Reducing the Number of Counters Needed for Integer Multiplication

Robert M. Owens Raminder S. Bajwa Mary Jane Irwin
Department of Computer Science and Engineering
Pennsylvania State University
University Park, PA., 16802

Abstract

In this paper we consider the problem of multiplying reasonably small integers using fewer counters than that required by straightforward partial product accumulation. Not surprisingly the method we use is based on the observation that integer multiplication can be formulated as aperiodic convolution. However, instead of using something like the Fast Fourier Transform to compute the aperiodic convolution, we use what are known as a "fast" convolution algorithms. In this way we can construct multipliers for as small as eighteen bit integers which use fewer counters than that required by straightforward partial product accumulation. Because of the perceived "overhead" involved with an aperiodic formulation of integer multiplication, the ability to do this goes somewhat against the conventional wisdom that aperiodic formulation of integer multiplication gains an advantage over a straightforward partial product formulation only for fairly large integers.

1 Partial Product Formulation

We will first consider the partial product formulation of integer multiplication. The formulation we describe is not new and has previously been considered in several different forms by several authors [4, 3]. A good overview of this formulation is given in [1]; we present only a sketch in the following. We assume in this paper that an integer representation system is defined by integer numbers r, the radix, and n, the precision. Within a given system, an integer x is representation by a sequence of integer numbers x_0, x_1, $\ldots$, x_{n-1} such that

$$x = \sum_{i=0}^{n-1} x_i r^i$$

and $x_i < r$. The equivalent binary precision b of a given representation system is given by

$$b = n log_2(r)$$

In turn, a given representation system is said to have b bits of precision. We will say in this paper that two different representation systems are equivalent if they have the same binary precision. We assume in this paper that the radix is a power of two. Under this assumption translation of a number between equivalent representation systems is trivial (it is just a regrouping of the underlying bits).

We will now consider the partial product formulation of the product of two b bit integers. In this case, the product is given by

$$z = xy = \left(\sum_{i=0}^{b-1} x_i 2^i\right)\left(\sum_{i=0}^{b-1} y_i 2^i\right)$$

$$= \sum_{i=0}^{b-1}\sum_{j=0}^{b-1} x_i\ y_j\ 2^{i+j}$$

It is instructive to view this summation as an array where initially each element is one of the $x_i y_j$'s such that the elements in the i^{th} row are those $x_j y_k$'s, such that $j = i$ and $0 \leq k < b$ and the elements in the i^{th} column are those $x_j y_k$'s, such that $j + k \equiv i$. Note that each $x_i y_j$ has the value of either 0 or 1. Hence we have an array of 0 and 1's. The final product can be obtained by consecutively applying a counter to each column of the array. Each application of counters reduces the number of rows. Application terminates when there is only a single row.

Counters can be identified by a pair $(p,\ q)$ of numbers. The first number p indicates the number of inputs (which are all input from a given row). The second number q indicates the number of outputs (which are all output to consecutive columns). In this paper we will consider only the use of (3, 2) and (2, 2) counters. It is then straightforward to show [3] that for $b \geq 3$ the minimum number of counters needed to reduce the partial produce array is $b(b-2)$ (3, 2) counters and b (2, 2) counters. In Table 1 we present

1063-6889/95 $4.00 © 1995 IEEE

Table 1
Partial Product Costs

bits	(3, 2) counters	(2, 2) counters	cost
6	24	6	426
12	120	12	2004
18	288	18	4734
24	528	24	8616
30	840	30	13650
36	1224	36	19836
60	3480	60	56100
126	15624	126	250866

data for multipliers of some selected sizes whose implementation is based on a partial product formulation.

The figures in the column labeled cost in Table 1 represent the sum of gate inputs for all the gates needed to implement the counters. This cost function is used by many VLSI CAD programs to estimate the chip area needed to implement given circuit.

2 Aperiodic Formulation

We will now consider the aperiodic formulation of the product of two b bit integers. In this case, the product is given by

$$z = x\,y = \left(\sum_{i=0}^{n-1} x_i r^i\right)\left(\sum_{i=0}^{n-1} y_i r^i\right)$$

$$= \sum_{i=0}^{2n-2} z_i r^i$$

where

$$z_i = \sum_{j=0}^{n-1} x_j y_{i-j}$$

Note that the z_i's are nothing more than the aperiodic convolution of the x_i's and the y_i's. However, instead of using something like the Fast Fourier Transform to compute this aperiodic convolution, we will consider in this paper using two of the so called "fast" convolution algorithms.

