A Radix-10 Combinational Multiplier

Tomás Lang and Alberto Nannarelli*

Dept. of Electrical Engineering and Computer Science, University of California, Irvine, USA
*Dept. of Informatics & Math. Modelling, Technical University of Denmark, Kongens Lyngby, Denmark

***Abstract*— In this work, we present a combinational decimal multiply unit which can be pipelined to reach the desired throughput. With respect to previous implementations of decimal multiplication, the proposed unit is combinational (parallel) and not sequential, has a simpler recoding of the operands which reduces the number of partial product precomputations and uses counters to eliminate the need of the decimal equivalent of a 4:2 adder. The results of the implementation show that the combinational decimal multiplier offers a good compromise between latency and area when compared to other decimal multiply units and to binary double-precision multipliers.**

I. Introduction

Hardware implementations of decimal arithmetic units have recently gained importance because they provide higher accuracy in financial applications [1]. 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 [2], [3] and [4], 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 [4] both operands are recoded in -5 to +5; 3) in the partial product generation only multiples of 5 and 2 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 [4] 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 [2] and [3]. 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. Multiplier Architecture

For the multiplication $p = x \cdot y$, we assume that both the multiplicand x and the multiplier y are sign-and-magnitude n-digit fractional numbers in BCD format normalized in $[0.1, 1.0)$. The multiplication shift-and-add algorithm is based on the identity

$$p = x \cdot y = \sum_{i=0}^{n-1} x y_i r^i$$

where for decimal operands $r = 10$, $y_i \in [0, 9]$ and x is a n-digit BCD vector. We consider in the following $n = 16$.

To avoid complicated multiples of x, we recode $y_i = y_{Hi} + y_{Li}$ with $y_H \in \{0, 5, 10\}$ and $y_L \in \{-2, 1, 0, 1, 2\}$ as indicated in Table I. With this recoding, we need to precompute only the multiples $5x$ and $2x$, while $10x$ is obtained by left-shifting x one digit. The negative values $-x$ and $-2x$ are represented in radix-10 radix-complement. Each partial product xy_i is positive and sign extension is not necessary. The partial products are accumulated by using an adder tree, and the multiplication is completed by a carry-save to BCD conversion, as shown in Fig. 1.

A. Precomputation of $2x$ and $5x$

The multiplication by 2 is straightforward. Each digit is multiplied by 2 and the carry is propagated to the next digit. The carry does not propagate any further.

In [5] the multiples $5x$ and $2x$ are used for decimal multiplication and division. However, the generation of $5x$ is performed with a carry propagation over the whole number. We now present the algorithm we use for the precomputation of $5x$ without carry propagation.

To obtain $g = 5x = 10x/2$ we perform the following two steps:

(1) $e = 10x$ (shift left one digit)

(2) $g = e/2$:

To perform this operation we divide by two each digit of e. However, since $e_i/2$ has a fractional part when e_i is odd, we have

$$\begin{array}{lcll} e_i/2 & = & f_i + h_{i+1}/2 & f_i = 0, \ldots, 4 \text{ and } h_i = 0, 1 \\ g_i & = & f_i + 5h_i & g_i = 0, \ldots, 9 \end{array}$$

That is, the algorithm to produce $g = 5x$ is

$$\begin{array}{lcll} h_{i+1} & = & 1 & \text{if } e_i \text{ (or } x_{i+1}) \text{ odd} \\ f_i & = & \lfloor e_i/2 \rfloor = \lfloor x_{i+1}/2 \rfloor & \\ g_i & = & f_i + 5h_i & \end{array}$$

y_i	y_H	y_L
0	0	0
1	0	1
2	0	2
3	5	-2
4	5	-1

y_i	y_H	y_L
5	5	0
6	5	1
7	5	2
8	10	-2
9	10	-1

TABLE I
Recoding of digits of y.

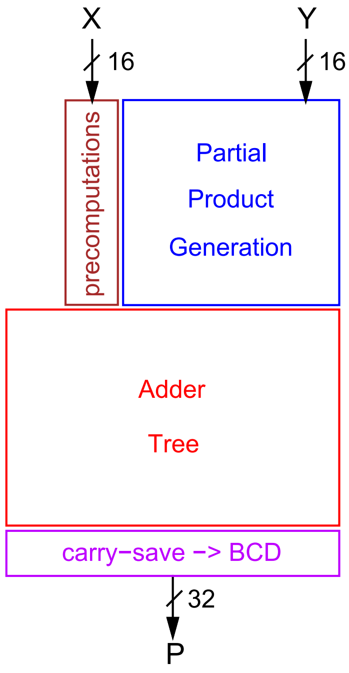


Fig. 1. Scheme of the multiplier.

