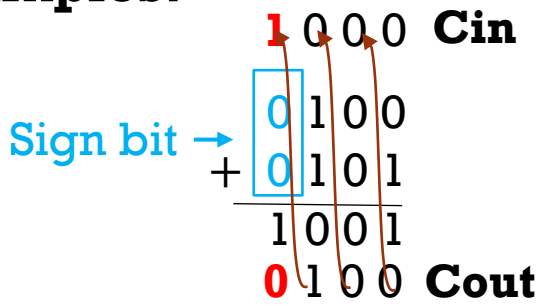
- We can detect overflow by checking the sign bit of the sum.

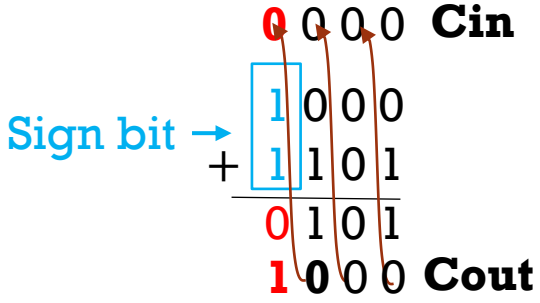
| Operation | Operand A | Operand B | Overflow if sum |
|---|---|---|---|
| A+B | >=0 | >=0 | <0 |
| A+B | <0 | <0 | >=0 |
| A-B | >=0 | <0 | <0 |
| A-B | <0 | >=0 | >=0 |

| Operation | Sign of A | Sign of B | Sign of sum | Carry in of sign bit | Carry out of sign bit |
|---|---|---|---|---|---|
| A+B | 0 | 0 | 1 | 1 | 0 |
| A+B | 1 | 1 | 0 | 0 | 1 |
| A-B | 0 | 1 | 1 | 1 | 0 |
| A-B | 1 | 0 | 0 | 0 | 1 |

