Kodiboyina Chanti* et al.
(IJITR) INTERNATIONAL JOURNAL OF INNOVATIVE TECHNOLOGY AND RESEARCH
Volume No.5, Issue No.3, April – May 2017, 6028-6035.

Implementation of Power and Delay Variant of a Radix-10 Combinational Multiplier Using Mixed Binary and BCD Code

KODIBOYINA CHANTI
M.Tech VLSI
Avanthi Institute of Engg & Technology
Makavarapalem, Narsipatnam, VSP

PENTA ASHOK
Assistant Professor
Avanthi Institute of Engg & Technology
Makavarapalem, Narsipatnam, VSP

Abstract: **The decimal multiplication is one of the most important decimal arithmetic operations which have a growing demand in the area of commercial, financial, and scientific computing. It has been revived in recent years due to the large amount of data in commercial applications. In this paper, we propose a parallel decimal multiplication algorithm with three components, which are a partial product generation, a partial product reduction, and a final digit-set conversion. First, a redundant number system is applied to recode not only the multiplier, but also multiples of the multiplicand in signed-digit (SD) numbers.**

Furthermore, we present a multi operand SD addition algorithm to reduce the partial product array.We consider the problem of multi operand parallel decimal addition with an approach that uses binary arithmetic, suggested by the adoption of binary-coded decimal (BCD) numbers. This involves corrections in order to obtain the BCD result or a binary-to-decimal (BD) conversion. The BD conversion moreover allows an easy alignment of the sums of adjacent columns. We treat the design of BCD digit adders using fast carry-free adders and the conversion problem through a known parallel scheme using elementary conversion cells. Spread sheets have been developed for adding several BCD digits and for simulating the BD conversion as a design tool. In this project Xilinx-ISE tool is used for simulation, logical verification, and further synthesizing.

LANGUAGE USED: VERILOG.

SOFTWARE USED: XILINX ISE 12.3i.

I. INTRODUCTION

Hardware implementations of decimal arithmetic units have recently gained importance because they provide higher accuracy in financial applications. In this work, we present a combinational decimal multiplier which can be pipelined to reach the desired throughput. The multiply unit is organized as follows: the multiplier is recoded; the partial products are kept in a redundant format; the partial product are accumulated by a tree of redundant adders and the final product is obtained by converting the carry-save tree's outputs into binary-coded decimal (BCD) format. With respect to previous implementations of radix-10 multipliers such as the ones in our design is different in the following aspects:

1) The multiplier is combinational (parallel) and not sequential;
2) We recode only the multiplier while in both operands are recoded in -5 to +5;
3) In the partial product generation only multiples of 5 and2 are required;
4) The accumulation of partial products is done in a tree of radix-10 carry-save adders and counters while in the sequential unit of signed-digit adders are used.

We present the standard cells implementation of the multiplier and compare its latency with those of the schemes presented in and Moreover, we compare the delay and the area of the decimal combinational multiplier with those obtained by the implementation of a binary (radix-4) double precision multiplier.

II. REDUNDANT BCD REPRESENTATIONS

The proposed decimal multiplier uses internally a redundant BCD arithmetic to speed up and simplify the implementation. This arithmetic deals with radix-10 ten's complement integers of the form: The value of Z_i depends on the decimal representation parameterized by (l; m; e). We use a 4-bit encoding to represent digits Z_i. This allows us to manage decimal operands in different representations with the same arithmetic, such as on the other hand, the binary value of the 4-bit vector representation of Z_i is given by them

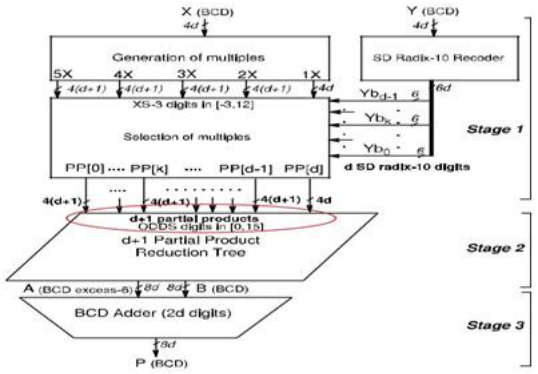


Fig: Combinational SD radix-10 architecture.

