DECIMAL MULTIPLIER ON FPGA USING EMBEDDED BINARY MULTIPLIERS

Horácio C. Neto

INESC-ID / IST / UTL
Technical University of Lisbon
Portugal
email: hcn@inesc-id.pt

Mário P. Véstias

INESC-ID / ISEL / IPL
Polytechnic Institute of Lisbon
Portugal
email: mvestias@deetc.isel.ipl.pt

ABSTRACT

Decimal arithmetic has become a major necessity in computer arithmetic operations associated with human-centric applications, like financial and commercial, because the results must match exactly those obtained by human calculations. The relevance of decimal arithmetic has become evident with the revision of the IEEE-754 standard to include decimal floating-point support. There are already a variety of IP cores available for implementing binary arithmetic accelerators in FPGAs. Thus far, however, little work has been done with regard to implementing cores that work with decimal arithmetic. In this paper, we introduce a novel approach to the design of a decimal multiplier in FPGA using the embedded arithmetic blocks and a novel method for binary to BCD conversion. The proposed circuits were implemented in a Xilinx Virtex 4sx35ff877-12 FPGA. The results indicate that the proposed binary to BCD converter is more efficient than the traditional shift and add-3 algorithm and that the proposed decimal multiplier is very competitive when compared to decimal multipliers implemented with direct manipulation of BCD numbers.

1. INTRODUCTION

Binary arithmetic has become the prevalent method in computer arithmetic due to the simplicity of binary hardware. Despite the widespread of binary calculations, decimal operations are regaining attention because many human-centric applications, like financial and commercial, require that the results match those obtained by human calculations. Decimal arithmetic becomes a necessity when decimal fractions are involved, because these cannot be precisely represented as binary numbers. For example, the number 0.1 has a precise decimal representation but cannot be exactly represented as a binary number with a finite number of bits.

Software implementations of decimal arithmetic are typically three or four orders of magnitude slower than binary arithmetic implemented in hardware [1]. Therefore, microprocessors, such as the IBM Power6 [2], recently started to include dedicated decimal-floating point hardware units to accelerate the execution of decimal operations.

A fundamental operation of a decimal hardware arithmetic unit is multiplication. Decimal multiplication is much more complicated than binary multiplication due to the inherent difficulty to represent decimal numbers using a binary number system. Both bit and digit carries, as well as invalid results, must be considered in decimal multiplication in order to produce the correct result.

Decimal multiplication can be implemented by direct manipulation of numbers in a BCD number system. Because of the increased complexity when compared to binary multiplication, decimal multipliers are usually implemented using an iterative approach [3]. At each iteration, the multiplicand is multiplied by one digit of the multiplier to generate a partial product. Partial products are then added to produce the final result. To speed-up the addition of partial products a decimal carry-save addition was already proposed in [4]. In that work, a set of multiplicand multiples is generated in a preprocessing step. Then, in the processing step, the multiples are selectively added according to the value of the multiplier digits. Other decimal multipliers have been proposed based on sequential units, such as [5], which is similar to [4]. Parallel decimal multipliers were recently proposed to improve performance. In [6] the partial products are generated in parallel and then reduced using a decimal carry-save addition tree.

An alternative to implement BCD multipliers by direct decimal multiplication is to convert the BCD input to binary, do a binary multiplication and then convert the result back to BCD. This approach is particularly attractive with FPGA technology where embedded binary multipliers are available. A major problem of this solution has to do with the process to convert from binary to BCD, which is usually done using the shift and add-3 algorithm. For large numbers, a parallel implementation of this algorithm consumes a considerable amount of hardware resources which usually makes this decimal multiplication solution worst. Alternatively, a serial solution of the same algorithm is possible [7] with much less resources, but at the cost of long latencies.

978-1-4244-1961-6/08/$25.00 ©2008 IEEE.

Authorized licensed use limited to: Bius Jussieu. Downloaded on November 13, 2008 at 09:08 from IEEE Xplore. Restrictions apply.

Therefore, for this approach to become competitive, other more efficient binary to BCD converters must be developed.

In this paper, we propose a BCD multiplier architecture based on binary arithmetic which is able to take advantage of the efficient embedded binary multipliers available in state-of-the-art FPGAs. This approach uses a novel method to convert from binary to BCD that is more efficient than the usual shift and add-3 algorithm. The effective combination of the new binary to BCD converter with the embedded binary multipliers, makes the proposed solution quite competitive with the more common approaches based on direct BCD manipulation.

