2006－ＳＬＤＭ－126（９）
2006／10／26

乗算器の回路規模縮小に関する研究

陳俊中　エドウイン*　　江川　隆輔*　　多田十兵衛**　　鈴木　健一*
後藤　源助**　　中村　維男*

*東北大学大学院情報科学研究科　〒980-8579 仙台市青葉区荒巻字青葉 6-6-01
**山形大学工学部　〒992-8510 山形県米沢市城南 4-3-16
E-mail: *{edwin,egawa,suzuki,nakamura} @archi.is.tohoku.ac.jp,**{jubee,gengoto}@yz.yamagata-u.ac.jp

あらまし　近年の半導体加工技術の著しい進歩および回路の大規模化に伴う静的消費電力増加を抑えることを目的とし、乗算回路の小規模化を試みる。本報告では、乗算回路を構成する部分積生成部、部分積削減部、および桁上げ吸収加算部における算術アルゴリズムと回路規模の削減効果の関係を評価する。

キーワード　低消費電力、乗算器

Reducing the Circuit Size of Multipliers

TAN Jiunn Jong Edwin*, Ryusuke EGAWA*, Jubei TADA**, Ken-ichi SUZUKI*,
Gensuke GOTO** and Tadao NAKAMURA*

*Graduate School of Information Sciences, Tohoku University　Aramaki Aza Aoba 6-6-01, Aoba-ku, Sendai-shi, 980-8579 Japan
** Faculty of Engineering, Yamagata University　Jo-nan 4-3-16, Yonezawa, Yamagata, 992-8510 Japan
E-mail: *{edwin,egawa,suzuki,nakamura} @archi.is.tohoku.ac.jp,**{jubee,gengoto}@yz.yamagata-u.ac.jp

Abstract　Aiming at reducing the power consumption of future VLSIs, small and fast arithmetic units are required. Since multipliers play an essential role in recent microprocessors, this study focuses on multipliers and discusses the effects of bit slicing technique on the latency and hardware cost. In particular, partial product generation by Booth method, partial product reduction by array and reduction tree, and final addition by a carry-independent adder are examined.

Key words　Low-power, Multiplier

1. Introduction

In recent years, a large increase in power consumption of VLSIs has been linked to heat-induced problems such as hardware damage and performance degradation of microprocessors [1]. Many research efforts have been put into sophisticated cooling systems and power saving techniques such as micro-fluids and power gating, but these provide only temporary solutions in an age when the demand for ever faster and smaller processors is growing [2], [3]. Moreover, it is predicted that for CMOS technology of less than 70nm, static power consumption will constitute more than half of total power consumption [4]. In order to cut back on total power consumption effectively, it becomes important to shift our focus from dynamic power to static power. This rise in static power consumption can be attributed to an increase in leakage current, which is proportional to the number of transistors in a circuit. Therefore, there is a need to reduce the number of transistors. In view of this, our research investigates the potential of bit slicing and small size circuits [5], [6] in reducing static power while keeping performance. In particular, emphasis is placed on the arithmetic unit as it is the basic building block in processors. Here, we focus on the multiplier because it plays an essential part in processors and occupies a large area. Our goal is to design a fast and low power multiplier. The rest of this paper is organized as follows. In Section 2, we describe downsizing methods for the multiplier. Section 3 covers the design details of our proposed multipliers. In Section 4, we evaluate our designs and discuss our findings. And finally, Section 5 concludes this paper.

2. Downsizing Methods for the Multiplier

The multiplication process can be divided into 3 stages, namely the partial product generation (PPG) stage, the reduction stage and the final addition stage. Fig.1 shows the structure of a conventional multiplier. The Booth method is often used in the PPG stage because of its ability to reduce the number of partial product rows generated before the reduction stage [7]. Although a higher radix Booth would result in a lesser number of partial product rows being generated, this comes at the cost of extra hardware in the PPG stage. For the reduction stage, the tree method [8] is hugely popular because of its high reduction speed. While the Wallace Tree suffers from an irregular structure and wiring complexity, the Binary Tree, which uses 4-2 compressors, on the other hand offers good regularity as well as design flexibility that are conducive for bit slicing [8], [9]. Furthermore, the array method [8], an alternative to the tree method, now demands a closer look with the implementation of bit slicing.