In contrast to our previous SD radix-10 implementations, 3X is obtained in a reduced constant time delay (3 XOR-gate delays) by using the XS-3 representation. Moreover, a negative multiple is generated from the correspondent positive one by a bitwise XOR operation. Consequently, the latency is reduced and the hardware implementation is simplified. The scheme proposed in also produces 3X in constant time but using redundant signed-digit BCD arithmetic. Decimal partial product reduction. In this stage, the array of dþ1 ODDS partial products is reduced to two 2d-digit words (A, B). Our proposal relies on a binary carry-save adder tree to perform carry-free additions of the decimal partial products. The array of dþ1 ODDS partial products can be viewed as adjacent digit columns of height h d þ 1. Since ODDS digits are encoded in binary, therules for binary arithmetic apply within the digit bounds, and only carries generated between radix-10 digits (4-bit columns) contribute to the decimal correction of the binary sum. That is, if a carry out is produced as a result of a 4-bit (modulo 16) binary addition, the binary sum must beincremented by 6 at the appropriate position to obtain the correct decimal sum (modulo 10 addition).Two previous designs implement tree structures for the addition of ODDS operands. In the non speculative BCD adder, a combinational logic block is used to determine the sum correction after all the operands have been added in a binary CSA tree, with the maximum number of inputs limited to 19 BCD operands. By contrast, in our method the sum correction is evaluated concurrently with the binary carry-save additions using columns of binary counters. Basically we count the number of carries per decimal column and then a multiplication by 6 is performed (a correction by 6 for each carry-out from each column). The result is added as a correction term to the output of the binary carry-save reduction tree. This improves significantly the latency of the partial product reduction tree. Moreover, the proposed architecture accepts an arbitrary number of ODDS or BCD operand inputs. Some of PPR tree structures presented in (the area-improved PPR tree) also exploit a similar idea, but rely on a custom designed ODDS adder to perform some of the stage reductions.

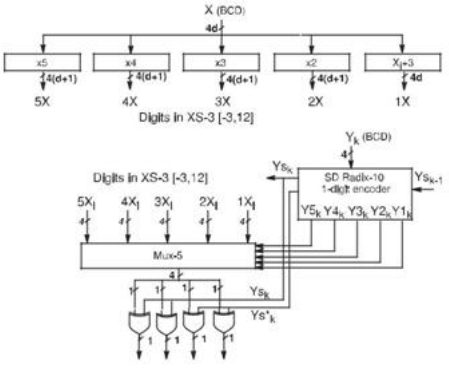
Our proposal aims to provide an optimal reuse of any binary CSA tree for multi operand decimal addition, as it was done in for the 4221 and 5211 decimal coding Conversion to (non-redundant) BCD. We consider the use of a BCD carry-propagate adder to perform the final conversion to a non-redundant BCD product P ¼ Aþ B. The proposed architecture is a 2d-digit hybridparallel prefix/carry-select adder, the BCD Quaternary Tree adder (see Section 6). The sum of input digits A_i, B_i at each position i has to be in the range ½0; 18& so that at most one decimal carry is propagated to the next position i þ 1 . Furthermore, to generate the correct decimalcarry, the BCD addition algorithm implemented requires A_i þ $B_i$to be obtained in excess-6. Several choices arepossible. We opt for representing operand A in BCD excess-6 (A_i2 ½0;9&, ½A_i& ¼$A_i$þe, e¼6), and B coded in BCD (B_i2 ½0;9&, e¼0).

III. DECIMAL PARTIAL PRODUCT GENERATION

The partial product generation stage comprises the recoding of the multiplier to a SD radix-10 representation, the calculation of the multiplicand multiples in XS-3 code and the generation of the ODDS partial products.The SD radix-10 encoding produces d SD radix-10 digits Yb_k, being the most significant digit (MSD) of the multiplier [29]. Each digit Yb_k is represented with a 5-bit hot-one code ($Y1_k$; $Y2_k$; $Y3_k$; $zY4_k$;$Y\ 5_k$) to select the appropriate multiple f1X; . . . ; 5Xg with a5:1 mux and a sign bit Ys_k that controls the negation of the selected multiple.The negative multiples are obtained by ten's complementing the positive ones. This is equivalent to taking the nine's complement of the positive multiple and then adding

1. As we have shown in Section 2, the nine's complement can be obtained simply by bit inversion.

2. Providing support for 20 or more input BCD operands would require a significant modification of the original non speculative addition algorithm.



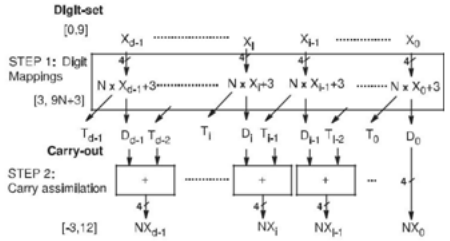


Fig: SD radix-10 generation of a partial product digit. Generation of decimal multiples NX.

