Majority Logic Formulations for Parallel Adder Designs at Reduced Delay and Circuit Complexity

Vikramkumar Pudi, K. Sridharan, *Senior Member, IEEE*, and Fabrizio Lombardi, *Fellow, IEEE*

Abstract—The design of high-performance adders has experienced a renewed interest in the last few years; among high performance schemes, parallel prefix adders constitute an important class. They require a logarithmic number of stages and are typically realized using AND-OR logic; moreover with the emergence of new device technologies based on majority logic, new and improved adder designs are possible. However, the best existing majority gate-based prefix adder incurs a delay of $2\log_2(n)-1$ (due to the nth carry); this is only marginally better than a design using only AND-OR gates (the latter design has a $2\log_2(n)+1$ gate delay). This paper initially shows that this delay is caused by the output carry equation in majority gate-based adders that is still largely defined in terms of AND-OR gates. In this paper, two new majority gate-based recursive techniques are proposed. The first technique relies on a novel formulation of the majority gate-based equations in the used *group generate* and *group propagate* hardware; this results in a new definition for the output carry, thus reducing the delay. The second contribution of this manuscript utilizes recursive properties of majority gates (through a novel operator) to reduce the circuit complexity of prefix adder designs. Overall, the proposed techniques result in the calculation of the output carry of an n-bit adder with only a majority gate delay of $\log_2(n)+1$. This leads to a reduction of 40percent in delay and 30percent in circuit complexity (in terms of the number of majority gates) for multi-bit addition in comparison to the best existing designs found in the technical literature.

Index Terms—Adder, carry, majority voting logic, emerging technologies, arithmetic complexity

1 Introduction

The design of high performance multi-bit adders has been an active research topic for many years; schemes have been developed with the predominant goal of reducing the worst-case delay under a possible CMOS implementation. Existing multi-bit adders reduce the worst case delay based on strategies such as (i) unrolling the carry recurrence, and (ii) calculating the results prior to each possible carry input. Based on these strategies, several high performance designs have been proposed in the literature [1]. Among them, *prefix adders* constitute an important class, because they yield high-performance at a relatively small fan-out and hardware requirements [2], [3], [4]. Prefix adders have been extensively used specially on critical paths [5] due to a compact yet fast implementation. Traditional prefix addition (as well as other schemes, such as carry look ahead) are based on *generate* and *propagate* signals derived using AND-OR logic [1]. An n-bit prefix adder requires only $O(\log_2 n)$ stages for the calculation of the delay. In terms of AND-OR gates, the Kogge-Stone and Ladner-Fischer adders incur a delay of $2\log_2 n+1$ gates.

Alternatives to AND-OR logic have also been considered for arithmetic circuit designs. In particular, majority logic has been of interest from the early 1960s. The realization of a one-bit adder using three majority gates and two inverters has been proposed as well as techniques based on decomposition and rearrangement to majority element-based synthesis of networks with limited fan-in components [6]. [7] has presented a *geometric* method that utilizes Veitch diagrams for synthesis using i-input majority gates for a variety of n-argument switching functions. An approach based on *Logically Passive Self-Dual (LPSD)* has been presented in [8]; an extension to this work has been presented in [9]. Interest in majority logic has been revived recently in the context of digital design for several emerging nanotechnologies (such as domain wall nanomagnets [10], resonant tunneling diodes [11] and quantum dot cellular automata (QCA) [12]) [13], [14]. A few multi-bit adder designs in QCA have been reported [15], [16], [17], [18], [19]. However, the best existing majority gate-based n-bit adder design still incurs a delay of $2\log_2(n)-1$ for the nth carry, so only slightly better than using merely AND-OR gates. A close examination of this design reveals the cause for this limitation, i.e., the AND-OR logic has been primarily used to derive the majority gate-based designs.

The goal of this paper is to first redefine the output carry of an n-bit adder in terms of *only* majority gates for delay reduction; the proposed formulations are useful for parallel adders not for synthesizing general majority gate circuits [13]. This paper provides two contributions to adder design using majority logic by which new majority gate-based recursive techniques are proposed. The first contribution is based on a new definition for the majority gate equations of the *group generate* and *group propagate* signals, such that the output carry is generated at a reduced delay. The second contribution is based on a new operator that exploits novel recursive properties of majority logic to achieve saving in circuit complexity (as given by the number of majority gates required in a design) for prefix adders. Overall, the proposed formulation results in calculating the output carry of an n-bit adder with a reduced majority gate delay of $\log_2(n)+1$. Moreover, the proposed approach leads to a reduction of 40percent in normalized delay and 30 percent in circuit complexity (in terms of required majority gates) for multi-bit addition compared to the best existing designs found in the technical literature.

2 Adder and Output Carry

Fig. 1 shows the block diagram of an n-bit binary adder. The inputs to the adder are given by $A = a_{n-1}\ a_{n-2}\dots a_1 a_0$, $B = b_{n-1}\ b_{n-2}\dots b_1 b_0$ and C_{in} while $Sum = S_{n-1}\ s_{n-2}\dots S_1 S_0$ and the carry C_n are the outputs. Many approaches have been proposed for computing Sum and C_n efficiently. These include the *carry lookahead adder* (CLA) and prefix adders. While CLA provides a means for fast two-operand addition, fan-out restrictions have led to the development of *prefix adders* to reduce the carry calculation to a "prefix" computation. Both CLA and the prefix adders are based on the principle of *generate* and *propagate* of two binary signals a_i and b_i: in particular, $g_i = a_i\, b_i$ while $p_i = (a_i + b_i)$. Prefix adders employ an associative operator [2] denoted by $\circ$ and defined as

$$(G_i, P_i)\circ(G_j, P_j) = \left(G_i + (P_i G_j), P_i P_j\right). \tag{1}$$

The $\circ$ operator is used to express the output carry C_n as

$$\begin{aligned}(C_n, 0) = (g_{n-1}, p_{n-1})\ \circ\ &(g_{n-2}, p_{n-2})\circ\dots\\ &\circ(g_1, p_1)\circ(g_0, p_0)\\ &\circ(C_0, 0).\end{aligned} \tag{2}$$

Efficient tree-based computation structures using (2) have been proposed [2], [3], [4]. Among these, the designs in [2] and [3] need exactly $\log_2 n$ stages for an n-bit adder. In terms of circuit components, these designs incur a delay of $2\log_2 n+1$ gates where each associative operation incurs 2 gate-delays and there are a total of

• V. Pudi is with the Hardware Embedded Systems Lab, Nanyang Technological University, Singapore 639798. E-mail: pudi@ntu.edu.sg.
• K. Sridharan is with the Department of Electrical Engineering, Indian Institute of Technology Madras, Chennai 600036, India. E-mail: sridhara@ee.iitm.ac.in.
• F. Lombardi is with Electrical and Computer Engineering, Northeastern University, Boston, MA 02115. E-mail: lombardi@ece.neu.edu.

