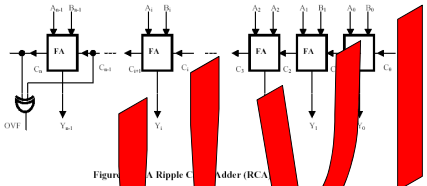
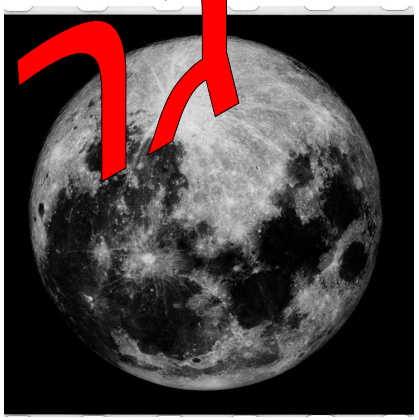
Adders,PPC

On the moon…

- RCA



- Linear Delay
- Linear Cost



גרוע!!!

Lower Bounds – Adder(*n*)

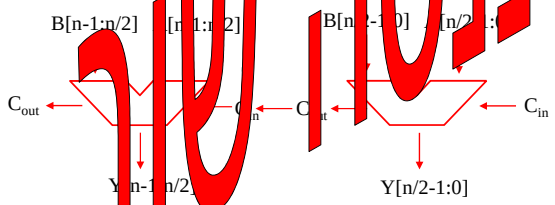
$$Cost(Adder(n)) = \Omega(n)$$

$$Delay(Adder(n)) = \Omega(\log n)$$

ננסה לממש...

First shot

- Split the numbers to LSB and MSB



- What is the problem with this solution?

נסה שוב...

Second Shot :Dancing on both weddings

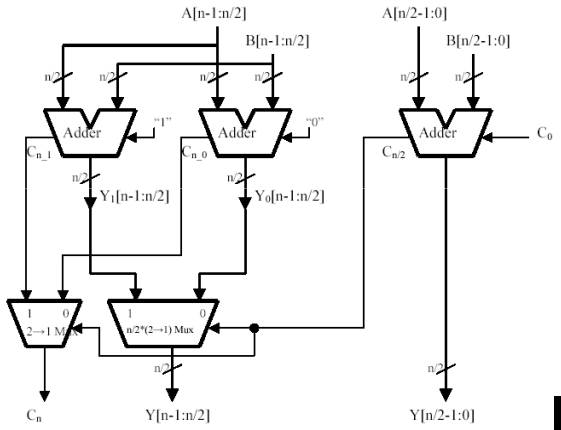


Figure 3.13 - Building a CSA recursively

White box

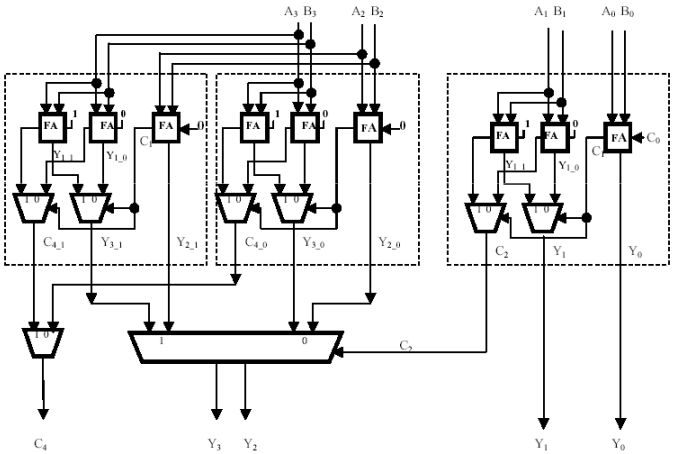
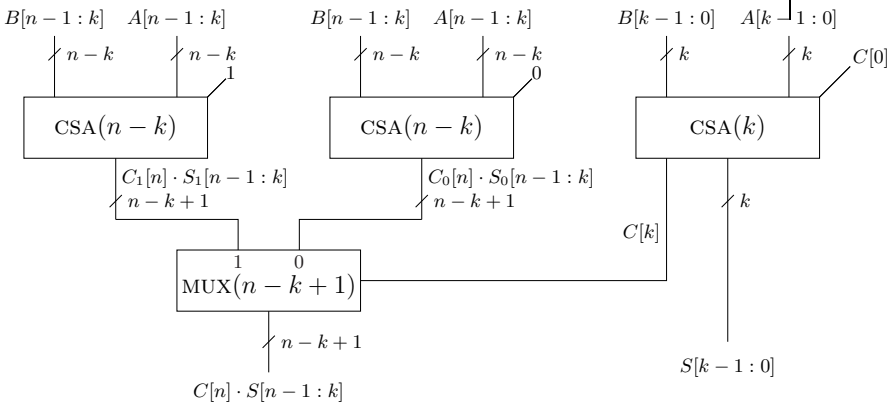