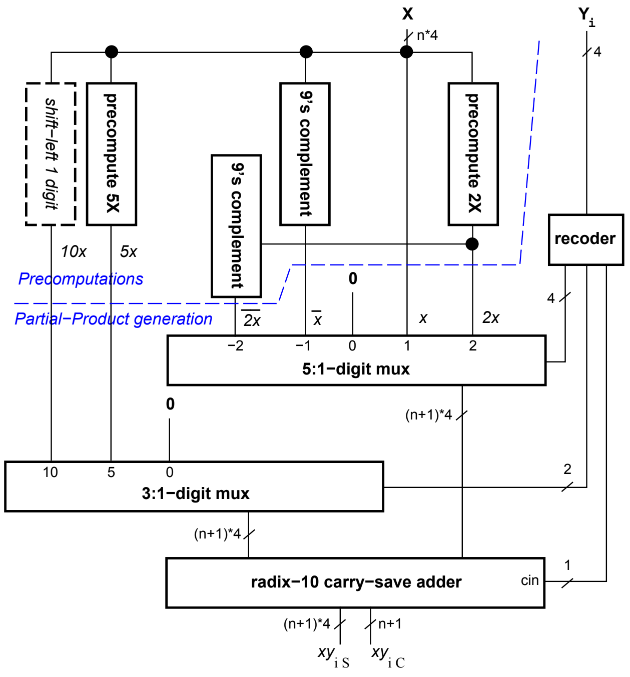


Fig. 2. Partial product generation.

B. Partial Product Generation

The recoding of Table I produces two n-digit terms for each partial product (PP). Table II shows an example of multiplication (4×4 digits). Negative numbers are represented in radix-10 complement and are implemented by doing the 9's complement of the BCD digit and adding a 1 in the least-significant position.

To reduce the delay of the additions, we perform the addition $xy_i = xy_{Hi} + xy_{Li}$ carry save and use a radix-10 carry-save representation of the partial products

$$xy_i = xy_{Hi} + xy_{Li} = \begin{cases} xy_{iS} & \in & [0,9] \\ xy_{iC} & \in & [0,1] \end{cases}$$

That is, the addition to produce a partial product results in

x :						1	9	6	3	$\times$
y :						8	1	4	5	$=$
xy_0 :	$5x$					9	8	1	5	
	0					0	0	0	0	
xy_1 :	$5x$				9	8	1	5		
	$-x$			-	1	9	6	3		
xy_2 :	x			1	9	6	3			
	0			0	0	0	0			
xy_3 :	$10x$	1	9	6	3	0				
	$-2x$	-	3	9	2	6				
		1	5	9	8	8	6	3	5	

TABLE II
EXAMPLE OF MULTIPLICATION (4×4 DIGITS).

	$n-1$	...	1	0
xy_{Hi}	xxxx	...	xxxx	xxxx
xy_{Li}	xxxx	...	xxxx	xxxx
xy_{iS}	ssss	...	ssss	ssss
xy_{iC}	c	...	c	b

b= 1 if $y_{Li} < 0$

The scheme for the generation of a partial product is shown in Fig. 2.

C. Partial Product Accumulation

The carry-free accumulation of the partial products is done using radix-10 carry-save adders (CSA), which add a carry-save operand plus another BCD operand to produce a carry-save result (see Fig. 3).

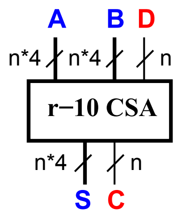
For 16-digit operands, we obtain 16 carry-save partial products. Because the radix-10 CSA reduces two BCD digits and one carry bit to one BCD digit and one carry bit, by arranging in a first-level of the tree the xy_{iS}s and by using 8 CSAs, we leave the carries (xy_{iC}s) of 8 partial products not accounted for. This is shown in Fig. 4.

These carry vectors are added by using an array of carry-counters (CC). A digit counter of this array adds 8 carries of the same weight and produces a decimal digit. That is,

inputs	8 m-bit vectors C_i	$c_j \in [0,1]$
output	1 m-digit vector Z	$z_j \in [0,8]$

as shown in Fig. 5.

By arranging the radix-10 carry-save adders and the carry-counters as in Fig. 6, we can accumulate the partial products in a 6-level tree.



	$n-1$	...	1	0
A	aaaa	...	aaaa	aaaa
B	bbbb	...	bbbb	bbbb
D	d	...	d	d
S	ssss	...	ssss	ssss
C	c	...	c	0

Fig. 3. n-digit radix-10 CSA.

