2019 IEEE 26th Symposium on Computer Arithmetic (ARITH)

An Ultra-Fast Parallel Prefix Adder

Kumar Sambhav Pandey
School of Comp & Elect Engg
Indian Institute of Technology
Mandi, India
kumar@nith.ac.in, pandey@iitmandi.ac.in

Dinesh Kumar B
School of Comp & Elect Engg
Indian Institute of Technology
Mandi, India
dinesh@projects.iitmandi.ac.in

Neeraj Goel
Dept of Comp Sc & Engg
Indian Institute of Technology
Ropar, India
neeraj@iitrpr.ac.in

Hitesh Shrimali
School of Comp & Elect Engg
Indian Institute of Technology
Mandi, India
hitesh@iitmandi.ac.in

***Abstract*—Parallel Prefix adders are arguably the most commonly used arithmetic units. They have been extensively investigated at architecture level, register transfer level (RTL), gate level, circuit level as well as layout level giving rise to a plethora of mathematical formulations, topologies and implementations. This paper contributes significantly to the understanding of these parallel prefix adders in a couple of ways. Firstly, it attempts to describe various such parallel prefix adders in elegant and consistent formulations. Secondly, a new family of parallel prefix adders is proposed at architecture level. The estimates of the area-throughput characteristics for an instance of this family are also presented. While the speeds achieved by this instance match those achieved by the state of the art adders, their area characteristics exhibit upto 26% improvement.**

***Keywords*-Parallel prefix adders; adder recurrence relations; digital arithmetic;**

I. Introduction

Binary addition of multi-bit operands is one of the fundamental operations in the domain of contemporary general purpose as well as signal processing applications. Many other operations like multiplication, floating point computations, multiply-accumulate units or increment-decrement operations are all composed of variants of adders. An efficient implementation of such adders has, therefore, a profound impact on the design of any data processing unit.

Parallel prefix adders are particularly interesting from the CMOS implementation point of view because of the simplicity of their cell designs and the regularity of interconnects connecting these cells. Moreover, the implementations of these adders are easy to pipeline. Various architectures and design alternatives have been investigated in literature [1]–[3]. All such investigations are based on computation of some local signals in parallel for all the bits of operands. These signals are then reduced over larger groups in tree-like fashion. Then the sum bits and carry_out bit from the adder are computed in parallel [4]–[11]. Reductions of these signals over larger groups are based on simple recursive algorithms which can be elegantly expressed as logical recurrence relations [12]–[16].

These local signals are mathematically defined and their useful properties are discussed in Section II where an elegant and consistent terminology for further discussions is introduced. Using this terminology, Weinberger-Smith [12], Ling [13], Doran [14] and Jackson-Talwar [15] recursions for multi-bit integer addition are presented, compared and contrasted with each other in the same section. The section also touches upon higher valency adders [17]–[19].

The existing literature on architectural innovations in parallel prefix adders is scant as well as sparsely populated in time, whereas the impementation community has been fairly active. It is our belief that the absence of a unified theory and a consistent terminology in the domain might be the reason d'etre. As the architectures of various parallel prefix adders are described in a consistent formulation, more efficient higher order recurrence relations and corresponding novel architectures of these adders emerge.

Section III presents our novel recursion for multi-bit integer addition and the corresponding realisation for one instance of the proposed family of parallel prefix adders. This section further demonstrates the scalability and the generic nature of the proposal. The discussions are restricted at architecture level and are, therefore, independent of the technology in which they are implemented.

Estimates on the number of logic levels in the critical path and the total gate count for an instance of the family (having recurrence of the order 4) are presented and discussed in section IV. Comparisons with corresponding figures of merit for Weinberger-Smith [12], Ling [13] and Jackson-Talwar [15] adders are also made in this section. Section V concludes the paper.

II. Parallel Prefix Adders: Mathematical Formalism and Terminology

We use capital letters to represent binary operands. For

978-1-7281-3366-9/19/$31.00 ©2019 IEEE
DOI 10.1109/ARITH.2019.00034

example A and B are used for addend and augend respectively. C_{in} and C_{out} denote the carry-in to and carry-out of the adder respectively whereas S denotes the sum. Small letters are used to represent bits (addend, augend, sum as well as some special local signals) and subscripts are used to indicate their arithmetic weight, increasing from 0 at the least significant bit. Thus a_i signifies the addend, b_i signifies the augend S_i signifies the sum, while g_i signifies a special signal (generate) at bit position i and so on. An exception is made here for representing local carries, real or pseudo, where capital letters are used, as in C_i, H_i, K_i etc. to differentiate them from other local signals like h_i, q_i, k_i etc.

The first stage of all the conventional parallel prefix adders computes two signals, carry generate (g_i) and carry propagate (p_i), which being local to each bit position can all be computed in parallel [8]. They are defined as:

$$g_i = a_i \cdot b_i \tag{1}$$

$$p_i = a_i + b_i \tag{2}$$

Let us also define another local signal (t_i) as follows:

$$t_i = a_i \oplus b_i \tag{3}$$

We also use C_i to denote input carry at bit position i. It is trivial to note that:

$$C_{i+1} = g_i + p_i \cdot C_i \tag{4}$$

The final stage computes the sum bits as:

$$s_i = t_i \oplus C_i \tag{5}$$

Note that the first and the last stages are purely local in nature as they operate on signals only at their respective bit positions. Hence all of the bits can be operated upon concurrently. However, there is a data dependence between C_{i+1} and C_i; and thus they cannot be computed concurrently. In conventional parallel prefix adders, carries are computed by using a special binary prefix operator ($\circ$) defined on pairs of operands as:

$$\begin{pmatrix} g_i \\ p_i \end{pmatrix} \circ \begin{pmatrix} g_j \\ p_j \end{pmatrix} = \begin{pmatrix} g_i + p_i \cdot g_j \\ p_i \cdot p_j \end{pmatrix} \tag{6}$$

Thus,

$$\begin{pmatrix} C_{i+1} \\ p_i \end{pmatrix} = \begin{pmatrix} g_i \\ p_i \end{pmatrix} \circ \begin{pmatrix} C_i \\ 1 \end{pmatrix} \tag{7}$$

Input carry at any bit position as a function of C_{in} can thus be trivially computed using a sequence of the prefix operations ($\circ$)s introduced above as:

$$\begin{pmatrix} C_{i+1} \\ p_i \cdot p_{i-1} \cdots p_0 \end{pmatrix} = \begin{pmatrix} g_i \\ p_i \end{pmatrix} \circ \begin{pmatrix} g_{i-1} \\ p_{i-1} \end{pmatrix} \cdots \begin{pmatrix} g_0 \\ p_0 \end{pmatrix} \circ \begin{pmatrix} C_{in} \\ 1 \end{pmatrix} \tag{8}$$

We also define a group generate signal $g_{i\cdots j}$ and a group propagate signal $p_{i\cdots j}$ as:

$$\begin{pmatrix} g_{i\cdots j} \\ p_{i\cdots j} \end{pmatrix} = \begin{pmatrix} g_i \\ p_i \end{pmatrix} \circ \begin{pmatrix} g_{i-1} \\ p_{i-1} \end{pmatrix} \cdots \begin{pmatrix} g_{j+1} \\ p_{j+1} \end{pmatrix} \circ \begin{pmatrix} g_j \\ p_j \end{pmatrix} \tag{9}$$

