On the design of a radix-10 online floating-point multiplier

Robert D. McIlhenny[a] and Miloš D. Ercegovac[b]

[a]California State University Northridge, 18111 Nordhoff Street, Northridge, CA 91330-8281;
[b]University of California Los Angeles, 4732 Boelter Hall, Los Angeles, CA 90095

ABSTRACT

This paper describes an approach to design and implement a radix-10 online floating-point multiplier. An online approach is considered because it offers computational flexibility not available with conventional arithmetic. The design was coded in VHDL and compiled, synthesized, and mapped onto a Virtex 5 FPGA to measure cost in terms of LUTs (look-up-tables) as well as the cycle time and total latency. The routing delay which was not optimized is the major component in the cycle time. For a rough estimate of the cost/latency characteristics, our design was compared to a standard radix-2 floating-point multiplier of equivalent precision. The results demonstrate that even an unoptimized radix-10 online design is an attractive implementation alternative for FPGA floating-point multiplication.

Keywords: Online arithmetic, radix 10, decimal, floating-point arithmetic, multiplier

1. INTRODUCTION

Decimal (radix-10) multiplication is regaining recognition as an attractive choice of radix over binary multiplication for various commercial applications, such as financial analysis, banking, tax calculation, currency conversion, insurance, and accounting.[1] It has the advantage that binary-to-decimal as well as decimal-to-binary conversion is not needed, thus avoiding conversion round-off errors. Several papers have presented efficient algorithms and/or designs for parallel multipliers[2,3,4,5]. To this date, the literature is scarce, if not absent, regarding a radix-10 online arithmetic multiplier. We are interested in online (left-to-right) multipliers because they allow concurrency between successive operations which is not possible with conventional right-to-left designs.

Online arithmetic[6] is a class of operators in which calculations are performed digit-serially, most-significant-digit-first. For all basic operators (addition, multiplication, division, square root), designs scale linearly with precision n, in other words, they grow on the order of $O(n)$. This yields generally a lower cost than for a comparable parallel operator which (with the exception of addition) grows on the order of $O(n^2)$ in terms of cost. The tradeoff is that the delay also grows on the order of $O(n)$, whereas parallel multiplication grows on the order of $O(\log n)$.

In online arithmetic, after a certain number of digits of the operands have been input, the first digit of the result is computed, and then is output on the next cycle. This delay is known as the online delay. Afterwards, successive output digits are produced one per cycle. The total latency in terms of cycles of an online operation, given operand digit precision m and online delay δ is: $T = m + \delta - 1$. If successive multiplications are performed on separate units, the performance may outperform a network of conventional multipliers which cannot initiate dependent operations as early as in the online case thus reducing the effect of linear latency. An operation with an online delay $\delta = 4$ is shown in Figure 1.

2. METHODOLOGY

Decimal floating-point multiplication ($z = xy$) is defined such that given inputs $x = X \cdot 10^{e_x}$ and $y = Y \cdot 10^{e_y}$, the output $z = Z \cdot 10^{e_z}$ is produced such that

$$\begin{aligned} Z &= XY \\ e_z &= e_x + e_y \end{aligned} \tag{1}$$

The exponent and significand calculation will be considered separately.

Send correspondence to R.McIlhenny: rmcilhen@csun.edu or M.D. Ercegovac: milos@cs.ucla.edu

Mathematics for Signal and Information Processing, edited by Mark S. Schmalz, Gerhard X. Ritter, Junior Barrera, Jaakko T. Astola, Franklin T. Luk, Proc. of SPIE Vol. 7444, 74440P · © 2009 SPIE
CCC code: 0277-786X/09/$18 · doi: 10.1117/12.826754

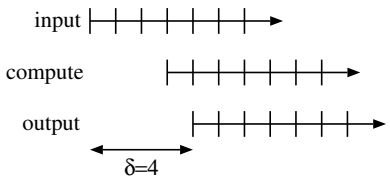

Figure 1. Online delay of a function

2.1 Exponent calculation

The exponents will be represented in binary, and the output exponent e_z will be produced using a standard parallel binary adder and computed in a single cycle. The exponent will be adjusted appropriately according to the normalization of the result significand Z.

