Synthesis of Asymptotically Optimal Adders for Multiple-Valued Logic

Philipp Niemann*† Rolf Drechsler*†
*Cyber-Physical Systems, DFKI GmbH, Bremen, Germany
†Department of Computer Science, University of Bremen, Bremen, Germany
philipp.niemann@dfki.de, drechsler@uni-bremen.de

***Abstract*—Addition is the most basic arithmetic operation and efficient adder realizations are crucial to performing arithmetic operations in an efficient way. Synthesis of efficient adders has been studied exhaustively for two-valued, Boolean logic and asymptotically optimal constructions have been derived w.r.t. time and space complexity. In contrast, for multiple-valued logic typically simple, linear-time adder structures like Ripple Carry Adders are employed and there have only been few case studies on more efficient adders for small radices. In this paper, we provide generic constructions of efficient adders with asymptotically optimal time (and space) complexity that can be used for arbitrary radices.**

I. Introduction

Boolean logic and binary number representations are being used in essentially all modern, CMOS-based processors, since it is significantly easier if there are only two different logic values to be represented, e.g. in terms of different voltage levels. Performing binary arithmetic operations like addition and multiplication in an efficient way is crucial for these systems and there has been a large body of research on efficient binary addition and multiplication circuits [1]–[5].

However, in the 1990s and 2000s there have also been proposals to operate standard MOS technology in different ways using four different logic values, i.e. realizing a quaternary logic, mainly as to reduce the number of interconnects [6]–[9]. In this context, there have been proposals for efficient quaternary adders which make use of the fact that the addition of two quaternary digits can be interpreted as the addition of two two-bit binary numbers [8], [9]. This interpretation allows for the rather straightforward generalization of the constructions for the binary case—as long as the input and intermediate carries are restricted to the values 0 and 1.

More recently, new computing technologies are explored like carbon nanotube field-effect transistors (CNTFETs) in which a ternary, i.e. three-valued, logic can readily be employed by using a mixture of transistors and memristors [10]–[12]. While most papers only consider 1-digit adders (so-called half and full adders) and straightforward N-digit adder constructions like ripple carry adders, in [11] the realization of an arbitrary-width fast ternary adder[1] is discussed—with the same restriction on the carry values as above.

Overall, the synthesis of efficient adders has only been considered for small radices so far and only for restricted carries. However, there are applications (e.g. in efficient multipliers) where the input carry is used as an (unrestricted) third summand such that the restriction to two values would be inappropriate.

In this paper, we provide generic constructions of fast adders for multiple-valued logic (MVL) that generalize the binary constructions for arbitrary radices. A key characteristic of these adders is that they work regardless of whether the carries are restricted to two values or not. Moreover, we show that the constructions are also asymptotically optimal regarding time (and space) complexity.

The remainder of the paper is structured as follows: Section II provides basic definitions of multiple-valued adders and complexity measures. In Section III we define a set of elementary MVL operators and prove some important properties and relationships that are required for the synthesis of efficient MVL adders that is discussed in detail in Sections IV and V. Finally, we provide our conclusions in Section VI.

II. Preliminaries

A. Multiple-Valued Adders

Let $a, b, s \in \{0, \ldots, p-1\}^n$ be p-valued numbers of n digits with the interpretation as unsigned positive numbers

$$d_p = [d_{n-1}, \ldots, d_0]_p = \sum_{i=0}^{n-1} d_i p^i$$

and $c_{n-1}, c_{-1} \in \{0, \ldots, p-1\}$.

The function that maps $(a, b, c_{-1}) \mapsto (c_{n-1}, s)$ such that $[c_{n-1}, s_{n-1} \ldots s_0]_p = a_p + b_p + c_{-1}$ is called an n-digit adder. Here, c_{-1} is called input carry and c_{n-1} is called output carry.

Special cases are given by the *Half Adder* (HA), a 1-digit adder without an input carry, i.e. $c_{-1} = 0$, which realizes the addition of two p-valued numbers, and the *Full Adder* (FA), which is a 1-digit adder of two p-valued numbers and an (unrestricted) input carry.

Concatenating multiple full adders allows for the realization of a simple n-digit adder using the output carry of each Full Adder as the input carry of the succeeding full adder as shown in Fig. 1. This well-known construction is termed *Ripple Carry Adder* (RCA).