Manuscript received 12 Jan. 2017; revised 16 Apr. 2017; accepted 17 Apr. 2017. Date of publication 23 Apr. 2017; date of current version 14 Sept. 2017.
Recommended for acceptance by V. Piuri.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TC.2017.2696524

0018-9340 © 2017 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission.
See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

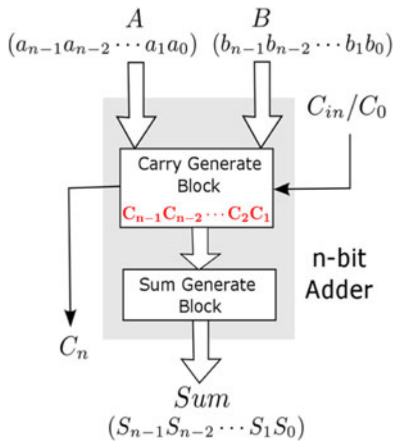

A
$(a_{n-1}a_{n-2}\cdots a_1a_0)$
B
$(b_{n-1}b_{n-2}\cdots b_1b_0)$
C_{in}/C_0
Carry Generate Block
$C_{n-1}C_{n-2}\cdots C_2C_1$
n-bit Adder
Sum Generate Block
C_n
Sum
$(S_{n-1}S_{n-2}\cdots S_1S_0)$

Fig. 1. Block diagram of n-bit binary adder.

$\log_2 n$ operations. Furthermore, initial generate-propagate calculations incur in one gate delay. An alternative definition of the output carry that leads to a lower gate delay, is presented in this manuscript.

The majority gate equations of the output carry C_{i+1} and the output sum S_i are given in (3) and (4); therefore, C_{i+1} depends on previous carry signals (such as C_i), whereas S_i depends only on the carry signals and the inputs. After generating all the carry signals, the sum signals are computed in parallel. Hence, the speed of an n-bit binary adder primarily depends on the carry generation process. The focus of this paper is on the carry generate process for low delay/circuit complexity in its design

$$C_{i+1} = M(a_i, b_i, C_i) \tag{3}$$

$$S_i = M(\overline{C_{i+1}}, M(a_i, b_i, \overline{C_{i+1}}), C_i). \tag{4}$$

The output carry C_n of an n-bit adder is usually specified in terms of Sum as

$$C_n = (Sum \geq 2^n) + (Sum = 2^n - 1)C_{in}, \tag{5}$$

where $Sum \geq 2^n$ and $Sum = 2^n - 1$ are binary signals. Upon generating the $(Sum \geq 2^n)$ and $(Sum = 2^n - 1)$ signals, computation of C_n requires just two gate delays. The logic implementation of C_n in (5) using AND-OR gates and majority gates is given in Table 1. However, recent designs in [18] and [17] do not achieve a low circuit/delay complexity because they rely on the output carry equation as in (5). The definition of the output carry *directly* in terms of majority gates is advantageous to achieve a low delay/circuit complexity for majority gate-based adders. (5) can be redefined such that the output carry is in terms of majority gates; this is given by

$$C_n = M(Sum \geq 2^n, Sum \geq (2^n - 1), C_{in}) \tag{6}$$

$$Sum \geq 2^n = C_n|_{(C_{in}=0)} \tag{7}$$

$$Sum \geq (2^n - 1) = C_n|_{(C_{in}=1)}. \tag{8}$$

(*The proof of (6) , (7), and (8) is provided in the supplemental material as Theorem 1 and Lemma 1,* which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TC.2017.2696524)

To compute the output carry in (6), one majority gate is required after computing the Sum signals. The same output carry in (5) requires two majority gates (therefore incurring in a two majority gate delay).

3 Majority Logic Formulation of Output Carry

A formulation for the carry output in terms of the Sum signal has been presented in the previous section; next the Sum signal is expressed in recursive majority logic form. This is used to efficiently generate the carry as in (6). Let $A = a_{n-1}\, a_{n-2} \ldots a_1 a_0$, $B = b_{n-1}\, b_{n-2} \ldots b_1 b_0$ and the (initial) carry C_0 be inputs to an n-bit binary adder. Then, the output carry C_n can be defined in terms of majority gates as

$$\begin{aligned} C_n = M(a_{n-1}, b_{n-1}, M(a_{n-2}, b_{n-2}, \ldots, \\ M(a_1, b_1, M(a_0, b_0, C_0))). \end{aligned} \tag{9}$$

Hence, $Sum \geq 2^n$ and $Sum \geq (2^n - 1)$ are expressed in terms of majority gates using (7) and (8) as

$$\begin{aligned} Sum \geq 2^n = M(a_{n-1}, b_{n-1}, M(a_{n-2}, b_{n-2}, \ldots, \\ M(a_1, b_1, M(a_0, b_0, 0)))) \end{aligned} \tag{10}$$

$$\begin{aligned} Sum \geq (2^n - 1) = M(a_{n-1}, b_{n-1}, M(a_{n-2}, b_{n-2}, \ldots, \\ M(a_1, b_1, M(a_0, b_0, 1)))) \end{aligned} \tag{11}$$

Direct application of (9) to compute C_n is not efficient (i.e., it incurs in a high delay); so, the recursive forms of C_n, $Sum \geq 2^n$ and $Sum \geq (2^n - 1)$ in (9), (10), and (11) are needed to derive two of the contributions of this paper. For ease of notation, let $R_i(x)$ and $R_{i:j}(x)$ be defined as

$$R_i(x) = M(a_i, b_i, x) \tag{12}$$

$$\begin{aligned} R_{i:j}(x) = M(a_i, b_i, M(a_{i-1}, b_{i-1}, \ldots, \\ M(a_{j-1}, b_{j-1}, M(a_j, b_j, x)))) \end{aligned} \tag{13}$$

$$R_{i:i}(x) = R_i(x). \tag{14}$$

$Sum \geq 2^n$ and $Sum \geq (2^n - 1)$ in (10) and (11) can be rewritten using $R_{i:j}(x)$ as

$$Sum \geq 2^n = R_{n-1:1}(a_0 \cdot b_0) \tag{15}$$

$$Sum \geq (2^n - 1) = R_{n-1:1}(a_0 + b_0). \tag{16}$$

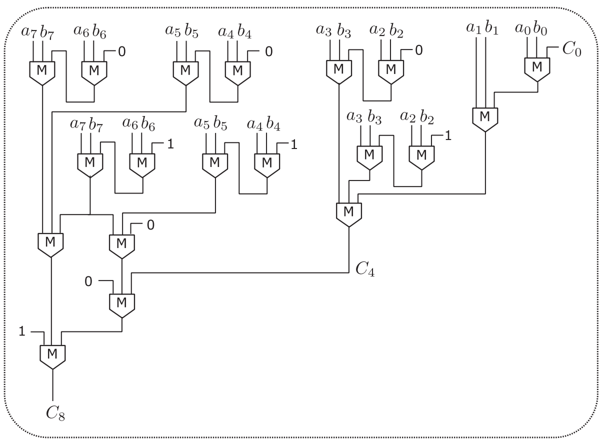
So the output carry C_n using (15) and (16) can be written as

$$C_n = M(R_{n-1:1}(a_0 \cdot b_0), R_{n-1:1}(a_0 + b_0), C_0). \tag{17}$$

(17) expresses the computation of all carries using C_0. For improvement in circuit complexity the lower order carries can be used for computation of the higher order carries. For example, when $n > i$, C_n can be expressed using C_i as

