An Efficient Algorithm to Compute the Square of a Binary Number

M.G. Arlis[1] and P.C. Perera[2]

[1]*Independent Researcher*

[2]*Department of Engineering Mathematics, Faculty of Engineering*
University of Peradeniya

Introduction

Much of contemporary research is geared to develop and devise efficient algorithms that are applicable in solving very complicated mathematical problems. Nevertheless, the development of efficient algorithms to perform calculations that are used ubiquitously, enhances the speed of almost every calculation performed by a computer. In this endeavor, it is aimed at developing an efficient algorithm that can be used in squaring binary integer.

It should be noted that for the process of squaring only the significant figures of the number under consideration do matter. Let $\hat{N} \in \mathbb{N}$ be a binary number. Let k be the largest nonnegative integer such that $2^k | \hat{N}$. Thus, the significant figures can be obtained by chopping off the last k integers of $\hat{N}$. Letting $N = \dfrac{\hat{N}}{2^k}$, the following procedure can be used to compute N^2 and $\hat{N}^2$ as well.

Preliminaries

Since the square of unity is trivial, suppose $N \neq 1$. Let p_1 be the least positive integer such that $m_0 = N < 2^{p_1}$. It is obvious that $p_1 \geq 2$. Define m_1 by letting $2^{p_1} - m_1 = N = m_0$, it turns out that

$$m_0^2 = N^2 = 2^{p_1}(2^{p_1} - 2m_1) + {m_1}^2,$$

and $m_0 > m_1$. If $m_1 \neq 1$, let p_2 be the least positive integer such that $m_1 < 2^{p_2}$.

Obviously, $p_2 \geq 2$. Letting $2^{p_2} - m_2 = m_1$, it turns out that

$$m_1^2 = 2^{p_2}(2^{p_2} - 2m_2) + {m_2}^2,$$

for some m_2 with $m_1 > m_2$. Since the finite sequence $\{m_i\}$ is a strictly decreasing sequence of odd positive numbers, $m_l = 1$ for some $l \in \mathbb{N}$. Iteratively, it can be shown that

$$N^2 = 1 + \sum_{i=1}^{l} 2^{p_i}(2^{p_i} - 2m_i). \qquad (1)$$

The terminating condition for the algorithm is that $m_i = 1$ for some $i \in \mathbb{N}$.

It turns out that for $i \in \mathbb{N}$, $2^{p_i} - 2m_i$ can be computed simply by means of shift registers. See the following example.

The main result

Theorem 1. *Let N be a binary integer consisting of a significant part with L binary transitions. Then*

$$N^2 = 1 + \sum_{i=1}^{L} 2^{p_i}(2^{p_i} - 2m_i). \qquad (2)$$

Remark 1. *In the light of Theorem 1, it can be concluded that the steps required for the computation of N^2 is exactly the transitions present in the significant part of N.*

The application of this algorithm is outlined with the help of the following examples.

Examples

Example 1. *Let* $m_0 = 1101101_2$. *The value of* $p_1 = 7$ *since* $1000000_2 < 1101101_2 < 10000000_2$. *($p_1$ is, in fact, equal to the number of digits of m_0.)* $m_1 = 10000000_2 - 1101101_2 = 10011_2$

$$\begin{aligned} 2^{p_1} - 2m_1 &= (10000000_2 - 100110_2) \\ &= 1011010_2 \end{aligned}$$

It turns out that

m_0	=	1	1	0	1	1	0	1
a_1	=	1	0	1	1	0	1	0

It can be observed that $a_1 = 2^{p_1} - 2m_1$ *can be obtained simply by shifting the bits of* m_0 *by one place to the left while introducing a 'zero' for the least significant bit.*

Example 2. *Let* $b = 1110001_2$.

m_0	=	1	1	1	0	0	1
m_1	=	–	–	–	1	1	1
m_2	=	–	–	–	–	–	1

and

$2^{p_1} - 2m_1$	=	1	1	0	0	1	0
$2^{p_2} - 2m_2$	=	–	–	–	1	1	0

Note that $2^{p_i} - 2m_i$ *is directly obtained by shifting* m_{i-1} *by one left shift while discarding the most significant bit of* m_{i-1}. *The number* m_i *is obtained by keeping the least significant digit of* m_{i-1} *as it is, while complementing the other bits.*
Since, in this case, it yields

$$N = m_0 = m_2^2 + \sum_{i=1}^{2} 2^{p_i} \left(2^{p_i} - 2m_i\right), \quad (3)$$

where $m_2 = 1$, *we have*

$2^{p_1}(2^{p_1} - 2m_1)$	=	1	1	0	0		
$2^{p_2}(2^{p_2} - 2m_2)$	=	–	–	–	–		
m_2^2	=	–	–	–	–		
N^2	=	1	1	0	0		

	1	0	–	–	–	–	–	–
	–	–	1	1	0	–	–	–
...	–	–	–	–	–	–	–	1
	1	0	1	1	0	0	0	1

Note that $p_1 = 6$ *and* $p_2 = 3$. *It can be observed that* $111001_2 = 57$ *and* $57^2 = 3249 = 110010110001_2$.

Discussion

The terminating condition for the algorithm is that $m_i = 1$ for some i. Thus, if $m_1 \neq 1$, letting m_2 be the least positive integer such that $m_1 < 2^{m_2}$, add $m_1 - m_2$ number of zeros to the end of the current answer in the register and repeat the aforementioned steps until $m_i = 1$ for some i. Note that $2^{p_i} - 2m_i$ is directly obtained by left-shifting m_{i-1} by one bit while discarding the most significant bit of m_{i-1}. Moreover, m_i is obtained by keeping the least significant digit of m_{i-1} as it is, while complementing the other bits.

If the binary integer N of our interest consists of m significant bits, N^2 has exactly $2m$ or $2m - 1$ significant bits. Clearly, when N becomes large, the overflow error occurred can be eliminated by using several registers in series. The above fact is not possible in the case of other existing algorithms.

Conclusions

In Phillips et al. (2001), the square of long integers was obtained by removing repeated digit products from the accumulation tree which is called optimized squaring using pre-computed partial products. It can be shown that the algorithm devised in this paper supersedes the method of optimized squaring using pre-computed partial products presented here. Shifting involved in the regular process of squaring is not an issue in the context of this new algorithm. Additions and subtractions are also kept minimal in this case.

References

Phillips, B. (2001) Optimized squaring of long integers using pre-computed partial products, *Proceedings. 15th IEEE Symposium on Computational Arithmetic*