The XOR gates at the output of the mux invert the multiplicand multiple, to obtain its 9's complement, if the SD radix-10 digit is negative (Ys_k¼1). On the other hand, if the signals ($Y1_k$; $Y2_k$; $Y3_k$; $Y4_k$; $Y5_k$) are all zero then PP½k& ¼0, but it has to be coded in XS-3 (bit encoding 0011). Then, to set the two least significant bits to 1, the input to the XOR gate is Ys_k¼Ys_k_Yb_kiszero (_ denotes the Boolean OR operator), where Yb_k is zero equals 1 if all the signals ($Y1_k$; $Y2_k$; $Y3_k$; $Y4_k$; $Y5_k$) are zero.In addition, the partial product signs are encoded into their MSDs (see Section 4.2). The generation of the most significant partial product PP½d& is described in Section 4.4, and only depends on Ys_{d1}, the sign of the most significant SD radix-10 digit.

Generation of the Multiplicand Multiples:

We denote by NX2 f1X;2X;3X;4X;5Xg, the set of multiplicand multiples coded in the XS-3 representation, withdigitsNX_i2 ½3;12&, being ½NX_i& ¼NX_iþ32 ½0;15& the corresponding value of the 4-bit binary encoding of NX_i given by Equation (2). the high-level block diagram of the multiples generation with just one carry propagation. This is performed in two steps:

Most-Significant Digit Encoding

The MSD of each PP½k&, PP_d½k&, is directly obtained in the ODDS representation. Note that these digits store the carries generated in the computation of the multiplicand multiples and the sign bit of the partial product. For positive partial products we haveNote that the term ½PP_d½k&& is always positive. Specifically, for positive partial products (Ys_k¼0), this term results in

	1X			2X			3X			4X			5X		
X_i	X_i+3	T_i	D_i	$(X_i\times2)+3$	T_i	D_i	$(X_i\times3)+3$	T_i	D_i	$(X_i\times4)+3$	T_i	D_i	$(X_i\times5)+3$	T_i	D_i
0	3	0	3	3	0	3	3	0	3	3	0	3	3	0	3
1	4	0	4	5	0	5	6	0	6	7	0	7	8	0	8
2	5	0	5	7	0	7	9	0	9	11	1	1	13	1	3
3	6	0	6	9	0	9	12	0	12	15	1	5	18	1	8
4	7	0	7	11	1	1	15	1	5	19	1	9	23	2	3
5	8	0	8	13	1	3	18	1	8	23	2	3	28	2	8
6	9	0	9	15	1	5	21	2	1	27	2	7	33	3	3
7	10	0	10	17	1	7	24	2	4	31	2	11	38	3	8
8	11	0	11	19	1	9	27	2	7	35	3	5	43	4	3
9	12	0	12	21	1	11	30	2	10	39	3	9	48	4	8

TABLE Preferred Digit Recoding Mappings for NX Multiples

Correction Term

The resultant partial product sum has to be corrected off-the-critical-path by adding a precomputed term, which only depends on the format precision d. This term has to gather: (a) the 8 constants that have not been included in the MSD encoding and (b) a Particularizing for d¼16 and d¼34 digit operands, the following expressions for the correction term in 10's complement are obtained. The correction term is allocated into the array of dþ1 partial products coded in ODDS (digits in ½0;15&), as we show in the next section.

Partial Product Array

As a conclusion of the considerations in the previous sections, Fig. illustrates the shape of the partial product array, particularizing for d¼16. Note that the maximum digit column height is dþ1.In each column several components can be observed. Digits labeled with O represent the redundant excess-3 BCD digits in the set ½0;15&. Digits labeled with S_k represent the MSD of each partial product, PP_d½k& (see Section 4.2). The 16 least significant digits of the correction term f_cð16Þ are placed at the least significant position of each row after being added to Ys_k, to complete the 10's complement in case of a negative partial product; thus H_k¼Ys_kþ f0;3;7g (digitwise addition, out of the critical path), so that H_k2 ½0;8&. Note that the negative bit 10^{32} is canceled with the carry-out of the partial product sum in excess. The 16 leading digits of the correction term, ½f_cð16Þ&$_d$, are added to the most significant partial product PP½d&. Thus, in parallel with the multiplicand his computation does not involve carry propagation and it is out of the critical path). Digits labeled as F in Fig., represent the most-significant partial product, PP½d&, where

IV. DECIMAL PARTIAL PRODUCT REDUCTION

The PPR tree consists of three parts:

(1) A regular binary CSA tree to compute an estimation of the decimal partial product sum in a binary carry-save form (S, C),

