Improved Combined Binary/Decimal Fixed-Point Multipliers

Brian Hickmann and Michael Schulte
University of Wisconsin - Madison
Dept. of Electrical and Computer Engineering
Madison, WI 53706
brian@hickmann.us and schulte@engr.wisc.edu

Mark Erle
International Business Machines
6677 Sauterne Drive
Macungie, PA 18062
merle@us.ibm.com

Abstract— **Decimal multiplication is important in many commercial applications including banking, tax calculation, currency conversion, and other financial areas. This paper presents several combined binary/decimal fixed-point multipliers that use the BCD-4221 recoding for the decimal digits. This allows the use of binary carry-save hardware to perform decimal addition with a small correction. Our proposed designs contain several novel improvements over previously published designs. These include an improved reduction tree organization to reduce the area and delay of the multiplier and improved reduction tree components that leverage the redundant decimal encodings to help reduce delay. A novel split reduction tree architecture is also introduced that reduces the delay of the binary product with only a small increase in total area. Area and delay estimates are presented that show that the proposed designs have significant area improvements over separate binary and decimal multipliers while still maintaining similar latencies for both decimal and binary operations.**

I. Introduction

Decimal arithmetic is necessary in many financial and commercial applications that process decimal values and perform decimal rounding. However, current software implementations have non-trivial performance penalties when compared against hardware implementations [1], prompting hardware manufacturers such as IBM to add or investigate adding decimal floating-point arithmetic support to microprocessors [2]. In addition, due to the importance of decimal arithmetic, it has been added to the revised IEEE 754 Standard for Floating-Point Arithmetic (IEEE 754-2008) [3].

A frequent and fundamental operation for any hardware implementation of decimal floating-point is multiplication and hence a low latency implementation is desired for high performance. However, to reduce the cost of supporting decimal arithmetic, a low area design is also a priority. Several previous decimal multipliers have primarily focused on binary coded decimal (BCD) fixed-point multiplication. Designs including [4], [5], [6], [7], [8] use a sequential approach of iterating over the digits of the multiplier and selecting an appropriate multiple of the multiplicand. Generally, these designs have high latency and low throughput due to their sequential approach, but they also have the reduced area.

A few parallel BCD fixed-point multiplier designs have also been proposed [9], [10]. These parallel decimal multipliers generate a sufficient set of multiplicand multiples and then select all of the partial products in parallel based on the digits of the multiplier operand. These designs offer the high performance that is desired, but also have significant area overheads due to their parallel nature. The only published parallel decimal multiplier design that provides high performance with a reasonably low area overhead is a combined binary/decimal multiplier design presented in [9]. This multiplier allows the decimal reduction tree and final carry-propagate adder to be shared with binary multiplication by using novel decimal digit encodings and a fast combined adder presented in [11]. The primary disadvantage of this design is the latency penalty that is placed on binary multiplication.

This paper presents three parallel binary/decimal fixed-point multipliers that offer enhancements to a combined multiplier design presented in [9]. The first of the proposed multipliers features an improved reduction tree that does not use binary/decimal 4:2 Carry-Save Adders (CSAs) during reduction. This improves the delay and especially the area of the multiplier. Another enhancement in our proposed design over the previous design in [9] is the use of improved binary/decimal doubling units that leverage the flexibility of the redundant decimal digit encodings to reduce their delay. The final two proposed multipliers incorporate the above two enhancements along with a reduction tree design that uses split binary and decimal outputs to significantly reduce the latency of the binary multiplication with a minor area penalty.

The combined multiplier designs presented in this paper are for 16-digit decimal multiplication using the Densely Packed Decimal (DPD) encoding format from the IEEE 754-2008 standard and either 64-bit or 53-bit binary multiplication, since these sizes are useful in the design of IEEE 754-2008 compliant double-precision floating-point multipliers. However, the techniques presented in the paper can be extended to other multiplication sizes. Compared to using separate binary and decimal multipliers, one of our proposed designs offers a 43% area savings while maintaining the same latency as the original multipliers. In addition, as compared to the combined design presented in [9], our split tree multiplier obtains improved binary multiplication latency while also obtaining up to a 25% area reduction.

The outline of the paper is as follows. Section II gives background information on decimal multiplication. Section III contains an in-depth description of a previous combined multiplier design from [9] on which our improvements are based. Section IV describes our improvements and presents three improved designs. Section V presents results, including

TABLE I

BCD-8421 AND BCD-4221 CODINGS

Decimal	BCD-8421	BCD-4221
0	0000	0000
1	0001	0001
2	0010	0010 or 0100
3	0011	0011 or 0101
4	0100	1000 or 0110
5	0101	1001 or 0111
6	0110	1010 or 1100
7	0111	1011 or 1101
8	1000	1110
9	1001	1111

area and delay estimates for the proposed designs and several other multiplier designs. This is followed by a summary in Section VI.

II. DECIMAL MULTIPLICATION BACKGROUND

Unsigned decimal multiplication performs the computation $P = A \times B$, where A is the multiplicand, B is the multiplier, and P is the product. It is assumed that A and B are each p_{dec} Binary Coded Decimal (BCD) digits and P is $2 \times p_{dec}$ BCD digits. As with binary multiplication, decimal multiplication consists of three primary stages: generation of partial products, fast addition or reduction of these partial products, and a final carry propagate addition (CPA). Decimal multiplication is much more complex, however, due to the higher range of decimal digits (0-9), which increases the number of multiplicand multiples that must be generated to form the partial products, and the inefficiency of adding or reducing BCD partial products.