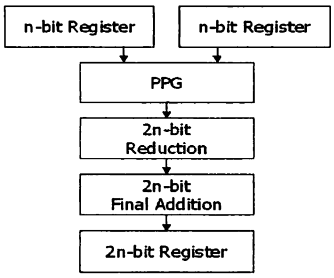

Fig. 1: Conventional Multiplier.

Fig.2 shows the process of bit slicing used in the proposed method. By splitting an n-bit data into four (n/4)-bit sets and processing them in parallel, the latency is reduced compared to using one large arithmetic unit. Processing the sets in series with one small arithmetic unit instead allows for a significant reduction in hardware. This is achieved by placing a selector before the arithmetic unit to channel the sliced bits in order. It is assumed that the single small arithmetic unit processes at a clock rate that is 4 times faster than the previous ones. Applying this technique to a multiplier shown in Fig.1 would result in Fig.3. Here, the PPG is replaced by the Booth Encoder. By selecting a fixed length of bits each cycle, the size of both the reduction unit and final adder can be reduced accordingly as shown in Fig.3. This length of bits is pre-determined by the size of the Final Adder used. The Distributor in the later stage rejoins the output bits from the final adder to yield the original length of result.

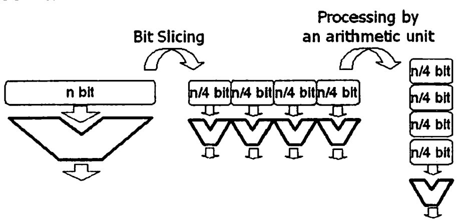

Fig. 2: Downsizing of Circuit Scale by Bit Slicing.

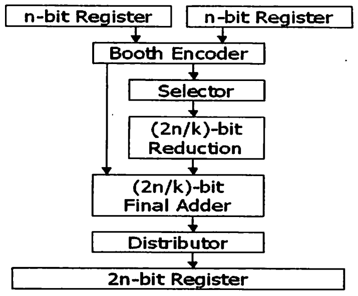

Fig. 3: Proposed Multiplier.

In this paper, the relation of Booth Order with bit slicing degree, hardware cost and latency for two reduction schemes: Binary Tree and arrays, is examined. The arrays are further divided into Simple, Double and Quadruple Arrays to look into the effects of bit slicing on low and high order arrays. For the final adder, a carry independent adder such as the Redundant Binary Adder [10] which has a latency that does not vary with its input bit size is used. In the following sections, the optimum conditions for a fast and low power multiplier are investigated.

3. Design Details of Proposed Multipliers

3.1. Structure of Reduction Unit

Fig.4 shows the structure of the proposed reduction unit based on Binary Tree for an 8-bit Final Adder. The input bit size of the reduction unit is fixed by the size of the final adder unit. The number of input bits N_{input} can be determined from that of the final adder N_{output} by

$$N_{input} = N_{output} + 2 \times (\log_2 n - t) \qquad (1)$$

Note that, N_{output} stands for the number of output bits of the reduction unit which is equal to the number of input bits of the final adder unit, n for initial input bit size (input bit size before bit slicing), and t for Booth Order (1 for non Booth, t for Booth t where $t = 2, 3, 4$ stands for radix 2, 4, 8, respectively). Fig.4 shows the case where $t = 1$, $n = 16$, and $N_{output} = 8$. Substituting the values into (1) yields $N_{input} = 14$ which stands for the input bit size of each partial products row into the reduction unit. The extra 6 bits is necessary for the propagation of carries. The number of arrows before each row of 4-2 compressors represents the remaining number of partial products rows after each intermediate reduction. Each row of 4-2 compressors reduces the number of partial products rows by half and the input bit size of each partial products row by 2.

