2009 20th IEEE International Conference on Application-specific Systems, Architectures and Processors

Division Unit for Binary Integer Decimals

Tomás Lang and Alberto Nannarelli*

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

Abstract

In this work, we present a radix-10 division unit that is based on the digit-recurrence algorithm and implements binary encodings (Binary Integer Decimal or BID) for significands. Recent decimal division designs are all based on the Binary Coded Decimal (BCD) encoding. We adapt the radix-10 digit-recurrence algorithm to BID representation and implement the division unit in standard cell technology. The implementation of the proposed BID division unit is compared to that of a BCD based unit implementing the same algorithm. The comparison shows that for normalized operands the BID unit has the same latency as the BCD unit and reduced area, but the normalization is more expensive when implemented in BID.

1. Introduction

In recent years, the miniaturization of devices and the resulting extra space available on silicon, made possible to implement decimal processors in hardware. Companies such as IBM are already commercializing processors which include decimal arithmetic units [1] [2]. Consequently, during the revision of IEEE standard 754 for floating-point representation, support for decimal representation was added to the binary one. In the revised IEEE standard 754 [3], the significand of decimal floating-point numbers can be represented in two different formats: Densely Packed Decimal (DPD) format or Binary Integer Decimal (BID) format.

In the DPD format, a word representing the significand of a decimal number is divided into 10 bit fields, called *"declets"*, representing 3 decimal digits each. The conversion from declet to Binary Coded Decimal (BCD) format, and vice versa, can be implemented with simple logic [4].

In the BID format, a word represents the significand of a decimal number as an unsigned integer in binary. For example, the decimal floating-point number 0.125 is represented in BID by 125×10^{-3} with significand $0\ldots0\ 0111\ 1101_2$ and exponent $-3 + bias$.

In decimal processors implementing the DPD/BCD format, operations such as operand shifting, i.e. radix multiplication, are easy to implement. On the other hand, addition and multiplication present overhead in BCD over their binary counterparts.

In the BID format, addition and multiplication are binary operations without overhead. However, when normalization for floating-point operands is required, a multiplication by 10^k, which can be implemented by a table and a multiplier or by a special radix-10 shifter [5], is needed. Right shift is even more complicated as it requires a division by 10^k.

Binary division is implemented in hardware in the most popular general-purpose processors. Division is implemented by two classes of algorithms: approximation methods (Newton-Raphson, Goldschmidt) or digit-recurrence [6]. Recently, a number of decimal implementations of division all based on BCD operand representation has been presented. An iterative divider based on the Newton-Raphson approximation has been presented in [7]. On the other hand, examples of decimal digit-recurrence division are described in [8], [9] and [10].

In this work, we implement a radix-10 digit-recurrence division unit for the Binary Integer Decimal (BID) format. To our knowledge, this is the first implementation of decimal division in BID. A BID floating-point adder and multiplier were presented in [11] and in [12].

Comparing to the corresponding radix-10 divider with BCD format [9], the following modifications are described: the normalization of operands using a rectangular multiplier, the recurrence using binary carry-save adders and reducing the iteration delay, the selection function for binary divisor, and the on-the-fly conversion from decimal signed-digit representation to binary, including rounding. The resulting unit has been synthesized and its delay and area compared to the corresponding BCD unit.

2. Decimal Division Algorithm for BID

The division $q = x/d$ is implemented by the radix-r digit-recurrence iteration [6]

$$w[j+1] = rw[j] - q_{j+1}d \qquad j = 0, 1, 2, \ldots \qquad (1)$$

where d is the divisor, $w[j]$ is the residual at iteration j and it is initialized with the dividend x. The quotient-digit q_{j+1} is computed at each iteration by a selection function

$$q_{j+1} = SEL(\hat{d}, \widehat{rw[j]})$$

from the truncated normalized divisor and the truncated residual.

1063-6862/09 $25.00 © 2009 IEEE
DOI 10.1109/ASAP.2009.42

Since in the IEEE Standard for decimal representation the operands are not normalized, to apply the algorithm it is necessary to normalize the divisor. Moreover, to reduce the number of leading zeros in the quotient, it is also convenient to normalize the dividend. In the sequel, d and x correspond to the normalized divisor and dividend, respectively.