Section 2 introduces the new approach for binary to BCD conversion. This algorithm will be used in section 3 to implement the BCD multiplier. Section 4 provides area and performance results for 5×5, 4×4, 3×3 digits using one embedded 18×18 multiplier. Section 5 concludes the paper and proposes future directions for decimal multiplication based on binary multipliers.

2. BINARY-BCD CONVERSION USING BASE 1000

Binary to BCD conversions are usually implemented, either sequentially or in parallel, with the shift and add-3 algorithm [7]. In this work, we propose an implementation for the binary to BCD converter that uses base 1000 as an intermediate base. The proposed algorithm first converts the binary number to a number represented in base 1000 and then converts each base 1000 digit to BCD using the shift and add-3 algorithm.

2.1. Conversion from binary to base 1000

Base 1000 is particularly attractive since 2^{10} is close to 10^3. So, it is potentially easier to convert a number in base 2 to a number in base 1000.

In the following it is shown how to convert a binary number $b \in [0, 999 \times 999]$ to a two-digit base-1000 number d, that is

$$b = d_1 \cdot 10^3 + d_0 = d$$

Considering that

$$b = b_1 \cdot 2^{10} + b_0$$

it follows that

$$\begin{aligned} b &= b_1 \cdot 1024 + b_0 && b_1 \leq 974 = \tfrac{999 \times 999}{1024} \\ &= b_1 \cdot 1000 + \underbrace{b_1 \cdot 24 + b_0}_{c} && b_0 \leq 1023 \end{aligned} \qquad (1)$$

Now, considering c

$$\begin{aligned} c &= b_1 \cdot 24 + b_0 && c \leq 24399 && \leftarrow 15 \text{ bits} \\ c &= c_1 \cdot 1024 + c_0 && && \\ &= c_1 \cdot 1000 && c_1 \leq 23 && \leftarrow 5 \text{ bits} \\ &\quad + c_1 \cdot 24 && c_1 \times 24 \leq 23 \times 24 && \\ &\quad + c_0 && c_0 \leq 1023 && \leftarrow 10 \text{ bits} \end{aligned} \qquad (2)$$

From equations (1) and (2), b is given by:

$$b = (b_1 + c_1) \cdot 1000 + c_1 \cdot 24 + c_0 \qquad (3)$$

From equation (3), a first approximation for the two digits is:

$$\hat{d}_1 = b_1 + c_1 \leq 997 \quad \leftarrow 10 \text{ bits} \qquad (4)$$

$$\hat{d}_0 = c_1 \cdot 24 + c_0 \leq 1585 \quad \leftarrow 11 \text{ bits} \qquad (5)$$

where

$$\hat{d}_1 \in [d_1 - 1, d_1]$$
$$\hat{d}_0 \in [0, 1575]$$

From these, the final step is to test if $\hat{d}_0$ is higher than 999. If yes, $\hat{d}_1$ must be incremented by one and $\hat{d}_0$ must be increased by 24 (which is equivalent to subtracting by 1000) to adjust the result. Otherwise, the result is already correct.

An hardware implementation of the converter can be easily designed using a set of adders (see Fig. 1)

Equation (2) requires 2 shifts and two adders (one 11-bit adder and one 15-bit adder) to implement $24 \times x + y$ as $(2^4 + 2^3) \times x + y$.

$$\begin{array}{r} 0bbbb\ bbbbbb0000 \\ +\ 00bbb\ bbbbbbb000 \\ +\ 00000\ bbbbbbbbbb \\ \hline \underbrace{ccccc}_{c_1}\ \underbrace{cccccccccc}_{c_0} \end{array}$$

To evaluate,

$$\hat{d}_1 = b_1 + c_1 \leq 997 \quad \leftarrow 10 \text{ bits} \qquad (6)$$

one 10-bit adder is needed, and to evaluate

$$\hat{d}_0 = c_1 \cdot 24 + c_0 \leq 1585 \quad \leftarrow 11 \text{ bits} \qquad (7)$$

2 shifts and two adders (one 7-bit adder and one 10-bit adder) are needed.

$$\begin{array}{r} 0\ 0ccccc0000 \\ +\ 0\ 00ccccc000 \\ +\ 0\ cccccccccc \\ \hline c\ cccccccccc \end{array}$$

The hardware circuit, b2TOb1000, of figure 1 uses one adder to calculate $b_1 + c_1$, one comparator to check if $\hat{d}_0$ is

Authorized licensed use limited to: Bius Jussieu. Downloaded on November 13, 2008 at 09:08 from IEEE Xplore. Restrictions apply.

