International Journal of Electronics Engineering Research.
ISSN 0975-6450 Volume 10, Number 1 (2018) pp. 9-18

http://www.ripublication.com

Fast Parallel Integer Adder in Binary Representation

Duggirala Meher Krishna
Student, Department of Electronics and Communication Engineering,
Gayatri Vidya Parishad College of Engineering (Autonomous), Madhurawada,
Visakhapatnam, Andhra Pradesh, India.

Duggirala Ravi
Professor, Department of Computer Science and Engineering,
Gayatri Vidya Parishad College of Engineering (Autonomous), Madhurawada,
Visakhapatnam, Andhra Pradesh, India.

Abstract

An integer adder for integers in the binary representation is one of the basic operations of any digital processor. For adding two integers of N bits each, the serial adder takes as many clock ticks. For achieving higher speeds, parallel circuits are discussed in the literature, and these circuits usually operate in two levels. At the lower level, integers represented by blocks of smaller number of bits are added, and in a cascade of stages in the next level, the carries produced in previous addition operations are summed to the augends. These circuits perform addition of integers of N bits in about $\log_2 N$ number of clock ticks and $O(N * \log_2 N)$ space. In this paper, we describe a fast method and an improvement of it. The first attempt resembles the operation method of the merge sort algorithm, from which some important properties of carries produced in each stage are analysed and assimilated, resulting in a parallel adder that runs in about $\log_2 N$ number of clock ticks and $O(N * \log_2 N)$ space. Then the crucial insights are brought to fruition in an improved design, which takes 2 clock ticks to perform the addition operation requiring only $O(N^2)$ space. The number of bits N is chosen usually to be a positive integer power of 2. The speedup is achieved by special purpose circuits for increment operations by 2^i, for $0 \leq i \leq N-1$, each operation taking only a single clock tick to complete.

Keywords: digital circuits, integer addition, parallelization methods.

INTRODUCTION

Addition operation of integers represented in binary is a basic operation on most, if not all, modern digital processors. The sequential or serial circuit for performing addition of two *N* bit integers takes *N* clock ticks. For parallelization of the addition operation, the main issue is to find an efficient method to deal with the carry produced by addition operation of smaller number of bits. Ripple carry adder or Carry propagate adder, Carry look-ahead adder, Carry skip adder, Manchester chain adder, Carry select adders, Prefix adders, Multi-operand adder, Carry save adder, Pipelined parallel adder (see [2—6]). These circuits perform addition of integers of *N* bits in about $\log_2 N$ number of clock ticks and $O(N * \log_2 N)$ space (see [1--3]).

In the next section, we present a *k*-stage cascade circuit, where $N = 2^k$, performing addition operation in only *k* clock ticks, requiring $k * 2^{k-1} - 1$ space for the special purpose circuits for carry addition. Some important insights are gained in the design of this circuit, which are exploited for realizing an improved circuit that adds in constant time, *i. e*., in 2 time delays, but requiring only at most $\frac{N*(N+1)}{2}$ space.

PARALLEL BINARY ADDER

The steps involved in a parallel adder, resembling the merge sort algorithm, are described in the following algorithm:

First Attempt Parallel Adder Circuit

1. Let the number of bits in the integers be $N = 2^k$, for some positive integer *k*.
2. Let the input integers in the binary form be $a_{N-1}\, a_{N-2}\, \ldots\, a_0$ and $b_{N-1}\, b_{N-2}\, \ldots\, b_0$.
3. Initially, compute 2^{k-1} sums of two bits each, $s_{1,\ 2*i+1}\, s_{1,\ 2*i}$, and the corresponding carries $c_{1,\ i}$, such that, the binary bit sequences $s_{1,\ 2*i+1}\, s_{1,\ 2*i}$ are the two lesser significant bits obtained by adding $a_{2*i+1}\, a_{2*i}$ and $b_{2*i+1}\, b_{12*i}$, with a carry bit $c_{1,\ i}$, for $0 \leq i \leq 2^{k-1} - 1$. This operation is performed separately by 2^{k-1} many programmable logic arrays or sequential adders, which compute in parallel for each index i, where $0 \leq i \leq 2^{k-1} - 1$.
4. For $l = 1, 2, \ldots, k-1$, in steps of 1, in the ascending order, after 2^{k-l} the sums of 2^l bits each, $s_{l,\ i*2^l+2^l-1}\, s_{l,\ i*2^l+2^l-2} \ldots s_{l,\ i*2^l}$, together with the carries $c_{l,\ i}$, for $0 \leq i \leq 2^{k-l} - 1$, the following increment operation is performed : $s_{l,\ (2*i+1)*2^l+2^l-1}\, s_{l,\ (2*i+1)*2^l+2^l-2} \ldots s_{l,\ (2*i+1)*2^l}$ is