Figure 3.14 – The entire recursion in a 4 bits CSA

Conditional Sum Adder – CSA(*n*)



Delay & Cost

- Delay

$$D(n) = D(n/2) + T_{MUX}$$

$$\Rightarrow D(n) = T_{MUX} \cdot \log n$$

בינתיים הכל בסדר..

- Cost
  - log*n* recursive stages
  - In each stage we triple the number of FA's
  - How many full adders?

    $3^{\log n} = n^{\log 3} = n^{1.5}$
  - How many MUX's?

    $(n/2)+(3/2)\cdot(n/2)+(3/2)^2\cdot(n/2)+\ldots=$

    $=(n/2)\cdot[1+(3/2)^2+(3/2)^3+\ldots]=$

    $=n\cdot(3/2)^{\log n}$

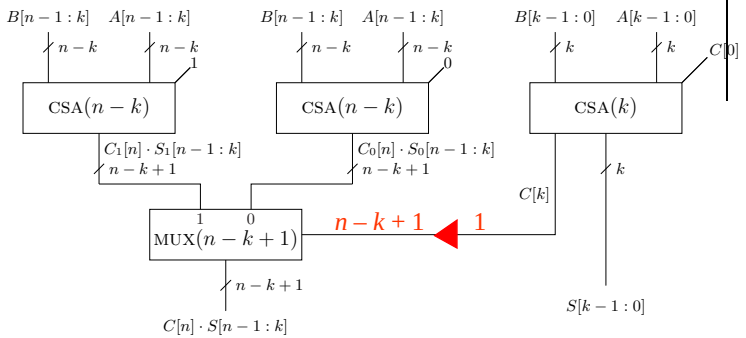
    $=n^{1.58}$

CSA(*n*) - Delay and Cost analysis under fan-out limitations

Question 8.6 (effect of fanout on CSA(n)) *The fanout of the carry-bit $C[k]$ is $n/2+1$ if $k = n/2$. Suppose that we associate a delay of $\log_2(f)$ with a fanout f. How would taking the fanout into account change the delay analysis of a* CSA(n)*?*

Suppose that we associate a cost $O(f)$ with a fanout f. How would taking the fanout into account change the cost analysis of a CSA(n)*?*

CSA(*n*) - Delay and Cost analysis under fan-out limitations



Assume that $n = 2^l$ and set $k = n/2$,

$$d(CSA(n)) = \begin{cases} d(FA) & \text{if } n=1 \\ d(CSA(\tfrac{n}{2})) + \log(\tfrac{n}{2})\cdot d(buf) + d(MUX) & \text{otherwise} \end{cases}$$

$$c(CSA(n)) = \begin{cases} c(FA) & \text{if } n=1 \\ 3\cdot c(CSA(\tfrac{n}{2})) + \Theta(n)\cdot c(buf) + (\tfrac{n}{2}+1)\cdot c(MUX) & \text{otherwise} \end{cases}$$

