Low Power Multiplication Algorithm for Switching Activity Reduction through Operand Decomposition*

Masayuki Ito
Renesas Technology Corp.
Tokyo 187-8588, Japan
ito.masayuki2@renesas.com

David Chinnery, Kurt Keutzer
Dept. of EECS, UC Berkeley
Berkeley, CA 94720, USA
{chinnery, keutzer}@eecs.berkeley.edu

Abstract

A novel low power multiplication algorithm for reducing switching activity through operand decomposition is proposed. Our experimental results show 12% to 18% reduction in logic transitions in both array multipliers and tree multipliers of 32 bits and 64 bits. Similar results are obtained for dynamic power dissipation after logic synthesis. One additional logic gate is required on the critical path for operand decomposition, which corresponds to only an additional 2% to 6% of total delay in these four cases. Thus, the proposed algorithm can be applied to many digital systems where power consumption is a major design constraint.

1 Introduction

Nowadays, many digital signal processing systems are targeted at portable, battery-operated systems, so power dissipation is one of the primary design constraints. Since multipliers are rather complex circuits and typically must operate at a high system clock rate, reducing logic transitions and thus the power dissipation of multipliers is a key to satisfying the overall power budget.

Digital CMOS circuits consume relatively less power and static CMOS logic is widely used because of its robustness, and its suitability for design automation [1]. In static CMOS, the dynamic power dissipation of a circuit depends primarily on the number of logic transitions per unit time [2].

In this paper, we propose a novel low power multiplication algorithm using operand decomposition. Unlike Booth's encoding schemes [3] that modify only one operand to reduce the number of partial products, our scheme decomposes both a multiplicand and a multiplier. A final product is generated through two multiplications. The decomposition is done to reduce the number of ones in total partial products. As a result, logic transitions are reduced, achieving lower power dissipation.

This paper is organized as follows: Section 2 briefly reviews well-known array and tree multipliers. Some existing approaches for low-power multipliers are also discussed. In Section 3, we propose our new multiplication algorithm. Then, in Section 4, we show our results for delay, area, and dynamic power for both conventional multipliers and the proposed multipliers. Conclusions are presented in Section 5.

2 Multiplier Architectures

In this section, we first review regular topologies for multipliers. They can be classified as either array topologies or tree topologies [4,5]. Then, we present some existing architectural and algorithmic approaches to reduce the consumed power.

2.1 Array Multipliers

A simple array multiplier consists of rows of (3,2) counters. In the simple array, each row of (3,2) counters adds a partial product (PP) to the partial sum, generating a new partial sum and a sequence of carries. Figure 1 shows a block diagram for a single array multiplier. The delay of the array to produce the final partial sum and carry is ($2n$-4) XOR delays for an n-bit multiplier.

The delay required to generate the result for the simple array can be halved by performing two additions in parallel. The final two partial sums and two carries are combined using a (4:2) compressor. Figure 2 shows a block diagram for a double array multiplier. The delay of the double array to produce the final partial sum and carry is (n-1) XOR delays.

* This work was performed while Masayuki Ito visited University of California at Berkeley as a visiting industrial fellow.

Proceedings of the 21st International Conference on Computer Design (ICCD'03)
1063-6404/03 $ 17.00 © 2003 IEEE

2.2 Tree Multipliers

Trees are very fast structures for summing PPs. The Wallace tree topology based on (3,2) counters [6] has an irregular structure which causes many spurious transitions due to imbalances of arrival time for each counter. A (4:2) compressor based multiplier forms a binary tree topology which leads to a fast, symmetric and regular design. Thanks to this regularity, spurious transitions can also be reduced [7]. Figure 3 shows a block diagram for a (4:2) compressor based multiplier. The delay of this multiplier to produce the final partial sum and carry is $3\log_2(n/2)$ XOR delays for an *n*-bit multiplier.

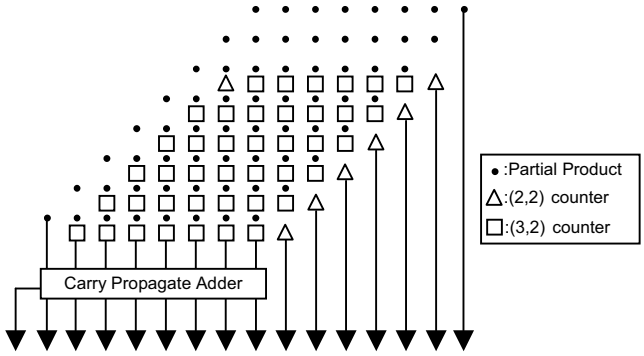

Figure 1. An 8-bit single array multiplier. The partial product generation logic that precedes this is omitted for clarity.