The result will be considered

$$\begin{array}{ll} \textit{normalized} & \text{if } 10^{-1} \leq Z < 1 \\ \textit{quasi-normalized} & \text{if } 10^{-2} \leq Z < 1 \\ \textit{under-normalized} & \text{if } Z < 10^{-2} \end{array} \tag{2}$$

Except for the maximally redundant digit set $\{-9, \ldots, 9\}$, the result is guaranteed to be quasi-normalized if the leading digit is non-zero. For the digit set $\{-9, \ldots, 9\}$, this does not hold true. For example, denoting negative digit $-x$ as $\overline{x}$, $(0.1\overline{99}) = 0.001 < 10^{-2}$.

In order to represent the result as quasi-normalized, the output exponent e_z is adjusted according to the output digit z_k at each cycle k and the online delay δ for the operation. A flag *done* is set to represent that the significand digits that are output from the unit are to be considered valid from that cycle on. Previous to the *done* flag being set, output significand digits will be ignored.

$$(e_z, done) = NORM(z_k, e_z, \delta) = \begin{cases} (e_z, false) & \text{if } k < \delta \\ (e_z - 1, false) & \text{if } z_k = 0 \text{ and } k \geq \delta \text{ and } not(done) \\ (e_z, true) & \text{if } (z_k \neq 0 \text{ and } k \geq \delta) \text{ or } done \end{cases} \tag{3}$$

Note that the normalization step can be done in a separate cycle as not to affect the cycle time.

2.2 Significand calculation

The two main equations for computing the significand in an online fashion are: (1) the recurrence: and (2) the output digit selection function. Each of these is presented in detail next.

The recurrence for online multiplication Z=XY, assuming radix 10 and on-line delay δ, where the input operands X and Y, as well as the output Z are m-digit vectors consisting of digits x_i, y_i, and z_i, respectively, in which, at each step k:

$$\begin{array}{ll} X[k] = (X[k-1], x_{k+\delta-1}), & X[0] = \sum_{i=0}^{\delta-1} x_i 10^{-i} \\ Y[k] = (Y[k-1], y_{k+\delta-1}), & Y[0] = \sum_{i=0}^{\delta-1} y_i 10^{-i} \\ Z[k] = (Z[k-1], Z_k), & Z[0] = 0 \end{array} \tag{4}$$

is derived from the bound

$$|X[k]Y[k] - Z[k] < 10^{-k} \tag{5}$$

The residual is defined as

$$W[k] = 10^{-k}(X[k]Y[k] - Z[k]) \tag{6}$$

The recurrence is

$$W[k] = 10(W[k-1] - z_{k-1}) + 10^{-\delta+1}(X[k]y_{k+\delta-1} + Y[k-1]x_{k+\delta-1}) \tag{7}$$

with initial condition

$$W[0] = X[0]Y[0] \tag{8}$$

The updates of the online forms for X, Y, and Z are obtained by concatenating digits, respectively.

2.3 Output digit selection

The output digit selection function is based on a truncated version of the recurrence, denoted $\widehat{W[k]}$, and represented as:

$$z_k = SEL(\widehat{W[k]}) \tag{9}$$

The selection function is based on the symmetric digit set $\{-a, \ldots, a\}$ with selection points m_d such that if

$$m_d \leq \widehat{W_k} < m_{d+1} \tag{10}$$

then

$$z_k = SEL(\widehat{W[k]}) = d \tag{11}$$

is a valid output digit. The midpoints are chosen based on the overlap, denoted Δ between the lower bound for selecting digit d, denoted L_d and the upper bound for selecting digit $d-1$, denoted U_d, i.e.

$$L_d \leq m_d \leq U_{d-1} \tag{12}$$

The bounds are based on the error of the residual, denoted ε. For radix-10 multiplication, the error is:

$$\varepsilon = \rho(1 - 2 \cdot 10^{-\delta+1}) \tag{13}$$

where ρ is the redundancy factor of the symmetric digit set $\{-a, \ldots, a\}$ in which

$$\rho = \frac{a}{r-1} = \frac{a}{9} \tag{14}$$

Then we have the upper bounds

$$\begin{aligned} L_d &= d - \varepsilon \\ U_d &= d + \varepsilon \end{aligned} \tag{15}$$

For radix $r = 10$, various choices for the digit set are available, ranging from the minimally redundant digit set $\{-5, \ldots, 5\}$ to the maximally redundant digit set $\{-9, \ldots, 9\}$ for both the inputs and the output. Table 1 summarizes the parameters.

