IOSR Journal of VLSI and Signal Processing (IOSR-JVSP)
Volume 4, Issue 2, Ver. I (Mar-Apr. 2014), PP 16-22
e-ISSN: 2319 – 4200, p-ISSN No. : 2319 – 4197
www.iosrjournals.org

Designing of signed multiplier without 2's compliment method

Shaik.Moulali[1],Dileep.G[2],kadhar basha[3]
Asst.prof dept of ECE[(1),] Moulali.moulali@gmail.com[(1)]

***Abstract**: Two's complement multipliers are important for a wide range of applications. In this paper, we present a technique to reduce by one row the maximum height of the partial product array generated by a radix-4 Modified Booth Encoded multiplier, without any increase in the delay of the partial product generation stage.*
This reduction may allow for a faster compression of the partial product array and regular layouts. This technique is of particular interest in all multiplier designs, but especially in short bit-width two's complement multipliers for high-performance embedded cores.
The proposed method is general and can be extended to higher radix encodings, as well as to any size square and m_ n rectangular multipliers. We evaluated the proposed approach by comparison with some other possible solutions; the results based on a rough theoretical analysis and on logic synthesis showed its efficiency in terms of both area and delay.

I. Introduction

The MAC(Multiplier and Accumulator Unit) is used for image processing and digital signal processing (DSP) in a DSP processor. Algorithm of MAC is Booth's radix-4 algorithm, wallace tree, 4:2 CSA, 64bit carry select adder and improves speed. MIPS was implemented as micro processors and permitted high performance pipeline implementations through the use of their simple register oriented instruction sets. Although those algorithms (radix-4 algorithm, pipelining, etc) are widely used technique for speeding up each part, the MAC on specific processor cannot be run at 100% efficiency. Due to the reasons of lower speed of MAC, MIPS instruction "mul" (multiplication) takes longer time than any other instruction in our MIPS processor. To improve speed of MIPS, MAC needs to be fast and MIPS must have special algorithm for "mul" instruction. One of the method we chose was to design multi-clock MAC instead of one-clock MAC which improved the speed of MIPS. In general, the instruction set of MIPS processor includes complex works like multiplication and floating point operation which has multi execution stage. Therefore, system clock of the processor was increased efficiently.We applied 2 stage pipelining to the MAC to MIPS processor and as a result we were able to get the result of matrix multiplication which was used .

MAC MULTIPLICATION PROCESS

The simplest multiplication operation is to directly calculate the product of two numbers by hand. This procedure can be divided into three steps: partial product generation, partial product reduction and the final addition.

To further specify the operation process, let us calculate the product of two two's complement numbers, for example, 1101_{two} $(-3)_{ten}$ and 0101_{two} $(5)_{ten}$, when computing the product by hand, which can be described according to figure 2.1.

```
                    1 1 0 1 Multiplicand
                  × 0 1 0 1 Multiplier
          ------------------------------------------------
           1 1 1 1 1 1 0 1      PP1
           0 0 0 0 0 0 0 0      PP2
           1 1 1 1 0 1          PP3
         + 0 0 0 0 0            PP4
          ------------------------------------------------
        1 1 1 1 1 1 0 0 0 1 = −15 Product
discard this bit
```

Figure 1 :Multiplication calculation by hand

The bold italic digits are the sign extension bits of the partial products. The first operand is called the multiplicand and the second the multiplier. The intermediate products are called partial products and the final result is called the product. However, the multiplication process, when this method is directly mapped to hardware, is shown in figure 2.2.