incremented by $c_{l,\ 2*i}$, to get an auxiliary carry $d_{l+1,\ i}$ and a sum $s_{l+1,\ i*2^{l+1}+2^{l+1}-1}\ s_{l+1,\ i*2^{l+1}+2^{l+1}-2} \cdots s_{l+1,\ i*2^{l+1}}$. where

$$s_{l+1,\ i*2^{l+1}+2^{l+1}-1}\ s_{l+1,\ i*2^{l+1}+2^{l+1}-2} \cdots s_{l+1,\ i*2^{l+1}+2^{l}}$$

$$= \quad s_{l,\ (2*i+1)*2^{l}+2^{l}-1}\ s_{l,\ (2*i+1)*2^{l}+2^{l}-2} \cdots s_{l,\ (2*i+1)*2^{l}}$$

as obtained after addition of the carry $c_{l,\ 2*i}$, and

$$s_{l+1,\ i*2^{l+1}+2^{l+1}+2^{l}-1}\ s_{l+1,\ i*2^{l+1}+2^{l+1}+2^{l}-2} \cdots s_{l+1,\ i*2^{l+1}}$$

$$= \quad s_{l,\ (2*i)*2^{l}+2^{l}-1}\ s_{l,\ (2*i)*2^{l}+2^{l}-2} \cdots s_{l,\ (2*i)*2^{l}} \quad ;$$

the carry $c_{l+1,\ i}$ is $c_{l,\ 2*i+1} \vee d_{l+1,\ i}$, for $0 \leq i \leq 2^{k-l-1}-1$; the number of bits in the sum in the $(l+1)$-th cascade stage is 2^{l+1} and there are 2^{k-l-1} such sums and carries. Although there appears to be the need for an auxiliary carry as just described, the combined operation of increment and carry update can be performed as a single step by incrementing $(2^{l}+1)$-bit integer in the binary form represented by

$$c_{l,\ 2*i+1}\ \ s_{l,\ (2*i+1)*2^{l}+2^{l}-1}\ s_{l,\ (2*i+1)*2^{l}+2^{l}-2} \cdots s_{l,\ (2*i+1)*2^{l}}$$

to get the binary sequence

$$c_{l+1,\ i}\ s_{l+1,\ i*2^{l+1}+2^{l+1}-1}\ s_{l+1,\ i*2^{l+1}+2^{l+1}-2} \cdots s_{l+1,\ i*2^{l+1}+2^{l}}$$

for $0 \leq i \leq 2^{k-l-1}-1$; this increment operation can be performed in a single clock tick by a special purpose circuit, which indentifies the least index j , where $0 \leq j \leq 2^{l}$, such that all the least significant bits up to (but not including) index j are 1 and the bit with index j is 0, by means of $(2^{l}+1)$ AND-gates implemented by negated NOR-gates, and instantly complements the bits with index j upto the least significant bit; if $c_{l,\ 2*i+1}$ is 0, then there is one such index j, and if $c_{l,\ 2*i+1}$ is 1, then it must have been produced in the previous, *i. e.*, l-th , cascade stage, and therefore, the integer represented by the binary sequence $s_{l,\ (2*i+1)*2^{l}+2^{l}-1}\ s_{l,\ (2*i+1)*2^{l}+2^{l}-2} \cdots s_{l,\ (2*i+1)*2^{l}}$ can be at most $2^{2^{l}}-2$, ensuring that there is such an index j as just being discussed, and the increment operation cannot further produce a carry,