For the decimal (radix-10) case, as done for BCD representation [9], the quotient-digit is split into two parts q_H and q_L such that

$$q_{j+1} = 5q_{Hj+1} + q_{Lj+1}$$

with digit sets $q_H = \{-1, 0, 1\}$ and $q_L = \{-2, -1, 0, 1, 2\}$, and a redundancy factor $\rho = 7/9$.

By the quotient-digit decomposition, we obtain from (1) the two recurrences

$$\begin{aligned} v[j] &= 10w[j] - q_{Hj+1}(5d) \\ w[j+1] &= v[j] - q_{Lj+1}d \end{aligned} \tag{2}$$

with quotient digit selection functions

$$\begin{aligned} q_{Hj+1} &= SEL_H(\widehat{10w}, \widehat{d}) \\ q_{Lj+1} &= SEL_L(\widehat{v}, \widehat{d}) \end{aligned}$$

The digit-recurrence algorithm converges if

$$|w[j]| \leq \rho d \tag{3}$$

Therefore, to ensure convergence for the given redundancy, the recurrence is initialized with a scaled value of the dividend such that $w[0] = \text{scaled}(x) < \rho d$.

The algorithm is completed by a conversion-and-round unit that converts the digits q_H and q_L from signed-digit to the required representation (sign-and-magnitude for floating-point quotient) and performs the rounding.

The architecture of the BID divider is shown in Figure 1. The input significands are two BID integers M_x and M_d. The inputs to the recurrence are the normalized BID representation x and d. The output from the convert-and-round unit is the non normalized BID quotient M_q. The unit is completed by the logic to compute the sign ($S_q = S_x \oplus S_d$), a controller and some controller signals not shown in Figure 1.

As indicated before, the operands should be normalized. This is performed by multiplying the operands by the required powers of 10. This corresponds to simple shifts in the BCD representation, but is more complicated in the Binary Integer Decimal (BID) decimal64 format. In this case, the normalization process consists of the following two steps:

1) Obtain the power of ten required for the normalization. To simplify this function, we use as input the number of leading zeros in the binary representation. Because of this, the normalized value can overflow by one bit.
2) Multiply the operand by the corresponding power of ten. This can be done by using a multiplier or a decimal shifter.

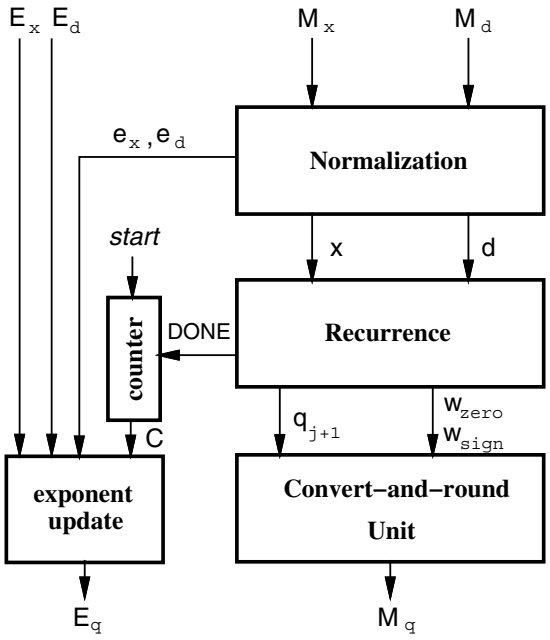

Figure 1. Architecture of BID significand divider.

In addition to the actual normalization, it is necessary to satisfy the condition for convergence. In the BCD case we divided the normalized dividend by 100 to achieve this. However, for the binary case, this division by 100 is not satisfactory when the original dividend is already normalized. Consequently, we normalize the divisor to a larger value so that the condition of convergence is satisfied. That is, the normalization is described as follows:

- For the dividend we obtain the normalized value by multiplying the integer dividend significand M_x by 10^{e_x} such that $0.1 \leq M_x \cdot 10^{e_x} \cdot 2^{-54} < 2$.
- Similarly, for the divisor we multiply M_d by 10^{e_d} such that $0.1 \leq M_d \cdot 10^{e_d} \cdot 2^{-(54+s)} < 2$, where s is the number of additional bits of the normalized divisor.