Thus equation (8) can also be written as:

$$\begin{pmatrix} C_{i+1} \\ p_{i\cdots 0} \end{pmatrix} = \begin{pmatrix} g_{i\cdots 0} \\ p_{i\cdots 0} \end{pmatrix} \circ \begin{pmatrix} C_{in} \\ 1 \end{pmatrix} \tag{10}$$

Using equation (4), C_{i+1} can be computed sequentially which results in carry ripple adders. However, a couple of properties of the prefix operator ($\circ$) are particularly insightful in reducing sequences of these operations in parallel [8]. Design of fast parallel prefix adders is based on these properties. Firstly, it is trivial to prove that the operator is associative, i.e.,

$$\left(\begin{pmatrix} g_i \\ p_i \end{pmatrix} \circ \begin{pmatrix} g_j \\ p_j \end{pmatrix}\right) \circ \begin{pmatrix} g_k \\ p_k \end{pmatrix} = \begin{pmatrix} g_i \\ p_i \end{pmatrix} \circ \left(\begin{pmatrix} g_j \\ p_j \end{pmatrix} \circ \begin{pmatrix} g_k \\ p_k \end{pmatrix}\right) \tag{11}$$

Secondly, the operator is idempotent, i.e.,

$$\begin{pmatrix} g_{h\cdots i} \\ p_{h\cdots i} \end{pmatrix} \circ \begin{pmatrix} g_{j\cdots k} \\ p_{j\cdots k} \end{pmatrix} = \begin{pmatrix} g_{h\cdots k} \\ p_{h\cdots k} \end{pmatrix} \tag{12}$$

where, $h > i$, $i \leq j+1$ and $j > k$ i.e. the prefix operator sequences should either overlap or should be adjacent.

Having introduced the terminology and mathematical formalism, we describe Weinberger-Smith [12], Ling [13], Doran [14], and Jackson-Talwar [15] recursions for multi-bit integer addition in the following subsections.

A. Weinberger-Smith Recurrence and Conventional Carry Look Ahead Adder

The Weinberger-Smith [12] recurrence relation for multi-bit integer addition is given by equations (7) and (10), where $C_0 = C_{in}$ and $g_i = 0$, $p_i = 1$ $\forall i < 0$. The algorithm based on Ladner-Fischer topology [5] for a radix-8 adder is as hereinafter given.

1) g_i, p_i and t_i ($\forall i : i \geq 0$) are computed in parallel.
2) Following are all computed in parallel:

$$\begin{pmatrix} g_{1\cdots 0} \\ p_{1\cdots 0} \end{pmatrix} = \begin{pmatrix} g_1 \\ p_1 \end{pmatrix} \circ \begin{pmatrix} g_0 \\ p_0 \end{pmatrix},$$

$$\begin{pmatrix} g_{3\cdots 2} \\ p_{3\cdots 2} \end{pmatrix} = \begin{pmatrix} g_3 \\ p_3 \end{pmatrix} \circ \begin{pmatrix} g_2 \\ p_2 \end{pmatrix},$$

$$\begin{pmatrix} g_{5\cdots4} \\ p_{5\cdots4} \end{pmatrix} = \begin{pmatrix} g_5 \\ p_5 \end{pmatrix} \circ \begin{pmatrix} g_4 \\ p_4 \end{pmatrix},$$

$$\begin{pmatrix} g_{7\cdots6} \\ p_{7\cdots6} \end{pmatrix} = \begin{pmatrix} g_7 \\ p_7 \end{pmatrix} \circ \begin{pmatrix} g_6 \\ p_6 \end{pmatrix},$$

$$\begin{pmatrix} C_1 \\ p_0 \end{pmatrix} = \begin{pmatrix} g_0 \\ p_0 \end{pmatrix} \circ \begin{pmatrix} C_{in} \\ 1 \end{pmatrix}$$

3) Following are all computed in parallel:

$$\begin{pmatrix} g_{2\cdots0} \\ p_{2\cdots0} \end{pmatrix} = \begin{pmatrix} g_2 \\ p_2 \end{pmatrix} \circ \begin{pmatrix} g_{1\cdots0} \\ p_{1\cdots0} \end{pmatrix},$$

$$\begin{pmatrix} g_{3\cdots0} \\ p_{3\cdots0} \end{pmatrix} = \begin{pmatrix} g_{3\cdots2} \\ p_{3\cdots2} \end{pmatrix} \circ \begin{pmatrix} g_{1\cdots0} \\ p_{1\cdots0} \end{pmatrix},$$

$$\begin{pmatrix} g_{6\cdots4} \\ p_{6\cdots4} \end{pmatrix} = \begin{pmatrix} g_6 \\ p_6 \end{pmatrix} \circ \begin{pmatrix} g_{5\cdots4} \\ p_{5\cdots4} \end{pmatrix},$$

$$\begin{pmatrix} g_{7\cdots4} \\ p_{7\cdots4} \end{pmatrix} = \begin{pmatrix} g_{7\cdots6} \\ p_{7\cdots6} \end{pmatrix} \circ \begin{pmatrix} g_{5\cdots4} \\ p_{5\cdots4} \end{pmatrix},$$

$$\begin{pmatrix} C_2 \\ p_{1\cdots0} \end{pmatrix} = \begin{pmatrix} g_{1\cdots0} \\ p_{1\cdots0} \end{pmatrix} \circ \begin{pmatrix} C_{in} \\ 1 \end{pmatrix}$$

4) Following are all computed in parallel:

$$\begin{pmatrix} g_{4\cdots0} \\ p_{4\cdots0} \end{pmatrix} = \begin{pmatrix} g_4 \\ p_4 \end{pmatrix} \circ \begin{pmatrix} g_{3\cdots0} \\ p_{3\cdots0} \end{pmatrix},$$

$$\begin{pmatrix} g_{5\cdots0} \\ p_{5\cdots0} \end{pmatrix} = \begin{pmatrix} g_{5\cdots4} \\ p_{5\cdots4} \end{pmatrix} \circ \begin{pmatrix} g_{3\cdots0} \\ p_{3\cdots0} \end{pmatrix},$$

$$\begin{pmatrix} g_{6\cdots0} \\ p_{6\cdots0} \end{pmatrix} = \begin{pmatrix} g_{6\cdots4} \\ p_{6\cdots4} \end{pmatrix} \circ \begin{pmatrix} g_{3\cdots0} \\ p_{3\cdots0} \end{pmatrix},$$

$$\begin{pmatrix} g_{7\cdots0} \\ p_{7\cdots0} \end{pmatrix} = \begin{pmatrix} g_{7\cdots4} \\ p_{7\cdots4} \end{pmatrix} \circ \begin{pmatrix} g_{3\cdots0} \\ p_{3\cdots0} \end{pmatrix},$$

$$\begin{pmatrix} C_3 \\ p_{2\cdots0} \end{pmatrix} = \begin{pmatrix} g_{2\cdots0} \\ p_{2\cdots0} \end{pmatrix} \circ \begin{pmatrix} C_{in} \\ 1 \end{pmatrix},$$

$$\begin{pmatrix} C_4 \\ p_{3\cdots0} \end{pmatrix} = \begin{pmatrix} g_{3\cdots0} \\ p_{3\cdots0} \end{pmatrix} \circ \begin{pmatrix} C_{in} \\ 1 \end{pmatrix}$$

