Efficient Modulo Adder Architectures
Mohammed Mosin A. Junjwadkar and Virapax M. Chougala

Abstract--- In this manuscript we introduce parallel-prefix architecture for the design of modulo $2^n+1$ adder. The proposed architecture is built around a sparse carry computation unit that computes only some of the carries of the modulo $2^n+1$ addition. This sparse approach is enabled by the introduction of the inverted circular idempotency property of the parallel-prefix carry operator and its regularity and area efficiency are further enhanced by the introduction of a new prefix operator. The resulting diminished-1 adders can be implemented in smaller area and consume less power compared to all earlier proposals, while maintaining a high operation speed.

Index Terms--- Computer Arithmetic, Modulo $2^n+1$ Adders, Residue Number System, Diminished-1 Representation, Parallel-prefix Carry Computation

I. INTRODUCTION

Modulo arithmetic has been used in digital computing systems for many years. In particular, modulo $2^n+1$ arithmetic appears to play an important role in a variety of applications. Arithmetic modulo $2^n+1$ is commonly met in their residue number system (RNS) which is an arithmetic system well-suited to applications in which the operations are limited to addition, subtraction and multiplication. The RNS has been used for the design of digital signal processors [7], [10], [11], FIR filters [12], [13], [14] and communication components [15], [16], [17], offering in several cases apart from enhanced operation speed, low-power characteristics [18].

The complexity of a modulo $2^n+1$ arithmetic unit is determined by the representation chosen for the input operands. Three representations have been considered; namely, the normal weighted one, the diminished-1 [19] and the signed-LSB representations [20]. We only consider the first two representations in the following, since the adoption of the signed-LSB representation does not lead to more efficient circuits in delay or area terms. In every case, when performing arithmetic operations modulo $2^n+1$ the input operands and the results are limited between 0 and $2^n$.

In the normal weighted representation, each operand requires $n+1$ bits for its representation but only utilizes $2^n+1$ representations out of the $2^n+1$ that these can provide. A denser encoding of the input operands and simplified arithmetic operations modulo $2^n+1$ are offered by the diminished-1 representation. In the diminished-1 representation, $A$ is represented as $a_1 A^*$, where $a_1$ is a single bit, often called the zero indication bit, and $A^*$, is an $n$-bit vector, often called the number part. If $A > 0$, then $a_1 = 0$ and $A^* = A - 1$, whereas for $A = 0$; $a_1 = 1$, and $A^* = 0$. For example, the diminished-1 representation of $A = 9$ modulo 17 is 010002.

Mohammed Mosin A. Junjwadkar, Industrial Electronics, ECE Department, KLSVDRIT, Haliyal, India. E-mail: 2vd12lie12@gmail.com
Virapax M. Chougala, Assistant Professor, ECE Department, KLSVDRIT, Haliyal, India.

ISSN: 2349-6657 @ JSET
II. PRELIMINARIES

Suppose that \( A = A_{n-1}, A_{n-2}, \ldots, A_0 \) and \( B = B_{n-1}, B_{n-2}, \ldots, B_0 \) represent the two numbers to be added and \( S = S_{n-1}, S_{n-2}, \ldots, S_0 \) denotes their sum. An adder can be considered as a three-stage circuit. The preprocessing stage computes the carry-generate bits \( G_i \), the carry-propagate bits \( P_i \), and the half-sum bits \( H_i \), for every \( i, 0 \leq i \leq n-1 \), according to

\[
G_i = A_i \cdot B_i \quad P_i = A_i + B_i \quad H_i = A_i \oplus B_i,
\]

Where \( \cdot \), + and \( \oplus \) denote logical AND, OR, and exclusive-OR, respectively. The second stage of the adder, hereafter called the carry computation unit, computes the carry signals \( C_i \), for \( 0 \leq i \leq n-1 \) using the carry generate and carry propagate bits \( G_i \) and \( P_i \). The third stage computes the sum bits according to

\[
S_i = H_i \oplus C_{i-1}.
\]