$n = 2^l$,

for simplicity: $d(FA) = d(MUX) = d(buf) = 1$

$$d(n) = d(\tfrac{n}{2}) + \log(\tfrac{n}{2}) + 1$$

$$= d(\tfrac{n}{2}) + \log(n) - \log(2) + 1$$

$$= d(\tfrac{n}{2}) + \log(n)$$

$$= d(1) + \left(\log(n) + \log(\tfrac{n}{2}) + \log(\tfrac{n}{4}) + \ldots + \log(\tfrac{n}{2^i}) + \ldots + \log(\tfrac{n}{2^l})\right)$$

$$= 1 + \left(l + (l-1) + \cdots + 1 + 0\right)$$

$$= \Theta(l^2)$$

$$= \Theta(\log^2(n))$$

$$c(n) = 3 \cdot c\left(\tfrac{n}{2}\right) + \left(\tfrac{n}{2}+1\right) \cdot c(MUX) + \Theta(n) \cdot c(buf)$$

$$c(n) = 3 \cdot c\left(\tfrac{n}{2}\right) + \Theta(n)$$

Master Theorem for recurrences provides:

$$c(n) = \Theta\left(n^{\log_2 3}\right) \approx \Theta\left(n^{1.58}\right)$$

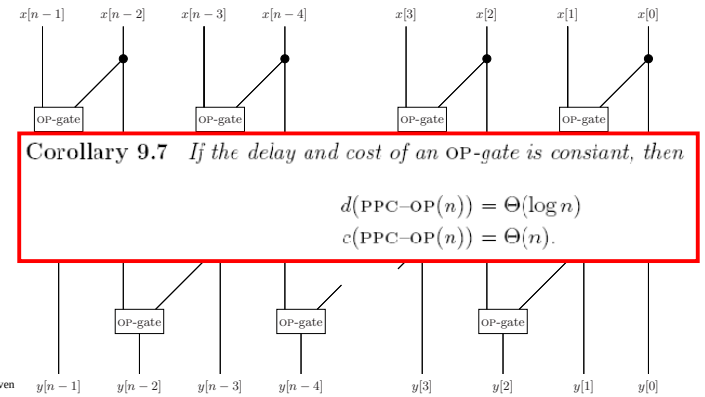
To conclude, in CSA(n) design:

- No change in cost asymptotics due to fan-out limitations.
- Quadratic increase in delay asymptotics due to fan-out limitations.

The PPC-OP Circuit

Definition 9.5 *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*

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



Corollary 9.7 *If the delay and cost of an* OP*-gate is consiant, then*

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

PPC-OR

Definition 6.3 *A parallel prefix* OR *circuit of length* n *is a combinational circuit specified as follows.*

Input: $x[0 : n-1]$.

Output: $y[0 : n-1]$.

Functionality:

$$y[i] = \text{OR}(x[0 : i]).$$

We denote parallel prefix OR circuit of length n by PPC–OR(n).

PPC-OR

Definition 6.4 *A unary priority encoder* U-PENC(n) *is a combinational circuit specified as follows.*

Input: $x[0 : n-1]$.

Output: $y[0 : n-1]$.

Functionality:

$$y[i] = \text{INV}(\text{OR}(x[0 : i])).$$

Example 6.3 *Consider the string* $x[0 : 6] = 0110100$. *The output of a* PPC–OR(7) *is* 0111111. *The output of a* U-PENC(7) *is* 1000000.

PPC-OR

A brute force design of a PPC–OR(n) uses a separate OR-tree for each output bit. The delay of such a design is $O(\log n)$ and the cost is $O(n^2)$. The issue of efficiently combining these trees will be discussed in detail when we discuss parallel prefix computation in the context of fast addition. In the meantime, we present here a (non-optimal) design based on divide-and-conquer.

The method of divide-and-conquer is applicable for designing a PPC–OR(n). We apply divide-and-conquer in the following recursive design. If $n = 1$, then PPC–OR(1) is the trivial design in which $y[0] \leftarrow x[0]$. A recursive design of PPC–OR(n) for $n > 1$ that is a power of 2 is depicted in Figure 6.1.