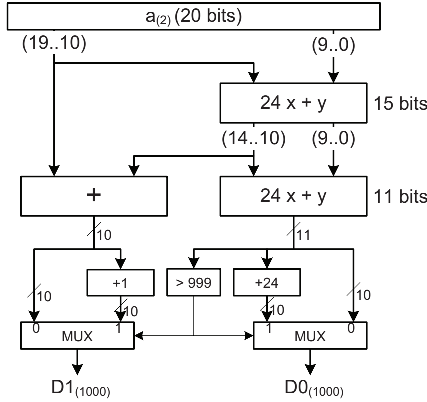

Fig. 1. Base 2 to base 1000 converter of binaries up to 999×999 - b2TOb1000.

higher than 999, one incrementer and one unit to add the constant 24 to selectively adjust the final result, and two multiplexers to select the final digit values. Two more blocks are used to calculate $24 \times x + y$.

2.2. Base 1000 conversion of numbers up to $(10^4 - 1) \times (10^4 - 1)$

The same procedure can be used to convert from larger binary numbers to numbers in base 1000. Basically, each digit of the result in base 1000, is calculated iteratively according to Horner's rule, starting with the least significant, that is:

$$b = (((d_{n-1}10^3 + d_{n-2})10^3 + ...)10^3 + d1)10^3 + d_0$$

To exemplify the process, let's consider the design of a circuit to convert a binary number $b \in [0, 9999 \times 9999]$ to a three-digit base-1000 number d, that is

$$b = (\underbrace{(d_2 10^3 + d_1)}_{\hat{d}_1} 10^3) + d_0 = d$$

Starting with the least significant digit

$$b = b_1 \cdot 2^{10} + b_0 = \hat{d}_1 \cdot 10^3 + d_0$$

so that,

$$\begin{aligned} b &= b_1 \cdot 1024 + b_0 & b_1 \leq 97636 = \tfrac{9999 \times 9999}{1024} \\ &= b_1 \cdot 1000 + \underbrace{b_1 \cdot 24 + b_0}_{c} & b_0 \leq 1023 \end{aligned} \quad (8)$$

Considering c,

$$\begin{aligned} c &= b_1 \cdot 24 + b_0 \\ & \quad c \leq (2^4 + 2^3) \times 2^{17} + b_0 < 2^{22} \quad \leftarrow 22 \text{ bits} \\ c &= c_1 \cdot 1024 + c_0 \\ &= c_1 \cdot 1000 + \underbrace{c_1 \cdot 24 + c_0}_{d} \qquad c_0 \leq 1023 \end{aligned} \quad (9)$$

so that,

$$d = (2^4 + 2^3) \times 2^{12} + c_0 \leq 99327 \leq 2^{17} \quad (10)$$

At this point, the b2TOb1000 unit described in the previous section can be used to determine the least significant digit, d_0, and part of the next digits, $\hat{d}_1$.

Formally,

$$d = \hat{d}_1 \times 10^3 + d_0, \text{ where } \hat{d}_1 \leq 100 \leq 2^7 \leftarrow 7 \text{ bits} \quad (11)$$

After the calculation of d, b is

$$b = \underbrace{(b_1 + \hat{d}_1)}_{e} \times 10^3 + d_0, \quad \text{with } e \leq 10^5 \leftarrow 17 \text{ bits} \quad (12)$$

Since e is representable with 17 bits, once again the b2TOb1000 unit can be used to determine the final two most significant base 1000 digits, that is

$$e = d_2 \times 10^3 + d_1 \quad (13)$$

After all these steps, the expected representation for b as a sum of powers of 10 is obtained.

$$b = d_2 \cdot 10^6 + d_1 \cdot 10^3 + d_0$$

where $d_0, d_1 \in [0, 999]$ and $d_2 \in [0, 99]$.

An hardware implementation of this converter can be designed using a set of adders and the b2TOb1000 unit (see Fig. 2)

The hardware circuit follows the algorithm described above. It uses two initial units $(24 \times x + y)$ to reduce the lowest part of the initial binary number up to a value manageable by the b2TOb1000 unit. Then, all values multiples of 1000 are added and sent to another b2TOb1000 unit that generates the other two digits.