5) Following are all computed in parallel:

$$\begin{pmatrix} C_5 \\ p_{4\cdots0} \end{pmatrix} = \begin{pmatrix} g_{4\cdots0} \\ p_{4\cdots0} \end{pmatrix} \circ \begin{pmatrix} C_{in} \\ 1 \end{pmatrix},$$

$$\begin{pmatrix} C_6 \\ p_{5\cdots0} \end{pmatrix} = \begin{pmatrix} g_{5\cdots0} \\ p_{5\cdots0} \end{pmatrix} \circ \begin{pmatrix} C_{in} \\ 1 \end{pmatrix},$$

$$\begin{pmatrix} C_7 \\ p_{6\cdots0} \end{pmatrix} = \begin{pmatrix} g_{6\cdots0} \\ p_{6\cdots0} \end{pmatrix} \circ \begin{pmatrix} C_{in} \\ 1 \end{pmatrix},$$

$$\begin{pmatrix} C_8 \\ p_{7\cdots0} \end{pmatrix} = \begin{pmatrix} g_{7\cdots0} \\ p_{7\cdots0} \end{pmatrix} \circ \begin{pmatrix} C_{in} \\ 1 \end{pmatrix}$$

6) s_i $(\forall i : 8 > i \geq 0)$ are computed in parallel using equation (5).

The realisation for a 4 bit adder is shown in Fig. 1. The blocks labled as gpt are the initial processing blocks that compute the signals g_i, p_i and t_i as given by the step (1) of the algorithm and defined in equations (1), (2) and (3) respectively. The blocks labled as $reduce$ compute the group signals defined in steps (2) to (5) of the algorithm in treelike fashion. The blocks labled $carry$ compute the carry out signals as defined by equation (4) while the blocks labled sum compute the sum bits as defined by equation (5) respectively at each bit position.

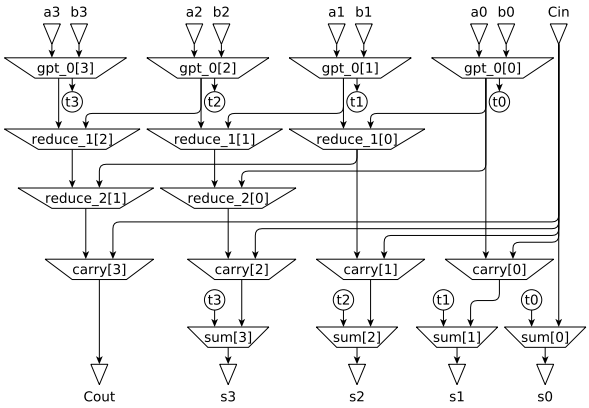

Figure 1. A 4-bit adder based on Weinberger-Smith recurrence in Ladner-Fischer topology.

B. Ling Adder

It is realised in [13] that the recurrence relation for parallel prefix carry computation can be simplified if a pseudo carry $H_i = C_i + C_{i-1}$ is propagated in lieu of the conventional carry C_i. Once these pseudo carries are known for all the bit positions, the conventional carries can be extracted from them by $C_i = p_{i-1} \cdot H_i$ as proved below:

$$\begin{aligned} p_{i-1} \cdot H_i &= p_{i-1} \cdot (C_i + C_{i-1}) \\ &= p_{i-1} \cdot (g_{i-1} + p_{i-1} \cdot C_{i-1} + C_{i-1}) \\ &= p_{i-1} \cdot (g_{i-1} + C_{i-1}) \\ &= p_{i-1} \cdot g_{i-1} + p_{i-1} \cdot C_{i-1} \\ &= g_{i-1} + p_{i-1} \cdot C_{i-1} \\ &= C_i \end{aligned}$$

In order to develop Ling recurrence relation, we define two more local signals (h_i) and (q_i) as:

$$h_i = g_i + g_{i-1} \tag{13}$$
$$q_i = p_i \cdot p_{i-1} \tag{14}$$

Starting with the definition of H_{i+1} and noting the fact that $g_i \cdot p_i = g_i$, we can define it in terms of H_{i-1} as given below:

$$\begin{aligned} H_{i+1} &= C_{i+1} + C_i \\ &= g_i + p_i \cdot C_i + C_i \\ &= g_i + C_i \\ &= g_i + g_{i-1} + p_{i-1} \cdot (g_{i-2} + p_{i-2} \cdot C_{i-2}) \\ &= g_i + g_{i-1} + p_{i-1} \cdot p_{i-2} \cdot (g_{i-2} + C_{i-2}) \\ &= h_i + q_{i-1} \cdot H_{i-1} \end{aligned} \tag{15}$$

Thus the above equation and the definition of q_{i-1} can be collected together using the binary prefix operator ($\circ$) as:

$$\begin{pmatrix} H_{i+1} \\ q_{i-1} \end{pmatrix} = \begin{pmatrix} h_i \\ q_{i-1} \end{pmatrix} \circ \begin{pmatrix} H_{i-1} \\ 1 \end{pmatrix} \tag{16}$$

where $H_{in} = H_0 = C_0 = C_{in}$, $h_0 = g_0$, $q_0 = p_0$ and $h_i = 0$, $q_i = 1\ \forall i < 0$.

Ling carry (pseudo carry) at any bit position as a function of $H_{in} = C_{in}$ can thus be trivially computed by a sequence of prefix operations using equation (17) as:

$$\begin{pmatrix} H_{i+1} \\ q_{i-1} \cdot q_{i-3} \cdots q_0 \end{pmatrix} = \begin{pmatrix} h_i \\ q_{i-1} \end{pmatrix} \circ \begin{pmatrix} h_{i-2} \\ q_{i-3} \end{pmatrix} \cdots \begin{pmatrix} H_{in} \\ 1 \end{pmatrix} \tag{17}$$

Ling [13] observed that, using equation (15), pseudo Ling carry H_4 can be written as:

$$\begin{aligned} H_4 &= h_3 + q_2 \cdot h_1 + q_2 \cdot q_0 \cdot H_{in} \\ &= g_3 + g_2 + p_2 \cdot g_1 + p_2 \cdot p_1 \cdot g_0 + p_2 \cdot p_1 \cdot p_0 \cdot C_{in} \end{aligned} \tag{18}$$

which is logically much more simpler than the corresponding expression for the conventional carry C_4 which is given below for ready reference:

$$C_4 = g_3 + p_3 \cdot g_2 + p_3 \cdot p_2 \cdot g_1 + p_3 \cdot p_2 \cdot p_1 \cdot g_0 + p_3 \cdot p_2 \cdot p_1 \cdot p_0 \cdot C_{in} \tag{19}$$

Obviously, because of smaller fan-outs of the gates and simpler logic expressions for the pseudo carries, Ling adders achieve better area-time characteristics than the conventional parallel prefix adders, albeit at the expense of having a more complex final stage that recovers the conventional carries from their respective pseudo Ling carries.

