Chapter 6: Addition

Computer Structure - Spring 2004

©Dr. Guy Even

Tel-Aviv Univ.

Goals

- Binary addition - definition
- Ripple Carry Adder - definition, correctness, cost, delay
- Carry bits - definition, properties
- (*) Conditional Sum Adder - definition, correctness, cost, delay
- (*) Compound Adder - definition, correctness, cost, delay

Binary Addition

DEF: A binary adder with input length n is a combinational circuit specified as follows.

Input: $A[n-1:0], B[n-1:0] \in \{0,1\}^n$, and $C[0] \in \{0,1\}$.

Output: $S[n-1:0] \in \{0,1\}^n$ and $C[n] \in \{0,1\}$.

Functionality:

$$\langle \vec{S} \rangle + 2^n \cdot C[n] = \langle \vec{A} \rangle + \langle \vec{B} \rangle + C[0]$$

- $\vec{A}, \vec{B}$ - binary representations of the addends.
- $C[0]$ - the carry-in bit.
- $\vec{S}$ - binary representation of the sum.
- $C[n]$ - the carry-out bit.

Question: is the functionality of ADDER(n) is well defined?

Lower bounds

Prove that for every ADDER(n):

- $c(\text{ADDER}(n)) = \Omega(n)$
- $d(\text{ADDER}(n)) = \Omega(\log n)$

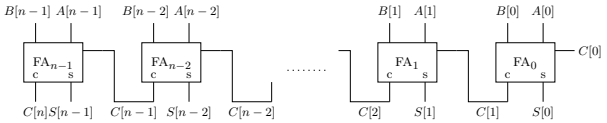
Full Adder

A Full-Adder is a combinational circuit with 3 inputs $x, y, z \in \{0,1\}$ and 2 outputs $c, s \in \{0,1\}$ that satisfies:

$$2c + s = x + y + z.$$

- A Full Adder computes a binary representation of the sum of 3 bits.
- s - called the sum output.
- c - called the carry-out output.
- We denote a Full-Adder by FA.

Ripple Carry Adder - RCA(n)



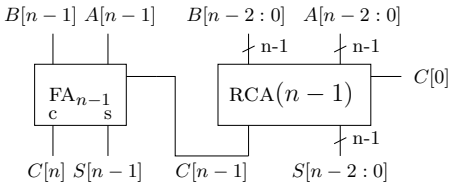
- carry-out output of FA$_i$ is denoted by $c[i+1]$.
- weight of every signal is two to the power of its index.
- RCA(n) - algorithm that we use for adding numbers by hand.

Correctness proof

To facilitate the proof, we use an equivalent recursive definition of RCA(n).

The recursive definition is as follows.

- Basis: an RCA(1) is simply a Full-Adder.
- Step:



Correctness - cont.

The proof is by induction on n.

The induction basis, for $n = 1$, follows directly from the definition of a Full-Adder.

Induction Step

The induction hypothesis, for $n-1$, is

(1) $\langle A[n-2:0]\rangle + \langle B[n-2:0]\rangle + C[0] = 2^{n-1}\cdot C[n-1] + \langle S[n-2:0]\rangle.$

Full-Adder definition

(2) $A[n-1] + B[n-1] + C[n-1] = 2\cdot C[n] + S[n-1].$

Multiply (2) by 2^{n-1} to obtain

(3) $2^{n-1}\cdot A[n-1] + 2^{n-1}\cdot B[n-1] + 2^{n-1}\cdot C[n-1] = 2^n\cdot C[n] + 2^{n-1}\cdot S[n-1].$

(1) $\langle A[n-2:0]\rangle + \langle B[n-2:0]\rangle + C[0] = 2^{n-1}\cdot C[n-1] + \langle S[n-2:0]\rangle.$

(3) $2^{n-1}\cdot A[n-1] + 2^{n-1}\cdot B[n-1] + 2^{n-1}\cdot C[n-1] = 2^n\cdot C[n] + 2^{n-1}\cdot S[n-1].$

Note that $2^{n-1}\cdot A[n-1] + \langle A[n-2:0]\rangle = \langle A[n-1:0]\rangle$.
(1) + (3) $\Longrightarrow$

$2^{n-1}\cdot C[n-1] + \langle A[n-1:0]\rangle + \langle B[n-1:0]\rangle + C[0] = 2^n\cdot C[n] + 2^{n-1}\cdot C[n-1] + \langle S[n-1:0]\rangle.$

Cancel out $2^{n-1}\cdot C[n-1]$. QED.