To address the first complexity of generating multiplicand multiples for partial products, many recent designs use either a reduced set of stored multiplicand multiples or a signed-digit recoding to decrease the required number of multiplicand multiples that must be generated. Proposed methods presented in [7], [9], [10] for using a reduced set of multiples can significantly decrease the delay for generating the partial products and the area needed to store the multiples. This can be further enhanced by using multiples from the set (1A, 2A, 4A, 5A, 8A, 10A), which can be generated without the need for carry propagation as presented in [7]. The use of a reduced set of multiples often doubles the number of required partial products to $2 \times p_{dec}$, however. An alternate method, as proposed in [9], [12], uses a signed digit recoding such as [-5, 5]. This recoding requires that only the (1A...5A) multiples and their inverses be generated, but it includes a 3A multiple that needs a carry propagate addition. The advantage of this method is that only $p_{dec} + 1$ partial products are needed.

To reduce the second complexity of decimal addition with BCD operands, several methods have been proposed, including using binary adders with correction logic [13] and direct decimal addition [14], [7]. In this paper, however, we use binary CSAs to sum decimal numbers using alternate BCD encodings. In [9], a reduction method is proposed for decimal multiplication that uses binary CSAs and the

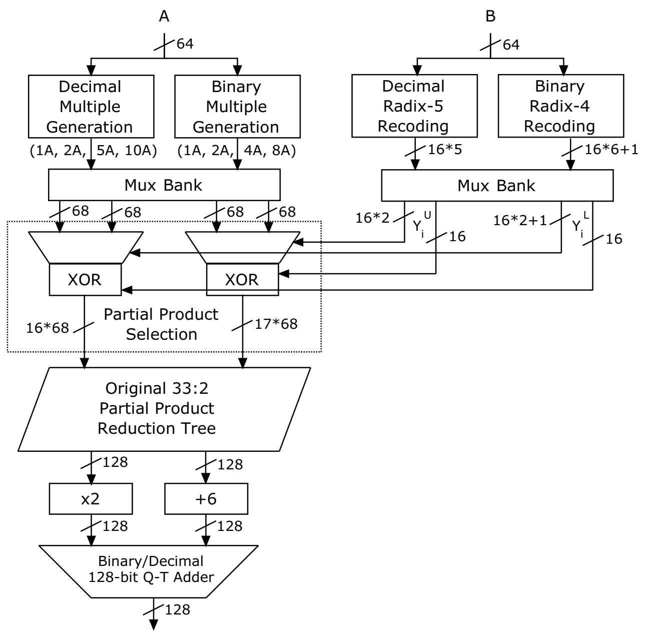


Fig. 1. Original Binary Radix-4/Decimal Radix-5 Combined Multiplier [9]

BCD-4221 encoding[1] as shown in Table I. The use of the BCD-4221 encoding has three advantages. First, since all 16 possible BCD-4221 encoding values are valid decimal digits, regular 4-bit binary CSAs can be used to perform the decimal addition as long as decimal doubling units (x2 units in [9]) are used to correct the carry digit. Second, BCD-4221 allows for efficient decimal doubling, requiring only a two-level logic function to perform this operation. Third, BCD-4221 is self-complementing, which allows the 9's complement to be computed with a simple bitwise inversion. This can be advantageous when using signed digit recodings, which require the inverses of the multiplicand multiples.

III. PREVIOUS COMBINED BINARY/DECIMAL MULTIPLIER DESIGN

The proposed combined binary/decimal multipliers presented in this paper are based on one of the combined multiplier designs presented in [9]. The multipliers presented in [9] use special BCD digit recodings to reduce the logic needed to perform decimal multiplication. While two combined multiplier architectures are presented in [9], we focus on extending the binary radix-4/decimal radix-5 multiplier. In this multiplier, the binary operands are recoded into signed radix-4 values using the standard modified Booth recoding [15] and the decimal operands are recoded into signed radix-5 values. We chose to extend this binary radix-4/decimal radix-5 combined multiplier design over the binary radix-4/decimal radix-4 design proposed in [9] because of its lower latency decimal multiple generation process. Also, in

[1]In this paper, we represent alternate decimal encodings in the format BCD-xxxx where the x's are the weights of each binary bit. For example, 1001_2 has a value of $4+0+0+1=5$ with the BCD-4221 encoding and a value of $8+0+0+1=9$ with the BCD-8421 encoding.