It may be noted that Ling recurrence relation has a striking similarity with Weinberger-Smith recurrence relation and therefore, all the topologies for parallel prefix reduction are equally applicable in the case of Ling adders too. Thus one may implement a Ling adder in any topology as proposed by Ladner-Fishcer [5], Brent-Kung [6] or Knowles [8].

C. Dimitrakopoulos-Nikolos Insight

Following is an expansion of Ling recurrence relation for Ling pseudo carries for a radix-8 adder.

$$\begin{pmatrix} H_8 \\ q_6 \cdot q_4 \cdot q_2 \cdot q_0 \end{pmatrix} = \begin{pmatrix} h_7 \\ q_6 \end{pmatrix} \circ \begin{pmatrix} h_5 \\ q_4 \end{pmatrix} \circ \begin{pmatrix} h_3 \\ q_2 \end{pmatrix} \circ \begin{pmatrix} h_1 \\ q_0 \end{pmatrix} \circ \begin{pmatrix} C_{in} \\ 1 \end{pmatrix},$$
$$\begin{pmatrix} H_7 \\ q_5 \cdot q_3 \cdot q_1 \end{pmatrix} = \begin{pmatrix} h_6 \\ q_5 \end{pmatrix} \circ \begin{pmatrix} h_4 \\ q_3 \end{pmatrix} \circ \begin{pmatrix} h_2 \\ q_1 \end{pmatrix} \circ \begin{pmatrix} h_0 \\ 1 \end{pmatrix},$$
$$\begin{pmatrix} H_6 \\ q_4 \cdot q_2 \cdot q_0 \end{pmatrix} = \begin{pmatrix} h_5 \\ q_4 \end{pmatrix} \circ \begin{pmatrix} h_3 \\ q_2 \end{pmatrix} \circ \begin{pmatrix} h_1 \\ q_0 \end{pmatrix} \circ \begin{pmatrix} C_{in} \\ 1 \end{pmatrix},$$
$$\begin{pmatrix} H_5 \\ q_3 \cdot q_1 \end{pmatrix} = \begin{pmatrix} h_4 \\ q_3 \end{pmatrix} \circ \begin{pmatrix} h_2 \\ q_1 \end{pmatrix} \circ \begin{pmatrix} h_0 \\ 1 \end{pmatrix},$$
$$\begin{pmatrix} H_4 \\ q_2 \cdot q_0 \end{pmatrix} = \begin{pmatrix} h_3 \\ q_2 \end{pmatrix} \circ \begin{pmatrix} h_1 \\ q_0 \end{pmatrix} \circ \begin{pmatrix} C_{in} \\ 1 \end{pmatrix},$$
$$\begin{pmatrix} H_3 \\ q_1 \end{pmatrix} = \begin{pmatrix} h_2 \\ q_1 \end{pmatrix} \circ \begin{pmatrix} h_0 \\ 1 \end{pmatrix},$$
$$\begin{pmatrix} H_2 \\ q_0 \end{pmatrix} = \begin{pmatrix} h_1 \\ q_0 \end{pmatrix} \circ \begin{pmatrix} C_{in} \\ 1 \end{pmatrix},$$
$$\begin{pmatrix} H_1 \\ 1 \end{pmatrix} = \begin{pmatrix} h_0 \\ 1 \end{pmatrix},$$
$$\begin{pmatrix} H_0 \\ 1 \end{pmatrix} = \begin{pmatrix} C_{in} \\ 1 \end{pmatrix}$$

Dimitrakopoulos and Nikolos [16] observed that in the above expansion the even and odd subscripted pseudo Ling carries are independent of each other. This essentially means that they can be reduced in mutually exclusive even and odd sub-trees. Therefore, the 8-input first stage of an 8-bit Ling adder is actually two separate 4-input stages, which eliminates one logic stage in the critical path!

D. Doran Recursion and Variants of Ling Adders

Doran [14] observed that there are a few local signals other than g_i and p_i or even h_i and q_i described by Weinberger-Smith or Ling recurrence respectively which may be used in lieu of them. There are a number of such signals but all of them have only theoretical significance. None of them actually provide any further gains in either area or latency vis-a-vis their conventional realisations. One of such few cases arises if we take another pseudo carry signal X_i in place of signal H_i and take $\overline{p_i}$ in place of g_i as defined below.

$$X_{i+1} = \overline{p_{i+1}} + G_i \tag{20}$$
$$G_i = p_i \cdot X_i \tag{21}$$

Thus,

$$X_{i+1} = \overline{p_{i+1}} + p_i \cdot X_i \tag{22}$$

This recurrence relation is also similar in structure as the other ones defined in earlier subsections and therefore, the parallel prefix reduction tree structure remains similar to that of the parallel prefix adder based on Ling recurrence.

The realisation for a 4-bit Ling adder or any of its variants as suggested by Doran [14] and modified by Dimitrakopoulos and Nikolos [16] is shown in Fig. 2. The blocks labled as hqt are the initial processing blocks that compute the signals h_i, q_i and t_i as defined in equations (13), (14) and (3) respectively. The blocks labled as $reduce$ compute the group signals in treelike fashion. The blocks labled $carry$ compute the carry out signals as defined by equation (4) while the blocks labled sum compute the sum bits as defined by equation (5) respectively at each bit position.

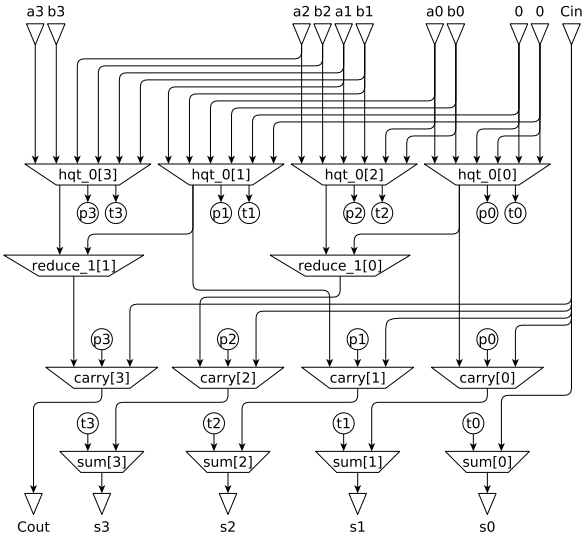

Figure 2. A 4-bit adder based on Ling recurrence in Ladner-Fischer topology.

