International Journal of Electronics Engineering Research.
ISSN 0975-6450 Volume 11, Number 2 (2019) pp. 99-105

http://www.ripublication.com

Fast Parallel Integer Multiplier in Binary Representation

Duggirala Meher Krishna

Gayatri Vidya Parishad College of Engineering (Autonomous), Madhurawada, Visakhapatnam – 530 048, Andhra Pradesh, India
Email: duggiralameherkrishna@gmail.com

Duggirala Ravi

Gayatri Vidya Parishad College of Engineering (Autonomous), Madhurawada, Visakhapatnam – 530 048, Andhra Pradesh, India
Email: ravi@gvpce.ac.in ; drdravi2000@yahoo.com

Abstract

Addition and multiplication of integers in the binary representation are basic operations of any digital processor. For adding two integers of N bits each, the serial adder takes as many clock ticks. In this paper, we describe a fast method for integer addition, 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. The usefulness of this adder for multiplication operation is discussed. The standard multiplication method utilizes quantizer and 3-bit to 2-bit consolidation circuits to produce an integer that represents in binary the number of 1s in a column corresponding to a place (weighted coefficient) of nonnegative integer power of 2. The last two consolidated integers are added by an adder in the end.

1. 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. For various methods discussed in the

literature, *viz*, 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 [3—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, 2]).

In the next section, we present a circuit that adds in constant time, *i. e.*, in 2 time delays, but requiring only at most $\frac{N*(N+1)}{2}$ space. Further improvements, including the application of the adder for fast multiplication of two integers represented in binary, are discussed towards the end of the article. The standard multiplication method utilizes quantizer and 3-bit to 2-bit consolidation circuits to produce an integer that represents in binary the number of 1s in a column corresponding to a place (weighted coefficient) of nonnegative integer power of 2. The last two consolidated integers are added by an adder in the end.

2. PARALLEL BINARY ADDER

The steps in a fast parallel adder are described in the following algorithm:

Parallel Adder Circuit

1. Let the input integers in the binary form be $a_{N-1}\, a_{N-2}\, \ldots\, a_0$ and $b_{N-1}\, b_{N-2}\, \ldots\, b_0$.
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 $s_N = 0$. All the operations are performed in parallel taking only 1 time delay.
3. In the second step, the carries c_i , for $0 \leq i \leq N-1$, 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 $\text{SC\_AND}(i, j) =$
   $$\begin{cases} \overline{s_{i+1}}\ AND\ c_i, & if\ \ j = i+1\ , \quad and \\ \overline{s_j}\ AND\ s_{j-1}\, AND\, \ldots\, AND\, s_{i+1}\, AND\, c_i\ , & if\ \ j > i+1 \end{cases}$$

   (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 $\text{SC\_AND}(i, j) = 1$, for $0 \leq i \leq N-1$, since s_N is initialized to 0 ; the uniqueness of the index can be easily deduced ; moreover, $c_l = 0$, 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 $\text{SC\_AND}(i,\ j) = 1$ 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 $0 \leq i \leq N - 1$.

(D) The sum together with the carry is $s_N\ s_{N-1}\ s_{N-2}\ \ldots\ s_0$, where s_N is the carry or overflow bit.

Proof of Correctness of the Algorithm:. Let $0 \leq i_1 < \cdots < i_r \leq N - 1$ 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 $\text{SC\_AND}(i_l,\ j_l) = 1$, 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.

3. MULTIPLICATION OF TWO INTEGERS IN BINARY REPRESENTATION

The time delay of multiplication of two *N*-bit integers is determined mostly by the time delay of addition of (2*N*)-bit integers, requiring at least one (2*N*)-bit adder and consolidation circuits that reduce a larger number of integers to a smaller number of integers for addition, such that the sum of the integers, before and after consolidation, is the same. For each index i, a Cauchy sum of product is formed, which corresponds to the coefficient of 2^i, for $0 \leq i \leq 2N - 1$. Then, the bit-planes of the coefficients are rearranged, similar to rearranging the order of summation of a doubly indexed sum, into $\log_2 N$ number of integers of at most (2*N*) bits, with (*N*+1) quantization levels, which can be classified by (*N*+1) comparators (Chapter 7 of [9]). The quantization intervals are recognized by two adjacent voltage levels. The voltages of the bits in a column corresponding to the same place of a nonnegative integer power