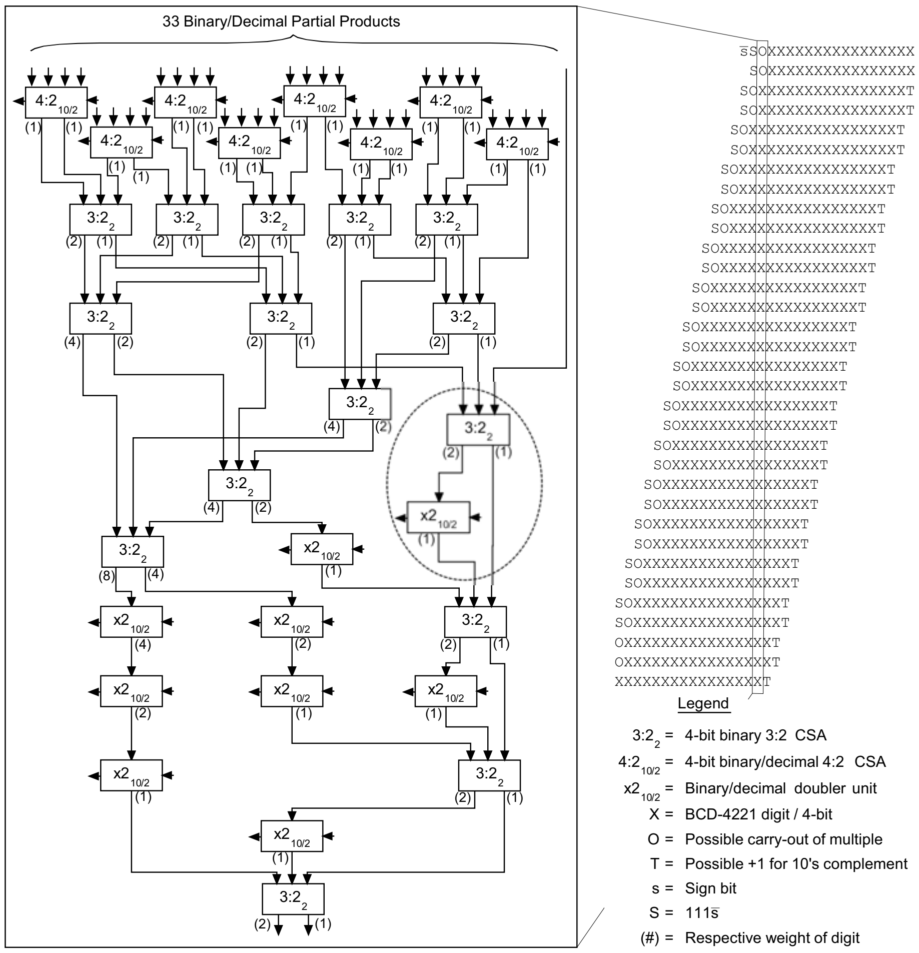

33 Binary/Decimal Partial Products
Legend
$3:2_2$ = 4-bit binary 3:2 CSA
$4:2_{10/2}$ = 4-bit binary/decimal 4:2 CSA
$x2_{10/2}$ = Binary/decimal doubler unit
X = BCD-4221 digit / 4-bit
O = Possible carry-out of multiple
T = Possible +1 for 10's complement
s = Sign bit
S = $111\bar{s}$
(#) = Respective weight of digit

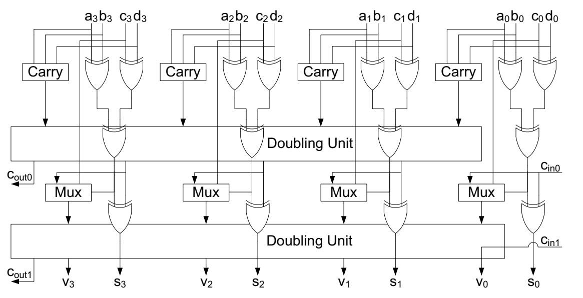
Fig. 2. Original Partial Product Reduction Tree and Alignment [9]

the decimal radix-4 recoding, an additional decimal partial product is required, which may increase the area and delay of the decimal multiplication.

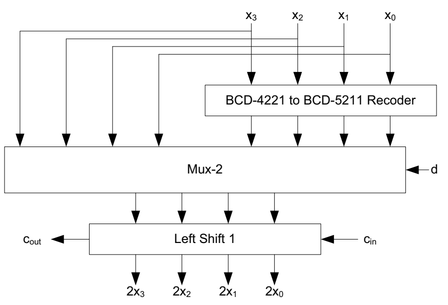
The combined binary radix-4/decimal radix-5 multiplier from [9] is pictured in Figure 1 for $p_{dec} = 16$ digits. It consists of five main components: generation of multiplicand multiples, recoding of the multiplier operand, partial product selection, partial product reduction, and a final carry propagate addition (CPA) to produce the non-redundant result. The partial product selection and reduction stages, along with the final CPA, are shared between binary and decimal operations while the multiplicand multiple generation and multiplier recoding are separate for each operation with multiplexers selecting the correct result based on the current operation type; binary or decimal.

To begin a multiplication, multiplicand multiple generation and multiplier recoding operate in parallel. Separate binary and decimal multiplicand multiples are generated. The binary portion generates the (1*A*, 2*A*, 4*A*, 8*A*) multiples using simple wired shifts. The decimal portion generates the (1*A*, 2*A*, 5*A*, 10*A*) multiples in a novel way using wired shifts and digit recodings [9]. All decimal multiples are encoded in BCD-4221 to allow the reduction tree to use binary CSAs even when performing decimal addition. A set of 2-to-1 multiplexers following the multiple generation selects between the binary and decimal multiples. The complements of both the binary and decimal multiples are also required and, since BCD-4221 is a self-complimenting code, this is done during partial product selection through simple bitwise inversion.

The multiplier digits are recoded for both the binary and decimal case in groups of four bits or one digit, respectively. For binary, the normal modified Booth recoding is performed on overlapping groups of three bits to produce signed digits with values of {-2, -1, 0, 1, 2}. In order to share the partial product selection multiplexers with decimal recoding that examines four bits (i.e. one digit) at a time, the binary recoding logically performs two modified Booth recodings on overlapping groups of five bits. This produces two signed digits with values of {-2, -1, 0, 1, 2} and {-8, -4, 0, 4, 8} respectively. The decimal recoding proceeds similarly but examines a single input digit during recoding to produce two output signed digits with values of {-2, -1, 0, 1, 2} and {0, 5, 10}. The two signed digits from both recodings are encoded into a one-hot form with a single sign bit and two selection bits, represented by Y_i^L and Y_i^U where *L* and *U* represent the



(a) Binary/Decimal 4:2 CSA



(b) Binary/Decimal Doubling Unit

Fig. 3. Reduction Tree Components [9]

Lower and Upper recoded digits, respectively, for the i^{th} digit or four bit group. A set of 2-to-1 multiplexers selects between the binary and decimal recodings to produce a single set of Y_i^L and Y_i^U values that are used to perform partial product selection.