TABLE 1
Different Realizations for Carry Components in (5) Where $g_i = a_i\ b_i$, $k_i = a_i \otimes b_i$ and $p_i = (a_i + b_i)$

Realization	$Sum \geq 2^n$	$Sum = (2^n - 1)$
AND-OR Gates	$g_{n-1} + \sum_{i=n-2}^{0}(\prod_{j=n-1}^{i+1} k_j)g_i$	$\prod_{i=n-1}^{0} k_i$
Majority Gates [15]	$M(a_{n-1}, b_{n-1}, M(a_{n-2}, b_{n-2}, \ldots, M(a_1, b_1, M(a_0, b_0, 0))))$	$\prod_{i=n-1}^{0} p_i$
Majority Gates [14]	$M(a_{n-1}, b_{n-1}, M(a_{n-2}, b_{n-2}, \ldots, M(a_1, b_1, M(a_0, b_0, 0))))$	$\prod_{i=n-1,n-3,\ldots,1} M(a_i, b_i, M(a_{i-1}, b_{i-1}, 1))$
Proposed	$M(a_{n-1}, b_{n-1}, M(a_{n-2}, b_{n-2}, \ldots, M(a_1, b_1, a_0)))$	$M(a_{n-1}, b_{n-1}, M(a_{n-2}, b_{n-2}, \ldots, M(a_1, b_1, b_0)))$



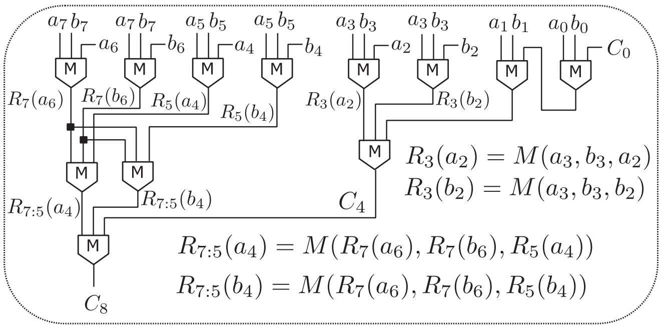

$R_3(a_2) = M(a_3, b_3, a_2)$
$R_3(b_2) = M(a_3, b_3, b_2)$
$R_{7:5}(a_4) = M(R_7(a_6), R_7(b_6), R_5(a_4))$
$R_{7:5}(b_4) = M(R_7(a_6), R_7(b_6), R_5(b_4))$
(b)

Fig. 2. Carry C_8 majority gate implementations. (a) Carry C_8 of [14]. (b) Proposed majority gate diagram of carry C_8.

$$C_n = M(a_{n-1}, b_{n-1}, M(a_{n-2}, b_{n-2}, \dots, M(a_{i+1}, b_{i+1}, M(a_i, b_i, C_i)))). \tag{18}$$

By applying (18), we can write C_8 using C_4 as

$$C_8 = M(a_7, b_7, M(a_6, b_6, M(a_5, b_5, M(a_4, b_4, C_4)))) \tag{19}$$

The expressions for C_n in (17) and (18) are equivalent; so (17) is applied for calculating C_n in (18) by replacing 0 by i. The generalized form is given as

$$C_n = M(R_{n-1:i+1}(a_i \cdot b_i), R_{n-1:i+1}(a_i + b_i), C_i) \tag{20}$$

(*The proof of* (20) *is provided in the supplemental material as Lemma* 2, available online)

C_n in (20) can be further improved; the direct calculation of the $a_i \cdot b_i$ and $a_i + b_i$ terms in (20) requires an additional AND gate and an OR gate. However an improvement for C_n requires few additional considerations. These are presented in terms of four properties. The properties are for recursive majority logic as $R_{n-1:i+1}(a_i + b_i)$, $R_{n-1:i+1}(a_i \cdot b_i)$, $R_{n-1:i+1}(a_i)$ and $R_{n-1:i+1}(b_i)$. The proofs are omitted because they follow directly from the definition of the majority function.

Property 1. $R_{n-1:i+1}(a_i \cdot b_i) = R_{n-1:i+1}(a_i) \cdot R_{n-1:i+1}(b_i)$

Property 2. $R_{n-1:i+1}(a_i + b_i) = R_{n-1:i+1}(a_i) + R_{n-1:i+1}(b_i)$

Property 3. $R_{n-1:i+1}(a_i \cdot b_i) + R_{n-1:i+1}(a_1 + b_i) = R_{n-1:i+1}(a_i + b_i)$

Property 4. $R_{n-1:i+1}(a_i \cdot b_i) \cdot R_{n-1:i+1}(a_1 + b_i) = R_{n-1:i+1}(a_i \cdot b_i)$

Let $A = a_{n-1}\, a_{n-2} \dots a_1 a_0$, $B = b_{n-1}\, b_{n-2} \dots b_1 b_0$ and the input carry C_i be the inputs to a binary adder; using the presented properties, then it is evident that

$$C_n = M(R_{n-1:i+1}(a_i), R_{n-1:i+1}(b_i), C_i) \tag{21}$$

(*The proof of* (21) *is provided in the supplemental material as Theorem* 2, available online)

Hence, C_4, C_8 and C_{16} can be rewritten as

$$C_4 = M(R_3(a_2), R_3(b_2), C_2) \tag{22}$$

$$C_8 = M(R_{7:5}(a_4), R_{7:5}(b_4), C_4) \tag{23}$$

$$C_{16} = M(R_{15:9}(a_8), R_{15:9}(b_8), C_8). \tag{24}$$

The carries in (22), (23), and (24) do not require the $a_i \cdot b_i$ and $a_i + b_i$ terms and hence, this makes possible a reduction in circuit complexity as well as delay. This formulation is not only useful for calculating the parallel carry but also for the parallel calculation of the recursive terms $R_{n-1:i+1}(a_i)$ and $R_{n-1:i+1}(b_i)$, so $R_{n-1:i+1}(a_i)$ (for $n-1 > h > i$) is given by

$$R_{n-1:i+1}(a_i) = M(R_{n-1:h+1}(a_h), R_{n-1:h+1}(b_h), R_{h-1:i+1}(a_i)). \tag{25}$$