(2) A sum correction block to count the carries generated between the digit columns,

(3) A decimal digit 3:2 compressor which increments the carry-save sum according to the carries count to obtain the final double-word product (A; B), A being represented with excess-6 BCD digits and B being represented with BCD digits. The PPR tree can be viewed as adjacent columns of h ODDS digits each, h being the column height (see Fig.), and hdþ1. the high-level architecture of a column of the PPR tree (the ith column) with h ODDS digits in [0, 15] (4 bits per digit). Each digit column of the binary CSA tree

(the gray colored box in Fig.) reduces the h input digits and n_{cin} input carry bits, transferred from the previouscorresponding dot-diagram representation for h¼17 (m¼414) in Fig. An efficient implementation is obtained by representing the digit of W_i6 with l ODDS digits, Wt_i½0&; . . . ; Wt_i½l 1&), being l ¼ 1 for Decimal64, and l ¼ 2 for Decimal128.

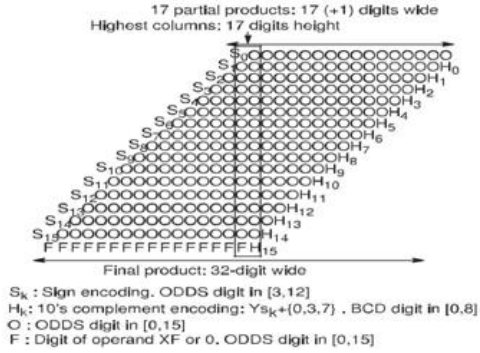


Fig. Decimal partial product array generated for d ¼ 16 (16 16-digit multiplier).

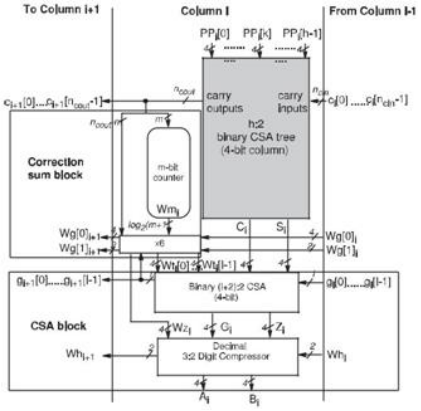


Fig. High-level architecture of the proposed decimal PPR tree (h inputs, 1-digit column).

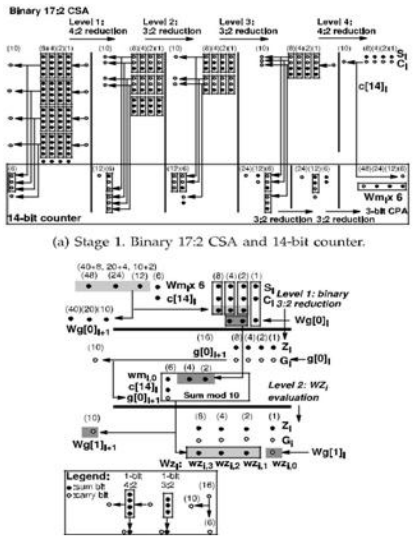


Fig. Dot-diagrams for the proposed decimal PPR (h ¼ 17 inputs, 1-digit column).

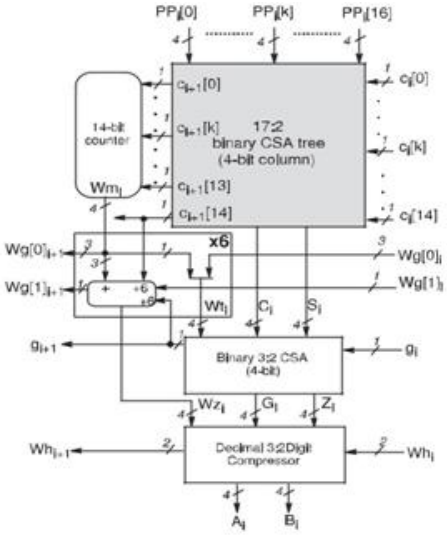


Fig Implementation of the PPR Tree Highest Column (h ¼ 17) for a 16 16-digit multiplication.