The value of s is determined so that we use the normalized x as the initial residual $w[0]$ and assure convergence. Specifically, we need that

$$\frac{x_{max}}{d_{min}} < \rho \tag{4}$$

where x and d are the normalized dividend and divisor, respectively. So, if we normalize d to s additional bits, we get

$$\frac{2}{0.1 \cdot 2^s} < 7/9 \tag{5}$$

resulting in $s \geq 5$.

That is, the normalized dividend x has 54+1=55 bits (bits 0 to 53 plus bit 54 if there is an overflow in the normalization) with up to five most-significant zeros and the divisor has 54+5+1=60 bits (bits 0 to 58 plus bit 59 if there

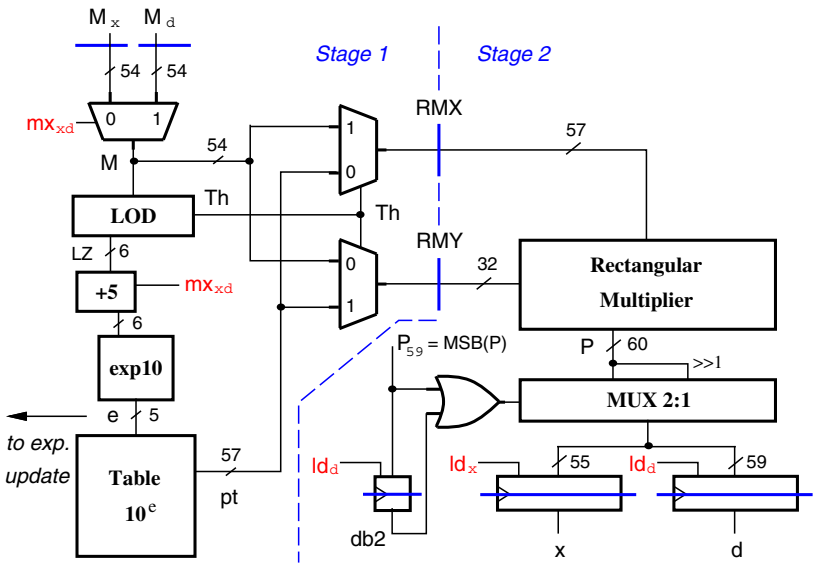

Figure 2. Normalization unit. The thicker marks indicate the position of the registers.

is an overflow), again with up to 5 most-significant zeros. To reduce the range of the divisor, in case of overflow (bit 59 equal to 1) both the divisor and the dividend are shifted one bit to the right, resulting in a normalized divisor with 59 bits (bits 0 to 58).

The operation stops when the result is exact ($w[j] = 0$) or after 17 iterations (the 17th iteration is used to determine the rounding digit). Calling C the number of iterations (up to 16), the exponent of the quotient is obtained as

$$E_q = (E_x - e_x) - (E_d - e_d) + 16 - C$$

where E_x and E_d are the exponent of the operands and 10^{e_x} and 10^{e_d} are the powers of ten used for the normalization.

3. Normalization

The normalization unit of Figure 2 is divided into two stages:

- In **Stage 1**, the number of leading zeros of each significand is obtained. If the operand is the dividend this number of leading zeros is used to obtain from a table the power of ten required to normalize it. On the other hand, if the operand is the divisor, we add five to the number of zeros and obtain the corresponding power of ten from the table.
- In **Stage 2**, the multiplication of the operands stored in registers RMX and RMY is performed. The 57×32 bit multiplier produces a 60-bit product P (57 bits are required to represent the largest power of ten, namely, 10^{17}). In case the normalized value of d overflows (bit 59 is equal to 1) both d and x are divided by two (shifted one bit to the right).

  We opted for a rectangular multiplier and operand swapping because one of the factors (either the power of ten or the non normalized operand) is less than 30 bits while the other is larger but at most 57 bits. We send to the multiplicand input (57-bit) the larger between the two operands.

