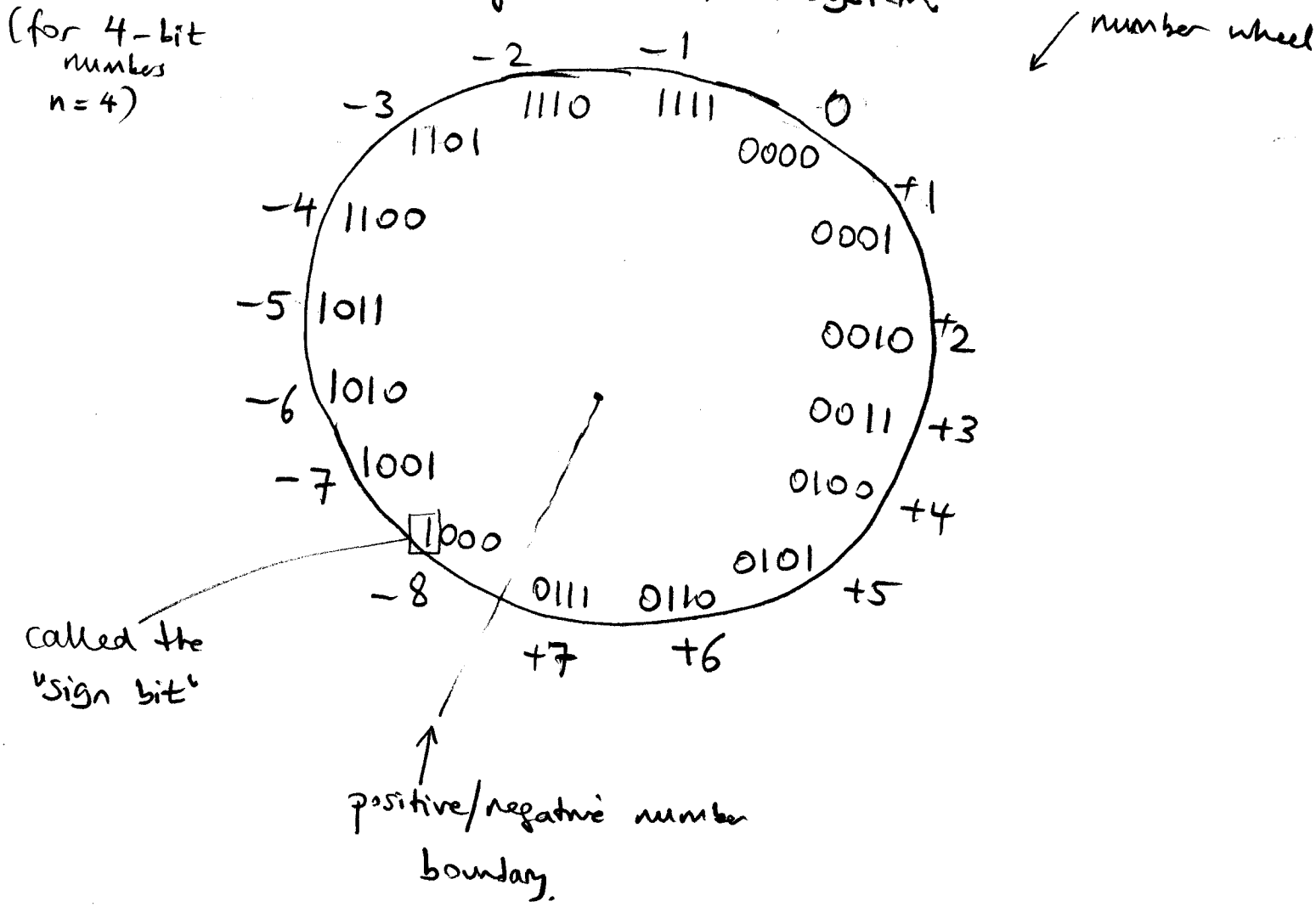
LECTURE #14

AIM: - understand number representation,

- overflow
- addition/subtraction circuits
- adders; fast adders: carry lookahead adder

I. Twos Complement numbers:

- the most widely used number system.



We form twos complement to represent negative number as follows:

Ex] $+3$: 0011

① Take ones complement.