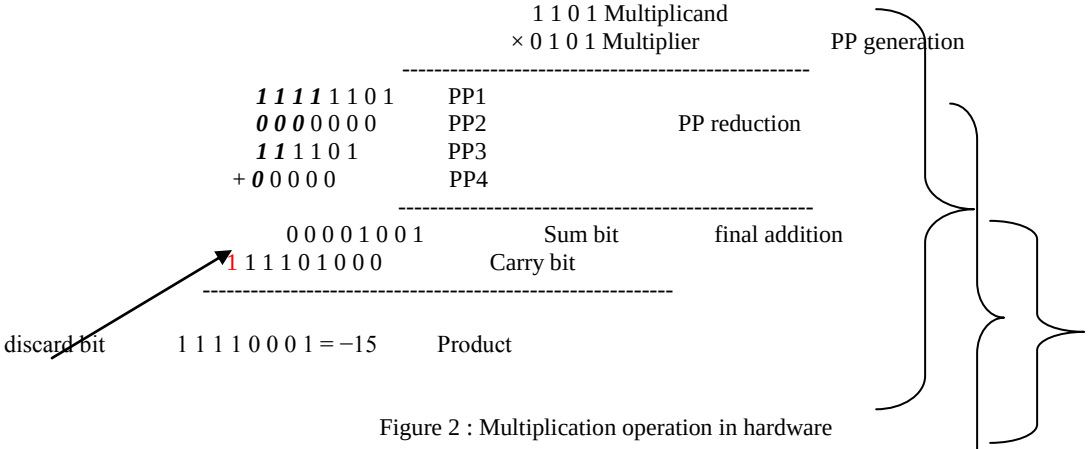


Figure 2 : Multiplication operation in hardware

As can been seen in the figures, the multiplication operation in hardware consists of PP generation, PP reduction and final addition steps. The two rows before the product are called sum and carry bits. The operation of this method is to take one of the multiplier bits at a time from right to left, multiplying the multiplicand by the single bit of the multiplier and shifting the intermediate product one position to the left of the earlier intermediate products. All the bits of the partial products in each column are added to obtain two bits: sum and carry. Finally, the sum and carry bits in each column have to be summed.
Similarly, for the multiplication of an *n*-bit multiplicand and an *m*-bit multiplier, a product with *n* + *m* bits long and *m* partial products can be generated.
The method shown in figure 2.2 is also called a non-Booth encoding scheme. Its advantages and drawbacks will be discussed in next section.

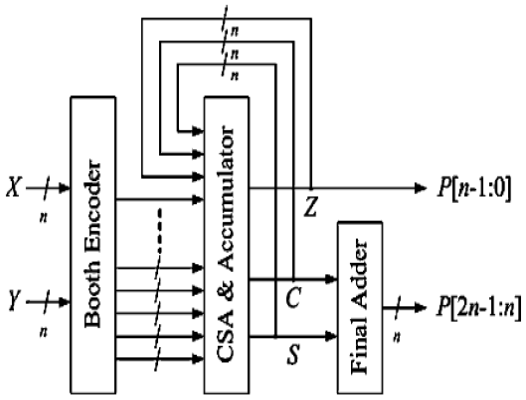
Non-Booth encoding

Using the non-Booth encoding method for partial product generation, the multiplier bits are examined sequentially starting from LSB to MSB. If the multiplier bit is one, the partial product is simply the multiplicand. Otherwise, the partial product is zero. Each new partial product is shifted one bit position to the left. Each partial product can be produced by just using a row of two-input AND gates. The number of partial products generated equals the size of the multiplier bits.
The advantage of this method is that the partial product circuit is simple and easy to implement. Therefore, this scheme is suitable for the implementation of small multipliers.
The drawback is that the method is not able to efficiently handle the sign extension and it generates a number of partial products as many as the number of bits of the multiplier, which results in many adders needed so that the area and power consumption increase. This method is not applicable for large multipliers.

Booth Encoding

The Booth encoding, or Booth algorithm, was proposed by Andrew D. Booth in 1951 [1]. This method can be used to multiply two two's complement number without the sign bit extension.
The operation of Booth encoding consists of two major steps [2]: the first one is to take one bit of the multiplier, and then to decide whether to add the multiplicand according to the current and previous bits of the multiplier. This encoding scheme is serial, which means that the different value of the 2 bits (current and previous bits) corresponds to the different operations. The serial encoding scheme is usually applied in serial multipliers. The operation procedure can be described with the following table.
00: no arithmetic operation.
01: adding the multiplicand to the left half of the product.
10: subtracting the multiplicand from the left half of the product.
11: no arithmetic operation.
The second step is to shift the product right one bit.

Modified Booth Encoding



The modified Booth encoding (MBE), or modified Booth's algorithm (MBA), was proposed by O. L. Macsorley in 1961. The encoding method is widely used to generate the partial products for implementation of large parallel multipliers, which adopts the parallel encoding scheme. The basic principle for the modified Booth encoding can be described as follows.
The bits of the multiplier are partitioned into sub-strings by the 3 adjacent bits and each sub-string group () corresponds to one of the value in the set {−2, −1, 0, +1, +2}[30]. This means that the each three adjacent bits of the multiplier can generate a single encoding digit, which is called the modified Booth recoding digit (*di)*, as shown in table 2.1. Each MBE blocks can work in parallel, therefore, all the partial product bits are generated simultaneously. The parallel encoding scheme is suitable for parallel multipliers.