Carry computation is transformed into a parallel prefix problem using the \( O \) operator, which associates pairs of generate and propagate signals and was defined in [35] as

\[
(G, P) \circ (G', P') = (G + P \cdot G', P \cdot P').
\]

In a series of associations of consecutive generate/propagate pairs \( (G, P) \), the notation \( (G_{k:j}, P_{k:j}) \), with \( k > j \), is used to denote the group generate/propagate term produced out of bits \( k, k-1, \ldots, j \), that is,

\[
(G_{k:j}, P_{k:j}) = (G_k, P_k) \circ (G_{k-1}, P_{k-1}) \circ \cdots \circ (G_j, P_j).
\]

Since every carry \( C_i = G_i:0 \), a number of algorithms have been introduced for computing all the carries using only \( O \) operators. Fig. 1 presents the most well-known approaches for the design of an 8-bit adder, while Fig. 2 depicts the logic-level implementation of the basic cells used throughout the paper.

For large word lengths, the design of sparse parallel prefix adders is preferred, since the wiring and area of the design are significantly reduced without sacrificing delay. The design of sparse adders relies on the use of a sparse parallel-prefix carry computation unit and carry-select (CS) blocks. Only the carries at the boundaries of the carry-select blocks are computed, saving considerable amount of area in the carry-computation unit [39]. A 32-bit adder with 4-bit sparseness is shown in Fig. 3a. The carry select block computes two sets of sum bits corresponding to the two possible values of the incoming carry. When the actual carry is computed, it selects the correct sum without any delay overhead. A possible logic-level implementation of a 4-bit carry-select block is shown in Fig. 3b.

![Fig. 1: Examples of 8-Bit Parallel-Prefix Structures for Integer Adders. (a) Kogge-Stone [36], (b) Ladner-Fischer [37], and (c) one Representative of the Knowles [38] Family of Adders](image1)

![Fig. 2: The Logic-Level Implementation of the Basic Cells Used in Parallel-Prefix Adders](image2)

III. MODULO 2n+1 ADDERS

Diminished-1 modulo 2\(^n+1\) addition is more complex since special care is required when at least one of the input operands is zero (100 . . . 0). The sum of a diminished-1 modulo adder is derived according to the following cases:

- When none of the input operands is zero \((a, b \neq 0)\) their number parts \( A' \) and \( B' \) are added modulo2\(^n+1\). This operation as discussed in the following, can be handled by an IEAC adder.
- When one of the two inputs is zero the result is equal to the nonzero operand.
- When both operands are zero, the result is zero.
In any case that the result is equal to zero (cases 1 or 3), the zero-indication bit of the sum needs to be set and the number part of the sum should be equal to the all-zero vector. According to the above, a true modulo addition in a diminished-1 adder is needed only in case 1, while in the other cases the sum is known in advance.

When none of the input operands is zero, \(a_x b_y \neq 1\), the number part of the diminished-1 sum is derived by the number parts \(A^*\) and \(B^*\) of the input operands as follows:

\[
S^+ = (A^* + B^*) \mod (2^n + 1)
\]

\[
= \begin{cases} 
(A^* + B^* + 1) \mod 2^n, & A^* + B^* < 2^n, \\
(A^* + B^*) \mod 2^n, & A^* + B^* \geq 2^n. 
\end{cases} 
\tag{3}
\]

Figure 3: Parallel Prefix Modulo 2\(^n\)+1 Adders: (a) Using an Additional Carry-Increment Stage and (b) Recirculating the End Around Carry within the Existing log2 n Prefix Levels

Equation (3) reveals that an IEAC adder can be used for providing the number part in this case. Fig. 3a [22], [23] presents the implementation of an IEAC adder by the addition of a carry increment stage to an integer parallel prefix adder.

