Chapter 7: Fast Addition: parallel prefix computation

Computer Structure
&
Intro. to Digital Computers

Dr. Guy Even

Tel-Aviv Univv.

Goals

- Design an adder with $O(\log n)$ delay and $O(n)$ cost.
- Learn some interesting methods along the way...

reminder: reduction sum-bits $\longmapsto$ carry-bits

The correctness of $\text{RCA}(n)$ implies that, for every $0 \leq i \leq n-1$,

$$S[i] = \text{XOR}(A[i], B[i], C[i]).$$

$\Longrightarrow$ constant-time linear-cost reduction:

$$S[n-1:0] \longmapsto C[n-1:0]$$

$\Longrightarrow$ if Circuit computes $C[n-1:0]$ with $O(n)$ cost and $O(\log n)$ delay, then we know how to add asymptotically optimally.

Computing the carry bits - preliminary

Functionality of Full-Adder:

$$C[i+1] = \begin{cases} 0 & \text{if } A[i] + B[i] + C[i] \leq 1 \\ 1 & \text{if } A[i] + B[i] + C[i] \geq 2. \end{cases}$$

Claim:

$$\begin{aligned} A[i] + B[i] = 0 &\implies C[i+1] = 0 \\ A[i] + B[i] = 2 &\implies C[i+1] = 1 \\ A[i] + B[i] = 1 &\implies C[i+1] = C[i] \end{aligned}$$

$\Longrightarrow$

- if $A[i] + B[i] \in \{0, 2\}$, then easy to compute $C[i+1]$.
- if $A[i] + B[i] = 1$, then “ripple effect” of carry.

definition of $\sigma[n-1:-1]$

DEF: for $i = -1, 0, \ldots, n-1$

$$\sigma[i] \triangleq \begin{cases} 2 \cdot C[0] & \text{if } i = -1 \\ A[i] + B[i] & \text{if } i \in [0, n-1]. \end{cases}$$

Note that $\sigma[i] \in \{0, 1, 2\}$.

Claim: for every $-1 \leq i \leq n-1$,

$$C[i+1] = 1 \qquad \Longleftrightarrow \qquad \exists j \leq i \ : \ \sigma[i:j] = 1^{i-j} \cdot 2.$$

example with $\sigma[n-1:-1]$

$$\sigma[i] \triangleq \begin{cases} 2 \cdot C[0] & \text{if } i = -1 \\ A[i] + B[i] & \text{if } i \in [0, n-1]. \end{cases}$$

Claim: for every $-1 \leq i \leq n-1$,

$$C[i+1] = 1 \qquad \Longleftrightarrow \qquad \exists j \leq i \ : \ \sigma[i:j] = 1^{i-j} \cdot 2.$$

Example: $A[3:0] = 0110, B[3:0] = 0011, C[0] = 0$.

position	4	3	2	1	0	-1
A		0	1	1	0	
B		0	0	1	1	
S		1	0	0	1	
C	0	1	1	0	0	
σ		0	1	2	1	0

Proof: $\sigma[i:j] = 1^{i-j} \cdot 2 \Rightarrow C[i+1] = 1$

- By induction on $i - j$.
- Basis $i - j = 0$: in this case $\sigma[i] = 2$.
  - If $i = -1$, then $C[0] = 1$.
  - If $i \geq 0$, then $A[i] + B[i] = 2$. Hence $C[i+1] = 1$.
- Ind. Step: note that $\sigma[i-1:j] = 1^{i-j-1} \cdot 2$.
- Ind. Hyp. $\Rightarrow C[i] = 1$.
- Since $\sigma[i] = 1$, we conclude that

$$\underbrace{A[i] + B[i]}_{\sigma[i]=1} + C[i] = 2.$$

- Hence, $C[i+1] = 1$.