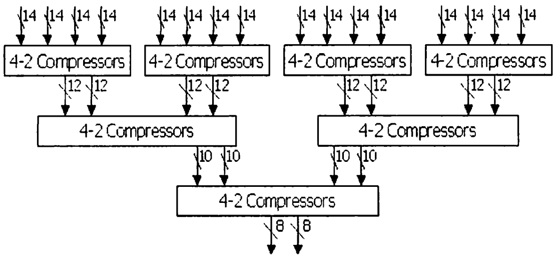

Fig. 4: Proposed Reduction Unit Based on Binary Tree ($t = 1$, $n = 16$).

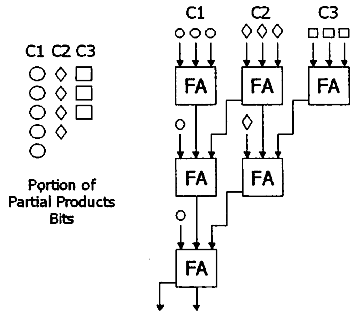

Fig. 6: Simple Array.

Fig.5 shows the reduction mechanisms for bit-sliced partial products using the proposed Binary Tree method. The partial products are divided into k (=4) segments denoted by input#1~4. All segments are equal in size and overlap one another. The size of each segment denotes the processing block that is needed to obtain output#1~4 of 8-bits each. The output bit size is determined by equation (2). The overlapping portion, $2(\log_2 n-1)$-bit in this case, is necessary to include the carry for each output. The process occurs from right to left.

$$N_{output} = 2 \times n / k \tag{2}$$

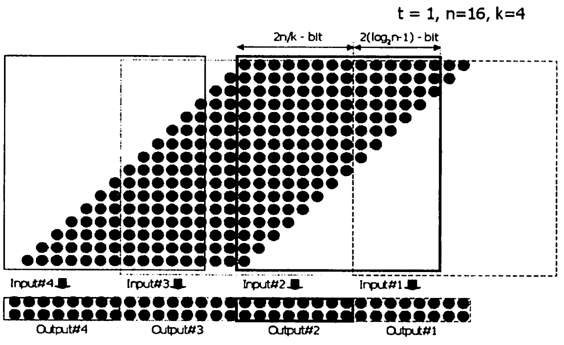

Fig. 5: Bit-sliced Partial Products Reduction Mechanism (Using Binary Tree).

The basic structure of the Simple Array is shown in Fig.6. The input order of the partial product bits are also shown on the left. Fig.7 shows the case for the Double Array. The Double Array comprises two Simple Arrays and a row of 4-2 compressors at the end to reduce 2 pairs of sums and carries to one single pair. The filled circles represent partial product bits in even rows while the unfilled circles represent those in odd rows. Similarly, the Quadruple Array comprises 4 Simple Arrays and two rows of 4-2 compressors to reduce 4 pairs of sum and carry bits to one single pair.

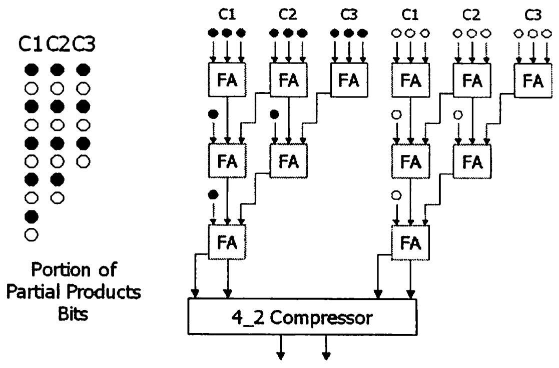

Fig. 7: Double Array.