We will first assume that $n = 2$ and that 2 evenly divides b. Hence, while x and y still have b bits of precision, x_0, x_1, y_0, and y_1 only have $b/2$ bit of precision. The first fast convolution algorithm [2] we will consider is given in Figure 1.

Note that the fast algorithm uses only three smaller multiplications while the straightforward way to compute the aperiodic convolution needs four. A simple argument shows that three smaller multiplications is minimal for $n = 2$. This algorithm can be used recursively to build multipliers which use order wise fewer counters than partial product multipliers. However, in this paper we will assume that the smaller submultiplications are simply implemented as partial product multipliers.

$$\begin{aligned} a_0 &= x_0 & b_0 &= y_0 \\ a_1 &= x_1 & b_1 &= y_1 \\ a_2 &= x_0 + x_1 & b_2 &= y_0 + y_1 \\ m_i &= a_i b_i, \quad i = 0, 1, 2 \\ z_0 &= m_0 \\ z_1 &= m_2 - m_0 - m_1 \\ z_2 &= m_1 \end{aligned}$$

Fig. 1. Fast Aperiodic Convolution Algorithm (n = 2)

Table 2
Operation Breakdown for n = 2

Operation	Number
addition of two $b/2$ bit integers producing a $b/2+1$ bit sum	1
multiplication of two $b/2$ bit integers producing a b bit product	2
multiplication of two $b/2+1$ bit integers producing a $b+2$ bit product	1
addition of three $b+2$ bit integers producing a $b+1$ bit sum	1
addition of three $b+1$ bit integers producing a $2b$ bit sum	1

Note that while it would initially appear tempting to generate the final product z directly from the m_i's, we instead elect to first generate the z_i's and then in turn generate the final product z from the z_i's. We do this for the following reason. If the final product were generated directly then some of the m_i's would need to be negated. In some cases this would result in values of approximately $2b$ bit of precision having to be added together in order to generate the final product. However, in generating the z_i's while some of the m_i's still need to be negated, the result has at most $b+1$ bits of precision. This occurs because the z_i's themselves are both positive and have at most $b+1$ bits of precision.

An examination of the algorithm then reveals that the following operations are required.

A program was written to compute the number of counters needed for multipliers whose implementation is based on this fast convolution algorithm. The data produced by this program is conservative in that the

Table 3
Fast $n = 2$ Algorithm Costs

bits	(3, 2) counters	(2, 2) counters	cost	over-head
6	36	16	688	0.573
12	129	28	2260	0.354
18	276	40	4696	0.256
24	477	52	7996	0.201
30	732	64	12160	0.166
36	1041	76	17188	0.141
60	2817	124	45940	0.088
126	12156	256	196288	0.043

program does not exploit many tricks which can be used to reduce the number of counters. In Table 3 we present data for multipliers of some select sizes.

The figures in the column labeled overhead in Table 3 represent the percentage of counters not being used to directly implement the underlying smaller submultipliers.

We will now assume that $n = 3$ and that 3 evenly divides b. Hence, while x and y still have b bits of precision, x_0, x_1, x_2, y_0, y_1, and y_2 only have $b/3$ bit of precision. The second fast convolution algorithm we will consider is given in Figure 2. While we can site no reference for this algorithm, we do not claim it's discovery.

Note that this fast algorithm uses only six smaller multiplications while the straightforward way to compute the aperiodic convolution needs nine. An exhaustive search shows that six smaller multiplications is minimal for $n = 3$. Note that periodic convolution needs a minimum of 5 multiplications but we are using aperiodic convolution. Again this algorithm can be used recursively to build multipliers which use order wise fewer counters than either partial product multipliers or the previous fast convolution multiplers. However, as stated earlier, we will assume that the smaller submultiplications are simply implemented as partial product multipliers.

$$
\begin{aligned}
a_0 &= x_0 & b_0 &= y_0 \\
a_1 &= x_1 & b_1 &= y_1 \\
a_2 &= x_2 & b_2 &= y_2 \\
a_3 &= x_0 + x_1 & b_3 &= y_0 + y_1 \\
a_4 &= x_0 + x_2 & b_4 &= y_0 + y_2 \\
a_5 &= x_1 + x_2 & b_5 &= y_1 + y_2 \\
m_i &= a_i b_i, \quad i = 0, 1, \ldots, 5 \\
z_0 &= m_0 \\
z_1 &= m_3 - m_0 - m_1 \\
z_2 &= m_4 - m_0 - m_2 + m_1 \\
z_3 &= m_5 - m_1 - m_2 \\
z_4 &= m_2
\end{aligned}
$$