5. The final sum is $s_{k,\ 2^{k}-1}\ s_{k,\ 2^{k}-2} \cdots s_{k,\ 0}$ with final carry $c_{k,\ 0}$.

Claim: The integer represented by $s_{m,\ i*2^m+2^m-1}\, s_{m,\ i*2^m+2^m-2} \ldots s_{m,\ i*2^m}$, together with the carries $c_{m,\ i}$, is the result of addition of the integers represented by the binary sequences $a_{\ i*2^m+2^m-1}\, a_{\ i*2^m+2^m-2} \ldots a_{\ i*2^m}$ and $b_{\ i*2^m+2^m-1}\, b_{\ i*2^m+2^m-2} \ldots b_{\ i*2^m}$, as expressed in the following, for $0 \leq i \leq 2^{k-m} - 1$ and $1 \leq m \leq k$:

$$c_{m,\ i} * 2^{2^m} + \sum_{j=0}^{2^m-1} s_{m,\ i*2^m+j} * 2^j =$$

$$\sum_{j=0}^{2^m-1} a_{i*2^m+j} * 2^j + \sum_{j=0}^{2^m-1} b_{i*2^m+j} * 2^j \qquad (1)$$

Proof: The claim is true for $m = 1$, by the construction in Step 3. Now, it is assumed be true through all cascade stages up to and including m and l, where $1 \leq m \leq l \leq k - 1$. Entering the second for-loop indexed by $0 \leq i \leq 2^{k-l-1} - 1$, in Step 4, it is required to show that the assertion in (1) holds true, for $m = l + 1$. Now, by inductive hypothesis, the following is assumed to hold true, for $0 \leq i \leq 2^{k-l-1} - 1$:

$$c_{l,\ 2*i} * 2^{2^l} + \sum_{j=0}^{2^l-1} s_{l,\ (2*i)*2^l+j} * 2^j =$$

$$\sum_{j=0}^{2^l-1} a_{(2*i)*2^l+j} * 2^j + \sum_{j=0}^{2^l-1} b_{(2*i)*2^l+j} * 2^j \quad and \qquad (2)$$

$$c_{l,\ 2*i+1} * 2^{2^l} + \sum_{j=0}^{2^l-1} s_{l,\ (2*i+1)*2^l+j} * 2^j =$$

$$\sum_{j=0}^{2^l-1} a_{(2*i+1)*2^l+j} * 2^j + \sum_{j=0}^{2^l-1} b_{(2*i+1)*2^l+j} * 2^j \qquad (3)$$

Multiplying by 2^{2^l} the equation (3) throughout, the following is obtained, for $0 \leq i \leq 2^{k-l-1} - 1$.:

$$c_{l,\ 2*i+1} * 2^{2^{l+1}} + \sum_{j=0}^{2^l-1} s_{l,\ (2*i+1)*2^l+j} * 2^{2^l+j} =$$

$$\sum_{j=0}^{2^l-1} a_{(2*i+1)*2^l+j} * 2^{2^l+j} + \sum_{j=0}^{2^l-1} b_{(2*i+1)*2^l+j} * 2^{2^l+j} \quad (4)$$

and adding the corresponding sides of the equations (4) and (2), the following is obtained :

$$c_{l,\ 2*i+1} * 2^{2^{l+1}} + \sum_{j=0}^{2^l-1} s_{l,\ (2*i+1)*2^l+j} * 2^{2^l+j} +$$

$$c_{l,\ 2*i} * 2^{2^l} + \sum_{j=0}^{2^l-1} s_{l,\ (2*i)*2^l+j} * 2^j =$$

$$\sum_{j=0}^{2^l-1} a_{(2*i+1)*2^l+j} * 2^{2^l+j} + \sum_{j=0}^{2^l-1} b_{(2*i+1)*2^l+j} * 2^{2^l+j}$$

$$+ \sum_{j=0}^{2^l-1} a_{(2*i)*2^l+j} * 2^j + \sum_{j=0}^{2^l-1} b_{(2*i)*2^l+j} * 2^j$$

$$= \sum_{j=0}^{2^l-1} a_{(2*i)*2^l+2^l+j} * 2^{2^l+j} + \sum_{j=0}^{2^l-1} b_{(2*i)*2^l+2^l+j} * 2^{2^l+j}$$

$$+ \sum_{j=0}^{2^l-1} a_{(2*i)*2^l+j} * 2^j + \sum_{j=0}^{2^l-1} b_{(2*i)*2^l+j} * 2^j$$

$$= \sum_{j=0}^{2^{l+1}-1} a_{(2*i)*2^l+j} * 2^j + \sum_{j=0}^{2^{l+1}-1} b_{(2*i)*2^l+j} * 2^j$$

