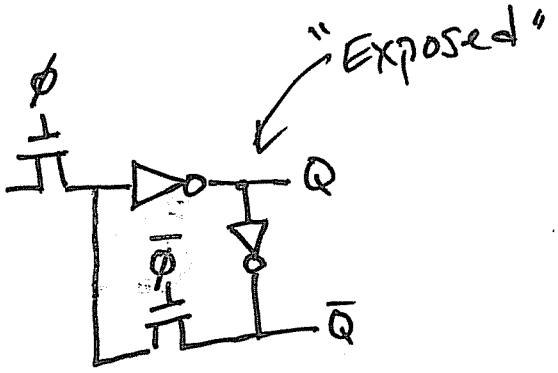
3/23/12 Please turn in your check-ins and lab 6 reports. Reminders:

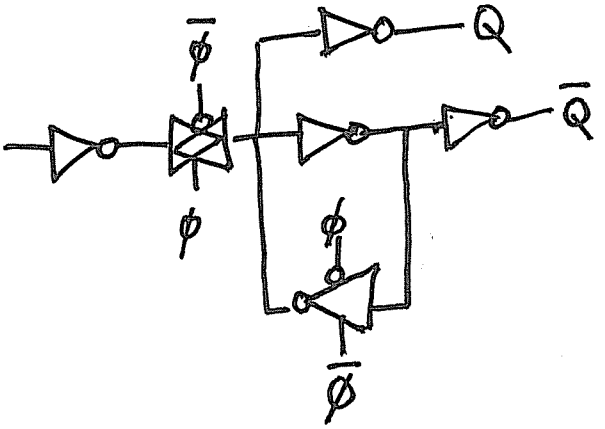
- lab 7 is cancelled
- Final Project Proposal Draft Due Mon.
- Lab 8 progress check off next Fri.
- No lecture next Fri (3/30): LAB DAY

WARM UP: What are the advantages and disadvantages of the latch & flip-flop designs used in modern processes commonly compared to the examples in your text?

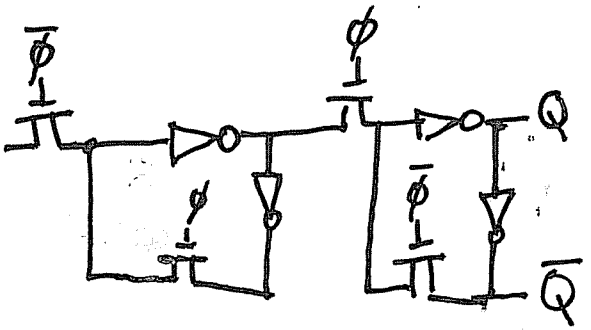
Latch: text example:



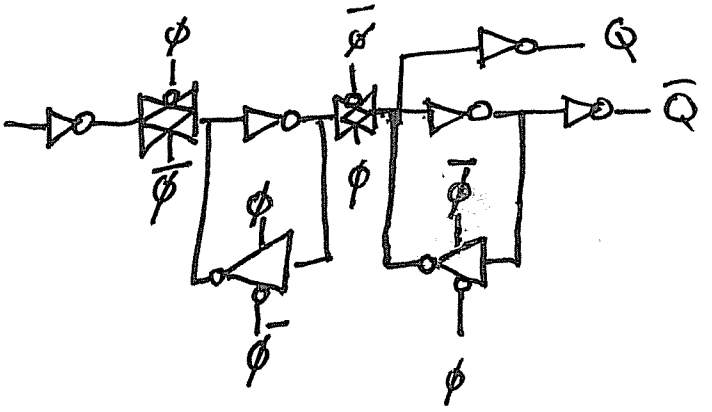
Latch: modern processes:



flop: text example:

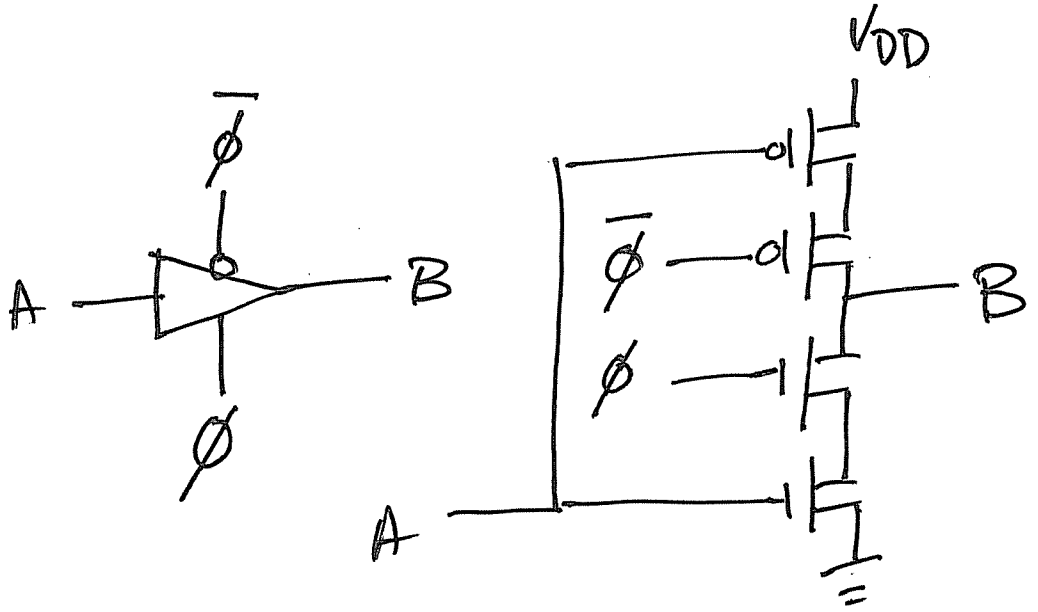


flop: modern process:



DIS ADVANTAGES: takes up more space
more power
longer delay

ADVANTAGES: No threshold voltage drops
No undriven diffusion capacitance
(FET cap. of TG)
Q output protected from noise



12.1 cont.

Last lecture we examined adding 2 binary numbers.

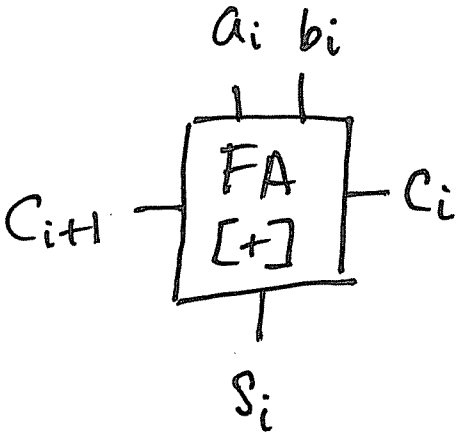
More complicated: adding *n*-bit binary numbers:

$$\begin{array}{r} a_3\ a_2\ a_1\ a_0 \\ b_3\ b_2\ b_1\ b_0 \\ \hline C_4\ S_3\ S_2\ S_1\ S_0 \end{array}$$

For each bit, this results in the **full-adder** equation:

$$\begin{array}{r} c_i \\ a_i \\ b_i \\ \hline c_{i+1}\ s_i \end{array}$$

From which we can design a **full adder**:

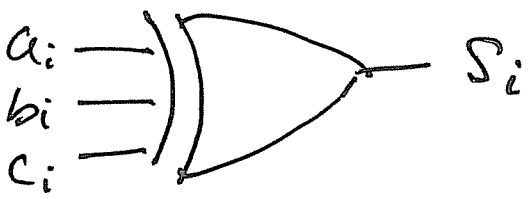


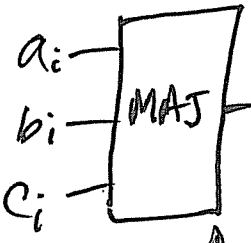
a_i	b_i	c_i	s_i	C_{i+1}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

