Modulo-$(2^q - 3)$ Multiplication with Fully Modular Partial Product Generation and Reduction

Ghassem Jaberipur[1], Saeid Gorgin[1], Navid Ahamadian[2], Jeong-A Lee[1]
[1]Department of Computer Engineering, Chosun University, Gwangju, Republic of Korea
[2]Department of Computer Science and Engineering, Shahid Beheshti University, Tehran, Iran
jaberipur@ chosun.ac.kr, gorgin@chosun.ac.kr, n.ahmadian@mail.sbu.ac.ir, jalee@chosun.ac.kr

Abstract*—Given the residue number systems that contain moduli of the form $2^q \pm 1$ and $2^q \pm 3$, it is desirable to employ delay-balanced adders and multipliers, in order to synchronize the operation of parallel residue channels. The required modulo-$(2^q \pm 3)$ adders, with compatible speed with modulo-$(2^q \pm 1)$ adders, already exist with parallel prefix architectures. However, the previously reported modulo-$(2^q \pm 3)$ multipliers, in one way or another, produce the non-modular products of the residues at the outset and work towards yielding the final modular product. This seems to be the main source of incompatible performance with the existing modulo-$(2^q \pm 1)$ fully modular multipliers. Therefore, as the first endeavor, we were motivated to design and implement efficient modulo-$(2^q - 3)$ multipliers with fully modular partial product generation and reduction that are more compatible with their modulo-$(2^q - 1)$ counterparts. However, unlike the case of modulo $2^q - 1$, it turns out that the straightforward modulo-$(2^q - 3)$ partial product reduction (e.g., via Wallace-tree reduction with greedy use of full adders and half adders) falls into an infinite loop of reduction stages. Therefore, we undertake a modified reduction algorithm that requires at most two reduction levels more than that of the modulo-$(2^q - 1)$ case to converge. To ensure the correct operation of the algorithm and ease the design process, an in-house software program produces the exact composition of reduction cells in each level of partial product reduction. Analytical and synthesis-based evaluations of the proposed design, and the previous ones, exhibit better figures of merit, as regards the delay ($\geq 24\%$), area-delay ($\geq 6\%$) and energy ($\geq 10\%$) measures.

Keywords— Residue number system, Fully modular multiplication, Partial product reduction

I. INTRODUCTION

Residue number systems (RNS) and the corresponding modular arithmetic are widely used in the realization of several add/multiply intensive arithmetic applications, such as digital signal processing [1], image processing [2], machine learning platforms [3], multi-layer convolutional neural networks [4], and cryptosystems [5]. Extra arithmetic speed, lower power dissipation, and fault tolerance capability are the most desired benefits of performing modular addition and multiplication in a multi-channel parallel architecture of an RNS. The optimum efficiency is brought about in the case of equal-width residue channels corresponding to moduli of the form $2^q \pm \delta$. Most often, $\delta(< 2^{q-1})$ is so chosen to yield pairwise mutually prime moduli in order to maximize the dynamic range (i.e., cardinality of the numbers represented by the underlined RNS). Enforcing the same q across all the residue channels is generally essential (but not always sufficient) for balanced-delay modular operations between channels. Unfortunately, however, such delay-balanced property does not spread over all the δ-values. That is probably why the relevant RNS literature offers very few balanced modulo-$(2^q \pm \delta)$ adders and even fewer multiplier architectures for $\delta > 3$, while ultra-efficient modulo-$(2^q \pm 1)$ adders [6, 7] and multipliers exist [8].

Fig. 1 describes how a $q \times q$ modular partial product (MPP) bit-matrix is formed via modulo-$(2^q - 1)$ partial product generation (PPG). Note that the weighted-2^{q+i} $(0 \leq i < q)$ columns do not actually exist since, given that $|2^{q+i}|_{2^q-1} = 2^i$, the virtual gray/black shaded pixels of the corresponding $(q-1) \times (q-1)$ triangle, are reentered in positions $0 \leq i \leq q-1$ (see the pure gray-shaded actual pixels). For example, a 2^{q+1}-weighted non-modular product bit ■ reenters in position 1, as a modular bit ■. As such, the multiplier design is fully modular from the beginning, whether for PPG or partial product reduction (PPR). However, the bit-products that weigh less than 2^q form the upper-left $q \times q$ triangle of actual pure black pixels (i.e., ■) in the rightmost q columns. The operands of the final carry-propagate addition may equal $2^q - 1$ (i.e., equal to the modulo), which given the common practice of safe double zero representation in modulo-$(2^q - 1)$ addition and multiplication, no problem arises.

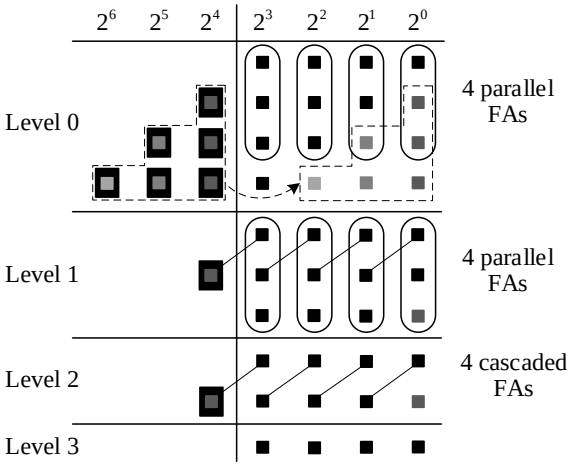

Fig. 1. Modulo-$(2^4 - 1)$ multiplication with fully modular PPG and PPR

In the case of modulo-$(2^q - 3)$ multipliers, we have encountered only three works in the relevant literature, where all base their designs, in one way or another, on the non-modular product of the residue operands. The final modulo-$(2^q - 3)$ adder, is preceded by alternative circuitries in each design; namely 1) Forward converter of the $2q$-bit plain binary product in [9]. 2) Forward converter of the $2q$-bit carry-save product in [10]. 3) A half-precision partial product generator for the least-significant half of the non-modular product, and another one for the most-significant half of the triple product, followed by a fused non-modular PPR of the two parts, with due aggregate end-around carry correction. The carry-save result of the latter enters a compound modulo-$(2^q - 3)$ adder whose outputs are multiplexed via the end-around carry digit [11]. A brief description of these methods is provided in Section II.A, and more details on our perception of the third one are in the Appendix 1.

In this work, in an effort for additional performance and closer compatibility with the companion modulo-$(2^q - 1)$ adder, we propose a modulo-$(2^q - 3)$ multiplication scheme with fully modular PPG and PPR, where the remaining sections are organized as follows. A brief background on RNS and the previous modulo-$(2^q \pm \delta)$ adder and multiplier architectures are offered in Section II, where in particular, we briefly examine the three previous contributions on modulo-$(2^q - 3)$ multipliers. In Section III, we propose the corresponding fully modular architecture and the results of analytical evaluations and synthesis outcomes of the previous designs where the due comparisons with the proposed one are compiled in Section IV. Finally, Section V contains our concluding remarks and plans for future works.