# OVERFLOW CHECKING FOR SIGNED NUMBER

**Examples:**

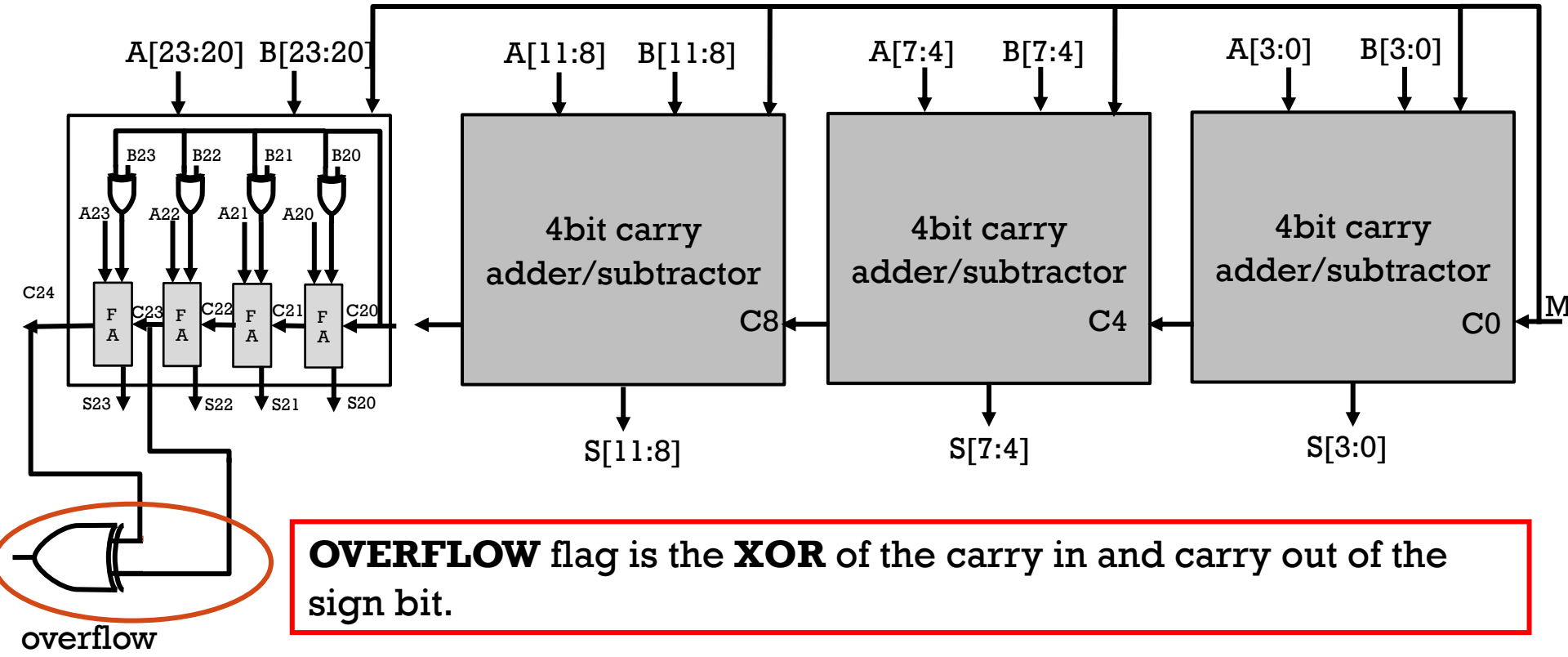



**OVERFLOW** flag = **XOR** of the carry in and carry out of sign bit.

| Operation | Sign of A | Sign of B | Sign of sum | Carry in of sign bit | Carry out of sign bit |
|---|---|---|---|---|---|
| A+B | 0 | 0 | 1 | 1 | 0 |
| A+B | 1 | 1 | 0 | 0 | 1 |
| A-B | 0 | 1 | 1 | 1 | 0 |
| A-B | 1 | 0 | 0 | 0 | 1 |

# OVERFLOW DETECTION




**OVERFLOW** flag is the **XOR** of the carry in and carry out of the sign bit.

# CLA LOGIC REVIEW

In CLA, we defined:

**$G_i$ ≡ Carry generate**
**$P_i$ ≡ Carry propagate**

} *These only depend on the inputs*

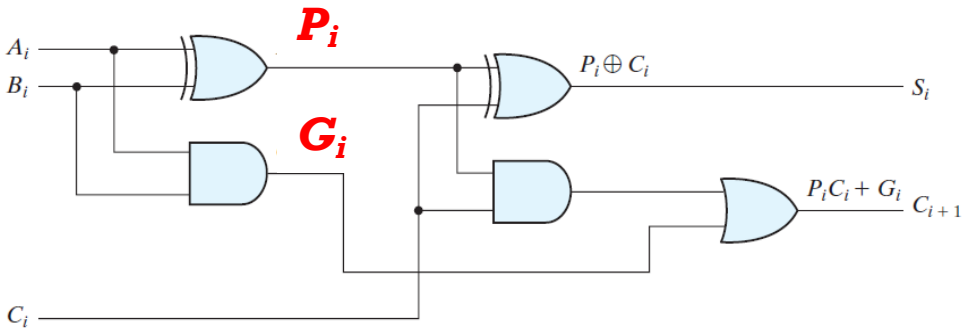
Therefore:

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

*If $G_i = 1$, then $C_{i+1} = 1$*

*If $P_i = 1$, then $C_i$ propagates to $C_{i+1}$*

**Full adder**



# SINGLE-LEVEL CLA

**4bit CLA implementation**

From the functions

$C_3 = G_2 + P_2G_1 + P_2P_1G_0 + P_2P_1P_0C_0$

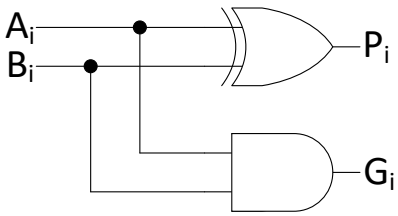
$C_2 = G_1 + P_1G_0 + P_1P_0C_0$

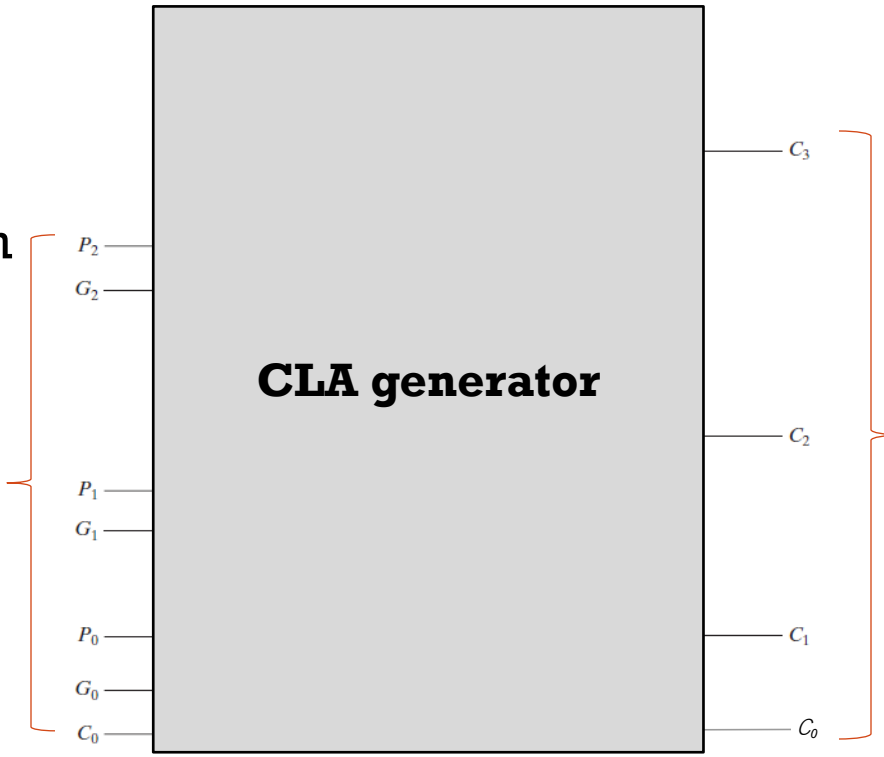
$C_1 = G_0 + P_0C_0$

$C_0 = C_0$

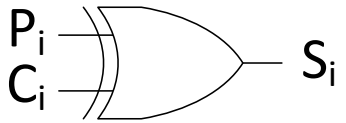
These are all **sum of products** form!

