Area-time efficient end-around inverted carry adders

H.T. Vergos*

Computer Engineering & Informatics Department, University of Patras 26 500, Greece

1. Introduction

A number of algorithms met in a variety of applications ranging from random number generation and cryptography [1] up to convolution/correlation computation without rounding and truncation errors [2] rely on the use of modulo $2^n + 1$ arithmetic. A channel performing its operations in modulo $2^n + 1$ arithmetic is commonly met in a residue number system (RNS) [3–5] application. The RNS has been proposed as a faster to the binary representation alternative for the design of FIR filters [6], specialized digital signal processors [7] and communication components [8]. Three-moduli bases of the form $\{2^n - 1, 2^n, 2^n + 1\}$ have received significant attention for an RNS. Therefore, the design of efficient modulo $2^n + 1$ arithmetic components is vital for RNS-based applications.

For deriving efficient components for the modulo $2^n + 1$ arithmetic, several representations have been researched (for example the carry-save diminished-1 [9] and the stored unibit transfer [10]); the most well known are the normal weighted one and the diminished-1 [11] representation. Irrespective of the representation used, all these components require a two-operand adder which is mainly built on an end-around inverted carry (EAIC) adder. More specifically, it has been recently shown [12] that a two operand adder for the weighted representation can be designed efficiently by using an EAIC adder and a carry save adder stage, while a diminished-1 two operand adder requires an EAIC adder and few more logic gates for handling zero operands and results.

Since the direct connection of the inverted carry output to the carry input of an integer adder would create a combinational loop and lead to an unwanted race condition, efficient designs for EAIC adders have been the focus of several research efforts. Carry look-ahead (CLA) adder architectures that take into account the inverted output carry equation have appeared in [13]. In [14] modulo $2^n + 1$ adders for diminished-1 operands have been considered. These are actually EAIC adders with a parallel-prefix carry computation unit. They are based on extending the carry computation unit of an integer adder by an extra prefix level for handling the EAIC. In [13] it has been shown that the recirculation of the EAIC can be performed within the existing prefix levels of an integer adder. As a result the extra prefix level is no longer required and parallel-prefix EAIC adders have been offered that can operate as fast as their integer counterparts, that is, they offer a logic depth of $\log_2 n$ prefix levels. Unfortunately, this level of performance requires significantly more cell and interconnect area than the solutions of [14]. In [15] select-prefix EAIC adders have been proposed that aim at reducing the area complexity of the parallel-prefix solutions. These adders offer a lower operating speed than the parallel-prefix ones of [13], but within an implementation area close to that of the CLA ones [13] or to that of the parallel-prefix ones with the extra prefix level [14]. Finally, fast implementations for EAIC adders have appeared in [16] that rely on the use of Ling carries. However, they also require increased cell and interconnect area over the solutions of [14].

Keywords:
Modulo $2^n + 1$ arithmetic
Diminished-1 representation
Parallel-prefix carry computation

Abstract

Novel architectures for end-around inverted carry adders are proposed in this manuscript, which use a sparse carry computation unit for deriving only some of the carries in $\log_2 n$ prefix levels, while all the rest are computed in an extra one. When used for the design of modulo $2^n + 1$ adders, the proposed designs offer significant area and power savings compared to earlier proposals, while maintaining a high operation speed.

© 2011 Elsevier B.V. All rights reserved.
The rest of the manuscript is organized as follows. Some background issues on parallel-prefix carry computation in integer and EAIC adders are briefly revisited in the next Section. In Section 3 the proposed family of adders is presented. Quantitative comparison results against previous solutions are presented in Section 4.

2. Preliminaries

The symbols ∧, +, ⊕, and ⊙ are used in the following to denote logic AND, inclusive-OR, exclusive-OR and complement, respectively.

2.1. Parallel-prefix addition

The addition of $A = a_{n-1}a_{n-2} \ldots a_0a_0$ and $B = b_{n-1}b_{n-2} \ldots b_1b_0$ in an n-bit parallel adder can be considered as a three-stage process. During the first stage the carry generate, $g_i$, the carry propagate, $p_i$, and the half-sum $h_i$ bits are computed for every $l \leq l \leq n-1$, according to $g_i = a_i \cdot b_i$, $p_i = a_i + b_i$, and $h_i = a_i \oplus b_i$. Then, the second stage (also called carry computation unit), computes the carry signals, $c_i$, for $0 \leq l \leq n-1$, using as its inputs the carry generate and propagate bits. Finally, in the third stage the sum $S = s_{n-1}s_{n-2} \ldots s_0$ is computed by $s_i = h_i \oplus c_{i-1}$.