The circuit has a critical path of two $(24 \times x + y)$ units with a size of 22 and 17 bits, respectively, two b2TOb1000 units and one 17 bits adder. The b2TOb1000 units are smaller than the original b2TOb1000 unit since the input and the output have only 17 bits instead of the 20 bits considered by the complete unit.

The circuit can be easily pipelined, but some care must be taken about the initial 22 bits adder in the critical path to avoid long carry propagate delays.

Authorized licensed use limited to: Bius Jussieu. Downloaded on November 13, 2008 at 09:08 from IEEE Xplore. Restrictions apply.

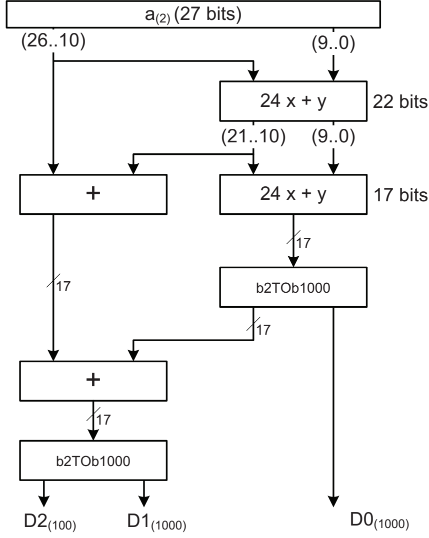

Fig. 2. Converter of a binary number up to 9999×9999 to base 1000.

2.3. Base 1000 conversion of numbers up to $(10^5 - 1) \times (10^5 - 1)$

The method introduced in the previous subsections can be applied iteratively to convert larger binary numbers to base 1000. The process will find all digits starting with the least significant.

A particularly important conversion is for binary numbers up to $(10^5 - 1) \times (10^5 - 1)$. This has to do with the fact that binary numbers up to $(10^5 - 1)$ are representable with 17 bits. Therefore, a binary multiplication of 17×17 without sign will result. Multipliers of this size are found on state-of-the-art FPGAs as embedded multipliers. So, to fully utilize these multipliers, to improve utilization ratio, a final binary to BCD converter of binary numbers up to $(10^5 - 1) \times (10^5 - 1)$ is needed.

Applying the procedure introduced in the previous subsection will result in the architecture of Fig. 3.

Three initial $(24 \times x + y)$ units are needed to reduce the lowest part of the number to a manageable number for the b2TOb1000 unit to generate the least significant digit. The values multiples of 10^3 are added and reduced with another $(24 \times x + y)$ unit followed by another b2TOb1000 unit to generate the next digit. A final b2TOb1000 unit will generate the remaining two digits.

The method can be generalized. If we look at the previous cases, we conclude that each $(24 \times x + y)$ unit reduces the lowest part of the number being processed by five bits. We need enough units to reduce the operand to less than or equal to 20 bits, in order to use the b2TOb1000 converter. Therefore, the number of serial $(24 \times x + y)$ units to be used in the calculation of each digit base 1000 depends on the number of bits of the input operand. A similar analysis can be used to determine the number of b2TOb1000 units and adders. Formally, given the size, N, of the input operand, the following equations determine the quantity of each unit type required.

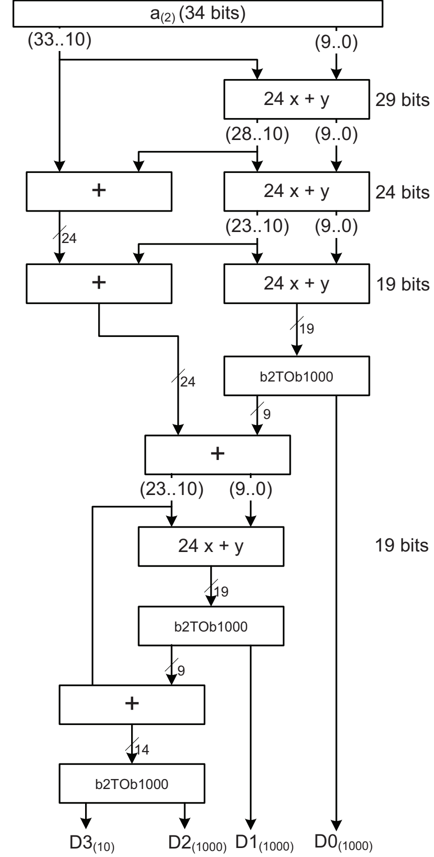

Fig. 3. Converter of a binary number up to 99999×99999 to base 1000.