This CSA generates l carry outs $g_{iþ1}$½0&;. . .; $g_{iþ1}$½l1& with weight 1610^i, which are transferred to the next column, and introduced into the 6 block to produce another ODDS digit, Wz_i 2 ½0; 15.The last step is the addition of digits G_i; Z_i; Wz_i of the column, $G_iþZ_iþWz_i$2 ½0;45&. We have designed a decimal 3:2 digit compressor that reduces digits Wz_i, G_i and Z_i to two digits A_i, B_i. The dot-diagram of the decimal 3:2 digit compressor is shown in Fig. To obtain the final BCD product by using a single BCD carry propagate addition, that is, P¼AþB, which is the last step in the multiplication (see Fig. and Section 3), it is required that A_i þ B_i 2 ½0; 18&. Moreover, to reduce the delay of the finalBCD carry-propagate adder (see Section 6) operand A is obtained in excess-6, so that we compute ½A_i& ¼$A_i$þe in excess e¼6 as defined by Equation (2), being the output digits sum ½A_i& þB_i2 ½6;24&.The evaluation is split in two parts:

Block A computes the sum of the two MSBS of the input digits (the bits with weights 8 and 4), and a two-bit carry input Wh_i2 f0;1;2;3g. This sum is in ½0; 39 &. The outputs of this block are a BCD digit A_i in excess-6 ½A_i& 2 ½6;15& and a two-bit decimal carry output $Wh_{iþ1}$2 f0;1;2;3g which is transferred to the next column (the iþ1th column). Note that the LSB of the carry output $Wh_{iþ1}$ depends on the MSB of the input carry Wh_i. However, there is no further carry propagation since the LSB of $Wh_{iþ1}$ is just the LSB of ½$A_{iþ1}$&, that is, ½$A_{iþ1;0}$&.

On the other hand, Block B implements the sum of the two LSB bits of the input digits (the bits with weights 2 and 1). This sum is in ½0;9&, so that B_i is evaluated as a regular binary addition.

After that, the sum correction digits (Wt_i½0&;. . .; Wt_i½l1&) and the output digits of the binary CSA tree (S_i, C_i)

V. FINAL CONVERSION TO BCD

The selected architecture is a 2d-digit hybrid parallel prefix/ carry-select adder, the BCD Quaternary Tree adder. The delay of this adder is slightly higher to the delay of a binary adder of 8d bits with a similar topology.

The decimal carries are computed using a carry prefix tree, while two conditional BCD digit sums are computed out of the critical path using 4-bit digit adders which implements ½A_i & þB_iþ0 and ½A_i& þB_iþ1. These conditional sums correspond to each one of the carry input values. If the conditional carry out from a digit is one, the digit adder performs a 6 subtraction. The selection of the appropriate conditional BCD digit sums is implemented with a final level of 2:1 multiplexers.

To design the carry prefix tree we analyzed the signal arrival profile from the PPRT tree, and considered the use of different prefix tree topologies to optimize the area for the minimum delay adder.

VI. EVALUATION AND COMPARISON

The proposed combinational architectures for BCD 1616-digit and 34 34-digit multipliers are evaluated andcompared with other representative BCD multipliers. The area and delay figures of our architectures were obtained from an area-delay evaluation model for static CMOS gates, and validated with the synthesis ofverified RTL models coded in VHDL. This evaluation is detailed in Section 6.1.

	16 × 16-digit mult.		34 × 34-digit mult.	
Component	Delay #FO4	Area #NAND2	Delay #FO4	Area #NAND2
SD rec.+buff	7.0*	500	7.5*	1000
Multiples+buff.	5.9	1500	6.3	3100
Selection	3.2*	12900	3.2*	57500
PPG stage	10.2	14900	10.7	61600
PPR tree	25.5	11700	32.3	50000
Conversion	4.2	1600	4.2	3400
Decimal adder	11.5	2400	13.6	4600
Total stage	51.4	30600	60.8	119600

*These terms contribute to the critical path delay.

TABLE Area and Delay (LE-Based Model) for the Proposed Mults

Finally, the most representative sequential and parallel decimal multipliers have been compared with our architecture. The results of the comparison are summarized in Section 6.2

Evaluation

As stated above, the evaluation has been performed in two steps. First, a technological independent evaluation using a model for static CMOS circuits based on Logical Effort (LE) has been carried out, and then the results obtained with this model have been validated with the synthesis and functional verification of the RTL model.

Technological Independent Evaluation

Our technological independent evaluation model allows us to obtain a rough estimation of the area and delay figures for the architecture being evaluated. It takes into account the different input and output gate loads, but neither interconnections nor gate sizing optimizations are modeled. The delay is given in FO4 units, that is, the delay of an 1 inverter with a fanout of four inverters. The hardware complexity is given as the number of equivalent minimum size NAND2 gates. We do not expect this rough model to give absolute area-delay figures, due to the high wiring complexity of parallel multipliers. However, based on our experience this model is good enough for making design decisions at gate level and it provides reasonable accuracy of area and delay ratios to compare different designs.

