Vol.2021-SLDM-195 No.2
2021/10/15

連想メモリベース超並列SIMD型演算コアを用いた乗算手法の実装と評価

蔭山　享佑[†1,a)]　荒井　聡太[†1]　濱野　甫[†1]　濱井　彰光[†1]　孔　祥博[†1,b)]　小出　哲士[†2,c)]
熊木　武志[†1,d)]

概要：近年，画像，AI 等，様々なマルチメディア処理がモバイル機器で実行されるに従って，高性能，プログラマブル及び汎用性のあるプロセッサが求められてきている．一般に，マルチメディア処理は繰り返し演算処理とテーブルルックアップ処理から構成されており，我々はこれらを高速に処理可能な連想メモリベース超並列 SIMD 型演算コア（CAMX）を提案してきた．CAMX は左右の連想メモリが 1,000 を超える小規模演算器群を挟み込んだ構成となっており，モバイル機器に組込まれているプロセッサのアクセラレータとして機能する．本論文では，CAMX に検索加算繰り返し演算処理及び Baugh-Wooley 演算処理による乗算を実装し，最適な乗算手法を検証する．その結果，15 ビット以上の乗算では Baugh-Wooley 演算処理による乗算が検索加算繰り返し演算処理による乗算よりも高速に演算可能であることを確認した．

Implementation and evaluation of multiplication algorithm for content addressable memory-based massive-parallel SIMD matrix core

1. はじめに

スマートフォンやウェアラブルデバイスの様なモバイル機器は，我々の日常生活において急速に利用される様になってきており，画像，AI 等のマルチメディア処理を容易に実行できるようになっている [1], [2], [3], [4], [5]. これはモバイル機器にとって，1 個のプロセッサ上で複数のマルチメディア処理を実行する “ディジタルコンバージェンス時代” の到来を意味している [6], [7]．つまり，モバイル機器のプロセッサは複数の専用回路コアに頼らず，高性能，プログラマブル及び汎用性を実現する必要がある．

一般に，マルチメディア処理は，AND，OR，XOR，加算，乗算等の繰り返し演算処理 [8], [9] と入力データを参照テーブルを用いて出力データに変換するテーブルルックアップ処理 [10] から構成される． 我々はこの構成に着目し，連想メモリベース超並列 SIMD 型演算コア（CAMX）を提案してきた [11], [12], [13], [14], [15]．CAMX は繰り返し演算処理とテーブルルックアップ処理を並列に実行可能であり，論理演算，算術演算，シフト，検索等，様々な基本命令の動作を確認してきている．本稿では，CAMX において高速に乗算を処理可能な手法について検討するため，検索加算繰り返し演算処理及び Baugh-Wooley 演算処理による乗算を実装及び評価する．

[†1] 現在，立命館大学
Presently with Kusatsu, Shiga 525–8577, Japan
[†2] 現在，広島大学ナノデバイス・バイオ融合科学研究所 (RNBS)
Presently with Higashi-Hiroshima, Hiroshima 739–8527, Japan
[a)] ri0010hr@ed.ritsumei.ac.jp
[b)] kong@fc.ritsumei.ac.jp
[c)] koide@hiroshima-u.ac.jp
[d)] kumaki@fc.ritsumei.ac.jp

2. 連想メモリベース超並列SIMD型演算コア（CAMX）

CAMX は Content Addressable Memory-based Massive-parallel SIMD matriX core の略であり，2 つの連想メモリが小規模演算器群を挟み込んだ形で配置されている．図 1 に，提案演算コアのブロック図を示す．連想メモリは高速なデータ検索が可能で，検索時にマスクを利用することで任意のビット位置に対してデータの検索も可能である．つまり，格納しているデータに対して 1 ビット単位に検索処理を行え，全データをビットシリアル・ワードパラレルに