Table 1. Decimal online multiplication parameters

Digit set $\{-a, \ldots, a\}$	Redundancy factor ρ	Minimum online delay δ	Error ε
$\{-5, \ldots, 5\}$	5/9	3	49/90
$\{-6, \ldots, 6\}$	6/9	3	49/75
$\{-7, \ldots, 7\}$	7/9	2	28/45
$\{-8, \ldots, 8\}$	8/9	2	32/45
$\{-9, \ldots, 9\}$	1	2	4/5

As an example, the lower bound, upper bound, and selection midpoints for the digit set $\{-5, \ldots, 5\}$ are shown in Table 2. The optimal midpoints for each of the possible digit ranges are shown in Table 3. The complete algorithm for decimal on-line floating-point multiplication is shown next.

Table 2. Output digit selection midpoint for digit set $\{-5, \ldots, 5\}$

d	-4	-3	-2	-1	0	1	2	3	4	5
L_d	-409/90	-319/90	-229/90	-139/90	-48/90	41/90	131/90	221/90	311/90	401/90
U_{d-1}	-401/90	-311/90	-221/90	-131/90	-41/90	48/90	139/90	229/90	319/90	409/90
m_d	-9/2	-7/2	-5/2	-3/2	-1/2	1/2	3/2	5/2	7/2	9/2

Table 3. Optimal midpoints for output digit selection

Digit set $\{-a, \ldots, a\}$	Optimal midpoint m_d
$\{-5, \ldots, 5\}$	$d - 1/2$
$\{-6, \ldots, 6\}$	$d - 3/5$
$\{-7, \ldots, 7\}$	$d - 3/5$
$\{-8, \ldots, 8\}$	$d - 3/5$
$\{-9, \ldots, 9\}$	$d - 4/5$

Decimal Online Floating-Point Multiplication

$\{Initialization\}$
$e_z = e_x + e_y$
$W[-\delta + 1] = 0$
$X[-\delta + 1] = 0$
$Y[-\delta + 1] = 0$
$z_0 = 0$

for $k = -\delta + 2$ to 0 do
$X[k] = X[k-1] + x_{k+\delta-1} 10^{-k-\delta+1}$
$W[k] = 10(W[k-1]) + 10^{-\delta+1}(X[k] y_{k+\delta-1} + Y[k-1] x_{k+\delta-1})$
$Y[k] = Y[k-1] + y_{k+\delta-1} 10^{-k-\delta+1}$
end for

$\{Recurrence\}$
for $k = 1$ to m do
$X[k] = X[k-1] + x_{k+\delta-1} 10^{-k-\delta+1}$
$W[k] = 10(W[k-1] - z_{k-1}) + 10^{-\delta+1}(X[k] y_{k+\delta-1} + Y[k-1] x_{k+\delta-1})$
$Y[k] = Y[k-1] + y_{k+\delta-1} 10^{-k-\delta+1}$
$z_k = SEL(\widehat{W[k]})$
$(e_z, done) = NORM(z_k, e_z, \delta)$
end for

3. IMPLEMENTATION

The main components of a digit slice of a decimal online floating-point multiplier are: (i) two digit-by-digit multipliers to compute carry digit $cx_{k,i}$ and product digit $px_{k,i}$ of the digit-by-vector product $X[k]y_{k+\delta-1}$ and carry digit $cy_{k,i}$ and product digit $py_{k,i}$ of the digit-by-vector product $Y[k-1]x_{k+\delta-1}$, (ii) a 6-to-2 digit adder for producing individual digit $w_{k,i}$ of the residual $W[k]$, (iii) and various digit-wide registers for storing digits of the residual. Due to a lower cost of the digit-by-digit multipliers, compared to the other digit sets, the digit set $\{-5, \ldots, 5\}$ will be the choice for implementation.

3.1 Digit-set encoding

For efficient implementation, encoding of the digits is essential. Since the range for $\{-5,\ldots,5\}$ is 11, a minimum of $\log_2 11 = 4$ bits is necessary. A simple encoding is an extension of two's complement representation for the digits. This is denoted as (-8)421 encoding, as shown in Table 4, where the value of a digit $x_k = (x_{k,3}, x_{k,2}, x_{k,1}, x_{k,0}) = -8x_{k,3} + 4x_{k,2} + 2x_{k,1} + x_{k,0}$. Assuming that the initial input will be in BCD format, and the final output should be in BCD format, conversion is necessary to and from the redundant digit set $\{-5,\ldots,5\}$, denoted RDS_5. Each conversion method will be described in detail.