$$= \sum_{j=0}^{2^{l+1}-1} a_{i*2^{l+1}+j} * 2^j + \sum_{j=0}^{2^{l+1}-1} b_{i*2^{l+1}+j} * 2^j$$

where the last term is the result of addition of the integers represented by the binary sequences $a_{i*2^{l+1}+2^{l+1}-1}\ a_{i*2^{l+1}+2^{l+1}-2} \ldots a_{i*2^{l+1}}$ and $b_{i*2^{l+1}+2^{l+1}-1}\ b_{i*2^{l+1}+2^{l+1}-2} \ldots b_{i*2^{l+1}}$, for $0 \leq i \leq 2^{k-l-1} - 1$. Now, either of the summands on the right hand side of (3) is at most $2^{2^l} - 1$, and therefore, their sum is

at most $2^{2^{l+1}} - 2$, while the maximum integer that can be represented by the left hand side is (3) is $2^{2^{l+1}} - 1$, which means that the single bit $c_{l,\ 2*i}$ can be added to the left hand side of (3) without an overflow, for $0 \leq i \leq 2^{k-l-1} - 1$. Thus, by the result of the carry increment in Step 4, the following holds, for $0 \leq i \leq 2^{k-l-1} - 1$:

$$c_{l+1,\ i} * 2^{2^l} + \sum_{j=0}^{2^l-1} s_{l+1,\ i*2^{l+1}+j} * 2^j =$$

$$c_{l,\ 2*i+1} * 2^{2^l} + \sum_{j=0}^{2^l-1} s_{l,\ (2*i+1)*2^l+j} * 2^j + c_{l,\ 2*i} \quad (5)$$

and

$$\sum_{j=0}^{2^l-1} s_{l+1,\ i*2^{l+1}+j} * 2^j = \sum_{j=0}^{2^l-1} s_{l,\ (2*i)*2^l+j} * 2^j \quad (6)$$

Now multiplying both sides of (5) by 2^{2^l} and adding the corresponding sides in (6) to the result, the following is obtained, for $0 \leq i \leq 2^{k-l-1} - 1$:

$$c_{l+1,\ i} * 2^{2^{l+1}} + \sum_{j=0}^{2^{l+1}-1} s_{l+1,\ i*2^{l+1}+j} * 2^j =$$

$$c_{l+1,\ i} * 2^{2^{l+1}} + \sum_{j=0}^{2^l-1} s_{l+1,\ i*2^{l+1}+2^l+j} * 2^{2^l+j}$$

$$+ \sum_{j=0}^{2^l-1} s_{l+1,\ i*2^{l+1}+j} * 2^j =$$

$$c_{l,\ 2*i+1} * 2^{2^{l+1}} + \sum_{j=0}^{2^l-1} s_{l,\ (2*i+1)*2^l+j} * 2^{2^l+j} +$$

$$c_{l,\ 2*i} * 2^{2^l} + \sum_{j=0}^{2^l-1} s_{l,\ (2*i)*2^l+j} * 2^j$$

$$= \sum_{j=0}^{2^{l+1}-1} a_{i*2^{l+1}+j} * 2^j + \sum_{j=0}^{2^{l+1}-1} b_{i*2^{l+1}+j} * 2^j$$

which proves the claim, for $m = l + 1$.

Circuit Complexity: We estimate the number of special purpose AND-gates required for performing the carry addition operation in Step 4. For $1 \leq l \leq k-1$, there are 2^{k-l} many sum sequences in the input at level l, and, of these, only 2^{k-l-1} many that constitute the higher precision subsequence at level $(l+1)$ are required to be incremented. Each sequence to undergo increment operation needs $(2^l + 1)$ AND-gates. Thus the total number of special purpose AND-gates of this implementation is found as follows:

$$\sum_{l=1}^{k-1} (2^l + 1) * 2^{k-l-1} = \sum_{l=1}^{k-1} 2^{k-1} + 2^{k-l-1} = (k-1) * 2^{k-1} + (2^{k-1} - 1)$$

$$= k * 2^{k-1} - 1 = \frac{N * \log_2 N}{2} - 1$$

The Usefulness of Special Purpose Circuits for Addition or Subtraction by 2^i

A processor can be furnished with a special purpose circuit for incrementing an integer represented by *N*-bit sequence by 2^i, for $0 \leq i \leq N-1$. This operation is useful in the following contexts: taking 2's complement operation, subtraction operation, increment of instruction pointer and as a special instruction, dedicated for this purpose, similar to shift operation. The special purpose circuit is expected to take only one clock tick to perform the specified increment operation. Further, for adding an integer represented by very sparsely occupied 1-bits, the addition operation can be implemented by a sequence of such instructions. The subtraction operation by 2^i, for $0 \leq i \leq N-1$, can be realized complementarily.