Cost & Delay Analysis

The cost of an RCA(n) satisfies:

$$c(\text{RCA}(n)) = n\cdot c(\text{FA}) = \Theta(n).$$

The delay of an RCA(n) satisfies

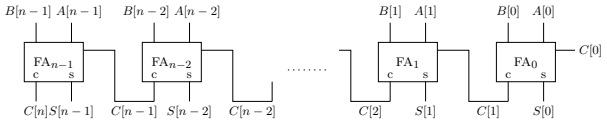
$$d(\text{RCA}(n)) = n\cdot d(\text{FA}) = \Theta(n).$$

Is RCA(n) good enough?

- Clock rate $= 1GHz = 10^9Hz$
  $\Rightarrow$ clock period $= 10^{-9}$sec $= 1ns$.
- Delay of gate $\approx 100ps = 0.1ns$.
- $d(\text{FA}) \approx 2\cdot d(\text{gate}) \approx 0.2ns$.
- $\Rightarrow$ Within a clock period we can only add 5-bit numbers...
- Question: How are > 100 bits added in one clock cycle?

Carry bits

DEF: The carry bits associated with an addition $\langle\vec{A}\rangle + \langle\vec{B}\rangle + C[0]$ are the signals $C[n:0]$ in an RCA(n).



remark 1: redundant & non-redundant representations

Functionality of an adder:

$$\langle A[n-1:0]\rangle + \langle B[n-1:0]\rangle + C[0] = 2^n \cdot C[n] + \langle S[n-1:0]\rangle.$$

- Let $x = \langle\vec{A}\rangle + \langle\vec{B}\rangle + C[0]$.
- x admits two representations (left-hand side, right-hand side)
- $C[n] \cdot S[n-1:0]$ - binary representation of x.
- Binary representation is non-redundant:
  - Every value has a unique representation.
  - $\langle\vec{X}\rangle = \langle\vec{Y}\rangle \iff X = Y$.

remark 1 - cont

Functionality of an adder:

$$\langle A[n-1:0]\rangle + \langle B[n-1:0]\rangle + C[0] = 2^n \cdot C[n] + \langle S[n-1:0]\rangle.$$

- $x = \langle A[n-1:0]\rangle + \langle B[n-1:0]\rangle + C[0]$.
- many possible combinations of $\langle\vec{A}\rangle, \langle\vec{B}\rangle$ and $C[0]$. For example: $8 = 4 + 3 + 1$, and also $8 = 5 + 3 + 0$.
- $\rightarrow$ redundant representation.
- in redundant representation:

$$X \neq Y \not\Longrightarrow \text{value(X)} \neq \text{value}(Y).$$

- $\Rightarrow$ in redundant representation: comparison is complicated.
- ADDER(n) - translates a redundant representation to a non-redundant binary representation.

remark 2: cones

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

$$\langle A[i:0]\rangle + \langle B[i:0]\rangle + C[0] = 2^{i+1} \cdot C[i+1] + \langle S[i:0]\rangle.$$

This equality means that:

$$\textit{cone}(C[i+1]), \textit{cone}(S[i:0]) \subseteq A[i:0] \bigcup B[i:0] \bigcup C[0].$$

Question: Prove that

$$\textit{cone}(S[i]), \textit{cone}(C[i+1]) = A[i:0] \bigcup B[i:0] \bigcup C[0].$$

remark 3

$$\langle A[i:0]\rangle + \langle B[i:0]\rangle + C[0] = 2^{i+1} \cdot C[i+1] + \langle S[i:0]\rangle.$$

$\Longrightarrow$ for every $0 \leq i \leq n-1$,

$$\langle S[i:0]\rangle = \text{mod}(\langle A[i:0]\rangle + \langle B[i:0]\rangle + C[0], 2^{i+1}).$$

remark 4: reductions sum-bits $\longleftrightarrow$ carry-bits

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

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

$\Longrightarrow$ for every $0 \leq i \leq n-1$,

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

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

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

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

Chapter 7: Fast Addition: parallel prefix computation

Computer Structure - Spring 2004

©Dr. Guy Even

Tel-Aviv Univ.