2.3 Previous work

Reducing the power consumption for multiplication can be tackled at different levels of the design hierarchy. Algorithmic level approaches can give a significant overall power reduction, but only a few approaches have been reported. Some existing solutions at this level for low-power multipliers try to reduce the switching activity by observing operands.

A dynamic operand exchange method observes the sign bits of the operands of two consecutive multiplications and conditionally exchanges the operands [8]. A dynamic operand complementation scheme utilizes the identity $(-1)^2 = 1$ in addition to the dynamic operand exchange [9]. For these two schemes, the required logic for judgment and selection for operand exchange is not trivial and the worst case delay is increased.

If the multiplication is divided into clusters of smaller multipliers with registers, clock gating techniques can be applied to clusters that produce a result of zero [10]. This approach is limited to small multipliers or applications that naturally contain a high probability of inputs with several consecutive zeros, where clusters can be turned off frequently.

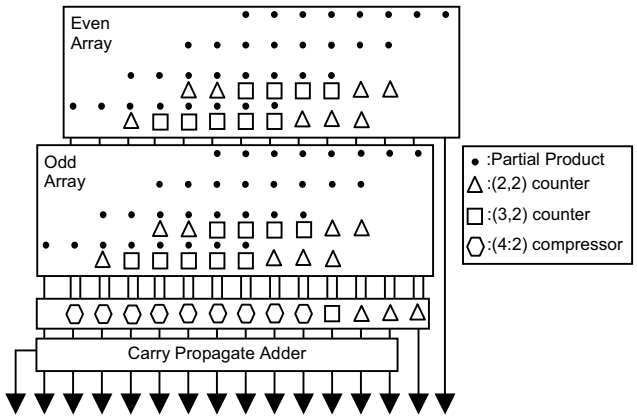

Figure 2. An 8-bit double array multiplier

3 Low Power Multiplication Algorithm

In this Section, we propose a novel low power Multiplication Algorithm for Switching Activity Reduction through operand decomposition (MASAR for short). For simplicity throughout this paper, we try to compute unsigned *n*-bit integer multiplications; Operands *X* and *Y* have *n* bits and a product $M = X \times Y$ has 2*n* bits. Let

$X = [x_{n-1}x_{n-2}\cdots x_1x_0]$, and

$Y = [y_{n-1}y_{n-2}\cdots y_1y_0]$.

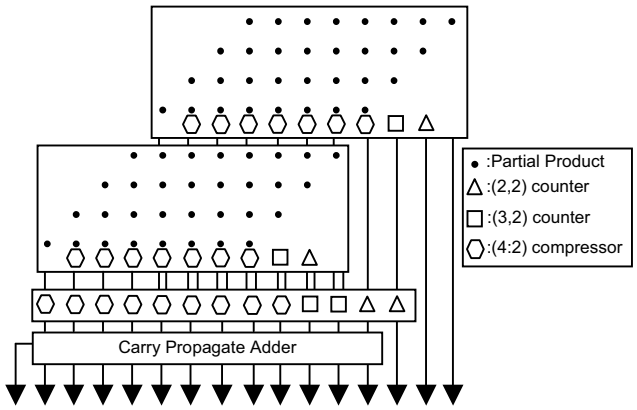

Figure 3. An 8-bit (4:2) compressor based multiplier

We decompose operand X and Y into 4 numbers A, B, C, and D to reduce the number of ones in partial products. For this purpose we generate four numbers below in Step 1 of MASAR:

$A = [-1 a_{n-1} a_{n-2} \cdots a_i \cdots a_1 (a_0 + 1)]$,

$B = [b_{n-1} b_{n-2} \cdots b_i \cdots b_1 b_0]$,

$C = [c_{n-1} c_{n-2} \cdots c_i \cdots c_1 c_0]$, and

$D = [d_{n-1} d_{n-2} \cdots\cdots d_i \cdots d_1 d_0]$, where

$a_i = \overline{x_i} \wedge \overline{y_i}$, $b_i = x_i \wedge y_i$, $c_i = \overline{x_i} \wedge y_i$, and $d_i = x_i \wedge \overline{y_i}$ for $(n-1 \geq i \geq 0)$. With the property that

$X \times Y = (C \times D) - (A \times B)$.