Proof: $C[i+1] = 1 \Rightarrow \exists j \leq i : \sigma[i:j] = 1^{i-j} \cdot 2$

- By induction on i.
- Basis $i = -1$: in this case $C[0] = 1$, hence $\sigma[-1] = 2$. Set $j = i$.
- Ind. Step: Assume $C[i+1] = 1$. Hence,

$$\underbrace{A[i] + B[i]}_{\sigma[i]} + C[i] \geq 2.$$

  - $\sigma[i] = 0$: contradiction.
  - $\sigma[i] = 2$: set $j = i$.
  - $\sigma[i] = 1$: $\Rightarrow C[i] = 1$.

$$C[i] = 1 \overset{\text{Ind. Hyp.}}{\Longrightarrow} \exists j \leq i : \sigma[i-1:j] = 1^{i-j-1} \cdot 2$$

$$\overset{\sigma[i]=1}{\Longrightarrow} \exists j \leq i : \sigma[i:j] = 1^{i-j} \cdot 2.$$

Corollary: method for computing $C[i+1]$

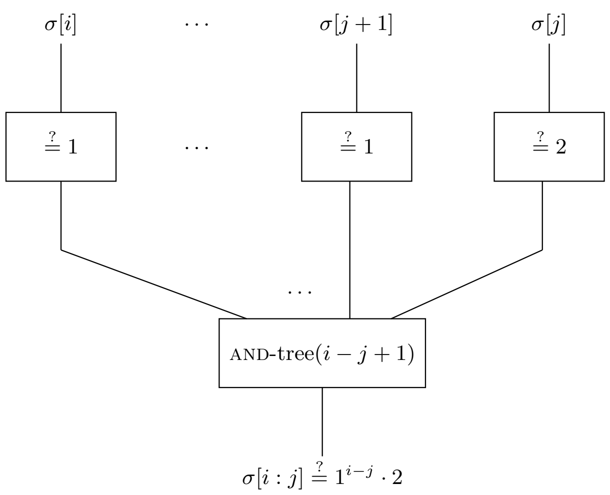
$$C[i+1] = 1 \quad \Longleftrightarrow \quad \exists j \leq i \ : \ \sigma[i:j] = 1^{i-j} \cdot 2.$$

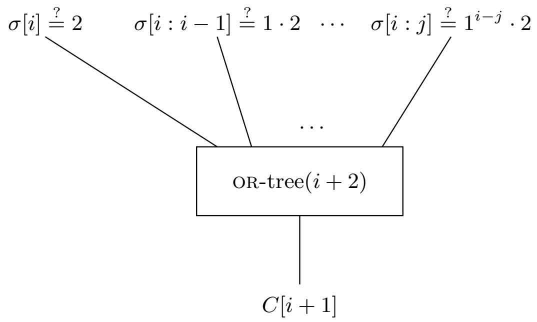
Corollary:

$$\begin{aligned} C[i+1] = {} & (\sigma[i:-1] = 1^{i+1} \cdot 2) \\ & \vee (\sigma[i:0] = 1^{i} \cdot 2) \\ & \vee (\sigma[i:1] = 1^{i-1} \cdot 2) \\ & \vdots \\ & \vee (\sigma[i:i-1] = 1 \cdot 2) \\ & \vee (\sigma[i] = 2) \end{aligned}$$

Carry-Lookahead Circuit

$$C[i+1] = 1 \quad \Longleftrightarrow \quad \exists j \leq i \ : \ \sigma[i:j] = 1^{i-j} \cdot 2.$$





Carry-Lookahead Circuit: cost & delay

- constant cost & depth comparison gates for deciding if :
  - $\sigma[i] = 1$
  - $\sigma[i] = 2$
- Use a row of comparison gates for $\sigma[i : j]$.
- Feed outputs of comparison gates to AND-tree$(i - j + 1)$.
- Cost of test $\sigma[i : j] = 1^{i-j} \cdot 2$

$$c(\text{AND-tree}(i - j + 1)) + (i - j + 1) \cdot c(\text{comparison}) \\ = \Theta(i - j).$$

- Delay of test $\sigma[i : j] = 1^{i-j} \cdot 2$

$$d(\text{comparison}) + (\text{AND-tree}(i + j + 1)) \\ = \Theta(\log(i - j)).$$

Carry-Lookahead Circuit: cost & delay - cont.

- Test if $\sigma[i : j] = 1^{i-j} \cdot 2$ for $j = -1, 0, \ldots, i$.
- Cost of computing $C[i + 1]$:

$$\sum_{j=-1}^{i} c(\text{testing if } \sigma[i : j] = 1^{i-j} \cdot 2) = \sum_{j=-1}^{i} \Theta(i - j) \\ = \Theta(i^2).$$

- Delay of computing $C[i + 1]$:

$$\max_{j=-1 \ldots i} \Theta(\log(i - j)) = \Theta(\log i).$$

- $\Rightarrow$ cost of computing $C[n : 1]$: $\sum_{i=0}^{n-1} \Theta(i^2) = \Theta(n^3)$.
- ...usually applied only to short blocks (e.g. 4 bits)