As examples, consider the calculations of $R_{7:5}(a_4)$ and $R_{7:5}(b_4)$ in (23); these are obtained as in (26) and (27). Similarly, $R_{15:9}(a_8)$ and $R_{15:9}(b_8)$ are calculated as in (28) and (29)

$$R_{7:5}(a_4) = M(R_7(a_6), R_7(b_6), R_5(a_4)) \tag{26}$$

$$R_{7:5}(b_4) = M(R_7(a_6), R_7(b_6), R_5(b_4)) \tag{27}$$

$$R_{15:9}(a_8) = M(R_{15:13}(a_{12}), R_{15:13}(b_{12}), R_{11:9}(a_8)) \tag{28}$$

$$R_{15:9}(b_8) = M(R_{15:13}(a_{12}), R_{15:13}(b_{12}), R_{11:9}(b_8)). \tag{29}$$

Fig. 2b shows the proposed majority gate diagrams of C_8 using the proposed formulation. From Fig. 2, the proposed carry C_4 requires 2 majority gates less than C_4 of [17]. Similarly, C_8 in Fig. 2 requires six majority gates less than in [17] (shown in Fig. 2), hence a saving of 33 percent is accomplished. The delay required for generating C_8 is four $(\log_2(8)+1)$ majority gates (Fig. 2) and this corresponds to a 20 percent saving compared to the design of [17]. Similarly, the delay required for generating C_n is $\log_2(n)+1$. The proposed majority gate formulation is applicable to various adders such as the Carry-Look Ahead adder (CLA), prefix adders, carry select adder and conditional sum adders. Due to space constraints, only the application to prefix adders is presented.

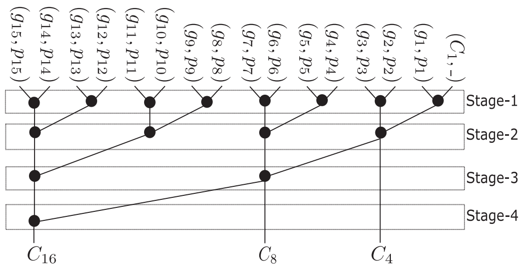
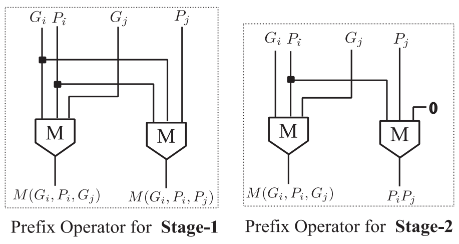


Fig. 3. Conventional prefix graph of carry C_{16}. Where $g_i = a_i b_i$ and $p_i = a_i + b_i$.

4 Prefix Operator for Adder Design

Prefix adders are defined in terms of an associative operator [1] as in (30). The carry C_4 is computed using the prefix associative operator in (31) such that *generate* $g_i = a_i\, b_i$ and *propagate* $p_i = a_i + b_i$. Fig. 3 shows its prefix graph. In this section, a *new associative operator* is presented in terms of majority logic for obtaining a design



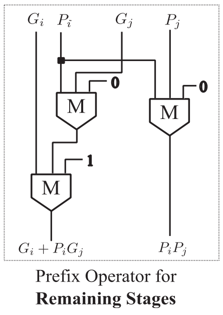


Fig. 4. Prefix-Adder associative operator of [14].

with reduced complexity (in terms of number of gates and delay in the adder design)

$$(g_i, p_i) \circ (g_j, p_j) = (g_i + p_ip_j, p_ip_j) \tag{30}$$

$$(C_4, \_) = (g_3, p_3) \circ (g_2, p_2) \circ (g_1, p_1) \circ (C_1, \_). \tag{31}$$

The motivation for the definition of this new operator is as follows. The existing operator in (30) requires three majority gates; [17] has derived new properties to reduce the number of majority gates for such existing operator. However, this can be improved further, because it reduces the number of majority gates for only specific stages of the adder. In the scheme of [17] (Fig. 4) each prefix operation requires two majority gates each for stage-1 and stage-2, while the remaining stages require three majority gates each. The proposed design requires only two majority gates in all stages of prefix addition and completely eliminates the calculations of g_is and p_is. The details are as follows. The components of C_8 in (23) can be rewritten as

$$C_8 = M(R_{7:5}(a_4), R_{7:5}(b_4), C_4) \tag{32}$$

$$(R_{7:5}(a_4), R_{7:5}(b_4)) = (M(R_7(a_6), R_7(b_6), R_5(a_4), M(R_7(a_6), R_7(b_6), R_5(b_4))) \tag{33}$$

$$(R_7(a_6), R_7(b_6)) = (M(a_7, b_7, a_6), M(a_7, b_7, b_6)) \tag{34}$$

$$(R_5(a_4), R_5(b_4)) = (M(a_5, b_5, a_4), M(a_5, b_5, b_4)). \tag{35}$$

Each pair in (32), (33), (34), and (35) requires two majority gates and all pairs require similar circuits in implementation. (32), (33), (34), and (35) utilize a new (generalized) prefix operator as well as its associative property. We now define the new prefix operator

Let a_i, b_i, a_j and b_j be the binary inputs; the proposed prefix operator is expressed in terms of majority logic as

$$(a_i, b_i) \bullet (a_j, b_j) = (M(a_i, b_i, a_j), M(a_i, b_i, b_j)). \tag{36}$$

It follows that the proposed operator is associative, i.e.,

$$[(a_3, b_3) \bullet (a_2, b_2)] \bullet (a_1, b_1) = (a_3, b_3) \bullet [(a_2, b_2) \bullet (a_1, b_1)].$$