The prefix ($\lor$) operator [17] which associates pairs of generate and propagate signals by:

$$\langle g_m, p_m \rangle \lor \langle g_k, p_k \rangle = \langle g_m + p_m, g_k + p_k \rangle$$

allows to map carry computation into a prefix problem. The notation $(g_i, p_i)$ is commonly used to denote the group generate and propagate signals that result from a certain pair of consecutive generate/propagate pairs associations, that is:

$$(g_{i-1}, p_{i-1}) \lor \langle g_j, p_j \rangle = \langle g_{i-1} + p_{i-1}, g_j + p_j \rangle \lor \langle g_{i-1}, p_{i-1} \rangle (2)$$

Every carry $c_i$ in an integer adder is equal to $G_{g_i}$ a number of distinct algorithms have been introduced for computing all the carries using only $\lor$ operators. Such algorithms lead to a carry computation unit composed by interconnections of blocks implementing a prefix operator and are well-known as parallel-prefix carry computation units. These algorithms are most often represented by acyclic directed graphs in which the required $\lor$ operators constitute the black nodes.

2.2. Parallel-prefix EAIC addition

The EAIC adders proposed in [14] have a carry computation unit composed of $\log_2 n + 1$ levels. The first $\log_2 n$ prefix levels are those of an integer parallel-prefix adder and in [14] two different algorithms were considered for them; namely, the Ladner–Fischer (LF) [18] and the Kogge–Stone (KS) [19] algorithms. The Kogge–Stone architecture requires the largest number of prefix operators but has the smallest possible fanout, which is equal to 2. The low fanout property helps in achieving lower delay with the cost of additional power due to the increased number of operators. On the contrary, the Ladner–Fisher design prefers sharing the intermediate results, as much as possible, and thus requires the smallest number of prefix operators but suffers from high fanout lines that increase its delay compared to the Kogge–Stone architecture. These first $\log_2 n$ prefix levels may also be designed according to the hybrid architectures proposed by Knowles [20] that mix levels from the Kogge–Stone and the Ladner–Fisher architectures, to achieve intermediate solutions slightly slower but more area efficient than the Kogge–Stone proposal and slightly faster but requiring more area (and power) than the approach proposed by Ladner and Fisher. The last level, which is a late carry increment stage, uses the EAIC and the group generate and propagate signals already derived at each bit position.

Fig. 1(a) presents the proposal of [14] for a LF EAIC 16-bit adder. The adders proposed by [14] unfortunately suffer from a fan-out equal to $n$ at the EAC signal.

For n-bit modular and EAIC adders, the group generate and propagate pair of terms $(G_{g_k}, P_{p_k})$ can be defined even when $j > k$ in a circular manner, as:

$$(G_{g_k}, P_{p_k}) = (G_{g_0}, P_{p_0}, G_{g_{n-1}}, P_{p_{n-1}})$$

For the EAIC addition, parallel-prefix adders have been developed [13] by showing that the EAIC carries $c_i^1, 0 \leq i \leq n-2$, are equal to $G_{g_i}$, where $(G_{g_i}, P_{g_i})$ are computed by:

$$(G_{g_i^1}, P_{g_i^1}) = (G_{g_{i-1}}, P_{g_{i-1}}) \lor (G_{g_{n-i}}, P_{g_{n-i}})$$