As presented in [9], it is important to note that the above recoding process will produce $2 \times p_{dec}$ partial products for both the binary and decimal case. However, when multiplying 64-bit unsigned binary numbers, this gives an incorrect result if the last recoded sign digit is negative. This case requires that an additional partial product of $1x$ be added to the reduction tree, resulting in $2 \times p_{dec} + 1$ partial products. This correction is discussed more thoroughly in Section IV.

Next, partial product selection is performed using the Y_i^L and Y_i^U values from the multiplier recoding and either the binary or decimal multiples from the multiplicand multiple generation unit. A set of two one-hot multiplexers per input digit or four bit group creates two partial products that are reduced in the partial product reduction tree. Following each set of one-hot multiplexers is an XOR gate that uses the sign bits from the recoding to perform conditional inversions of the multiples when needed.

After selection is performed, the partial products are aligned and then sent to the partial product reduction tree. Due to the alignment process, the number of partial products that must be accumulated in any one column ranges from 4 to $2 \times p_{dec} + 1$ as shown on the right-side of Figure 2 for $p_{dec} = 16$. During the alignment process, additional bits or digits are added in order to correctly handle the sign extension of the partial products as also shown in Figure 2. A single worst-case column of the reduction tree with 33 input digits is shown on the left side of Figure 2. In this figure, we use the subscript "10/2" to indicate the circuit contains additional logic to correctly generate both binary and decimal outputs while a subscript of "2" or "10" indicates that the circuit has purely binary or decimal logic, respectively. This reduction tree has been slightly modified from the original tree in [9] to account for the additional binary partial product. The small correction added to handle the extra partial product is highlighted with a dotted circle. The reduction tree is made up of 4-bit binary 3:2 CSAs, 4-bit binary/decimal doubling units that contain logic to perform decimal digit doubling in the decimal case and a simple left shift in the binary case, and combined binary/decimal 4:2 CSAs. The doubling and 4:2 CSA units are the same as those from [9] and are pictured in Figure 3. In this figure, the d signal indicates whether the current operation is decimal ($d = 1$) or binary ($d = 0$).

Finally, after the partial products have been reduced, the result must be converted into non-redundant form. This is done with a 128-bit/32-digit combined binary/decimal conditional speculative quaternary tree adder [11]. This adder is based on a sparse quaternary tree presented in [16] that generates only every fourth carry. In parallel with the carry tree, a carry-select adder generates the intermediate 4-bit sums for both a carry-in of one and a carry-in of zero. A final multiplexer uses the results of the carry tree to select the results. To perform decimal addition, six is speculatively added to the digits of one of the operands, and corrective logic in the carry select adders coerces the output to the correct value in case of a mis-speculation.

IV. Improved Combined Binary/Decimal Multiplier Designs

In this section, we present several novel improvements over the multiplier design presented in Section III. These improvements aim to improve the delay and especially the area of the multiplier. Another goal is to reduce the latency of the binary multiplication so that it is not penalized as compared to a standalone binary multiplier. The improvements include an improved reduction tree that does not use 4:2 CSAs during reduction. The 4:2 CSAs presented in [9] use two binary/decimal doubling units which include multiplexers that can significantly add to the delay and area of the multiplier. We also extend the tree to handle reducing the necessary $2 \times p_{dec} + 1$ partial products to correctly handle unsigned binary operands. Another improvement over the previous design is the use of improved binary/decimal doubling units that use the flexibility of the redundant BCD-4221 encoding to improve the speed of the doubling units. Finally, we present a new reduction tree design with split binary and decimal outputs. This design avoids having the additional multiplexers needed in the original design to share the doubling units. The result is that the latency of the binary multiplication is significantly reduced with only a reasonable area penalty. Each of these improvements are discussed in more detail in the following subsections.

We present three separate improved designs that combine the above improvements. The first improved design uses

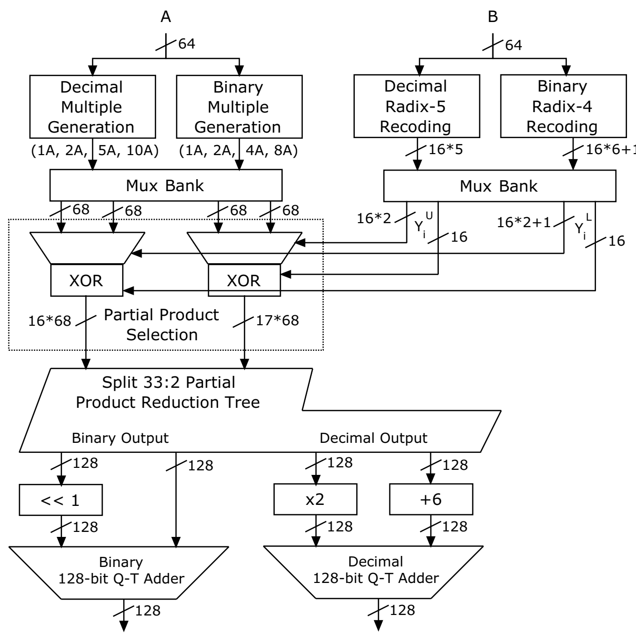


Fig. 4. Proposed Split Binary/Decimal Multiplier

the improved binary/decimal doubling units and improved reduction tree in an organization that is similar to the combined binary/decimal multipliers presented in [9]. The original method for the generation of multiplicand multiples, multiplier recoding, and partial product selection from [9] are used in this design unchanged due to their efficiency. The high level diagram of this improved design is similar to Figure 1 except for the use of an improved reduction tree. A 128-bit version of the combined binary/decimal conditional speculative quaternary tree adder from [11] is used to perform the final carry-propagate addition. While we investigated other adder designs, to our knowledge this is the fastest combined binary/decimal adder, and hence we use it unchanged.