The structure of the proposed Simple Array is shown in Fig.8. Here, a regular block of full adders (marked by the dotted box) is attached to the Simple Array in Fig.6. The number of full adders in each row within the regular block corresponds to the output bit size. The Simple Array of Fig.6 serves to propagate the necessary carries to the 4-bit output. As such, the size of the proposed Simple Array will grow by m+1 full adders (where m is number of full adders in the previous row) with every partial product row. With Double Array implementation, the number of rows of full adders needed to form one proposed Simple Array can be halved. Using Quadruple Array, this can be halved further. The only drawback lies in the logarithmic growth of 4-2 compressors at the end of the arrays (as shown in Fig.9, 10).

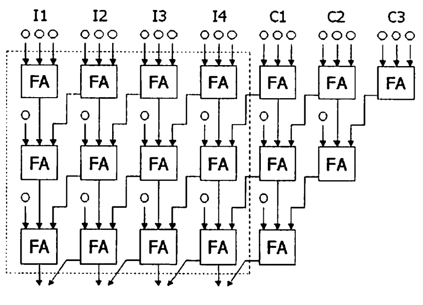

Fig. 8: Proposed Simple Array (4-bit output).

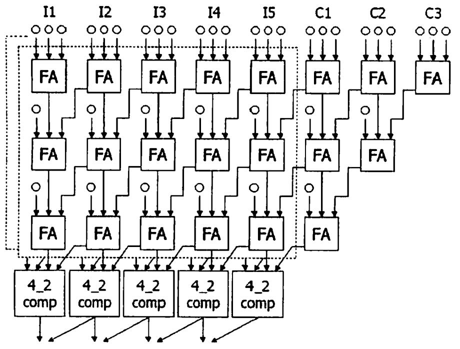

Fig. 9: Proposed Double Array (4-bit output).

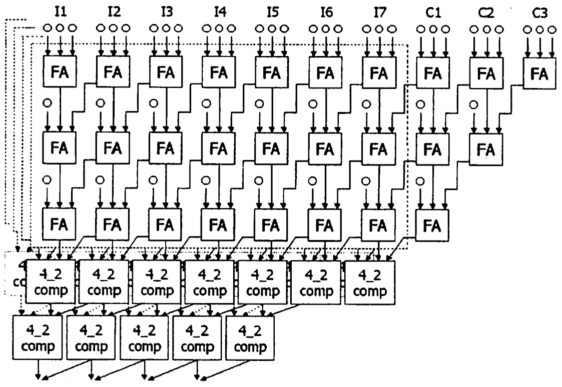

Fig. 10: Proposed Quadruple Array (4-bit output).

Fig.9 and Fig.10 show the structures of the proposed Double and Quadruple Arrays, respectively. While the proposed Simple Array can process up to 5 partial product rows, the proposed Double Array does up to 10 and the proposed Quadruple Array 20. Note that the regular blocks of full adders attached to the basic Simple Arrays expand in length along with the 4-2 compressors (I1~I5 in Fig.9 and I1~I7 in Fig.10) This observation leads to equation (3) for the proposed Simple, Double and Quadruple Arrays. Unlike for the Binary Tree, N_{input} for the arrays refers to the input bit size of the first partial product row only.

$$N_{input} = N_{output} + p + c + (n / 2^{(p+a)}) - 2 \qquad (3)$$

where $p = 0,1,2$ for Simple, Double, Quadruple Array, respectively; $c = p - 1$ for $p \geq 1$, $c = 0$ for $p = 0$; $a = 0$ for non-Booth, $a = t - 1$ for Booth t. As an example for the reduction unit based on arrays, Fig.11 shows the reduction mechanisms for bit-sliced partial products using the proposed Double Array method. The partial products are segmented into 4 equal parts denoted by input#1~4. The segments overlap one another. The processing block necessary to obtain 8-bit output#1~4 (using equation (2)) is denoted by each segment. Note that the 4-2 compressors are missing from this figure. The carry generation part observes a staircase-like structure due to the nature of the Arrays. The initial input bit size for this structure is (n/2)-2 bits while that of the regular block is (N_{output} + 1) bits. The process occurs from right to left.