To help understand this equation, we introduce a temporal number $T = -A$. Then,

$T = [t_{n-1} t_{n-2} \cdots t_1 t_0]$, $t_i = x_i \vee y_i$.

It is easy to see that $X = B + D$, $Y = B + C$, and $T = B + C + D$. Then

$$\begin{aligned} X \times Y &= (B + D) \times (B + C) \\ &= (C \times D) + (B + C + D) \times B \\ &= (C \times D) + (T \times B) \\ &= (C \times D) - (A \times B) \end{aligned}$$

There is another important relation between these four numbers. Since a_i, b_i, c_i and d_i are generated by ANDing x_i and y_i or their inversions, $a_i + b_i + c_i + d_i$ is always 1 for each i. When we denote the probability of $a_i = 1$ by $\Pr(a_i)$,

$\Pr(a_i) = \Pr(b_i) = \Pr(c_i) = \Pr(d_i) = 1/4$, while

$\Pr(x_i) = \Pr(y_i) = 1/2$.

In Step 2 of MASAR, we generate and compress PPs for each multiplication, $C \times D$ and $A \times B$. About twice as many PPs as in conventional multipliers are generated. Let

$P = A \times B = [p_{n-1,n-2} \cdots p_{i,j} \cdots p_{1,0}]$, and

$Q = C \times D = [q_{n-1,n-2} \cdots q_{i,j} \cdots q_{1,0}]$.

We generate $p_{i,j}$ and $q_{i,j}$ by

$p_{i,j} = (a_i \wedge b_j) \vee (b_i \wedge a_j)$, and

$q_{i,j} = (c_i \wedge d_j) \vee (d_i \wedge c_j)$

for $(n-1 \geq i > j \geq 0)$.

This PP reduction can be done by simple OR logic because only one of a_i and b_i are 1, and only one of c_i and d_i is 1. Note also that $p_{i,i} = q_{i,i} = 0$ for $(n-1 \geq i \geq 0)$ by the same reason. Now, the total number of PPs with MASAR is almost the same as conventional multipliers. Here, $\Pr(p_{i,j}) = \Pr(q_{i,j}) = 1/8$ while the probability of one for each partial product of conventional multipliers is $1/4$. Thus the number of logic transitions is reduced.

We need to account for the complementary terms generated by the most significant and least significant bits of A. The n-th position of A is always -1 and additional n PPs are generated. The 0-th position of A is $(a_0 + 1)$ instead of a_0, which requires a simple modification to the logic to generate the corresponding (n+1) PPs. Tables 1 and 2 give the total number of PPs and the average number of ones in PPs with and without MASAR. The numbers for MASAR are after PP reduction in Step 2. Figure 4 shows an example of our operand decomposition and Figure 5 shows the MASAR PP reduction scheme.

After PPs are generated and reduced, PPs for each P and Q are summed up by either an array or tree topology. Then the final partial sum and carry from both P and Q can be combined using (4:2) compressors. Since $-P + Q$ is required instead of $P + Q$, the (4:2) compressors should be modified so that two input signals from P are negated. +2 must also be added as we negate two numbers. This complement can be handled easily by the final carry propagate adder. Figures 6 and 7 show block diagrams for an array multiplier with MASAR and a tree multiplier with MASAR. Positive PPs and negative PPs shown in these diagrams correspond to the values $(C \times D + B \cdot 2^n)$ and $(-A \times B - B \cdot 2^n)$, respectively. Figure 8 shows circuits for the (2,2) counter, the (3,2) counter, and the (4:2) compressor used in our evaluations.