[1]The term ternary adder is sometimes also used for adders with three summands, especially when Boolean logic is considered.

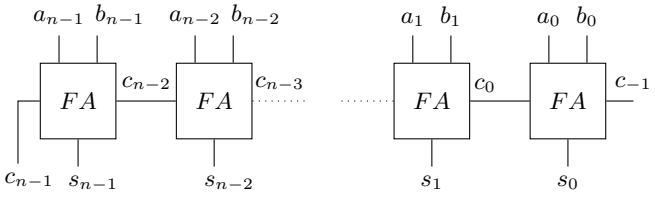

Fig. 1. Ripple Carry Adder

B. Time and Space Complexity

To quantify the complexity of adders we make use of their realization as combinational p-valued circuits. These can be formalized as directed, acyclic graphs $G = (V, E)$ whose vertex set consists of primary input nodes (for the operands' digits $a_{n-1}, b_{n-1}, \ldots, a_0, b_0, c_{-1}$), primary output nodes (for the result's digits $c_{n-1}, s_{n-1}, \ldots, s_0$) and internal nodes with at most two incoming edges representing primitive p-valued logic gates with up to 2 inputs. Note that the primary inputs are root nodes of the graph with no incoming edges and the primary outputs are leaves of the graph with no outgoing edges.

The *cost* (space complexity) of an adder is then defined as the number of internal nodes, i.e. primitive p-valued logic gates, and its *depth* (time complexity) is given by the maximum length of the shortest path between any primary in- and output.

Thus, the cost of an RCA is linear in the number of input digits since it requires n Full Adders (with constant cost):

$$cost(RCA) = n \cdot cost(FA) = n \cdot O(1) = O(n)$$

To determine the depth, we observe that both outputs of a Full Adder depend on three inputs and it is clear that the subgraph of G that computes a function depending on three inputs requires at least two levels of primitive logic gates. In fact, the computation can be expanded to a binary in-tree and in general a binary in-tree with k leaves (here: primary inputs) has at least depth $\lceil \log_2(k) \rceil$ and contains at least $k-1$ internal nodes. The case for $k = 3$ is shown in Fig. 2.

Thus, also the depth of a ripple carry adder is linear in the number of input digits, since there is a shortest/critical path that traverses all full adders via the carry in- and outputs.

Overall, the interpretation as binary in-trees implies that an asymptotically optimal adder, for which the output carry depends on all $2n + 1$ primary inputs, has cost $O(n)$ (like the ripple carry adder), while the depth could potentially be logarithmic in n in the optimal case.

III. MVL Operators

In this section we define four operators for MVL and prove a few properties and relationships required for the construction of MVL adders with asymptotically optimal time (and space) complexity.

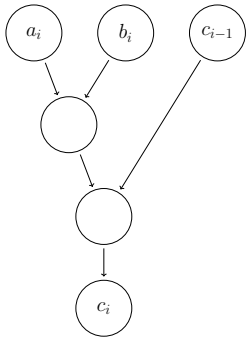

Fig. 2. Binary In-Tree for Full Adder

Definition 1. *For $a, b \in \{0, \ldots, p-1\}$ we define binary operators $SUM, \times, \oplus, \cdot$ as follows:*

- $SUM(a, b) = (a + b) \mod p$
- $a \times b = \begin{cases} 1 & a + b \geq p \\ 0 & else \end{cases}$
- $a \oplus b = \max(a, b)$
- $a \cdot b = \max(0, a + b - (p - 1))$

Remark 1. *Note that SUM and $\times$ can be interpreted as the sum and carry outputs of a p-valued half-adder, i.e.*

$$HA(a, b) := (a \times b, SUM(a, b)) \quad \textit{and}$$

$$a + b = p \cdot (a \times b) + SUM(a, b) = [HA(a, b)]_p.$$

Thus we say that a and b generate a carry *if, and only if, $a \times b = 1$.*

It is immediately clear from these definitions that all operators are commutative and SUM as well as $\oplus$ are associative. Associativity also holds for $\cdot$ according to

Lemma 1. *For $a, b, c \in \{0, \ldots, p-1\}$ it holds that*

$$(a \cdot b) \cdot c = a \cdot (b \cdot c)$$

Proof. We observe that $0 \leq b \cdot c \leq \min(b, c) \leq b, c \leq p - 1$ and $a \cdot b = 0 \Leftrightarrow a + b \leq p - 1$. The proof is done by case analysis.