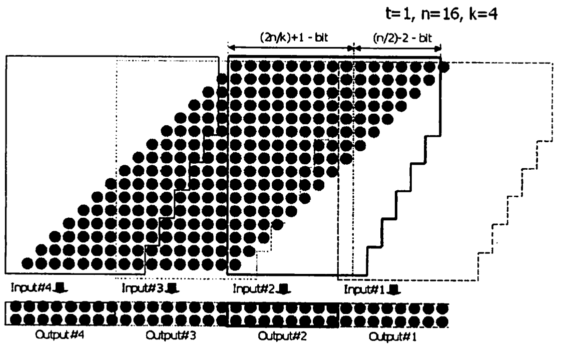

Fig. 11: Bit-sliced Partial Products Reduction Mechanism (Using Double Array).

3.2. Structure of Final Adder Unit

Fig.12 shows an 8-bit Redundant Binary Adder. The characteristic of the Redundant Binary Adder lies in its latency which is fixed at one EXOR + one AND gates for any input bit size. This adds up to less than the latency of one full adder which is two EXOR gates. A direct consequence of the fixed latency is that with larger inputs, the Redundant Binary Adder expands in width but not in logical depth. The output from the Redundant Binary Adder is in Redundant Binary format, but can be easily converted back to normal binary. In the next part, the Redundant Binary Adder is omitted from hardware evaluation because its area is considered insignificant compared to those of the Reduction units used in this study.

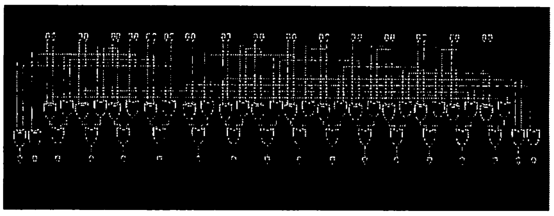

Fig. 12: 8-bit Redundant Binary Adder.

3.3. Estimation of Hardware Cost

For comparison, the hardware costs for the proposed reduction units are estimated in terms of number of 4-2 compressors or full adders used. Here, N_{42}, N_{FA} stand for number of 4-2 compressors and full adders, respectively.

The Binary Tree:

$$N_{42,Bin} = \sum_{i=1}^{(\log_2 n)-t} (n/2^{(i+1)}) \times (N_{input} - 2 \times (i-1)) \tag{4}$$

The Simple Array:

$$N_{FA,Sim} = ((n/2^{(p+a)}) - 2)(N_{input} + 3 - (n/2^{(p+a)})) + \sum_{j=4}^{n/2^{(p+a)}} (j-3) \tag{5}$$

The Double Array:

$$N_{FA,Dou} = 2 \times N_{FA,Sim} + 2 \times (N_{output} + 1) \tag{6}$$

The Quadruple Array:

$$N_{FA,Quad} = 4 \times N_{FA,Sim} + [4 \times (N_{output} + 3) + 2 \times (N_{output} + 1)] \tag{7}$$

The second term on the right hand side of equations (6), (7) represents the hardware required by the 4-2 compressors at the end of the Double and Quadruple Arrays, respectively. In the following section, the hardware costs as well as the latencies of the proposed designs are evaluated and discussed.

4. Evaluations and Discussions

In this section, the hardware costs and latencies of our designs are evaluated in terms of Booth Order and Slice Degree. Slice Degree refers to the number of times by which a set of partial products is equally divided for reduction with a smaller unit. Equation (8) expresses Slice Degree in terms of k, where $k = 2^0$, 2^1, 2^2, 2^3.

$$Slice\ Degree = \log_2 k \tag{8}$$

In this study, the design conditions are set as follows. Initial input bit size = 32-bit; Final output bit size = 64-bit; Non-Booth and Booth 2, 3, 4; Slice degree 0~4, whereby a Slice degree of 0 refers to no bit slicing at all. Target designs are the Binary Tree, Simple Array, Double Array and Quadruple Array.

