Reduced Redundant Arithmetic Applied on Low Power Multiply-Accumulate Units

DAVID NEUHÄUSER
Friedrich Schiller University
Department of Computer Science
D-07737 Jena
GERMANY
david.neuhaeuser@uni-jena.de

EBERHARD ZEHENDNER
Friedrich Schiller University
Department of Computer Science
D-07737 Jena
GERMANY
nez@uni-jena.de

Abstract: We propose a new redundant approach on designing multiply-accumulate units for low power. State of the art implementations make use of redundant registers to obtain low delay times by moving any carry propagate adder out of the operation cycle. Our contribution is optimizing the level of redundancy by adjusting the size of the carry register. This optimization is performed by a VHDL generator, creating a carry save reduction tree meeting given delay constraints. This generator uses a delay-driven, list-based algorithm, optimized by synthesis timing results. With this method, the carry register and final carry propagate adder are shortened. Applying our reduced redundant approach to different sized twos complement multiply accumulate units under low power constraints, we gain delay savings up to 21 percent without increased area or power for 16 bit multiply-accumulate units and up to 6 percent for 32 bit multiply-accumulate units.

Key–Words: carry-save, multiply-accumulate, merged arithmetic, reduced redundancy, low power, two’s complement

1 Introduction and State-of-the-Art

Any operation that can be computed as a sum of at least three partial products can be efficiently implemented with carry-save arithmetic. Applications are, for instance, plain multiplication [20], fused multiply-add [4] and multiply-accumulate, digital filters [13], or cryptographic algorithms [8].

The most common building blocks in carry-save arithmetic are full adders, also known as (3,2) counters [5], and half adders, i.e. (2,2) counters. Although other reduction elements like [4:2] compressors [10, 14] or [5:2] compressors [9] may be more favorable in general, here we focus on reduction strategies using (3,2) counters to highlight the essential properties of our approach.

Computing a multiplication, multiply-add, multiply-accumulate, or digital filter operation can be seen as the task to reduce $n$ partial products to only two. These final two partial products can then be used as input to further carry-save operations or summed up to a binary number with any carry-propagate adder (CPA).

Several reduction schemes essentially based on (3,2) counters and with minimal height of the reduction tree have been devised in the past, see for instance [15], including the strategies of Wallace [21] and Dadda [5].

In this paper, we discuss multiply-accumulate units. To reduce the partial products, we apply a list-based approach discussed in [7], taking latencies of used components into account. This approach allows to shape the degree of redundancy within the final two partial products with respect to predefined latency constraints.

By changing the latency constraints and therefore the degree of redundancy within the resulting two partial products, our approach lets us not only control the shape of the resulting two partial products, but also the latency of the final CPA, the size of the carry accumulate registers, the total area required, and the power consumption of the resulting multiply-accumulate units.

The following section provides a short review of merged and redundant merged multiply accumulate designs. In Section 3 we discuss the partial product reduction strategy applied to our design. In Section 4 we introduce our reduced redundant approach for designing multiply-accumulate units.

In Section 5 we show synthesis results for these designs, and discuss expected area, latency, and power consumption. We conclude in Section 6, and suggest further possible improvements and areas of interest concerning carry-save based arithmetic.
Inner product calculations as for instance

\[ C = \sum_{i=1}^{n} A(i) \times B(i) \]  \hspace{1cm} (1)

can profit from using multiply-accumulate operations of the kind \( C(i + 1) = A(i) \times B(i) + C(i) \).

Any multiply-accumulate operation can be implemented as a merged arithmetic [17, 15]. The carry-save reduction tree of the multiplier is combined with the accumulator containing the stored \( C(i) \) as shown in Figure 1 (a). The resulting merged multiply-accumulate unit is called M-MAC. This merging technique is also used in single multiply-add operations as fused multiply-add [1].

Keeping the stored \( C(i) \) itself in a redundant form moves the slow CPA out of every multiply-accumulate cycle and results in a redundant merged multiply-accumulate unit, RM-MAC, as shown in Figure 1 (b).