**IV. NEW SPARSE MODULO 2\(n\)+1 ADDERS**

Sparse refer to the design of the adder where a carry select block will be used to obtain the output sum. The possible outputs of sum will be calculated and the output carry will select which output sum will be selected. The design is based at parallel-prefix adders. In the sparse carry computation unit for sparse modulo 2\(n\) +1 diminished adders some prefix operators are doubled up, since 2 carry computations need to be performed in parallel; one on normal propagate and generate signals, while the other on their complements. The problem gets worse when the input operands’ width is not a power of two. So there is still a lot of space for improvement. This problem is removed by introducing a new prefix operator and an even simpler carry computation unit. While calculating the carry for the spares adder we found that several operators double up since the operators is calculated in parallel. For larger adders, significantly more operators need to be doubled up, leading to increased area and wiring. To overcome this problem, we need a prefix operator that can associate the operation. We introduce a new operator, hereafter called gray operator.

![Fig. 4: Modulo 2\(^{16}\)+1 Diminished Adder Using a Sparse Carry Computation Unit](image1)

![Fig. 5: Sparse-4 Modulo 2\(^{16}\)+1 Diminished-1 Adder](image2)

![Fig. 6: Gray Prefix Operator; Notation and Implementation](image3)
Diminished-1 modulo $2n+1$ addition is not used for addition of zero operands. When the input operand is zero, it should be handled separately. Zero treatment leads to slow and area-consuming implementations.

Architecture of Zero handling unexceptional modulo adders will be designed and the performance evaluation by means of area, delay, power dissipation will be compared with existing modulo adder. The problem of designing an Extra hardware for handling Zero addition problem is recovered in our proposal by implementing a MUX based selection line is used to decide whether to perform addition in case of input zero.

![Fig. 7: Adder with Zero Handling](image)

**Table 1: Operation of Adder in Fig 7**

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>OR o/p</th>
<th>AND o/p</th>
<th>Select</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>B</td>
<td>B</td>
<td>0</td>
<td>0</td>
<td>B</td>
</tr>
<tr>
<td>A</td>
<td>0</td>
<td>A</td>
<td>0</td>
<td>0</td>
<td>A</td>
</tr>
<tr>
<td>A</td>
<td>B</td>
<td>AB</td>
<td>1</td>
<td></td>
<td>A+B mod</td>
</tr>
</tbody>
</table>

**V. APPLICATION (FIR FILTER)**

A Filter is frequency selective network, which is used to modify an input signal in order to facilitate further processing. Basically there are two types of filters-analog and digital. Digital Filters are widely used in different areas, because Digital filters have the potential to attain much better signal to noise ratio than analog filters. The digital filter performs noiseless mathematical operations at each intermediate step in the transform and their precise reproducibility allows design engineers to achieve performance levels that are difficult to obtain with analog filters Digital filters operate on numbers opposite to analog filters, which operates on voltages. The basic operation of digital filter is to take a sequence of input numbers and compute a different sequence of output numbers. There exists a range of different digital filters. FIR and IIR filters are the two common filter forms. A drawback of IIR filters is that the closed-form IIR designs are preliminary limited to low pass, band pass, and high pass filters, etc. secondly FIR filters can have precise linear phase. Also, in the case of FIR filters, closed-form design equations do not exist and the design problem for FIR filters is much more under control than the IIR design problem. A FIR filter is a filter structure that can be used to implement almost any sort of frequency response digitally. It is usually implemented by using a series of delays, multipliers, and adders to create the filter's output. The architecture of FIR filter is shown in Fig 8.

![Fig. 8: FIR Filter Architecture Canonical Form](image)

**VI. SIMULATION RESULTS**

![Simulation Result of Kogge stone adder in fig 1(a)](image)
VII. PERFORMANCE COMPARISON

The proposed multiplier is synthesized in Xilinx 13.1 ISE simulator to find optimized delay and area of different modulo adder architectures. The modulo adders are compared that can be seen below.