Table 4. (-8)421 encoding of digits

Digit	-5	-4	-3	-2	-1	0	1	2	3	4	5
(-8)421 encoding	1011	1100	1101	1110	1111	0000	0001	0010	0011	0100	0101

3.2 BCD to RDS_5 recoding

Given a BCD input digit x_k, recoding to a RDS_5 digit χ_k is performed according to the algorithm $RECODE(x_k)$. The digit χ_k is represented as a pair of digits (t_{k-1}, s_k), in which $t_{k-1} \in \{0,1\}$ and $s_k \in \{-5,\ldots,4\}$, and $x_k = 10t_{k-1} + s_k$. Then $\chi_k = t_k + s_k$.

$RECODE(x_k)$

$$(t_{k-1}, s_k) = \begin{cases} (0,0) & \text{if } x_k = 0 \\ (0,1) & \text{if } x_k = 1 \\ (0,2) & \text{if } x_k = 2 \\ (0,3) & \text{if } x_k = 3 \\ (0,4) & \text{if } x_k = 4 \\ (1,\overline{5}) & \text{if } x_k = 5 \\ (1,\overline{4}) & \text{if } x_k = 6 \\ (1,\overline{3}) & \text{if } x_k = 7 \\ (1,\overline{2}) & \text{if } x_k = 8 \\ (1,\overline{1}) & \text{if } x_k = 9 \end{cases}$$

$\chi_k = t_k + s_k$

3.3 On-the-fly conversion

Given a RDS_5 digit z_k, on-the-fly conversion (OFC) to a vector of BCD digits, denoted ζ is performed according to the following algorithm.[6]

$OFC(z_k)$

$$A[k] = \begin{cases} A[k-1] + z_k 10^{-k} & \text{if } z_k \geq 0 \\ B[k-1] + (10 - |z_k|)10^{-k} & \text{if } z_k < 0 \end{cases}$$

$$B[k] = \begin{cases} A[k-1] + (z_k - 1)10^{-k} & \text{if } z_k > 0 \\ B[k-1] + (9 - |z_k|)10^{-k} & \text{if } z_k \leq 0 \end{cases}$$

Initially $A[0] = 0$ and $B[0] = 1$. Assuming m digit precision, the final significand $\zeta = A[m]$.

3.4 Digit-by-digit multiplier

A digit-by-digit multiplier ($\otimes$)takes digit $x_{k,i}$ of weight 10^{-i} and digit $y_{k,j}$ of weight 10^{-j} and produces digit $c_{k,i+j-1}$ of weight 10^{-i-j+1} and digit $p_{k,i+j}$ of weight 10^{-i-j}, in which the digit product $x_{k,i}y_{k,j} = 10c_{k,i+j-1} + p_{k,i+j}$. Different possible values of input and output digits are shown in Table 5. To simplify the logic, the absolute values of the input digits are generated before multiplication. Appropriate negation of the output digits is applied afterwards. In other words, we have the following digit-by-digit multiplication operation:

$$\begin{aligned} & x_i^* = |x_i|, y_j^* = |y_j| \\ & neg = \begin{cases} -1 & \text{if } (x_i < 0 \text{ and } y_j \geq 0) \text{ or } (x_i \geq 0 \text{ and } y_j < 0) \\ 1 & \text{otherwise} \end{cases} \\ & 10c_{k,i+j-1} + p_{k,i+j} = (neg)(x_i^* \cdot y_j^*) \end{aligned} \tag{16}$$

Table 5. Digit-digit multiplication for digit set $\{-5, \ldots, 5\}$

x_i^*,y_j^*	1	2	3	4	5
1	0,1	0,2	0,3	0,4	0,5
2	0,2	0,4	1,-4	1,-2	1,0
3	0,3	1,-4	1,-1	1,2	1,5
4	0,4	1,-2	1,2	2,-4	2,0
5	0,5	1,0	1,5	2,0	2,5
	$c_{k,i+j-1}$,$p_{k,i+j}$				