$$\text{b2TOb1000 unit} \rightarrow k = int\left\lceil \frac{N}{10} \right\rceil \tag{14}$$

$$\text{(24x + y) unit} \rightarrow l = \sum_{i=0}^{\lceil \frac{N-20}{10} \rceil} \left\lceil \frac{N - 10 \times i - 20}{5} \right\rceil \tag{15}$$

$$\text{adder} \rightarrow m = \left\lceil \sum_{i=0}^{\lceil \frac{N-20}{10} \rceil} \left\lceil \frac{N - 10 \times i - 20}{5} - 1 \right\rceil \right\rceil + (k-1) \tag{16}$$

The total area A_{conv} occupied by the converter is

$$A_{conv} = k \times Area_{b2b1000} + l \times Area_{24x+y} + m \times Area_{adder}$$

The critical path includes all b2TOb1000 units, all (24x + y) units and k - 1 adders, with different sizes.

Authorized licensed use limited to: Bius Jussieu. Downloaded on November 13, 2008 at 09:08 from IEEE Xplore. Restrictions apply.

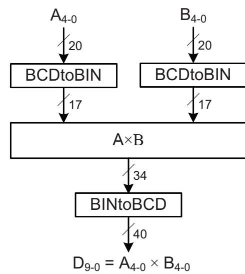

Fig. 4. BCD multiplication $A \times B$ (widths shown are for 5-digit A and B operands)

3. BCD MULTIPLIER

To implement a BCD multiplication using a binary multiplier the following steps must be considered:

1. Convert each of the BCD operands to binary
2. Multiply the binary numbers
3. Convert the binary result to BCD

This sequence of steps corresponds to the hardware implementation of Fig. 4

The BCD to binary converter calculates the binary value according to equation:

$$b = d_{n-1} \times 10^{n-1} + d_{n-2} \times 10^{n-2} + ... + d_1 \times 10^1 + d_0 \quad (17)$$

where the powers of 10 are calculated as a sum of powers of 2.

The binary to decimal converter first converts the binary value to base 1000 as explained in section 2 and then converts each base 1000 digit to three BCD digits. As this last conversion involves only 3 BCD digits it can be adequately implemented using the shift and add-3 algorithm.

Given a multiplication $A \times B$ with 5-digit operands A, B $\in [0, 99999]$, the circuit of Fig. 4 has the particular dimensions illustrated.

In this particular case, the binary multiplier is a 17x17 multiplier which, in a state-of-the-art Virtex FPGA, can be implemented with a single embedded multiplier.

For larger BCD multipliers with A and B values $\geq 10^5$, the sub-units of the architecture must be increased to support the range of values.

An alternative to increasing the sub-units is to consider a mixed BCD multiplier, in which the proposed multiplier is used to calculate partial products to be added with a carry-save BCD adder (see for example [4]). The analysis of this design space is left for future work.

Table 1. BCD to binary converter.

	LUTs	Latency	Freq
3 Digits	23	2	242 MHz
4 digits	48	2	232 MHz
5 digits	67	2	222 MHz

4. IMPLEMENTATION AND RESULTS

Three BCD multipliers for multiplications of 3×3, 4×4 and 5×5 digits were considered for testing purposes. The decimal multipliers and its sub-units were specified in VHDL, synthesized with Xilinx ISE9.2 and then implemented in a Virtex-4 SX35-12 FPGA.

Decimal to binary converters were implemented to support three different operand sizes, namely 3, 4 and 5 digit numbers.

A set of three different designs were considered for the implementation of the binary to BCD converter, namely:

I1 - A binary to base 1000 converter and then a parallel shift and add-3 algorithm for each b1000 “digit”.

I2 - A binary to decimal converter using the parallel shift and add-3 algorithm.

I3 - A binary to decimal converter using the serial shift and add-3 algorithm

For the complete decimal multiplier, two designs were tested with our architecture (see section 3, figure 4):

Mult1 - Decimal multiplier with converter I1

Mult2 - Decimal multiplier with converter I2

These designs were compared with a decimal multiplier implemented with direct manipulation of BCD numbers:

Mult3 - Decimal multiplier similar to the approach of [4]

For each circuit, an operating frequency near 200 MHz was selected (other frequencies could have been considered since the embedded multiplier supports working frequencies up 500 MHz) and the latency and the area occupation were determined.