XOR $\longrightarrow$ $s_i = a_i \oplus b_i \oplus c_i$

MAJ $\longrightarrow$ $C_{i+1} = a_i \cdot b_i + c_i \cdot (a_i \oplus b_i)$

$= a_i + b_i + c_i \cdot (a_i + b_i)$ ← also OK





covered in HW1

Layout in HW3

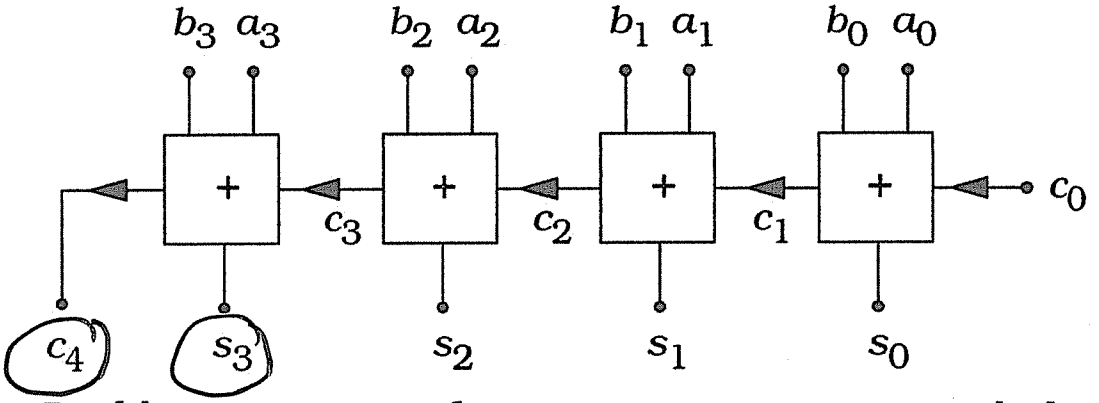
Several other FA designs in your text

12.2 Ripple-Carry Adders

Consider the carry-in bit as a sort of third word in the equation:

$$\begin{array}{cccc} c_3 & c_2 & c_1 & c_0 \\ a_3 & a_2 & a_1 & a_0 \\ b_3 & b_2 & b_1 & b_0 \\ \hline c_4 & s_3 & s_2 & s_1 & s_0 \end{array}$$

The full adders are then put together in a long series:



Problem: carry-out from previous stage is needed to compute the sum in the next stage -> leads to a long delay through the critical path: the last outputs are dependent on every previous carry in bit.

What is the solution? Find a way to compute the c bits more efficiently so that the downstream delays are not so dependent on all the previous calculations.

12.3 Carry Look-Ahead (CLA) Adders

Let's take a closer look at the ways in which carries move through the network:

a_i	b_i	c_i	c_{i+1}	
0	0	0/1	0/0	← Always 0 ⇒ Kill
0	1	0/1	0/1	← $c_{i+1} = c_i$ ⇒ Propagate
1	0	0/1	0/1	
1	1	0/1	1/1	← Always 1 ⇒ generate

Carry-kill (k) - $C_{out} = 0$ independent of C_{in}

Propagate (p) - $C_{out} = C_{in}$

Generate (g) - $C_{out} = 1$ independent of C_{in}

$$P_i = a_i \oplus b_i$$

$$k_i = \overline{a_i} \cdot \overline{b_i} = \overline{(a_i + b_i)}$$

$$g_i = a_i \cdot b_i$$

a b	g	p	c_i	s_i	c_{i+1}
0 0	0	0	0	0	0
0 0	0	0	1	1	0
0 1	0	1	0	1	0
0 1	0	1	1	0	1
1 0	0	1	0	1	0
1 0	0	1	1	0	1
1 1	1	0	0	0	1
1 1	1	0	1	1	1