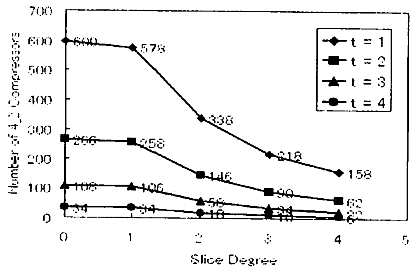

Fig.13: Hardware cost vs. Slice Degree (Binary Tree).

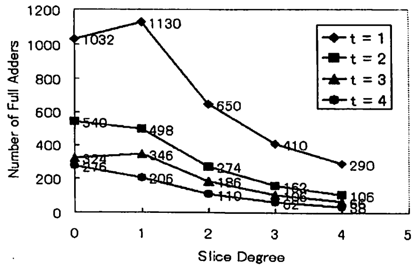

Fig.14: Hardware cost vs. Slice Degree (Quadruple Array).

The graphs of Hardware cost against Slice Degree for different Booth Orders are shown in Fig.13, 14 for Binary Tree and Quadruple Array, respectively. It can be observed that hardware savings improve with higher Slice Degrees. This improvement is more significant at non-Booth and lower Booth Orders. For Binary Tree and high-order arrays, hardware reduction by up to 50% is attainable with Slice Degree 2 compared to no bit slicing. With Slice Degree 4, hardware savings reach up to 75%.

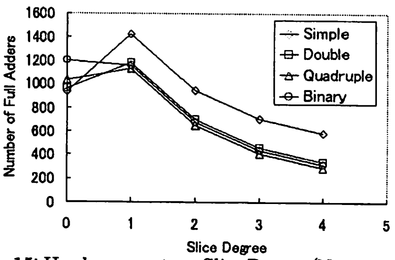

Fig.15: Hardware cost vs. Slice Degree (Non-Booth).

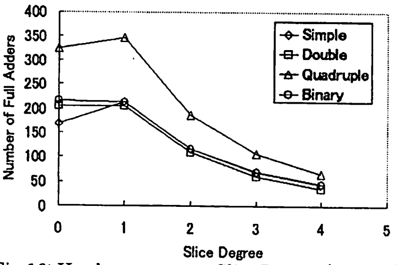

Fig.16: Hardware cost vs. Slice Degree (Booth 3).

Fig.15, 16 compare the hardware cost of Binary Tree to arrays at non-Booth and Booth 3 respectively. From the results, we see that high-order arrays require lesser hardware than Binary Tree at non-Booth, and this phenomenon is also observed with low-order Arrays at Booth 3. At Booth 3, hardware cost by Quadruple Array exceeds other schemes for all Slice Degrees. This is because the proportion of hardware dominated by the 4-2 compressors becomes critical. The number of these 4-2 compressors increases logarithmically with higher order arrays as can be observed in equations (6), (7). Notwithstanding, it can be deduced that a larger multiplication will sustain the trend as shown in Fig.15.

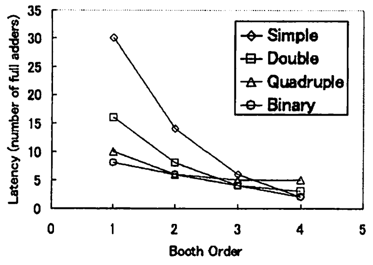

Fig.17: Latency vs. Booth Order (independent of Slice Degree).