処理できる．1 エントリ内には，1 ワードの連想メモリセルが左右にあり，中央に 1 ビット演算器 (PE) が配置されている．演算を実行する際には，どちらか一方の連想メモリに対してマスクを利用した 1 ビット検索処理を行い，読み出したデータをレジスタに格納する．次のクロックサイクルで反対側の連想メモリに対してマスクを利用した 1 ビット検索処理によるデータ読み出しを実行し，レジスタに格納されているデータと演算を行い，その結果をレジスタに格納する．この結果は，その次のクロックサイクルで始めにデータを読み出した連想メモリセルに書き込まれる．以上の動作はパイプライン処理で高速に行われる．

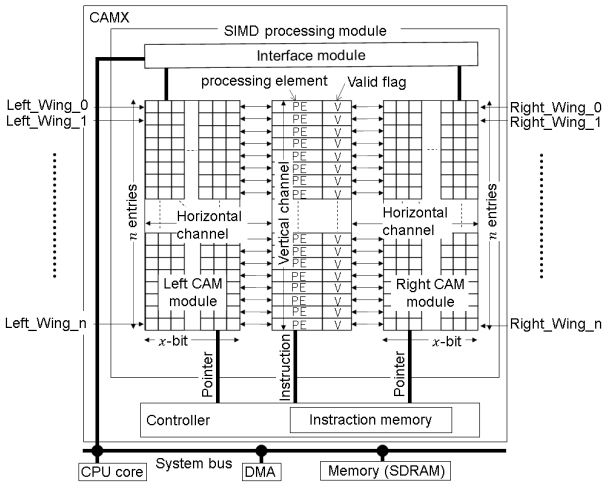

図 1　CAMX のアーキテクチャ
Fig. 1　Detailed CAMX architecture.

3. 符号付き乗算処理

CAMX は，汎用性及び省面積化を重視しているため，乗算処理を行う専用回路は組込んでいない．そのため，乗算は加算及び論理演算を用いて計算する必要がある．乗算はマルチメディア処理において必要な演算であり，乗算処理を高速化することは必須である．そこで，本章では，検索加算繰り返し演算処理及び Baugh-Wooley 演算処理による乗算について検討を行い，実行方法について詳述する．また，CAMX は乗算処理の乗数及び被乗数のビット幅を自由に変更が可能であり，並列度も自由に調整可能である．

3.1 検索加算繰り返し演算処理による乗算手法

本節では，検索加算繰り返し演算処理による乗算手法について説明する．検索加算繰り返し演算処理による乗算はこれまでのビットシリアルの乗算よりも少ないクロックサイクル数で処理可能であることを確認している [14]．この手法は，乗数が 1 の場合に被乗数のビット幅を調整して加算を行い，乗数が 0 の場合には加算を行わない手法である．これにより，ビットシリアルな通常の乗算では乗数が 0 と 1 の場合それぞれに対して加算を行わなければならないが，乗数が 1 であるか検索し，一致した時にのみ加算を行うことで，乗数が 0 の場合には加算を行う必要がない．そのため，乗算時の加算回数を削減することができ，高速に乗算が可能となる．4 ビットの乗算を例とした場合，検索加算繰り返し演算処理は図 2 の様に，乗数及び被乗数を拡張した後に被乗数のビット幅を調整しながら加算を繰り返すことになる．なお，加算は乗数が 1 の場合のみに行われる．CAMX では，乗算対象の乗数と被乗数を左右の連想メモリに格納し，右の連想メモリに格納された乗数を検索命令により，1 ビットずつ検索を行い，1 である場合にのみ，左の連想メモリの被乗数のビット幅を調整しながら加算していく（図 3）．