1100

② Add 1: 1

1101

Subtraction:

$+5 - 3 = \underbrace{(+5)}_{0101} + \underbrace{(-3)}_{1101}$.

$$\begin{array}{r} 0101 \\ 1101 \\ \hline \cancel{1}\,0010 \end{array} \equiv +2 \quad \checkmark$$

↓ ignore.

① Note that the last stage had a carry-out but this did not indicate overflow!

② Number wheel interpretation: We start at +5, go back 3 steps to arrive at +2.

Overflow $\stackrel{def}{=}$ An overflow occurs in an operation iff the resulting number has no representation in the number system.

- In case of addition and subtraction of twos complement n-bit numbers, an overflow occurs iff:

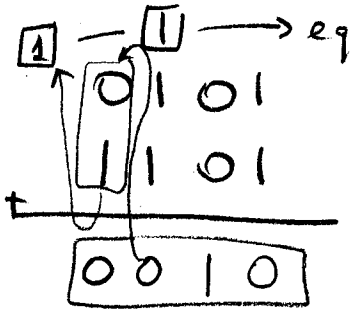
Operation	A	B	Result (n-bit)
A + B	>0	>0	<0
A + B	<0	<0	>0
A − B	>0	<0	<0
A − B	<0	>0	>0

- Number wheel interpretation: overflow occurs when you have crossed the boundary shown.

- Detecting overflow: (positive/negative)
  - When we build adders, it is good to build overflow detection circuitry to flag us when we have a bogus result.

Claim: An overflow in twos complement n-bit addition/subtraction occurs iff the carry-in of the last stage is $\neq$ carry-out of the last stage.

Example:



→ equal ∴ no overflow.

(+5)

(−3)

PROOF: need to consider only addition (since $A-B = A+(-B)$).

We know that overflow occurs iff

	A	+	B	.	Result
case#1:	>0		>0		<0
, or					
case#2:	<0		<0		>0.

case#1:

$$\begin{array}{r} 0 \\ \boxed{a_{n-1}}\ a_{n-2} \cdots a_1\ a_0 \\ 0\ \boxed{b_{n-1}}\ b_{n-2} \cdots b_1\ b_0 \\ \hline \end{array}$$

↗ 1 (negative)

will get a 1 here iff there is a carry-in to the last stage and in this case there will be no carry-out.

case#2:

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

happens iff ∃ no carry-in to the last stage (and note that there is no carry-out)

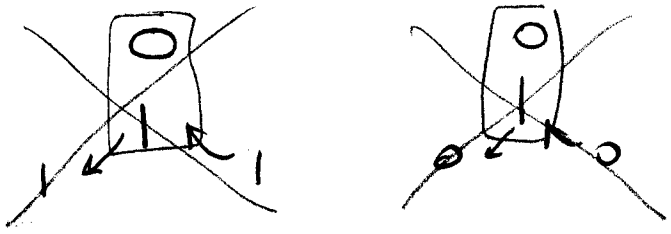
This shows that overflow conditions ⇒

carry-in ≠ carry-out in last stage.

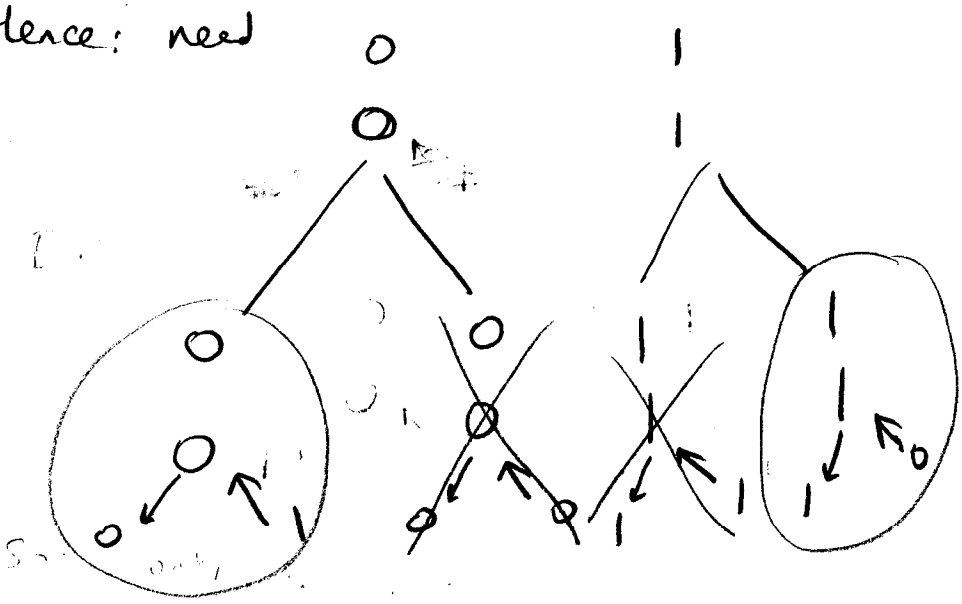
- Now, let's show the reverse:

If

~~i.e.~~ carry_in ≠ carry_out: Then:



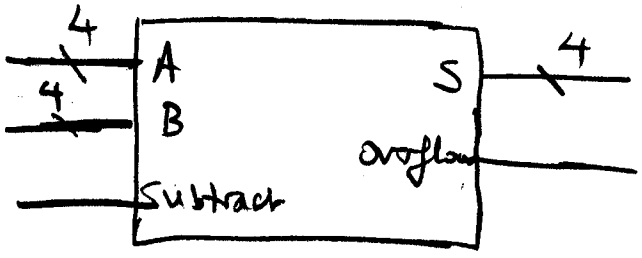
Hence: need



Q.E.D

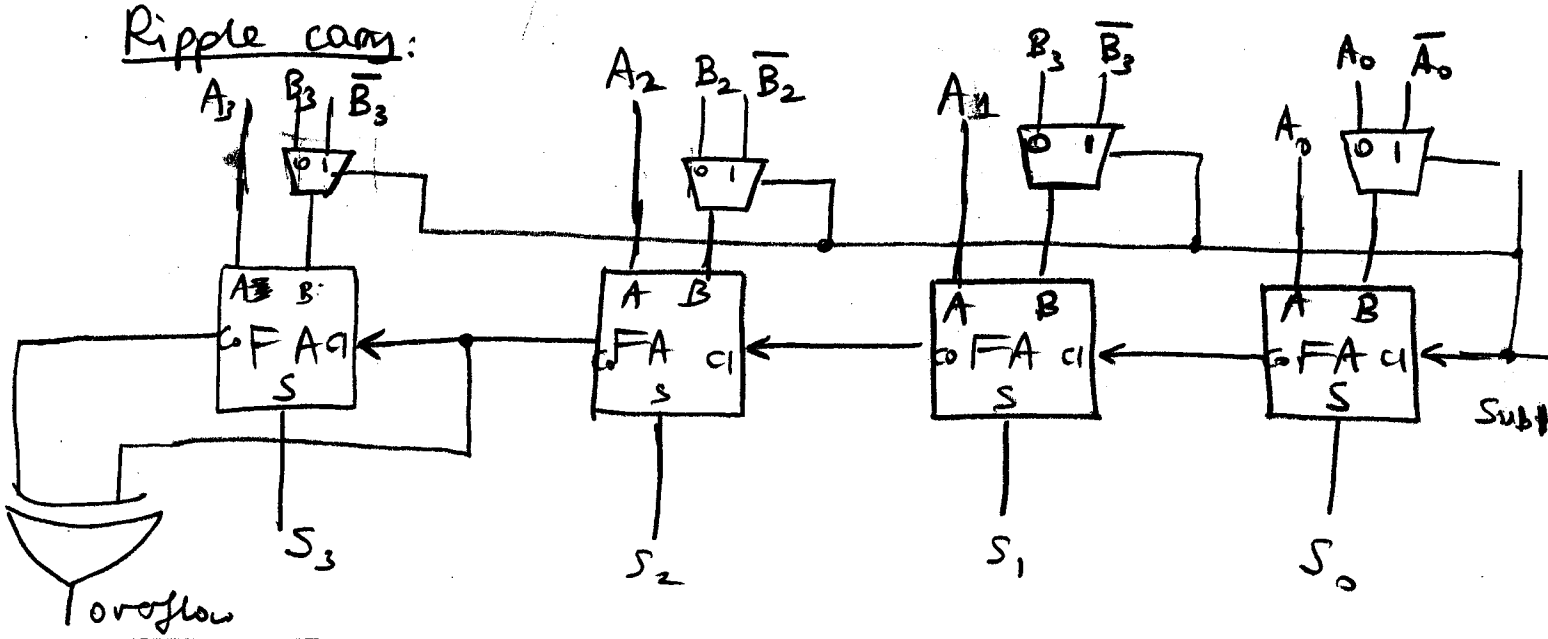
Networks for Binary Addition/Subtraction