Improved Parallel Adder Circuit

1. Let the input integers in the binary form be $a_{N-1}\, a_{N-2}$ ... a_0 and $b_{N-1}\, b_{N-2}$ ... b_0.
2. In the first step, compute N sums of two bits each, $s_i = a_i\, XOR\, b_i$ and $c_i = a_i\, AND\, b_i$, for $0 \leq i \leq N-1$. Set $s_N = c_{N-1}$. All the operations are performed taking only 1 time delay.
3. In the second step, the carries c_i, for $0 \leq i \leq N-2$, are added in parallel, in a single time delay, using about $\frac{N*(N+1)}{2}$ special purpose AND-gates, for $0 \leq i < j \leq N$, as follows:
    a. Let SC_AND(i, j) =

$$\begin{cases} \overline{s_{i+1}}\ AND\ c_i, \quad if\ \ j = i+1\ , \qquad and \\ \overline{s_j}\ AND\ s_{j-1}\ AND\ \ldots\ AND\ s_{i+1}\ AND\ c_i\ , \\ \qquad\qquad if\ \ j > i+1 \end{cases}$$

b. It is shown that for each index *i*, if $c_i = 1$, then there exists exactly one index *j*, where $i + 1 \leq j \leq N$, such that SC_AND$(i,\ j) = 1$, for $0 \leq i \leq N - 2$: if $s_N = 1$, then $s_{N-1} = 0$, and so, in any case, there is an index *j*, where $i + 1 \leq j \leq N$, such that $s_j = 0$; the uniqueness of the index can be easily inferred; and $c_l = 0$, for $i + 1 \leq l \leq j - 1$, which means that there are no more carries to be added in between the indexes $i + 1$ and $j - 1$, when $i + 2 \leq j \leq N$.
c. Let *j* be the unique index as in (b), such that SC_AND$(i,\ j) = 1$ and $i + 1 \leq j \leq N$; Then, SC_AND instantly complements the bit string $s_j\ s_{j-1}\ \ldots\ s_{i+1}$, for $0 \leq i \leq N - 2$.
d. The sum together with the carry is $s_N\ s_{N-1}\ s_{N-2}\ \ldots\ s_0$.

Proof of Correctness of the Algorithm: Assume that there are *r* carries of 1s to be added, where $1 \leq r \leq N$. Let $0 \leq i_1 \leq \cdots \leq i_r \leq N - 1$ be the indexes such that $c_{i_l} = 1$, for $1 \leq l \leq r$, for some *r*, where $1 \leq r \leq N$. If $r = 1$, then c_{i_1} is the only carry to be added, and this case is easily handled by the algorithm. Let $2 \leq r \leq N$. The main point in the proof is that the addition operation of a carry c_{i_l} does not affect the addition operation of the carry $c_{i_{l+1}}$, for $1 \leq l \leq r - 1$, as observed in the following. The bit $s_{i_{l+1}}$must be 0, because $c_{i_{l+1}} = 1$ and $c_{i_{l+1}} s_{i_{l+1}}$, being the result of adding only two bits, $a_{i_{l+1}}$ and $b_{i_{l+1}}$, cannot be the bit string 11, for $1 \leq l \leq r - 1$. Thus, there exists an index j_l, such that $i_l + 1 \leq j_l \leq i_{l+1}$ and SC_AND$(i_l,\ j_l) = 1$, for $1 \leq l \leq r - 1$. Now, since there are no carries of 1s in between the indexes $i_l + 1$ and $i_{l+1} - 1$, the complementation of the string $s_{j_l}\ s_{j_l-1}\ \ldots\ s_{i_l+1}$ is equivalent to adding 1 to the corresponding integer represented by it, without affecting the carry addition of $c_{i_{l+1}}$, for $1 \leq l \leq r - 1$. The last carry c_{i_r} is added, as if it were lone carry to be added.