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 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 (ith FA in RCA(n)):

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

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 $-1 \leq i \leq n-1$,

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

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 $-1 \leq i \leq n-1$,

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

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 $C[i+1]$

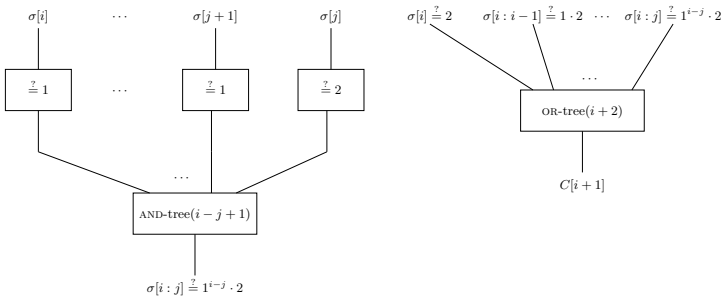
$$C[i+1] = 1 \iff \exists j \leq i \ : \ \sigma[i:j] = 1^{i-j} \cdot 2.$$

Corollary:

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

Carry-Lookahead Generator

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



Carry-Lookahead Generator: 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 Generator: 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

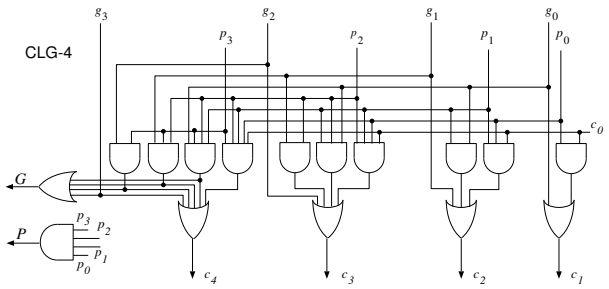
- INTERMEDIATE CARRIES:

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

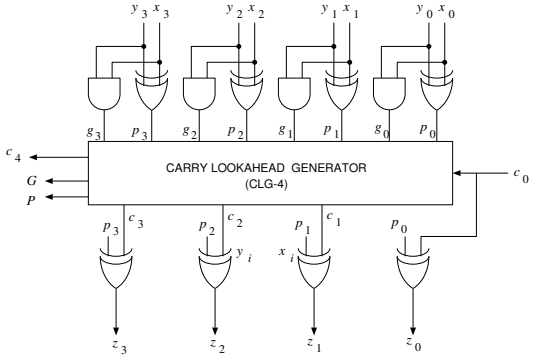
BY SUBSTITUTION,

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

Carry-Lookahead Adder: typical description



Carry-Lookahead Adder: typical description



Two-level Carry-Lookahead Adder

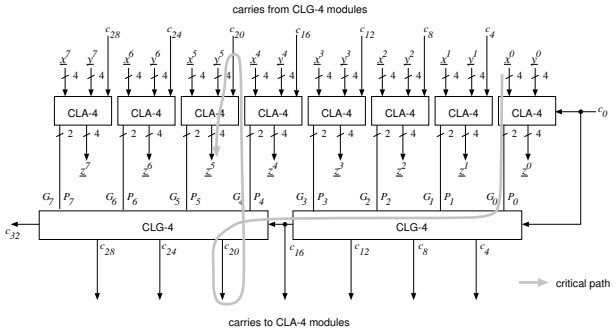


Figure 10.7: 32-BIT CARRY-LOOKAHEAD ADDER USING CLA-4 AND CLG-4 MODULES.