3.5 6-to-2 digit adder

To produce the residual requires at each digit slice the addition of six digits. The sum will be represented in a modified carry-save format consisting of a carry digit and a sum digit. The choice of a carry-save format is to make the delay independent of the precision of the operation. Each 6-to-2 digit adder adds: (i) carry digit $wc_{k-1,i}$ and sum digit $ws_{k-1,i}$ from the previous residual $W[k-1]$, (ii) carry digit $xc_{k,i}$ and product digit $xp_{k,i}$ from the preceding digit-by-digit multiplier which multiplies an individual digit x_i from the vector $X[k]$ with digit $y_{k+\delta-1}$, and (iii) carry digit $yc_{k,i}$ and product digit $yp_{k,i}$ from the digit-by-digit multiplier which multiplies an individual digit y_i from the vector $Y[k-1]$ with digit $x_{k+\delta-1}$, to produce sum $w_{k,i}$.

The result $w_{k,i}$ can be converted to carry digit $wc_{k,i}$ and sum digit $ws_{k,i}$. For the digit set $\{-5, \ldots, 5\}$ the conversion is based on the following:

$$(wc_{k,i}, ws_{k,i}) = \begin{cases} (-2, w_{k,i} + 20) & \text{if } w_{k,i} < -15 \\ (-1, w_{k,i} + 10) & \text{if } -15 \leq w_{k,i} < -5 \\ (0, w_{k,i}) & \text{if } -5 \leq w_{k,i} \leq 5 \\ (1, w_{k,i} - 10) & \text{if } 5 < w_{k,i} \leq 15 \\ (2, w_{k,i} - 20) & \text{if } w_{k,i} > 15 \end{cases} \tag{17}$$

The complete diagram of a decimal online floating-point multiplier consisting of M modular slices, representing m-digit precision significand calculation, is shown in Figure 2.

4. RESULTS

The design of a decimal online floating-point multiplier was coded in VHDL and compiled, synthesized, and mapped using the Xilinx Foundation 10.1 design tool. It was mapped to a Virtex 5 FPGA to measure cost in terms of LUTs (5- and 6-input look-up tables). The count and cost of individual components is shown in Table 6, assuming a binary e-bit exponent and a radix-10 m-digit significand. The logic delay in the main slice is

$$T_{logic} = t_L + t_{DM} + t_{ADD} + t_{SEL} = 0.396 + 0.258 + 0.086 + 0.644 = 1.384ns \tag{18}$$

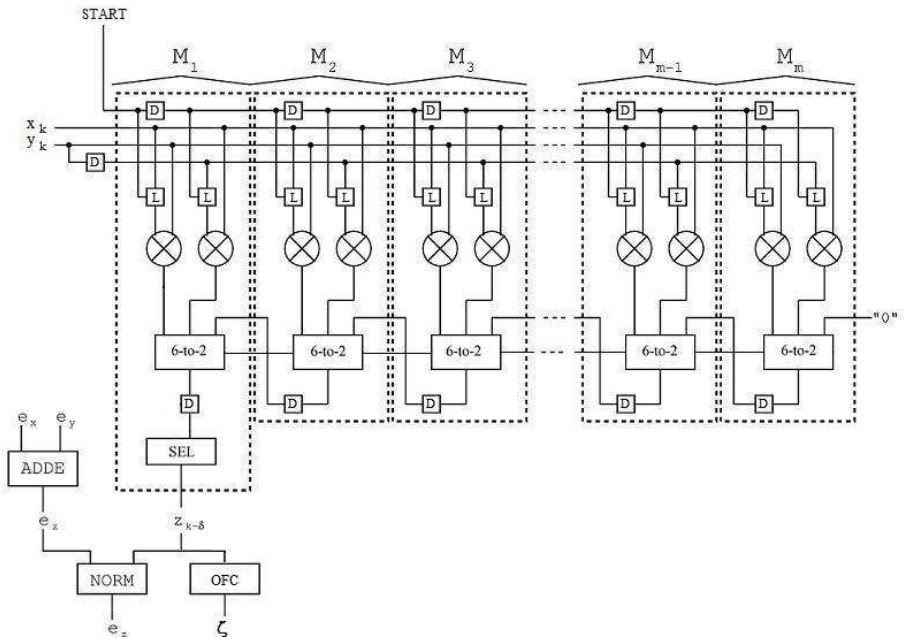

Figure 2. Decimal online floating-point multiplier

The normalization logic has a delay of $t_{NORM} = 1.802ns$. The slice cycle time, dominated by the delay due to unoptimized routing, is:

$$T_{cycle} = t_L + t_{DM} + t_{ADD} + t_{SEL} + t_{ROUTE} = 0.396 + 0.258 + 0.086 + 0.644 + 4.837 = 6.221ns \tag{19}$$