$$
\begin{array}{cccccccccc}
 & x_7 & x_6 & x_5 & x_4 & x_3 & x_2 & x_1 & x_0 \\
\times & y_7 & y_6 & y_5 & y_4 & y_3 & y_2 & y_1 & y_0 \\
\hline
 & x_7y_0 & x_6y_0 & x_5y_0 & x_4y_0 & x_3y_0 & x_2y_0 & x_1y_0 & x_0y_0 \\
 & x_6y_1 & x_5y_1 & x_4y_1 & x_3y_1 & x_2y_1 & x_1y_1 & x_0y_1 & \\
 & x_5y_2 & x_4y_2 & x_3y_2 & x_2y_2 & x_1y_2 & x_0y_2 & & \\
 & x_4y_3 & x_3y_3 & x_2y_3 & x_1y_3 & x_0y_3 & & & \\
 & x_3y_4 & x_2y_4 & x_1y_4 & x_0y_4 & & & & \\
 & x_2y_5 & x_1y_5 & x_0y_5 & & & & & \\
 & x_1y_6 & x_0y_6 & & & & & & \\
 & x_0y_7 & & & & & & & \\
\hline
 & p_7 & p_6 & p_5 & p_4 & p_3 & p_2 & p_1 & p_0
\end{array}
$$

図 2　検索加算繰り返し演算処理による乗算．
Fig. 2　Multiplication by search-addition iterative arithmetic processing.

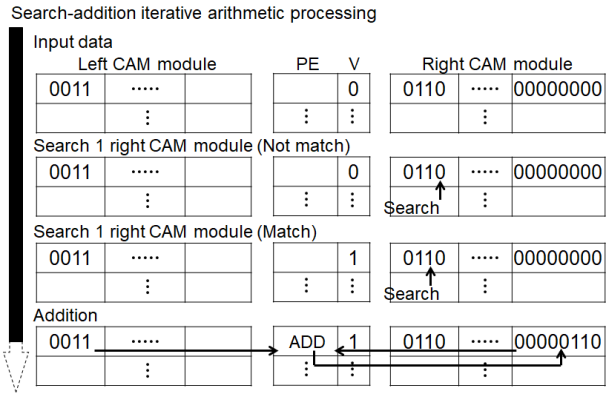

図 3　CAMX における検索加算繰り返し演算処理による乗算方法．
Fig. 3　Multiplication flow by search-addition iterative arithmetic processing.

3.2 Baugh-Wooley 演算処理による乗算手法

本節では，Baugh-Wooley 演算処理による乗算手法について説明する．Baugh-Wooley 演算処理による乗算は，n ビットの 2 進数として，A を $a_{n-1}a_{n-2}...a_1a_0$ 及び B を

$b_{n-1}b_{n-2}...b_1b_0$ とすると，2 の補数を考慮して式（1）で表される [22], [23], [24]．つまり，式（1）を 4 ビット乗算と考えた場合，図 4 に示すように最上位ビットの符号部を考慮した手法となる．

$$A = -a_{n-1}2^{n-1} + \sum_{i=0}^{n-2} a_i 2^i$$

$$B = -b_{n-1}2^{n-1} + \sum_{i=0}^{n-2} b_i 2^i$$

$$\begin{aligned} A \times B = {} & a_{n-1}b_{n-1}2^{2n-2} + \sum_{i=0}^{n-2}\sum_{j=0}^{n-2} a_i b_j \cdot 2^{i+j} \\ & + 2^{n-1}\left(-2^{n-1} + \sum_{j=0}^{n-2} \overline{a_{n-1}b_j}2^j + 1\right) \\ & + 2^{n-1}\left(-2^{n-1} + \sum_{i=0}^{n-2} \overline{a_{n-1}b_i}2^i + 1\right) \end{aligned} \quad (1)$$