- Case $a \cdot b = 0$: this means that the left-hand side of the equation evaluates to 0

  $$0 \leq (a \cdot b) \cdot c = 0 \cdot c \leq 0$$

  and the same also holds for the right-hand side since

  $$a + (b \cdot c) \leq a + b \leq p - 1$$

- Case $a \cdot b > 0$:
  - If $(a \cdot b) \cdot c = 0$, then $a + b + c \leq 2(p - 1)$ or equivalently $a + b + c - 2(p - 1) \leq 0$.
    - If $b \cdot c = 0$, also the right hand side evaluates to 0 with the same argumentation as for case $a \cdot b = 0$.
    - If $b \cdot c > 0$, we have for the right-hand side

      $$a \cdot (b \cdot c) = \max(0, a + b + c - 2(p - 1)) = 0$$

      so both sides equal to 0.

– If $(a \cdot b) \cdot c > 0$, we obtain

$$\begin{aligned}(a \cdot b) \cdot c > 0 &\Leftrightarrow a + b + c > 2(p-1)\\ &\Rightarrow b + c > p - 1\\ &\Leftrightarrow b \cdot c > 0\end{aligned}$$

Thus, $0 < (a \cdot b) \cdot c = a + b + c - 2(p-1) = a \cdot (b \cdot c)$. Overall, the equality is proven for all possible cases. □

Though $\times$ is not associative, we can nonetheless prove a useful property:

Lemma 2. *For $a \leq 1$ and $b, c \in \{0, \ldots, p-1\}$ it holds that*

$$(a \times b) \times c = a \times (b \cdot c) \tag{1}$$

Proof. Both sides of the equation can only become 1 if $a = 1$ and $b = c = p - 1$. □

Remark 2. *For $a \geq 2$, the equation does not hold, since for $b = p - 1$ and $c = p - 2$ the left-hand side becomes 0, while the right-hand side evalutes to 1.*

Moreover, a distributive law holds for $\oplus$ and $\times$.

Lemma 3. *For $a, b, c \in \{0, \ldots, p-1\}$ it holds that*

$$(a \oplus b) \times c = (a \times c) \oplus (b \times c) \tag{2}$$

Proof. The maximum of a and b generates a carry together with c if, and only if, at least one of them generates a carry with c. □

IV. Synthesis of MVL Conditional Sum Adder

We begin our studies on efficient MVL adders by noting that the *Conditional Sum Adder* (CoSA, [4])—a simple adder with logarithmic depth, but super-linear cost—can be generalized in a straightforward way from two-valued to p-valued logic. More precisely, the basic idea of a CoSA is to split up the addition of $2n$ bits into multiple additions of n bit (as shown in Fig. 3). The lower n bits only need to be computed once, but the higher n bits are computed multiple times in parallel, once for each possible value of the input carry at bit n, i.e. the carry output of the lower n bits. Note that in MVL adders, the internal carry bits can not only assume values 0 and 1, but also 2, namely if the input carry c_{-1} is greater than or equal to 2 (since $a_i + b_i \leq 2p - 2$). Thus, we require three computations for the higher n bits and once the correct value of the intermediate carry has been computed, 3-to-1 multiplexer (MUX_3) can be used to select the outputs of the corresponding addition of the higher n bits (the carry can assume only three different values 0,1, and 2).