Fig. 2. Fast Aperiodic Convolution Algorithm (n = 3)

Table 4
Operation Breakdown for n = 3

Operation	Number
addition of two $b/3$ bit integers producing a $b/3+1$ bit sum	3
multiplication of two $b/3$ bit integers producing a $2b/3$ bit product	3
multiplication of two $b/3+1$ bit integers producing a $2b/3+2$ bit product	3
addition of three $2b/3+2$ bit integers 2 producing a $2b/3+1$ bit sum	2
addition of four $2b/3+2$ bit integers producing a $2b/3+2$ bit sum	1
addition of five $2b/3+2$ bit integers producing a $2b$ bit sum	1

Table 5
Fast n = 3 Algorithm Costs

bits	(3, 2) counters	(2, 2) counters	cost	over-head
6	55	31	1097	0.773
12	163	49	2951	0.562
18	319	67	5573	0.443
24	523	85	8963	0.366
30	775	103	13121	0.312
36	1075	121	18047	0.271
60	2755	193	45431	0.179
126	11335	391	184097	0.093

Note that as with the previously presented fast convolution algorithm, we elect to first generate the z_i's and then in turn generate the final product z. An examination of the algorithm then reveals that the following operations are required.

Again the program we wrote was used to compute the number of counters needed for multipliers whose implementation is based on this fast convolution algorithm. In Table 5 we present data for multipliers of some select sizes.

3 Comparison

Figure 3 concisely captures the relative (to partial product multipliers) costs of the different multiplier formulation.

As can be seen, partial product multipler multiplers are the most cost efficient for less than 18 bits.

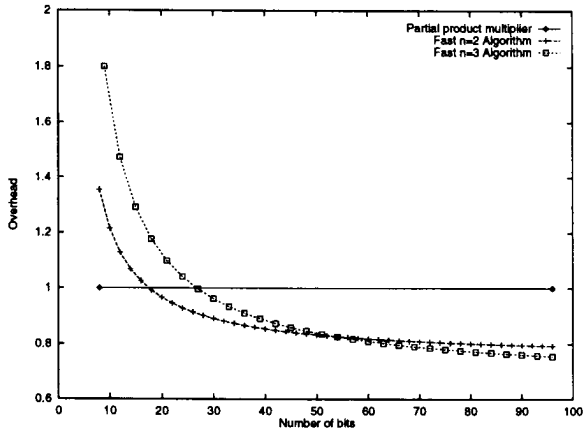

Fig. 3. Multiplier cost comparison

The fast convolution $n = 2$ multiplers are the most cost efficient for between 18 and 54 bits. Lastly, the fast convolution $n = 3$ multiplers are the most cost efficient for more than 54 bits. Since, floating point word (mantissa) sizes from 24 to 112 are commonly used, both of the fast convolution multiplier have their "place".

4 Conclusions

We have shown in this paper that it is possible to multiply reasonably small integers using fewer counters than that required by straightforward partial product accumulation. However, we have not considered a number of other issues. One of these issues is routing area. The cost function we have chosen is thought to be a good estimation of gate area but completely ignores chip area devoted solely to interconnect. Another of these issues is speed. Both of these issues are far more complex and interrelated than they appear at first. For example as is the case with partial product based multipliers, speed can be increased by using more than the minimal number of counters. Also there is the same tradeoff between regularity of layout and speed. We are continuing to investigate these issues.

References

[1] L. Dadda. On Parallel Digital Multipliers. *Alta Freq*, pages 574–580, 1976.

[2] H. J. Nussbaumer. *Fast Fourier Transform and Convolution Algorithms*. Springer-Verlag, 1982.

[3] W. J. Stenzel. A Class Of Compact High-Speed Parallel Multipliers Schemes. Technical Report TR UIUCDCS-R-75-756, Department of Computer Science, University of Illinois at Urbana, Sept. 1975.

[4] C. G. Wallace. A suggestion for a fast multiplier. *IEEE Trans. on Computers*, pages 14–17, Feb. 1964.