この手法を CAMX で実行する方法を図 5 に示す．まず，左右の連想メモリに乗数及び被乗数を格納し，左の連想メモリに格納した被乗数の最上位ビットを反転する．そして，検索加算繰り返し演算処理と同様に，右の連想メモリに格納した乗数を検索命令により最上位ビット以外の各ビットの検索を行い，検索結果が 1 の場合には左の連想メモリに格納された被乗数を加算する．検索結果が 0 の場合には左の連想メモリに格納された “1000” を加算する．この処理を繰り返し，最上位ビットを計算する場合，被乗数の全ビットを反転した後に，乗数の最上位ビットを検索し，1 の場合に被乗数を加算する．

				x_3	x_2	x_1	x_0
×				y_3	y_2	y_1	y_0
				$\overline{x_3y_0}$	x_2y_0	x_1y_0	x_0y_0
			$\overline{x_3y_1}$	x_2y_1	x_1y_1	x_0y_1	
		$\overline{x_3y_2}$	x_2y_2	x_1y_2	x_0y_2		
	$\overline{x_3y_3}$	$\overline{x_2y_3}$	$\overline{x_1y_3}$	$\overline{x_0y_3}$			
1	0	0	1				
p_7	p_6	p_5	p_4	p_3	p_2	p_1	p_0

図 4　Baugh-Wooley 演算処理による乗算．
Fig. 4　Multiplication by Baugh-Wooley arithmetic processing.

4. 実験結果

4.1 シミュレーション結果

本節では，4 ビット乗算のシミュレーション結果を示す（図 6）．シミュレーションでは，Xillinx 社の Vivado v2019.2.1 を利用する．CAMX の連想メモリサイズは 1,024 エントリ（$= n$）かつ 256 ビット（$= x$）とする．図 6 では，1 エントリのみの結果を示しているが，1,024 エントリが並列に実行される．CLK はシステムクロックであり，$BUSY$ は命令タイミング信号である．$Lmem$ 及び $Rmem$ は左右の CAM モジュールにおける 1 エントリの格納データである．

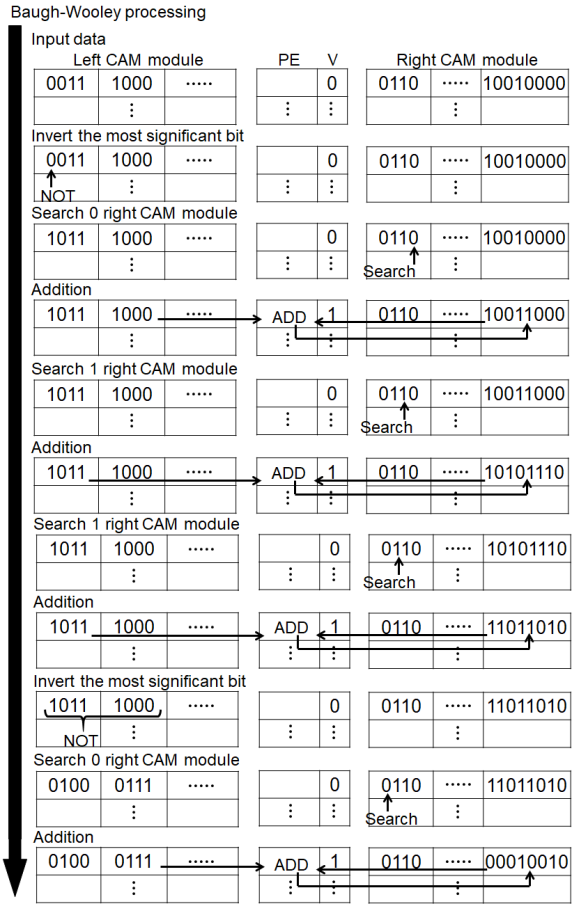

図 5　CAMX における Baugh-Wooley 演算処理による乗算方法．
Fig. 5　Multiplication flow by Baugh-Wooley arithmetic processing.