Table shows the delay, input capacitance (L_{in}) and area of the main building blocks used in the BCD multipliers. The input capacitance is normalized to the input capacitance of the 1 inverter. The L_{out} parameter represents the normalized output load connected to the gate. The XOR2 gate is implemented with CMOS transmission gates.

To evaluate our architectures, gates with the drive strength of the minimum sized (1) inverter have been assumed, and buffers have been inserted to deal with high loads. The critical path delay in every stage of the multiplier has been estimated as the sum of the delays of the gates on this critical path. The area and delay figures obtained for the 16 16-digit and 34 34-digit architectures are shown inTable 4.

Synthesis Results

The designs have been synthesized using Synopsys Design Compiler B-2012.09-SP3 and a 90 nm CMOS standard cell library [11]. The FO4 delay for this library is 49 ps undertypical conditions (1 V, 25 C). The area-delay curves of Fig. have been obtained with the constraint C_{out}¼C_{in}¼4C_{inv}, where$C_{inv}$is the input capacitance of a 1inverterof the library.

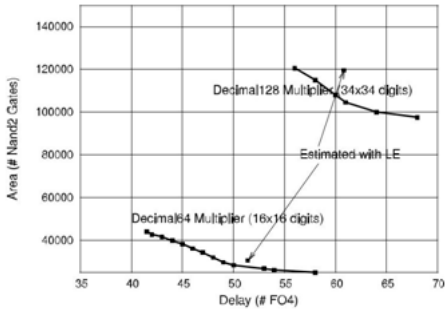


Fig. Area-delay space obtained from synthesis.

We also include in Fig. the area-delay points estimated from the LE-based model evaluation. We

have kept the hierarchy of the design in the synthesis process as described in Sections 3 to 6 (no top level architecture optimization options). Nevertheless, some specific structures have been optimized internally to reduce the overall delay.

To ensure the correctness of the designs we have simulated the RTL models of the 1616-digit and 3434-digit multipliers using the Synopsys VCS-MX tool and a comprehensive set of random test vectors.

Comparison

Table shows the area and delay estimations obtained from synthesis for some representative BCD sequential and combinational multipliers. As far as we know, the most representative high-performance BCD multipliers are 1616-digit combinational and sequential implementations. The area and delay figures shown in Table correspond to the minimum delay point of each implementation, and were obtained from the synthesis results provided in the respective reference, except for the two multipliers of reference, which correspond to an estimation obtained by their authors using a LE-based model. The comparison ratios are given with respect to the area and delay figures of a 53-bit binary Booth radix-4 multiplier extracted from.

Architecture	Delay (# FO4)	# Cycles	Latency (# FO4)	Ratio	Throughput Mult./Cycle	Area (NAND2)	Ratio
Combinational multipliers:							
Binary 53 × 53-bit:							
Booth radix-4	31.1	1	31.1	1.00	1	34000	1.00
BCD 16 × 16-digit:							
Ref. [19]	58.9	1	58.9	1.90	1	68000	2.00
Ref. [7]	55.8	1	55.8	1.80	1	68000	2.00
Ref. [16]	53.5	1	53.5	1.70	1	79600	2.35
Delay-Improved [12]	47	1	47	1.50	1	48600	1.45
Area-Improved [12]	51	1	51	1.65	1	39100	1.15
SD Radix-10 [30]	48	1	48	1.55	1	44500	1.30
SD Radix-5 [30]	46.5	1	46.5	1.50	1	49700	1.45
Ref. [14]	43.1	1	43.1	1.40	1	49900	1.45
Proposed	41.5	1	41.5	1.35	1	44200	1.30
BCD 34 × 34-digit:							
Proposed	56	1	56	1.80	1	120600	3.55
Sequential multipliers (BCD 16 × 16-digit):							
Ref. [10]	16	20	320	10.20	1/17	16000	0.50
Ref. [9]	14.7	20	294	9.45	1/17	18550	0.55
Ref. [17]	12.7	24	305	9.80	1/17	31500	0.90

TABLE Synthesis Results for Fixed-Point Multipliers.

The PPG of multipliers is based on a SD radix-5 scheme that generates 32 BCD partial products for a 16-digit multiplier. Though it only requires simple constant time delay BCD multiplicand multiples, the 9's complement operation for obtaining the negative multiples is more complex than a simple bit inversion. The partial product reduction implemented in is a BCD carry-save adder tree build of BCD digit adders. On the other hand, the BCD partial products are reduced in by using counters that compute the binary sum of each column of digits sum, and subsequent binary to decimal conversions.