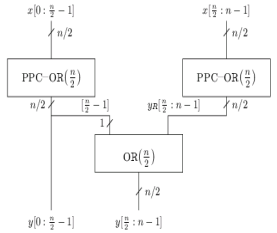


Figure 6.1: A recursive implementation of PPC–OR(n).

PPC-OR

Question 6.1 *This question deals with the recursive design of the* PPC–OR(n) *circuit depicted in Figure 6.1.*

1. *Prove the correctness of the design.*
2. *Extend the design for values of n that are not powers of 2.*
3. *Analyze the delay of the design.*

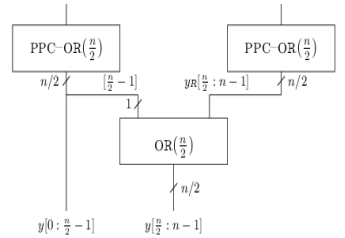


Figure 6.1: A recursive implementation of PPC–OR(n).

Correctness of PPC-OR

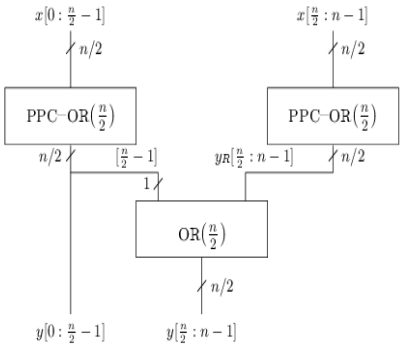


Figure 6.1: A recursive implementation of PPC–OR(n).

Correctness of PPC-OR (cont.)

- The correctness of the design:
  - Note the design is given for n = 2^k .
  - Prove by induction on k.
  - Basis: k = 1, Trivial (correctness of OR).
  - Hypothesis: Assume correct for all j<k+1.
  - Step: Prove for k+1.
  - Observation: If y[s] = 1, then y[t] = 1 for t >= s.

Correctness of PPC-OR (cont.)

Separate into two cases:

If the leading one is on the [n/2:n-1]

→ The leading x values are 0, and hence y[0:n/2-1] = 0 (as should be).

y[n/2:n-1] are correct by the induction.

If the leading one is on the [0:n/2-1]

→ All the trailing outputs should be 1(observation). So y[n/2:n-1] = 1.

y[0:n/2-1] are correct by the induction.

Non 2^k PPC-OR

•For this case, we use the padding idea.

•We pad the number of bits to be a whole power of 2.

•Question: What should it be padded with?

•Answer: Doesn't matter for the PPC-OR as long as you pad at the end of the input and ignore the relevant bits in the output.

Delay analysis of PPC-OR

•d(n) = d(n/2) + 1.

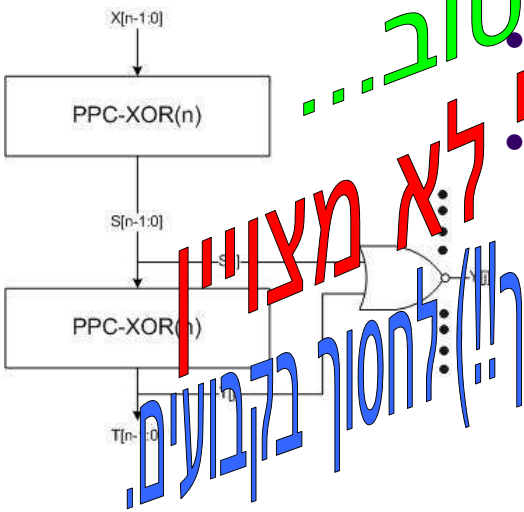
•→ d(n) = log(n).

•For $n \neq 2k$, the padding leads to $d(n) = \lceil \log(n) \rceil$.

Q1 from Exam 2000a

- Input: x[0:n-1]
- Output y[0:n-1]
- Functionality:
  - Let $s[j] = \Sigma\, x[k]$ (k=0..j)
  - Let $t[j] = \Sigma\, s[k]$ (k=0..j)
  - y[j] = 1 if and only if s[j] and t[j] are even.