図 6 (a) は検索加算繰り返し演算処理による乗算，図 6 (b) は Baugh-Wooley 演算処理による乗算であり，どちらも “0011” と “0110” の乗算結果を示している．右の連想メモリに乗算結果である “00010010” が格納されていることを確認した．また，4 ビット乗算の場合，乗算のデータを 1,024 エントリに入出力する処理まで含めて，検索加算繰り返し演算処理による乗算は 4,225 クロックサイクルであり，Baugh-Wooley 演算処理による乗算は 4,298 クロックサイクルである（図 6 (c)）．

4.2 乗算ビット幅による処理速度比較

本節では，乗算ビット幅を 4 ビットから 32 ビットまで変更した場合のクロックサイクル数を比較する．図 7 は検索加算繰り返し演算処理及び Baugh-Wooley 演算処理によって乗算対象のビット幅を 4 ビットから 32 ビットまで変更してシミュレーションしたクロックサイクル数である．横軸は乗算対象のビット幅である．縦軸は乗数及び被乗数を

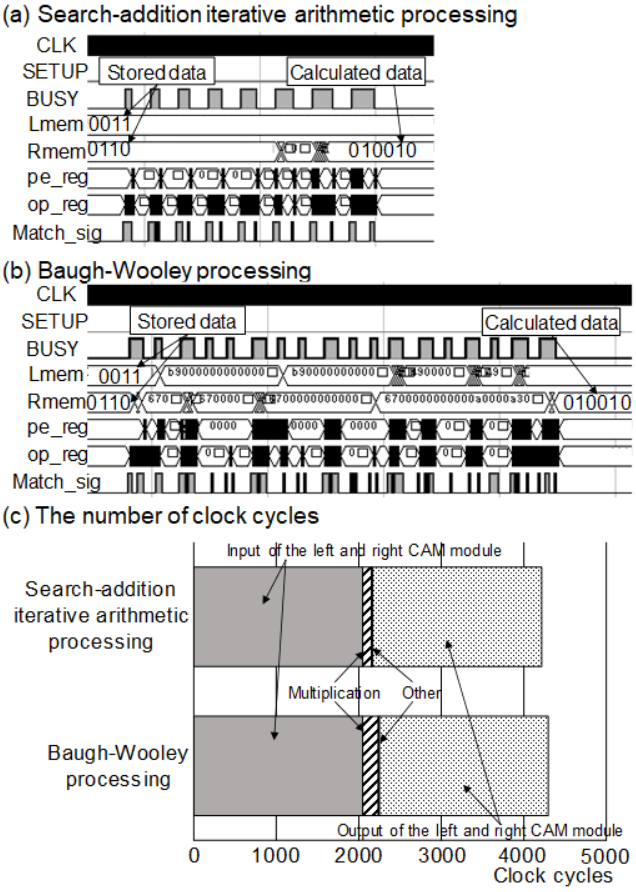

図 6 4 ビット乗算のシミュレーション結果.
Fig. 6 Simulation result of 4-bit multiplication.

1,024 エントリにデータを入力し，1,024 エントリの乗算結果を出力するまでのクロックサイクル数である．この結果，15 ビット未満の乗算では，検索加算繰り返し演算処理による手法の方が，少ないクロックサイクル数で処理でき，15 ビット以上の乗算では，Baugh-Wooley 演算処理による手法の方が，少ないクロックサイクル数で処理できることを確認できた．これは，乗算時における加算の回数，ビット幅が影響している．CAMX は加算するデータのビット幅に併せてクロックサイクル数が増減するため [13]，加算するデータのビット幅が小さいほどクロックサイクル数は減少する．さらに，加算の回数が減少するとクロックサイクル数も減少する．また，命令間のクロックサイクル数も影響する．このため，15 ビット以上の乗算では，検索加算繰り返し演算処理は Baugh-Wooley 演算処理より加算の回数が増加し，加算のビット幅も大きくなったことから，クロックサイクル数も増加したと考えられる．さらに，検索加算繰り返し演算処理は検索の後に加算処理を繰り返すが，Baugh-Wooley 演算処理は加算の後に加算時の桁上げデータの一時的な格納処理等を行うことから，命令順序に影響して命令間のクロックサイクル数は Baugh-Wooley 演算処理より検索加算繰り返し演算処理の方が増加したと考えられる．以上から，CAMX においては，15 ビット未満の乗算では検索加算繰り返し演算処理を利用し，15 ビット以上の乗算では Baugh-Wooley 演算処理を利用した方がより高速に処理できる．