II. Background

A rather brief look at the previous modulo-$(2^q - 3)$ multiplication schemes follows a short introduction to RNS.

A. General RNS

The arithmetic operations of a k-moduli RNS take place in k independent parallel residue channels corresponding to moduli $\{m_1, \dots, m_k\}$. The i^{th} $(1 \le i \le k)$ channel commonly provides for a modulo-m_i operation-trio; namely, 1) residue generation, 2) addition, and 3) multiplication. There are commonly several (say $\mathcal{N}$) operands that arrive in sequence and enter the unit 1), aka forward converters, which receive an n-bit operand X and generates the $q(\ll n)$-bit residues $x_i = |X|_{m_i} = X - m_i \lfloor X/m_i \rfloor$, in parallel across the k channels, for $1 \le i \le k$. The x_i residues serve as one operand of units 2) or 3), where the other operand is commonly the interim result y_i of the previous consecutive modulo-m_i operations. On the arrival of the last input operand, the final k-tuple residue results feed an RNS-to-binary converter (aka reverse converter) that produces the final non-modular result.

The efficiency of an RNS computation depends on:

1) Number of modular operations: At the end of $\mathcal{N}$ modular operations on q-bit residues of the non-modular n-bit operands, the aggregate time savings due to $q \ll n$ may be much more than the time needed for the required final reverse conversion, if $\mathcal{N}$ is large enough.
2) The bit-width Q of the widest residue channel: The number of bits required for representing the residues determines the speed of operations in the corresponding channel. Therefore, RNS designers try to cover the dynamic range of the application via balanced width channels in order to minimize the delay of critical path. For example, the RNS with moduli set $\{2^q, 2^q \pm 1\}$ is very popular, where $Q = q$, across all three channels, for which balanced adders and multipliers exist.
3) Efficiency of the modular adders, multipliers, and less critically of forward and reverse converters.
4) Mutual primality of all moduli: This property maximizes the dynamic range. However, despite exhibiting the highest efficiency, only one power-of-two modulo 2^q is allowed, where the rest of moduli are commonly of the form $2^q \pm \delta$, with selected odd $\delta \in [1, 2^{q-1} - 1]$.

Efficient parallel prefix realizations of modulo-$(2^q \pm \delta)$ adders for $\delta \in \{1, 3, 2^j + 1\}$ exist [6, 7, 12, 13, 14] that provide for almost balanced delay. On the other hand, efficient modular multipliers have been offered for $\delta = 1$ [8], based on fully modular PPG and PPR (e.g., as in Fig. 1). However, the relevant literature is not as reach for $\delta \ge 3$.

B. Revisiting the previous contributions on modulo-$(2^q - 3)$ multiplication

For ease of evaluation and comparisons, we use the same notation for the explanation of the previous modulo-$(2^q - 3)$ multipliers. Given that $A, B \in [0, 2^q - 4]$, the non-modular product $A \times B$, satisfies $P = 2^q P_h + P_l \in [0, 2^{2q} + 16 - 2^{q+3}]$, where $P_h = p_{2q-1} p_{2q-2} \dots p_q$, and $P_l = p_{q-1} \dots p_1 p_0$. The number of reduction levels $\mathcal{L}(q)$ must satisfy the following.

$$2\left(\frac{3}{2}\right)^{\mathcal{L}(q)} \approx q \Rightarrow \mathcal{L}(q) \log\frac{3}{2} \approx \log\frac{q}{2} \Rightarrow \mathcal{L}(q) \approx \left\lceil 1.7 \log\frac{q}{2} \right\rceil$$

[9]: Equation (1), describes the essence of the contribution in [9], whose implementation is depicted in Fig. 2a, with three carry propagate additions (CPA) in sequence.

$$|A \times B|_{2^q - 3} = |2^q P_h + P_l|_{2^q - 3} = |3P_h + P_l|_{2^q - 3} \quad (1)$$

The first adder is roughly $2q - \mathcal{L}(q) \ge \frac{3q}{2}$-bit wide, with at least $\left(3 + 2\left\lceil \log\frac{3q}{2} \right\rceil\right) \Delta G \ge (4.2 + 2\log q)\Delta G$ delay, in parallel prefix realization, where ΔG denotes the delay of a simple 2-input gate. This is preceded by $\mathcal{L}(q) = \left\lceil 1.7 \log\frac{q}{2} \right\rceil$ CSA reduction levels, with the delay of $\lceil 6.8(\log q - 1) \rceil \Delta G$, ($4\Delta G$, for each level). Then the authors use a modulo-$(2^q - 3)$ (4:1) compressor, where no architecture nor implementation details are provided. However, the actual design and structure of the aforementioned compressor, unlike its modulo-$(2^q - 1)$ counterpart, is not trivial and bears several levels of modulo-$(2^q - 3)$ carry-save additions due to reentrant carry digits $\in \{0, 3\}$. Moreover, a single modulo-$(2^q - 3)$ addition at the last stage of the forward converter does not always yield the desired result. The reason is that the two q-bit operands may assume excess-modulo values in $\{2^q - 3, 2^q - 2, 2^q - 1\}$. Therefore, the interim sum can be greater than twice the modulo, which requires another round of modulo deduction. Nevertheless, in the lack of implementation details in the article, we assume the lower bounds for the delay components. That is $6\,\Delta G$, for one (4; 2) compressor and $2(3 + 2\lceil \log q \rceil)\Delta G$, for two parallel prefix q-bit adders, amounting to $(12 + 4\log q)\ \Delta G$. Therefore, a quite optimistic delay evaluation of this work leads to the overall delay figure roughly equal to $(10 + 13\log q)\Delta G$.

[10]: The multiplier of [10] is represented by (2), where $U + V$ is the carry-save representation of the product P, obtained in $6.8(\log q - 1)\Delta G$, the same as in [9]. Two levels of (4; 2) compressors modulo $(2^q - 3)$ reduce the six operands $(2U_h + U_h + 2V_h + V_h + U_l + V_l)$ of (2) to the q-bit input operands of the final modulo-$(2^q - 3)$ adder, where $U_h = u_{2q-1} \dots u_q$, $U_l = u_{q-1} \dots u_0$, $V_h = v_{2q-1} \dots v_q$, $V_l = v_{q-1} \dots v_{\mathcal{L}(q)} 0 \dots 0$. The corresponding delay is at least $12\Delta G$, for the (4; 2) compressors, and $2(3 + 2\lceil \log q \rceil)\ \Delta G$, for the final q-bit modular adders.

