Alexander Lifshitz

COMPUTER ARITHMETIC

Lecture #3: *UNSIGNED ADDITION*

Lecturer: Dr. Guy Even
Summarized by: Alexander Lifshitz

In this lecture:

1. Ripple Carry Adder
2. Carry Look-Ahead Adder

Problem Definition:

We would like to design a combinational arithmetic circuit that performs addition of two unsigned binary numbers. Such circuit will be called an "adder". We are also interested to perform addition in such manner that associated adder will be of as low cost and will possess as small propagation time delay as possible.

I. DEFINITIONS

1. Notation

Unsigned addition is actually equivalent to addition of positive integers. Assume A is a positive integer and its binary representation is a string: $\{a_{n-1}a_{n-2}\ldots a_0 \mid a_i \in \{0,1\}\}$, which we denote by $a[n-1:0]$. The binary number represented by $a[n-1:0]$ is denoted by $\langle a[n-1:0]\rangle$ and is defined by

$$A = \langle a[n-1:0]\rangle = \sum_{i=0}^{n-1} a[i]2^i$$

Concatenation is denoted by •, e.g.

$$a[n-1:0] \bullet b[n-1:0] = \{a_{n-1}\ldots a_0 b_{n-1}\ldots b_0\}$$
$$\langle a[n-1:0] \bullet b[n-1:0]\rangle = 2^n \langle a[n-1:0]\rangle + \langle b[n-1:0]\rangle$$

2. Definition of Addition

Input:

$a[n-1:0], \in \{0,1\}^n$
$b[n-1:0], \in \{0,1\}^n$ } Addends

$c[0], \in \{0,1\}$ Carry In

Output:

$s[n-1:0], \in \{0,1\}^n$ Sum

$c[n], \in \{0,1\}$ Carry Out

Specification:

$$\langle a[n-1:0]\rangle + \langle b[n-1:0]\rangle + c[0] = \langle c[n] \bullet s[n-1:0]\rangle = 2^n c[n] + \langle s[n-1:0]\rangle \qquad (1)$$

II. IMPLEMENTATION

The operation of addition is implemented by an n-bit adder, optimal design of which is the subject of the following sections.

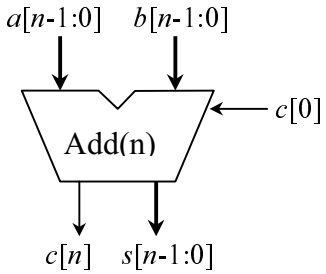
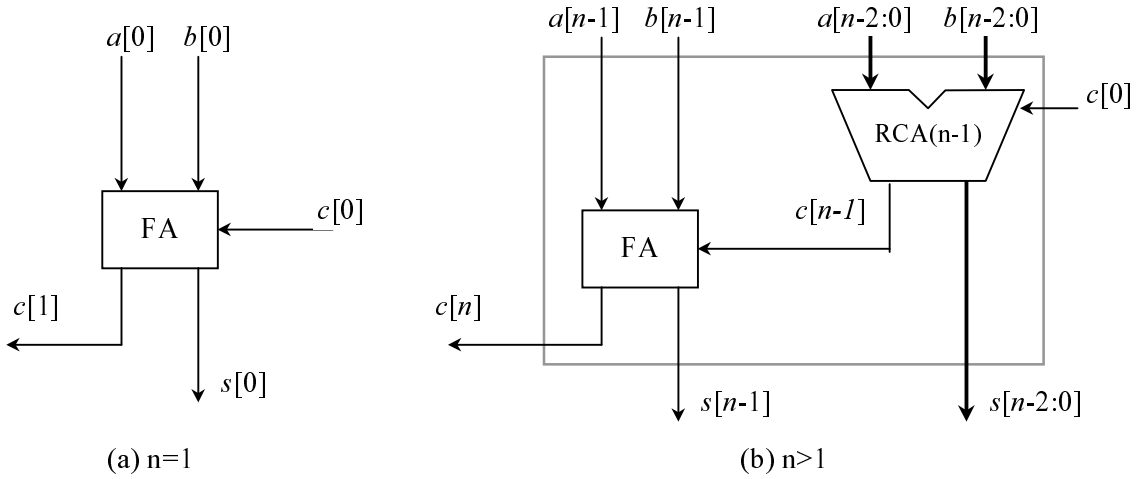