Block	Re - coded digit	Operation on X
000	0	0 X
001	+1	+1 X
010	+1	+1 X
011	+2	+2 X
100	-2	-2 X
101	-1	-1 X
110	-1	-1 X
111	0	0 X

MODIFIED BOOTH RECODED MULTIPLIERS

In general, a radix-B ¼ 2b MBE leads to a reduction of the number of rows to about dn=be while, on the other hand, it introduces the need to generate all the multiples of the multiplicand X, at least from _B=2 _ X to B=2 _ X. As mentioned above, radix-4 MBE is particularly of interest since, for radix-4, it is easy to create the multiples of the multiplicand 0;_X;_2X. In particular, _2X can be simply obtained by single left shifting of the corresponding terms _X. It is clear that the MBE can be extended to higher radices (see [12] among others), but the advantage of getting a higher reduction in the number of rows is paid for by the need to enerate more multiples of X. In this paper, we focus our attention on radix-4 MBE, although the proposed method can be easily extended to any radix-B MBE.

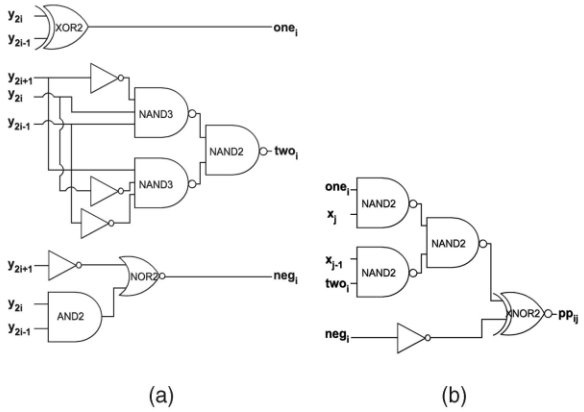
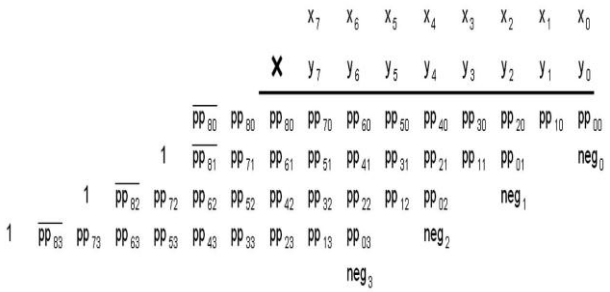


Fig. 3. Gate-level diagram for partial product generation using MBE

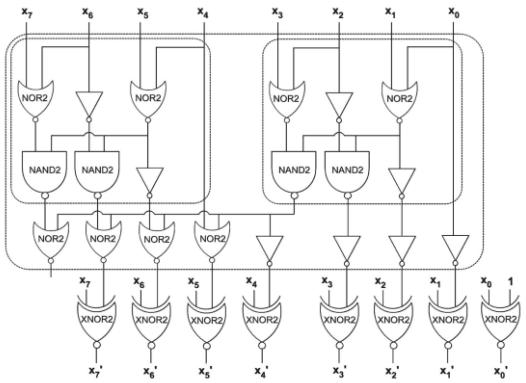
(a) MBE signals generation. (b) Partial productgeneration.

From an operational point of view, it is well known that the radix-4 MBE scheme consists of scanning the multiplier operand with a three-bit window and a stride of two bits (radix-4). For each group of three bits (y2i+1, y2i, y2i-1), only one partial product row is generated according to the encoding in Table 1. A possible implementation of the radix-4 MBE and of the corresponding partial product generation is shown in Fig. 1, which comes from a small adaptation. For each partial product row, Fig. 1a produces the one, two, and neg signals. These signals are then exploited by the logic in Fig. 1b, along with the appropriate bits of the multiplicand, in order to generate the whole partial product array. Other alternatives for the implementation of the recoding and partial product generation can be found among others.