These come from the half adders

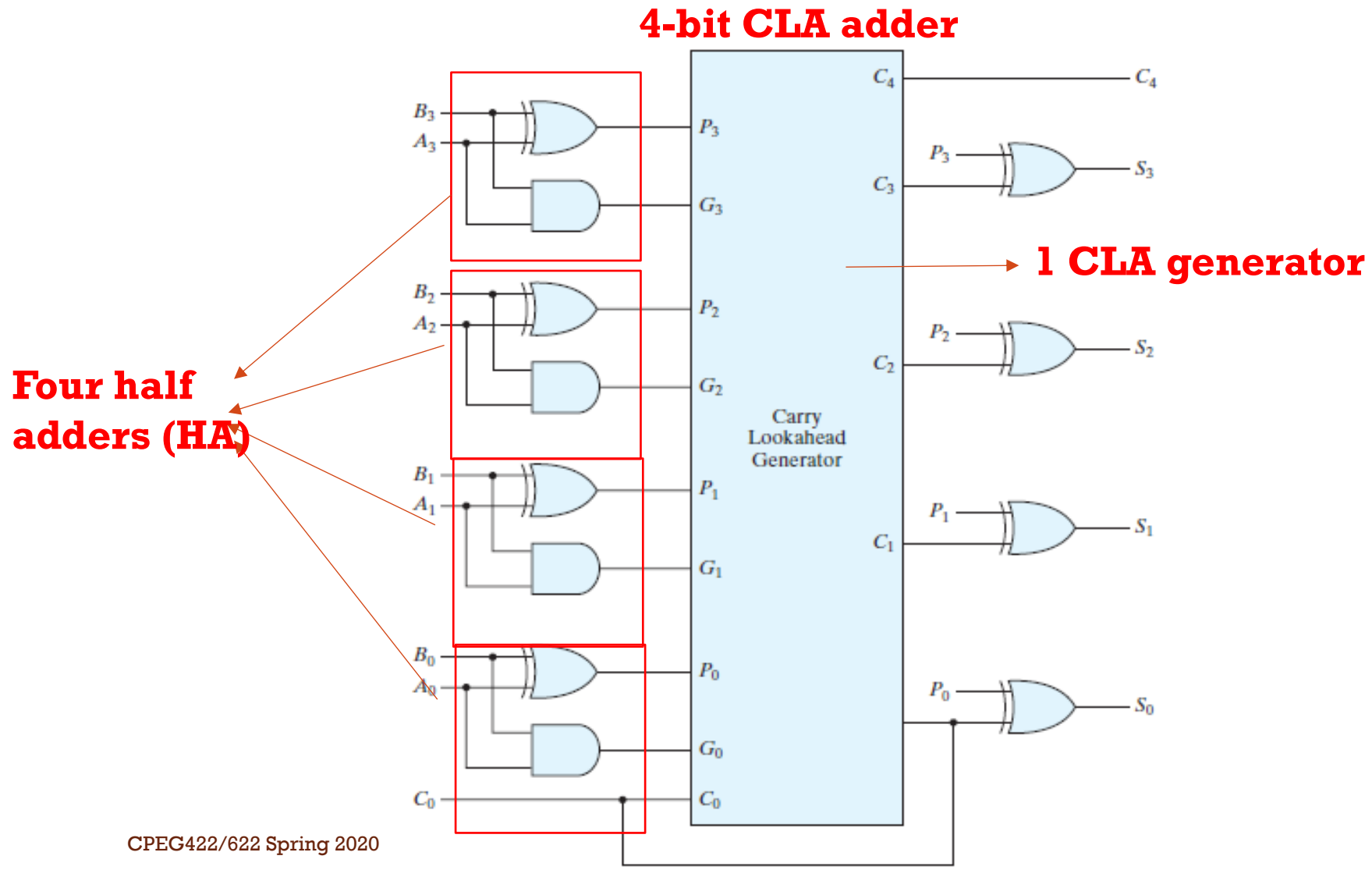


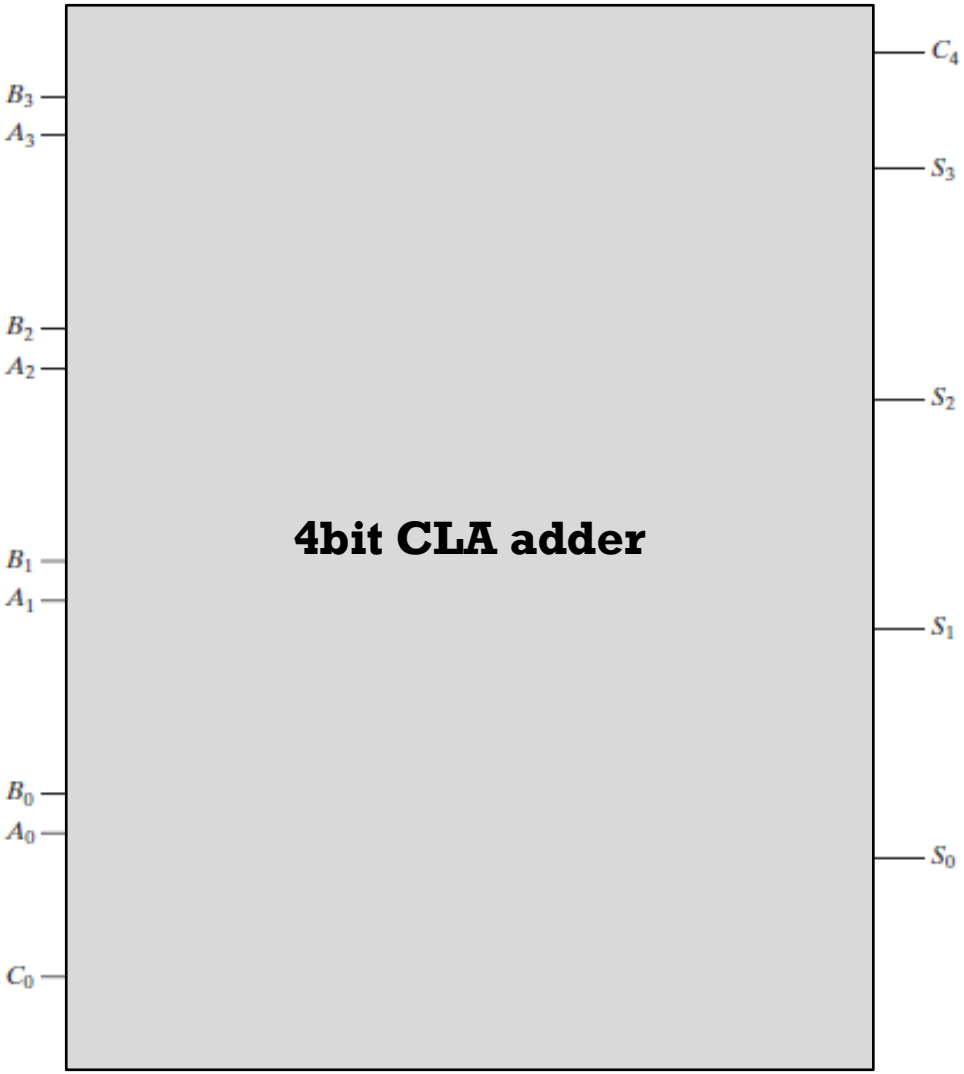

These enter XOR gates to compute $S_i$



# 4-BIT CLA ADDER



# 4-BIT CLA ADDER

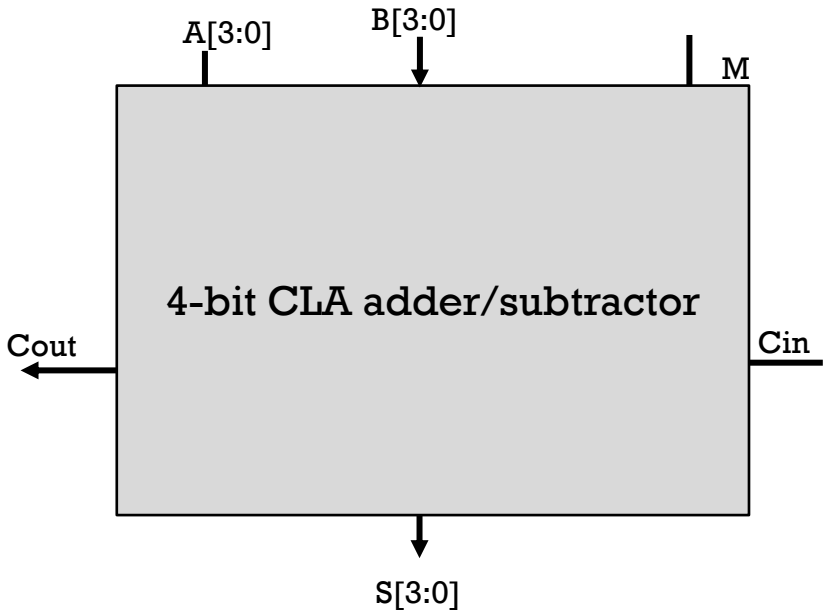


# 4-BIT CLA ADDER/SUBTRACTOR

CLA adder ➡ CLA adder/subtractor?

*The same as before!*