Fig. 2b depicts the required implementation steps, where one carry-propagate addition is saved due to the carry-save representation of the non-modular product. The same problem of excess-modulo operands persists as in the first work. Therefore, a very optimistic delay evaluation of this work leads to the overall delay figure of $(12 + 11\log q)\,\Delta G$.

$$|A \times B|_{2^q-3} = |U + V|_{2^q-3} = |2^q(U_h + V_h) + U_l + V_l|_{2^q-3} = |3(U_h + V_h) + U_l + V_l|_{2^q-3} \quad (2)$$

[11]: The main idea of the more recent third design is to produce a carry-save representation of $3P_h$ (P_l) via the most (least) significant half of the multiplier for $\times 3B$ $(A \times B)$. The triple and single half products are organized in the same partial product bit-matrix and reduced together. The carry-save result enters a compound modulo-$(2^q - 3)$ adder that yields the final modular product, where there are no details (though nontrivial) on the compound adder. For self-containment and sound basis for evaluation of Seidel's [11] work, we provide in the Appendix 1 a few clarifying illustrations, besides Fig. 2c, based on our comprehension of this innovative method. Nevertheless, unlike the aforementioned two older contributions, the double $\{0,1,2\}$ representation modulo-$(2^q - 3)$ adder of [12] has been used, with $(4 + 2\lceil\log q\rceil)\Delta G$ delay. The other delay components of this work are $4\Delta G$ for the PPG and Booth recoding, $4\mathcal{L}(2q)\Delta G = 4\lceil 1.7\log q\rceil\Delta G$ for PPR, and $6\Delta G$, for the (4; 2) compressor, which all together roughly amounts to $(14 + 9\log q)\Delta G$.

III. The Proposed Solution

The straightforward fully modular approach in designing the modulo-$(2^q - 3)$ multiplier follows the same path as in the similar design for modulo $2^q - 1$ (see Fig. 1). However, each 2^{q+i}-weighted bit ($0 \le i < q$), of a partial product, is reentered as two bits in positions i and $i+1$, since $\left|2^{q+i}c\right|_{2^q-3} = \left|2^q \times 2^i c\right|_{2^q-3} = 3 \times 2^i c = 2^{i+1}c + 2^i c$.

Therefore, as is depicted by Fig. 3 (for $q = 4$), the obtained Level 0 of the actual MPP matrix contains a $q \times q$ triangle of black pixels (same as in Fig. 1), a $(q-1)\times(q-1)$ triangle due to $2^i c$ component of reentrant product bits (also as in Fig. 1), and an additional shifted-left triangle due to $2^{i+1}c$ components. Consequently, there are q product bits in the rightmost column and $(2q - i)$ bits in the rest of columns for i $(1 \le i < q)$. In particular, the 2nd column (i.e., $i = 1$) from the right is the deepest one with $2q - 1$ bits. Fig. 3 further illustrates the next levels of the reduced MPP matrices, where the new reentrant product bits land only in the first two rightmost columns.

Unfortunately, however, utilizing the conventional Wallace-tree reduction (as in Fig. 1) leads to a repeated pattern of product bits in Level 5; hence an infinite loop of reduction levels that never reduces to a 2-deep MPP matrix. This problem, which is further analyzed in Section III.A below, is shared for all q-values, as is examined via an in-house software program, for $q \in [3,64]$.

A. The Problem

Recalling Fig. 3 and the aforementioned problem in the convergence of Wallace-tree reduction of modulo-$(2^q - 3)$ partial products, note that the main cause of such anomaly is due to the peculiarity of the second column from the right in the MPP matrix. This column is originally the deepest with $2q - 1$ bits. Moreover, it receives carry bits from two sources. 1) Ordinary carry spillovers due to reduction cells utilized in its right (i.e., weighted-1) column. 2) Direct reentrant carries from the reduction cells of the leftmost (i.e., weighted-2^{q-1}) column. A copy of the latter carries is also sent to the weighted-1 column, which in turn may cause additional spillovers to the left (i.e., to the weighted-2 column). On the other hand, a full adder (FA), as a reduction cell, decreases the column's depth effectively by 1, but an HA does not decrease the depth except for the rare cases of no carry spillover from the right. Such HA behavior suggests that when there are only two bits left in a column that are not assigned to an FA, one does not assign them to an HA and rather leaves them intact to be possibly used in the next reduction level by another FA.

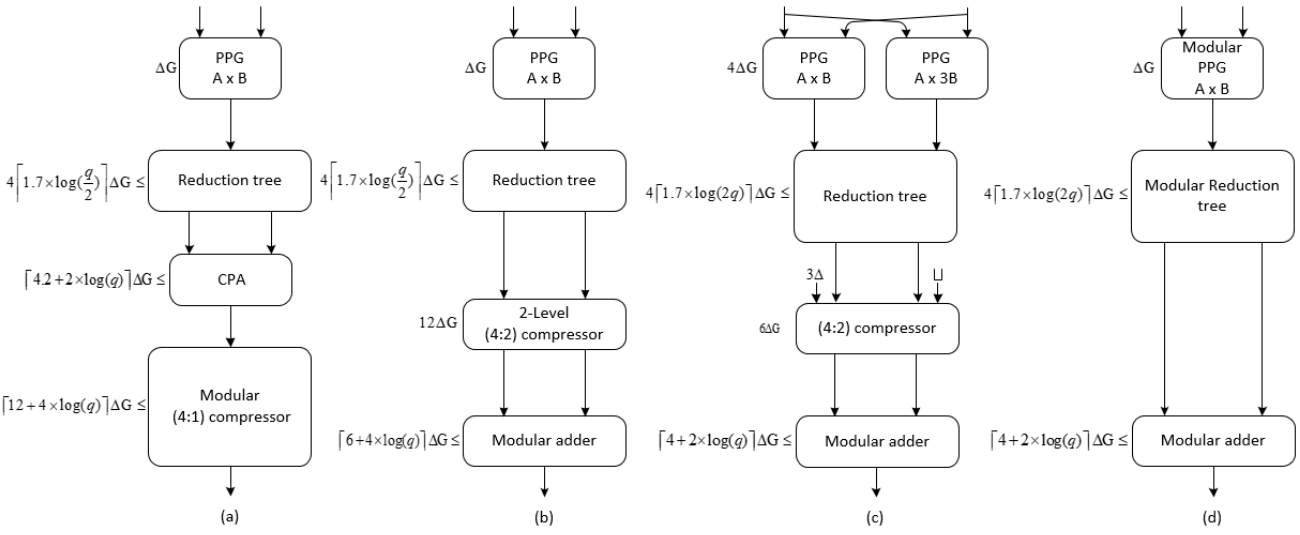

Fig. 2. Alternative designs for modulo-$(2^q - 3)$ multiplication, (a) [9], (b) [10], (c) [11], and (d) Proposed design. Components' delays are in terms of ΔG (i.e., the delay of a simple 2-input gate).

This strategy very much resembles the Dadda reduction method for non-modular PPR. However, it cannot be considered here as an exact application of Dadda reduction. The reason is that the essential property of equal depth of partial product matrices of Wallace and Dadda trees, in all reduction levels, may not be observed in modular reduction.