Using the proposed associative operator, $(R_{7:5}(a_4), R_{7:5}(b_4))$, $(R_7(a_6), R_7(b_6)), (R_5(a_4), R_5(b_4))$ and $(R_3(a_2), R_3(b_2))$ can be rewritten as

$$(R_{7:5}(a_4), R_{7:5}(b_4)) = (R_7(a_6), R_7(b_6)) \bullet (R_5(a_4), R_5(b_4)) \tag{37}$$

$$(R_7(a_6), R_7(b_6)) = (a_7, b_7) \bullet (a_6, b_6) \tag{38}$$

$$(R_5(a_4), R_5(b_4)) = (a_5, b_5) \bullet (a_4, b_4) \tag{39}$$

$$(R_3(a_2), R_3(b_2)) = (a_3, b_3) \bullet (a_2, b_2). \tag{40}$$

Substituting $(R_7(a_6), R_7(b_6))$ and $(R_5(a_4), R_5(b_4))$ in $(R_{7:5}(a_4), R_{7:5}(b_4))$, $(R_{7:5}(a_4), R_{7:5}(b_4))$ is derived in a fully associative form as

$$(R_{7:5}(a_4), R_{7:5}(b_4)) = (a_7, b_7) \bullet (a_6, b_6) \bullet (a_5, b_5) \bullet (a_4, b_4) \tag{41}$$

The application of the proposed prefix-operator leads to C_8 as

$$\begin{aligned} C_8 &= M(R_{7:5}(a_4), R_{7:5}(b_4), C_4) \\ &= (R_{7:5}(a_4), R_{7:5}(b_4)) \bullet (C_4, \_) \\ &= (a_7, b_7) \bullet (a_6, b_6) \bullet (a_5, b_5) \bullet (a_4, b_4) \bullet (C_4, \_). \end{aligned}$$

Under the proposed prefix operator, the carry C_n and the recursive term $(R_{n-1:i}(a_i), R_{n-1:i}(b_i))$ are expressed as

$$(C_n, \_) = (a_{n-1}, b_{n-1}) \bullet (a_{n-1}, b_{n-1}) \bullet \cdots \bullet (a_i, b_i) \bullet (C_i, \_) \tag{42}$$

$$(R_{n-1:i}(a_i), R_{n-1:i}(b_i)) = (a_{n-1}, b_{n-1}) \bullet (a_{n-1}, b_{n-1}) \bullet \cdots \cdots \bullet (a_{i+1}, b_{i+1}) \bullet (a_i, b_i). \tag{43}$$

The recursive terms $(R_{7:5}(a_4), R_{7:5}(b_4))$ and the carry C_8 can be rewritten using the proposed prefix operator in (36). The general form of representing carries and recursive terms using the proposed prefix operator is given in (42) and (43) respectively. Fig. 5 shows the proposed prefix operator symbol and the majority gate diagram. C_{16} using (42) and (43) is given by (44). Fig. 6 shows the proposed prefix graph for calculating C_{16}. The majority gate diagram of C_{16} is shown in Fig. 7 (the proposed prefix associative operator is marked with dotted lines in Fig. 7)

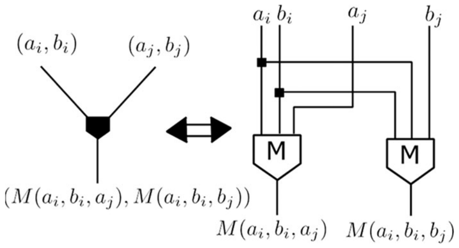


Fig. 5. Proposed prefix operator symbol and majority gate diagram.

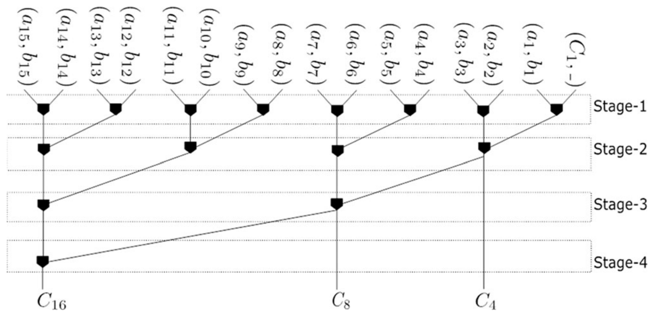


Fig. 6. Proposed prefix graph of carry C_{16}.

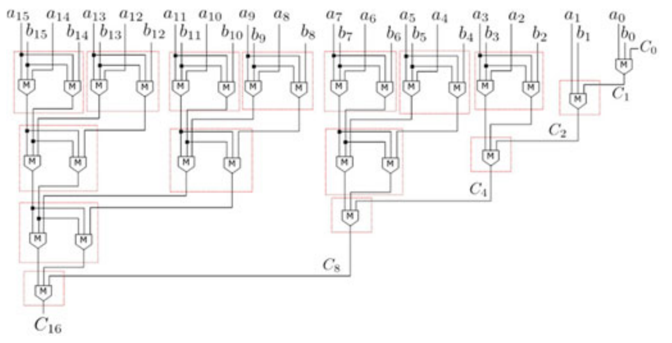

a_{15} b_{15} a_{14} b_{14} a_{13} b_{13} a_{12} b_{12} a_{11} b_{11} a_{10} b_{10} a_9 b_9 a_8 b_8 a_7 b_7 a_6 b_6 a_5 b_5 a_4 b_4 a_3 b_3 a_2 b_2 a_1 b_1 a_0 b_0
C_0
C_1
C_2
C_4
C_8
C_{16}

Fig. 7. Majority gate diagram of carry C_{16} of proposed prefix adder (C_{16} is calculated the same way for various adder schemes such as Kogge-Stone, Ladner-Fischer and Brent-Kung).

$$\begin{aligned} C_{16} &= (R_{15:9}(a_8), R_{15:9}(b_8)) \bullet (C_8, \_) \\ &= (R_{15:13}(a_{12}), R_{15:13}(b_{12})) \\ &\quad \bullet (R_{11:9}(a_8), R_{11:9}(b_8)) \bullet (C_8, \_). \end{aligned} \tag{44}$$

The proposed prefix operator has the following advantages compared to prior designs: (a) It completely removes the calculation of g_is and p_is from the prefix adders. (b) The computation of each prefix operator requires only two majority gates in all stages of prefix adders. (c) The computation of each prefix operator involves only one majority gate delay in all stages of the prefix adders.

The proposed prefix operator is directly applicable to all types of prefix adder. In this paper, three types of prefix adders are considered, namely Kogge-Stone, Ladner-Fisher and Brent-Kung adders. The proposed majority gate diagrams for various prefix adders are shown in Fig. 8. Fig. 8a shows the proposed majority gate diagram of an 8-bit Kogge-Stone prefix adder; it requires three stages and four majority gates delay for calculation of all carries, so $\log_2 n$ stages for calculation of all carries in a n-bit adder. Fig. 8b shows the proposed majority gate diagram of an 8-bit Ladner-Fischer adder; it requires three stages and four majority gates delay for calculation of all carries. An n-bit Ladner-Fischer adder requires $\log_2 n$ stages for calculation of all carries. Fig. 8c shows the proposed majority gate diagram of an 8-bit Brent-Kung adder. It requires five stages and incurs in a six majority gates delay for the calculation of all carries. An n-bit Brent-Kung adder requires $2\log_2 n - 1$ stages for the calculation of the carries. The proposed prefix graphs of Kogge-Stone, Ladner-Fischer and Brent-Kung adders are similar to the conventional prefix-graphs [2], [3], [4], the significant difference is that the proposed prefix graphs do not require the calculation of the g_is and p_is.

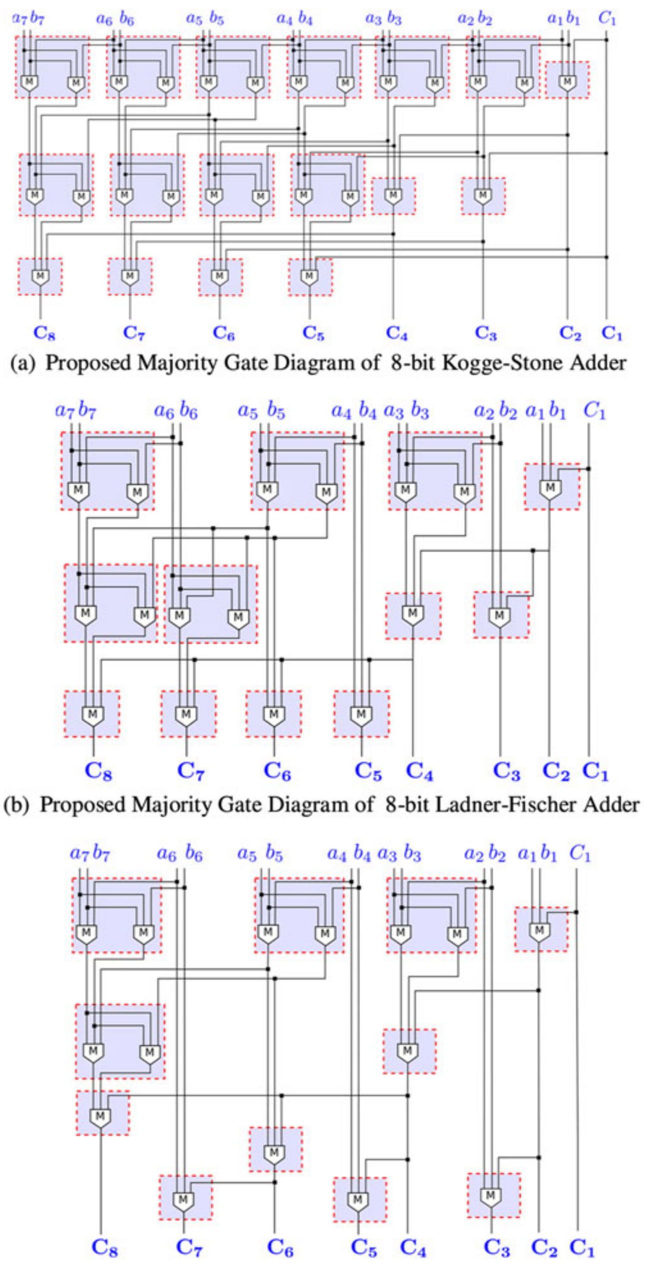

$a_7 b_7$ a_6 b_6 a_5 b_5 a_4 b_4 a_3 b_3 $a_2 b_2$ $a_1 b_1$ C_1
M
C_8 C_7 C_6 C_5 C_4 C_3 C_2 C_1
(a) Proposed Majority Gate Diagram of 8-bit Kogge-Stone Adder
(b) Proposed Majority Gate Diagram of 8-bit Ladner-Fischer Adder
(c) Proposed Majority Gate Diagram of 8-bit Brent-Kung Adder

Fig. 8. Majority gate diagrams of different 8-bit prefix adders using proposed prefix operator.

5 Complexity Analysis

In this section, generalized expressions are derived for calculating the circuit complexity (i.e., the total number of majority gates) and delay complexity (i.e., the total normalized delay also in terms of majority gates) for various n-bit prefix adders.

5.1 Circuit Complexity

Let $I_c(n)$ and $I_s(n)$ denote the number of majority gates required for calculation of all carries and sums respectively. Each proposed prefix operator consists of two majority gates, except for the final stage of calculating the carries in a prefix adder, where it requires only one majority gate (Fig. 8). In all n-bit prefix adder designs, $n-1$ prefix operators are present at the final stage when calculating the carries. So, $n-1$ majority gates must be subtracted from the total majority gates required for computing all prefix operators in a n-bit adder. Only an additional majority gate is required when calculating C_1. The expression for $I_c(n)$ is given by

$$\begin{aligned} I_c(n) &= 2 \times \begin{pmatrix} \text{Total number of} \\ \text{Prefix Operators} \end{pmatrix} + 1 - (n-1) \\ &= 2 \times \begin{pmatrix} \text{Total number of} \\ \text{Prefix Operators} \end{pmatrix} - n + 2. \end{aligned} \tag{45}$$

After calculating all carries, each sum requires two majority gates (as per (4)). The number of majority gates required for calculation of all sums in an n-bit adder is given by

$$I_s(n) = 2n. \tag{46}$$

The generalized expression for calculating the circuit complexity of a n-bit prefix adder (as the total number of majority gates) is given by

$$I(n) = 2 \times \begin{pmatrix} \text{Total number of} \\ \text{Prefix Operators} \end{pmatrix} + n + 2, \tag{47}$$

TABLE 2
Comparison of Proposed n-bit Prefix Adders with a Recent Design

Adder Type	No. of Stages	No. of Prefix-Operators	$I_{gp}(n)$	$I_c(n)$	$I_s(n)$	Total Maj. Gates $I(n)$	Normalized Delay $d(n)$
Proposed Kogge-Stone	$\log_2 n$	$n(\log_2 n-1)+1$	0	$2n\log_2 n-3n+4$	$2n$	$2n\log_2 n-n+4$	$\log_2 n+3$
Kogge-Stone [14]	$\log_2 n$	$n(\log_2 n-1)+1$	$2n-2$	$3n\log_2 n-6n+8$	$2n$	$3n\log_2 n-2n+6$	$2\log_2 n+1$
Kogge-Stone [2]	$\log_2 n$	$n(\log_2 n-1)+1$	$2n-2$	$3n\log_2 n-2n+5$	$2n$	$3n\log_2 n+2n+3$	$2\log_2 n+3$
Proposed Ladner-Fischer	$\log_2 n$	$\frac{n}{2}\log_2 n$	0	$n\log_2 n-n+2$	$2n$	$n\log_2 n+n+2$	$\log_2 n+3$
Ladner-Fischer [14]	$\log_2 n$	$\frac{n}{2}\log_2 n$	$n-2$	$\frac{3n}{2}\log_2 n-2n+2$	$2n$	$\frac{3n}{2}\log_2 n+n$	$2\log_2 n+1$
Ladner-Fischer [3]	$\log_2 n$	$\frac{n}{2}\log_2 n$	$2n-2$	$\frac{3n}{2}\log_2 n-n+2$	$2n$	$\frac{3n}{2}\log_2 n+3n$	$2\log_2 n+3$
Proposed Brent-Kung	$2\log_2 n-1$	$2n-\log_2 n-2$	0	$3n-2\log_2 n-2$	$2n$	$5n-2\log_2 n-2$	$2\log_2 n+2$
Brent-Kung [14]	$2\log_2 n-1$	$2n-\log_2 n-2$	$n-2$	$\frac{7}{2}n-3\log_2 n-2$	$2n$	$\frac{13}{2}n-3\log_2 n-4$	$4\log_2 n-3$
Brent-Kung [4]	$2\log_2 n-1$	$2n-\log_2 n-2$	$2n-2$	$5n-3\log_2 n-4$	$2n$	$9n-3\log_2 n-6$	$4\log_2 n+1$