To see that the depth of this adder is logarithmic in n, we observe that

$$\begin{aligned}depth(CoSA_n) &= depth(CoSA_{n/2}) + depth(MUX_3)\\ &= depth(CoSA_{n/4}) + 2 \cdot depth(MUX_3)\\ &= \ldots\\ &= depth(CoSA_1) + \log_2(n) \cdot depth(MUX_3)\end{aligned}$$

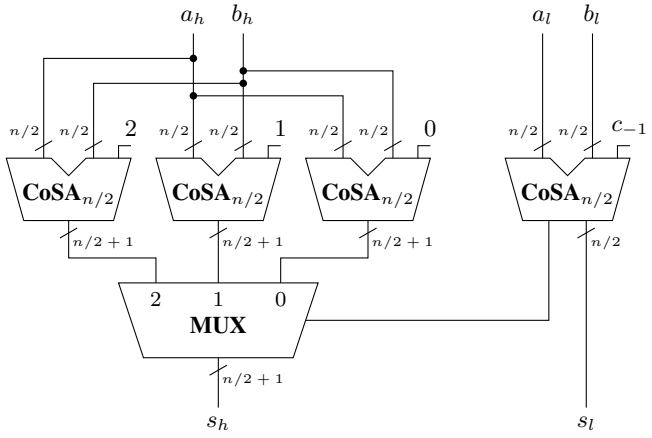

Fig. 3. Conditional Sum Adder

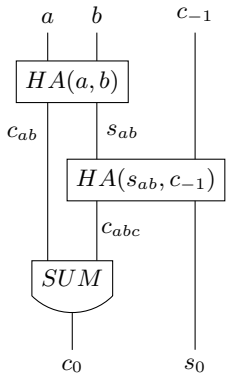

Fig. 4. Full Adder constructed from Half Adders and SUM operation

and $CoSA_1$ can be interpreted as a full adder (with constant depth). More precisely, a full adder can be constructed from two half adders and one SUM operation as shown in Fig. 4.

The following equations show that this indeed realizes a 1-digit adder as defined in Section II-A:

$$\begin{aligned}[c_0, s_0]_p &= p \cdot c_0 + s_0 && \mid SUM(c_{ab}, c_{abc})\\ &= p \cdot [(c_{ab} + c_{abc}) \mod p] + s_0 && \mid c_{ab}, c_{abc} \leq 1\\ &= p \cdot (c_{ab} + c_{abc}) + s_0\\ &= p \cdot c_{ab} + (p \cdot c_{abc} + s_0) && \mid HA(s_{ab}, c_{-1})\\ &= p \cdot c_{ab} + (s_{ab} + c_{-1})\\ &= (p \cdot c_{ab} + s_{ab}) + c_{-1} && \mid HA(a, b)\\ &= (a + b) + c_{-1}\end{aligned}$$

Regarding the super-linear cost, we observe that the cost increases at least by factor 4 when n is doubled, since $cost(CoSA_{2n}) = 4 \cdot cost(CoSA_n) + (n+1) \cdot cost(MUX_3)$.

V. Synthesis of MVL Carry-Lookahead Adder

The *Carry Lookahead Adder* (CLA) for MVL is a more sophisticated adder that combines logarithmic depth *and* linear cost. It can be obtained by generalizing the construction

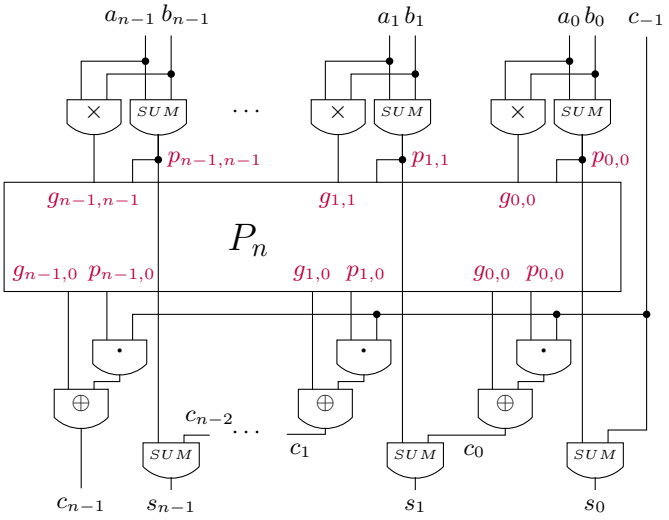

Fig. 5. Carry Lookahead Adder

for two-valued logic [1]. The basic idea of the construction is that it is sufficient to compute the carry bits c_i for all i, from which the output bits can be computed as $s_i = SUM(SUM(a_i, b_i), c_{i-1})$. Here, a parallel prefix computation based on the generation and propagation properties for addition is employed to efficiently compute the carry bits (shown as a block P_n in Fig. 5).