B. The Solution

Based on the above discussion, limiting the reduction cells to FAs (unless extra reduction levels are not avoidable) seems to lead to the desired convergence that was not possible when Wallace tree reduction rules are strictly observed. For example, Fig. 4 provides for the Dadda-like MPP matrix for $q = 4$, where the reduction process converges after five levels, which is only one level more than the minimum number of (3:2) reduction levels required for a (7:2) reduction.

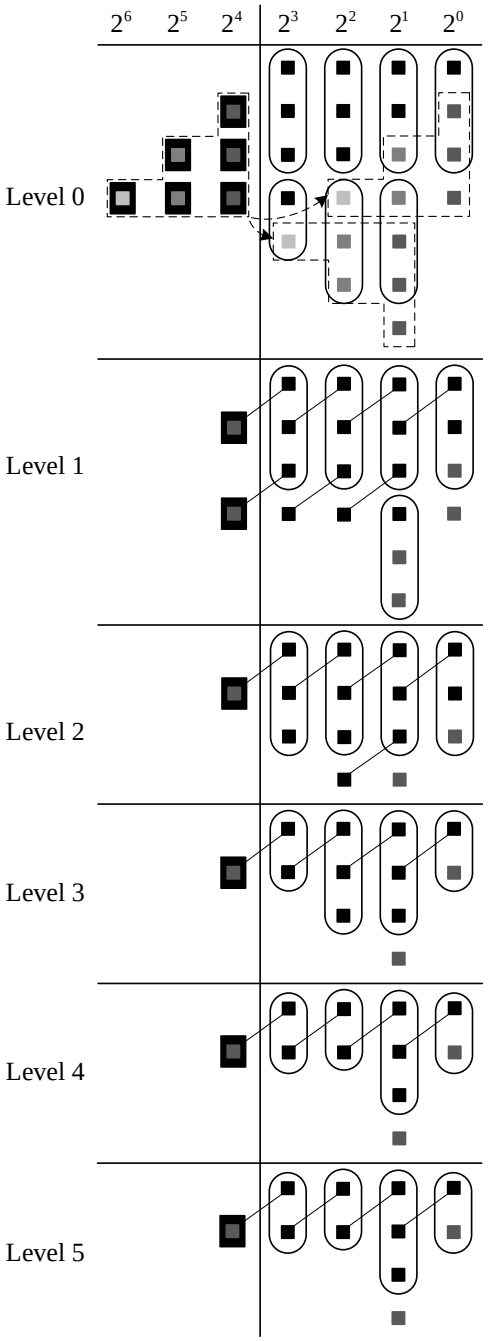

Fig. 3 Modulo-13 Wallace-tree PPR

However, via running the aforementioned in-house software program, for $q \in [4,64]$, we have found out that the latter undesired extra level (i.e., for $q = 4$) is required only for the case of $q = 32$ (i.e., 10 levels instead of 9 levels of non-modular reduction.

The latter Dadda-like reduction scheme is outlined in Algorithm 1, where $d_i = 2q - i (0 < i < q)$ denotes the depth of Column i in terms of bits, and $d_0 = q$. Recalling the identical reentrant bits in the Level 0 of Fig. 3, note that two outputs are forked out from $q(q-1)/2$ of the total $q \times q$ AND gates. A sample output of the corresponding software (slightly decorated) is given in the Appendix 2.

Algorithm 1 (Modulo-$(2^q - 3)$ fully modular multiplication):

Input: q-bit multiplicand and multiplier residues
Output: Composition of the required FA cells for all the reduction levels and columns

1. **Init**: $d_0 = q$, For $i \coloneqq 1$ to $q-1$ do $d_i = 2q - i$.
2. **MPPG**: Generate the Level-0 MPP matrix via a matrix of $q \times q$ AND gates;
3. Do while there exists a column with a depth more than 2
   a. Column #0: Apply $\left\lfloor \frac{d_0}{3} \right\rfloor$ FA reductions $\Longrightarrow$
      $d_0 = d_0 - 2\left\lfloor \frac{d_0}{3} \right\rfloor + \left\lfloor \frac{d_{q-1}}{3} \right\rfloor$; /*Sending (Receiving) $\left\lfloor \frac{d_0}{3} \right\rfloor$ $\left(\frac{d_{q-1}}{3}\right)$ carry bits to (from) Column $1 (q-1)$*/
   b. Column #1: Apply $\left\lfloor \frac{d_1}{3} \right\rfloor$ FA reductions $\Longrightarrow$
      $d_1 = d_1 - 2\left\lfloor \frac{d_1}{3} \right\rfloor + \left\lfloor \frac{d_0}{3} \right\rfloor + \left\lfloor \frac{d_{q-1}}{3} \right\rfloor$;
      /*Sending (Receiving) $\left\lfloor \frac{d_1}{3} \right\rfloor$ $\left(\left\lfloor \frac{d_0}{3} \right\rfloor + \left\lfloor \frac{d_{q-1}}{3} \right\rfloor\right)$ carry bits to (from) Column 2 (0 and $q-1$)*/
   c. Columns $1 < i < q$:
      For $i = 2$ to $q-1$ do Apply $\left\lfloor \frac{d_i}{3} \right\rfloor$ FA reductions $\Longrightarrow d_i = d_i - 2\left\lfloor \frac{d_i}{3} \right\rfloor + \left\lfloor \frac{d_{i-1}}{3} \right\rfloor$; /*Sending (Receiving) $\left\lfloor \frac{d_i}{3} \right\rfloor$ $\left(\frac{d_{i-1}}{3}\right)$ carry bits to (from) Column $i+1$ $(i-1)$*/
      End For
   End Do while.■

C. Proof of Convergence of Algorithm 1

The number of reduction levels $\mathcal{L}(q)$, for a full $q \times q$ bit MPP matrix (e.g., Fig. 1, for $q = 4$), must satisfy the following.

$$2\left(\frac{3}{2}\right)^{\mathcal{L}(q)} \approx q \Longrightarrow \mathcal{L}(q)\log\frac{3}{2} \approx \log\frac{q}{2} \Longrightarrow$$

$$\mathcal{L}(q) \approx \left\lceil \frac{\log\frac{q}{2}}{\log\frac{3}{2}} \right\rceil \approx \left\lceil 1.7\log\frac{q}{2} \right\rceil$$

Let $p = \lfloor \log q \rfloor \Longrightarrow q = 2^p\gamma (1 \le \gamma < 2)$, which leads to (3), since $\mathcal{L}(q) \approx \left\lceil 1.7\log\frac{2^p\gamma}{2} \right\rceil \approx \lceil 1.7(p - 1 + \log\gamma) \rceil$.

$$\lceil 1.7(p-1) \rceil \le \mathcal{L}(q) \le \lceil 1.7p \rceil, p = \lfloor \log q \rfloor \quad (3)$$