i.e., $I(n)$ denotes the circuit complexity as the total number of majority gates required for a n-bit prefix adder as the sum of $I_c(n)$ and $I_s(n)$.

The prefix-adder circuit complexity depends on the number of prefix operators. The n-bit Kogge-Stone, the Ladner-Fischer and the Brent-Kung prefix adders require $n(\log_2 n-1)+1$, $\frac{n}{2}\log_2 n$ and $2n-\log_2 n-2$ prefix operators respectively [16]. Therefore, using (47), the total numbers of majority gates required for the n-bit Kogge-Stone, Ladner-Fischer and Brent-Kung adders are given by

$$I(n)=2n\log_2 n-n+4 \tag{48}$$

$$I(n)=n\log_2 n+n+2 \tag{49}$$

$$I(n)=5n-2\log_2 n-2. \tag{50}$$

5.2 Delay Complexity

The total normalized delay required for any binary adder is the sum of the normalized delays required for generation of all carries and sums; let $d_c(n)$ and $d_s(n)$ denote the normalized delays required for calculation of all carries and sums respectively. Each proposed prefix operator requires one-majority gate delay. The delay for calculation of all carries in prefix adders is given by the sum of the total prefix adder stages and one majority gate delay for calculating carry C_1. The delay $d_c(n)$ is given by

$$d_c(n) \;=\; (Total\ number\ of\ stages)+1. \tag{51}$$

TABLE 3
Circuit and Delay Comparison for Different Sizes of Prefix Adders

Adder Type	Size (bits)	Proposed			[14]			Saving in MNDP (%)
		I(n)	d(n)	MNDP	I(n)	d(n)	MNDP	
KS	8	44	6	264	62	7	434	39
	16	116	7	812	166	9	1494	46
	32	292	8	2,336	422	11	4,642	50
	64	708	9	6,372	1,030	13	13,390	52
	128	1,668	10	16,680	2,438	15	36,570	54
LF	8	34	6	204	44	7	308	34
	16	82	7	574	112	9	1,008	43
	32	194	8	1,552	272	11	2,992	48
	64	450	9	4,050	640	13	8,320	51
	128	1,026	10	10,260	1,472	15	22,080	54
BK	8	32	8	256	39	9	351	27
	16	70	10	700	88	13	1,144	39
	32	148	12	1,776	189	17	3,213	45
	64	306	14	4,284	394	21	8,274	48
	128	624	16	9,984	807	25	20,175	51

After computing all carries, all sums are calculated in parallel; the delay required for each sum is two majority gates (the delay for the inverters is not included). The delay $d_s(n)$ is given by

$$d_s(n) \;=\; 2. \tag{52}$$

The generalized expression for the delay complexity as the total normalized delay required for an n-bit prefix adder in terms of majority gates is given by the sum of $d_c(n)$ and $d_s(n)$, i.e.,

$$\begin{aligned} d(n) &= d_c(n)+d_s(n) \\ &= Total\,number\,of\,stages+1+2 \\ &= Total\,number\,of\,stages+3. \end{aligned} \tag{53}$$

The expressions for the prefix-adder normalized delay depends on the number of stages of an adder. The n-bit Kogg-Stone, Ladner-Fischer and Brent-Kung prefix adders require $\log_2 n$, $\log_2 n$ and $2\log_2 n-1$ prefix stages respectively. The total normalized delay (as delay complexity) required for n-bit Kogge-Stone, Ladner-Fischer and Brent-Kung adders using (53) is given in (54), (55), and (56) respectively

$$d\ (n) = \;2n\log_2 n-n+4 \tag{54}$$

$$d\ (n) = \;n\log_2 n+n+2 \tag{55}$$

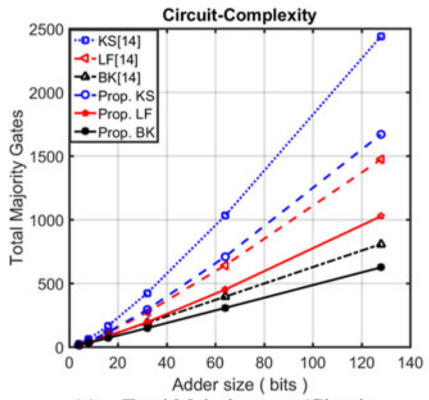
$$d\ (n) = \;5n-2\log_2 n-2. \tag{56}$$

6 Comparison

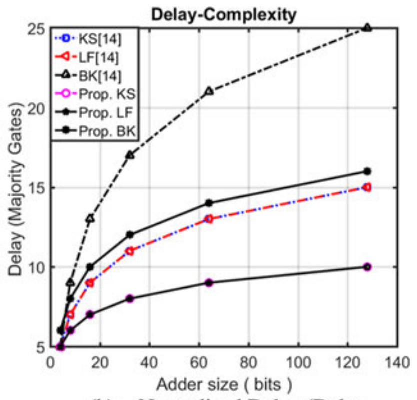
Table 2 presents a comparison of the proposed design with [17]. The proposed prefix adders achieve approximately a 40 percent reduction in delay compared to the designs of [17]. Similarly, the proposed designs achieve more than 30 percent saving in circuit complexity. These substantial savings are due to the proposed new formulations as outlined in the previous sections.

Table 3 presents the product of the circuit and delay complexities given by the number of majority gates and the Normalized Delay (in terms of gates along the critical path). This metric is referred to as MNDP. The percentage reduction in MNDP is more than 50 percent for the proposed Kogge-Stone and Ladner-Fischer adders and 48 percent for the Brent-Kung adder.