The operations in **Stage 1** can be detailed as:

1) Control signal mx_{xd} selects which significand M is to be normalized.
2) Leading one detection (LOD) for M. This produces the number of leading zeros LZ. If the significand corresponds to the divisor add five to LZ. Furthermore, the LOD sets the bit Th if $M \geq 2^{30}$. The value of LZ is used to obtain e, such that multiplying by 10^e produces the normalized significand.
3) Signal e addresses the table to determine the corresponding power of 10 to be used in the multiplier.

$$pt = \text{Table}10^e(e)$$

4) Based on the size of the significand and power of 10, signal Th selects which operands are multiplicand and multiplier.

$$\text{RMX} = \begin{cases} M \\ pt \end{cases} \quad \text{RMY} = \begin{cases} pt & \text{if } Th = 1 \\ M & \text{otherwise} \end{cases}$$

The operations in **Stage 2** are:

1) Multiplication $P = M \times pt$.
2) If during the computation of d, the most-significant bit of P $P_{59} = 1$ then P is shifted one position to the right:

$$d \leftarrow \begin{cases} P >> 1 & \text{(shift 1-bit to right)} \quad \text{if } P_{59} = 1 \\ P & \text{otherwise} \end{cases}$$

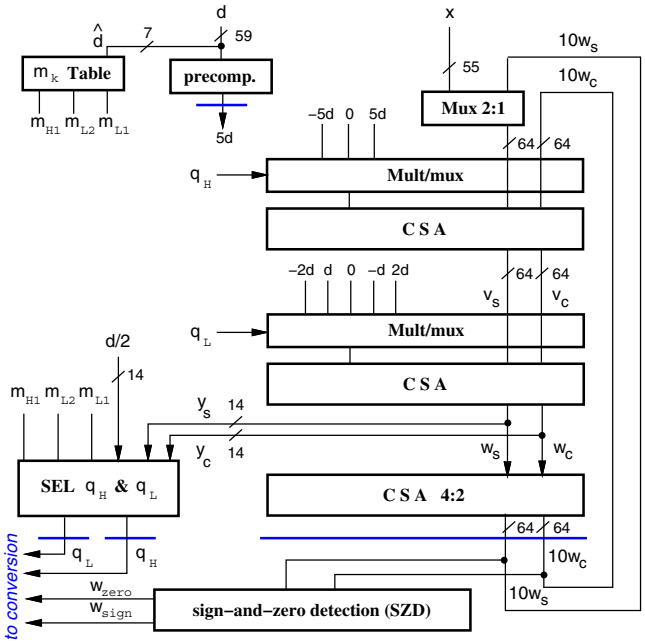

Figure 3. Recurrence.

Then the shifted/unshifted product is stored in register d and P_{59} in a 1-bit register ($db2$). This is necessary to *"remember"* that x must be shifted if d was shifted. Similarly,

$$x \leftarrow \begin{cases} P >> 1 & \text{if } db2 = 1 \\ P & \text{otherwise} \end{cases}$$

and the shifted/unshifted product is stored in register x.

4. Recurrence

Because the retiming of the recurrence (1) is advantageous to reduce the latency of the division [13], we can retime the recurrence of (2) to obtain

$$\begin{aligned} v[j] &= 10w[j-1] - q_{Hj}(5d) \\ w[j] &= v[j] - q_{Lj}d \end{aligned} \tag{6}$$

with quotient digit selection functions

$$q_{Hj} = SEL_H(\widehat{10w}, \widehat{d}) \tag{7}$$

$$q_{Lj} = SEL_L(\widehat{v}, \widehat{d}) \tag{8}$$

The recurrence, implemented as shown in Figure 3, is initialized by $w[0] = x$. The residual $w[j]$ (and $v[j]$) is kept in carry-save format to speed-up the iteration time. The negative multiples of d ($-5d$, $-2d$ and $-d$) are obtained by inverting the bits (one's complement) and by setting the carry-in to one in the carry-save adder (CSA) which follows.

A sign-and-zero detection unit (SZD) is used in each iteration to determine if $w[j] = 0$. In this case, the quotient is exact and we stop the iterations. The SZD is also needed in the rounding step to determine the sign of the remainder and if it is zero.

4.1. Selection Function

The selection function is similar to the one for the decimal divider with BCD format [9]. The corresponding block diagram is shown in Figure 4. Note that to avoid the delay of the multiplication by 10 in the critical path, we use $\widehat{w}$ instead of $\widehat{10w}$ for the selection function.