The second design, pictured in Figure 4, replaces the reduction tree with a split reduction tree design. This design significantly improves the latency of the binary multiplication and is explained in Section IV-C. The split tree results in two separate outputs: a binary output and a decimal output. In this design we use two separate adders, one for the binary path and one for the decimal path, to generate separate non-redundant results. The two adders are the binary only and decimal only versions, respectively, of the conditional speculative quaternary tree adder from [11]. While using two adders increases the area of the design, it also reduces the delay of both the binary and decimal outputs since the adders are optimized for each operation. This design also allows a new binary or decimal multiplication to be started each cycle in a pipelined design without the need to wait for the pipeline to empty, as is true with the original design in [9] and the first improved design described above.If lower area is desired and the above property is not critical, then the combined binary/decimal adder from the first design may be used in place of the separate adders.

The final proposed design is similar to the previous split tree design but shrinks the design to be a 53-bit/16-digit multiplier. This design point is significant because 53-bits and 16-digits are the lengths of the significands of double precision binary and decimal floating-point numbers, respectively, from the IEEE 754-2008 standard [3]. The reduced binary width has several advantages that can be exploited in the design of the split reduction tree as described in Section IV-C. This also allows for the use of a smaller 106-bit binary CPA to further reduce the latency of the binary multiplication. The overall layout of the multiplier is very similar to the previous split design pictured in Figure 4, but the binary CPA is reduced to 106-bits and the tree is reorganized as described in Section IV-C.

A. *Improved Reduction Tree*

A single worst-case column of the proposed improved standard reduction tree for $p_{dec} = 16$ is illustrated in Figure 5. The primary advantage of this improved tree over the one shown in Figure 2 is the removal of the leading combined binary/decimal 4:2 CSAs at the top of the tree. As pictured in Figure 3, each binary/decimal 4:2 compressor contains two binary/decimal doubling units. In a combined binary/decimal multiplier, these doubling units represent a significant overhead inside the reduction tree due to the multiplexers needed to select between the binary wired shift and the decimal digit recoding logic, depending on the current operation. To reduce this overhead, we present an improved reduction tree that uses only binary 3:2 CSAs and binary/decimal doubling units. It is organized in a manner similar to that of the reduction tree from [9], but reduces the number of doubling units needed as additional calculations can be performed before X2 units are needed. A single worst-case column of the proposed reduction tree only has 16 doubling units as compared to the original's 25 doubling units.

In addition to the reduction in the number of doubling units, the proposed tree also correctly handles the $2 \times p_{dec} + 1$ partial products that may be needed in the binary case. The additional partial product arises from the fact that the modified Booth recoding used to recode the multiplier operand during binary multiplication is designed to work with signed input operands [15]. When the multiplier has a one in the most significant bit, the modified Booth recoding will select a negative multiple and produce the incorrect result. In order to apply this recoding to an unsigned operand, an implicit zero must be prepended to the input operand, resulting in $2 \times p_{dec} + 1$ partial products from the Booth recoding [17]. The correction added to the largest columns in the reduction tree can be found in the dashed circles in Figure 2 and Figure 5. For these figures, $p_{dec} = 16$ and hence, the trees must correctly reduce 33 partial products. In both cases, the addition of the extra partial product simply introduces a single extra 3:2 CSA to the reduction tree and does not significantly impact the delay of the tree.

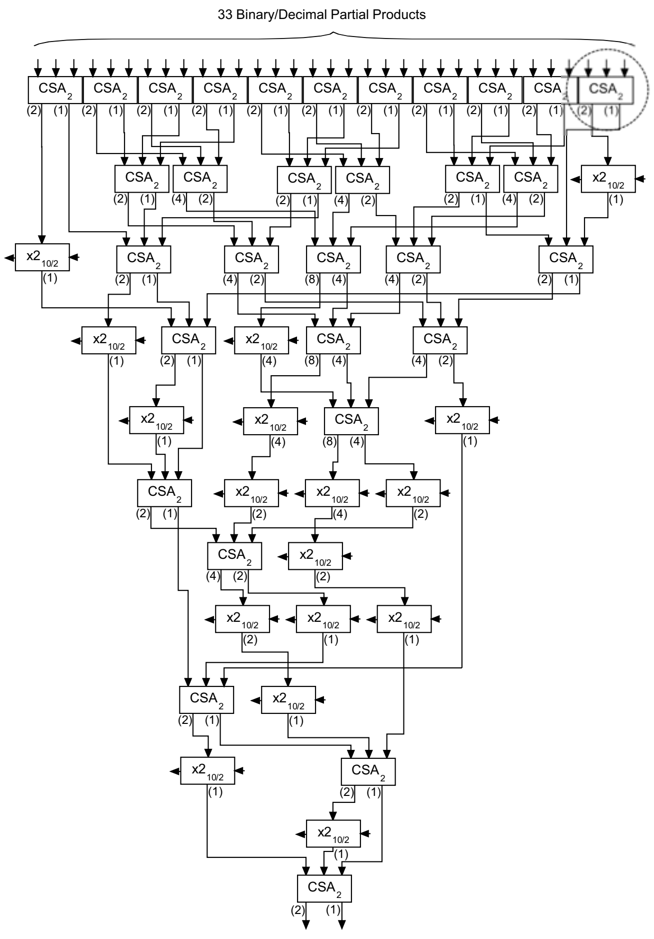


Fig. 5. Improved Partial Product Reduction Tree

B. Improved Doubling Unit Design

To reduce the delay of the binary/decimal doubling units found within the reduction tree, we propose a new doubling unit that significantly reduces delay as compared to the original doubling unit presented in [9]. The original design, pictured in Figure 3, has a high delay due to the addition of a multiplexer to select between the binary and decimal result. The critical path involves four levels of logic from the decimal inputs to the output. To reduce this delay, we propose folding the multiplexer into the logic needed to perform the decimal doubling operation. This creates two-level functions that are then optimized by our synthesis tools. Since both the binary and decimal doubling operations involve a wired shift, there is an opportunity to share terms in the output.