Table 1. Total number of PPs

Type/Width	8	16	32	64
Conventional	64	256	1024	4096
MASAR	66	258	1026	4098

Table 2. Average number of ones in PPs

Type/Width	8	16	32	64
Conventional	16	64	256	1024
MASAR	12	40	144	544

$X = \{00110111\} = \text{d'}55$

$Y = \{01000101\} = \text{d'}69$

$A = \{\text{-}110001001\} = \text{d'-}119$

$B = \{00000101\} = \text{d'}5$

$C = \{01000000\} = \text{d'}64$

$D = \{00110010\} = \text{d'}50$

$55 \times 69 = 64 \times 50 - (-119 \times 5)$

Figure 4. An example of operand decomposition

Proceedings of the 21st International Conference on Computer Design (ICCD'03)
1063-6404/03 $ 17.00 © 2003 IEEE

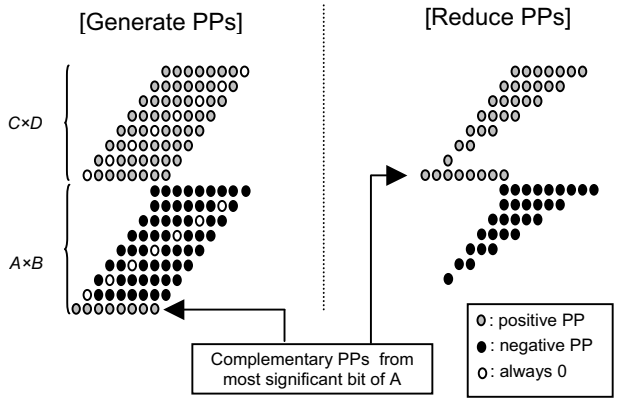

Figure 5. PP reduction by MASAR

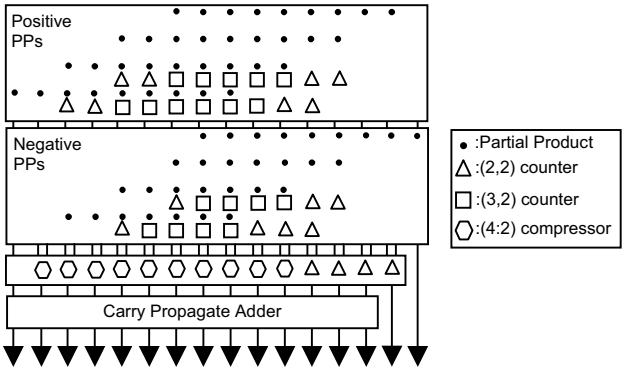

Figure 6. An 8-bit array multiplier with MASAR

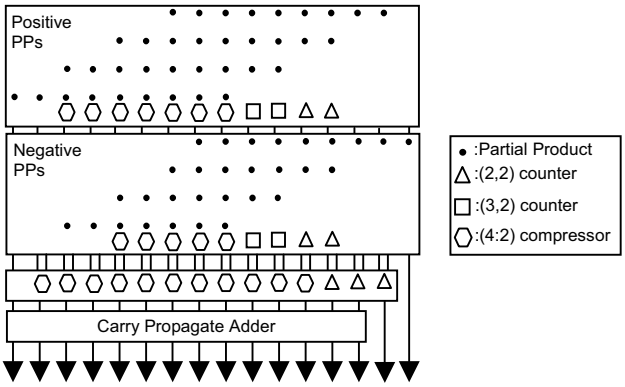

Figure 7. An 8-bit tree multiplier with MASAR

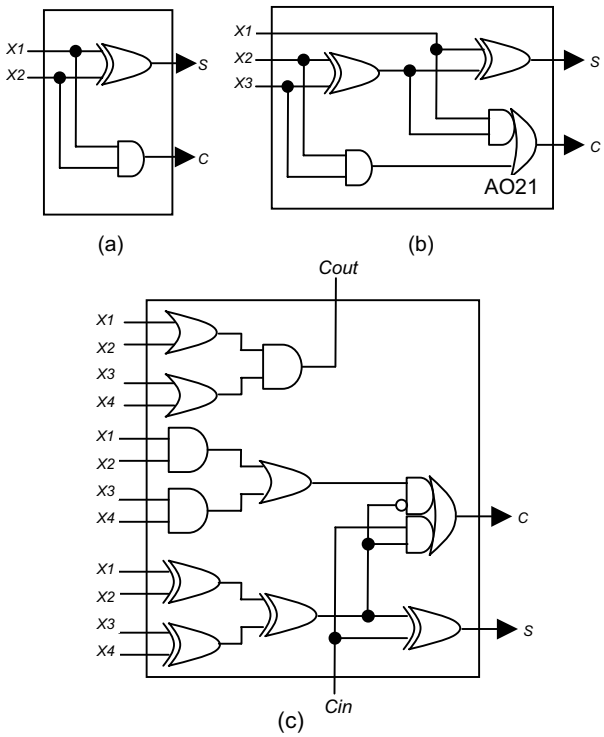

Figure 8. Circuits for (a) a (2,2) counter, (b) a (3,2) counter, (c) a (4:2) compressor