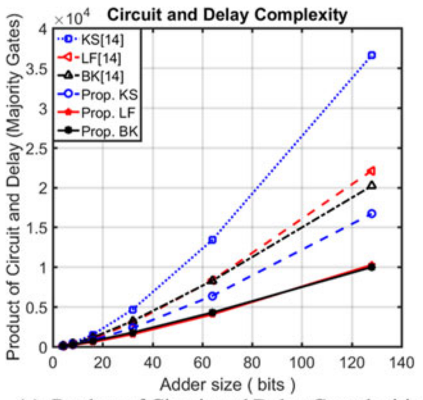
Figs. 9a, 9b, and c9 show the comparison graphs of total number of majority gates, normalized delay and the product of these two complexities (MNDP) for the Kogge-Stone, Ladner-Fischer and Brent-Kung adders for different adder sizes. The proposed Brent-Kung adder (Fig. 9a) has a low circuit complexity as reflected by the reduced number of majority gates. The delay required for the proposed Ladner-Fischer and Kogge-Stone adders is lower compared with the different designs of [17].



(a) Total Majority gates(Circuit Complexity) Comparison of Prefix Adders



(b) Normalized Delay (Delay Complexity) Comparison of Prefix Adders



(c) Product of Circuit and Delay Complexities of Prefix Adders

Fig. 9. Comparison for total number of majority gates (circuit complexity), normalized delay (delay complexity) and their product for prefix adders.

7 Conclusion

A new majority gate-based approach for high performance adder design has been presented. The two contributions of this manuscript (the formulation of the carry output and the recursive prefix operator for majority logic) have resulted in a reduction in circuit complexity (as requiring a lower number of majority gates in an adder design) as well as lower propagation delay for the leading carry.

The proposed strategy has been applied to various prefix adders including the Kogge-Stone, Ladner-Fischer and Brent-Kung adders. It is observed that these new results achieve a reduction in delay of at least $\log_2 n$ over the best existing majority gate-based adders found in the technical literature. Specifically, reductions of 40 percent in delay and 30 percent in circuit complexity (in terms of the number of majority gates) has been accomplished for multi-bit adder schemes. As shown in Table 2, $I_s(n)$ remains constant and for very large values of n, $I_c(n)$ and the normalized delay $d(n)$ show considerable reductions, for example a nearly 50 percent for $d(n)$ for the three prefix adders considered in this manuscript.

Current research deals with the applications of these findings to emerging technologies and in particular, the implications on different applications requiring fast arithmetic processing.

References

[1] I. Koren, *Computer Arithmetic Algorithms*. Natick, MA, USA: A.K. Peters Ltd., 2002.

[2] P. Kogge and H. Stone, "A parallel algorithm for the efficient solution of a general class of recurrent equations," *IEEE Trans. Comput.*, vol. C-22, no. 8, pp. 786–793, Aug. 1973.

[3] R. Ladner and M. Fischer, "Parallel prefix computation," *J. Assoc. Comput. Mach.*, vol. 27, no. 4, pp. 831–838, Oct. 1980.

[4] R. P. Brent and H. T. Kung, "A regular layout for parallel adders," *IEEE Trans. Comput.*, vol. C-31, no. 3, pp. 260–264, Mar. 1982.

[5] D. Harris and S. Harris, *Digital Design and Computer Architecture*, 2nd ed. Burlington, MA, USA: Morgan Kaufmann, 2012.

[6] F. Miyata, "Realization of arbitrary logical functions using majority elements," *IEEE Trans. Electron. Comput.*, vol. EC-12, no. 3, pp. 183–191, Jun. 1963.

[7] H. S. Miller and R. O. Winder, "Majority-logic synthesis by geometric methods," *IEEE Trans. Electron. Comput.*, vol. EC-11, no. 1, pp. 89–90, Feb. 1962.

[8] S. B. Akers, "Synthesis of combinational logic using three-input majority gates," in *Proc. 3rd Annu. Symp. Switching Circuit Theory Logical Des.*, 1962, pp. 149–185.

[9] E. Riseman, "A realization algorithm using three-input majority elements," *IEEE Trans. Electron. Comput.*, vol. EC-16, no. 4, pp. 456–462, Sep. 1967.

[10] A. Roohi, R. Zand, and R. F. DeMara, "A tunable majority gate-based full adder using current-induced domain wall nanomagnets," *IEEE Trans. Magn.*, vol. 52, no. 8, pp. 1–7, Aug. 2016.

[11] P. Mazumder, S. Kulkarni, M. Bhattacharya, J. Sun, and G. Haddad, "Digital circuit applications of resonant tunneling devices," *Proc. IEEE*, vol. 86, no. 4, pp. 664–686, Aug. 1998.

[12] P. Tougaw and C. Lent, "Logical devices implemented using quantum cellular automata," *J. Appl. Phys.*, vol. 75, no. 3, pp. 1818–1825, 1994.

[13] R. Zhang, P. Gupta, and N. K. Jha, "Majority and minority network synthesis with application to QCA-, SET-, and TPL-based nanotechnologies," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 26, no. 7, pp. 1233–1245, Jul. 2007.

[14] L. Amaru, P. E. Gaillardon, A. Chattopadhyay, and G. De Micheli, "A sound and complete axiomatization of majority- n logic," *IEEE Trans. Comput.*, vol. 65, no. 9, pp. 2889–2895, Sep. 2016.

[15] H. Cho and E. Swartzlander, "Adder and multiplier designs in quantum-dot cellular automata," *IEEE Trans. Comput.*, vol. 58, no. 6, pp. 721–727, Jun. 2009.

[16] V. Pudi and K. Sridharan, "Low complexity design of ripple carry and brent-kung adders in QCA," *IEEE Trans. Nanotechnol.*, vol. 11, no. 1, pp. 105–119, Jan. 2012.

[17] S. Perri and P. Corsonello, "New methodology for the design of efficient binary addition circuits in QCA," *IEEE Trans. Nanotechnol.*, vol. 11, no. 6, pp. 1192–1200, Nov. 2012.

[18] V. Pudi and K. Sridharan, "New decomposition theorems on majority logic for low-delay adder designs in quantum dot cellular automata," *IEEE Trans. Circuits Syst. II: Express Briefs*, vol. 59, no. 10, pp. 678–682, Oct. 2012.

[19] S. Perri, P. Corsonello, and G. Cocorullo, "Area-delay efficient binary adders in QCA," *IEEE Trans. Very Large Scale Integr. VLSI Syst.*, vol. 22, no. 5, pp. 1174–1179, May 2014.

[20] R. Lindaman, "A theorem for deriving majority-logic networks within an augmented Boolean algebra," *IEEE Trans. Electron. Comput.*, vol. EC-9, no. 3, pp. 338–342, Sep. 1960.