The BCD multiplier pre-computes all the positive decimal multiplicand multiples f0X;. . .;9Xg. The delay of PPG is reduced by representing the complex operands (3X;6X;7X;8X;9X) as the sum of two simpler multiples. The number of partial products generated is therefore equivalent to that of the SD radix-5 scheme. The PPR tree is implemented with BCD digit adders as in. This has the disadvantage of a large area compared to the other BCD multipliers analyzed.

The two 1616-digit BCD multipliers of implement an easy-multiple PPG (only precomputes f2X;4X;5Xg) that produces 32 BCD partial products. The intermediate decimal partial product sums are computed in overloaded BCD to speed up the PPR evaluation. The delay-improved design uses a tree structure built of five levels overloaded BCD digit adders, while the area-improved design replaces two levels of these custom designed adders by three levels of 4 : 2 compressors and a binary counter. This reduces the area consumption but at the cost of introducing a significant latency penalty.

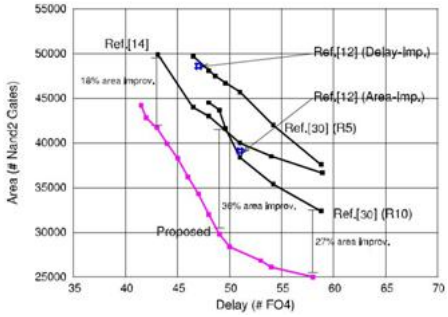


Fig. Area-delay space for the fastest 1616-digit mults.

The BCD multipliers in use either the SD radix-5 PPG scheme or a SD radix-10 PPG scheme. The last one has the advantage that practically it halves the number of partial products generated by the former (17 against 32 for 1616-digit multiplications). However, it has the disadvantage of a significant latency overhead due to the generation of the complex multiple 3X. The latency and area of priorart multipliers are improved by representing the partial products in (4221) or (5211) decimal codes, which allow them to implement PPR using a very regular and compact tree of 4-bit binary carry-save adders (built of 3 : 2 or 4 : 2 compressors) and decimal digit doublers.

The most recent implementation is presented in. It also uses a SD radix-10 PPG scheme to reduce the number of partial products generated to 17, and subsequently, the area of the PPR tree. To avoid the latency overhead of the 3 multiple generation, the partial products are coded in a redundant SD representation.Sequential1616-digit (Decimal64) BCD multipliers are about two times smaller than equivalent parallel implementations, but have higher latency and reduced throughput (one mult issued every 17 cycles). For example, the proposed multiplier is about seven times faster than the best

sequential implementation proposed in, but requires 2.5 times more area.

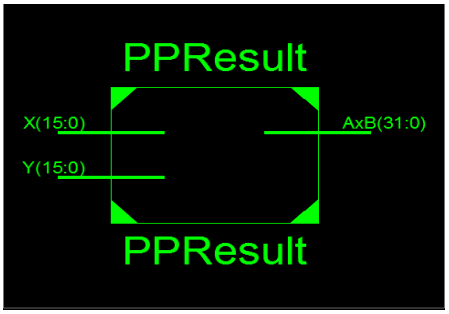
To compare the high hardware cost of a combinational Decimal128 implementation, we also include in Table 5 the area and delay figures obtained for our 3434-digit BCD multiplier. Due to the tight area and power consumption constraints of current DFUs , a sequential architecture seems a more realistic solution than a fully pipelined implementation for a commercial Decimal128 multiplier.

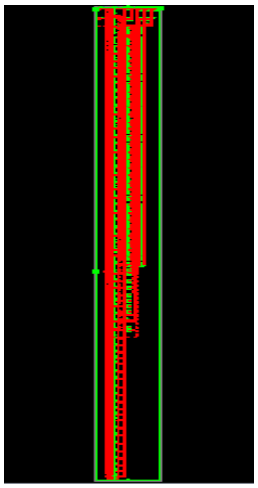
Finally, we present a more detailed comparison of the fastest BCD 1616-digit combinational multipliers (SD Radix-5 and SD Radix-10, and the proposed one) in terms of latency and area. The corresponding area-delay synthesis values are shown in Fig.