E. Jackson-Talwar Factorization and Generalization of Ling Adders

Jackson and Talwar [15] have generalized the Ling factorization and introduced Reduced Generate (R) and Hyper Propagate (Q) signals in lieu of conventional Generate (G) and Propagate (Q) to further speed up multi-bit addition and proved that the relations for carries can be factorized even beyond what was established by Ling [13]. For example, the expression for the conventional carry C_4 given in equations (19) can be further factorized as given below:

$$\begin{aligned} C_4 &= (g_3 + p_3g_2 + p_3p_2g_1 + p_3p_2p_1g_0 + p_3p_2p_1p_0C_{in}) \\ &= (p_3) \cdot (g_3 + g_2 + p_2g_1 + p_2p_1g_0 + p_2p_1p_0C_{in}) \\ &= (g_3 + p_3p_2) \cdot (g_3 + g_2 + g_1 + p_1g_0 + p_1p_0C_{in}) \\ &= (g_3 + p_3g_2 + p_3p_2p_1) \cdot (g_3 + g_2 + g_1 + g_0 + p_0C_{in}) \end{aligned} \tag{23}$$

It is readily observed that the second equality above is the Ling factorization and the quantity in the second factor is actually the Ling pseudo group carry H_4 defined in (18). The third and the fourth equalities are the higher order factorizations. The quantities in the second factors in these equalities may be called the Jackson-Talwar pseudo carries of order 3 and 4 respectively. For example, consider the case for a fourth order factorization. In this case, in order to understand the theory, we define two more local signals (b_i) and (d_i) as:

$$b_i = g_i + g_{i-1} + g_{i-2} + g_{i-3} \tag{24}$$

$$d_i = g_i + p_ig_{i-1} + p_ip_{i-1}p_{i-2} \tag{25}$$

Let us also define a Jackson-Talwar fourth order group carry J_i as given in equation (26) below:

$$J_i = C_i + C_{i-1} + C_{i-2} + C_{i-3} \tag{26}$$

Starting with this definition for J_{i+1}, we can define it in terms of J_{i-3} as given below:

$$\begin{aligned} J_{i+1} &= C_{i+1} + C_i + C_{i-1} + C_{i-2} \\ &= g_i + p_i \cdot C_i + C_i + C_{i-1} + C_{i-2} \\ &= g_i + C_i + C_{i-1} + C_{i-2} \\ &= g_i + g_{i-1} + p_{i-1} \cdot C_{i-1} + C_{i-1} + C_{i-2} \\ &= g_i + g_{i-1} + C_{i-1} + C_{i-2} \\ &= g_i + g_{i-1} + g_{i-2} + p_{i-2} \cdot C_{i-2} + C_{i-2} \\ &= g_i + g_{i-1} + g_{i-2} + C_{i-2} \\ &= g_i + g_{i-1} + g_{i-2} + g_{i-3} + p_{i-3} \cdot C_{i-3} \\ &= b_i + d_{i-3} \cdot J_{i-3} \end{aligned} \tag{27}$$

Thus the above equation and the definition of b_{i-3} can be collected together using the binary prefix operator ($\circ$) as:

$$\begin{pmatrix} J_{i+1} \\ d_{i-3} \end{pmatrix} = \begin{pmatrix} b_i \\ d_{i-3} \end{pmatrix} \circ \begin{pmatrix} J_{i-3} \\ 1 \end{pmatrix} \tag{28}$$

where $J_{in} = J_0 = C_0 = C_{in}$, $j_0 = g_0$, and $b_0 = p_0$ and $j_i = 0$ and $b_i = 1$ $\forall i < 0$.

The realisation for a 4-bit Jackson-Talwar adder based on the propagation of pseudo carries defined above is the same as that for a 4-bit novel adder proposed by us in Section III and shown in Fig. 3. The blocks labled as krt are the initial processing blocks for computing the signals d_i, b_i and t_i as defined above. The blocks labled as $reduce$ compute the group signals in treelike fashion. In this figure there

are no $reduce$ blocks as it is a realisation of a 4-bit adder only and there are 4 subtrees in case of Jackson-Talwar adders and also in case of our proposed adders. However, in case of adders with operand widths of more than 4-bit, corresponding subtrees with $reduce$ blocks are present. The blocks labled $carry$ compute the carry out signals as defined by equation (4) while the blocks labled sum compute the sum bits as defined by equation (5) respectively at each bit position.

F. Higher Valency Adders

The discussions in the above subsections are based on the assumption that the generate signals g_i and the propagate signals p_i are computed for each bit. There is no reason why group generate signal $g_{i\cdots j}$ and group propagate signal $p_{i\cdots j}$ as defined in equation (9) can not be computed for groups of adjacent bits in place of individual bits. These signals defined over such groups can then be reduced in similar treelike structures as discussed earlier. Such parallel prefix adders [17], [18] are known as higher valency adders. It is worth noting that a parallel prefix adder with valency 2 is not the same as a Ling adder or that with valency 4 is not the same as a Jackson-Talwar adder which are different and architecturally more efficient.

III. Proposed Novel Higher Order Recurrences and Corresponding Adders

In case of Ling adders [13], the generate signals g_i are combined as conjunctions and the propagate signals p_i are combined as disjunctions and given in equations (13) and (14) respectively. These combinations are, however, limited to only 2 adjacent signals. Arguably Jackson-Talwar adders [15] are motivated by the fact that more than 2 adjacent generate signals can be combined as conjunctions (Reduced Generate) and corresponding propagate signals (Hyper Propagate) can be calculated in such a way that the overall addition of multi-bit integers remains correct. In case more than 2 propagate signals are combined as disjunctions and the corresponding genarate signals are calculated in similar way to preserve the multi-bit addition semantics, one can create a new family of adders which when looked from the Dimitrakopolous-Nikolos [16] perspective, can be decoupled in more than 2 subtrees and are consequently faster and more efficient. The following subsection describes such a decoupling in 4 subtrees to introduce the concept and subsection III-B generalizes the concept at higher orders.

A. Order 4 Recurrence and Corresponding Adders

In line with the thought above, it is straight forward to show that the recurrence relation for parallel prefix carry computation can be simplified further if another pseudo carry $K_i = H_i + q_{i-2} \cdot H_{i-2} = C_i + C_{i-1} + p_{i-2} \cdot p_{i-3}(C_{i-2} + C_{i-3})$ is propogated in place of either the conventional carry C_i or the Ling pseudo carry H_i. Once these pseudo carries are known for all the bit positions, the conventional carries can be extracted from them by $C_i = p_{i-1} \cdot K_i$ as proved below:

$$\begin{aligned}
K_i &= C_i + C_{i-1} + q_{i-2} \cdot (C_{i-2} + C_{i-3}) \\
&= g_{i-1} + p_{i-1} \cdot C_{i-1} + C_{i-1} + q_{i-2} \cdot (C_{i-2} + C_{i-3}) \\
&= g_{i-1} + C_{i-1} + q_{i-2} \cdot (C_{i-2} + C_{i-3}) \\
&= g_{i-1} + g_{i-2} + p_{i-2} \cdot C_{i-2} + q_{i-2} \cdot C_{i-2} + q_{i-2} \cdot C_{i-3} \\
&= g_{i-1} + g_{i-2} + p_{i-2} \cdot g_{i-3} + p_{i-2} \cdot p_{i-3} \cdot C_{i-3} + q_{i-2} \\
&\qquad \cdot C_{i-3} \\
&= g_{i-1} + g_{i-2} + p_{i-2} \cdot g_{i-3} + q_{i-2} \cdot C_{i-3} \\
&= g_{i-1} + g_{i-2} + q_{i-2} \cdot g_{i-3} + q_{i-2} \cdot C_{i-3} \\
&= g_{i-1} + g_{i-2} + q_{i-2} \cdot g_{i-3} + q_{i-2} \cdot g_{i-4} + q_{i-2} \cdot p_{i-4} \\
&\qquad \cdot C_{i-4}
\end{aligned} \tag{29}$$