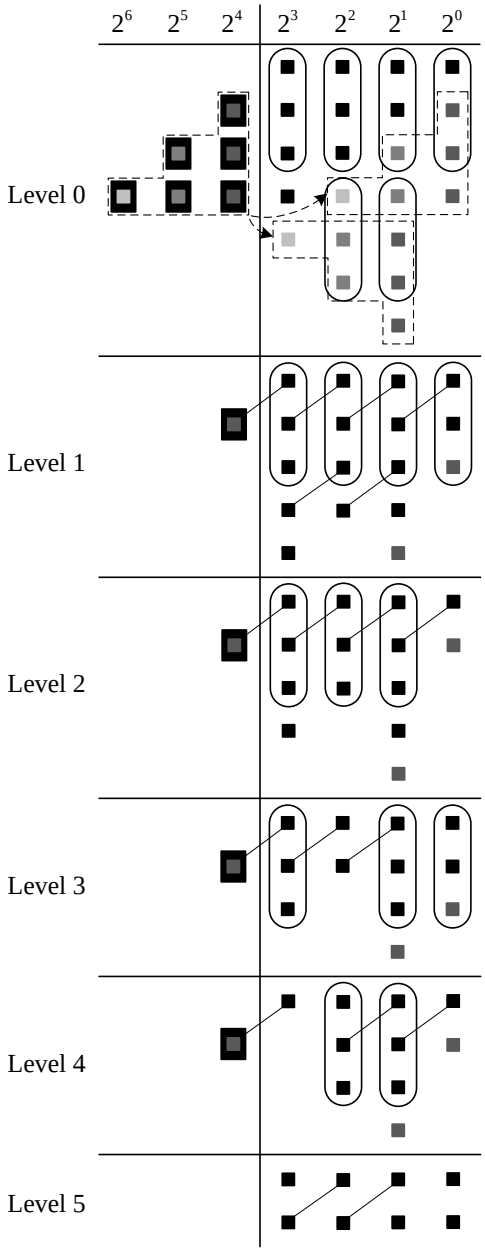

Fig. 4 Modulo-13 fully modular reduction in five levels with 18 full adders

To figure out the number of reduction levels for modulo-$(2^q - 3)$ multiplication, note that the maximum depth, in all reduction levels, occurs for Column 1. The reason is that it is originally maximum (i.e., $2q - 1$, as there are 7 bits in Level 0 of Fig. 3, for $q = 4$) and is the only column that can receive carry bits from two other columns within each level of reduction; namely the right- and left-most columns, with the original depth of q, and $q + 1$, respectively. This is approximately the same as the hypothetical case that Column 1 receives carry bits only from a single $(2q + 1)$-deep column on its right. Therefore, the number of reduction levels L_1, for Column 1, can be approximated with $\mathcal{L}(2q)$, based on (3) (i.e., for a full $2q \times 2q$ bit MPP matrix), which leads to (4). The case of $q = 10$ is further illustrated in the Appendix 3. Therefore, Algorithm 1 converges in $\lceil 1.7 \log q \rceil$ levels of reduction, with the delay of $4\lceil 1.7 \log q \rceil \Delta G$, where the total delay roughly amounts to $(5 + 9 \log q)\Delta G$.

$$\lceil 1.7p \rceil \leq L_1 \approx \mathcal{L}(2q) \approx \lceil 1.7 \log q \rceil \leq \lceil 1.7(p+1) \rceil \qquad (4)$$

D. MPPR Delay Analysis

Delay of the proposed modulo-$(2^q - 3)$ PPR is only 2FA-delay more than that of modulo-$(2^q - 1)$, since $\mathcal{L}(2q) \geq \mathcal{L}(q) + 2$ via comparing (3) and (4). For example, for the common cases of $q = 2^p$ (i.e., $\gamma = 1$), $\mathcal{L}(q) \approx 2,4,6,7,10$, and $L_1 \approx \lceil 1.7 \times 2 \rceil = 4$, $\lceil 1.7 \times 3 \rceil \approx 6$, $\lceil 1.7 \times 4 \rceil \approx 7$, $\lceil 1.7 \times 5 \rceil \approx 9$, $\lceil 1.7 \times 6 \rceil \approx 11$ for $q = 4,8,16,32,64$, respectively.

On the other hand, given the approximate nature of the latter evaluation, the actual L_1 values may be slightly different. As a matter of fact, the exact results, as are produced by the aforementioned software program, are $L_1 = 4,6,8,10,11$ in contrast to the above approximate results $L_1 = 4,6,7,9,11$.

Nevertheless, for other q-values (i.e., $1 < \gamma < 2$), lower and upper bounds can be figured out from (4). For example, the case of $q = 9 = 2^3 \times \frac{9}{8}$ (i.e., $\gamma = \frac{9}{8}$) leads to $\lceil 1.7 \times 3 \rceil \leq L_1 \leq \lceil 1.7 \times 4 \rceil \Longrightarrow 6 \leq L_1 \leq 7$, which is in accordance with the actual number of reduction levels $L_1 = 6$, as is derived by the aforementioned program.

IV. Evaluations and Comparisons

As was mentioned in Section II, the previous works in [9] and [10] do not address the problem of excess-modulo operands of the final modulo-$(2^q - 3)$ adder, which requires another round of carry-propagate addition, unless it is somehow avoided, as is the case in the work of [11] that refers to [12], for implementing the utilized compound modulo-$(2^q - 3)$ adder. However, the nontrivial details of the 4-way compound structure is not provided therein.

The proposed solution also uses the modulo-$(2^q - 3)$ adder of [12] but as the only carry-propagate adder in the whole design, where the overall delay (see Fig. 2.d and Section III.C) roughly amounts to $(5 + 9 \log q)\Delta G$. Recalling the optimistic evaluations of the three previous works (see Section II.B), no optimism is practiced in the evaluation of the current work.

Recalling the PPR of Fig. 4, the number of utilized FAs amounts to 18, for $q = 4$. This figure for $q \in \{8,16,32,64,128\}$ is $\{84,360,1488,6048,24384\}$, respectively. However, the three reference designs require less FAs (e.g., about equivalent of 10, 7, and 8 FAs, for $q = 4$), while are quite more costly on the other components.

For circuit synthesis, we have described the home design and those of [9] and [10], for $q \in \{4, 8, 16\}$, with Verilog code, which is used for correctness tests and simulation via Synopsis Design Vision, with TSMC 90nm CMOS technology. The same experience was not possible for the work of [11], due to lack of sufficient information, and that no such results are provided for in that work. Therefore, we present Fig. 5 to show the relative delay measures for the four designs based on analytical gate level evaluations (see Section II.B), where the proposed design offers the fastest PPR.

The experimental results are compiled in Table I, where the delay and energy measures of the proposed fully modular approach is superior to those of previous works, as are reflected by bolded ratio measures.