$$s_i = p_i \oplus c_i$$

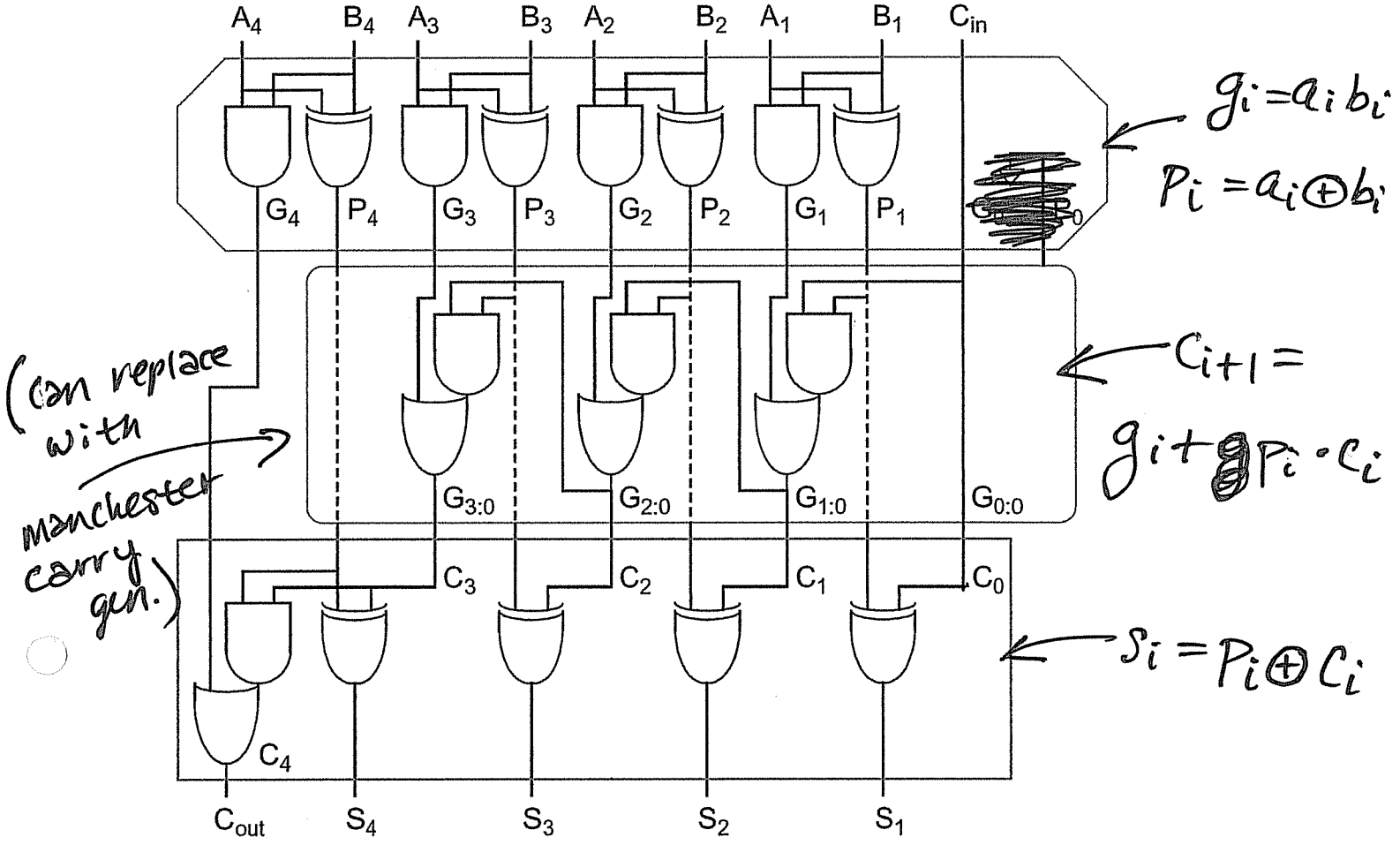
$$c_{i+1} = g_i + p_i \cdot c_i$$

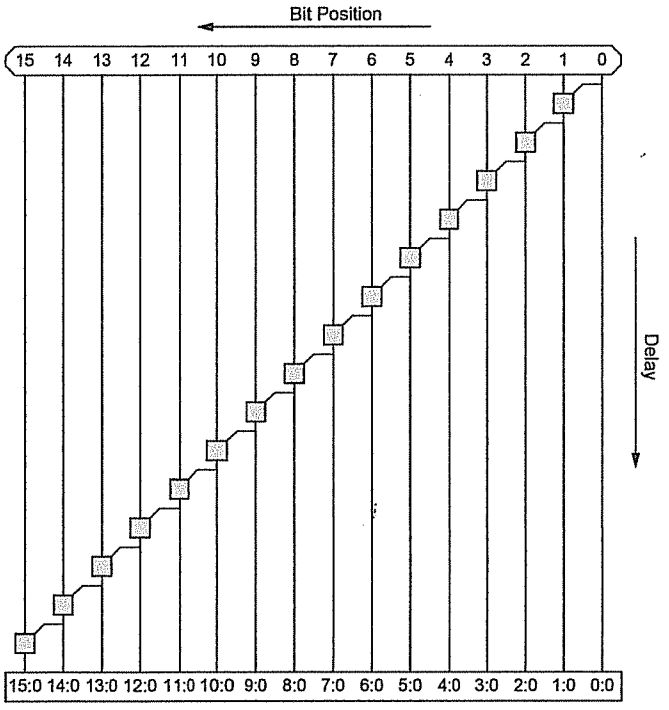
① $g_i = a_i b_i$, $p_i = a_i \oplus b_i$ } independant of carries

② $c_{i+1} = g_i + p_i \cdot c_i$ } one stage at a time

③ $s_i = p_i \oplus c_i$ } do after

Can use these g_i and p_i bits to calculate carries more quickly, and more quickly to send the carry bit to the next stage:





This now calculates s and c more efficiently, but is still a carry-ripple adder where every carry is dependent on all previous stages.