Therefore,

$$\begin{aligned}
p_{i-1} \cdot K_i &= p_{i-1} \cdot g_{i-1} + p_{i-1} \cdot g_{i-2} + p_{i-1} \cdot q_{i-2} \cdot g_{i-3} \\
&\quad p_{i-1} \cdot q_{i-2} \cdot g_{i-4} + p_{i-1} \cdot q_{i-2} \cdot p_{i-4} \cdot C_{i-4} \\
&= g_{i-1} + p_{i-1} \cdot g_{i-2} + p_{i-1} \cdot p_{i-2} \cdot g_{i-3} \\
&\quad + p_{i-1} \cdot p_{i-2} \cdot p_{i-3} \cdot g_{i-4} + p_{i-1} \cdot p_{i-2} \cdot p_{i-3} \\
&\quad \cdot p_{i-4} \cdot C_{i-4} \\
&= C_i
\end{aligned} \tag{30}$$

In order to develop recurrence relation for the above defined pseudo carry, we define two more local signals (k_i) and (r_i) as:

$$k_i = g_i + g_{i-1} + q_{i-1} \cdot (g_{i-2} + g_{i-3}) = h_i + q_{i-1} \cdot h_{i-2} \tag{31}$$

$$r_i = p_i \cdot p_{i-1} \cdot p_{i-2} \cdot p_{i-3} = q_i \cdot q_{i-2} \tag{32}$$

Using equation (29) for the definition of K_{i+1} we can define it in terms of K_{i-3} as given below:

$$\begin{aligned}
K_{i+1} &= g_i + g_{i-1} + q_{i-1} \cdot g_{i-2} + q_{i-1} \cdot g_{i-3} + q_{i-1} \cdot p_{i-3} \\
&\quad \cdot C_{i-3} \\
&= k_i + q_{i-1} \cdot p_{i-3} \cdot C_{i-3} \\
&= k_i + q_{i-1} \cdot p_{i-3} \cdot p_{i-4} \cdot K_{i-3} \\
&= k_i + r_{i-1} \cdot K_{i-3}
\end{aligned} \tag{33}$$

Thus the above equation and the definition of r_i can be collected together using the binary prefix operator ($\circ$) as:

$$\begin{pmatrix} K_{i+1} \\ r_{i-1} \end{pmatrix} = \begin{pmatrix} k_i \\ r_{i-1} \end{pmatrix} \circ \begin{pmatrix} K_{i-3} \\ 1 \end{pmatrix} \tag{34}$$

where $K_{in} = K_0 = H_0 = C_0 = C_{in}$, $k_0 = h_0 = g_0$ and $r_0 = q_0 = p_0$, $K_1 = H_1 = C_1 + C_0 = p_0 \cdot (g_0 + C_0) =$

$g_0 + p_0 \cdot C_0 = g_0 + p_0 \cdot C_{in}$, $k_1 = h_1 = g_1 + g_0$ and $r_1 = q_1 = p_1 \cdot p_0$, $K_2 = H_2 + q_0 \cdot H_0 = C_2 + C_1 + q_0 \cdot C_0 = g_1 + g_0 + p_0 \cdot C_0 = g_1 + g_0 + p_0 \cdot C_{in}$, $k_2 = h_2 + q_1 \cdot h_0 = g_2 + g_1 + p_1 \cdot g_0$ and $r_2 = q_2 \cdot q_0 = p_2 \cdot p_1 \cdot p_0$ and $k_i = 0$ and $r_i = 1 \quad \forall i < 0$.

Input new pseudo carry K_{i+1} at any bit position as a function of $K_{in} = H_{in} = C_{in}$ can thus be easily computed by a sequence of prefix operations as:

$$\begin{pmatrix} K_{i+1} \\ r_{i-1} \cdot r_{i-5} \cdots r_4 \cdot r_0 \end{pmatrix} = \begin{pmatrix} k_i \\ r_{i-1} \end{pmatrix} \circ \begin{pmatrix} k_{i-4} \\ r_{i-5} \end{pmatrix} \cdots \begin{pmatrix} k_1 \\ r_0 \end{pmatrix} \circ \begin{pmatrix} C_{in} \\ 1 \end{pmatrix} \tag{35}$$

It is readily observed that, using equation (33), the new pseudo carry K_4 can be written as:

$$\begin{aligned} K_4 &= k_3 + r_2 \cdot K_{in} \\ &= h_3 + q_2 \cdot h_1 + q_2 \cdot q_0 \cdot H_{in} \\ &= g_3 + g_2 + p_2 \cdot g_1 + p_2 \cdot p_1 \cdot g_0 + p_2 \cdot p_1 \cdot p_0 \cdot C_{in} \end{aligned} \tag{36}$$

which is logically same as that of the pseudo Ling carry H_4 and obviously simpler than the corresponding expression for the conventional carry C_4.

Striking similarity of the above relations with Ling recurrence relation and Weinberger-Smith recurrence relation is worth noting. It is, therefore, trivial to implement our new adders modelled as Ladner-Fischer [5], Brent-Kung [6] or any other topology as proposed by Knowles [8]. Following is an expansion of our new higher order recurrence relation for our new pseudo carries for a radix-8 adder.

$$\begin{pmatrix} K_8 \\ r_6 \cdot r_2 \end{pmatrix} = \begin{pmatrix} k_7 \\ r_6 \end{pmatrix} \circ \begin{pmatrix} k_3 \\ r_2 \end{pmatrix} \circ \begin{pmatrix} C_{in} \\ 1 \end{pmatrix}$$

$$\begin{pmatrix} K_7 \\ r_5 \cdot r_1 \end{pmatrix} = \begin{pmatrix} k_6 \\ r_5 \end{pmatrix} \circ \begin{pmatrix} k_2 \\ r_1 \end{pmatrix} \circ \begin{pmatrix} C_{in} \\ 1 \end{pmatrix}$$

$$\begin{pmatrix} K_6 \\ r_4 \cdot r_0 \end{pmatrix} = \begin{pmatrix} k_5 \\ r_4 \end{pmatrix} \circ \begin{pmatrix} k_1 \\ r_0 \end{pmatrix} \circ \begin{pmatrix} C_{in} \\ 1 \end{pmatrix}$$

$$\begin{pmatrix} K_5 \\ r_3 \end{pmatrix} = \begin{pmatrix} k_4 \\ r_3 \end{pmatrix} \circ \begin{pmatrix} k_0 \\ 1 \end{pmatrix} \circ \begin{pmatrix} C_{in} \\ 1 \end{pmatrix}$$

$$\begin{pmatrix} K_4 \\ r_2 \end{pmatrix} = \begin{pmatrix} k_3 \\ r_2 \end{pmatrix} \circ \begin{pmatrix} C_{in} \\ 1 \end{pmatrix}$$

$$\begin{pmatrix} K_3 \\ r_1 \end{pmatrix} = \begin{pmatrix} k_2 \\ r_1 \end{pmatrix} \circ \begin{pmatrix} C_{in} \\ 1 \end{pmatrix}$$

$$\begin{pmatrix} K_2 \\ r_0 \end{pmatrix} = \begin{pmatrix} k_1 \\ r_0 \end{pmatrix} \circ \begin{pmatrix} C_{in} \\ 1 \end{pmatrix}$$

$$\begin{pmatrix} K_1 \\ 1 \end{pmatrix} = \begin{pmatrix} k_0 \\ 1 \end{pmatrix} \circ \begin{pmatrix} C_{in} \\ 1 \end{pmatrix}$$

$$\begin{pmatrix} K_0 \\ 1 \end{pmatrix} = \begin{pmatrix} C_{in} \\ 1 \end{pmatrix}$$