Since the quotient-digit sets are the same, we obtain the same selection intervals (except that the values are divided by 10, because of the note above). However, the selection constants are somewhat different because 1) the divisor is now in a binary representation, so that the divisor regions are different, 2) the error committed by using a redundant representation of the residual is now a power of two, and 3) to convert the selection function to operate on integers, it is necessary to multiply by a power of two instead of a power of ten. The resulting constants are shown in Table 1. The 7 MSBs of d ($\hat{d}$ in Figure 3 and Table 1) are required to select the interval. From the table we can notice that $m_{H0} = -m_{H1} - 1 = \overline{m_{H1}}$ that is m_{H1} one's complement. In this way, we just need to store one constant per $\hat{d}$ interval, and the other can be obtained by bit-inversion. Similarly, $m_{L\overline{1}} = -m_{L2} - 1 = \overline{m_{L2}}$ and $m_{L0} = -m_{L1} - 1 = \overline{m_{L1}}$.

The selection function of Figure 4 is realized by overlapping the computation of q_L to that of q_H. To speculatively compute all the possible outcomes of q_H selection we need to use a truncated $d/2$ because the selection of q_H has been done on $\widehat{w}$ and not $\widehat{10w}$:

$$\widehat{v[j]} = \frac{\widehat{10w}[j-1] - q_H(5d)_t}{10} = \widehat{w}[j-1] - q_H\left(\frac{d}{2}\right)_t$$

$\hat{d}$	q_H		q_L			
$[d_i, d_{i+1})$	m_{H1}	m_{H0}	m_{L2}	m_{L1}	m_{L0}	$m_{L\overline{1}}$
13 , 14	28	-29	18	4	-5	-19
14 , 16	30	-31				
16 , 17	34	-35	22	8	-9	-23
17 , 18	34	-35				
18 , 19	36	-37				
19 , 22	42	-43	24			-25
22 , 26	48	-49	27			-28
26 , 30	56	-57	32			-33
30 , 33	64	-65	40			-41
33 , 39	72	-73				
39 , 46	84	-85	48	16	-17	-49
46 , 54	100	-101	56			-57
54 , 64	115	-116	68			-69
64 , 77	139	-140	84			-85
77 , 90	166	-167	105	32	-33	-106
90 , 108	195	-196	113			-114
108 , 128	230	-231	128			-129

Table 1. Selection constants.

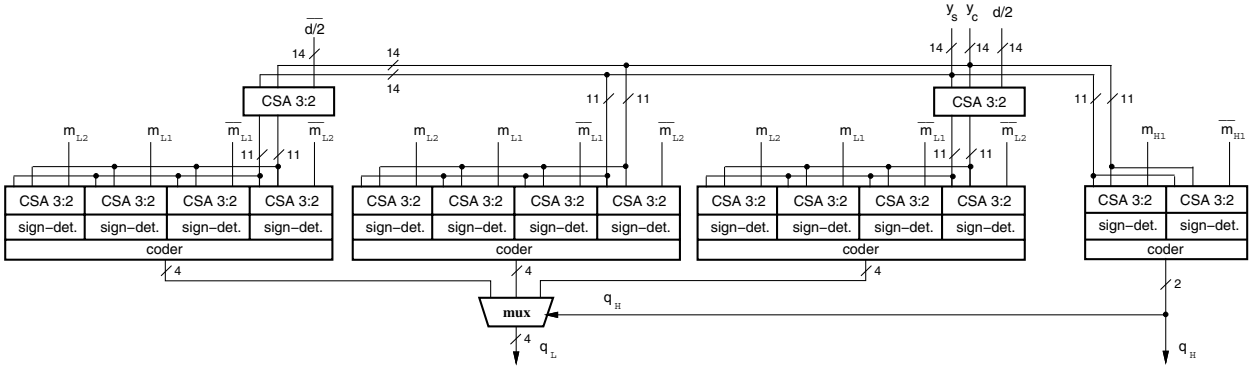

Figure 4. Detail of the selection function.

with the carry-save $\widehat{w}[j-1]$ indicated as y_s and y_c in Figure 3 and Figure 4.

5. Conversion and Rounding

The conversion of the quotient-digits $q_j = 5q_H + q_L$ from signed-digit (SD) to two's complement and their assimilation into the quotient are performed in the unit of Figure 5.