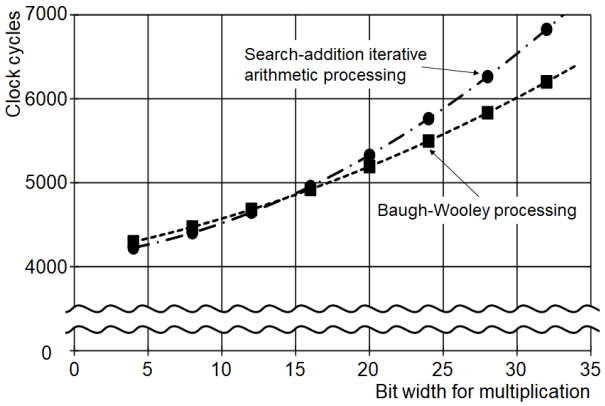

図 7 乗算ビット数によるクロックサイクル数の比較.
Fig. 7 Comparison clock cycle of n-bit multiplication.

5. まとめ

近年，モバイル機器において，高性能，プログラマブル及び汎用性のあるプロセッサでマルチメディア処理を高速に実行する要求が増加している．マルチメディア処理は繰り返し演算処理とテーブルルックアップ処理を並列に実行することでより高速に実行可能であり，我々はこれらを処理可能な連想メモリベース超並列 SIMD 型演算コア（CAMX）を提案してきた．本論文では，CAMX で最適な乗算手法を検証するため，検索加算繰り返し演算処理及び Baugh-Wooley 演算処理による乗算を実装した．この結果，15 ビット以上の乗算では Baugh-Wooley 演算処理による乗算が検索加算繰り返し演算処理による乗算よりも高速に演算可能であることを確認した．

謝辞 本研究の一部は，日本学術振興会科学研究費補助金 (基盤研究 (C)) (19K04561，2018) と 2021 年度村田学術振興財団研究助成の支援により行われた．

参考文献

[1] R. Thabet, R. Mahmoudi and M. H. Bedoui: *Image Processing on Mobile Devices: An Overview*, International Image Processing, Applications and Systems Conference, pp.1–8 (2014).
[2] L. Xiu, B. Ma, K. Zhu and L. Zhang: *Implementation and Optimization of Image Acquisition with Smartphones in Computer Vision*, 2018 International Conference on Information Networking, pp.261–266 (2018).
[3] K. Kageyama, K. Sugiyama, T. Kumaki and T. Fujino: *1/f fluctuation-based visible light beacon for spy-photo prevention system*, RISP International workshop on Nonlinear Circuit, computer and Signal Processing (2016).
[4] K. Kageyama, T. Kumaki, T. Ogura and T. Fujino: *Digital image forensics using morphological pattern spectrum*, Journal of Signal Processing, Vol.19, No.4, pp.159–162 (2015).