Given the premise that all carries are restricted to two values (0 and 1), the generalization of this construction to arbitrary radices boils down to finding operations that work like Boolean AND and OR on the two possible carry values (e.g. generalized min and max operations), as also done in [11] for $p = 3$. In contrast, the following construction also supports the value 2 as intermediate carry which occurs if the input carry c_{-1} is greater than or equal to 2 (since $a_i + b_i \leq 2p-2$). However, in order to simplify our considerations, we restrict to an input carry $c_{-1} \in \{0, 1, 2\}$. Larger input carries can be supported by applying a Full Adder to digit 0. This yields an output carry $c_0 \leq 2$ and allows to apply the proposed CLA construction for the remaining digits. To distinguish the different carry values, we will briefly denote them as 1-carry and 2-carry, respectively.

A. Carry Generation and Propagation Properties

Generation and propagation properties for addition are described by function families $g = \{g_{j,i} \mid 0 \leq i \leq j < n\}$ and $p = \{p_{j,i} \mid 0 \leq i \leq j < n\}$, respectively, where

- $g_{j,i} \colon \{0, \ldots, p-1\}^{2n} \to \{0, 1\}$ indicates whether bits i to j generate a carry,
- $p_{j,i} \colon \{0, \ldots, p-1\}^{2n} \to \{0, \ldots, p-1\}$ indicates whether bits i to j propagate a carry. More precisely $p_{j,i}$ shall assume
  - value $p-1$ if, and only if, both a 1-carry and a 2-carry are propagated (which is consistent with the definition in the binary case) and
  - value $p-2$ if, and only if, a 2-carry is propagated, but is reduced to a 1-carry.

Using these functions, the carry bit c_j can be computed as

$$c_j = g_{j,i} \oplus (p_{j,i} \cdot c_{i-1}) \tag{3}$$

In fact, $c_j = 1$, if bits i to j

- generate a 1-carry (they cannot generate a 2-carry),
- propagate a 1-carry (i.e., $p_{j,i} = p-1$ and $c_{i-1} = 1$), or
- reduce an incoming 2-carry to a 1-carry (i.e., $p_{j,i} = p-2$ and $c_{i-1} = 2$).

In the latter two cases we have $p_{j,i} \cdot c_{i-1} = 1$. The only way for having $c_j = 2$ is, when a 2-carry is propagated, i.e. $c_{i-1} = 2$ and $p_{j,i} = p-1$ which implies $p_{j,i} \cdot c_{i-1} = 2$.

For the case $i = j$, the functions are defined as follows:

$$p_{i,i} = SUM(a_i, b_i) \tag{4}$$

$$g_{i,i} = a_i \times b_i \tag{5}$$

For $i < j$, we obtain for an arbitrary k with $i \leq k < j$:

$$g_{j,i} = g_{j,k+1} \oplus (g_{k,i} \times p_{j,k+1}) \tag{6}$$

$$p_{j,i} = p_{k,i} \cdot p_{j,k+1} \tag{7}$$

Regarding the correctness of Eqn. (6), we note that the operator $\oplus$ corresponds to a logical OR, as $g_{*,*}$ can only assume the values 0 or 1. For the same reason, the term in the parentheses evaluates to 1 if, and only if, $p_{j,k+1} = p-1$ and $g_{k,i} = 1$. Thus, a carry is generated if bits $k+1$ to j generate a carry themselves or if bits i to k generate a carry and bits $k+1$ to j propagate it.

Regarding the correctness of Eqn. (7), the expression on the right-hand side evaluates to $p-1$ if, and only if, both operands have value $p-1$. Thus, a carry is propagated in the entire range if, and only if, both bit ranges propagate a carry. Value $p-2$ is assumed if, and only if, one operand has value $p-1$ and the other has value $p-2$. That means that a 2-carry either passes the range i to j unaltered and then reduces to a 1-carry in the remaining bits or the 2-carry reduces to a 1-carry in the range i to j and is then propagated over the remaining bits.