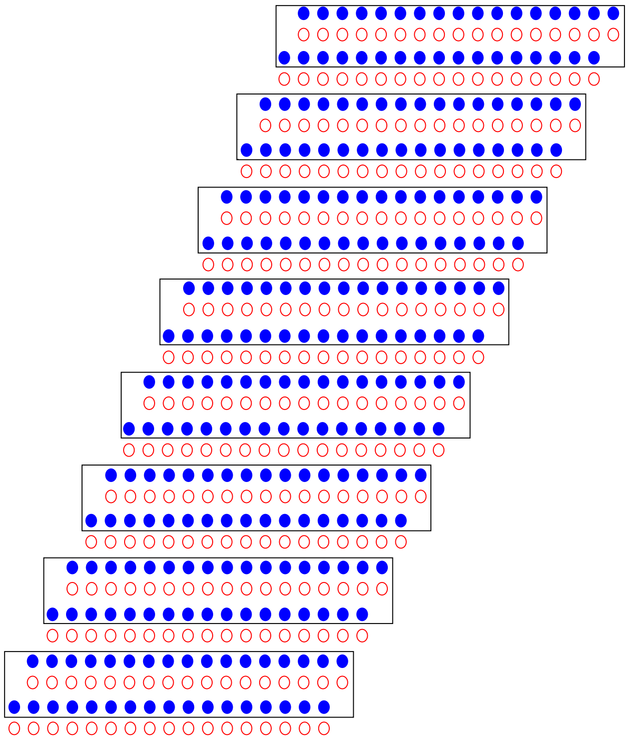
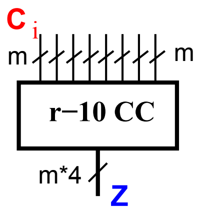


Fig. 4. Array for partial products. Solid circles indicate BCD digits, hollow circles indicate bits.

D. Radix-10 carry-save to BCD converter

The final carry-propagating addition consists in converting the radix-10 carry-save representation into the BCD one. This can be done with a simplified radix-10 CPA in which the input is just a value in radix-10 carry-save representation. The inputs are the $2n$-digit/bit outputs of the adder tree and the output is the product of x and y represented in 32-digit BCD (no truncation or rounding are considered)

inputs:	A	$2n$-digit vec. with	$a_i \in [0,9]$	
	D	$2n$-bit vec. with	$d_i \in [0,1]$	
output:	S	$2n$-digit vec. with	$s_i \in [0,9]$	(BCD)



	$m-1$	$m-2$	...	2	1	0
C_1	c	c	...	c	c	c
C_2	c	c	...	c	c	c
C_3	c	c	...	c	c	c
C_4	c	c	...	c	c	c
C_5	c	c	...	c	c	c
C_6	c	c	...	c	c	c
C_7	c	c	...	c	c	c
C_8	c	c	...	c	c	c
Z	zzzz	zzzz	...	zzzz	zzzz	zzzz

Fig. 5. m-digit radix-10 counter.

The adder can be divided into three stages. Like in radix-2, the first stage of the unit produces p and g defined as

$$p_i = \begin{cases} 1 & \text{if } a_i + d_i = 9 \\ 0 & \text{otherwise} \end{cases} \qquad g_i = \begin{cases} 1 & \text{if } a_i + d_i = 10 \\ 0 & \text{otherwise} \end{cases}$$

In parallel, the following operations are computed (modulo 10)

$$\begin{aligned} s0_i &= \langle a_i + d_i \rangle_{10} \\ s1_i &= \langle a_i + d_i + 1 \rangle_{10} \end{aligned} \qquad i = 0, 1, \ldots, 2n-1$$

A second stage consists of a parallel prefix structure to compute the carries c_i from the ps and gs and a final stage selects the precomputed digits of $s0$ or $s1$ according to the specific carry bit c_i

$$s_i = \begin{cases} s0_i & \text{if } c_i = 0 \\ s1_i & \text{if } c_i = 1 \end{cases} \qquad i = 0, 1, \ldots, 2n-1$$

This adding scheme can be generalized to a BCD carry-propagate adder by first applying a simplified radix-10 CSA[1] which reduces the two BCD vectors to a radix-10 carry-save representation. Then, the above presented converter is used to complete the carry-propagate addition.

III. Implementation and Comparisons

The 16-digit radix-10 combinational multiplier (Fig. 1) has been synthesized using the STM 90 nm CMOS standard cells library [6] and Synopsys Design Compiler. The synthesized unit has a critical path of 2.65 ns and an area of 0.3 mm^2 equivalent to the area of about $68,000$ NAND-2 gates (Table III).