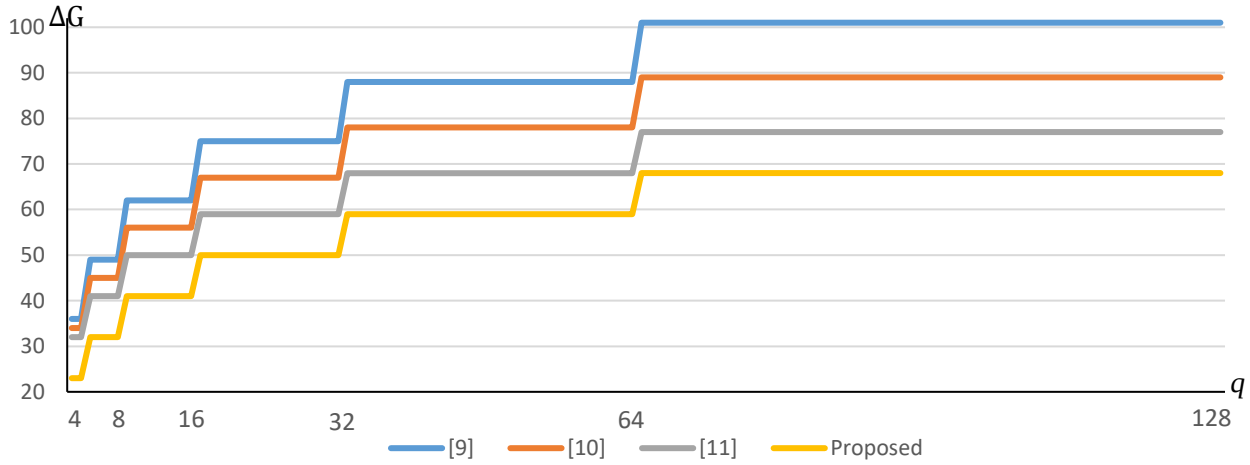

Fig. 5 Analytical gate level delay evaluations (Optimistic evaluations for the three reference works due to the lack of implementation details)

V. CONCLUSIONS

Modulo-$(2^q - 1)$ parallel multipliers that are realized via fully MPP generation and reduction, are known to be the most efficient of their kind. In particular the number of reduction levels is the same as in the case of non-modular multiplication. Similar approach on the design of modulo-$(2^q - 3)$ multipliers has not been practiced before. In fact exact mimicking of the greedy approach of using full and half adders, as in the commonly practiced Wallace-tree reduction, does not converge the reduction process to two accumulated partial products for the final carry-propagate modular addition. That is probably why instead of fully modular reduction, the residue generation of non-modular products has been the trend. However, the incompatible speed with that of modulo-$(2^q - 1)$ fully modular multipliers reduces the performance of RNS applications that use both moduli in the working moduli set. Therefore, by getting around the aforementioned convergence problem, we designed and implemented the first modulo-$(2^q - 3)$ multiplier with fully modular PPG and PPR. This fairly speed-compatible design with its modulo-$(2^q - 1)$ counterpart yields the modular carry-save product within at most two more FAs in the critical delay path. Notably important is the use of the latest parallel prefix modulo-$(2^q - 3)$ adder, in the final stage of multiplier that is also speed-compatible with the modulo-$(2^q - 1)$ case. In particular, handling the possible excess-modulo $2^q - 3$ operands with no delay overhead leads to more speed-balance [12]. In this work, at least 24% less delay, and 10% less PDP is gained in comparison to the figures of merit of the previous designs, at the cost of at most 20% area/power consumptions.

As for the future relevant work, applying the similar fully modular approach for modulo-$(2^q + 3)$ and modulo-$(2^q - 2^j - 1)$ is in order, with the sound expectation of extending the working moduli set with more balanced moduli.

ACKNOWLEDGMENT

This research was supported by Brain Pool program funded by the Ministry of Science and ICT through the National Research Foundation of Korea (NRF-2022H1D3A2A01062978 and 2021H1D3A2A02040040) and in part supported by Basic Science Research Program funded by the Ministry of Education through the National Research Foundation of Korea (NRF-2020R1I1A3063857).

REFERENCES

[1] Rooju Chokshi, Krzysztof S. Berezowski, Aviral Shrivastava, and Stanislaw J. Piestrak, "Exploiting residue number system for power-efficient digital signal processing in embedded processors," *in Proceedings of the 2009 international conference on Compilers, architecture, and synthesis for embedded systems (CASES '09)*, New York, NY, USA, 19–28. https://doi.org/10.1145/1629395.1629401.

[2] D. Younes and P. Steffan, "Efficient image processing application using residue number system," *in Proceedings of the 20th International Conference Mixed Design of Integrated Circuits and Systems* - MIXDES 2013, Gdynia, Poland, 2013, pp. 468-472.

[3] N. Samimi, M. Kamal, A. Afzali-Kusha and M. Pedram, "Res-DNN: A Residue Number System-Based DNN Accelerator Unit," *in IEEE Transactions on Circuits and Systems I: Regular Papers*, Vol. 67, No. 2, pp. 658-671, Feb. 2020, doi: 10.1109/TCSI.2019.2951083.

[4] M.V. Valueva, N.N. Nagornov, P.A. Lyakhov, G.V. Valuev, N.I. Chervyakov, "Application of the residue number system to reduce hardware costs of the convolutional neural network implementation," *Mathematics and Computers in Simulation*, Vol. 177, pp. 232-243, 2020. https://doi.org/10.1016/j.matcom.2020.04.031.

TABLE I. EXPERIMENTAL RESULTS VIA CIRCUIT SIMULATIONS

	Area		Delay		Power		PDP	
	μm^2	Ratio	ns	Ratio	mW	Ratio	pj	Ratio
	$q = 4$							
Home	36314	**1**	4.34	**1**	0.73	**1**	3.19	**1**
[9]	37449	1.03	6.10	1.41	0.80	1.09	4.89	1.53
[10]	39349	1.08	6.41	1.48	0.86	1.17	5.52	1.73
	$q = 8$							
Home	158935	1	5.41	**1**	4.50	1	24.36	**1**
[9]	132620	**0.83**	8.92	1.65	3.90	**0.87**	34.83	1.43
[10]	137372	0.86	6.70	1.24	4.11	0.91	27.58	1.13
	$q = 16$							
Home	661472	1	6.32	**1**	22.80	1	144.11	**1**
[9]	530077	**0.80**	13.83	2.19	18.03	**0.79**	249.39	1.73
[10]	545047	0.82	8.40	1.33	18.88	0.83	158.60	1.10

[5] Schinianakis, D., Stouraitis, T. " Residue Number Systems in Cryptography: Design, Challenges, Robustness," *Secure System Design and Trustable Computing. Springer*, https://doi.org/10.1007/978-3-319-14971-4_4

[6] L. Kalampoukas, D. Nikolos, C. Efstathiou, H. T. Vergos, and J. Kalamatianos, "High-speed parallel-prefix modulo 2^n-1 adders," *in IEEE Transactions on Computers*, vol. 49, no. 7, pp. 673–680, Jul. 2000.