As introduced previously, the use of radix-4 MBE allows for the (theoretical) reduction of the PP rows to [n /2], with the possibility for each row to host a multiple of yi x X, with yi Є2 {0;+1;+2}. While it is straightforward to generate the positive terms 0, X, and 2X at least through a left shift of X, some attention is required to generate the terms -X and -2X which, as observed in Table 1, can arise from three configurations of the y2i+1, y2i, and y2i-1 bits. To avoid computing negative encodings, i.e., -X and -2X, the two's complement of the multiplicand is generally used. From a mathematical point of view, the use of two's complement requires extension of the sign to the leftmost part of each partial product row, with the consequence of an extra area overhead. Thus, a number of strategies for preventing sign extension have been developed. For instance, the scheme in [1] relies on the observation that -pp = pp + 1 ¼=pp -1 -2 + 4. The array resulting from the application of the sign extension prevention technique to the partial product array of a 8 x8 MBE multiplier is shown in Fig. 2. The use of two's complement requires a neg signal (e.g., neg0, neg1, neg2, and neg3 in Fig. 2) to be added in the LSB position of each partial product row for generating the two's complement, as needed. Thus, although for a n xn multiplier, only [n/2] partial products are generated, the maximum height of the partial product array is [n/ 2]+ 1.



Application of the sign extension prevention measure on the partial product array of a 8 x 8 radix-4 MBE multiplier.



Two's complement computation (n =8)

The case of nxn square multipliers is quite common, as the case of n that is a power of two. Thus, we start by focusing our attention on square multipliers, and then present the extension to the general case of m x n rectangular multipliers.

The proposed approach is general and, for the sake of clarity, will be explained through the practical case of 8 x 8 multiplication (as in the previous figures). As briefly outlined in the previous sections, the main goal of our approach is to produce a partial product array with a maximum height of [n/2] rows, without introducing any additional delay. Let us consider, as the starting point, the form of the simplified array as reported in Fig. 2, for all the partial product rows except the first one. As depicted in Fig. 6a, the first row is temporarily considered as being split into two subrows, the first one containing the partial product bits from right to left) from pp00 to pp80 and the second one with two bits set at "one" in positions 9 and 8. Then, the bit neg3 related to the fourth partial product row, is moved to
become a part of the second subrow.

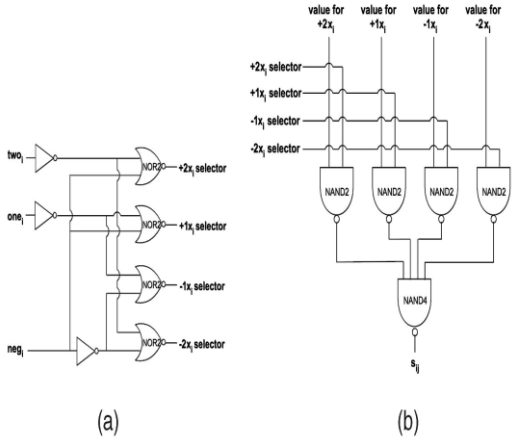
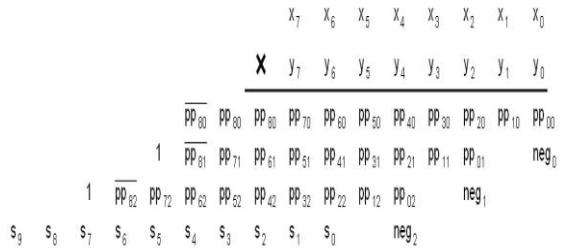


Fig. 4.Gate-level diagram for the generation of two's complement partial product rows (a) 3-5 decoder. (b) 4-1 multiplexer



.Fig. 5. Partial product array by applying the two's complement computation method in [9] to the last row.

The key point of this "graphical" transformation is that the second subrow containing also the bit neg3, can now be easily added to the first subrow, with a constant short carry propagation of three positions (further denoted as "3-bits addition"), a value which is easily shown to be general, i.e., independent of the length of the operands, for square multipliers. In fact, with reference to the notation of Fig. 6, we have that qq90 qq90 qq80 qq70 qq60 = 0 0 pp80 pp70 pp60 + 0 1 1 0 neg3. As introduced above, due to the particular value of the second operand, i.e., 0 1 1 0 neg3, in [11], we have observed that it requires a carry propagation only across the least-significant three positions, a fact that can also be seen by the implementation shown in Fig. 7.