The unit can be pipelined to achieve a target clock cycle T_C. For example, allowing a latch overhead of 20% of T_C, our unit can be pipelined into

$$m = \left\lceil \frac{\text{critical path}}{0.8 \cdot T_C} \right\rceil \quad \text{stages.}$$

[1]Obtained by omitting the input D in the scheme of Fig. 3.

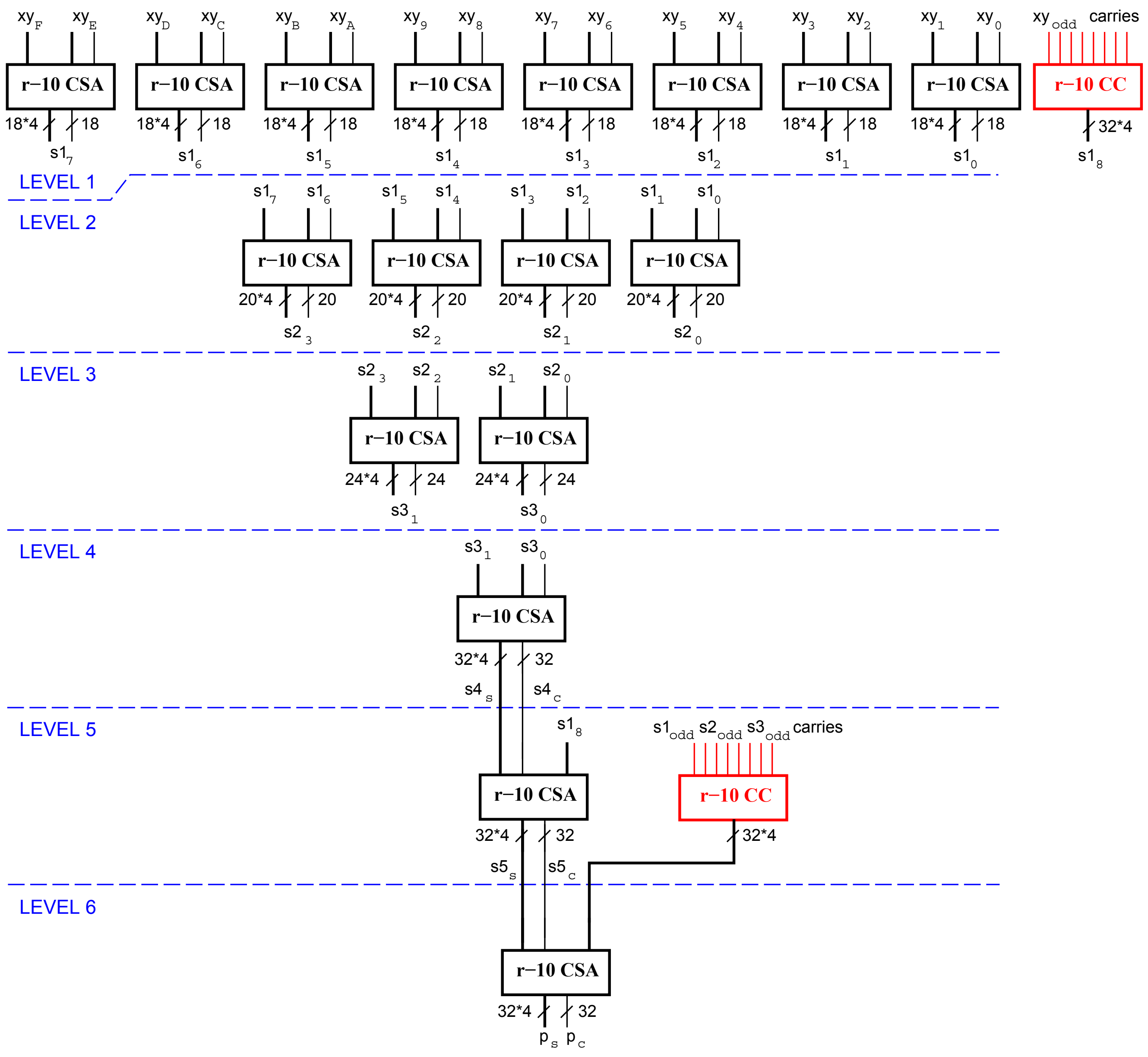


Fig. 6. Adder tree.

To clock the unit at a frequency of 1 GHz, the multiplier can be split into 4 stages with a resulting latency of 4.0 ns.

