

ALU Design

⊕ The fundamental operation of the arithmetic is addition.

⊕ All others:

⊕ Subtraction

⊕ Multiplication

⊕ Division

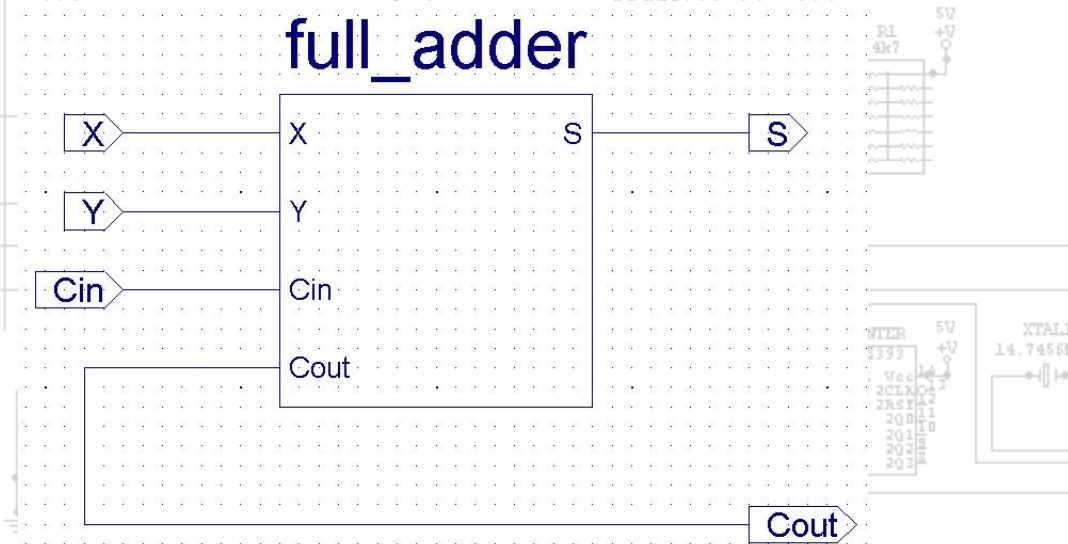
⊕ are implemented in terms of it.

⊕ We need therefore an efficient implementation.

N-bit Ripple-Carry-Adder (RCA)