Interface schematic:



Implementation:

Ripple carry:



Idea: For subtraction:

$$A - B \equiv A + \underbrace{(\text{onescomplement}(B))}_{\substack{B \text{ bitwise} \\ \text{inverted}}} + 1.$$

FAST ADDERS:

- at increased hardware complexity, reduce the delay

Example: Carry lookahead adder:

First, we need to develop some new concepts:

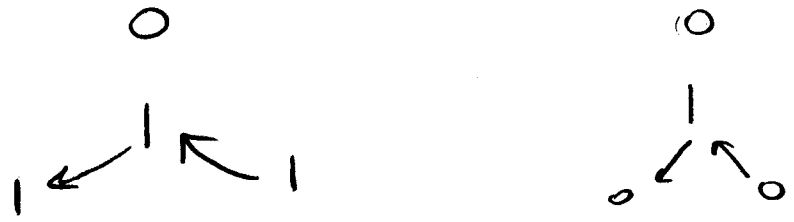
"generate" and "propagate".

- Generate:
  At the ith stage of addition, a carry-out is generated by the ith stage iff $A_i = B_i = 1$, i.e

generates a carry-out of 1 no matter what the carry-in to the ith stage is.

We write: $G_i = A_i \cdot B_i$

- Propagate: The ith stage will propagate the carry-in iff A_i and B_i are different.



We write: $P_i = A_i \oplus B_i$

Now, let's write the sum: (S_i) and C_{i+1} (carry out of stage i) in terms of P_i, G_i's.

Sum: (sth ...)

$$S_0 = (A_0 \oplus B_0) \oplus C_0$$

$$= P_0 \oplus C_0$$

$$S_1 = A_1 \oplus B_1 \oplus C_1$$

$$= P_1 \oplus C_1$$

$$\boxed{S_i = P_i \oplus C_i}$$

Carryout:

$$C_1 = A_0 B_1 + C_0 (A_0 \oplus B_0)$$

$$C_{i+1} = A_i B_i + C_i (A_i \oplus B_i)$$

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

∴ The carryout will be 1 iff either a carryout is generated in that stage OR there is a carry-in to that stage and it is propagated.

Explicitly, for the first four stages:

$$C_1 = G_0 + P_0 C_0$$

$$C_2 = G_1 + P_1 C_1 = G_1 + P_1(G_0 + P_0 C_0)$$

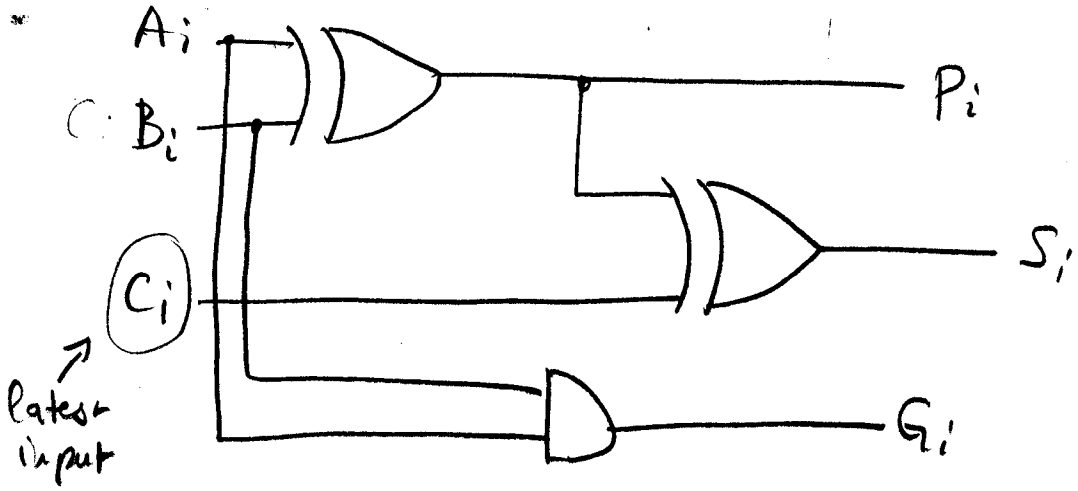
$$= G_1 + P_1 G_0 + P_1 P_0 C_0$$

$$C_3 = G_2 + P_2 C_2 = G_2 + P_2 G_1 + P_2 P_1 G_0 + P_2 P_1 P_0 C$$

$$C_4 = G_3 + P_3 C_3 = G_3 + P_3 G_2 + P_3 P_2 G_1$$

$$+ P_3 P_2 P_1 G_0 + P_3 P_2 P_1 P_0 C_0$$

- The <u>main idea</u> is that the propogate and generate can be computed based on A_i, B_i with only 1 gate delay. ($G_i = A_i \cdot B_i$, $P_i = A_i \oplus B_i$). So, after a 1-unit gate delay, $\{P_i, G_i\}$ are all available regardless of the n (# of stages)



Precompute P_i and G_i ; then wait for C_i.
After C_i arrives then S_i will be available 1 gate delay after that.

- The carry's create the critical path.

The 2-stage computation reduces the time to obtain the carryout

$$C_1 = G_0 + \underbrace{P_0 C_0}_{@2} \qquad @3 \quad @1 \quad @1 \; @0$$

if up-to fanin of 5 has delay $\leq$ one

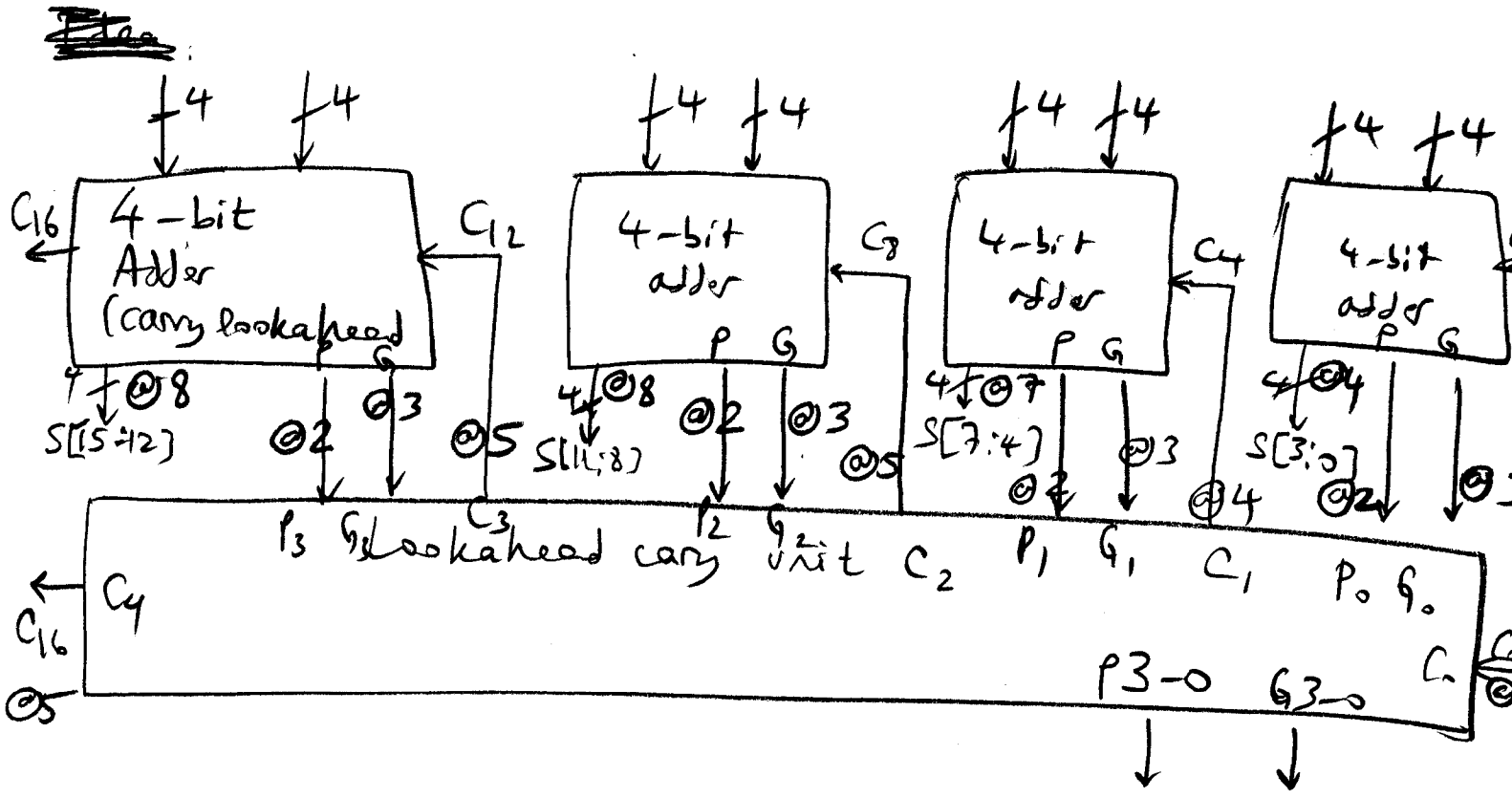
$$C_2 = G_1 + \underbrace{P_1 G_0}_{@2} + \underbrace{P_1 P_0 C_0}_{@2} \qquad @3$$