12.3.1 Manchester Carry Chains

Elegant approach to dealing with CLA bits, using a switch-logic implementation of the carry-out equation:

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

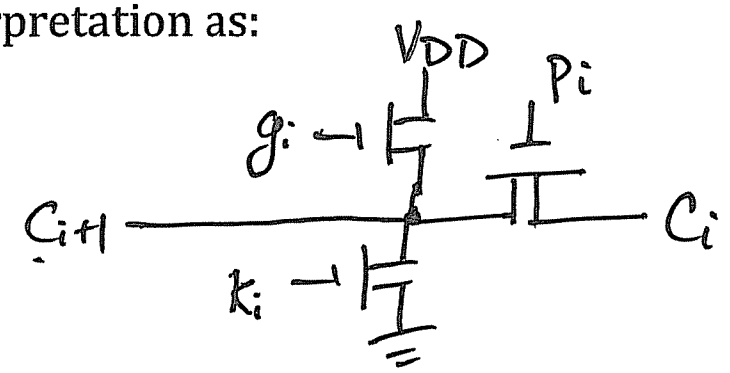
Recall generate, propagate, and carry-kill:

$C_{i+1} = 0$ when kill

$C_{i+1} = C_i$ when propagate

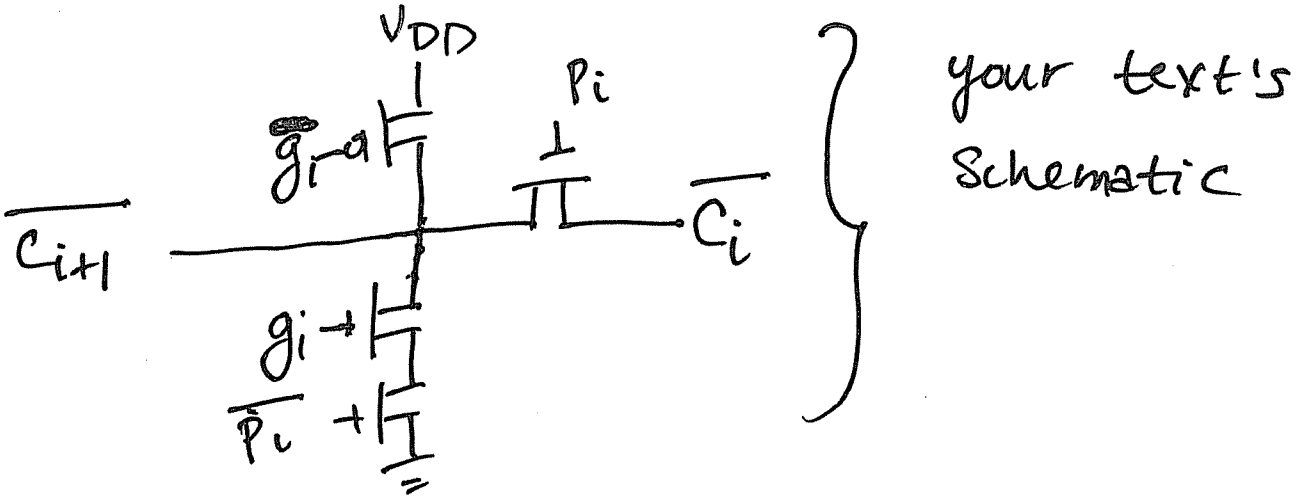
$C_{i+1} = 1$ when generate

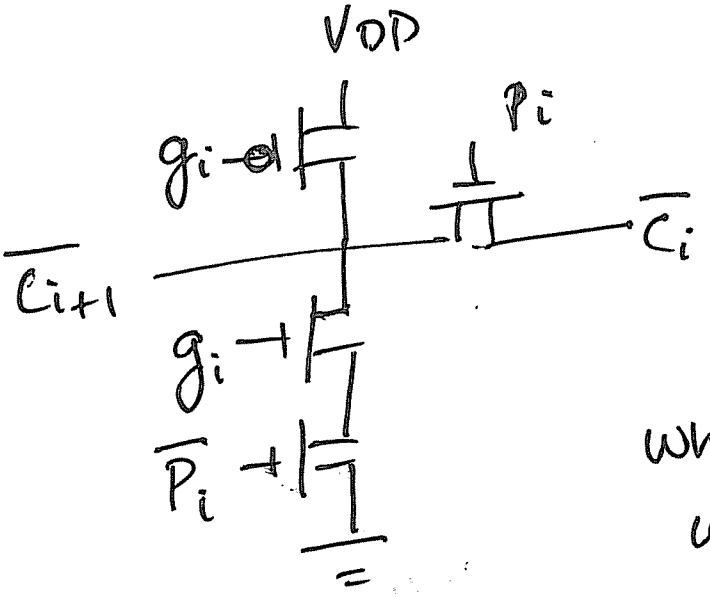
Which could be implemented at the most basic interpretation as:



This implementation suffers from Threshold drop, and k_i could be derived from p_i and g_i which would save space.