It is worth observing that, in order not to have delay penalizations, it is necessary that the generation of the other rows is done in parallel with the generation of the first row cascaded by the computation of the bits qq90 qq90 qq80 qq70 qq60 in Fig. 6b. In order to achieve this, we must simplify and differentiate the generation of the first row with respect to the other rows. We observe that the Booth recoding for the first row is computed more easily than for the other rows, because the y_1 bit used by the MBE is always equal to zero. In order to have a preliminary .

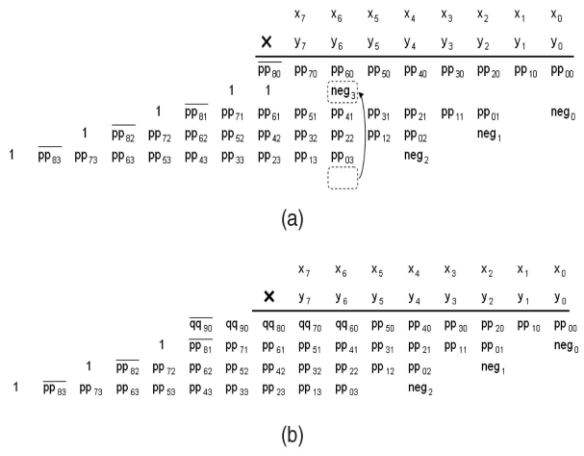


Fig. 6. Partial product array after adding the last neg bit to the first row.
(a) Basic idea. (b) Resulting array.

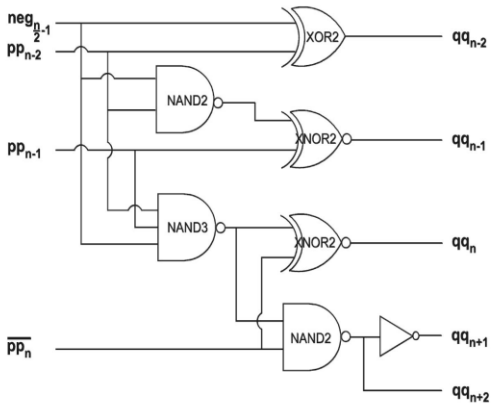


Fig. 7. Gate-level diagram of the proposed method for adding the last neg bit in the first row.

In order to have a preliminary analysis which is possibly independent of technological details, we refer to the circuits in the following figures:

- Fig. 1, slightly adapted for the partial product generation using MBE;
- Fig. 7, obtained through manual synthesis (aimed at modularity and area reduction without compromising the delay), for the addition of the last neg bit to the three most significant bits of the first row;
- Fig. 8, obtained by simplifying Fig. 1 (since, in the first row, it is y2i-1 = 0), for the partial product generation of the first row only using MBE; and
- Fig. 9, obtained through manual synthesis of a combination of the two parts of Fig. 8 and aimed at decreasing the delay of Fig. 8 with no or very small area increase, for the partial product generation of the first row only using MBE.

In particular, we observe that, by direct comparison of Figs. 1 and 8, the generation of the MBE signals for the first row is simpler, and theoretically allows for the saving of the delay of oneNAND3gate. In addition, the implementation in Fig. 9 has a delay that is smaller than the two parts of Fig. 8, although it could require a small amount of additional area. As we see in the following, this issue hardly has any significant impact on the overall design, since this extra hardware is used only for the three most significant bits of the first row, and not for all the other bits of the array.

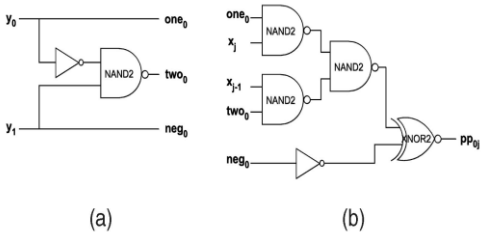


Fig. 8. Gate-level diagram for first row partial product generation.
(a) MBE signals generation. (b) Partial product generation.

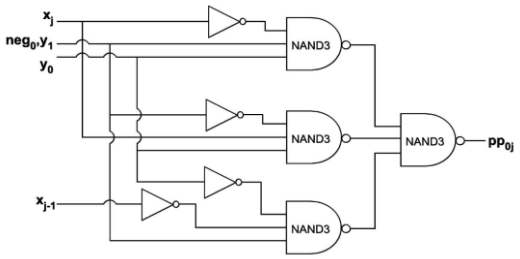


Fig. 9. Combined MBE signals and partial product generation for the first row (improved for speed).