$$C_3 = G_2 + P_2 G_1 + P_2 P_1 G_0 + P_2 P_1 P_0 C_0 \qquad @3$$

$$C_4 = G_3 + P_3 G_2 + \cdots + P_3 P_2 P_1 P_0 C_0 \qquad @3$$

- But this constant time (@3) for carryout cannot be maintained because fanin is getting larger. (@4 for Sum)
- And hardware is getting large

How can we scale to larger adders such as 16-bit adder?



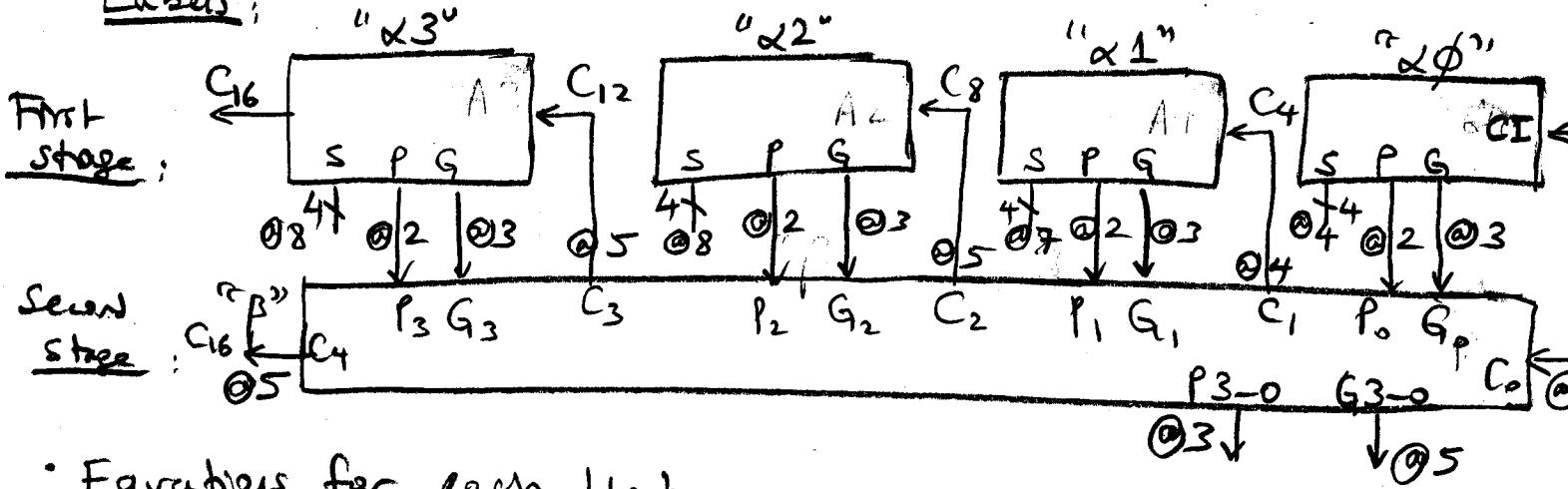
Idea: Each 4-bit adder computes its own "group" carry propagate and generate.