First Shot



טוב...
אבל לא מעניין
אפשר (ועדיף!!) לחסוך בקבועים.

- The Design is correct.
- Asymptotically optimal:
  - *Delay = O(log(n))*
  - *Cost = O(n)*

Q1 from Exam 2000a (cont.)

- Note that:
  - s[0] = x[0].
  - s[1] = x[0] + x[1].
  - s[2] = x[0] + x[1] + x[2].
  - s[3] = x[0] + x[1] + x[2] + x[3].
  - t[0] = x[0].
  - t[1] = 2x[0] + x[1].
  - t[2] = 3x[0] + 2x[1] + x[2].
  - t[3] = 4x[0] + 3x[1] + 2x[2] + x[3].

Q1 from Exam 2000a (cont.)

- Note that:
  - Parity(s[0]) = XOR(x[0]).
  - Parity(s[1]) = XOR(x[0] + x[1]).
  - Parity(s[2]) = XOR(x[0] + x[1] + x[2]).
  - Parity(s[3]) = XOR(x[0] + x[1] + x[2] + x[3]).
  - Parity(t[0]) = XOR(x[0]).
  - Parity(t[1]) = XOR(x[1]).
  - Parity(t[2]) = XOR(x[0] + x[2]).
  - Parity(t[3]) = XOR(x[1] + x[3]).

- s[0] = x[0].
- s[1] = x[0] + x[1].
- s[2] = x[0] + x[1] + x[2].
- s[3] = x[0] + x[1] + x[2] + x[3].
- t[0] = x[0].
- t[1] = 2x[0] + x[1].
- t[2] = 3x[0] + 2x[1] + x[2].
- t[3] = 4x[0] + 3x[1] + 2x[2] + x[3].

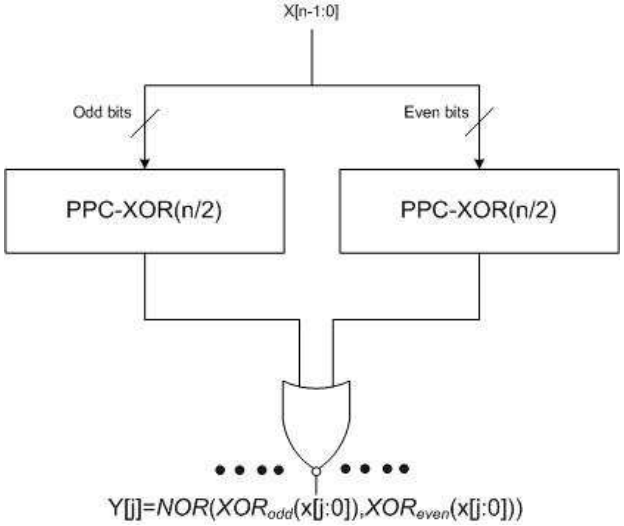
Q1 from Exam 2000a (cont.)

- y[2k+1] = 1 if and only if:
  - The XOR of the odd bits is 0 (For t[2k+1]).
  - The XOR of all bits is 0 (For s[2k+1]).
- In order that y[2k] = 1 if and only if:
  - The XOR of the even bits is 0 (For t[2k]).
  - The XOR of all bits is 0 (For s[2k]).
- → Conclusion: The XOR of even bits and the XOR of odd bits should be 0.

Q1 from Exam 2000a (cont.)

- Conclusion2:
  - We need to calculate the PPC-XOR of the odd bits.
  - We need to calculate the PPC-XOR of the even bits.
  - We combine them with NOR gate, for each j.
- Conclusion3: The circuit is composed of 2 PPC-XOR (n/2) and linear number of gates.

Q1 from Exam 2000a (cont.)



Q1 from Exam 2000a (cont.)

- Using the optimal Implementation of PPC-OP we will achieve:

$$Cost(n) = O(n)$$

$$Delay(n) = O(\log n)$$