Table 1 shows the results for the decimal to binary converter. As can be seen, the area increases almost linearly at this range of frequencies. The frequency goes from 242 to 222 MHz using two levels of pipeline.

Table 2 contains the figures for the binary to BCD implementations. As expected the serial implementation has a very low area occupation relative to the other two implementations. However, it has higher latencies and a new conversion can only start after n cycles (n is the number of bits of

Authorized licensed use limited to: Bius Jussieu. Downloaded on November 13, 2008 at 09:08 from IEEE Xplore. Restrictions apply.

Table 2. Binary to BCD converter.

	digits	bits	LUTs	Latency	Freq
I1	6	20	136	4	200 MHz
I2	6	20	213	3	230 MHz
I3	6	20	29	20	597 MHz
I1	8	27	250	5	197 MHz
I2	8	27	410	4	230 MHz
I3	8	27	39	27	597 MHz
I1	10	34	388	6	197 MHz
I2	10	34	677	5	230 MHz
I3	10	34	49	34	597 MHz

Table 3. BCD multiplier.

	digits	bits	DSP48	LUTs	Latency
Mult1	3×3	20	1	182	6
Mult2	3×3	20	1	259	5
Mult3	3×3	20	0	274	7
Mult1	4×4	27	1	346	7
Mult2	4×4	27	1	506	6
Mult3	4×4	27	0	486	8
Mult1	5×5	34	1	522	11
Mult2	5×5	34	1	811	10
Mult3	5×5	34	0	730	9

the binary number), while the other two solutions can start a new conversion at each cycle. The approach proposed herein uses less area (about 60%) than the commonly used parallel shift and add-3 algorithm. However, it needs one more pipeline level to reach the desired frequency.

Finally, table 3 presents the results for the complete BCD multiplier.

From the table it is evident that the new approach uses less resources (about 65%) than an approach using the parallel shift and add-3 algorithm but with one more pipeline level. Compared to the third solution, the new BCD multiplier uses less resources (about 70%) at the cost of one embedded binary multiplier. All implementations work at frequencies around 200 MHz. Considering that one of the objectives was to be able to take advantage of the embedded multipliers, it may be stated that, considering this assumption, this new solution is better.

Another advantage of this new multiplier is that a new calculation can begin each cycle, while the Mult3 design can begin a multiplication only after $(n + 1)$ cycles, where n is the number of digits.

5. CONCLUSION

The BCD multiplier implemented in this work, which is based on the use of embedded binary multipliers, is a quite competitive alternative to algorithms based on direct BCD manipulation. Also, our approach for binary to BCD conversion using base 1000 is more efficient than the shift and add-3 algorithm, making the solution based on binary multipliers competitive with those based on direct BCD manipulation.

The following step of this project is to analyze design alternatives for larger multipliers, using the mixed BCD multiplier mentioned above. This will allow the implementation of efficient BCD multipliers according to the IEEE-754r ongoing revision.

Finally, the proposed design should be implemented with other technologies to see if it is also advantageous when compared to direct BCD manipulation algorithms at technologies different than FPGA.

6. REFERENCES

[1] M. F. Cowlishaw, "Decimal floating-point: Algorism for computers," in *Proc. IEEE 6th IEEE Int. Symp. on Computer Arithmetic*, June 2003, pp. 104–111.

[2] "IBM Power6," IBM Corporation, May 2007, http://www2.hursley.ibm.com/decimal/.

[3] T. O. et al., "Apparatus for decimal multiplication," U.S. Patent 4 677 583, June, 1987.

[4] M. A. Erle and M. J. Schulte, "Decimal multiplication via carry-save addition," in *Proc. IEEE 14th IEEE Int. Conf. Application Specific Systems*, June 2003, pp. 348–358.

[5] M. J. S. R. D. Kenney and M. A. Erle, "High-frequency decimal multiplier," in *Proc. IEEE Int. Conf. on Computer Design: VLSI in Computers and Processors*, Oct. 2004, pp. 26–29.

[6] T. Lang and A. Nannarelli, "A radix-10 combinational multiplier," in *Proc. IEEE 40th Int. Asilomar Conf. on Signals, Systems, and Computers*, Oct. 2006, pp. 313–317.

[7] P. Alfke and B. New, "Serial code conversion between BCD and binary," in *Xilinx application note XAPP 029*, Oct. 1997.

Authorized licensed use limited to: Bius Jussieu. Downloaded on November 13, 2008 at 09:08 from IEEE Xplore. Restrictions apply.