These numbers are based on results obtained from synthesizing the design, without any optimizations in the layout taken into account. Improved results may be obtained by optimizing the floorplanning. Note a rather large penalty in the routing delay. For delay comparisons, the latency in terms of the total number of cycles, the cycle delay, and the total delay are included.

Table 6. Cost of a decimal online floating-point multiplier

Module	Count	LUTs	Delay (ns)
Exponent adder (ADDE)	1	e	$t_E = 0.094$
Latches (L) and D flip-flops (D)	$4m + 1$	n/a	$t_L = 0.396$
Digit-By-Digit multiplier ($\otimes$)	$2m$	$36m$	$t_{DM} = 0.258$
6-to-2 digit adder (6-to-2)	m	$31m$	$t_{ADD} = 0.086$
Output digit selection unit (SEL)	1	7	$t_{SEL} = 0.644$
Normalization unit (NORM)	1	3e	$t_{NORM} = 1.802$
On-the-fly conversion unit (OFC)	1	8m	$t_{OFC} = 1.035$
Routing logic	—	60	$t_{ROUTE} = 4.836$
Total cost		$93m + 4e + 23$	

The design of a decimal online floating-point multiplier for single-precision ($m = 8$, $e = 7$) and double-precision ($m = 16$, $e = 9$), denoted *r10-olfp* is compared to a radix-2 parallel floating-point multiplier, denoted *r2-fp* for both single-precision ($m = 24$, $e = 8$) and double-precision ($m = 53$, $e = 11$) within the LogicCORE family of modules, optimized for the Xilinx Virtex-5 FPGA. The results are summarized in Table 7.

Table 7. Comparison of decimal online floating-point multiplier and binary parallel floating-point multiplier

Precision	Cost (LUTs)		Latency (cycles)		Cycle delay (ns)		Total delay (ns)	
	r2-fp	r10-olfp	r2-fp	r10-olfp	r2-fp	r10-olfp	r2-fp	r10-olfp
Single	627	795	11	10	2.994	6.221	32.934	62.210
Double	2296	1547	27	18	4.219	6.221	113.913	111.978

5. SUMMARY

In summary, the methodology and design of a decimal online floating-point multiplier were presented. As a rough estimate of its cost/performance, we compared our design to a binary parallel floating-point multiplier of equivalent precision: our design has a slightly larger cost for single-precision, but a significantly lower cost for double-precision. It terms of latency, it has a significantly larger total delay for single-precision, but a slightly smaller total delay for double-precision. The total delay is expected to significantly improve if the floorplanning is optimized. The proposed design is attractive for overlapped arithmetic operations to increase the overall performance of a group of arithmetic operations.

REFERENCES

1. M. Erle, E. Schwartz, and M. Schulte, "Decimal multiplication with efficient partial product generation," *17th Symposium on Computer Arithmetic* , pp. 21–28, 2005.
2. M. Cowlishaw, "Decimal floating-point: algorism for computers," *16th IEEE Symposium on Computer Arithmetic* , pp. 104–111, 2003.
3. M. Erle and M. Schulte, "Decimal multiplication via carry-save addition," *14th IEEE International Conference on Application-Specific Systems, Architectures and Processors (ASAP'03)* , pp. 348–358, 2003.
4. R. Kenney, M. Schulte, and M. Erle, "A high-frequency decimal multiplier," *2004 IEEE International Conference on Computer Design (ICCD'04)* , pp. 26–29, 2004.
5. H. Neto and M. Vestias, "Decimal multiplier on FPGA using embedded binary multipliers," *International Conference on Field Programmable Logic and Applications* , pp. 197–202, 2008.
6. M. Ercegovac and T. Lang, *Digital arithmetic*, Morgan Kaufman Publishers, 2004.