Fig. 1 General n-bit adder

1. RIPPLE CARRY ADDER (RCA)

We define n-bit Ripple Carry Adder in the recursive manner as shown in Fig. 2



(a) n=1 (b) n>1

Fig. 2 RCA(n) – Recursive implementation

Note: In the figure above module FA is a Full-Adder circuit, for which the following is true:

$$a[0]+b[0]=\langle c[1]\bullet s[0]\rangle$$

when

$$\begin{aligned} s[0] &= XOR(a[0],b[0],c[0]) \\ c[1] &= (a[0]\oplus b[0])c[0]+a[0]b[0] \end{aligned} \tag{2}$$

➢ ***Assertion 1.*** *RCA(n) implements Equation 1*

Proof. The proof is by induction on n.

1) For $n=1$ the assertion is true, since RCA(1) is equivalent to full-adder.
2) Assume that for some $n>1$ the assertion is true, then from Fig. 2(b)

$$\langle a[n-2:0]\rangle+\langle b[n-2:0]\rangle+c[0]=\langle c[n-1]\bullet s[n-2:0]\rangle \tag{3}$$

$$\text{and} \quad a[n-1]+b[n-1]+c[n-1]=\langle c[n]\bullet s[n-1]\rangle \tag{4}$$

Multiplying both sides of (4) by 2^{n-1} and summing with (3) obtain

$$\langle a[n-1]\bullet a[n-2:0]\rangle+\langle b[n-1]\bullet b[n-2:0]\rangle+2^{n-1}c[n-1]+c[0]=$$
$$=\langle c[n]\bullet s[n-1]\bullet s[n-2:0]\rangle+2^{n-1}c[n-1]$$

This implies that**:**

$$\langle a[n-1:0]\rangle+\langle b[n-1:0]\rangle+c[0]=\langle c[n]\bullet s[n-1:0]\rangle \tag{5}$$

Q.E.D

Fig. 3 depicts a "flattened" block diagram of RCA(*n*).

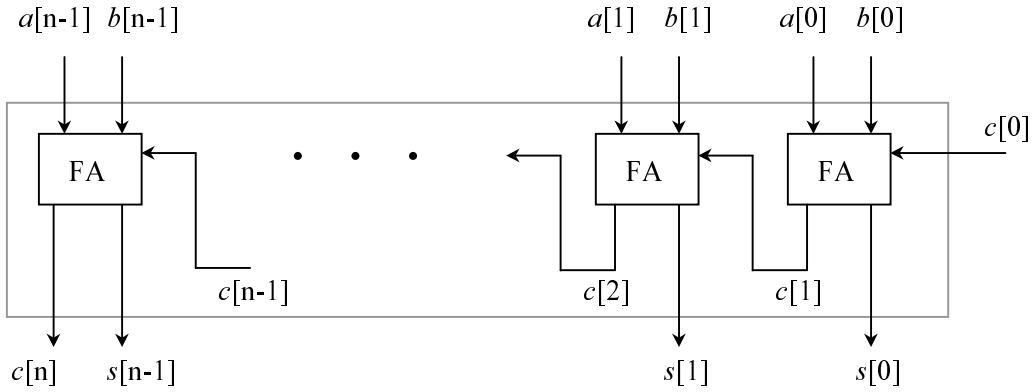


Fig. 3 RCA(n) – Implementation with full-adders

We define $c[i]$ to be the carry bit input to the i'th FA.

Corollary 1: From Assertion 1 it follows that

$$\langle a[i:0]\rangle+\langle b[i:0]\rangle+c[0]=2^{i+1}c[i+1]+s[i:0] \tag{6}$$

Performance evaluation:

Notation:

We'll denote by *T(•)* propagation time delay of the element and by *C(•)* its cost, which is proportional to the total number of basic gates in that element.

First, we note that *C(FA)* is constant and so is *T(FA)*. The computational cost of RCA(n) will be then:

$$C(RCA(n))=\begin{cases}C(FA), n=1\\ C(RCA(n-1))+C(FA), n>1\end{cases}$$

$$\Rightarrow C(RCA(n))=nC(FA)=\theta(n) \tag{7}$$

It is obvious that linear cost is the lowest cost we can achieve, since for each distinct output we need at least one gate and since there are *n* outputs there should be at least *n* gates.

$$T(RCA(n)) = \begin{cases} T(FA), n = 1 \\ T(RCA(n-1)) + T(FA), n > 1 \end{cases}$$

$$\Rightarrow T(RCA(n)) = nT(FA) = \theta(n) \tag{8}$$

A lower bound on the time delay.