Fig.17 shows the graph of Latency against Booth Order for all schemes. It is observed that the latency of each array type matches with that of the Binary Tree at a certain Booth Order. At Booth 2, the latency of Binary Tree matches with that of Quadruple Array; at Booth 3, with Double Array; and at Booth 4, with Simple Array. Based on this, it can be said that as Booth Order increases, the performance of higher order arrays drops relative to that of the Binary Tree. This is due to the latency incurred by the above-mentioned 4-2 compressors. A consequence of this tradeoff between the latency of the arrays (which decreases with higher order) and the latency of the 4-2 compressors (which increases with higher order) is that the reduction speed of arrays will never exceed that of the Binary Tree for any Slice Degree. Nevertheless, where the latency of both array and Binary Tree matches, hardware savings by the arrays are always better than those of the Binary Tree. This is shown in Table1. For the same latency, hardware savings are largest at 18.2% with Booth 3 for the Double Array and Binary Tree pair (using equations (9), (10)). Hardware savings are in- dependent of Slice Degree except at Slice Degree 0. Based on the findings, it can be concluded from this study that the arrays with bit slicing technique applied are more suitable to realize a compact and fast multiplier than with the Binary Tree. However, it should be added that a further study into the hardware requirements for the Booth method is necessary to determine the optimum Booth Order and Slice Degree.

Table.1: Hardware Savings at Same Latency - Arrays vs. Binary Tree (independent of Slice Degree except 0).

	Latency (#FA)	Hardware Savings (%)
Non-Booth Octuple / Binary	8	12.0
Booth 2 Quadruple / Binary	6	14.5
Booth 3 Double / Binary	4	18.2
Booth 4 Single / Binary	2	8.3

$$Hardware\ Savings = \frac{\#FA_{Binary} - \#FA_{Array}}{\#FA_{Binary}} \times 100\% \quad (9)$$

$$N_{FA} = 2 \times N_{42} \quad (10)$$

5. Conclusions

A comparison of regular reduction schemes, namely the Binary Tree, Simple, Double, and Quadruple Arrays is done by examining the effects of bit slicing and Booth Order on hardware cost and latency. It is found that hardware cost can be reduced by up to 75% with intensive bit slicing, and the Arrays show more potential in realizing a fast and compact arithmetic unit than the Binary Tree. Future works include SPICE simulations for advanced CMOS technologies.

Acknowledgments

This work is supported by VLSI Design and Education Center (VDEC), the University of Tokyo in collaboration with Synopsys, Inc. and Grant-in-Aid for Scientific Research (KAKENHI) #17700044 and #18700040.

References

[1] G. M. Link, N. Vijaykrishnan," Thermal Trends in Emerging Technologies," ISQED proc. Of Quality Electronic Design, pp.625 - 632, Mar. 2006.

[2] V. K. Pamula, K. Chakrabarty," Cooling of integrated circuits using droplet-based microfluidics," GLSVLSI proc. Of VLSI, pp.84-87, Apr. 2003.

[3] Z. Hu, A. Buyuktosunoglu, V. Srinivasan, V. Zyuban, H. Jacobson, P. Bose," Microarchitectural Techniques for Power Gating of Execution Units," ISLPED proc. Of Low Power Electronics and Design, pp.32 - 37, Aug. 2004.

[4] Kamal S. Khouri, Niraj K. Jha," Leakage Power Analysis and Reduction During Behavioral Synthesis", IEEE trans. on VLSI Systems, Vol. 10, No. 6, pp.876-885, Dec. 2005.

[5] J. Tada, R. Egawa, G. Goto and T. Nakamura," Compaction of Arithmetic Unit with Bit-Level-Parallelism", IEICE Tech. Report, Vol. 105, No. 350, pp.31-35, Oct. 2005.

[6] R. Egawa, J. Tada, G. Goto, T. Nakamura," A Sophisticated Multiplier in Advanced CMOS Technologies," ITC proc. Of Circuits/Systems, Computers and Communications, ITC-CSCC, Vol. 2, pp.53-56, 2006.

[7] D. Booth," A Signed Binary Multiplication Technique", Quarterly J. Mech. and Applied Math., 4, pp.236-240, 1951.

[8] H. Al-Twaijry," Area and Performance Optimized CMOS Multipliers," Aug. 1997.

[9] C. S. Wallace," A suggestion for a fast multiplier", IEEE trans. on Computers, Vol.13, pp.14-17, Feb. 1964.

[10] I. Koren," Computer Arithmetic Algorithms", 2, pp.17-24, 1983.