[7] C. Efstathiou, H. T. Vergos and D. Nikolos, "Fast parallel-prefix modulo 2/sup n/+1 adders," *in IEEE Transactions on Computers*, vol. 53, no. 9, pp. 1211-1216, Sept. 2004, doi: 10.1109/TC.2004.60.

[8] C. Efstathiou, H. T. Vergos, G. Dimitrakopoulos and D. Nikolos, "Efficient diminished-1 modulo 2^n+1 multipliers," *in IEEE Transactions on Computers*, vol. 54, no. 4, pp. 491-496, April 2005, doi: 10.1109/TC.2005.63.

[9] P. M. Matutino, R. Chaves and L. Sousa, "Arithmetic Units for RNS Moduli $\{2^n-3\}$ and $\{2^n+3\}$ Operations," *in Proceedings of the* 13th *Euromicro Conference on Digital System Design: Architectures, Methods and Tools*, Lille, France, 2010, pp. 243-246, doi: 10.1109/DSD.2010.77.

[10] H. Ahmadifar and G. Jaberipur, "Improved modulo- $(2^n \pm 3)$ multipliers," *in Proceedings of the 17th CSI International Symposium on Computer Architecture & Digital Systems (CADS 2013)*, Tehran, Iran, 2013, pp. 31-35, doi: 10.1109/CADS.2013.6714234.

[11] P. -M. Seidel, "High-Performance Multiplication Modulo 2^n-3," *52nd in Proceedings of the Asilomar Conference on Signals, Systems, and Computers*, Pacific Grove, CA, USA, 2018, pp. 130-134, doi: 10.1109/ACSSC.2018.8645523.

[12] G. Jaberipur and S. H. F. Langroudi, "$(4+2log\ n)\Delta G$ Parallel Prefix Modulo- (2^n-3) Adder via Double Representation of Residues in $[0,2]$," *in IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 62, no. 6, pp. 583-587, June 2015, doi: 10.1109/TCSII.2015.2407772.

[13] S. Ma, .l.H. Hu, C.H. Wang, "A Novel Modulo-(2^n-2^k-1) Adder for Residue Number System," *in IEEE Transactions on Circuits and Systems I*, vo1.60, no.ll, pp. 2962-2972, Nov. 2013.

[14] S. H. F. Langroudi and G. Jaberipur, "Modulo-(2^n-2^q-1) Parallel Prefix Addition via Excess-Modulo Encoding of Residues," *in Proceedings of the 22nd Symposium on Computer Arithmetic*, Lyon, France, 2015, pp. 121-128, doi: 10.1109/ARITH.2015.9.

APPENDIX 1 (HOME DESCRIPTION OF THE MAIN IDEA OF SEIDEL'S WORK [11])

Fig. A1-1 provides for partial product matrices for $A \times B$ and $A \times 2B$, where A, $B \in [0,12]$ are modulo-(2^q-3) residues, for $q=4$. The symbols $\triangleright$, $\triangleleft$, and $\sqcup$, represent the value of the corresponding 10-, 6-, and 4-bit collections. In addition, $\triangle$ and $\triangle_3$ denote the collective values of 2^q-weigthed carries that are generated during partial product reduction for $A \times B$ and $A \times 3B$ (i.e., $A \times B$ and $A \times 2B$, ensemble), respectively. Recalling that $A \times B = 2^q P_h + p_l$, it is easy to see that:

$$P_h = \triangleleft + \triangle, A \times 3B = 3 \triangleleft + \sqcup + \triangle_3 \qquad \text{(A1)}$$

Similarly, Fig. A1-2 represents the partial product MPP matrix for $A \times (3B)$, where B_1, B_2, and B_3 denote the carry-save digits of $3B$, and $\triangleleft_3$ corresponds to the collective value of the shaded 10-bit collection that is indeed the output of the most significant half multiplier for $A \times 3B$.

Therefore, the two expressions representing the most significant half of $A \times 3B$, in Figs. A1 and A2 yield the same value, which leads to (A2), and (A3) for implementation, where $U+V$ represents the carry-save result of $\triangleleft_3 + P_l$.

$$\triangleleft_3 + \triangle_3 = 3 \triangleleft + \sqcup + \triangle_3 \Longrightarrow 3 \triangleleft = \triangleleft_3 - \sqcup \Longrightarrow 3P_h = 3 \triangleleft + 3 \triangle = \triangleleft_3 + 3 \triangle - \sqcup \qquad \text{(A2)}$$

$$|A \times B|_{2^{q-3}} = |3P_h + P_l|_{2^{q-3}} = |\triangleleft_3 + 3 \triangle - \sqcup + P_l|_{2^{q-3}} = |U + V + 3 \triangle - \sqcup|_{2^{q-3}} \qquad \text{(A3)}$$

Finally, the main idea of [11] is illustrated by Fig. A1-3, where d denotes the actual bit-depth of each column, which depends on the number of carry-save digit-products. Therefore, the maximum depth is $2(q-1)+2 = 2q$, where recalling (3), the number of reduction levels, without the end-around carry reentrance is $\lceil 1.7p \rceil \leq \mathcal{L}(2q) \leq \lceil 1.7(p+1) \rceil$. Note that $\triangleleft_3 + P_l$ is obtained via joint partial product reduction of the bits of $A \times B$ least significant half multiplier and $A \times 3B$ most significant half multiplier. This result must undergo a correction by $3 \triangle - \sqcup$, where $\triangle$ ($\sqcup$) is obtained via a q-bit (q-element) carry generation network (counter).

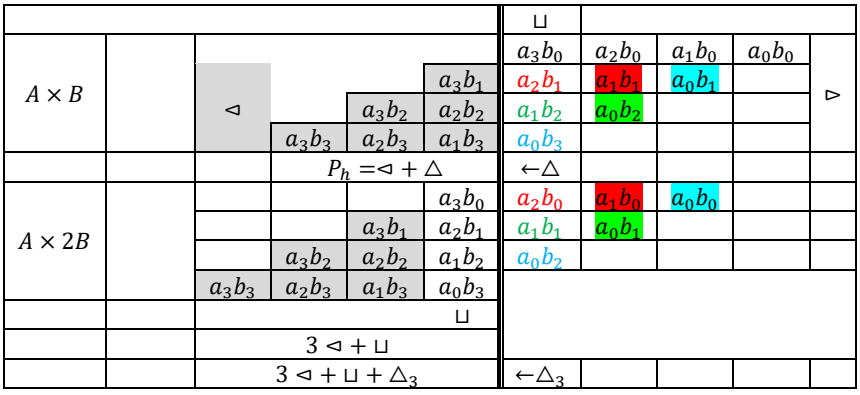

Fig. A1-1 Partial product MPP matrix for $A \times B$, $A \times 2B$, and collectively $A \times 3B$