### 3 Reducing Partial Products

The RM-MAC approach leads to a complex structure of partial products to be reduced.

As an example, we assume \( A(i) \) and \( B(i) \) to be 4 bit wide, and \( C(i) \) resp. \( C(i + 1) \) to be redundantly coded as carry and sum, each 8 bit wide. This leads to the task of reducing 4 partial products, having different weights, as well as a sum and a carry word, results from the previous multiplication or accumulation. In Figure 2 (a), bit positions 0 to 7 are shown, every circle denotes a bit to be reduced.

A Baugh-Wooley multiplication array is used for two's complement multiplication (proposed in [3], reviewed in [18]). "1" in a circle in Figure 2 (a) denotes the fixed value of "1", coming from the Baugh-Wooley method.

Bits in the leftmost column are most significant and signed (bold); they are removed for separated overflow detection and sign correction. The "1" at bit position 4 is treated as every other bit for now. When synthesizing the design, the full adder resp. half adder assigned to this bit will be automatically optimized.

The actual reduction task is shown in Figure 2 (b). "1"'s and signed bits are removed. Every bit contains its actual latency, which is "0" for all latched carry and sum bits and estimated 0.44 ns for every bit of \( A(i) \) and \( B(i) \). This is the latency of the partial product generator, experimentally derived from the synthesis CMOS library [6].

By applying full adders and half adders according to [7], we must get a result, consisting of at most 2 unreduced bits per bit position. Every input bit is marked with thick lines, two bits in the same column denote a half adder, three bits denote a full adder. Output bits are shown in the following figure, marked by a dotted line. Every application of a full or half adder increases the latency of the output bits, according to Table 1. These delays are derived from the synthesis CMOS library [6].

Figures 2 (c) - (f) show the first reduction steps, (g) - (k) the last, and (l) shows the reduced result.

### 4 Reduced Redundant Merged Multiply-Accumulate

Applying this reduction strategy to a more realistic example with \( A(i) \) and \( B(i) \) 16 bit wide, and \( C(i) \) resp. \( C(i + 1) \) to be redundantly coded as carry and sum, each 31 bit wide.
The latencies of the bits in the resulting two partial products after reduction, denoted as carry and sum, can be seen in Figure 3. Note that the maximum latency is 1.64 ns at bit position 18.

Looking at the estimated latencies of the redundant result bits, we notice that the least significant bits have less latency than the maximum latency. We can reduce least significant bits to binary representation as long as we don’t violate the maximum latency. We call this \textit{reduced redundancy}, therefore the design will be called reduced redundant merged multiply-accumulate unit, RRM-MAC.

Reducing redundancy up to and including bit number 8, the estimated maximum latency of 1.64 ns at carry bit 18 is not violated, as shown in Figure 4. By this method, a different structure of partial results is gained. 10 carry bits less have to be kept in a register, saving 16 percent of registers. The number of input bits of the final CPA is reduced by 32 percent.

At carry bit 9 and 10, local latency maxima of 1.59 ns are created. Trying to reduce the number of carry bits further, the former maximum latency of 1.64 ns is violated in carry bits 16 with 1.71 ns resp. 17...
Figure 5: RRM-MAC: Multiply-accumulate unit based on reduced redundant arithmetic.

This would introduce a latency increase of 4 percent compared to Figure 4, but save another carry register (1.9 percent) and shorten the final CPA by 4.8 percent.

This shows that carry bits can be reduced up to a certain point without violating any timing constraints. Reducing the number of carries even further brings a tradeoff between increased reduction tree latency and decreased latency and area of the final CPA.

Our reduction strategy differs significantly from the strategy proposed in [11] by considering full adder and half adder latencies. The RRM-MAC design is shown in Figure 5.

5 Synthesis Results

The example architectures are described in VHDL, where the VHDL reduction tree is generated by an VHDL code generator. For this purpose, the generator proposed in [12] has been improved by the reduction strategy presented in Section 4.