➢ ***Assertion 2.*** *c*[*n*] *depends on all bits of the inputs, i.e.*

$$c[n]=f(a[n-1:0], b[n-1:0])$$

Proof. A proof is by contradiction

Assume *c*[*n*] does not depend on *a*[*i*], then given $a[n-1:i+1, i-1:0]$ and $b[n-1:0]$, *c*[*n*] may be determined with no ambiguity.

If we choose

a	*000… 00*	a_i	*000 … 00*
b	*111… 11*	*1*	*000 … 00*

we can easily see that *c*[*n*] depends upon *a*[*i*], therefore our assumption was wrong.

Corollary 2: From Assertion 2 it follows that there exists a path between every input bit and *c*[*n*].

The major drawback of RCA circuit is its propagation time delay, which is of order *n* as can be seen from Equation 8. It is clear that linear time delay results from the fact that carry signal must propagate through all *n* full-adder modules.

From Corollary 2 we may conclude that the lower bound on time delay must be logarithmic, indeed if we assume that all gates may have only two input lines then

$$\begin{aligned} &Delay(k) \geq 1 + Delay(\frac{k}{2}) \\ &\Rightarrow Delay(N) \geq \log_2 N \end{aligned} \tag{9}$$

when *N* is the total number of input bits.

Reductions between computing the sum-string and the carry-string.

From Fig.3 we conclude that

$$\begin{aligned} &s[i] = XOR(a[i], b[i], c[i]) \\ &\Rightarrow c[i] = XOR(a[i], b[i], s[i]) \end{aligned} \tag{10}$$

Therefore, it is possible to compute *s*[*i*] from *c*[*i*] with constant time delay and constant cost. Consequently,

$$\begin{aligned} &C(\text{computing}(s[n-1:0])) = C(\text{computing}(c[n-1:0])) + O(n) \\ &T(\text{computing}(s[n-1:0])) = T(\text{computing}(c[n-1:0])) + O(1) \end{aligned} \tag{11}$$

Now we are ready to ask a question whether it is possible to design a circuit with logarithmic time delay and linear cost. This will be possible, as can be seen from (11), only if we manage to compute all carries in logarithmic time and linear cost. Carry look-ahead adder with parallel prefix computation is the circuit that meets these requirements. We discuss it in the next section.

2. CARRY LOOK-AHEAD ADDER (CLA)

The basic idea of carry look-ahead adder is that all the carries can be generated simultaneously rather than successively.

Idea:

$$\begin{aligned} &a[i]=b[i]=0 \quad \Rightarrow \quad c[i+1]=0 \\ &a[i]=b[i]=1 \quad \Rightarrow \quad c[i+1]=1 \\ &a[i]\neq b[i] \quad \Rightarrow \quad c[i]=c[i+1] \end{aligned} \tag{12}$$

Definition:

$$\sigma[i]=\begin{cases} 2c[0], & i=-1 \\ a[i]+b[i], & 0\le i\le n-1 \end{cases}$$

$$\begin{aligned} &\sigma[i]=0 \quad \text{means "kill"} && \Rightarrow c[i+1]=0 \\ &\sigma[i]=1 \quad \text{means "propagate"} && \Rightarrow c[i+1]=c[i] \\ &\sigma[i]=2 \quad \text{means "generate"} && \Rightarrow c[i+1]=1 \end{aligned}$$

Notation:

$$1^{*}\equiv\{\phi,1,11,111,\ldots\},\quad 1^{k}=\underbrace{11\cdots1}_{k\text{ times}}$$

➢ ***Assertion 3.*** *For all* $-1\le i\le n-1$

$$c[i+1]=1 \quad \Leftrightarrow \quad \exists j\le i:\sigma[i:j]=1^{i-j}\bullet 2 \tag{13}$$

Proof.

$\Leftarrow$ Given $j\le i$ such that $\sigma[i:j]\in\left(1^{*}2\right)$, carry, generated at position *j*, will propagate till position *i*, so that *c*[*i*+1]=1.