To increase the effectiveness of this method, we also take advantage of the redundancy of the BCD-4221 encoding. As illustrated in Table I, there are multiple ways to represent each of the decimal digits. By selecting an efficient set of BCD-4221 encodings, this can reduce the logic for the doubling unit. Based on our work with the various BCD-4221 encodings, we have selected the following equations:

$$\begin{aligned}
C_{out} &= dx_2x_1x_0 + dx_3x_2\overline{x_0} + \overline{d}x_3 + x_3x_0 + x_3x_1 \\
2x_3 &= d\overline{x_3}\overline{x_2}\overline{x_1}\overline{x_0} + dx_3\overline{x_2}\overline{x_1}x_0 + \\
&\quad \overline{x_3}x_2\overline{x_1} + x_2x_1\overline{x_0} + x_3x_2x_0 + \overline{d}x_2 \\
2x_2 &= dx_3\overline{x_2}\overline{x_1}\overline{x_0} + d\overline{x_3}x_1\overline{x_0} + \\
&\quad dx_3x_2\overline{x_0} + \overline{d}x_1 + x_3x_1 \\
2x_1 &= d\overline{x_3}\overline{x_2}x_1x_0 + dx_3\overline{x_2}\overline{x_1}\overline{x_0} + \\
&\quad d\overline{x_3}x_1\overline{x_0} + \overline{x_3}\overline{x_1}x_0 + x_3x_1x_0 + \overline{d}x_0 \\
2x_0 &= C_{in}
\end{aligned}$$

By synthesizing these equations (our synthesis methodology is discussed in detail in Section V-A) against the original design from [9], we obtain a 63% reduction in critical path delay with a corresponding 53% area increase when both designs are optimized for speed. When optimizing for area, the new design has a 2.1% area advantage and a 1.4% delay advantage over the original design. This second comparison is important because during synthesis of the entire multiplier, less critical trees using the new design can still be optimized to offer significant area savings. It is important to note that these equations and results are somewhat dependent on the technology being used, its area and delay characteristics, and the cells available. However, the general insight of combining the multiplexer with the decimal doubling logic and taking advantage of the redundancy of the BCD-4221 encoding can be applied in any technology.

C. Split Reduction Tree

While the combined binary/decimal designs presented in [9] allow for significant hardware sharing between the binary and decimal multiplication, they also significantly increase the latency of the binary multiplication as compared to a standalone binary multiplier. This is primarily due to the fact that several additional multiplexers inside the binary/decimal doubling units must be traversed in the reduction tree in order to allow it to be shared with the decimal multiplication. In the binary case, the logic in the doubling units are pure overhead since only wired shifts are needed if the binary multiplication is performed in a stand-alone unit.

To improve the latency of the binary multiplication, we propose a split reduction tree architecture in which only the upper portion of the reduction tree is shared between the binary and decimal operations. A worst-case column from a 64-bit/16-digit version of this split reduction tree is pictured in Figure 6. The upper shared portion contains no shared doubling units in order to avoid penalizing the binary path. Once it becomes infeasible to avoid using doubling units to continue the reduction, the tree is split into separate binary and decimal portions. Inside these portions, shared doubling units do not need to be used and hence the binary portion uses wired shifts to achieve the doubling operation. The decimal portion can also use simpler decimal doubling units that no longer contain a multiplexer. This split reduction tree design significantly reduces the delay in producing a binary result. While there is an obvious area penalty due to the replication of parts of the reduction tree, this area penalty is reduced considerably by the use of simpler doubling units without multiplexers. To further reduce the area overhead of this design, 4-bit binary 4:2 CSAs are used in the binary

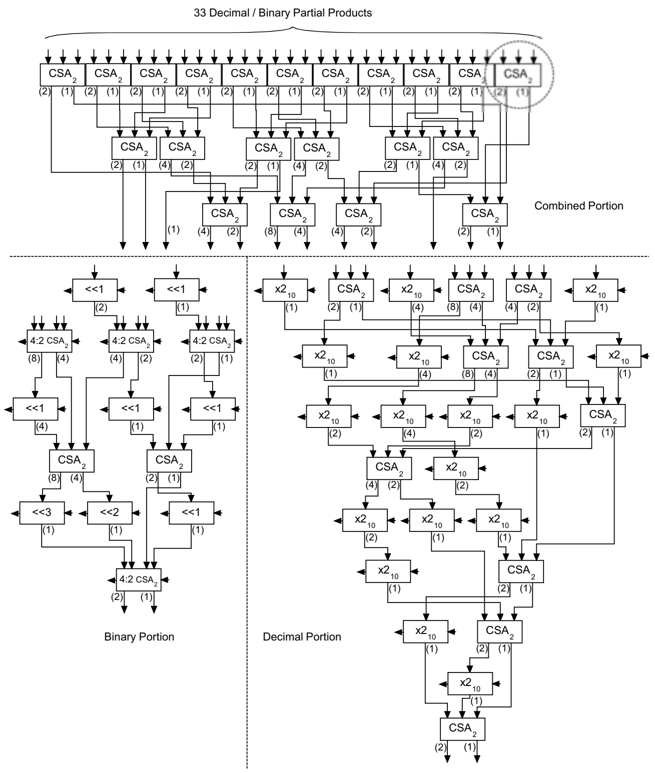


Fig. 6. 64-bit/16-digit Split Partial Product Reduction Tree

portion of the tree based on the design in [18]. While these 4:2 CSAs do not offer any significant delay advantages in our technology, the ability to share logic within the 4:2 CSAs allows for significantly reduced area as compared to using only 3:2 CSAs in the binary portion.