It is readily observed in the above expansion that our new pseudo carries are dependent on their previous pseudo carries only with a stride of 4. This essentially means that they can be reduced in 4 mutually exclusive sub-trees. Therefore, the 8-input first stage of the 8 bit proposed adder is actually 4 separate 2-input stages, which eliminates 2 logic levels in the critical path!

The realisation for a 4-bit adder based on the propagation of pseudo carries defined by us above from an augmented Dimitrakopoulos and Nikolos [16] perspective is shown in Fig. 3. The blocks labled as krt are the initial processing blocks that compute the signals k_i, r_i and t_i as defined in equations (31), (32) and (3) respectively. The blocks labled as $reduce$ compute the group signals in treelike fashion. Obviously, as it is a realisation of a 4-bit adder, these $reduce$ blocks are not explicitly present. However, for adders with higher operand widths they will appear in treelike fashion. The blocks labled $carry$ compute the carry out signals as defined by equation (4) while the blocks labled sum compute the sum bits as defined by equation (5) respectively at each bit position.

B. Higher Order Recurrences

The generalization of the concept to higher levels is also straight forward. For example, if yet other pseudo carries $L_i = K_i + r_{i-2} \cdot K_{i-4} = H_i + q_{i-2} \cdot H_{i-2} + q_{i-2} \cdot q_{i-4} \cdot H_{i-4} + q_{i-2} \cdot q_{i-4} \cdot q_{i-6} \cdot H_{i-6} = C_i + C_{i-1} + p_{i-2} \cdot p_{i-3} \cdot (C_{i-2} + C_{i-3}) + p_{i-2} \cdot p_{i-3} \cdot p_{i-4} \cdot p_{i-5} \cdot (C_{i-4} + C_{i-5}) + p_{i-2} \cdot p_{i-3} \cdot p_{i-4} \cdot p_{i-5} \cdot p_{i-6} \cdot p_{i-7} \cdot (C_{i-6} + C_{i-7})$ are propogated in place of the pseudo carries K_i introduced above, the conventional carries can be extracted from them by $C_i = p_{i-1} \cdot L_i$ exactly as proved in equations (29) and (30) above for K_i.

In order to develop recurrence relation for the above defined pseudo carry L_i, we define two more local signals (l_i) and (s_i) as:

$$\begin{aligned} l_i = {} & g_i + g_{i-1} + q_{i-2} \cdot (g_{i-2} + g_{i-3}) \\ & + q_{i-2} \cdot q_{i-4} \cdot (g_{i-4} + g_{i-5}) \\ & + q_{i-2} \cdot q_{i-4} \cdot q_{i-6} \cdot (g_{i-6} + g_{i-7}) \end{aligned} \tag{37}$$

$$s_i = p_i \cdot p_{i-1} \cdot p_{i-2} \cdot p_{i-3} \cdot p_{i-4} \cdot p_{i-5} \cdot p_{i-6} \cdot p_{i-7} \tag{38}$$

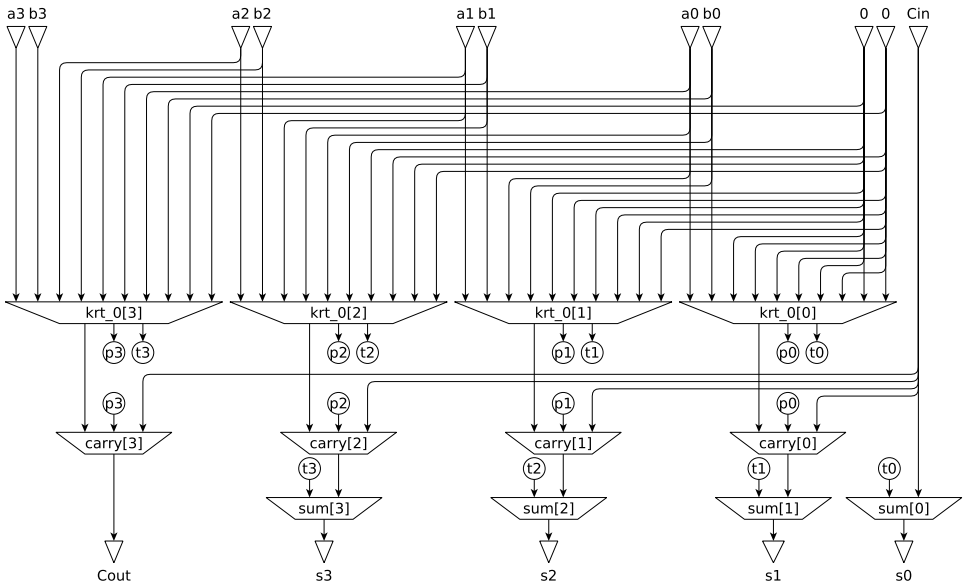

Figure 3. A 4-bit adder based on our recurrence in Ladner-Fischer topology.

Using the definition of L_{i+1} we can define it in terms of L_{i-7} as:

$$L_{i+1} = l_i + s_{i-1} \cdot L_{i-7} \tag{39}$$

Thus the above equation and the definition of s_i can be collected together using the binary prefix operator ($\circ$) as:

$$\begin{pmatrix} L_{i+1} \\ s_{i-1} \end{pmatrix} = \begin{pmatrix} l_i \\ s_{i-1} \end{pmatrix} \circ \begin{pmatrix} L_{i-7} \\ 1 \end{pmatrix} \tag{40}$$

where $L_{in} = L_0 = K_0 = H_0 = C_0 = C_{in}$ and $l_i = 0$ and $s_i = 1 \quad \forall i < 0$ as before.

The definitions of K_{i+1} and L_{i+1} as given in equations (34) and (40) have a striking similarity except that the former has a stride 4 while the later has a stride 8. Thus, if the pseudo carry L_i is propagated instead of the pseudo carry K_i, then the reduction tree for the adder based on it can be partitioned into 8 subtrees instead of 4 which takes our concept at a higher level. In fact the concept proved in this subsection can be recursively applied again and again to get 16 subtrees and even beyond subject only to the limitations of implementation technology.