*Use XOR gates and an M signal.*




4-bit CLA adder/subtractor

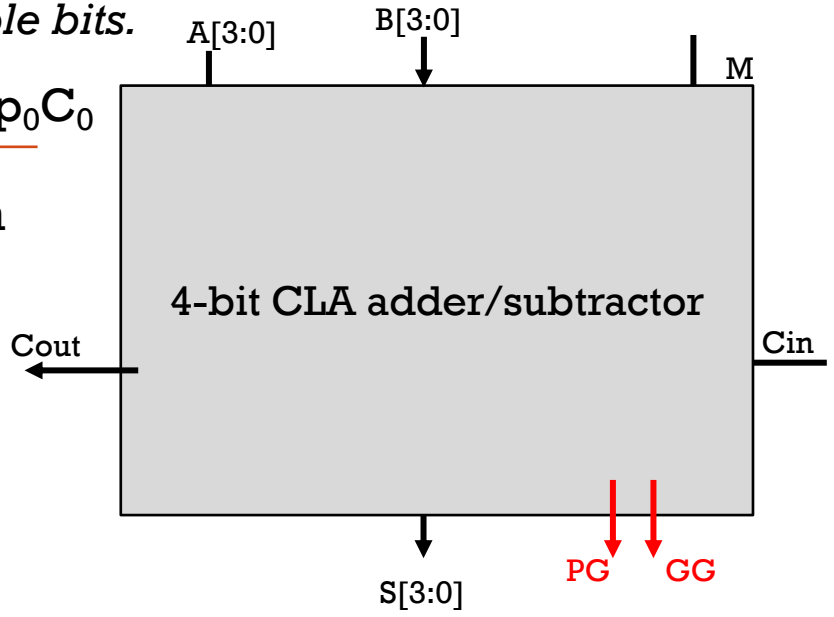
# TWO-LEVEL CLA

*Q: Why not just use single level CLA?*

*A: Too many logic fanin and fanout for multiple bits.*

$$C_4 = g_3 + p_3g_2 + p_3p_2g_1 + p_3p_2p_1g_0 + p_3p_2p_1p_0C_0$$

Image $C_8$, $C_{12}$ would be crazy complex in single level expression!