Carry-Lookahead Adder: typical description

SWITCHING EXPRESSIONS

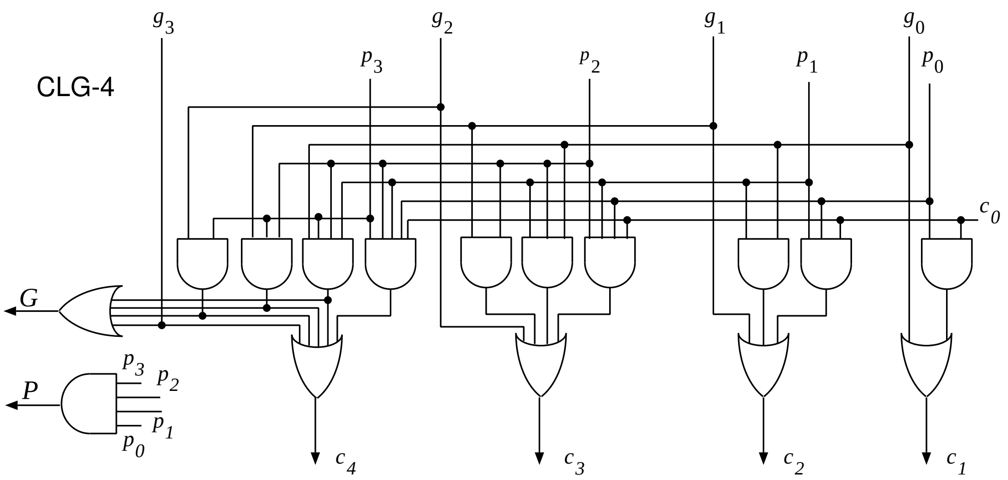
- INTERMEDIATE CARRIES:

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

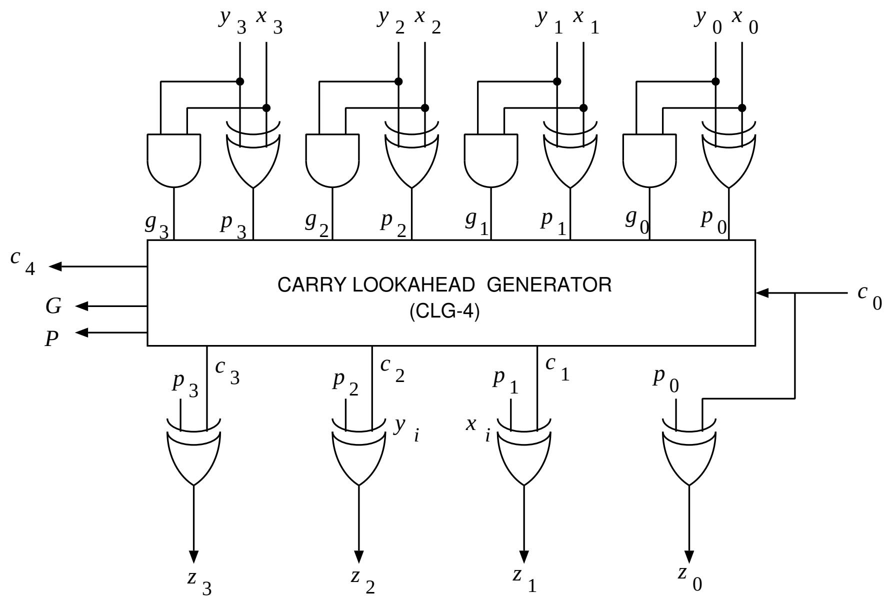
BY SUBSTITUTION,

$$\begin{aligned}
c_1 &= g_0 + p_0 c_0 \\
c_2 &= g_1 + p_1 c_1 \\
&= g_1 + p_1 g_0 + p_1 p_0 c_0 \\
c_3 &= g_2 + p_2 c_2 \\
&= g_2 + p_2 g_1 + p_2 p_1 g_0 + p_2 p_1 p_0 c_0 \\
c_4 &= g_3 + p_3 g_2 + p_3 p_2 g_1 + p_3 p_2 p_1 g_0 + p_3 p_2 p_1 p_0 c_0
\end{aligned}$$

Carry-Lookahead Adder: typical description



Carry-Lookahead Adder: typical description



from: Introduction to Digital Systems, M.D. Ercegovac, T. Lang, and J.H. Moreno, Wiley and Sons, 1998. – p.15