4 Evaluations

In this section, we compare the proposed multiplier to conventional multipliers in terms of dynamic power, area, and delay. We have applied MASAR to both array multipliers and tree multipliers. Since MASAR generates two arrays, a double array multiplier (D-array) is compared with an array multiplier with MASAR (M-array). For tree multipliers, a (4:2) compressor based multiplier without MASAR (C-tree) and with MASAR (M-tree) are compared. Note that MASAR only affects the PP generation and reduction stages and the remainder of the multiplier logic in each comparison are implemented in the same manner. Kogge-Stone adders [11] are adopted as the final carry propagate adder in the multipliers for our evaluations.

4.1 Logic transitions and dynamic power

In order to evaluate dynamic power savings with MASAR, we first specified the multipliers in C, with the logic shown in the figures. The partial products were generated using AND gates, or AO22 gates for MASAR. (AO22 is a complex logic gate with four inputs, for

computing $p_{i,j}$ and $q_{i,j}$.) The remaining logic was implemented using counters and compressors shown in Figure 8. The final Kogge-Stone adder was specified using XOR, AND, and AO21 logic. Then the average number of gates that change output value per multiplication was counted. Each multiplier was subjected to 1000 pseudo-random input patterns. We initialized these multipliers to 0 only before the first pseudo-random data. Table 3 shows the resulting total number of logic transitions. This shows that MASAR saves 12% and 15% in array multipliers of 32-bits and 64-bits, respectively, and 12% and 18% for tree multipliers of 32-bits and 64-bits, respectively.

Then Verilog RTL was written and verified using Synopsys VCS HDL Simulator. We synthesized each multiplier with Synopsys Design Compiler using STMicroelectronics' HCMOS9 0.13um high-speed standard cell library at the worst case process corner. We collected switching activity data, without glitches, for these gate-level netlists with the same random data as used in C simulation. The switching activities were then back-annotated to calculate the dynamic power assuming 100MHz clock frequency, at 85°C, with supply voltage of 1.08V. Table 4 shows the results. MASAR can also reduce the number of spurious logic transitions as it reduces the probability of ones in PPs.

Table 4 shows that MASAR saves 12.8% and 8.9% in array and tree multipliers of 64-bit respectively, and 9.7% and 5.1% comparing the 32-bit multipliers**.** Thus, MASAR is effective for both array and tree multipliers for sizes of 32-bit width and larger. The actual dynamic power savings correspond fairly closely to reduction in the number of logic transitions calculated by simulation in C in Table 3.

4.2 Area

The additional logic required for Step 1 of MASAR is the generation of *A*, *B*, *C*, and *D* and the complementary logic for the most significant bit and least significant bit of *A* in Step 1. Recall that $p_{i,j} = (a_i \wedge b_j) \vee (b_i \wedge a_j)$, and $q_{i,j} = (c_i \wedge d_j) \vee (d_i \wedge c_j)$ in Step 2 of MASAR. Generally $n(n-1)$ OR gates are required for the generation of $p_{i,j}$ and $q_{i,j}$ in addition to generating PPs by $2n(n–1)$ AND gates. Here, we can use AO22 gates for compressing PPs to reduce the number of gates. Thanks to this scheme, the increase in cell area required for implementing the whole multiplier with MASAR can be kept small. It is difficult to reduce the number of gates in conventional multipliers by the same scheme; partial product generation logic and the first level of additions cannot be merged efficiently, because fan-outs of these PPs are at least 2.

Table 5 presents the corresponding cell area for the multipliers. The required cell area for MASAR is about 10% for multipliers of 16 bits or larger width.

4.3 Delay

MASAR only affects the critical path delay with the changes in the logic used to generate the partial products. The logic on the critical path of the remainder of the multiplier is the same. Therefore, we synthesized the partial product logic with a tight delay constraint to compare the differences. The additional delay for MASAR is about 0.2ns, which amounts to as much as 8% for the 8 bit multipliers and as little as 2% and 4% overhead for the 64 bit array and tree multipliers respectively. Library wire load models were used for the synthesized netlist delay analysis.