For second-level CLA, Let's define:

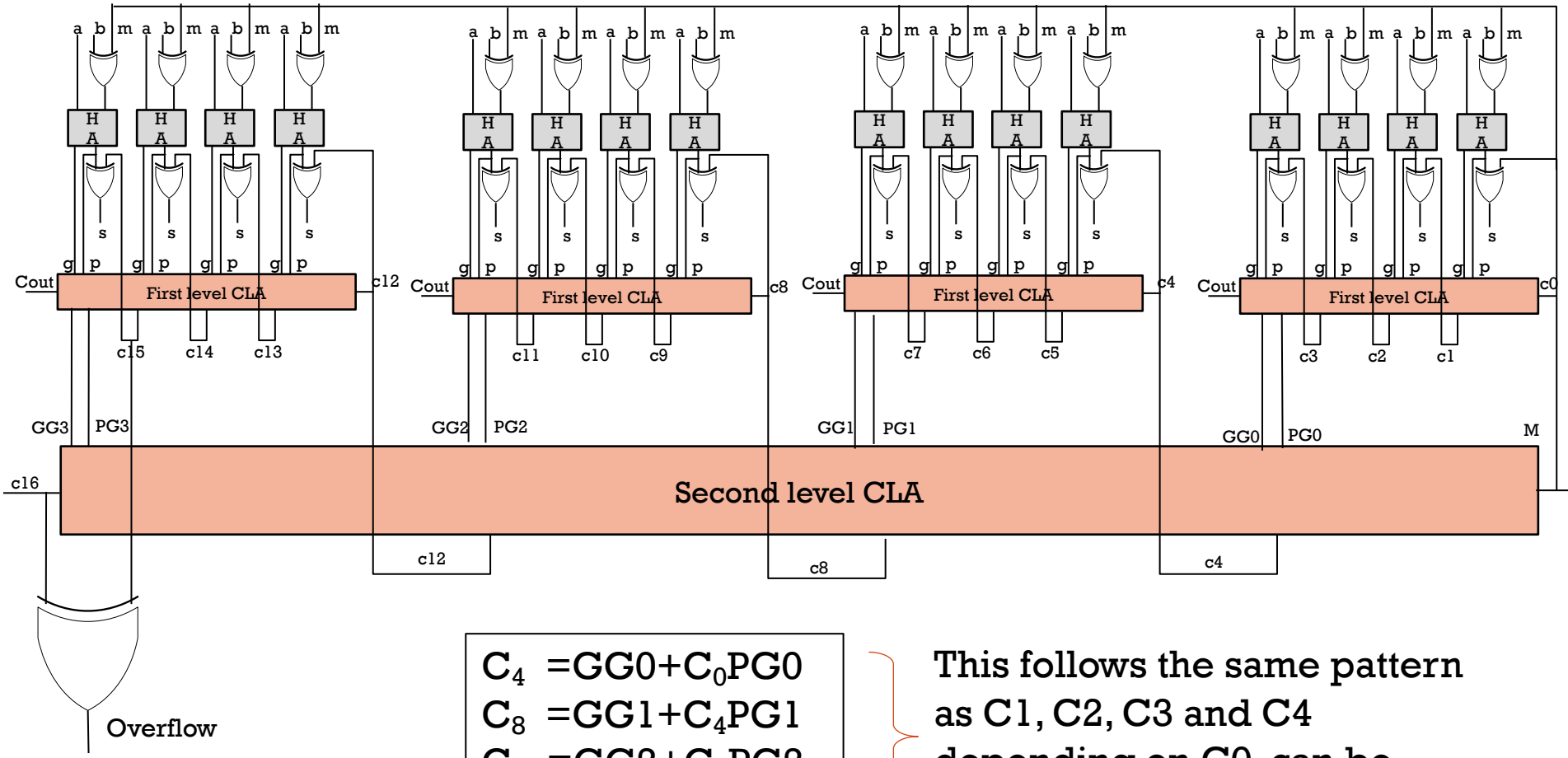
$$PG = p_3p_2p_1p_0$$
$$GG = g_3 + p_3g_2 + p_3p_2g_1 + p_3p_2p_1g_0$$

Therefore:

$$C_{out} = GG + C_{in}PG$$

*PG & GG are inputs to next level CLA.*

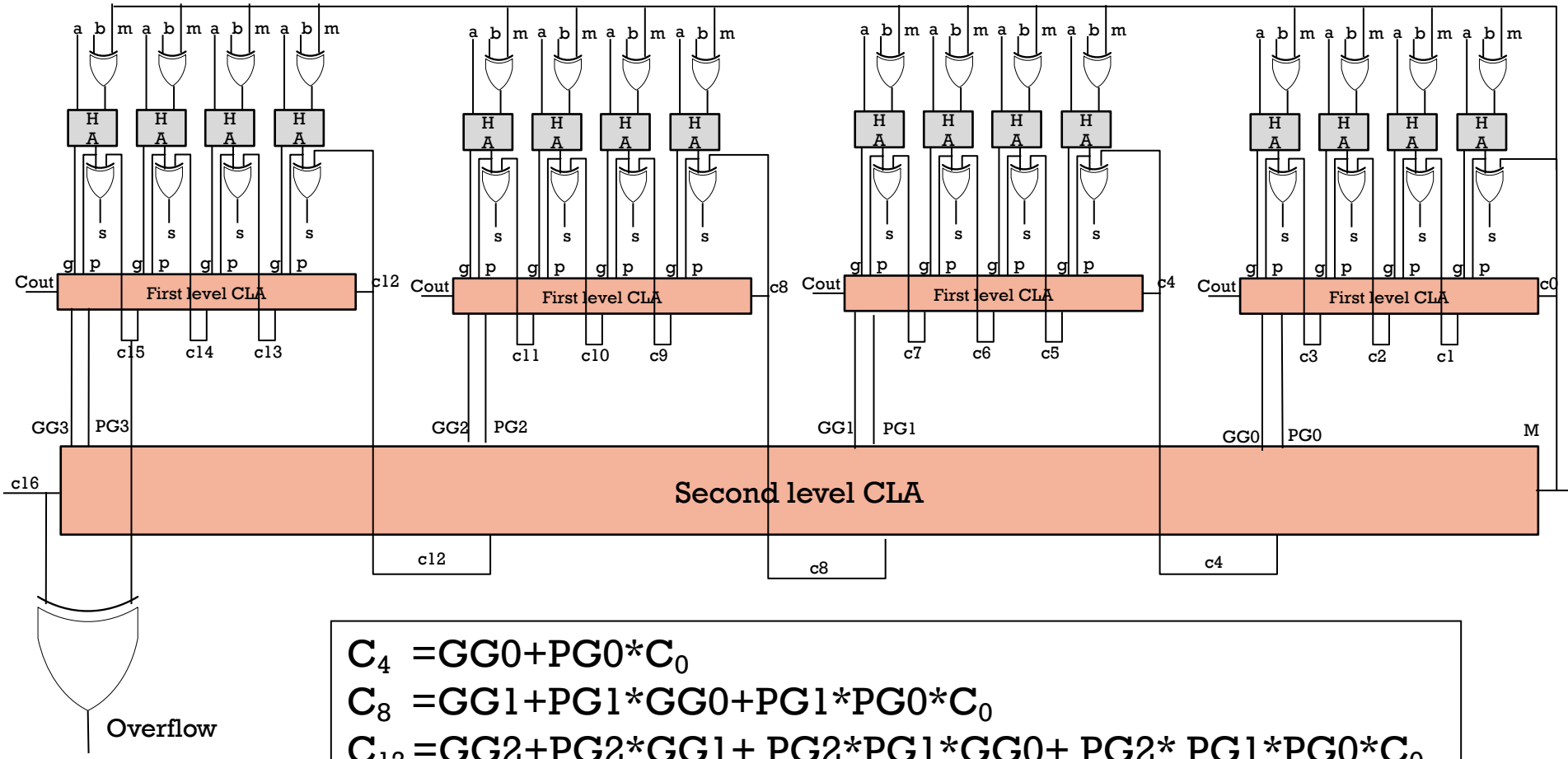
# TWO-LEVEL CLA ADDER/SUBTRACTOR




$C_4 = GG0 + C_0 PG0$
$C_8 = GG1 + C_4 PG1$
$C_{12} = GG2 + C_8 PG2$
$C_{16} = GG3 + C_{12} PG3$

This follows the same pattern as C1, C2, C3 and C4 depending on C0, can be expended as well