of 2 are connected in series, to get the sum of voltages, which encodes the number of 1s in the column. If the bits are sensitive to current measurements, then they are added in parallel, to form the sum of currents, and the common junction point is connected to the ground by an additional resistor. Thus, in any case, the sum of the voltages is measured at a particular junction point. The sum falls (after accounting for small errors and fluctuations) somewhere in the middle of exactly one quantization interval, which is recognized by the conjugation of the conditions that (*i*) the upper limit voltage is larger, and (*ii*) the lower limit voltage is smaller than the sum of the voltages in a column. The conjunction of the two conditions is fed to a switching circuit (Chapter 8 of [9]), which switches an associative memory entry containing the bit pattern that encodes the integer to count the number of 1s in the column. Thus, the sum of $r \geq 3$ integers can be reduced to a sum of $\lfloor \log_2 r \rfloor + 1$ integers, in a constant number of (which may be two) clock ticks. However, when the number of integers to be added falls to a small number (such as below 6), the consolidation method described in Slide 45 of [8] may be faster than the quantizer circuit. The quantizer based consolidation method achieves higher speed, when the number of integers to be consolidated is larger than a prescribed number, and as such may be qualified to be called optimal, owing to its constant time operational performance. The final two integers after the consolidation stages are added to get the integer which is the product of the two integers, given as input in the beginning.

The consolidation operation is illustrated for the 64-bit multiplication. Initially, there are 64 integers to be added, which are aligned properly adjusting for the respective binary places. Two cases are described for comparison: one with only 3-bit to 2-bit consolidation circuits described in Slide 45 of [8], and the other with quantizers for about two stages followed by 3-bit to 2-bit consolidation circuits described in Slide 45 of [8] in the remaining stages, until both reduce the sum of the initially given 64 integers into a sum of two integers, where the latter could be 128-bit long, unlike in the input, which are at most 64-bit long. The quantizer is assumed to take two clock ticks to produce the required integers, as follows: in the first clock tick, the lower and upper bounds of interval of quantization are detected, consequently initiating the corresponding switching circuit, and in the second clock tick, the initiated switching circuit activates an associative memory unit, which places the contents in appropriate places, taking care also of the binary places, positioning the resulting integers as in a staircase, for the next stage. The circuit initialization phase is sensitive to the leading or trailing edge of a switching (initiating) pulse, giving the pipeline or cascade effect, which is partly folded into (overlapped with) the duration of the switching pulse. The edges are not always sharp or crisp, and edge sensitivity is exploited for gaining speedup in cascading (during both feed-forward and feedback stages of) compound circuits. The measurements for settling time for the overall circuit are explicitly performed, by trying out its response for various pulses that arise in typical (empirical) situations.

(A) With only 3-bit to 2-bit consolidation. The numbers of integers to be consolidated in a sequence of stages taking only one clock tick per stage are as follows (where the serial number stands for the clock tick offset number): (1) 64 to 43 (with only 63 to 42 consolidation and one integer left out), (2) 43 to 29 (with only 42 to 28 consolidation and one integer left out), (3) 29 to 20 (with only 27 to 18 consolidation and two integers left out), (4) 20 to 14 (with only 18 to 12 consolidation and two integers left out), (5) 14 to 10 (with only 12 to 8 consolidation and two integers left out), (6) 10 to 7 (with only 9 to 6 consolidation and one integer left out), (7) 7 to 5 (with only 6 to 4 consolidation and one integer left out), (8) 5 to 4 (with only 3 to 2 consolidation and two integers left out), (9) 4 to 3 (with only 3 to 2 consolidation and one integer left out) and (10) 3 to 2 consolidation, taking 10 clock ticks to complete the task. The overall consolidation factor for consolidating 64 integers into 2 integers is 32, and with a consolidation factor of $\left(\frac{3}{2}\right)$ per stage, the lower bound for the number of stages is $\lfloor \log_{(3/2)}\ (32) \rfloor = \lfloor 8.547 \ldots \rfloor = 9$. The overrun of the number of stages is caused by the indivisibility of the number of integers to be consolidated by the integer 3 in some stages.

It may be observed that, with required quantizers to add up 14 bits to produce 4-bit integers in binary representation, steps (5) through (8) can be replaced with a single quantizer step, which may take two clock ticks to perform this particular subtask, saving two clock ticks. As another opportunity, again with required quantizers to add up 7 bits to produce 3-bit integers in binary representation, for instance, steps (7) through (9) can be replaced with a single quantizer step, which may take two clock ticks to perform this particular subtask, but saving just one clock tick.