- Group propagate: $P_3 \cdot P_2 \cdot P_1 \cdot P_0$.
- Group generate: $G_3 + P_3 G_2 + P_3 P_2 G_1 + P_3 P_2 P_1 G_0$.

- The second-stage circuit computes the lookaheads and the output carries between first-stage 4-bit adders.
- The logic inside this second stage circuit is exactly the same as the one in a 4-bit adder block.

Worst-case Delay Analysis of 16-bit hierarchical CLA Adder:

Labels:



- Equations for each block.

$\alpha\phi$: Propogate: $P_j^{@1} = A_j^{@0} \oplus B_j^{@0}$ $\quad 0 \le j \le 3$

Generate: $G_j^{@1} = A_j^{@0} \cdot B_j^{@0}$ $\quad 0 \le j \le 3$

carries:

$$C_1^{@3} = G_0^{@1} + \underbrace{C_0^{@0} P_0^{@1}}_{@2}$$

$$C_2^{@3} = G_1^{@1} + P_1^{@1} G_0^{@1} + P_1^{@1} P_0^{@1} C_0^{@0}$$

$$C_3^{@3} = G_2 + P_2 G_1 + P_2 P_1 G_0 + P_2 P_1 P_0 C_0$$

(not needed/used in above design →)

$$C_4^{@3} = G_3 + P_3 G_2 + P_3 P_2 G_1 + P_3 P_2 P_1 G_0 + P_3 P_2 P_1 P_0 C_0$$

Sum:

$$S_j^{@4} = \underbrace{P_j^{@1} \oplus C_j^{@3}}_{@4} \qquad 0 \le j \le 3$$

L. THE SCOPE OF ALL VARIABLES HERE IS THIS BLOCK!]

group propagate:

$$P = P_3 \cdot P_2 \cdot P_1 \cdot P_0 \qquad @2 \quad @1\ @1\ @1\ @0$$

group generate:

$$G = G_3 + P_3 G_2 + P_3 P_2 G_1 + \underbrace{P_3 P_2 P_1 G_0}_{@2} \qquad @3 \quad @1 \quad @1\ @1 \quad @1\ @1\ @1 \quad @1\ @1\ @1\ @1$$

---

* Note that <u>all</u> group propagates will be ready @2 (i.e. for $\alpha_0, \alpha_1, \alpha_2, \alpha_3$) and <u>all</u> group generates will be ready @3 (i.e. for $\alpha_0, \alpha_1, \alpha_2, \alpha_3$).

[B]: <u>Equations</u>:

$$C_1 = G_0 + \underbrace{C_0 P_0}_{@3} \qquad @4 \quad @3 \quad @0\ @2$$

$$C_2 = G_1 + \underbrace{P_1 G_0}_{@4} + \underbrace{P_1 P_0 C_0}_{@3} \qquad @5 \quad @3 \quad @2\ @3 \quad @2\ @2\ @0$$

$$C_3 = G_2 + P_2 G_1 + \underbrace{P_2 P_1 G_0}_{@4} + P_2 P_1 P_0 C_0 \qquad @5 \quad @3 \quad @2\ @3 \quad @2\ @2\ @3 \quad @2\ @2\ @2\ @0$$

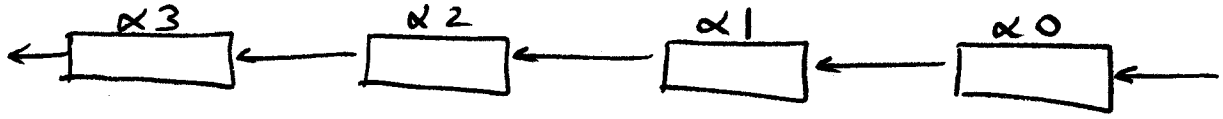
$$C_4 = G_3 + P_3 G_2 + \underbrace{P_3 P_2 G_1}_{@4} + P_3 P_2 P_1 G_0 + \underbrace{P_3 P_2 P_1 P_0 C_0}_{@3} \qquad @5 \quad @3 \quad @2\ @3 \quad @2\ @2\ @3 \quad @2\ @2\ @2\ @3 \quad @2\ @2\ @2\ @2\ @0$$

[SCOPE OF ALL THESE VARIABLES IS THIS BLOCK!]