Two-level Carry-Lookahead Adder

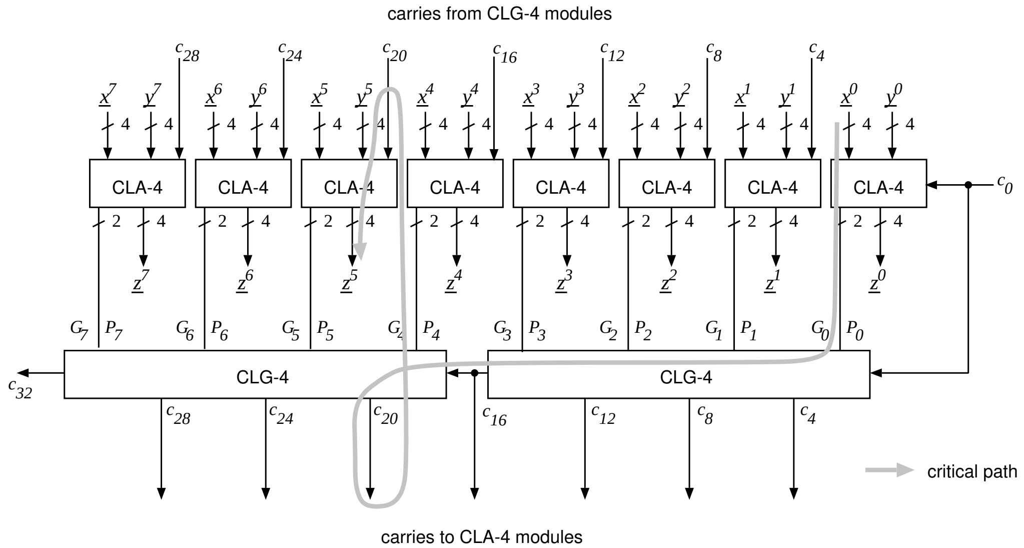


Figure 10.7: 32-BIT CARRY-LOOKAHEAD ADDER USING CLA-4 AND CLG-4 MODULES.

from: Introduction to Digital Systems, M.D. Ercegovac, T. Lang, and J.H. Moreno, Wiley and Sons, 1998. – p.16

Definition of $* : \{0,1,2\} \times \{0,1,2\} \longrightarrow \{0,1,2\}$

$*$	0	1	2
0	0	0	0
1	0	1	2
2	2	2	2

Remark: for every $a \in \{0,1,2\}$:

$$
\begin{aligned}
0 * a &= 0 \\
1 * a &= a \\
2 * a &= 2.
\end{aligned}
$$

Claim: (homework) $*$ is an associative function. Namely,

$$\forall a, b, c \in \{0,1,2\} : (a * b) * c = a * (b * c).$$

Question: Is $*$ commutative?

$*$-products

For $j \geq i$:

$$\prod[j:i] \triangleq \sigma[j] * \cdots * \sigma[i].$$

Associativity of $*$ implies that for every $i \leq j < k$:

$$\prod[k:i] = \prod[k:j+1] * \prod[j:i].$$

A stronger claim

Claim: For every $-1 \leq i \leq n-1$,

$$C[i+1] = 1 \qquad \Longleftrightarrow \qquad \prod[i:-1] = 2.$$

Corollary: Can compute $C[i+1]$ using a $*$-tree$(i+2)$.

- $\Rightarrow$

$$c(\text{compute } C[i+1]) = O(i)$$
$$d(\text{compute } C[i+1]) = O(\log i).$$

- $\Rightarrow$

$$c(\text{compute } C[n:1]) = \sum_{i=1}^{n} O(i) = O(n^2)$$
$$d(\text{compute } C[n:1]) = O(\log n).$$

explains carry-lookahead generator... still too expensive!