The VHDL RTL models are synthesized with Synopsys Design Compiler [19] and UMC 180 nm CMOS library [6]. To achieve low power designs, area constraints were set to 0 to gain the smallest designs possible. The final CPA was described in VHDL as “+”. Synopsys Design Compiler then used a DesignWare adder, which fits the given constraints best.

For a $16 \times 16$ bit RRM-MAC are the area, power and timing characteristics depending on the redundancy degree shown in Figures 6 through 8.

Figure 6: Area characteristics of a $16 \times 16$ bit RRM-MAC depending on the redundancy degree.

Figure 7: Power characteristics of a $16 \times 16$ bit RRM-MAC depending on the redundancy degree.

Figure 8: Timing characteristics of a $16 \times 16$ bit RRM-MAC depending on the redundancy degree.

In Figure 8 the multiply-accumulate cycle shows no increased latency by reducing up to 9 carry bits. Reducing 10 or more carry bits increases the multiply-accumulate cycle time, which is not desired.

The power consumption is for every level of reduction at about equal, see Figure 7 while the area slightly decreases, see Figure 6.

The computation in Equation 1 corresponds to $n - 1$ multiply-accumulate cycles yielding a result in carry sum representation, and one final multiply-
accumulate cycle that is followed by a recoding phase adding up the two remaining partial products. This is denoted as total in the Figures 6 through 8.

Figure 9 shows timing characteristics resulting from different cycle runs followed by one CPA recoding phase. The needed time to conduct the operations decreases up to 9 reduced carry bits and increased from 10 reduced carry bits.

Figure 10 shows the according power characteristics which are about equal for every level of reduction. The corresponding area characteristic can be found in Figure 6.

Table 2 shows the timing characteristics in percent for a 16 × 16 bit and a 32 × 32 bit RRM-MAC with 9 reduced carry bits normalized to 0 reduced bits.

<table>
<thead>
<tr>
<th>Reduced Carry Bits</th>
<th>Latency in ns</th>
<th>Power in mW</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>79</td>
<td>84</td>
</tr>
<tr>
<td>2</td>
<td>84</td>
<td>87</td>
</tr>
<tr>
<td>4</td>
<td>88</td>
<td>90</td>
</tr>
<tr>
<td>8</td>
<td>93</td>
<td>93</td>
</tr>
<tr>
<td>16</td>
<td>96</td>
<td>96</td>
</tr>
</tbody>
</table>

Table 2: Latency characteristics of 16 × 16 bit and a 32 × 32 bit RRM-MAC with 9 reduced carry bits normalized to 0 reduced bits.

6 Conclusion and Future Work

Applying our reduced redundant approach to different sized twos complement multiply accumulate units under low power constraints, we gain delay savings up to 21 percent without increased area or power for 16 bit multiply-accumulate units and up to 6 percent for 32 bit multiply-accumulate units.

These results clearly show that reduced redundancy in multiply-accumulate units brings strong timing advantages for few successive operations and fewer timing advantages for many successive operations while not altering area and power consumption.

We investigated 16 × 16 bit multiply-accumulate units and 32 × 32 bit MAC units. MAC units up to 128 × 128 should be investigated, too.

We investigated multiply-accumulate units with N × N bit inputs and 2N bit outputs. Designs with only N bit outputs are also very interesting, because the final CPA is reduced even more. For a 16 × 16 bit multiply-accumulate unit it consist of only 2 to 3 bits.

Comparing the reduced and non-reduced redundant approach should include other carry-save reduction elements, as (4,2)-counters and (5,2)-counters, as well as other redundant representations like signed binary [2]. Incorporating the option of increased pipelining by inserting registers between partial product generation and reduction tree as well as into the actual reduction tree, as done in [16], can be another future improvement to compare a wider variety of multiply-accumulate designs.

Looking beyond synthesis into implementation, wiring effects are much more important. Implementing the proposed designs seems worthwhile, too.

Comparing the noticed effects with further CMOS libraries like 90 nm or 65 nm should be also considered.
References:


