ISSN No: 2348-4845
International Journal & Magazine of Engineering, Technology, Management and Research
A Peer Reviewed Open Access International Journal

A Novel High Radix Booth Multiplication Algorithm for High Speed Arithmetic Logics

Kanakam Srikanth
M.Tech (VLSI-SD),
Dept ECE,
TKREC, Meerpet, Hyderabad.

N. Rajkumar
Assistant Professor,
Dept ECE,
TKREC, Meerpet, Hyderabad.

Dr.P.Ram Mohan Rao
FIE,CE(I),MISTE,MISH,MISCEE,MASCE(I),MISNT,
Principal,
TKREC, Meerpet, Hyderabad.

ABSTRACT:

We present the algorithm and architecture of a BCD parallel multiplier that exploits some properties of two different redundant BCD codes to speedup its computation: the redundant BCD excess-3 code (XS-3), and the overloaded BCD representation (ODDS). In addition, new techniques are developed to reduce significantly the latency and area of previous representative high performance implementations.

Partial products are generated in parallel using a signed-digit radix-10 recoding of the BCD multiplier with the digit set [-5, 5], and a set of positive multiplicand multiples (0X, 1X, 2X, 3X, 4X, 5X) coded in XS-3. This encoding has several advantages. First, it is a self-complementing code, so that a negative multiplicand multiple can be obtained by just inverting the bits of the corresponding positive one.

Also, the available redundancy allows a fast and simple generation of multiplicand multiples in a carryfree way. Finally, the partial products can be recoded to the ODDS representation by just adding a constant factor into the partial product reduction treeSince the ODDS uses a similar 4-bit binary encoding as non-redundant BCD, conventional binary VLSI circuit techniques, such as binary carry-save adders and compressor trees, can be adapted efficiently to perform decimal operations.

To show the advantages of our architecture, we have synthesized a RTL model for 16 16-digit and 34 34-digit multiplications and performed a comparative survey of the previous most representative designs. We show that the proposed decimal multiplier has an area improvement roughly in the range 20-35 percent for similar target delays with respect to the fastest implementation.

1.INTRODUCTION:

The need for computing directly decimal numbers (avoiding the decimal-to-binary conversion of input data and the reverse conversion for output data, both necessary with purely binary processors) has been stressed by Cowlishaw. The most relevant data of a future IEEE standard for decimal floating point numbers can be found in[2].Lang and Nannarelli proposed a combinational 16×16-digit decimal multiplier in. The unit of is organized as follows: the multiplier is recoded in such a way that only multiples 2 and 5 of the multiplicand are required; 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.

The unit of synthesized in a 90 nm library of standard cells has an operation latency of 2.65 ns and a total area of 300,000µm2.In this work, we will deal with the accumulation of partial products by proposing a different architecture for it.The partial product array of a 16×16-digit multiplier, ob-tained from the partial product generator of the unit of , is shown in Fig. 1.The first row is composed of 17 small discs, each representing a BCD digit. The second row is composed of 17 small circles, each representing a bit. The whole array is composed by 16 pairs of such rows, suitable shifted, as shown in the figure.In the value of the array is computed via a tree of additions of rows. Each addition is performed using a carry-free decimal adder similar to that proposed by Erle and Schulte [4]. The product is the value of such array, where the weights 10c of the digits of each column assume the value 100 for the rightmost column to 1031 for the leftmost column. We compute the value of each decimal column assuming for the binary weights within each column the values 23, 22, 21, 20.

ISSN No: 2348-4845
International Journal & Magazine of Engineering, Technology, Management and Research
A Peer Reviewed Open Access International Journal

In the next sections the basis of the array reduction scheme, implementation details and the results in terms of delay and area are presented. The results show that the variant of the multiplier proposed is about 5% faster and has roughly the same area with re-spect to the scheme.

II. THE CARRY-SAVE ADDITION OF THE COL-UMNS:

The column sums can be computed using carry save ad-ditions. We show them in dot-notation schemes, as in Fig. 2. Such kind of schemes, introduced and extend-ed for obtaining more compact versions, can be easily drawn by hand or, to avoid mistakes, using a program on a spreadsheet. The program for our specific case can be freely downloaded from .

Fig. 1. The partial product array of a 16×16 digits mul-tiplier.

In Fig. 2, each scheme is marked with an integer c, which is the number of digit-bit pairs composing a col-umn of the array. In the example: 1 ≤ c ≤ 16. The first row of each scheme is composed of 3 dots with a num-ber n next to each dot and a fourth dot with a number 2n. The number n represents the

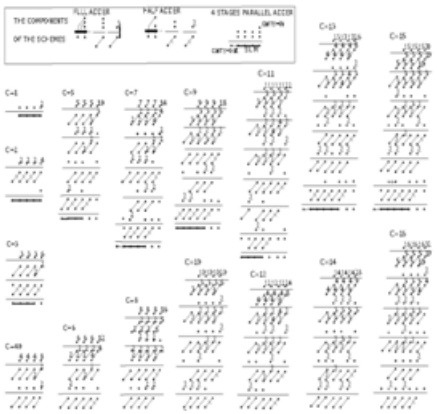


Fig. 2. The binary column adders for a 16×16 digit mul-tiplier.

cardinality of dots having the same weight. A dot with no number implies n = 1 and n is omitted for simplic-ity. The same rule is used for the dots composing each scheme. Each dot scheme (except the one marked c = 1) is composed from a number of compression stages each of three rows (except the last composed from two rows). The last row in each scheme is in general composed from a number of dots (in the left part of the row) connected with a thick line, and some isolated dots to its right. The thick line represents a binary ad-der, the connected dots represent the output variables of such an adder, the inputs (at most two for each out-put dots) are two variables (dots) per column to be found in the last compression stages composed from two lines only.The compression algorithm works as fol-lows. Starting from the first row at the top, the number of dots in each binary column is divided by three.

The (integer) quotient q represents the number of full adders needed for the compression of the column. They are represented in the three-rows stage that fol-lows by two dots: the sum in the same column of the inputs (same weight), and the carry in the column at the right of the inputs, plus the number q written next to them. A segment (representing the full adder) joins the two dots. The remainder r of the division (r = {0, 1, 2}) is represented in the third row with a dot (r = 1) or a dot with a 2 (r = 2) or without any dot (r = 0). Each binary column in a stage has a value given by the sum of the dots (at most three, some can be multiple dots) composing it. The values of the (binary) columns are decreasing through the succeeding stages. When the maximum value of the columns becomes three we use a specific new algorithm for obtaining the successive last stage composed only of columns of at most two dots.

ISSN No: 2348-4845
International Journal & Magazine of Engineering, Technology, Management and Research
A Peer Reviewed Open Access International Journal

This algorithm is very simple: starting from the right side we examine the successive columns: if their value is 1 or 2, we transfer the column to the next stage. When a 3 is found we put in the next stage a full adder. We continue with the column to the left, adding to it the carry just generated. To keep the number of dots in each column not higher than 2, a half adder must be used if a 2 is found in the preceding stage. When a 1 is found in that stage, we can simply transfer the corresponding dot to the final 2-row stage.

The whole algorithm can be implemented via a spreadsheet as shown in [8]. An interested reader can download it. The program gives also the number of full and half adders, the number of stages, the length in bits of the output of the final binary adder.In the above carry-free addition, we can also use half adders in the compression stages for obtaining a number of single bits (in the least significant part). This requires a smaller number of stages in the binary parallel adder decreasing both the cost and the total delay. The number of such single bits in the final sum is also given by the spreadsheet program.

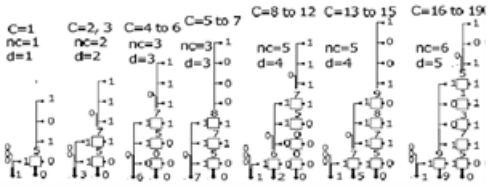


Fig. 3. The BD converters for different values C of digit-bit pairs in a column of Fig. 2 arrays.

III. THE BINARY-TO-DECIMAL CONVERSION:

The conversion from binary to decimal has been treated in [9] for the case of adding a number of BCD digits. The same methodology can be applied to the case considered here, of adding a number of BCD digit-bit couples. The conversion schemes use a cell defined by Nicoud. All modules in Fig. 3 are composed from a number of identical cells connected in a nearest-neighbor way. Each right inputs and left outputs are binary, while the upper inputs and lower outputs are BCD digits. We now briefly describe t The (upper) digit input di is multiplied by 2 and added to the binary (right) input bi obtaining S = 2di + bi. The maximum value of S is 19; its minimum is obviously 0. he algorithms implemented in a cell.

We then write the most significant digit of S (i.e. 0 or 1) at its binary output (the left side of the cell); and its least significant digit (0 to 9) at its decimal output (the lower side of the cell).The decimal input to the topmost cell can either be 3 or 4 bits. If it is 4 bits the corresponding value is 1000 or 1001. In the other case, only the bit of weight 8 in the BCD representation has value 0 and the three bits of weight 1, 2 and 4 determine a value 0, . . . , 7 (see Fig. 3 examples).Note that the binary numbers input to the BD converters correspond to the maximum values expected at the outputs of the various columns, i.e. 10, 20, 30, . . . , 140, 150, 160. These values are consequently found at the output of each scheme, at its bottom. Note also that each scheme except the first (C = 1) is valid for the range of C shown in the figure.

IV. THE ADDITION OF THE MAJOR PARTIAL PRODUCTS:

Fig. 4 shows the scheme of a 16×16 digit decimal multiplier were the Partial Product Array feeds 32 column adders, as-sumed to include the respective BD converters. The outputs of those converters are shown in a skew-tiled form and compose the Major Partial Product (MPP) array. This appears as a set of tree BCD numbers, the topmost composed by the most significant digits of the column sums, assuming the values 0 or 1 only. The digits composing the second and the third Major Partial Products are generic BCD digits. In order to obtain the sum of the three MPPs we first compress each column into an equivalent set of two digits, through a compressor whose dot-scheme is shown in Fig. 5.

Compressors in dot-schemes have been introduced . An extension to the decimal case has been shown in [9] with a family of compressors applicable to decimal columns with any number of digits, the decimal carry-free addition being the simplest case.The scheme of Fig. 5 shows an input of two BCD digits, each represented by 4 dots, and a single bits in the rightmost place, that feeds the carry input of the first stage.The scheme is composed from a 4-stage binary adder (a carry-look-ahead adder, for speed reason), and a single BD conversion cell. The cell decimal output represents the digit do having the same decimal weight of the input digits, while the single bit from the binary output represents the digit with the weight of the column input to the left of the column input to the compressor.

ISSN No: 2348-4845
International Journal & Magazine of Engineering, Technology, Management and Research
A Peer Reviewed Open Access International Journal

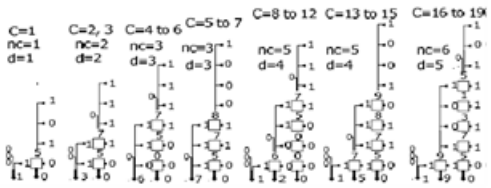


Fig. 5. A decimal carry-save adder, or column compressor.

component	delay [ps]	area [μm^2]
full-adder	100	90
half-adder	50	45
BD	50	118
CLA-4	200	200
CLA-5	210	350
CLA-6	220	415
PPG	700	155K
CPA	400	10K

TABLE I: Delay and Area for components in multiplier:

Note that the columns from i = 11 to i = 24 (see Fig. 4) can have a 1 in the first row of the MPP array. The column from i = 1 to i = 10 and from i = 25 to i = 31 (last) use the same compressor, since it is desirable to have as inputs to the final Decimal Adder one of the addends composed of 0s or 1s only. The decimal adder is in this case somewhat simpler.

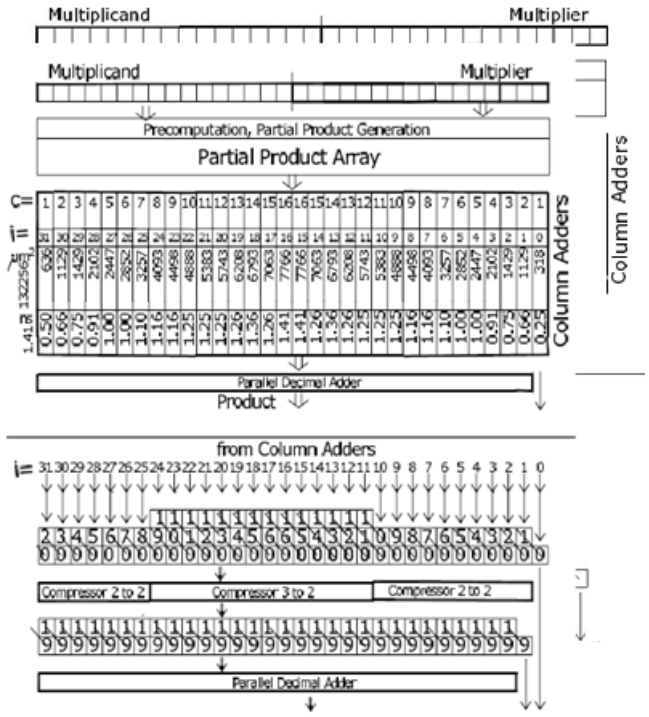


Fig. 4. Top part: the scheme for a 16x16 digit multiplier, with the delay and the area of each column. Bottom part: the maximum values (in skew form) of the outputs of the column adders and of the compressors.

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, and (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 asadjacent columns of h ODDS digits each, h being the columnheight (see Fig. 4), and h _ d þ 1. Fig. 5 shows 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. 5) reduces the h input digits and ncin input carry bits, transferred from the previous column of the binary CSA tree, to two digits, Si, Ci, with weight 10i. Moreover, a group of ncout carry outputs are generated and transferred to the next digit column of the PPR tree. Roughly, the number of carries to the next column is ncout ¼ h _ 2.The digit columns of the binary CSA tree are implemented efficiently using 4-bit 3:2, 4:2 and higher order compressors made of full adders.

These compressors take advantage of the delay difference of the inputs and of the sum and carry outputs of the full adders, allowing significant delay reductions. The weight of the carry-outs generated at the ith column, ciþ1½0_; . . . ; ciþ1½ncout _ 1_, is 16 _ 10i because the addition of the 4-bit digits is modulo 16. These carries are transferred to the ði þ 1Þth column of the PPR tree, with weight 10iþ1 ¼ 10 _ 10i.

Thus, there is a difference between the value of the carry outs generated at the i-column and the value of the carries transferred to the (i þ 1)-column. This difference, T, is computed in the sum correction block of every digit column and added to the partial product sum (S, C) in the decimal CSA.Defining

ISSN No: 2348-4845
International Journal & Magazine of Engineering, Technology, Management and Research
A Peer Reviewed Open Access International Journal

$$W_i = \sum_{k=0}^{n_{cout}-1} c_{i+1}[k],$$

The contribution of the column i to the sum correction term

T is given by

$$W_i \times 16 - W_i \times 10 = W_i \times 6.$$

Therefore, the sum correction is given by

$$T = \sum_{i=0}^{2d-1} (W_i \times 6 \times 10^i) = 6 \times \sum_{i=0}^{2d-1} W_i \times 10^i.$$

Consequently, the sum correction block evaluates Wix6.This module is composed of a m-bit binary counter and ax6 operator. A straightforward implementation would usem ¼ ncout and a decomposition of the x6 operator into x5and x1 (both without long carry propagations), and then afour to two decimal reduction to add the correction to thePPR tree result.However, to balance paths and reduce the critical path delay we considered some optimizations. Specifically, the optimized implementation of this block heavily depends on the precision of the decimal representation; therefore its implementation is merely outlined here, without going into details.

A detailed description of the implementation of the sum correction block is provided in Sections 5.1 and 5.2 for the Decimal64 and Decimal128 formats, respectively.To obtain Wi, the carries generated in the column are split into two parts: the m-bit counter adds the m first carries of the binary digit column and produces a binary sum Wmi of blog2ðm þ 1Þc bits. The counter is implemented with full adders. To reduce the delay, the different arrival times of the carries have been taken into account.Fig. 6a shows the dot-diagram representation of this reduction for a digit column with h ¼ 17 (max. column height for Decimal64).On the other hand, the remaining ncout _m carries are introduced directly into the _6 block. Note that a suitable value for m minimizes the delay overhead due to the sum correction and simplifies the logic of the _6 operation. The best value formdepends basically on h, the height of the corresponding digit column. It was first estimated using the delay evaluation model described in Section 7.1 and then validated by automated RTL synthesis of the VHDL model.

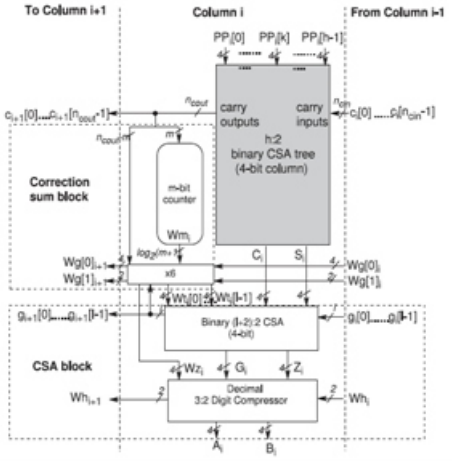
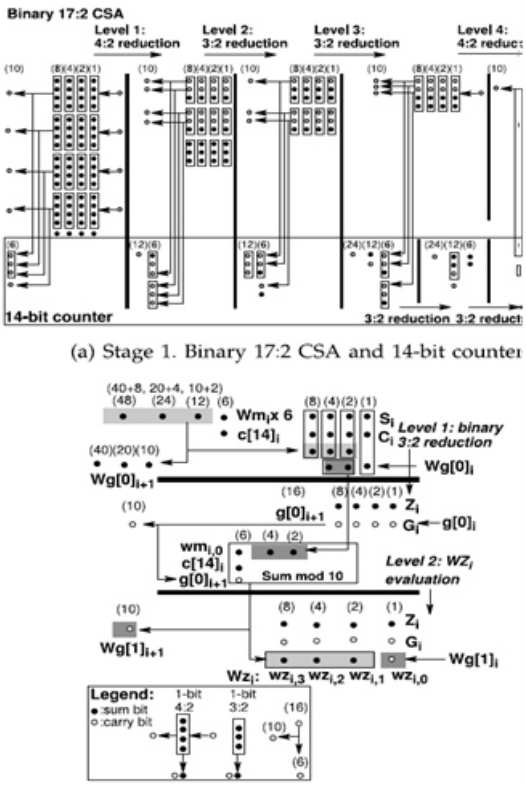


Fig. 5. High-level architecture of the proposed decimal PPR tree (h inputs, 1-digit column).

The low-level implementation details of the _6 module depend on the number of carry-outs, ncout and on the size of the counter, m, and are explained in Sections 5.1 and 5.2. However, it can be advanced that the _6 operation generates at most two carry digits Wg½0_iþ1, Wg½1_iþ1 to the next column. Moreover, to illustrate the stage, we show the corresponding dot-diagram representation for h =17 (m =14) in Fig. 6b. An efficient implementation is obtained by representing the digit of Wi x6 with l ODDS digits, Wti[0]; . . .;Wti[l-1]), being l=1 for Decimal64, and l=2 for Decimal128.

After that, the sum correction digits (Wti[0]; . . .;Wti[l-1]) and the output digits of the binary CSA tree (Si, Ci) are reduced to two ODDS digits Gi €[0; 15], and Zi €[0; 15], using a 4-bit binary ðl þ 2Þ : 2 CSA. This CSA generates l carry outs giþ1[0]; . . . ; giþ1[l -1] with weight 16 *10i, which are transferred to the next column, and introduced into the *6 block to produce another ODDS digit, Wzi €[0; 15].The last step is the addition of digits Gi; Zi;Wzi of the column, Gi þ Zi þ Wzi 2 ½0; 45_. We have designed a decimal 3:2 digit compressor that reduces digits Wzi, Gi and Zi to two digits Ai, Bi. The dot-diagram of the decimal 3:2 digit compressor is shown in Fig. 6c. To obtain the final BCD product by using a single BCD carry propagate addition,that is, P ¼ A þ B, which is the last step in the multiplication (see Fig. 1 and Section 3), it is required that Ai þ Bi 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 ½Ai_ ¼ Ai þ e in excess e ¼ 6 as defined by Equation (2), being the output digits sum [Ai]+Bi€[6; 24].

ISSN No: 2348-4845
International Journal & Magazine of Engineering, Technology, Management and Research
A Peer Reviewed Open Access International Journal



(a) Stage 1. Binary 17:2 CSA and 14-bit counter

(b) Stage 2. Decimal ×6 correction.

(c) Stage 3. Decimal 3:2.

Fig. 6. Dot-diagrams for the proposed decimal PPR (h ¼ 17 inputs, 1-digit column)

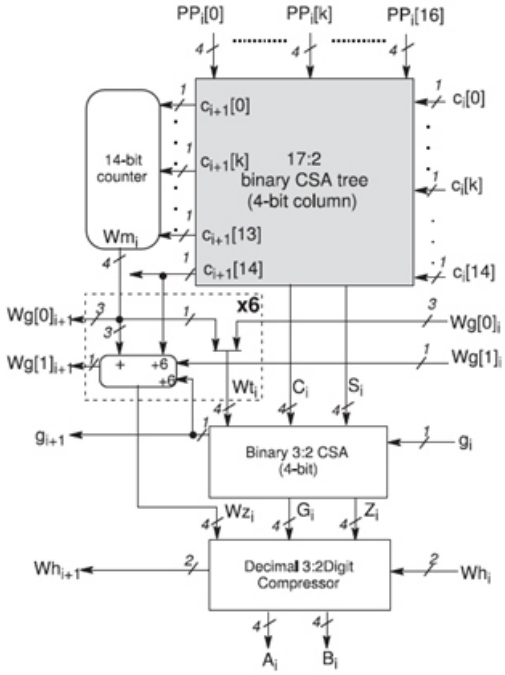
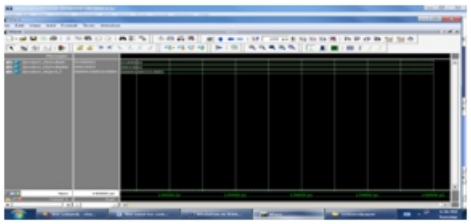


Fig. 7. Implementation of the PPR Tree Highest Column (h =17) for a 16 _ 16-digit multiplication

The evaluation is split in two parts:

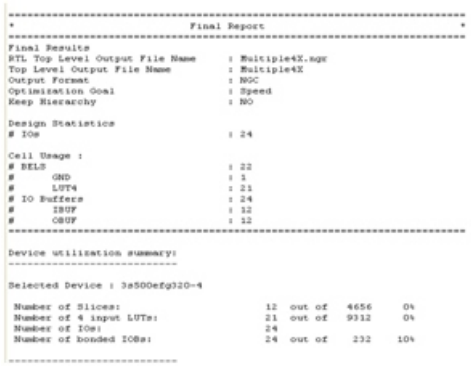
Block A computes the sum of the two MSBS of the input digits (the bits with weights 8 and 4), and a two-bit carry input Whi€{0,1, 2, 3}. This sum is in [0; 39]. The outputs ofthis block are a BCD digit Ai in excess-6 [Ai] €[6; 15] and a two-bit decimal carry output Whi+1€{0,1, 2, 3} which is transferred to the next column (the i +1th column). Note that the LSB of the carry output Whi+1 depends on the MSB of the input carry Whi. However, there is no further carry propagation since the LSB of Whiþ1 is just the LSB of [Ai+1], that is, [Ai+1,0].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[0, 9], so that Bi is evaluated as a regular binary addition.

V.RESULTS AND DISCUSSION:



ISSN No: 2348-4845
International Journal & Magazine of Engineering, Technology, Management and Research
A Peer Reviewed Open Access International Journal

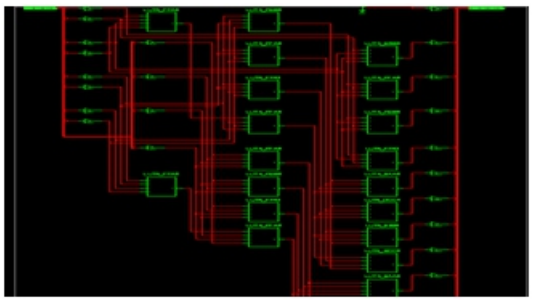
Synthesis Report:



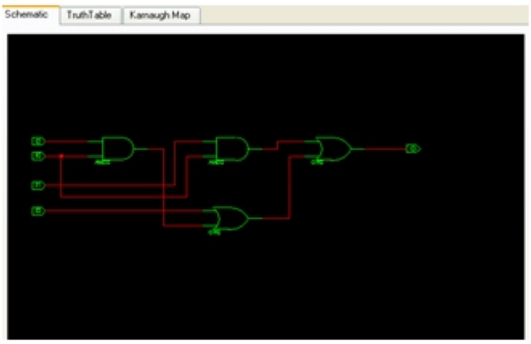
```
=========================================================================
*                            Final Report                               *
=========================================================================
Final Results
RTL Top Level Output File Name     : Multiple4X.ngr
Top Level Output File Name         : Multiple4X
Output Format                      : NGC
Optimization Goal                  : Speed
Keep Hierarchy                     : NO

Design Statistics
# IOs                              : 24

Cell Usage :
# BELS                             : 22
#      GND                         : 1
#      LUT4                        : 21
# IO Buffers                       : 24
#      IBUF                        : 12
#      OBUF                        : 12
=========================================================================

Device utilization summary:
---------------------------

Selected Device : 3s500efg320-4

 Number of Slices:                       12  out of   4656     0%
 Number of 4 input LUTs:                 21  out of   9312     0%
 Number of IOs:                          24
 Number of bonded IOBs:                  24  out of    232    10%

---------------------------
```

RTL Schematic:

Internal RTL Schematic:



LUT Diagram:



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.

REFERENCES:

[1] W. S. Brown and P. L. Richman, "The Choice of Base," Comm. of the ACM, vol. 12, pp. 560–561, Oct 1969.

[2] R. P. Brent, "On the Precision Attainable with Various Floating-Point Number Systems," IEEE Trans. Comp., vol. C, pp. 601–607, Jan 1973.

[3] R. W. Hamming, "On the Distribution of Numbers," Bell Syst. Tech. J., vol. 49, pp. 1609–1625, Oct 1970.

[4] W. Buchholz, "Fingers or fists? (The Choice of Decimal or Binary Representation)," Communications of the ACM, vol. 2, no. 12, pp. 3–11, 1959.

[5] A. Tsang and M. Olschanowsky, "A Study of Database 2 Customer Queries," IBM Technical Report 03.413, IBM, San Jose, CA, Apr 1991.

[6] ——, "A compact dot notation for the design of binary adders, multipliers and adders of products," AlaRI internal report, Dec. 2005.

[7] ——. Spreadsheet tools for the design of a parallel decimal multiplier. AlaRI internal report, Dec. 2005. [Online]. Available: http://www.alari.ch/people/dadda/

[8] ——. Spreadsheet tools for the design of a radix-10 combinational multiplier. AlaRI internal report, 2007. [Online]. Available: http://www.alari.csh/people/dadda/

[9] ——, "Multi Operand Parallel Decimal Adders: a mixed Binary and BCD Approach," IEEE Transactions on Computers, vol. 56, pp. 1320– 1328, Oct. 2007.

[10] J. Nicoud, "Iterative Arrays for Radix Conversion," IEEE Transactions on Computers, vol. C-20, pp. 1479–1489, Nov. 1971.