Proof: $C[i+1] = 1 \Longleftrightarrow \prod[i:-1] = 2$

From previous claim, it suffices to prove that

$$\exists j \leq i \;:\; \sigma[i:j] = 1^{i-j} \cdot 2 \qquad \Longleftrightarrow \qquad \prod[i:-1] = 2.$$

Proof: $\sigma[i:j] = 1^{i-j} \cdot 2 \Rightarrow \prod[i:-1] = 2$

- Assume that $\sigma[i:j] = 1^{i-j} \cdot 2$.
- $\Rightarrow$
$$\prod[i:j] = 2.$$
- If $j = -1$ we are done.
- Otherwise,
$$\prod[i:-1] = \underbrace{\prod[i:j]}_{=2} * \prod[j-1:-1]$$
$$= 2.$$

Proof: $\prod[i:-1] = 2 \Rightarrow \exists j \leq i \;:\; \sigma[i:j] = 1^{i-j} \cdot 2$

- Assume that $\prod[i:-1] = 2$.
- If, for every $\ell \leq i$, $\sigma[\ell] \neq 2$, then $\prod[i:-1] \neq 2$, a contradiction. Hence
$$\{\ell \in [-1, i] \;:\; \sigma[\ell] = 2\} \;\neq\; \emptyset.$$
- Let
$$j \triangleq \max\{\ell \in [-1, i] \;:\; \sigma[\ell] = 2\}.$$
- $\prod[j:-1] = 2$ (since $2 * a = 2$).
- We claim that $\sigma[\ell] = 1$, for every $j < \ell \leq i$.

Proof: $\prod[i:-1]=2 \Rightarrow \exists j \le i \ : \ \sigma[i:j]=1^{i-j}\cdot 2$

Let

$$j \triangleq \max\{\ell \in [-1,i] \ : \ \sigma[\ell]=2\}.$$

We claim that $\sigma[\ell]=1$, for every $j<\ell\le i$.

- max. of j $\Rightarrow$ for every $j<\ell\le i$: $\sigma[\ell]\neq 2$.
- if $\sigma[\ell]=0$, for $j<\ell\le i$, then $\prod[i:\ell]=0$.
  $\Rightarrow$
  $$\prod[i:-1]=\prod[i:\ell]*\prod[\ell-1:-1]=0,$$
  a contradiction.
- since $\sigma[i:j+1]=1^{i-j}$, we conclude that $\sigma[i:j]=1^{i-j}\cdot 2$, QED.

Prefix Computation Problem

DEF: Let Σ denote a finite alphabet. Let $\textsc{op}:\Sigma^2\longrightarrow\Sigma$ denote an associative function. A prefix computation over Σ with respect to $\textsc{op}$ is defined as follows.

Input $x[n-1:0]\in\Sigma^n$.

Output: $y[n-1:0]\in\Sigma^n$ defined recursively as follows:

$$y[0]\leftarrow x[0]$$
$$y[i+1]=\textsc{op}(x[i+1],y[i]).$$

Note that $y[i]$ can be also expressed simply by

$$y_i=\textsc{op}_{i+1}(x[i],x[i-1],\ldots,x[0]).$$

Reduction: $C[n:1] \longmapsto$ Prefix Computation Prob.

The Claim

$$C[i+1] = 1 \qquad \Longleftrightarrow \qquad \prod[i:-1] = 2$$

implies a reduction to the Prefix Computation Problem:

- $\Sigma = \{0, 1, 2\}$
- OP $= *$
- input: $\sigma[-1:n]$
- output: $y[i] = \prod[i:-1]$.

Prefix Computation Problem - example

- $\Sigma = \{0, 1\}$
- OP $=$ OR

$\Longrightarrow$ Unary Priority Encoder U-PENC(n).

Parallel Prefix Circuit

DEF: A Parallel Prefix Circuit, PPC–OP(n), is a combinational circuit that computes a prefix computation. Namely, given input $x[n-1:0] \in \Sigma^n$, it outputs $y[n-1:0] \in \Sigma^n$, where

$$y_i = \text{OP}_{i+1}(x[i], x[i-1], \ldots, x[0]).$$

- representation of values in Σ - not addressed.
- assume: some fixed representation is used.
- OP-gate: given representations of $a, b \in \Sigma$, outputs a representation of OP(a, b).

PPC–OP(n) - questions

Question: Design a PPC–OP(n) circuit with linear delay and cost.