The sequential decimal multipliers of [2] and [4] for 16-digit operands have a latency of $n + 4 = 20$ cycles. The unit of [2] has been implemented in [3] in a 0.11 μm library of standard cells and the reported critical path is 0.58 ns. Because between the 0.11 μm and the 90 nm libraries, the CMOS transistor channel lenght L and the supply voltage V_{DD} scale by about the same amount

$$S = \frac{L_{(0.11\mu m)}}{L_{(90\ nm)}} = \frac{110}{90} \simeq \frac{1.2\ V}{1.0\ V} = \frac{V_{DD(0.11\mu m)}}{V_{DD(90\ nm)}}$$

we can assume the *constant field scaling* model applies. As a consequence, the critical path 0.58 ns scales to $\frac{0.58}{S} = 0.47\ ns$ in the 90 nm library. Therefore, the latency of the the radix-10 multiplier of [2] for 16-digit operands can be estimated to be $(n + 4) \cdot 0.47 = 9.5\ ns$.

The sequential multiplier of [3] has a latency of $n + 8$ cycles and is implemented in the 0.11 μm library for different operand sizes. The reported value of $T_C = 0.5\ ns$ for 16-digit operands scales to $T_C = 0.4\ ns$ in our library with a corresponding latency of $(n + 8) \cdot 0.40 = 9.6\ ns$.

By pipelining the combinational multiplier to match $T_C = 0.4\ ns$, we get 11 stages (see delays in Table III) and an operation latency of $11 \cdot 0.40 = 4.4\ ns$.

Furthermore, we compared the radix-10 combinational mul-

Block	Delay $[ns]$	Area $[\mu m^2]$
Comp. $2x$, $5x$	0.20	
PPs gen. mux	0.21	155,000
PPs gen. CSA	0.29	
Tree level 1	0.26	
Tree level 2	0.27	
Tree level 3	0.21	135,000
Tree level 4	0.27	
Tree level 5	0.29	
Tree level 6	0.21	
r-10 CS→BCD	0.40	10,000
Whole multiplier	2.65 crit. path	300,000

TABLE III

RESULTS OF IMPLEMENTATION.

	Critical path		Area	
Unit	$[ns]$	ratio	$[mm^2]$	ratio
radix-4 mult	1.40	1.00	0.20	1.00
decimal mult	2.65	1.90	0.30	1.50

TABLE IV

COMPARISON OF RADIX-4 AND RADIX-10 MULTIPLIERS.

tiplier with a binary double-precision tree-multiplier (with radix-4 recoding) which has a comparable dynamic range ($2^{53} < 10^{16}$). The latency of the binary unit is $1.4\ ns$ and its area $0.20\ mm^2$ (Table IV). By comparing the radix-10 and the binary multipliers, the binary one is about two times faster and 33% smaller.

IV. CONCLUSIONS

In this work, we presented the architecture of a radix-10 combinational multiplier and its implementation in standard cells. The partial products are generated in such a way that only multiples 2 and 5 of the multiplicand are required, and their accumulation is done by using radix-10 CSAs and counters.

The synthesized unit has a operation latency of $2.65\ ns$ and can be pipelined to obtain a target throughput.

Although it might not be reasonable to compare the latencies of combinational and sequential units, the proposed multiplier has the shortest latency when pipelined and clocked at the maximum frequency of pre-existing radix-10 sequential multipliers.

Finally, the radix-10 multiplier is compared with a binary double-precision multiplier, which is a de facto standard in most processors. The delay of the radix-10 multiplier is about twice that of the binary multiplier, and its area its about 50% larger.

REFERENCES

[1] M. F. Cowlishaw, "Decimal floating-point: algorism for computers," in *Proc. of 16th Symposium on Computer Arithmetic*, June 2003, pp. 104–111.

[2] M. Erle and M. Schulte, "Decimal Multiplication via Carry-save Addition," in *Proc. of 14th International Conference on Application-Specific Systems, Architectures and Processors*, July 2003, pp. 337–347.

[3] R. Kenney, M. Erle, and M. Schulte, "A High-Frequency Decimal Multiplier," in *Proc. of International Conference on Computer Design (ICCD)*, Oct. 2004, pp. 26–29.

[4] M. Erle, E. Schwarz, and M. Schulte, "Decimal multiplication with efficient partial product generation," in *Proc. of 17th Symposium on Computer Arithmetic*, June 2005, pp. 21–28.

[5] R. K. Richards, *Arithmetic Operations in Digital Computers*. D. Van Nostrand Company, Inc., 1955.

[6] STMicroelectronics. 90nm CMOS090 Design Platform. [Online]. Available: http://www.st.com/stonline/prodpres/dedicate/soc/asic/90plat.htm