Table 2: Comparison of 8 bit Adders

<table>
<thead>
<tr>
<th>Adder Type</th>
<th>Delay (ns)</th>
<th>Slices</th>
<th>4 i/p LUT's</th>
<th>IO's</th>
<th>Bonded IOB's</th>
<th>Logic Levels</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kogge stone(8bit) fig 1(a)</td>
<td>9.959</td>
<td>11</td>
<td>20</td>
<td>24</td>
<td>24</td>
<td>7</td>
</tr>
<tr>
<td>Lander Fisher(8bit) fig 1(b)</td>
<td>11.003</td>
<td>8</td>
<td>14</td>
<td>24</td>
<td>24</td>
<td>8</td>
</tr>
<tr>
<td>Knowles(8bit) fig 1(c)</td>
<td>9.082</td>
<td>10</td>
<td>19</td>
<td>24</td>
<td>24</td>
<td>6</td>
</tr>
<tr>
<td>Parallel Prefix(8bit) fig 3(a)</td>
<td>14.506</td>
<td>19</td>
<td>34</td>
<td>24</td>
<td>24</td>
<td>6</td>
</tr>
<tr>
<td>Parallel Prefix(8bit) fig 3(b)</td>
<td>9.09</td>
<td>17</td>
<td>31</td>
<td>24</td>
<td>24</td>
<td>6</td>
</tr>
</tbody>
</table>

Table 3: Comparison of 16 bit Adders

<table>
<thead>
<tr>
<th>Adder Type</th>
<th>Delay (ns)</th>
<th>Slices</th>
<th>4 i/p LUT's</th>
<th>IO's</th>
<th>Bonded IOB’s</th>
<th>Logic Levels</th>
</tr>
</thead>
<tbody>
<tr>
<td>2^{16}+1 Diminished 16 bit</td>
<td>13.303</td>
<td>30</td>
<td>50</td>
<td>48</td>
<td>48</td>
<td>11</td>
</tr>
<tr>
<td>2^{16}+1 Sparse 4 16 bit</td>
<td>7.577</td>
<td>8</td>
<td>16</td>
<td>48</td>
<td>48</td>
<td>19</td>
</tr>
</tbody>
</table>

Table 4: Comparison of Filter Output

<table>
<thead>
<tr>
<th>Adder Type</th>
<th>Using216+1 Diminished16 bit</th>
<th>Using216+1 Sparse 4 16 bit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Delay</td>
<td>24.962ns</td>
<td>17.778ns</td>
</tr>
<tr>
<td>Slices</td>
<td>40</td>
<td>24</td>
</tr>
<tr>
<td>4 i/p LUT’s</td>
<td>70</td>
<td>46</td>
</tr>
<tr>
<td>IO’s</td>
<td>56</td>
<td>56</td>
</tr>
<tr>
<td>Bonded IOB’s</td>
<td>40</td>
<td>40</td>
</tr>
<tr>
<td>Logic levels</td>
<td>11</td>
<td>14</td>
</tr>
</tbody>
</table>

VIII. CONCLUSION

Efficient modulo 2n+1adders are appreciated in a variety of computer applications including all RNS implementations.

In this paper, a contribution is offered to the modulo 2n+1addition problem. A novel architecture has been proposed that uses a sparse totally regular parallel-prefix
carry computation unit. This architecture was derived by proving the inverted circular idem potency property of the parallel-prefix carry operator in modulo 2n+1addition and by introducing a new prefix operator that eliminates the need for a double computation tree in the earlier fastest proposals. The experimental results indicate that the proposed architecture heavily outperforms the earlier solutions in implementation area and power consumption, while offering a high execution rate. The drawback for the modulo adder is that it is not used for adding zero operands. The adder is proposed that will eliminate this drawback, and zero operands can be added effectively and fast. An FIR filter is implemented using the proposed adders.

REFERENCES