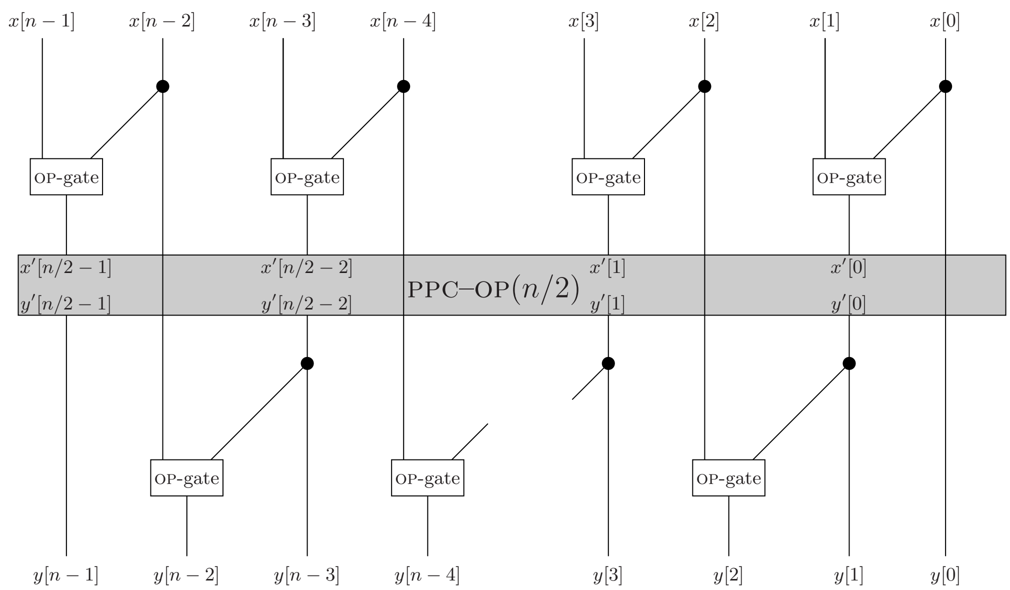
Question: Design a PPC–OP(n) circuit with logarithmic delay and quadratic cost.

Question: Assume that a design $C(n)$ is a PPC–OP(n). This means that it is comprised only of OP-gates and works correctly for every alphabet Σ and associative function OP : $\Sigma^2 \rightarrow \Sigma$. Can you prove a lower bound on its cost and delay?

PPC–OP - implementation

- A recursive design.
- We already saw a divide-and-conquer design for U-PENC(n) with cost $\Theta(n \cdot \log n)$.
- Aim for $O(n)$ cost.
- "odd-even" divide-and-conquer (as opposed to left/right side divide-and-conquer).
- basis $n = 2$: an OP-gate.
- recursion step...

PPC–OP(n) - recursion step



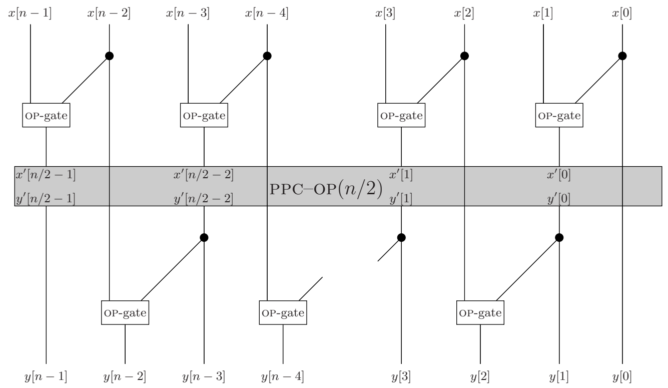
PPC–OP(n) - correctness

- By induction. Basis: holds trivially for $n = 2$. We now prove the induction step.
- $x'[n/2-1:0], y'[n/2-1]$ - inputs/outputs of PPC–OP$(n/2)$.
- $x'[i] \leftarrow$ OP$(x[2i+1], x[2i])$.
- Induction hypothesis:

$$\begin{aligned} y'[i] &= \text{OP}_{i+1}(x'[i], \ldots, x'[0]) \\ &= \text{OP}_{2i+1}(x[2i+1], \ldots, x[0]). \end{aligned}$$

- $y[2i+1] \leftarrow y'[i] \Rightarrow$ odd indexed outputs $y[1], y[3], \ldots, y[n-1]$ are correct.
- $y[2i] \leftarrow$ OP$(x[2i], y'[i]) \Longrightarrow y[2i] =$ OP$(x[2i], y[2i-1])$.
- $\Rightarrow$ even indexed outputs are also correct. QED