[5] A. Ahmed and E. Ahmed: *A Survey on Mobile Edge Computing*, 2016 10th International Conference on Intelligent Systems and Control (2016).
[6] K. Uchiyama: *Processor technology in system LSI*, Journal of the Institute of Electronics, Information and Communication Engineers, Vol.95, No.7, pp.582–588 (2012).
[7] K. Uchiyama: *Power-efficient heteregoneous parallelism for digital convergence*, 2008 IEEE Symposium on VLSI Circuits, pp.6–9 (2008).
[8] M. K. Mandal: *Multimedia signals and systems*, Springer US (2003).
[9] L. Vincent: *Morphological Algorithms*, Mathematical Morphology in Image Processing, pp.255–288 (1992).
[10] AES の処理について, 入手先 ⟨https://www.tutorialspoint.com/cryptography/advanced_encryption_standard.htm⟩ (参照 2021-09-20).
[11] T. Kumaki: *Development of Content Addressable Memory-based Massive-parallel SIMD Matrix Core*, LSI and Systems Workshop 2017 (2017).
[12] K. Watanabe, A. Sekino, K. Kageyama, T. Koide and T. Kumaki: *Verification by Simulating Content Addressable Memory-based Massive-parallel SIMD Matrix Core*, LSI and Systems Workshop 2019 (2019).
[13] K. Kageyama, A. Sekino, K. Watanabe, A. Hamai, T. Koide and T. Kumaki: *Proposal of content addressable memory-based massive-parallel SIMD matrix core*, RISP International workshop on Nonlinear Circuit, computer and Signal Processing (2020).
[14] K. Kageyama, K. Watanabe, A. Hamai, T. Koide and T. Kumaki: *Acceleration of arithmetic processing with CAM-based massive-parallel SIMD matrix core*, IEEE International MidWest Symposium on Circuits And Systems (2020).
[15] K. Watanabe, K. Kageyama, A. Sekino, A. Hamai and T. Kumaki: *Basic operation verification of content addressable memory-based massive-parallel SIMD matrix core for multimedia applications*, International symposium on biomedical engineering (2019).
[16] M. Nakajima, H. Noda, K. Dosaka, K. Nakata, M. Higashida, O. Yamamoto, K. Mizumoto, H. Kondo, Y. Shimazu, K. Arimoto, K. Saitoh and T. Shimizu: *A 40GOPS 250mW massively parallel processor based on matrix architecture*, 2006 IEEE International Solid State Circuits Conference - Digest of Technical Papers (2006).
[17] T. Kumaki, M. Ishizaki, T. Koide, H. J. Mattausch, Y. Kuroda, T. Gyohten, H. Noda, K. Dosaka, K. Arimoto and K. Saito: *Integration architecture of content addressable memory and massive-parallel memory-embedded SIMD matrix for versatile multimedia processor*, IEICE Trans. Electron., Vol.E91-C, No.9, pp.1409–1418 (2008).
[18] ルネサス SH-2A コアを搭載した画像処理プロセッサ発売 入手先 ⟨http://www.kumikomi.net/article/news/2008/06/25_01.php⟩ (参照 2021-09-20).
[19] H. Noda, T. Tanizaki, T. Gyohten, K. Dosaka, M. Nakajima, K. Mizumoto, K. Yoshida, T. Iwao, T. Nishijima, Y. Okuno, and K. Arimoto: *The Circuits and Robust Design Methodology of the Massively Parallel Processor Based on the Matrix Architecture*, 2006 Symposium on VLSI Circuits Digest of Technical Papers, pp. 260–261 (2006).
[20] T. Kumaki, T. Koide, and T. Fujino: *Secure data processing with massive-parallel simd matrix for embedded soc in digital-convergence mobile devices*, IEEJ Transactions on Electrical and Electronic Engineering, Vol. 12, No. 1, pp. 96–104 (2017).
[21] K. Kageyama, T. Koide, and T. Kumaki: *Parallel Processing of Morphological Pattern Spectrum for a Massive-Parallel Memory-Embedded SIMD Matrix Processor MX-1*, IEEJ Transactions on Electronics, Information and Systems, Vol. 139, No. 3, pp. 237–246 (2019).
[22] A. Badawi, A. Alqarni, A. Aljuffri, M. S. BenSaleh, A. M. Obeid, and S. M. Qasim: *FPGA Realization and Performance Evaluation of Fixed-Width Modified Baugh-Wooley Multiplier*, 2015 Third International Conference on Technological Advances in Electrical, Electronics and Computer Engineering, pp. 155–158 (2015).
[23] M. Sj?lander, and P. Larsson-Edefors: *High-Speed and Low-Power Multipliers Using the Baugh-Wooley Algorithm and HPM Reduction Tree*, 2008 15th IEEE International Conference on Electronics, Circuits and Systems, pp. 33–36 (2008).
[24] A. Mukherjee, and A. Asati: *Generic Modified Baugh Wooley Multiplier*, 2013 International Conference on Circuits, Power and Computing Technologies, pp. 746–751 (2013).