Manchester Carry Generator:





what happens when $P_i = 1$

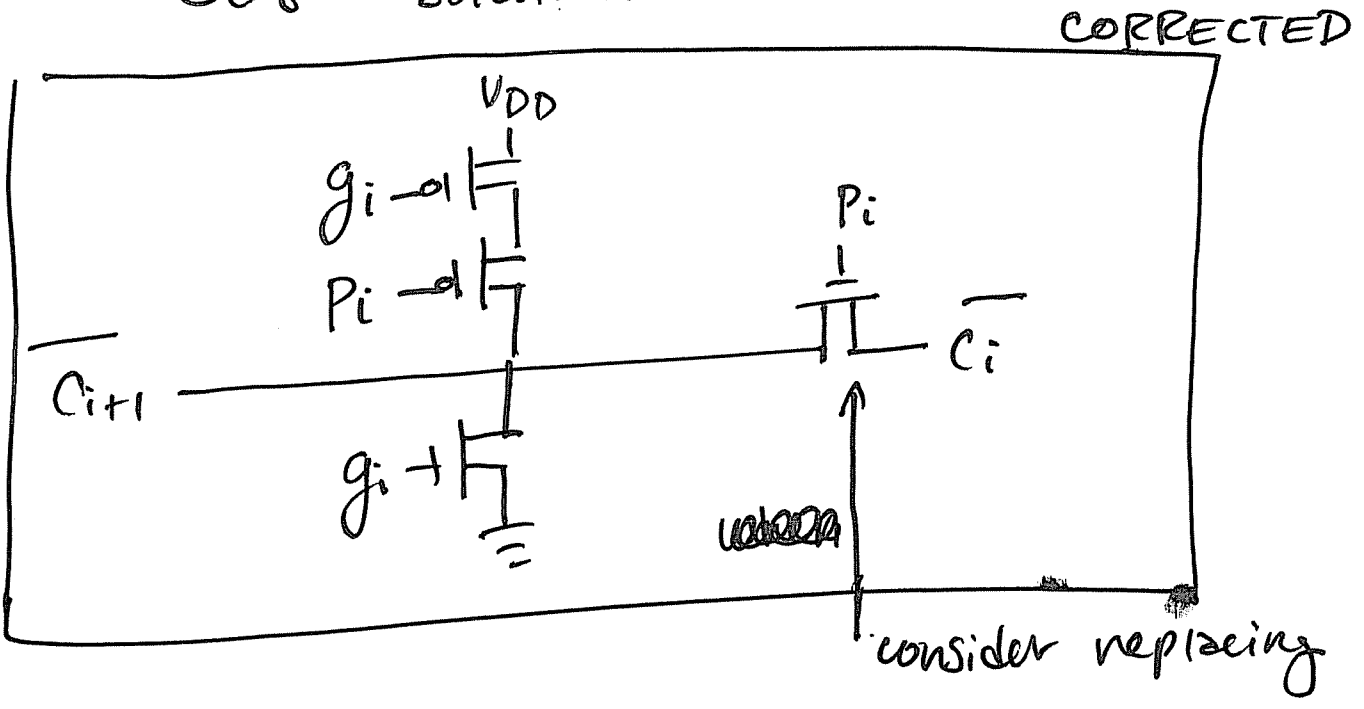
and $g_i = 0$

and $\overline{C_i} = 0$

$\Rightarrow \overline{C_{i+1}} = \overline{C_i}$

BUT, you get VDD because g_i PFET is stronger than P_i pass nFET

SOS solution:



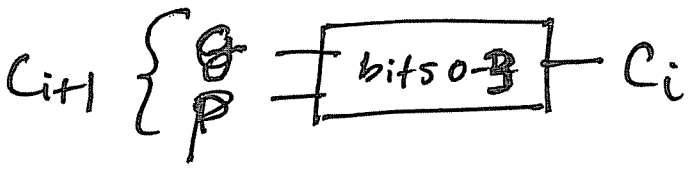
A significant design challenge in your final project is how to get the ~~CLA~~ from C_{in} to C_{out} as low as possible.