We also investigated applying the split reduction tree multiplier in a 53-bit/16-digit combined binary/decimal multiplier. This design point is significant because 53-bits and 16-digits are the lengths of the significands of double precision binary and decimal floating-point numbers, respectively, from the IEEE 754-2008 standard [3]. In the worst-case we will need to reduce only 32 partial products at this design point because the extra partial product from the unsigned binary multiplication does not need to be handled. We also use the fact that, in the binary case, at most only $\lceil 2 \times p_{bin}/4 \rceil = \lceil 2 \times 53/4 \rceil = 27$ partial products need to be reduced and hence only a portion of the tree needs to be shared between the binary and decimal multiplication. The split reduction tree for this design point is not pictured due to space constraints but is similar in design to the previous split tree picture in Figure 6.

V. Results

A. Verification/Synthesis Methodology

To test and analyze the proposed combined binary/decimal multipliers, register transfer level (RTL) models of the designs were coded in Verilog. An RTL model of the original binary radix-4/decimal radix-5 combined multiplier from [9] was also created. To validate the functionality of the designs, over 5,000,000 random test cases were used, along with hand-picked vectors to test the corner cases. These test cases were applied to the designs using ModelSim 6.3, a HDL simulation and debug environment.

TABLE II

Synthesis Results for Worst-Case Column in Reduction Tree Designs

Tree Design	Bin/Dec Delay		Area	
	FO4	Ratio	μm^2	Ratio
64-bit/16-dig Orig. Tree [9]	34.2	1.00	6,130	1.00
64-bit/16-dig Improved Tree	33.3	0.97	4,375	0.71
64-bit/16-dig Split Tree	19.6/34.5	0.58/1.01	4,951	0.81
53-bit/16-dig Split Tree	17.5/35.3	0.52/1.03	4,354	0.71

TABLE III

Synthesis Results for Various Multiplier Designs

Mult Design	Latency		Area	
	Bin	Dec	μm^2	Ratio
64-bit Bin CSA	5	-	76,973	-
53-bit Bin CSA	5	-	53,724	-
16-dig Radix-5 Dec [9]	-	8	99,911	-
64-bit/16-dig Baseline	5	8	176,884	1.00
53-bit/16-dig Baseline	5	8	153,635	1.00
Orig. 64-bit/16-dig [9]	8	8	135,184	0.76
Improved 64-bit/16-dig	8	8	101,229	0.57
Split 64-bit/16-dig	5	8	112,952	0.64
Split 53-bit/16-dig	5	8	104,789	0.68

To estimate the delay and area of our designs, the Verilog models were synthesized using TSMC's tcbn65gplus 65nm CMOS standard cell library and Synopsys Design Compiler Z-2007.03-SP3. All environmental and process parameters were set to their nominal or typical conditions. We have also included synthesis results for a 64-bit binary multiplier, a 53-bit binary multiplier, and a 16-digit radix-5 decimal multiplier presented in [9]. These multipliers were combined to create two baseline designs using separate multipliers: one using the 64-bit binary multiplier along with the 16-digit decimal multiplier and the other using the 53-bit binary multiplier along with the 16-digit multiplier. In order to provide a fair comparison between the proposed designs, the original design in [9], and our baseline binary and decimal designs, no units from the Synopsys's DesignWare IP library were used; all Verilog models were specified at the structural level. The full multiplier designs were synthesized with an aggressive clock speed goal of 500 *ps*. The fan-out-of-four (FO4) inverter delay, which is the delay of an inverter driving four similar inverters in the cell library, is 31.3 *ps* for this library. This gives a critical path delay of about 16 FO4 for the full multiplier designs. The pipelining for the baseline designs was done by the automatic pipelining feature of Design Compiler. The pipelining for the combined binary/decimal designs was done by hand due to the complexity of the designs. Synthesis results comparing the combinational delay and area of a single worst-case column from the original and proposed reduction trees are given in Table II. The synthesis results for the fully pipelined multipliers are given in Table III. In order to ensure that the synthesis tool optimized the binary paths of the split tree designs despite not having this output on the critical path, we added additional weight to the delay of the binary paths during synthesis.

B. Results Analysis

First, examining the results for the worst-case columns from the original and proposed reduction tree found in Table II, it is easy to see that the proposed reduction trees offer significant area advantages, up to a 29% savings in area. This is primarily due to the reduction in the number of binary/decimal doubling units from the original tree in [9]. Our improved tree from Figure 5 also improves delay by about 2.6%. This minor improvement is most likely due to the fact that the binary 3:2 CSAs set the critical path delay and hence the improved doubling units do not help improve the critical path delay as much. The results for the two split tree designs show a large improvement in the critical path delay of nearly 50% for the binary multiplication over the fully shared design. This comes at only a slight increase in area and decimal critical path delay of this design, making the split tree designs the most attractive reduction trees. The delay penalty on the decimal path is most likely due to the additional fan-out of the shared portion of the tree.