and $c_i^1$ is equal to $G_{g_n}$. By definition, $(G, P)$ is equal to $(g, p)$. It should be noted that the above equations have a cyclic form and in contrast to integer addition, the number of generate and propagate pairs that have to be associated for each carry is equal to $n$. This means that a parallel-prefix carry computation unit of an EAIC adder has significantly increased area complexity than that of a corresponding integer adder. Fig. 1(b) presents the proposal of [13] for a 16-bit EAIC adder. The carry computation unit of the adders proposed in [13] have a carry computation unit composed of just $\log_2 n$ prefix levels. For implementing (3), for every $i$, within $\log_2 n$ prefix levels, in [13] a transformation method was proposed. For example, $c_i = (G_{g_{i+1}}, P_{g_{i+1}}) \lor (G_{g_{i+2}}, P_{g_{i+2}})$ is equivalently computed as $c_i = (G_{g_{i+1}}, P_{g_{i+1}}) \lor (G_{g_{i+2}}, P_{g_{i+2}})$, which unfortunately leads to a parallel-prefix computation unit that needs a double computation tree. One tree is used to associate generate and propagate signals in their normal form, while the second to associate the complemented form of them. This is indicated in Fig. 1(b) by the double operators required in some columns of the same prefix level.

By comparing Fig. 1(a) and (b) it becomes obvious that the increased speed of [13] comes at the penalty of heavily increased cell and interconnect area. The same observation holds for the full parallel prefix (FPP) and reduced area parallel prefix (RAPP) architectures proposed in [16] that follow a similar [13] prefix algorithm but rely on Ling carries. It should be noted that as we move to deeper sub-micron technologies, interconnect parasitics have a growing effect to the delay of a design. To this end, in the next section, novel architectures for EAIC adders are presented, with significantly reduced cell and interconnection area requirements.

3. Proposed EAIC adders

The proposed family of EAIC adders stems from considering the proposals of [14,13], as the two end cases of the number of EAIC addition carries that are computed within the first $\log_2 n$ prefix levels. In the first case, only one carry is computed within $\log_2 n$ prefix levels, while in the second case every carry is computed. The first choice leads to high fan-out, whereas the second leads to increased cell and interconnect requirements. The proposed new adders are derived considering the alternative of computing only some of the carries within the first $\log_2 n$ prefix levels. To this end, it is firstly shown that $c_{i+1}$ can be computed based on $c_i$.

Suppose that $c_i$ has been computed according to (3). It then holds that:

$$c_{i+1} = (c_i, c_i) \lor (G_{g_{i+1}}, P_{g_{i+1}})$$

and

$$c_{i+1} = (c_{i+1}, c_{i+1}) \lor (G_{g_{i+1}}, P_{g_{i+1}})$$

and

$$c_{i+1} = (c_{i+1}, c_{i+1}) \lor (G_{g_{i+1}}, P_{g_{i+1}})$$

The proposed family of EAIC adders stems from considering the proposals of [14,13], as the two end cases of the number of EAIC addition carries that are computed within the first $\log_2 n$ prefix levels. In the first case, only one carry is computed within $\log_2 n$ prefix levels, while in the second case every carry is computed. The first choice leads to high fan-out, whereas the second leads to increased cell and interconnect requirements. The proposed new adders are derived considering the alternative of computing only some of the carries within the first $\log_2 n$ prefix levels. To this end, it is firstly shown that $c_{i+1}$ can be computed based on $c_i$.
Fig. 1. The 16-bit EAIC adder of [14] (a) and [13] (b).
\[
\begin{align*}
&= (g_{i+1}, p_{i+1}) \circ (G_0, P_0) \circ (G_{n-1-i+2}, P_{n-1-i+2}) \\
&= (G_{i+1}, P_{i+1}) \circ (G_{n-1-i+2}, P_{n-1-i+2})
\end{align*}
\]

where the group generate term of the last relation is equal to \(c'_{i+1}\). That is, the next carry of the EAIC addition can be computed straightforwardly, by associating in a prefix operator the \((g_{i+1}, p_{i+1})\) pair of generate and propagate terms and the carry in the previous position. Since \(c'_{i+2}\) can be similarly computed using \(c'_{i+1}\), which as before can be computed based on \(c'_{i}\), it becomes obvious that we can compute every carry \(c'_{i+k}\) of the EAIC addition associating \(c'_{i}\) and the \((G_{i+k}, P_{i+k})\) pair of group generate and propagate terms in a prefix operator. Stated otherwise, while the architecture of [14] computes the EAIC addition carry at position \(i\) by the association of \((n+i)\ (g, p)\) terms and the architecture of [13] using \(n\) such terms, relation (4) reveals that we can use any number of \((g, p)\) between these two extremes.

As a result, we can compute in \(\log_2 n\) prefix levels any number of the EAIC addition carries and then use as many of them as we wish to compute the rest in a further prefix level. In this way a whole family of EAIC adders is derived. While all adders of the family have a carry computation unit composed of \(\log_2 n+1\) prefix levels, each member has its own fan-out requirements and cell and interconnection area. The notation Prop-\(k\) is used in the following, to denote the proposed adders in which \(k\) out of the total \(n\) carries of the EAIC addition are computed in the first \(\log_2 n\) prefix levels. Under this definition, the adders proposed in [14] are the Prop-1, while the adders of [13] are the Prop-\(n\) members of the family.

Fig. 2(a) and (b) presents the proposed Prop-\(n/2\) and Prop-\(n/4\) 16-bit EAIC adders. In the Prop-\(n/2\) adder case only the odd numbered carries are computed in \(\log_2 n\) prefix levels, while the even numbered ones in the last prefix level. This adder offers a fan-out equal to 2 and has a similar structure to the area-time efficient adders derived by the Han-Carlson algorithm [21] for integer addition. The Prop-\(n/4\) adder on the other hand has a fan-out equal to 4 but requires significantly less prefix operators along with their interconnections than the Prop-\(n/2\) adder. Considering the number of prefix operators as a qualitative metric of the area efficiency of each adder, it can be computed that the LF and KS proposals of [14] require 47 or 64 prefix operators, respectively, with a maximum fan-out equal to \(n\), the adders of [13] require 74 and the FPP and the RAPP proposals of [16] 68 and 52 prefix operators respectively, with a maximum fan-out equal.
to 2. The Prop-n/2 and Prop-n/4 explored in this manuscript require only 45 and 39 prefix operators and offer a maximum fan-out of 2 and 4, respectively. It is noted that the elimination of several prefix operators also removes their associated interconnections. As a result the interconnect area is also significantly reduced.

### 4. Comparisons

Since every EAIC adder with the addition of a few gates for handling zero operands and results can be used for diminished-1 modulo 2^n+1 addition, the Prop-n/2 and Prop-n/4 EAIC adders are first quantitatively compared against the EAIC adders proposed in [13–16]. For the adders of [14], a LF or a KS prefix structure is considered for the first log2 n prefix levels. Both the RAPP and the FPP architectures of the adders that use Ling carries [16] are also examined.

For attaining the comparison data, structural Verilog descriptions for adders of 4, 8, 16 or 32 bits were first generated. Each description was then mapped in a power characterized 90 nm implementation technology [22]. For the synthesis and mapping of the designs the Synopsys Design Compiler tool was used in its topographical operation mode. For achieving faster timing closure, in this mode floorplanning is done in parallel with synthesis and mapping and the design is annotated with actual wiring parasitics and fan-out capacitances coming directly from the floorplan of the design and not from a wire load model. Each adder’s input and output is assumed to be driven and drive the output and the input of a D flip flop of the same implementation library, respectively. A typical corner (1.2 V, 25 °C) was considered. For obtaining the power data, a simulation driven approach was followed. 2^16 random input vectors were applied at a 500 MHz frequency at each netlist for deriving the average power dissipation of each. The attained results for each adder are given in Table 1 under the Delay, Area and Power columns. Delay results are given in ps, area results in μm^2, and average power results in mW. The A × T^2 column compares the different architectures under the well-known area × time^2 metric. The values of this column are normalized with respect to the best offered by any architecture for a particular adder.

The derived experimental data reveal that one or both examined members of the proposed family of adders outperform the earlier proposals of [14,15] in delay, area and average power consumption terms. They also outperform the adders designed according to the RAPP architecture of [16] in all terms in the two widest examined cases. However, they can not reach the speed of the adders proposed in [13] and the ultimate speed of the FPP adders [16]. Unfortunately, in both these proposals, this level of speed performance is achieved at a very high area and average