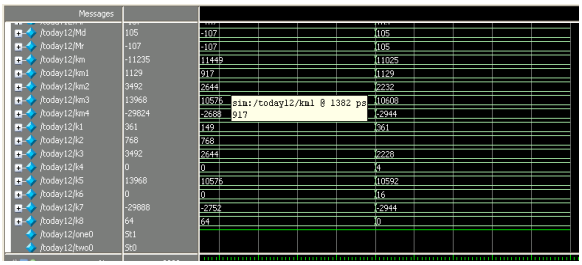
The high-level description of our idea is as follows:

1. generation of the three most significant bit weights of the first row, plus addition of the last neg bit:possible implementations can use a replication of three times the circuit of Fig. 9 (each for the three most significant bits of the first row), cascaded by the circuit of Fig. 7 to add the neg signal;
2. parallel generation of the other bits of the first row: possible implementations can use instances of the circuitry depicted in Fig. 8, for each bit of the first row, except for the three most significant;

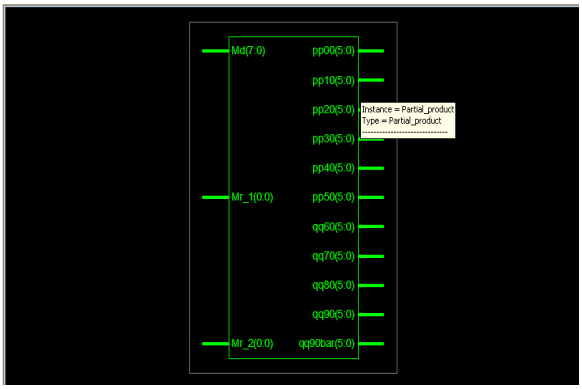
3. parallel generation of the bits of the other rows: possible implementations can use the circuitry of Fig. 1, replicated for each bit of the other rows. All items 1 to 3 are independent, and therefore can be executed in parallel. Clearly if, as assumed and expected, item 1 is not the bottleneck (i.e., the critical path), then the implementation of the proposed idea has reached the goal of not introducing time penalties.

II. Results:

simulation



Synathasis:



III. Conclusions And Future Work

Two's complement n x n multipliers using radix-4 Modified Booth Encoding produce [n/2] partial products but due to the sign handling, the partial product array has a maximum height of [n /2]+ 1. We presented a scheme that produces a partial product array with a maximum height of [n/2], without introducing any extra delay in the partial product generation stage. With the extra hardware of a (short) 3-bit addition, and the simpler generation of the first partial product row, we have been able to achieve a delay for the proposed scheme within the bound of the delay of a standard partial product row generation.

Reference:

[1]. M.D. Ercegovac and T. Lang, Digital Arithmetic. Morgan Kaufmann Publishers, 2003.
[2]. S.K. Hsu, S.K. Mathew, M.A. Anders, B.R. Zeydel, V.G. Oklobdzija, R.K. Krishnamurthy, and S.Y. Borkar, "A 110GOPS/ W 16-Bit Multiplier and Reconfigurable PLA Loop in 90-nm CMOS," IEEE J. Solid State Circuits, vol. 41, no. 1, pp. 256-264, Jan. 2006.
[3]. H. Kaul, M.A. Anders, S.K. Mathew, S.K. Hsu, A. Agarwal, R.K. Krishnamurthy, and S. Borkar, "A 300 mV 494GOPS/W Reconfigurable Dual-Supply 4-Way SIMD Vector Processing Accelerator in 45 nm CMOS," IEEE J. Solid State Circuits, vol. 45, no. 1, pp. 95- 101, Jan. 2010.
[4]. LAMBERTI ET AL.: REDUCING THE COMPUTATION TIME IN (SHORT BIT-WIDTH) TWO'S COMPLEMENT MULTIPLIERS 155
[5]. Fig. 10. Postsynthesis results of product generation for different sizes. (a) Size of 8. (b) Size of 16. (c) Size of 32.
[6]. M.S. Schmookler, M. Putrino, A. Mather, J. Tyler, H.V. Nguyen, C. Roth, M. Sharma, M.N. Pham, and J. Lent, "A Low-Power, High-Speed Implementation of a PowerPC Microprocessor Vector Extension," Proc. 14th IEEE Symp. Computer Arithmetic, pp. 12-19, 1999.