$\Rightarrow$ This direction is proved by induction on *i.*

1) Base: *i*=-1,

$$c[0]=1\Rightarrow\sigma[-1]=2 \text{ by definition}$$

2) Step:

Assume that $c[i]=1 \quad \Rightarrow \quad \exists j\le i-1:\sigma[i-1:j]=1^{i-1-j}\bullet 2$

By definition: $c[i+1]=1\Rightarrow \underbrace{a[i]+b[i]}_{\sigma[i]}+c[i]\ge 2$

Keeping in mind that $c[i]\in\{0,1\}$ then only two cases are possible:

One) If $\sigma[i]=2$, then we choose *j*=*i*

Two) If $\sigma[i]=1$, then according to our assumption $\sigma[i:j]=1^{i-j}\bullet 2$

Q.E.D

Define binary operator '$*$' by the following truth table

		b		
	a*∗*b	**0**	**1**	**2**
a	**0**	0	0	0
	1	0	1	2
	2	2	2	2

Properties of '$*$':

1. Operator '$*$' is not commutative: $a*b \neq b*a$
2. Operator '$*$' is associative: $a*(b*c)=(a*b)*c$ (see Assertion 4 below)
3. For $\sigma \in \{0,1,2\}$
   a) $0*\sigma = 0$
   b) $1*\sigma = \sigma$
   c) $2*\sigma = 2$

➢ ***Assertion 4.*** *Operator '$*$' is associative, i.e.* $\forall a,b,c \in \{0,1,2\}\ a*(b*c)=(a*b)*c$

Proof. Consider three cases

One) $a=0$

$$0*(b*c)=(0*b)*c=0 \qquad \text{(Property 3a)}$$

Two) $a=1$

$$1*(b*c)=(1*b)*c=b*c \qquad \text{(Property 3b)}$$

Three) $a=2$

$$2*(b*c)=(2*b)*c=2 \qquad \text{(Property 3c)}$$

Q.E.D

➢ ***Assertion 5.*** *For all* $-1 \le i \le n-1$

$$c[i+1]=1 \quad \Leftrightarrow \quad \sigma[i]*\sigma[i-1]*\ldots*\sigma[-1]=2 \tag{14}$$

Proof. From Assertion 3 $c[i+1]=1 \Leftrightarrow \exists j \le i : \sigma[i:j]=1^{i-j}\bullet 2$

Therefore it suffices to show

$$\exists j \le i : \sigma[i:j]=1^{i-j}\bullet 2 \Leftrightarrow \sigma[i]*\sigma[i-1]*\ldots*\sigma[-1]=2$$

$\Rightarrow$

$$\sigma[i]*\sigma[i-1]*\ldots*\sigma[j]=1*1*\ldots*1*2=2$$

$$\underbrace{(\sigma[i]*\sigma[i-1]*\ldots*\sigma[j])}_{2}*(\sigma[j-1]*\ldots*\sigma[-1])=2 \qquad \text{(by Property 3c)}$$

$\Leftarrow$

Since the product of all σ's is equal to 2 then at least one of them must be 2. (see truth table or Properties 3b, 3c), therefore $\mathcal{A}=\{-1 \le k \le i \,|\, \sigma[k]=2\} \neq \phi$.

Thus $\max\{k|k \in \mathcal{A}\}$ is well defined and we choose $j=\max\{k|k \in \mathcal{A}\}$. Therefore

$$\sigma[j]*\sigma[j-1]*\ldots*\sigma[-1]=2 \quad \text{(Property 3c)}$$

Now, by induction on *i* we may show that in order to obtain $\sigma[i]*\sigma[i-1]*\ldots*\sigma[-1]=2$, all $\sigma_k, j+1 \le k \le i$ must be equal to 1.

Q.E.D

Define $\Pi[i] \equiv \prod_{k=-1}^{i}\sigma[k] \equiv \sigma[i]*\sigma[i-1]*\ldots*\sigma[-1]$

Notice that, since $\sigma[-1] \in \{0,2\}$, then from truth table of $'*'$ we conclude that $\Pi[i] \in \{0,2\}$.

Our goal now is to compute all $\Pi[i]$ simultaneously with logarithmic time delay and linear cost. When all $\Pi[i]$ are computed, then according to Assertion 5 we may convert them into carry bits by the following rule:

$$\text{if } \Pi[i]=0 \text{ then } c[i+1]=0$$
$$\text{if } \Pi[i]=2 \text{ then } c[i+1]=1$$

Once all carries are computed, we may compute $s[n-1:0]$ according to (10).