### Table 1

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>n</td>
<td>Delay</td>
<td>Area</td>
<td>Power</td>
</tr>
<tr>
<td>4</td>
<td>299</td>
<td>768.37</td>
<td>0.245</td>
</tr>
<tr>
<td>8</td>
<td>371</td>
<td>1671.57</td>
<td>0.574</td>
</tr>
<tr>
<td>16</td>
<td>431</td>
<td>3805.50</td>
<td>1.198</td>
</tr>
<tr>
<td>32</td>
<td>509</td>
<td>8343.34</td>
<td>2.594</td>
</tr>
<tr>
<td>Ref. [16]</td>
<td>Ref. [16] RAPP</td>
<td>Prop-n/2</td>
<td>Prop-n/4</td>
</tr>
<tr>
<td>n</td>
<td>Delay</td>
<td>Area</td>
<td>Power</td>
</tr>
<tr>
<td>4</td>
<td>N/A</td>
<td>N/A</td>
<td>0.629</td>
</tr>
<tr>
<td>8</td>
<td>302</td>
<td>1952.76</td>
<td>0.629</td>
</tr>
<tr>
<td>16</td>
<td>370</td>
<td>5118.35</td>
<td>1.711</td>
</tr>
<tr>
<td>32</td>
<td>442</td>
<td>14986.88</td>
<td>5.354</td>
</tr>
</tbody>
</table>

**Fig. 3.** Area-time design space exploration of 16-bit EAIC adders.
power consumption price. More specifically, the totally parallel-prefix adders of [13] require from 21% up to 95% more implementation area and consume from 32% up to 134% more power than the proposed adders, while the FPP adders of [16] require from 37% up to 122% more implementation area and consume from 47% up to 157% more power. As a result, the proposed adders are the most efficient of all examined architectures when the \(A \times T_T\) is considered. Under this metric, the proposed adders are also more efficient than the LF and the KS adders of [14] by 29–54% and by 27–132% respectively. They also outperform the adders of [13] by up to 92%, the adders of [15] by up to 32% and the FPP and RAPP proposals of [16] by up to 102% and 93% respectively.

Since \(2^{16} + 1\) is the Fermat number with the most practical interest, for \(n = 16\), all above adder architectures were synthesized under several delay targets. The derived area-delay curves are plotted in Fig. 3. This area-time exploration reveals that both examined members of the proposed family of adders offer significantly smaller implementations than any previously proposed architecture at all delay targets larger or equal to 397 ps. The area savings offered range from 19.7% up to 57.3% depending on the architecture that the proposed adders are compared against and the delay targeted.

In the unifying architecture proposed in [12], a slightly modified EAIC adder is used along with a CSA stage for building a modulo \(2^n + 1\) adder for operands in the normal weighted representation. The results of Table 1 indicate that if the proposed EAIC adders are used as building blocks, the resulting weighted adders will also be more area-time efficient than those that include some other EAIC adder. Therefore, the weighted adders that result by using the proposed EAIC adders in the unifying architecture of [12] are only compared against the weighted adders of [23], which have been shown to be more efficient in both area and time terms than the earlier proposal of [24]. The comparison results given in Table 2, reveal that the weighted adders resulting from using the proposed EAIC adders are faster, smaller and consume less power on the average than the adders of [23] throughout the examined range. The savings offered are about 2% in operation speed and range from 11% up to 45% and from 15% up to 59% in the required implementation area and average power consumption, respectively.

### Table 2

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Delay</td>
<td>Area</td>
<td>Power</td>
</tr>
<tr>
<td>4</td>
<td>420</td>
<td>1009.68</td>
<td>0.277</td>
</tr>
<tr>
<td>8</td>
<td>484</td>
<td>2344.67</td>
<td>0.720</td>
</tr>
<tr>
<td>16</td>
<td>530</td>
<td>5709.32</td>
<td>1.902</td>
</tr>
<tr>
<td>32</td>
<td>623</td>
<td>14161.76</td>
<td>5.049</td>
</tr>
</tbody>
</table>

### References


**Haridimos T. Vergos** received his Diploma in Computer Engineering in 1991, and his Ph.D. in 1996, from the Department of Computer Engineering & Informatics of the University of Patras, Greece, where he currently holds an Associate Professor position. He was a member of Atmel Multimedia & Communications Group and worked on the development of the first IEEE 802.11 compliant wireless MAC processor. His research interests include computer arithmetic and architecture, dependable system architectures and low power design and test. Dr. Vergos holds one worldwide patent and has authored or coauthored more than 70 scientific papers.