The synthesis results for the fully pipelined multiplier designs are given in Table III. All the designs listed in the table are pipelined for a clock cycle of 500 *ps* or 16 FO4 and hence only latency in clock cycles is reported in the table. From these results, we can draw several conclusions. First, as compared to our baseline separate binary and decimal multiplier designs, the combined binary/decimal designs give significant area savings. The previous design presented in [9] has an area savings of 24% over separate multipliers. The proposed designs offer additional savings of up to 42% over using separate multipliers. In addition, the previous design from [9] has a significant 3 cycle latency penalty for the binary multiplication as compared to a stand-alone binary multiplier. Our split tree designs allow this latency penalty to be eliminated while still offering an area savings of 36%. The split tree designs significantly reduce the area overhead of adding a decimal multiplier without significantly penalizing the latency of the binary multiplication.

VI. Summary

In this paper, several novel combined binary/decimal parallel fixed-point multiplier designs were presented based on a previous parallel fixed-point multiplier. The elements of the previous and improved designs were described, including the generation of multiples, operand recoding, partial product selection, partial product reduction, and the final carry-propagate adder. Novel elements include an improved reduction tree, improved binary/decimal doubling units, and a new split reduction tree. The designs were implemented using Verilog and verified using extensive random vectors. Synthesized standard cell estimates are presented for several design points. These results show that the proposed split tree design significantly reduces the area overhead of decimal multiplication without significantly penalizing the latency of the binary multiplication.

Acknowledgment

This work is sponsored in part by International Business Machines. The authors are grateful to Alvaro Vazquez, Elisardo Antelo, and Paolo Montuschi for their parallel fixed-point multiplier design used as the basis for our designs.

References

[1] L. K. Wang, C. Tsen, M. J. Schulte, and D. Jhalani, "Benchmarks and Performance Analysis for Decimal Floating-Point Applications," in *25th IEEE International Conference on Computer Design*, Oct. 2007, pp. 164–170.

[2] L. Eisen, J. W. Ward III, H.-W. Tast, N. Mäding, J. Leenstra, S. M. Mueller, C. Jacobi, J. Preiss, E. M. Schwarz and S. R. Carlough, "IBM POWER6 Accelerators: VMX and DFU," *IBM Journal of Research and Development*, vol. 51, no. 6, pp. 663–684, November 2007.

[3] "IEEE 754-2008 - IEEE Standard for Floating-Point Arithmetic," The Institute of Electrical and Electronics Engineer, Inc., New York, 2008.

[4] R. H. Larson, "High Speed Multiply Using Four Input Carry Save Adder," *IBM Technical Disclosure Bulletin*, pp. 2053–2054, December 1973.

[5] T. Ohtsuki, Y. Oshima, S. Ishikawa, K. Yabe, and M. Fukuta, "Apparatus for Decimal Multiplication," *U.S. Patent*, June 1987, #4,677,583.

[6] R. L. Hoffman and T. L. Schardt, "Packed Decimal Multiply Algorithm," *IBM Technical Disclosure Bulletin*, vol. 18, no. 5, pp. 1562–1563, October 1975.

[7] M. A. Erle and M. J. Schulte, "Decimal Multiplication Via Carry-Save Addition," in *International Conference on Application-Specific Systems, Architectures, and Processors*, June 2003, pp. 348–358.

[8] R. D. Kenney, M. J. Schulte, and M. A. Erle, "A High-Frequency Decimal Multiplier," in *Proceedings of the IEEE International Conference on Computer Design*, October 2004, pp. 26–29.

[9] A. Vazquez, E. Antelo, and P. Montuschi, "A New Family of High–Performance Parallel Decimal Multipliers," in *18th IEEE Symposium on Computer Arithmetic*, June 2007, pp. 195–204.

[10] T. Lang and A. Nannarelli, "A Radix-10 Combinational Multiplier," in *Proc. of 40th Asilomar Conference on Signals, Systems, and Computers*, Oct 2006, pp. 313–317. [Online]. Available: http://www2.imm.dtu.dk/pubdb/p.php?5010

[11] A. Vazquez and E. Antelo, "Conditional Speculative Decimal Addition," in *7th Conference on Real Numbers and Computers*, July 2006, pp. 47–57.

[12] M. Erle, E. Schwarz, and M. Schulte, "Decimal Multiplication with Efficient Partial Product Generation," in *ARITH '05: Proceedings of the 17th IEEE Symposium on Computer Arithmetic*. Washington, DC, USA: IEEE Computer Society, 2005, pp. 21–28.

[13] R. K. Richards, *Arithmetic Operations in Digital Computers*. New Jersey: D. Van Nostrand Company, Inc., 1955.

[14] M. S. Schmookler and A. W. Weinberger, "High Speed Decimal Addition," *IEEE Transaction on Computers*, vol. C, no. 20, pp. 862–867, August 1971.

[15] S. Vassiliadis, E. Schwarz, and B. Sung, "Hard-Wired Multipliers with Encoded Partial Products," *Computers, IEEE Transactions on*, vol. 40, no. 11, pp. 1181–1197, Nov 1991.

[16] S. Matthew, M. Anders, R. Krishnamurthy, and S. Borkar, "A 4-GHz 130-nm Address Generation Unit with 32-bit Sparse-Tree Adder Core," *Solid-State Circuits, IEEE Journal of*, vol. 38, no. 5, pp. 689–695, May 2003.

[17] I. Koren, *Computer Arithmetic Algorithms*. New Jersey: Prentice-Hall, Inc., 1993.

[18] R. Hussin, A. Y. M. Shakaff, N. Idris, Z. Sauli, R. C. Ismail, and A. Kamarudin, "Redesign the 4:2 Compressor for Partial Product Reduction," in *ACST'07: Proceedings of the 3rd IASTED Conference on Advances in Computer Science and Technology*. Anaheim, CA, USA: ACTA Press, 2007, pp. 54–57.