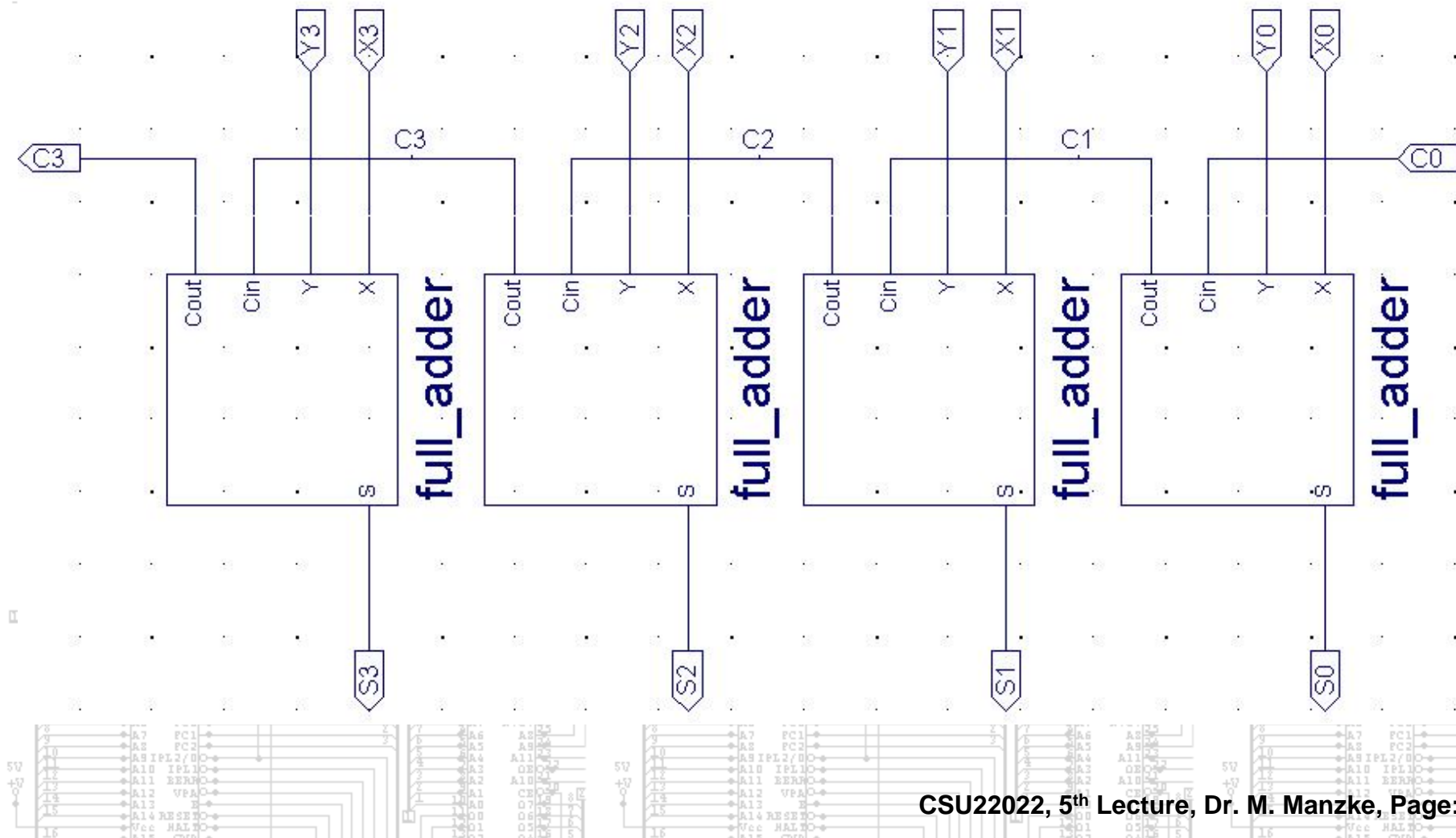
n Full Adders

⊕ An n-bit ripple-carry-adder is constructed from n full-adders



$$S = x \oplus y \oplus C_{in}$$
$$C_{out} = XY + xC_{in} + yC_{in}$$

4-bit Ripple Carry Adder



RCA Equations

$$S_i = x_i \oplus y_i \oplus C_i$$
$$C_{i+1} = x_i y_i + x_i C_i + y_i C_i$$

⊕ Hence, using an AND+wired-OR and n-bit RCA introduces n gate delays.

⊕ For 64 bit calculations this is too slow

Carry Lookahead Boolean Expression

$$C_{i+1} = x_i y_i + C_i(x_i + y_i)$$

$$C_1 = x_0 y_0 + C_0(x_0 + y_0)$$

$$C_2 = x_1 y_1 + C_1(x_1 + y_1)$$

$$= x_1 y_1 + [x_0 y_0 + C_0(x_0 + y_0)](x_1 + y_1)$$

with $g_i = x_i y_i$

Generate Carry

$$p_i = x_i + y_i$$

Carry Propagate

$$C_{i+1} = g_i + p_i C_i$$

Carry Lookahead Boolean Expression

$$C_{i+1} = g_i + p_i C_i$$

$$C_1 = x_0 y_0 + C_0(x_0 + y_0)$$

$$= g_0 + C_0 p_0$$

$$C_2 = x_1 y_1 + C_1(x_1 + y_1)$$

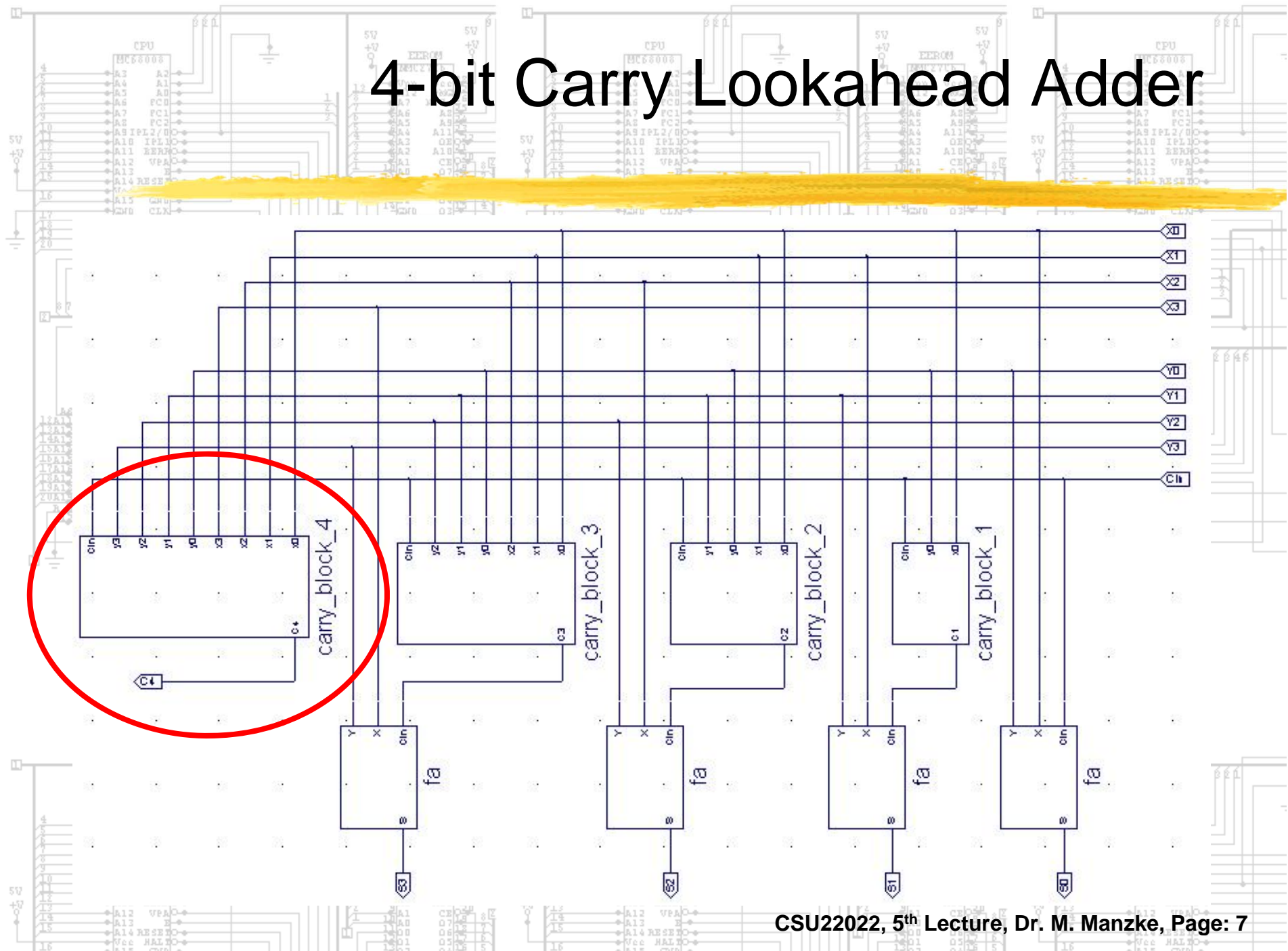
$$= x_1 y_1 + [x_0 y_0 + C_0(x_0 + y_0)](x_1 + y_1)$$