The partial quotient (Q), which is a binary integer while the division produces decimal digits, is updated, at iteration j, as

$$Q[j] \leftarrow 10Q[j-1] + B \tag{9}$$

where B is the two's complement sign-extended representation of q_j. In addition to q_j, $q_j + 1$ and $q_j - 1$ (necessary for rounding) are stored as well. Expression (9) is implemented by a CSA followed by a carry-propagate adder (CPA) adding two left-shifted (3 and 1 positions) copies of $Q[j-1]$ and B. The multiplexer in the figure, always selects q_j when not in the rounding cycle.

The converted digit q_j assimilation is delayed one cycle to avoid division by 10 when the quotient is exact. By looking at Figure 3, it is clear that $10w[j] = 0$ ($w_{zero} = 1$) is detected at the same time q_j is converted (and assimilated). By storing q_j in a register its assimilation is delayed one cycle when the outcome of the SZD (w_{zero}) is known. When the result is exact, a completion flag (DONE=1) is set. We show an example of the conversion when the quotient is exact in Table 2 for $M_x = 1$ and $M_d = 8$.

5.1. Rounding

In the implementation of the unit, we only consider the rounding mode *roundTiesToEven* that in the revised IEEE 754 standard corresponds to the old denomination *round-to-the-nearest-even*. The other rounding modes can be implemented similarly.

We refer to Figure 5 and we call q_R the last digit to be assimilated in Q, q_{R+1} the rounding digit, and B the two's

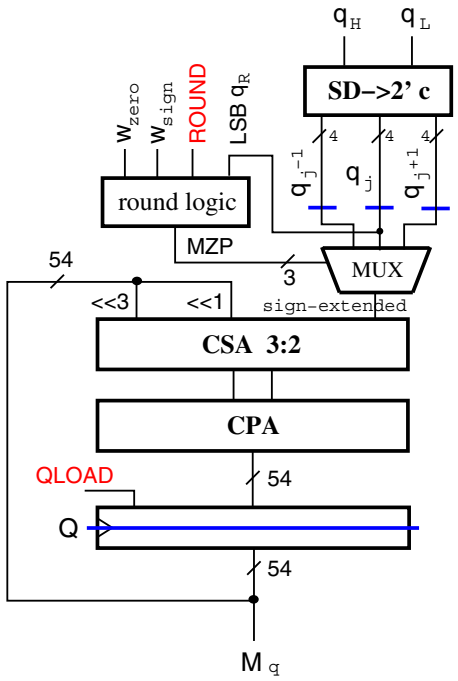

Figure 5. Convert and round unit.

complement value of q_{R+1}. The rounding amount ($ulp/2$ plus conditions on the remainder) is

$$R = 5 - w_{sign} - (w_{zero} \text{ AND } \overline{\text{LSB}(q_R)})$$

and the rounding is performed as follows:

Condition	digit to assimilate	M	Z	P
$(B+R) \geq 10$	$q_R + 1$	0	0	1
$0 \leq (B+R) < 10$	q_R	0	1	0
$(B+R) < 0$	$q_R - 1$	1	0	0

The three signals M, Z and P select the digit to be added to Q (Figure 5).

j	$Q[j]$	QLOAD	reg. q_j	w_{zero}	DONE
1	0000 0000	0	-	0	0
2	0000 0000	0	0	0	0
3	0000 0000	1	1	0	0
4	0000 0001	1	3	0	0
5	0000 000D	1	-5	1	0
6	0000 007D	1	0	1	1

$(0000\ 007D)_{16} = (125)_{10}$

Table 2. Example of conversion for exact quotient $q = \frac{1}{8} = 0.125$.

The same delaying mechanism for the assimilation of the converted digit, introduced for exact quotient, is used to avoid division by 10 after rounding.

6. Implementation and Comparisons

We implemented the BID division unit of Figure 1 using the STM 90 nm CMOS standard cells library[1] [14] and Synopsys Design Compiler.

The target was to obtain the shortest delay in the recurrence which we consider the main block of the unit. Then we adapted the delay of the other blocks, when necessary, to the recurrence delay. The delay of the critical path in the recurrence (Figure 3) is $1.0\ ns$ and it is illustrated in detail in Figure 6.