PPC–OP(n) - delay analysis $(n = 2^k)$

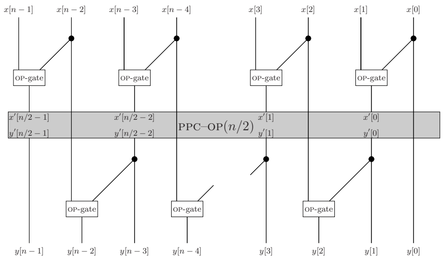


$$d(\text{PPC–OP}(n)) = \begin{cases} d(\text{OP-gate}) & \text{if } n = 2 \\ d(\text{PPC–OP}(n/2)) + 2 \cdot d(\text{OP-gate}) & \text{otherwise.} \end{cases}$$

If follows that

$$d(\text{PPC–OP}(n)) = (2\log n - 1) \cdot d(\text{OP-gate}).$$

PPC–OP(n) - cost analysis $(n = 2^k)$



$$c(\text{PPC–OP}(n)) = \begin{cases} c(\text{OP-gate}) & \text{if } n = 2 \\ c(\text{PPC–OP}(n/2)) + (n-1) \cdot c(\text{OP-gate}) & \text{otherwise.} \end{cases}$$

It follows that

$$\begin{aligned} c(\text{PPC–OP}(n)) &= \sum_{i=2}^{k} (2^i - 1) \cdot c(\text{OP-gate}) + c(\text{OP-gate}) \\ &= (2n - 4 - (k-1) + 1) \cdot c(\text{OP-gate}) \\ &= (2n - \log n - 2) \cdot c(\text{OP-gate}). \end{aligned}$$

PPC–OP(n) - corollary

Corollary: If the delay and cost of an OP-gate is constant, then

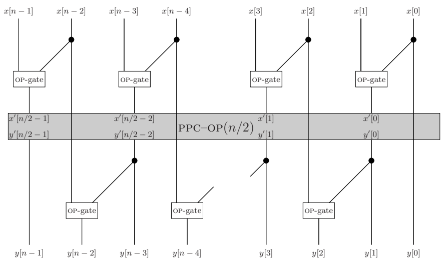
$$\begin{aligned} d(\text{PPC–OP}(n)) &= \Theta(\log n) \\ c(\text{PPC–OP}(n)) &= \Theta(n). \end{aligned}$$

$\Rightarrow$

- $\Sigma = \{0, 1\}$ & OP = OR
  $\Rightarrow$ asymptotically optimal U-PENC(n).
- $\Sigma = \{0, 1, 2\}$ & OP $= *$
  $\Rightarrow$ compute carry-bits $C[n:1]$ with $O(n)$ cost and $O(\log n)$ delay.

PPC–OP(n) - fanout

- Insert a buffer in every branching point of the PPC–OP(n) design.
- $\Rightarrow$ constant fanout.
- Question: What is the maximum fanout in the PPC–OP(n) design. Analyze the effect of inserting buffers to the cost and delay of PPC–OP(n).



putting it all together

Compute $\sigma[n-1:-1]$: Cost & delay are constant per $\sigma[i]$.
$\Rightarrow$ total cost is $O(n)$ & the total delay is $O(1)$.

PPC–$*$ (n): Compute product $\prod[i:-1]$ from $\sigma[i:-1]$, for every $i \in [n-1:0]$.
The cost $O(n)$ and delay $O(\log n)$.

Extraction of $C[n:1]$: Recall $C[i+1]=1$ iff $\prod[i:-1]=2$.
Compare each $\prod[i:-1]$ with 2. The result of this comparison equals $C[i+1]$.
The cost and delay is constant per carry-bit $C[i+1]$.
Total cost of this step is $O(n)$ and the delay is $O(1)$.

Computation of sum-bits: The sum bits are computed by

$$S[i] = \text{XOR}_3(A[i], B[i], C[i]).$$

Cost of this step is $O(n)$ and the delay is $O(1)$.

Fast Addition

By combining the cost and delay of each stage we obtain the following result.

Theorem: The adder based on parallel prefix computation is asymptotically optimal; its cost is linear and its delay is logarithmic.

Summary

- Presented an adder with asymptotically optimal cost and delay.
- Design based on two reductions:
  - reduction of the task of computing the sum-bits to the task of computing the carry bits.
  - reduction of the task of computing the sum bits to a prefix computation problem.
- A prefix computation problem is the problem of computing $\text{OP}_i(x[i-1:0])$, for $0 \leq i \leq n-1$, where OP is an associative operation.
- PPC–OP(n) - a linear cost logarithmic delay circuit for the prefix computation problem.
- Can use PPC–OP(n) for asymptotically optimal U-PENC(n).