# TWO-LEVEL CLA ADDER/SUBTRACTOR




$C_4 = GG0 + PG0 * C_0$

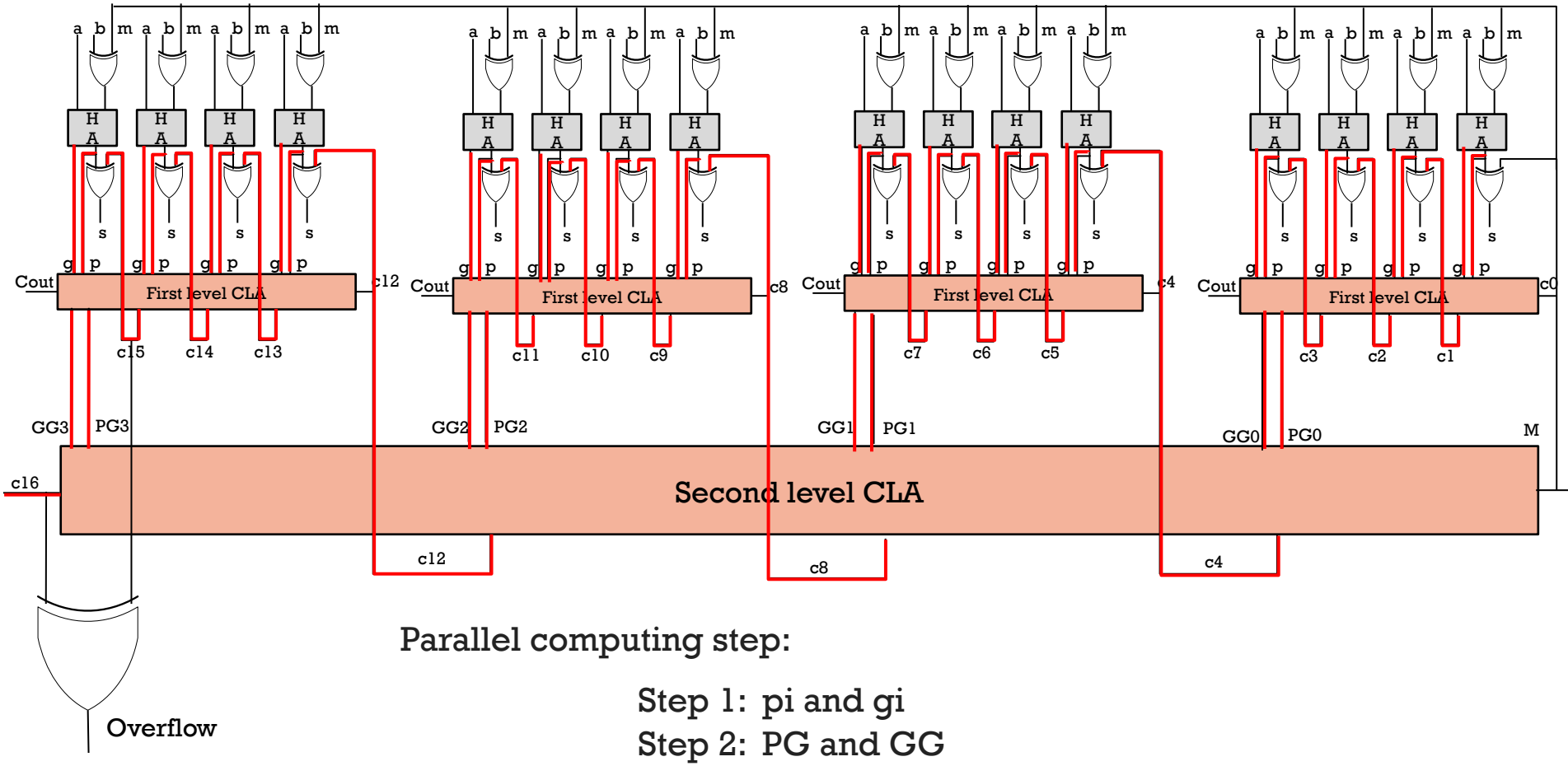
$C_8 = GG1 + PG1 * GG0 + PG1 * PG0 * C_0$

$C_{12} = GG2 + PG2 * GG1 + PG2 * PG1 * GG0 + PG2 * PG1 * PG0 * C_0$

$C_{16} = GG3 + PG3 * C_{12}$

No expending of $C_{16}$ as it is not on critical path

# TWO-LEVEL CLA ADDER/SUBTRACTOR




Parallel computing step:

Step 1: pi and gi
Step 2: PG and GG
Step 3: C4, C8, C12
Step 4: other Ci
Step 5: S and overflow ( output )

# 24-BIT CLA ADDER/SUBTRACTOR

- A standard two-level CLA ADDER/SUBTRACTOR is 16 bits
- How to design a 24-bit CLA ADDER/SUBTRACTOR?
  - Idea 1: use one 16-bit two-level CLA and two 4-bit one-level CLA
  - Idea 2: use two 12-bit two-level CLA

# FINAL REPORT

- Describe two types of adder/subtractor design structure.
  I. 24-bit ripple
  II. 24-bit CLA
- Give testbench simulation results:
  I. 24-bit ripple
  II. 24-bit CLA
- Show the design report in Vivado, including:
  I. Hardware utilization and schematic
  II. Power
  III. Timing
- Compare 24-bit CLA and ripple adders for cost, power, and timing.

# NEXT LECTURE

- Test bench
- Sequential circuits in VHDL