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 $j \geq i$:

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

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

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

A stronger claim

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

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

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

From previous claim, it suffices to prove that

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

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

- Assume that $\sigma[i:j] = 1^{i-j} \cdot 2$.
- $\Rightarrow$

$$\pi[i:j] = 2.$$

- If $j = -1$ we are done.
- Otherwise,

$$\pi[i:-1] = \underbrace{\pi[i:j]}_{=2} * \pi[j-1:-1]$$
$$= 2.$$

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

- Assume that $\pi[i:-1] = 2$.
- If, for every $\ell \le i$, $\sigma[\ell] \neq 2$, then $\pi[i:-1] \neq 2$, a contradiction. Hence

$$\{\ell \in [-1,i] \ : \ \sigma[\ell] = 2\} \ \neq \ \emptyset.$$

- Let

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

- $\pi[j:-1] = 2$ (since $2 * a = 2$).
- We claim that $\sigma[\ell] = 1$, for every $j < \ell \le i$.

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

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 $\pi[i:\ell] = 0$.
  $\Rightarrow$

$$\pi[i:-1] = \pi[i:\ell] * \pi[\ell-1:-1] = 0,$$

  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 $\text{OP} : \Sigma^2 \longrightarrow \Sigma$ denote an associative function. A prefix computation over Σ with respect to 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:

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

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

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

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

The Claim

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

implies a reduction of the problem of computing $C[n:1]$ to a Prefix Computation Problem:

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

Prefix Computation Problem - example

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

$\Longrightarrow$ PPC–OR(n) used to design a 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.

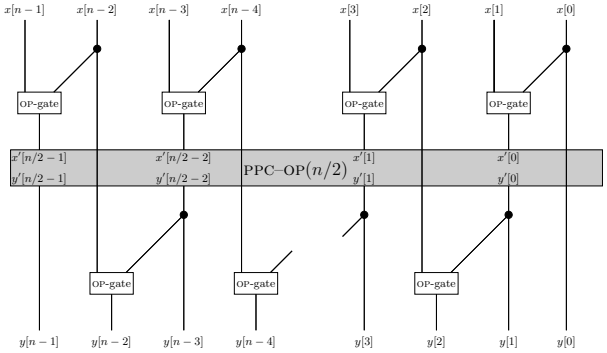
Question: Design a PPC–OP(n) circuit with logarithmic delay and quadratic cost.

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 : $\Sigma^2 \to \Sigma$. 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 PPC–OR(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



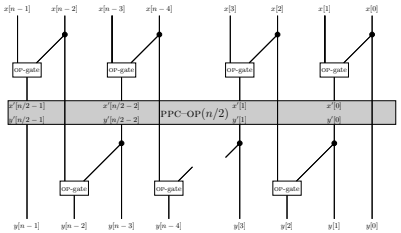
PPC–OP(n) - correctness

- 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$(n/2)$.
- $x'[i] \leftarrow \text{OP}(x[2i+1], x[2i])$.
- Induction hypothesis:

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

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

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

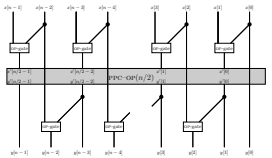


$$d(\text{PPC–OP}(n)) = \begin{cases} d(\text{OP-gate}) & \text{if } n = 2 \\ d(\text{PPC–OP}(n/2)) + 2 \cdot d(\text{OP-gate}) & \text{otherwise.} \end{cases}$$

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

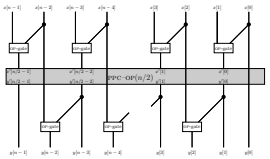
$$\begin{aligned} d(\text{PPC–OP}(n)) &= \Theta(\log n) \\ c(\text{PPC–OP}(n)) &= \Theta(n). \end{aligned}$$

$\Rightarrow$

- $\Sigma = \{0, 1\}$ & 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 $\pi[i:-1]$ 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 $\pi[i:-1] = 2$.
Compare each $\pi[i:-1]$ 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 carry 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).