It may be observed that addition of two (2*N*)-bit integers takes only 3 time delays by means of two *N*-bit adders as just described. Two lower and higher significant *N*-bit integers are added, and if a carry is produced by the addition operation of the two lower significant *N*-bit integers, then it is added to the sum of the two higher significant *N*-bit integers, in just one time delay. The time delay of multiplication of two *N*-bit integers is determined by the time delay of addition of (2*N*)-bit integers, requiring about $\log_2 N$ of adders. For each index *i*, a Cauchy sum of product is formed, which corresponds to the coefficient of 2^i, for $0 \leq i \leq 2N - 1$. Since there

are at most N products of two bits in each sum, they are added in $\log_2 N$ stages, to get $2N$ coefficients represented by at most $\log_2 N$ bits each. Then, the bit-planes of the coefficients are rearranged into $\log_2 N$ integers of at most $2N$ bits, which are added by the $\log_2 N$ integer adders in parallel, in at most $\log_2 \log_2 N$ stages.

In the first attempt algorithm described in the previous section, we started at leaf node with sums of two bits of *a*'s and *b*'s each, at a time. If we assume a similar initialization to compute the sum *s* and carry *c* bits, we could reduce the space required by a factor of 2 for the special purpose AND-gates, in the algorithm just described in this section. Another possibility for reduction of the number of special purpose AND-gates, for the sake of economy, is to consider a two-level cascaded implementation. In the first cascade stage, about $\sqrt{N}$ blocks are taken for addition in parallel, each block consisting of again about $\sqrt{N}$ sum and carry bits. In this circuit design, the number of special purpose AND-gates in the first cascade stage would be about $\sqrt{N} \times \left[\frac{\sqrt{N}(\sqrt{N}+1)}{2}\right] \approx \frac{N(\sqrt{N}+1)}{2}$. In the second cascade stage, there are about $\sqrt{N}$ carry bits to be added, which would require about $\frac{\sqrt{N}(N+1)}{2}$ special purpose AND-gates. Thus, the total number of special purpose AND-gates could be about $N\sqrt{N} = N^{\frac{3}{2}}$. Combining with the previous observation, *i. e.*, starting with two bits of *a*'s and *b*'s to get the s and c bits in the initialization step, it is possible to realize a $2N$-bit integer adder performing the addition operation in three clock ticks, requiring only $N\sqrt{N} = N^{\frac{3}{2}}$ special purpose AND-gates. For typical numbers, if $N = 64$, then $N\sqrt{N} = 512$, as compared to $\frac{N(N+1)}{2} = 1056$, both circuits taking only three clock ticks to add two 128-bit integers. On a 64-bit processor, 128-bit integer adder is needed for multiplication operation. The first attempt design circuit of the previous section would need $\frac{128*7}{2} - 1 = 447$ special purpose AND-gates, performing the addition of two 128-bit integers in about 7 clock ticks, while a two-stage cascade circuit would need about 512 special purpose AND-gates, to repeat, performing the addition of two 128-bit integers in 3 clock ticks. In slide 83 of [8], it is stated that the Pentium processor performs the 32-bit integer addition in 11 gate delays.

REFERENCES

[1] Avinash Shrivastava, and Chandrahas Sahu, "Performance analysis of parallel prefix adder based on FPGA", International Journal of Engineering Trends and Technology (IJETT), vol. 21, no. 6, March 2015, pp. 281 – 286.

[2] Jasbir Kaur, and Lalit Sood, "Comparison between various types of adder topologies", IJCST, vol. 6, issue 1, Jan-March 2015

[3] Richard P. Brent, and H. T. Kung, "A regular layout for parallel adders", IEEE Transactions on Computers, vol. 31, no. 03, March 1982, pp. 260—264.
[4] Vitit Kantabutra, "Designing optimum one-level carry-skip adders", IEEE Transactions on Computers, vol.42, no.6, June 1993.
[5] Luigi Dadda and Vincenzo Piuri, "Pipelined adders", IEEE Transactions on Computers, vol.45, no.3, March 1996.
[6] Jien-Chung Lo, "A fast binary adder with conditional carry generation" IEEE Transactions on Computers, vol.46, no.2, February 1997.
[7] A. Guyot, B. Hochet and J.M. Muller, "A way to build efficient carry-skip adders", IEEE Transactions on Computers, pp.1144-1152, October 1987.
[8] Steven Rudich, "Great theoretical ideas in computer science", CMU Lecture 17, CS 15-251, Carnegie Mellon University, March 2004