(B) With quantizers and 3-bit to 2-bit consolidation. The numbers of integers to be consolidated in a sequence of stages taking one or two clock ticks per stage, depending on the particular stage, are as follows (the serial number marking for the end of the clock tick offset number): (2) 64 to 7 (with 63-bit to 6-bit consolidation based on quantizers, taking two clock ticks, and one integer left out), (4) 7 to 3 (with 7-bit to 3-bit consolidation based on quantizers, taking two clock ticks), and (5) 3 to 2 consolidation (with only 3-bit to 2-bit consolidation, taking one clock tick), taking 5 clock ticks to complete the task.

The total time needed is calculated as follows: 5 clock ticks for consolidation of 64 to 2 integers of at most 128 bits each, added to about 3 clock ticks for the addition of the two 128-bit integers, to get the final result of multiplication of the two input 64-bit integers in about 8 clock ticks, in case (B), and, about 9 clock ticks obtained by the theoretical lower bound for consolidation of 64 to 2 integers of at most 128 bits each,

added to about 15 clock ticks for the addition of the two 128-bit integers, to get the final result of multiplication of the two input 64-bit integers in about 24 clock ticks, in case (A). Thus, the speedup factor is at least $\frac{24}{8} = 3$.

In the following discussion, the circuit complexity for the two cases discussed above is estimated. The initial 64 number of 64-bit integers are arranged in a parallelogram staircase, in the standard presentation. They can be arranged to foom a nabla (▼) or Delta (▲) shape staring at 127-bit integer in the first row, followed by 125-bit integer in the second row and so on, until 1-bit integer in the last (64-th) row. In the first stage, since 64 itself is not divisible by 3, there are 63 rows to be consolidated, and 121 number of 3-bit to 2-bit consolidation circuits, required in the second row, followed by 115 number of 3-bit to 2-bit consolidation circuits, required in the fifth row, until one 3-bit to 2-bit consolidation circuit, in the 62-nd row, skipping two rows in between, with 6 circuits less in succession. These consolidation circuits must perform in parallel in the first stage at least. This number can also be arrived at by observing that 21 rows of 3-bit to 2-bit consolidation circuits are required to consolidate 63 rows to 42 rows in the first step. Thus, there are $\sum_{i=0}^{20}(6 * i + 1) = 1 + 7 + \cdots + 121 = 21 * 61 = 1281$ number of 3-bit to 2-bit circuits (associative memory units) required, in case (A), each circuit containing 8 entries of 2-bit associative memory. Now, in case (B), in addition to 128 number of 3-bit to 2-bit consolidation circuits in the final consolidation stage, the number of 63-bit to 6-bit quantizers needed is about 128, with possible reuse in the second stage, and if no reuse is possible, another 128 number of 7-bit to 3-bit quantizers in the second consolidation stage are needed. For comparison, 128 number of 63-bit to 6-bit quantizers hold 64*128 = 8192 associative memory entries of 6-bits each, while 1281-128 = 1153 number of 3-bit to 2-bit consolidation circuits hold 1153 * 8 = 9224 number of 2-bit associative memory entries. If reuse of the quantizers in the second stage is possible, the associative memory space requirement in case (B) is less than 3 times that in case (A), with a speedup factor of at least 3. It may be observed that the well-known Amdahl's law for speedup bound is applicable for the same programs or circuits, when executed in parallel by replication of resources. An interesting situation is when different tasks together require some resources in total, which can be allocated to them to execute in parallel, without requiring any additional resources. Quantizers are more commonly well-known in the analog-to-digital (ADC) converters. However, the inputs to the quantizers in this section take only finitely many discrete values, and the required precision for the lower and upper bounds of the interval of quantization for the sum offers considerable tolerance for accounting for small errors and fluctuations in the current or voltage measurements taken at the input.

REFERENCES

[1] Avinash Shrivastava, and Chandrahas Sahu, "Performance analysis of parallel prefix adder based on FPGA", *International Journal of Engineering Trends and Technology* (IJETT), Volume 21, Number 6, March 2015, pp. 281 – 286.

[2] Jasbir Kaur, and Lalit Sood, "Comparison between various types of adder topologies", *International Journal of Computer Science and Technology* (IJCST˘), Volume 6, Issue 1, Jan-March 2015, pp. 62 – 66

[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.

[9] Jacob Millman, and Herbert Taub, "Pulse, Digital, and Switching Waveforms (Devices and Circuits for their Generation and Processing)", *McGraw-Hill International*, 1965.