IV. Experimental Results and Discussions

Individual structural blocks of the parallel prefix adders based on Weinberger-Smith [12], Ling [13], Jackson-Talwar [15] and our proposed recurrence relations (order 4) were synthesized using EDA tools at gate level with the available technology library which has synthesizable gates with a maximum fan-in of 4. As the intent of this paper is to propose an ultra fast adder at architecture level, we restrict our experiments at gate level alone and leave the layout details and post-layout simulations for extracting the exact speed and area for future work. The number of logic levels in the critical path and the total gate count are good estimates for operating speed and layout area respectively. These logic levels and gate counts were counted manually and are reported for 8-bit, 16-bit, 32-bit, 64-bit and 128-bit adders in Table I.

Table I
Estimates for Speed and Area of Various Parallel Prefix Adders.

Number of Bits	Recurrence Relations	Logic Levels in Critical Path	Number of Gates
8	Weinberger-Smith	11	107
	Ling	10	118
	Jackson-Talwar	9	212
	Proposed	9	164
16	Weinberger-Smith	13	259
	Ling	12	274
	Jackson-Talwar	11	460
	Proposed	11	364
32	Weinberger-Smith	15	611
	Ling	14	646
	Jackson-Talwar	13	1,004
	Proposed	13	812
64	Weinberger-Smith	17	1,411
	Ling	16	1,478
	Jackson-Talwar	15	2,188
	Proposed	15	1,804
128	Weinberger-Smith	19	3,203
	Ling	18	3,334
	Jackson-Talwar	17	5,260
	Proposed	17	3,980

The number of logic levels in the critical path for all the

adders based on Ling recurrence [13] are 1 less than the values for the corresponding adders based on Weinberger-Smith recurrence [12]. These levels in case of adders based on Jackson-Talwar recurrence [15] as well as those based on our proposed novel recurrence are still lower by 1, as expected. The total gate count for all the adders are increasing from Weinberger-Smith adder [12] to Ling adder [13] to Jackson-Talwar adder [15]. This trend is in line with the expectation.

The comparison between Jackson-Talwar adder [15] and our proposed adder is particularly interesting. Though the speeds achieved by both the adders is the same, yet the total gate count in case of our proposed adder is much lower as compared to the former. This is in line with the complexities expressed in equations (24), (31) and (25), (32) respectively. The order of recurrence presented in this paper is only 4. In case of recurrences at higher order the layout area for Jackson-Talwar adder [15] is expected to deteriorate at a higher rate than that for our proposal. In fact, the scalabilty achieved by the proposed adder family is much better than that of the Jackson-Talwar adders [15] both in terms of the operand widths as well as in terms of the order of factorization. The proposed adder family differs with other parallel prefix adders [12]–[15] only in the first stage, while the reduction tree topologies remain exactly the same. This fact highlights its generic nature as any tree reduction topology [4]–[11] is equally applicable. In the interest of consistency, however, we have chosen only Ladner-Fischer [5] topology.

V. Conclusions

Systematic and consistent development of the theory behind parallel prefix adders has revealed newer factorisations of logical expressions for computation of bitwise carries in parallel as well as more efficient recurrence relations. This study has demonstrated existence of better energy-area efficient and faster multi-bit integer adder architectures. The methodology proposed in this paper for higher order recurrences is shown to be recursive in nature. Though the present study has proposed novel factorisation of the order 4 in some detail, yet it is not difficult to extend the same theory to higher order factorisations as well.

It would also be worthwhile to study hybrid designs involving some of the architectures presented in this proposal in novel ways to achieve better results. The present study is limited in its scope at architecture level, which needs to be probed further at implementation levels. Both of these could be topics of future investigations.

Acknowledgment

The authors would like to thank MeitY, Govt. of India and Digital India Corporation (formerly Media Lab asia) for the Visweswaraya PhD scheme and supporting this R&D work.

References

[1] I. Koren, *Computer Arithmetic Algorithms*, 2nd ed. Natick, MA, USA: A. K. Peters, Ltd., 2001.

[2] B. Parhami, *Computer Arithmetic: Algorithms and Hardware Designs*. New York, NY, USA: Oxford University Press, Inc., 2000.

[3] M. D. Ercegovac and T. Lang, *Digital Arithmetic*. New York, NY, USA: Morgan Kaufmann, 2004.

[4] P. M. Kogge and H. S. Stone, "A parallel algorithm for the efficient solution of a general class of recurrence equations," *IEEE Transactions on Computers*, vol. C-22, no. 8, pp. 786–793, Aug 1973.

[5] R. E. Ladner and M. J. Fischer, "Parallel prefix computation," *J. ACM*, vol. 27, no. 4, pp. 831–838, Oct. 1980.

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

[7] T. Han and D. A. Carlson, "Fast area-efficient VLSI adders," in *1987 IEEE 8th Symposium on Computer Arithmetic (ARITH)*, May 1987, pp. 49–56.

[8] S. Knowles, "A family of adders," in *Proceedings IEEE Symposium on Computer Arithmetic. ARITH-15 2001*, 2001, pp. 277–281.

[9] A. Beaumont-Smith and C. C. Lim, "Parallel prefix adder design," in *Proceedings IEEE Symposium on Computer Arithmetic. ARITH-15 2001*, 2001, pp. 218–225.

[10] V. G. Oklobdzija, B. R. Zeydel, H. Dao, S. Mathew, and R. Krishnamurthy, "Energy-delay estimation technique for high-performance microprocessor VLSI adders," in *Proceedings 2003 IEEE Symposium on Computer Arithmetic*, June 2003, pp. 272–279.

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

[12] A. Weinberger and J. Smith, "A logic for high-speed addition," *Nat. Bur. Stand. Circ.*, vol. 591, pp. 3–12, 1958.

[13] H. Ling, "High-speed binary adder," *IBM Journal of Research and Development*, vol. 25, no. 3, pp. 156–166, March 1981.

[14] R. W. Doran, "Variants of an improved carry look-ahead adder," *IEEE Transactions on Computers*, vol. 37, no. 9, pp. 1110–1113, September 1988.

[15] R. Jackson and S. Talwar, "High speed binary addition," in *Conference Record of the Thirty-Eighth Asilomar Conference on Signals, Systems and Computers, 2004.*, vol. 2, Nov 2004, pp. 1350–1353 Vol.2.

[16] G. Dimitrakopoulos and D. Nikolos, "High-speed parallel-prefix VLSI Ling adders," *IEEE Transactions on Computers*, vol. 54, no. 2, pp. 225–231, Feb 2005.

[17] T. Kocak and P. Patil, "Design and implementation of high-performance high-valency Ling adders," in *2012 IEEE 15th International Symposium on Design and Diagnostics of Electronic Circuits Systems (DDECS)*, April 2012, pp. 224–229.

[18] T. McAuley, W. Koven, A. Carter, P. Ning, and D. M. Harris, "Implementation of a 64-bit Jackson adder," in *Signals, Systems and Computers, 2013 Asilomar Conference on*. IEEE, 2013, pp. 1149–1154.

[19] N. Poornima and V. K. Bhaaskaran, "Design and implementation of 32-bit high valency jackson adders," *Journal of Circuits, Systems and Computers*, vol. 26, no. 07, p. 1750123, 2017.