A. Implementation with Binary Trees

Assume we have a circuit, which implements $'*'$ (Fig. 4). It is clear that implementation of $'*'$ requires constant time and constant cost. We may compute all $\Pi[i]$ in parallel using a separate binary tree for each $\Pi[i]$. For example, in Fig. 5 implementation of $\Pi[6]$ is shown.

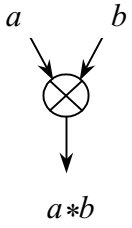


Fig. 4 $'*'$ gate

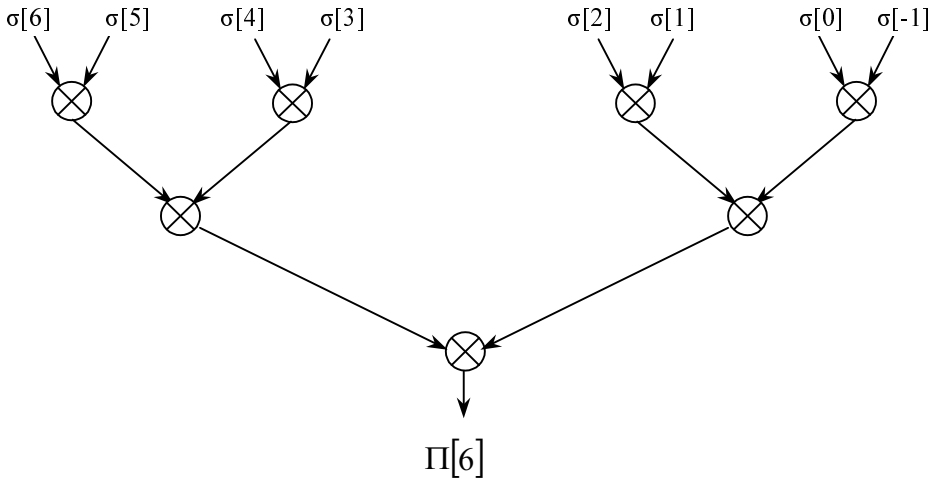


Fig. 5 Computation of $\Pi[6]$

Performance evaluation.

It is obvious that for each i

$$T(\Pi[i])=O(\log(i))$$
$$C(\Pi[i])=O(i) \qquad (15)$$

Therefore, computation of all $\Pi[i]$ will still require logarithmic time delay (since computed in parallel), but will be of squared cost, i.e.

$$T\left(\Pi[n-1:0]\right) = O\left(\log(n)\right)$$
$$C\left(\Pi[n-1:0]\right) = 1+2+\ldots+n = \frac{n(n+1)}{2} = O\left(n^2\right) \tag{16}$$

However, we are interested to obtain linear cost, therefore this implementation is not satisfactory.

B. Implementation with PPC (Parallel Prefix Computation)

The problem of parallel prefix computation is defined as follows:

Input: $\Sigma = \{0,1,2\}$
$*: \Sigma \times \Sigma \to \Sigma$ an associative operator
$\sigma[n-1:0] \in \Sigma^n$

Output: $\Pi[i], 0 \le i \le n-1$, where $\Pi[i] \equiv \sigma[i] * \sigma[i-1] * \ldots * \sigma[0]$ (17)