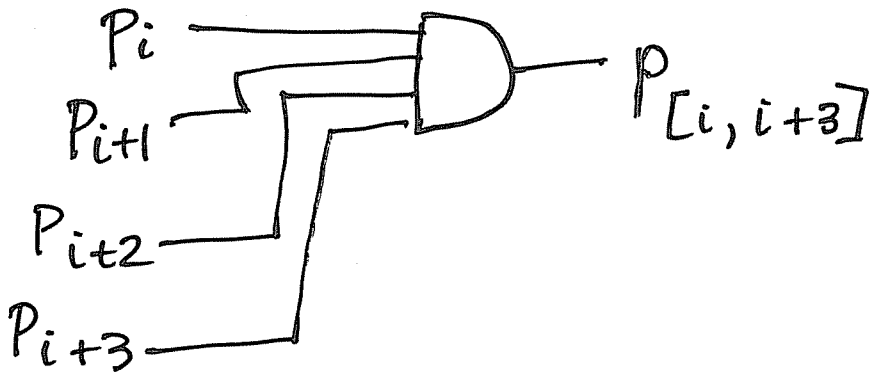
delay

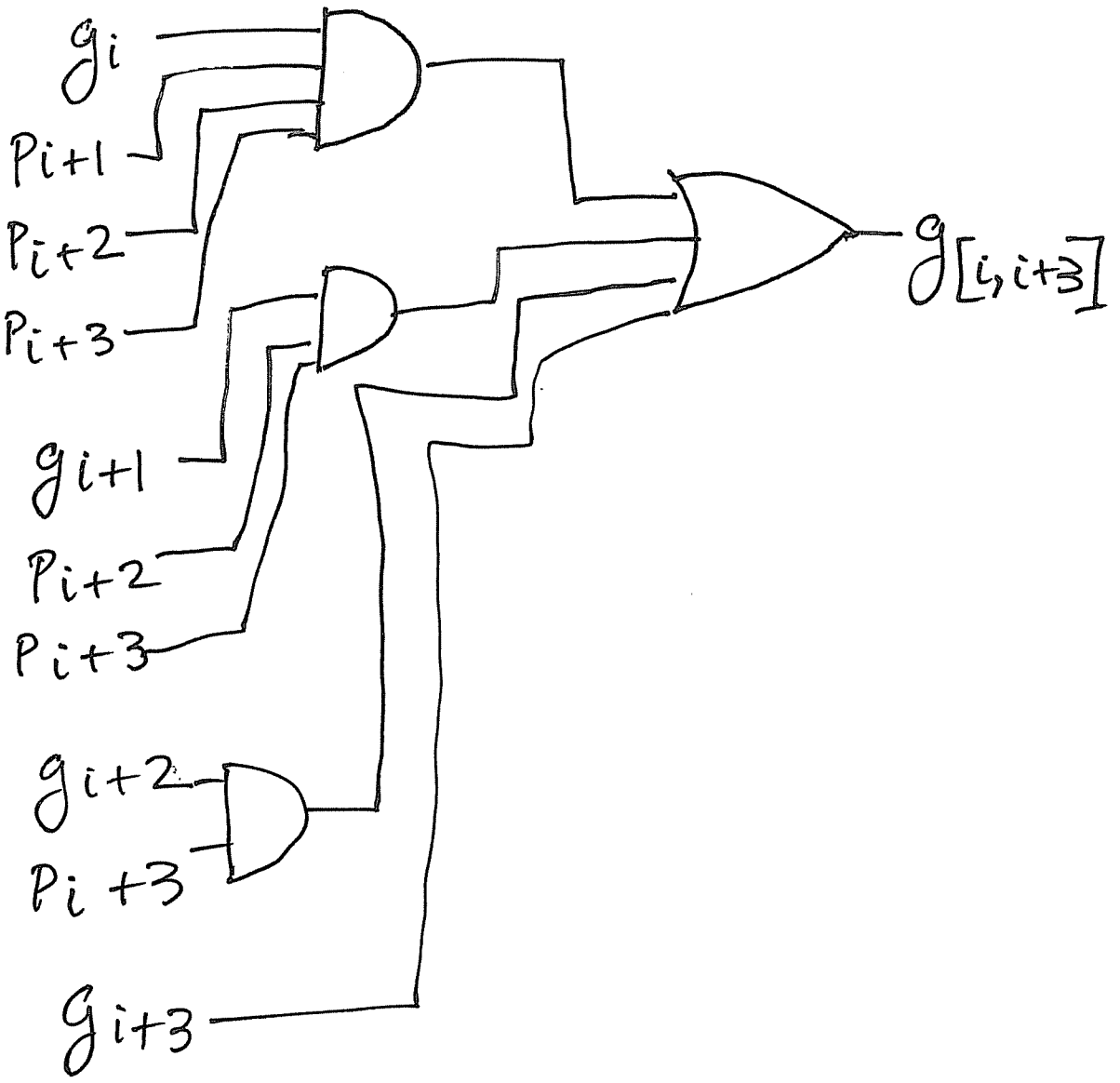
12.3.2 Extension to Wide Adders

What if we could break longer words into blocks so that if we found the **group generate** and **group propagate** signals the next block could start right away, and wait on the ripple for less time.