B. Parallel Prefix Computation

The general idea of parallel prefix computation is that the computation of the products $x_i = y_i \circ y_{i-1} \circ \ldots \circ y_0$ for $i = 0, \ldots, 2n-1$ can be conducted in parallel for an associative operator $\circ$ using the following construction:

1) Compute $y'_j = y_{2j+1} \circ y_{2j}$ for $j = 0, \ldots, n-1$.
2) Compute $x'_j = y'_j \circ y'_{j-1} \circ \ldots \circ y'_0 = y_{2j+1} \circ y_{2j} \circ \ldots \circ y_0$.
3) Output $x_i = x'_{(i-1)/2}$ for odd i and $x_i = y_i \circ x'_{i/2}$ for even i.

The first step can be conducted in parallel for all j, the third step can be conducted in parallel for all i, so the depth of both steps is $depth(\circ)$ and both steps require n applications of $\circ$. The second step can be conducted by parallel

prefix computation with half the number of operands. So $depth(P_n) = (log_2(n) - 1) \cdot depth(\circ)$ and

$$\begin{aligned} cost(P_n) &= 2n \cdot cost(\circ) + cost(P_{n/2}) \\ &= 2n \cdot cost(\circ)(1 + \frac{1}{2}) + cost(P_{n/4}) \\ &= 2n \cdot cost(\circ)(1 + \frac{1}{2} + \frac{1}{4} + \ldots) + cost(P_2) \\ &\leq 4n \cdot cost(\circ) \end{aligned}$$

In order to exploit parallel prefix computation for the CLA, we note that according to Eqn. (3) the carry bits c_i can be computed from the input carry c_{-1} as

$$c_i = g_{i,0} \oplus (p_{i,0} \cdot c_{-1}) \tag{8}$$

In order to compute the $g_{i,0}$ and $p_{i,0}$ in an efficient way, we require an associative operator $\circ$ that satisfies

$$(g_{i,0}, p_{i,0}) = (g_{i,i}, p_{i,i}) \circ \ldots \circ (g_{1,1}, p_{1,1}) \circ (g_{0,0}, p_{0,0})$$

Thus, according to Eqn. (6) and (7) the operator $\circ$ shall be defined as follows

$$(g_2, p_2) \circ (g_1, p_1) := \big(g_2 \oplus (g_1 \times p_2), p_2 \cdot p_1\big) \tag{9}$$

We will now prove that this definition indeed yields an associative operator under certain conditions.

Theorem 1. *The operator $\circ$ is associative if, and only if, it is applied on pairs of 2-valued and p-valued functions over $\{0, \ldots, p-1\}^{2n}$.*

Proof of Theorem 1. Let g_1, g_2, g_3 and p_1, p_2, p_3 be p-valued functions over $\{0, \ldots, p-1\}^{2n}$. We observe that

$$\begin{aligned} &(g_3, p_3) \circ \big((g_2, p_2) \circ (g_1, p_1)\big) \\ &= (g_3, p_3) \circ \big(g_2 \oplus (g_1 \times p_2), p_2 \cdot p_1\big) \\ &= (g_3 \oplus [(g_2 \oplus (g_1 \times p_2)) \times p_3], p_3 \cdot (p_2 \cdot p_1)) \\ &= (g_3 \oplus (g_2 \times p_3) \oplus ((g_1 \times p_2) \times p_3), p_3 \cdot (p_2 \cdot p_1)) \\ &= (g_3 \oplus (g_2 \times p_3) \oplus [g_1 \times (p_2 \cdot p_3)], p_3 \cdot (p_2 \cdot p_1)) \end{aligned}$$

using Eqn. (2) and (1) to come from the third to the fourth and the fifth line, respectively. Note that according to Lemma 2 and Remark 2 the step from the fourth to the fifth line is only valid if $g_1 \leq 1$, i.e. if g_1 is a 2-valued function.

On the other hand we obtain

$$\begin{aligned} &\big((g_3, p_3) \circ (g_2, p_2)\big) \circ (g_1, p_1) \\ &= \big(g_3 \oplus (g_2 \times p_3), p_3 \cdot p_2\big) \circ (g_1, p_1) \\ &= \big([g_3 \oplus (g_2 \times p_3)] \oplus [g_1 \times (p_3 \cdot p_2)], (p_3 \cdot p_2) \cdot p_1\big) \\ &= \big(g_3 \oplus (g_2 \times p_3) \oplus [g_1 \times (p_2 \cdot p_3)], (p_3 \cdot p_2) \cdot p_1\big) \end{aligned}$$

using associativity of $\oplus$ and commutativity of $\cdot$ to come from the third to the fourth line. Since $\cdot$ is associative, the resulting terms in the second component are equivalent. Overall, associativity is proven for $\circ$ under the condition that g_1 is a 2-valued function. □

Since all generation functions from g are 2-valued functions, the operator $\circ$ as defined above is associative in our setting and can be used for parallel prefix computation.