All the other blocks, except the second stage of the normalization (Figure 2) had a shorter delay than the critical path in the recurrence. In the normalization, the rectangular multiplier plus the multiplexer used to conditionally divide by 2 had a delay of $1.4\ ns$. The rectangular multiplier is implemented with radix-4 Booth recoding of the 32-bit multiplier which results in 16 partial-products (PPs). The PPs are then reduced by a 3-level tree of 4:2 CSAs. The tree is followed by a 61-bit CPA which is synthesized in a prefix-adder structure as fast as allowed by the time slack available for the target clock period.

Because the normalization unit is just used once for each operand during the whole division, we introduced a pipeline register inside the rectangular multiplier (between the reduction tree and the CPA). With this modification the whole unit can clocked at a frequency of $\frac{1}{1.0\ ns} = 1\ GHz$ at expenses of an extra cycle of latency.

The total latency of the division when the quotient is not exact (worst case) is 24 clock cycles: 4 for the normalization and 20 for the recurrence and rounding.

Table 3 reports the area of the unit and its break-down in the main composing blocks. From the table, it is clear that the multiplier has a large impact on the total area. We considered the alternative of using a decimal shifter similar

1. For comparison purposes, the FO4 inverter delay is $45\ ps$ and the area of the NAND2 gate is $4.4\ \mu m^2$ in this library.

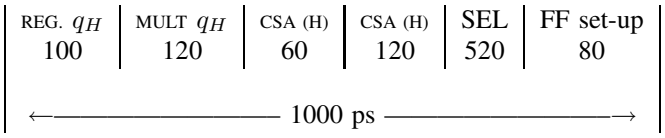

Figure 6. Delay of the critical path in the recurrence of Figure 3.

AREA $[\mu m^2]$	block
LOD	850
tables	770
rect. multiplier	65,500
registers	10,400
MUXes	1840
normalization	79,500
SEL	13,231
SZD	3,600
recurrence	35,630
convert-round	8,870
controller	550
divider	124,500

Table 3. Area break-down for main blocks.

to that of [5]. The shifter of [5] (r10SHL) cannot be directly used in our divider, because we need to shift up to 10^{17} while r10SHL can perform shifts 1 to 10^{15}. Increasing the shift amount to 17 would imply the use of an extra level of carry-save adders and multiplexers. Because we felt that the complexity of the resulting shifter would be similar to that of the rectangular multiplier we use, we performed a rough evaluation based on the results presented in [5]. The r10SHL shifter is implemented in a different technology (110 nm). Therefore, we need to apply technology scaling to have a consistent comparison. Assuming a constant field technology scaling[2] [15] the scaling factor for delays is $S = 110/90 = 1.22$ and for area $S^2 = 1.5$.

The delay of the rectangular multiplier is $1200\ ps$ which scales to approximately to $1500\ ps$ in a 110 nm technology. The fastest r10SHL reported in [5] has a delay of about $1800\ ps$ and a corresponding area of about $147,000\ \mu m^2$ which scales to $98,000\ \mu m^2$ in a 90 nm technology. Therefore, the fastest decimal shifter $1 - 10^{15}$ seems larger than the rectangular multiplier we used ($65,500\ \mu m^2$).

Summarizing, although the rectangular multiplier contributes to about 50% of the total divider area this seems the most convenient way of performing the normalization.

We compared the BID divider to the BCD divider of [9]. In [9] the operands are assumed already normalized and such that $w[0] < \rho d$. Also the quotient is normalized. Therefore, we implemented a leading-non-zero-digit detector (LNZD)

2. In the LSI Logic Gflxp 0.11 μm CMOS library used in [5] the supply voltage is $V_{DD} = 1.2\ V$, while in ours is $V_{DD} = 1.0\ V$. Because the ratio of V_{DD}s matches that of the channel lengths $\frac{110}{90} \simeq 1.2$ the constant field scaling applies.

	Latency*		AREA $[\mu m^2]$	
	BCD	BID	BCD	BID
normalization	2	4	12,500	79,500
recurrence+C&R	20	20	59,700	45,000
divider	22	24	72,200	124,500

* Both units have clock period $1.0\ ns$

Table 4. Comparison BCD vs. BID divider.

and a BCD barrel shifter to be able to compare between the two implementations. In the BCD divider, when the quotient is exact, it is still normalized because the quotient-digit conversion is performed by starting in the most-significant digit and by appending the digits at right. The conversion can be easily modified at not additional cost (in BCD) by shifting left and appending the converted digit in the least-significand position. In this way, non-normalized exact quotients can be obtained.