We define PPC circuit in recursive manner as depicted in Fig. 6

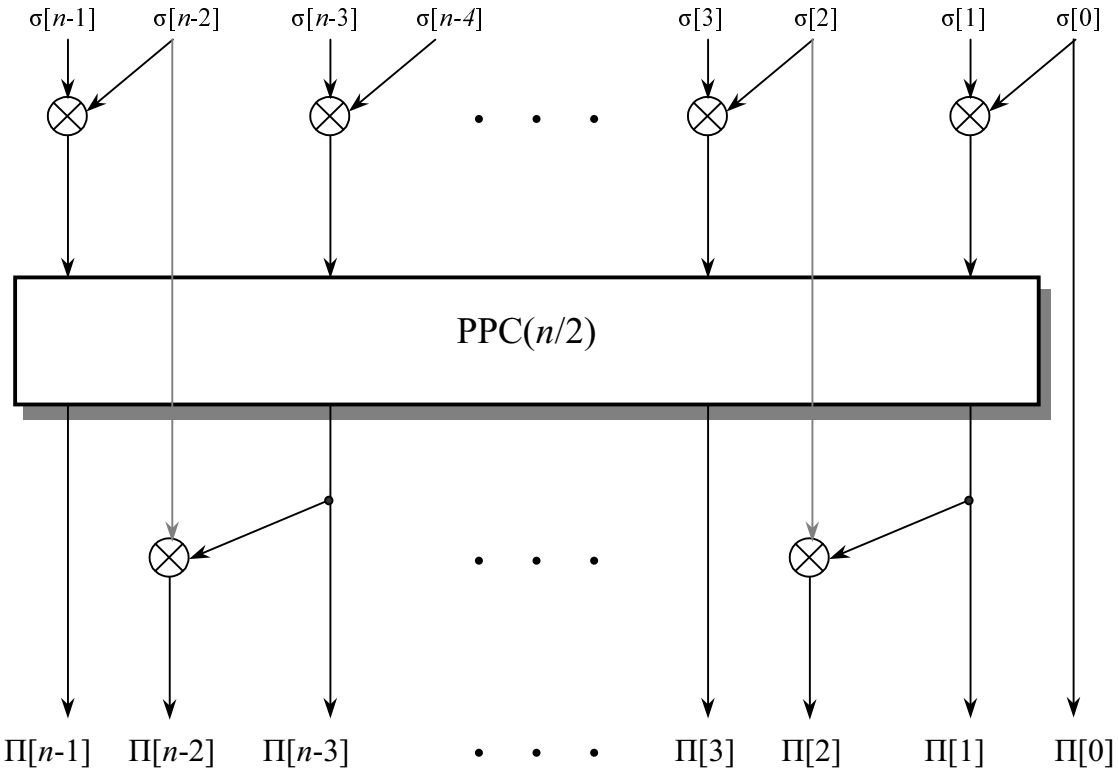


Fig. 6 PPC(n)

- ***Assertion 6.*** PPC(n) computes $\Pi[i], 0 \le i \le n-1$ defined in (17)

  Proof: Assume $n = 2^k$. The proof is by induction on k.

  1) For k=0 (n=1), the assertion is true, since $\Pi[0] = \sigma[0]$.
  2) Assume $\mathrm{PPC}\left(\frac{n}{2}\right)$ works correctly.

We denote its inputs by $\hat{\sigma}_i, 0 \le i \le \frac{n}{2}-1$

The outputs of $\mathrm{PPC}\left(\frac{n}{2}\right)$ are denoted by $\hat{\Pi}[i] = \hat{\sigma}[i] * \hat{\sigma}[i-1] * \ldots * \hat{\sigma}[0], 0 \le i \le \frac{n}{2}-1$

From Fig. 6 it is clear that $\hat{\sigma}[i] = \sigma[2i+1] * \sigma[2i]$, therefore

$$\hat{\Pi}[i] = \sigma[2i+1] * \sigma[2i] * \ldots * \sigma[0] = \Pi[2i+1] \tag{18}$$

Consequently, outputs of $\mathrm{PPC}\left(\frac{n}{2}\right)$ are outputs of PPC(n) with odd index.

To compute outputs with even index we perform (see Fig. 6)

$$\Pi[2i] = \sigma[2i] * \Pi[2i-1], 1 \le i \le \frac{n}{2}-1 \tag{19}$$

Q.E.D

Performance evaluation.

$$C(PPC(n)) = \begin{cases} 0, n = 1 \\ C\left(PPC\left(\frac{n}{2}\right)\right) + (n-1)C(*), n > 1 \end{cases} \tag{20}$$

Solving recursion in (17) we obtain

$$C(PPC(n)) \le C(*)\left[n + \frac{n}{2} + \frac{n}{4} + \frac{n}{8} + \cdots\right] \le 2n\underbrace{C(*)}_{const} = \theta(n) \tag{21}$$

$$T(PPC(n)) = \begin{cases} 0, n = 1 \\ T\left(PPC\left(\frac{n}{2}\right)\right) + 2T(*), n > 1 \end{cases} \tag{22}$$

Solving recursion in (19) we obtain

$$T(PPC(n)) \le 2T(*)\underbrace{[1+1+\cdots+1]}_{\log n} = \theta(\log n) \tag{23}$$

Conclusion: *Carry Look-Ahead Adder with Parallel Prefix Computation is the optimal scheme for adding two unsigned binary numbers in the sense that it performs addition with logarithmic time delay and has linear cost.*