Table 3. Average number of logic transitions

Type/Width	8	16	32	64
D-array	146	588	2344	9267
M-array	169	572	2069	7864
Ratio (%)	115.8	97.3	88.3	84.9
C-tree	156	671	2722	10770
M-tree	178	663	2400	8850
Ratio (%)	114.1	98.8	88.2	82.2

Table 4. Power dissipation (uW)

Type/Width	8	16	32	64
D-array	27	110	440	1786
M-array	31	108	397	1558
Ratio (%)	114.9	97.6	90.3	87.2
C-tree	29	126	519	2200
M-tree	33	127	493	2003
Ratio (%)	113.8	100.6	94.9	91.1

Table 5. Cell area (um^2)

Type/Width	8	16	32	64
D-array	5315	21380	86108	343890
M-array	6098	23837	95310	376737
Ratio (%)	114.7	111.5	110.7	109.6
C-tree	5212	22056	89310	356790
M-tree	6207	24309	98270	389215
Ratio (%)	119.1	110.2	110.0	109.1

5 Conclusions

We have proposed a novel low power multiplication algorithm for reducing switching activity through operand decomposition. Functional simulation to estimate the number of logic transitions in C corresponds well with the dynamic power savings demonstrated by the synthesized netlists. The power savings with MASAR are 9.7% and 5.1% for the 32-bit array and tree multipliers respectively, and 12.8% and 8.9% respectively for the 64-bit array and tree multipliers.

Partial product generation for MASAR causes an extra 8% of the total delay for the 8 bit multipliers, to as little as 2% and 4% overhead for the 64 bit array and tree multipliers respectively. The area overheads for MASAR are only 10% for the 16, 32 and 64 bit multipliers.

Our algorithm is effective for both array and tree multipliers of 32 bits or larger width. The proposed algorithm can be applied to many digital systems practically, where minimizing power consumption is important. There is a substantial reduction in the number of logic transitions and a corresponding power savings, with only a small delay and area increase. More detailed power analysis, with dynamic timing analysis for glitches, will show similar advantages as there are a reduced number of transitions.

Acknowledgments

The authors would like to thank STMicroelectronics for providing access to the 0.13um standard cell library. We would also like to thank Professor Borivoje Nikolić for his valuable comments.

References

[1] J. M. Rabaey, A. Chandrakasan and B. Nikolic, *Digital Integrated Circuits (Second Edition)*, Prentice Hall, 2003.

[2] A. P. Chandrakasan, S. Sheng and R. W. Brodersen, "Low-Power CMOS Digital Design," *IEEE Journal of Solid-State Circuits*, Vol. 24, 1992, pp.487-493.

[3] A. D. Booth, "A Signed Binary Multiplication Technique," *Quarterly J. Mechanical Applications in Math*., vol. 4, part 2, 1951, pp.236-240.

[4] D. Goldberg, "Computer Arithmetic", in D. A. Patterson and J. L. Hennessey, *Computer Architecture: A Quantitative Approach*, Morgan Kaufmann Publishers, 1990.

[5] M. J. Flynn and S. F. Oberman, *Advanced Computer Arithmetic Design*, Wiley-Interscience, 2001.

[6] C. S. Wallace, "A Suggestion for a Fast Multiplier," *IEEE Transactions on Electronic Computers*, Vol. EC-13, 1964, pp.14-17.

[7] V. G. Moshnyaga and K. Tamaru, "A Comparative Study of Switching Activity Reduction Techniques for Design of Low-Power Multipliers," *Proceedings of the IEEE International Symposium on Circuits and Systems*, 1995, pp.1560-1563.

[8] T. Ahn and K. Choi, "Dynamic operand interchange for low power," *Electronics Letters*, Vol.33, No.25, 1997, pp.2118-2220.

[9] P.-M. Seidel, "Dynamic Operand Modification for Reduced Power Multiplication," *Proceedings of the 36th Asilomar Conference on Signals*, *Systems and Computers*, 2002, pp.52-56.

[10] A. A. Fayed and M. A. Bayoumi, "A Novel Architecture for Low-Power Design of Parallel Multipliers," *Proceedings of the IEEE Computer Society Workshop on VLSI*, 2001, pp.149-154.

[11] P. M. Kogge and H. S. Stone, "A Parallel Algorithm for the Efficient Solution of a General Class of Recurrence Equations," *IEEE Transactions on Computers*, Col. C22, No.8, 1973, pp.786-793.