The cost of the CLA is linear in n, since the cost of parallel prefix computation is linear and it furthermore requires

1) a single SUM or $\times$ operation, respectively, to compute the $g_{i,i}$ and $p_{i,i}$ (c.f. Eqn. (4) and (5)),
2) two operations are required to compute the carry bits (c.f. Eqn. (8)) and
3) one more SUM is required to obtain the sum bits $s_i = SUM(SUM(a_i, b_i), c_{i-1}) = SUM(p_{i,i}, c_{i-1})$.

The depth is logarithmic in n, since the parallel prefix computation can be done in logarithmic time and the surrounding logic has constant depth of 4.

VI. Conclusions

In this work, we considered the synthesis of efficient adders for multiple-valued logic. In contrast to previous work, we provided two generic constructions that work for arbitrary radices and input carry values. While the generalized Conditional Sum Adder achieves asymptotically optimal time complexity (logarithmic in the number of input digits), it has super-linear and, hence, sub-optimal cost. The second, more sophisticated construction which is a generalized Carry Lookahead Adder achieves both, asymptotically optimal time *and* space complexity.

References

[1] A. Weinberger and J. L. Smith, "A one-microsecond adder using one-megacycle circuitry," *IRE Transactions on Electronic Computers*, vol. EC-5, no. 2, pp. 65–73, 1956.

[2] Brent and Kung, "A regular layout for parallel adders," *IEEE Transactions on Computers*, vol. C-31, no. 3, pp. 260–264, 1982.

[3] T. Kilburn, D. Edwards, and D. Aspinall, "A parallel arithmetic unit using a saturated-transistor fast-carry circuit," in *Proc. IEE*, vol. 107 (B), 1960, pp. 573–584.

[4] J. Sklansky, "Conditional-sum addition logic," *IRE Transactions on Electronic Computers*, vol. EC-9, no. 2, pp. 226–231, 1960.

[5] ——, "An evaluation of several two-summand binary adders," *IRE Transactions on Electronic Computers*, vol. EC-9, no. 2, pp. 213–226, 1960.

[6] N. W. Umredkar, M. A. Gaikwad, and D. R. Dandekar, "Review of quaternary adders in voltage mode multi-valued logic," *IJCA Special Issue on Recent Trends in Engineering Technology*, vol. RETRET, pp. 17–21, March 2013.

[7] I. M. Thoidis, D. Soudris, J. M. Fernandez, and A. Thanailakis, "The circuit design of multiple-valued logic voltage-mode adders," in *IEEE International Symposium on Circuits and Systems*, vol. 4, 2001, pp. 162–165 vol. 4.

[8] V. Patel K.S. and K. S. Gurumurthy, "Design of high performance quaternary adders," in *Int'l Symp. on Multiple-Valued Logic*, 2011, pp. 22–26.

[9] H. Shirahama and T. Hanyu, "Design of high-performance quaternary adders based on output-generator sharing," in *Int'l Symp. on Multiple-Valued Logic*, 2008, pp. 8–13.

[10] A. Herrfeld and S. Hentschke, "Ternary multiplication circuits using 4-input adder cells and carry look-ahead," in *Int'l Symp. on Multiple-Valued Logic*, 1999, pp. 174–179.

[11] N. S. Soliman, M. E. Fouda, L. A. Said, A. H. Madian, and A. G. Radwan, "N-digits ternary carry lookahead adder design," in *Int'l Conf. on Microelectronics (ICM)*, 2019, pp. 142–145.

[12] D. Etiemble, "Comparing ternary and binary adders and multipliers," *CoRR*, vol. abs/1908.07299, 2019. [Online]. Available: http://arxiv.org/abs/1908.07299