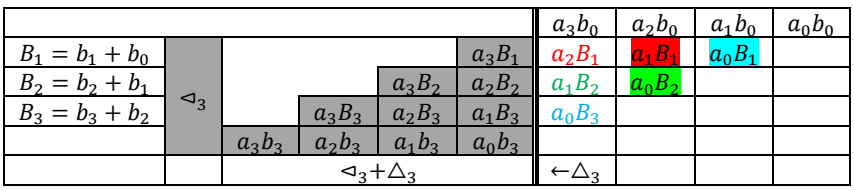

Fig. A1-2 Partial product MPP matrix for $A \times (3B)$

However, the Booth recoding is not applied, due to its trivial impact on the overall delay, since the delay of recoding is equal to the delay saving via one less reduction level. Therefore, the overall delay consists of those of PPG, PPR, △, (4; 2) compressor, and final modulo-$(2^q - 3)$ adder.

d	5	6	7	8
⊔	a_3b_0	a_2b_0	a_1b_0	a_0b_0
	a_2b_1	a_1b_1	a_0b_1	a_3B_1
	a_1b_2	a_0b_2	a_3B_2	a_2B_2
	a_0b_3	a_3B_3	a_2B_3	a_1B_3
	a_3b_3	a_2b_3	a_1b_3	a_0b_3
	$\lhd_3$			

Fig. A1-3 Joint partial product reduction

APPENDIX 2 (OUTPUT OF THE HOME SOFTWARE FOR $q = 10$)

Fig. A2 depicts the output generated by the in-house software, for $q = 10$, where each dot represents a product bit The top dot matrix represents the original MPP, where $L_1 = 19$, and integer list, on the right, regards the number of FAs used for the product bits of each column, from left to right, respectively. The next seven reduced MPPs can be explained likewise. The total FA counts in Levels 0, 1, 2, 3, 4, 5, and 6 amount to 45, 31, 22, 15, 9, 9, and 4, , respectively.

APPENDIX 3 (CLARIFICATION OF (4), FOR $q = 10$)

Table A3 presents the reduction levels of Fig. A2 in terms of the corresponding depths. Note that the leftmost column (i.e., $q - 1$) has been moved around to the right of Column 0, for better illustration of the end-around carry movements, where the carry emitting columns are indicated by ← and the receiving ones by ↓, with the same highlight color. The number of emitted and received carries are also denoted by the same font color.

Using (4), in case of $q = 10$ (i.e., $p = \lfloor \log q \rfloor = 3$), as follows, gives an estimation of 6-7 reduction levels that is confirmed by the actual seven reduction levels shown in Table A3.

$$\lceil 1.7 \times 3 \rceil \le \mathcal{L}(20) \le \lceil 1.7 \times 4 \rceil \Longrightarrow$$
$$\lceil 5.1 \rceil \le \mathcal{L}(20) \le \lceil 6.8 \rceil \Longrightarrow 6 \le \mathcal{L}(20) \le 7$$

Table A3 Alternative illustration of Fig. A2

L^*	$q-2$	$q-3$	...	2	1	0	$q-1^{**}$
***	$q+2$	$q+3$		$2q-2$	$2q-1$	q	$q+1$
	← ↓	←		← ↓	← ↓ ↓	← ↓	← ↓
0	12	13		18	19	10	11
1	8	9		12	$6+1+3$ $+3=13$	$3+1$ $+3=7$	$3+2$ $+4=9$
2	7	6		8	$4+1+3$ $+2=10$	$2+1$ $+3=6$	$3+2$ $=5$
3	5	4		7	$3+1+1$ $+2=7$	$2+1$ $=3$	$1+2$ $+2=5$
4	4	3		5	$2+1+1$ $+1=5$	$1+1$ $=2$	$1+2$ $+1=4$
5	3	2		4	$1+2$ $+1=4$	$2+1$	$1+1$ $+1=3$
6	2	2		3	$1+1$ $+1=3$	1	2
7	2	2		2	1	1	2

* Level #,
** displaced from the left most position, for better illustration of spill over carries to Columns 0 and 1
*** Original depth

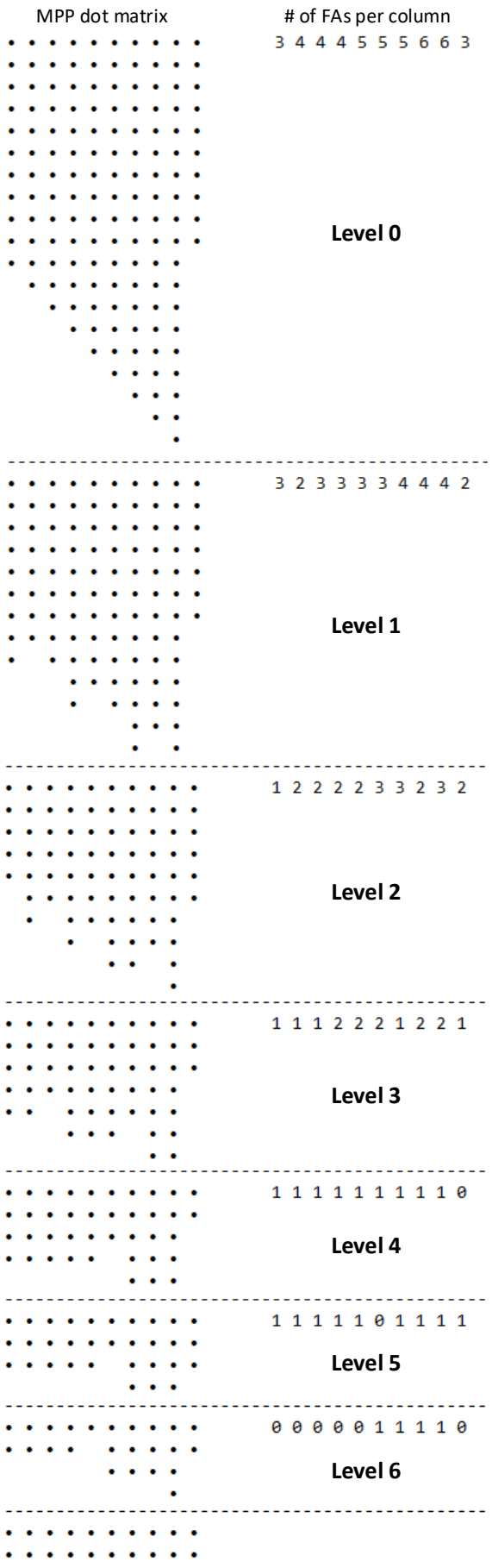

Fig. A2 Seven reduction levels for $q = 10$, with the number of FAs indicated for each column and Level.