The synthesis of the LNZD and the BCD-shifter resulted in area of $980\ \mu m^2$ and $4,800\ \mu m^2$, respectively. By considering an architecture for the BCD significand divider similar to that of Figure 1, and in which we use one LNZD and one BCD-shifter shared by M_x and M_d, the characteristics of the BCD and BID dividers are compared in Table 4.

Although the area of the recurrence, plus conversion and rounding, is larger for the BCD divider (+33%) the normalization in BID makes the total area of the BID divider 71% larger than the BCD unit. The minimum clock period is $1.0\ ns$ for both units, but the latency is also affected by the more expensive normalization in BID.

7. Conclusions

In this work we adapted the digit-recurrence algorithm to the case of Binary Integer Decimal encoding, specified in the IEEE Standard 754-2008, and implemented a BID division unit in standard cells. We compared the BID divider with one based on the BCD encoding. The results of the comparison show that the latency of the division when the operands are normalized is the same and that the implementation of the recurrence in BID takes about 30% less area. The results also show that operand normalization is an expensive operation in BID. However, the impact of the normalization in BID can be alleviated by sharing a BID normalization unit among several other units, such as BID adders and BID multipliers. On the other hand, the binary encodings of BID make it compatible with units designed for binary (radix-2) operations, so that the combination binary/decimal might be easier.

References

[1] L. Eisen *et al.*, "IBM POWER6 accelerators: VMX and DFU," *IBM Journal of Research and Development*, vol. 51, no. 6, pp. 663–684, 2007.

[2] C. H. Webb, "IBM z10: The next-generation mainframe microprocessor," *IEEE Micro*, vol. 28, pp. 19–29, Mar./.Apr. 2008.

[3] *IEEE Standard for Floating-Point Arithmetic*, IEEE Computer Society Std. 754, 2008.

[4] M. F. Cowlishaw, "Densely packed decimal encodings," *IEE Proceedings - Computers and Digital Techniques*, vol. 149, no. 3, pp. 102–104, May 2002.

[5] J. Hormigo, S. Gonzalez-Navarro, and M. Schulte, "Decimal Left Shifters for Binary Numbers," *Proc. of 8th Conference on Real Numbers and Computers (RNC 8)*, July 2008.

[6] M. Ercegovac and T. Lang, *Division and Square Root: Digit-Recurrence Algorithms and Implementations*. Kluwer Academic Publisher, 1994.

[7] L.-K. Wang and M. Schulte, "Decimal floating-point division using Newton-Raphson iteration," in *Proc. of 15th International Conference on Application-Specific Systems, Architectures and Processors*, Sept. 2004, pp. 84–95.

[8] H. Nikmehr, B. Phillips, and C.-C. Lim, "Fast decimal floating-point division," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 14, no. 9, pp. 951–961, Sept. 2006.

[9] T. Lang and A. Nannarelli, "A Radix-10 Digit-Recurrence Division Unit: Algorithm and Architecture," *IEEE Transactions on Computers*, vol. 56, no. 6, pp. 727–739, June 2007.

[10] A. Vazquez, E. Antelo, and P. Montuschi, "A radix-10 SRT divider based on alternative BCD codings," *Proc. of 25th International Conference on Computer Design (ICCD)*, pp. 280–287, Oct. 2007.

[11] C. Tsen, S. Gonzalez-Navarro, and M. Schulte, "Hardware Design of a Binary Integer Decimal-based Floating-point Adder," *Proc. of 25th International Conference on Computer Design (ICCD)*, pp. 288–295, Oct. 2007.

[12] S. Gonzalez-Navarro, C. Tsen, and M. Schulte, "A Binary Integer Decimal-based Multiplier for Decimal Floating-Point Arithmetic," *Proc. of 41st Asilomar Conference on Signals, Systems, and Computers*, pp. 353–357, Nov. 2007.

[13] E. Antelo, T. Lang, P. Montuschi, and A. Nannarelli, "Digit-recurrence dividers with reduced logical depth," *IEEE Transactions on Computers*, vol. 54, pp. 837–851, July 2005.

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

[15] N. H. E. Weste and K. Eshraghian, *Principles of CMOS VLSI Design*, 2nd ed. Addison-Wesley Publishing Company, 1993.