Is a given set of bits (like bits 0:3) going to propagate or generate? We would need C_{in} to determine C_{out}, but we could determine $g_{[0,3]}$ and $p_{[0,3]}$ before C_{in} arrives:



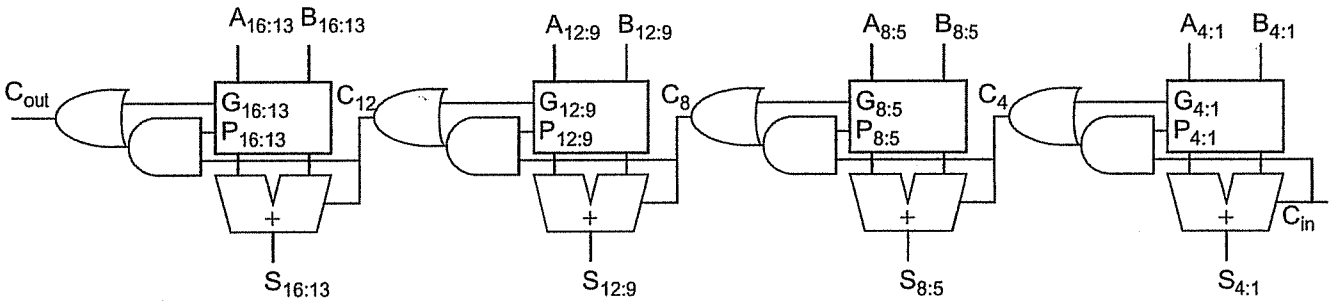




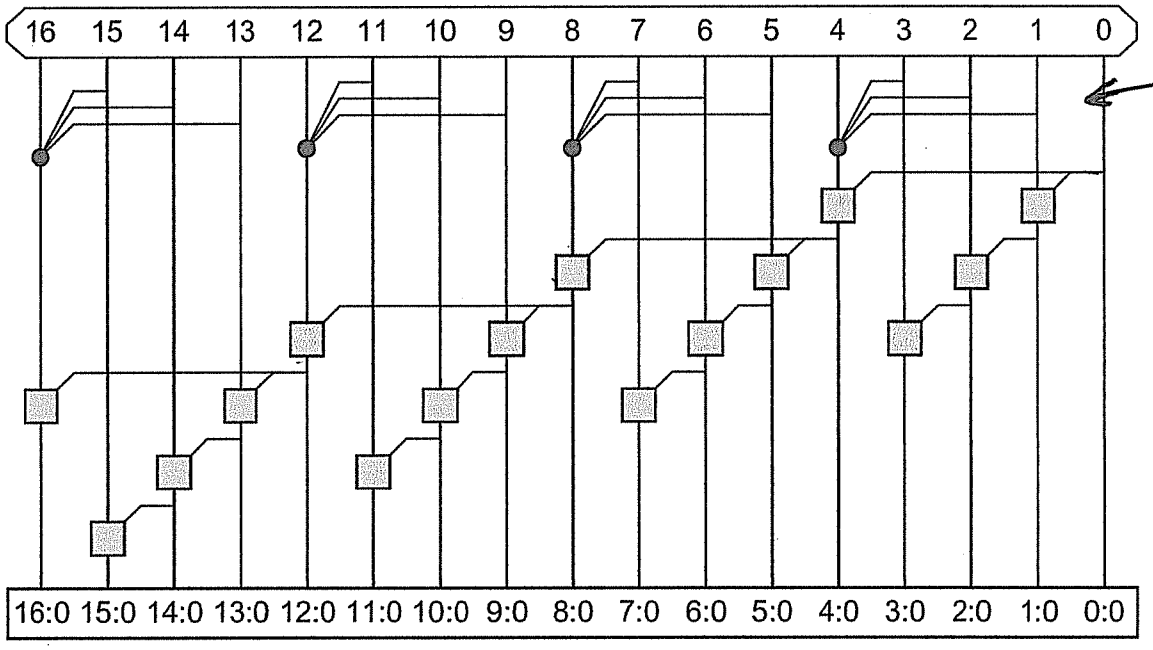
P, g only functions of a and b.

Could generate $g_{[i,i+3]}$ and $P_{[i,i+3]}$ before the "carry ripple" ever reached a block

This carry-lookahead implementation looks like this:



And reduces the delay to later stages:



Group generates and Group Propagates

Reading: Finish Ch 12, Start Ch 13

Check-In: "What is Booth encoding?"