$$= g_1 + p_1 g_0 + p_0 p_1 C_0$$

$$C_3 = g_2 + p_2 g_1 + p_1 p_2 g_0 + p_0 p_1 p_2 C_0$$

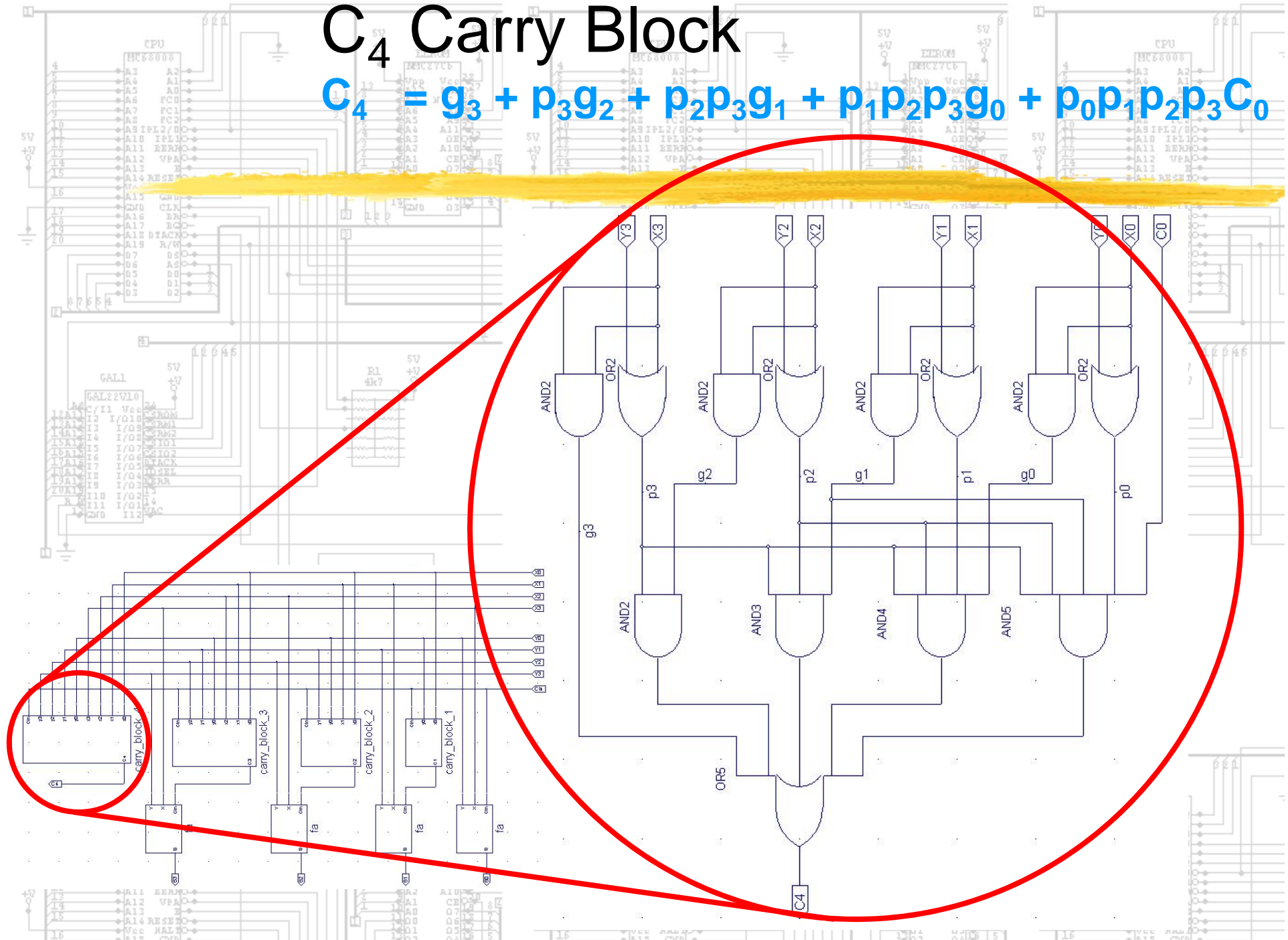
$$C_4 = g_3 + p_3 g_2 + p_2 p_3 g_1 + p_1 p_2 p_3 g_0 + p_0 p_1 p_2 p_3 C_0$$

4-bit Carry Lookahead Adder



C₄ Carry Block

$$C_4 = g_3 + p_3g_2 + p_2p_3g_1 + p_1p_2p_3g_0 + p_0p_1p_2p_3C_0$$



Carry Lookahead Adder

$$C_{i+1} = g_i + p_i g_{i-1} + p_i p_{i-1} g_{i-2} + \dots + p_i p_{i-1} \dots p_0 C_0$$

- ⊕ This requires just two gate delays:
 - ⊕ One to generate g_i and p_i
 - ⊕ Another to AND them
- ⊕ Again we can use wired OR
- ⊕ But, it requires AND gates with a fan in of n
- ⊕ In practice we can only efficiently build single gates with a limited fan-in
- ⊕ we build the lookahead circuit as a multi-level circuit

Groups of Input Bits

⊕ For example, let fan-in = 4 and define:

⊕ G_i' A carry out is generated in the i^{th} group of four input bits

⊕ P_i' A carry out is propagated by the i^{th} group of four input bits

$$G_0' = g_3 + p_3g_2 + p_2p_3g_1 + p_1p_2p_3g_0$$

$$P_0' = p_0p_1p_2p_3$$

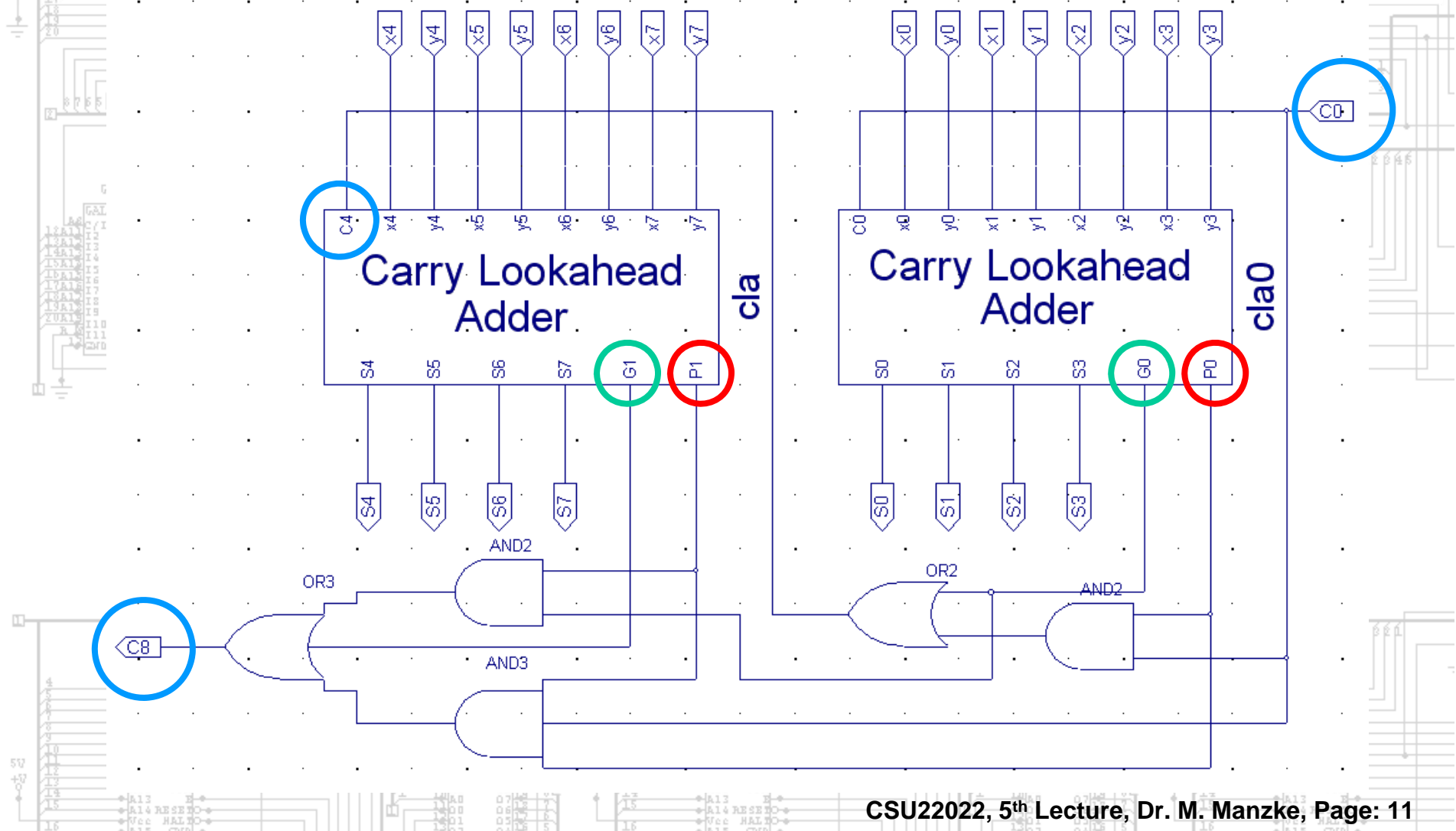
$$C_4 = G_0' + C_0P_0'$$

$$C_8 = G_1' + P_1'G_0' + P_0'P_1'C_0$$

$$C_{12} = G_2' + P_2'G_1' + P_1'P_2'G_0' + P_0'P_1'P_2'C_0$$

$$C_4 = G_0' + C_0 P_0'$$

$$C_8 = G_1' + P_1' G_0' + P_0' P_1' C_0$$



Generate **G''** and Propagate **P''**

⊕ The next level of generate **G''** and propagate **P''** terms will cover 16 bits

$$\begin{aligned} G'' &= G_3' + P_3'G_2' + P_3'P_2'G_1' + P_3'P_2'P_1'G_0 \\ P'' &= P_3'P_2'P_1'P_0 \end{aligned}$$

64-bit Adder Propagation Delay

⊕ We can implement a 64-bit adder using AND-or logic with a fan-in = 4 and a maximum propagation delay of:

$$\begin{aligned} t_{pmax} &= 3(G_1') + 2(G_1'') + 2(C_{48}) + 2(C_{60}) + 3(S_{63}) \\ &= 12 \text{ gate delays} \end{aligned}$$

⊕ Compare this with RCA using AND-wiredOR which requires 64 gate delays.

⊕ If we add a third layer (**G'''**, **P'''**) we can construct a $4 \times 64 = 256$ bit adder with maximum delay:

$$\begin{aligned} t_{pmax} &= 3 + 2 + 2 + 2 + 2 + 2 + 3 \\ &= 16 \text{ gate delays} \end{aligned}$$