We have directly introduced in the figure the area-delay curves of referenced multipliers and as provided by their authors, since all of them were synthesized using 90 nm CMOS standard cell libraries and similar conditions. The area-delay points for the two multipliers of reference correspond to an estimation obtained by their authors using a LE-based model. From the area-delay space represented in Fig., we observe that our proposed decimal multiplier has an area improvement roughly in the range 20-35 percent depending on the target delay. On the other hand, for the minimum delay point (44FO4), the proposed multiplier is still competitive with the fastest design shown in .More recently, the authors of reference have presented in a comparison study between their delay-improved multiplier and the multiplier of reference based on synthesis results using a TSMC 130 nm standard CMOS process under typical conditions (1.2 V, 25 C). They show that for the minimum delay point of each one of the two area-delay curves obtained, the delay-improved multiplier is 20 percent faster and has 10 percent less area compared to the design of. Therefore, according to the curve corresponding to the design presented in should be to the left of the area-delay points corresponding to the delay-improved design presented in.

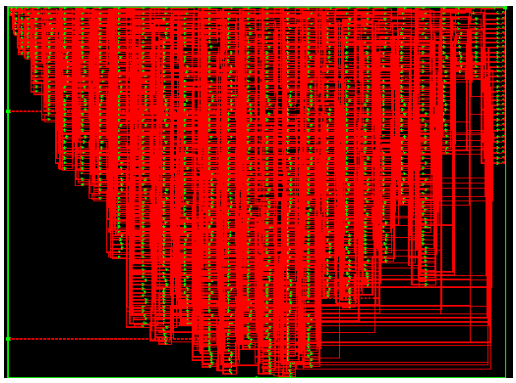
VII. RESULTS

RTL SCHEMATIC:





TECHNOLOGY SCHEMATICS:



WAVEFORM:

VIII. CONCLUSION

In this paper we have presented the algorithm and architecture of a new BCD parallel multiplier. The improvements of the proposed architecture rely on the use of certain redundant BCD codes, the XS-3 and ODDS representations. Partial products can be generated very fast in the XS-3 representation using the SD radix-10 PPG scheme: positive multiplicand multiples (0X, 1X, 2X, 3X, 4X, 5X) are precomputed in a carry-free way, while negative multiples are obtained by bit inversion of the positive ones. On the other hand, recoding of XS-3 partial products to the ODDS representation is straightforward. The ODDS representation uses the redundant digit set [0, 15] and a 4-bit binary

encoding (BCD encoding), which allows the use of a binary carry-save adder tree to perform partial product reduction in a very efficient way. We have presented architectures for IEEE-754 formats, Decimal64 (16 precision digits) and Decimal128 (34 precision digits). The area and delay figures estimated from both a theoretical model and synthesis show that our BCD multiplier presents 20-35 percent less area than other designs for a given target delay

IX. REFERENCES

[1] A. Aswal, M. G. Perumal, and G. N. S. Prasanna, "On basic financial decimal operations on binary machines," IEEE Trans. Comput., vol. 61, no. 8, pp. 1084–1096, Aug. 2012.

[2] M. F. Cowlishaw, E. M. Schwarz, R. M. Smith, and C. F. Webb, "A decimal floating-point specification," in Proc. 15th IEEE Symp. Comput. Arithmetic, Jun. 2001, pp. 147–154.

[3] M. F. Cowlishaw, "Decimal floating-point: Algorism for computers," in Proc. 16th IEEE Symp. Comput. Arithmetic, Jul. 2003, pp. 104–111.

[4] . Carlough and E. Schwarz, "Power6 decimal divide," in Proc. 18th IEEE Symp. Appl.-Specific Syst., Arch., Process., Jul. 2007, pp. 128–133.

[5] S. Carlough, S. Mueller, A. Collura, and M. Kroener, "The IBM zEnterprise-196 decimal floating point accelerator," in Proc. 20th IEEE Symp. Comput. Arithmetic, Jul. 2011, pp. 139–146.

[6] L. Dadda, "Multioperand parallel decimal adder: A mixed binary and BCD approach," IEEE Trans. Comput., vol. 56, no. 10, pp. 1320–1328, Oct. 2007.

[7] L. Dadda and A. Nannarelli, "A variant of a Radix-10 combinational multiplier," in Proc. IEEE Int. Symp. Circuits Syst., May 2008, pp. 3370–3373.

[8] L. Eisen, J. W. Ward, H.W. Tast, N. Mading, J. Leenstra, S. M. Mueller, C. Jacobi, J. Preiss, E. M. Schwarz, and S. R. Carlough, "IBM POWER6 accelerators: VMX and DFU," IBM J. Res. Dev., vol. 51, no. 6, pp. 663–684, Nov. 2007.

[9] M. A. Erle and M. J. Schulte, "Decimal multiplication via carry-save addition," in Proc. IEEE Int. Conf Appl.Specific Syst., Arch., Process., Jun. 2003, pp. 348–358.