Comments:

- So, the main idea in having the second-stage CLA circuit is to speed up the carries that need to be transferred from one α block to the next.

[Exercise: compare the performance of the above adder to:



i.e. a ripple-carry adder on the top-level, and carry-lookahead adders for the α blocks.]

Going back to the worst-case delay analysis:

Block: α_i, $1 \le i \le 3$

Equations:

Propagate: $P_j^{@1} = A_j^{@0} \oplus B_j^{@0}$ $\quad 0 \le j \le 3$

Generate: $G_j^{@1} = A_j^{@0} \cdot B_j^{@0}$ $\quad 0 \le j \le 3$

---

carries: (assume available @X.)

$@(X+2)$ $\quad C_1 = G_0 + \underbrace{C_0^{@X} P_0^{@1}}_{@(X+1)}$

$@(X+2)$ $\quad C_2 = G_1 + P_1 G_0 + \underbrace{P_1 P_0 C_{P_i}}_{@(X+1)}$

$@(X+2)$ $\quad C_3 = G_2 + P_2 G_1 + P_2 P_1 G_0 + P_2 P_1 P_0 C_0$

$@(X+2)$ $\quad C_4 = G_3 + P_3 G_2 + P_3 P_2 G_1 + P_3 P_2 P_1 P_0 C_0$

[THE SCOPE OF ALL THESE VARIABLES IS THIS BLOCK!
[C_0 is the carry-in bit to this block]

Sum: @(X+3) @1 @(X+2)

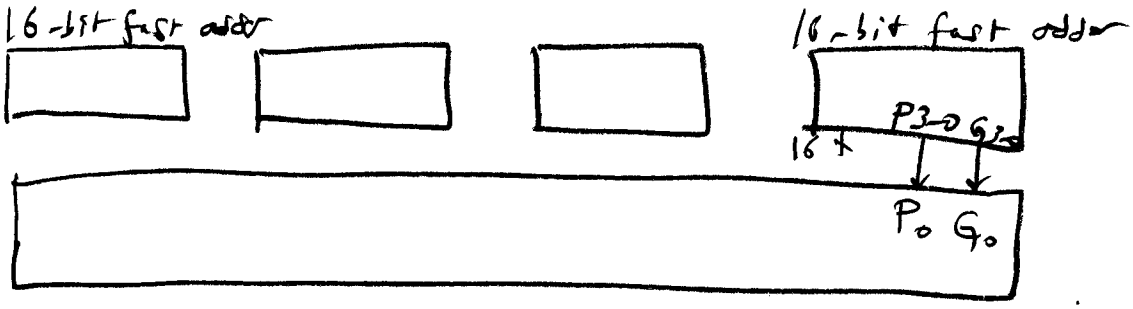
$$S_j = P_j \oplus C_j \qquad 0 \le j \le 3$$

This shows that the sum vector will be ready 3 gate delays after the carry-in to this adder is ready.

Therefore:

Adder Block	Sum is ready vector
$\alpha 2$	@ 7
$\alpha 3$	@ 8
$\alpha 4$	@ 8

Finally: The group propagate (P3-0) and group generate (G3-0) of the β block can be computed (e.g. to build a 64-bit adder:



you can use these 16-bit fast adders as shown above.)

$$P_{3-0} = \underbrace{P_3 \cdot P_2 \cdot P_1 \cdot P_0}_{@3} \qquad \begin{matrix}@3 & @2 & @2 & @2 & @2\end{matrix}$$

[these variables are internal to the P block.]

$$G_{3-0} = G_3 + P_3 G_2 + P_3 P_2 G_1 + \underbrace{P_3 P_2 P_1 G_0}_{@4} \qquad \begin{matrix}@5 & @3 & @2\ @3 & @2\,@2\,@3 & @2\,@2\,@2\,@3\end{matrix}$$