Hardware Design of a Binary Integer Decimal-based IEEE P754 Rounding Unit

Charles Tsen
University of Wisconsin
stsen@wisc.edu

Michael Schulte
University of Wisconsin
schulte@engr.wisc.edu

Sonia González-Navarro
Universidad de Málaga
sonia@ac.uma.es

Abstract

Because of the growing importance of decimal floating-point (DFP) arithmetic, specifications for it were recently added to the draft revision of the IEEE 754 Standard (IEEE P754). In this paper, we present a hardware design for a rounding unit for 64-bit DFP numbers (decimal64) that use the IEEE P754 binary encoding of DFP numbers, which is widely known as the Binary Integer Decimal (BID) encoding. We summarize the technique used for rounding, present the theory and design of the BID rounding unit, and evaluate its critical path delay, latency, and area for combinational and pipelined designs. Over 86% of the rounding unit's area is due to a 55-bit by 54-bit binary multiplier, which can be shared with a double-precision binary floating-point multiplier. To our knowledge, this is the first hardware design for rounding IEEE P754 BID-encoded DFP numbers.

1. Introduction

Decimal floating-point (DFP) arithmetic is important in many applications because of its ability to represent decimal fractions exactly and to mimic manual calculations that perform decimal rounding. Because binary floating-point arithmetic can neither provide correct decimal rounding nor exactly represent many decimal fractions, such as 0.01, 0.10, 0.0475, and 10^{-35} [5], numerous applications require DFP arithmetic. Such applications include currency conversion, billing, insurance, tax calculations, and banking. One study estimates that errors from binary floating-point arithmetic can accumulate to a yearly billing error of over $5 million for large billing systems [10].

Applications that cannot tolerate errors due to binary floating-point arithmetic often use software to perform DFP arithmetic [5]. Software packages for DFP arithmetic include IBM's decNumber library [6] and the Java BigDecimal library [14]. Also, Intel recently published results for a prototype software library using the Binary Integer Decimal (BID) encoding [3,4]. These software packages are adequate for many of today's applications, but as trends towards globalization and e-commerce continue, the performance of software for DFP arithmetic may not suffice.

Several hardware designs for DFP arithmetic have been developed using encodings other than BID [1, 2, 12, 17, 18, 19]. Recently, IBM announced DFP hardware on its Power6 server processor [13] and System z9 processor [8] using the Densely Packed Decimal (DPD) encoding, discussed in Section 2. Previous designs for DFP arithmetic differ from our design in that they operate on significands with a decimal radix of 10, such as Binary Coded Decimal (BCD) or DPD, as opposed to BID's binary radix of 2. We believe ours is the first hardware design published for BID-encoded floating-point numbers. This may be due to a perception that the BID format is more appropriate for implementing in software than in hardware. Contrarily, we argue that BID is well suited for hardware implementations, since it can share hardware with binary arithmetic units.

A well-designed rounding unit is an important component of a BID-based hardware DFP solution, since decimal rounding is frequently performed in commercial and financial applications. One such example is accumulating daily interest on credit card accounts, where the daily periodic interest rate is precise to five digits after the decimal point. The accumulated balance must be rounded to the minimum currency unit of one cent. Similar calculations are found with interest, tax, currency conversion, and billing applications, in which such rounding operations are very common [20]. In some applications, rounding is required after every intermediate calculation. For such workloads, rounding is very important. Furthermore, any addition, subtraction, multiplication, division, or square root operation requires a correctly rounded result. Thus, a rounding unit has a significant effect on decimal floating-point performance.

Innovative techniques are needed to round BID results to a specified number of decimal digits. This allows hardware to leverage the BID significand's compact encoding, which lends itself well to existing high-speed binary arithmetic circuits.

In this paper, we present a BID-based hardware design that correctly rounds BID-encoded decimal64 numbers for all IEEE P754 rounding modes. The design can be adapted to provide rounding for BID *addition*, *subtraction*, *multiplication*, *roundToIntegral*, and *quantize*. We focus on the theory and design of the rounder rather than the specifics of how to adapt it for each operation. Section 2 discusses the DFP formats in IEEE P754 and the challenge of rounding BID-encoded numbers. Section 3 presents the technique and theory for BID rounding. Section 4 discusses the design details and testing methodology of the BID rounding unit. Section 5 analyzes the critical path delay, latency, and area of combinational and pipelined BID rounding units. Section 6 presents our conclusions.

2. Decimal Floating-Point Formats and Rounding

Due to the importance of DFP arithmetic, the IEEE P754 Standard for Floating-Point Arithmetic includes specifications for DFP formats and operations [11]. IEEE P754 was recently voted on and initially approved in sponsor ballot during December 2006. It specifies five rounding modes: `roundTiesToEven (RTE)`, `roundTiesToAway (RTA)`, `roundTowardZero (RTZ)`, `roundTowardNegative (RTN)`, and `roundTowardPositive (RTP)`.

In IEEE P754, the value of a finite DFP number is:

$$(-1)^S \times 10^{E-bias} \times C$$

where S is the sign bit, E is a biased exponent, *bias* is a constant value that makes E non-negative, and C is the significand. IEEE P754 specifies two methods for encoding the significands of DFP numbers; Binary Integer Decimal (BID) [15] and Densely Packed Decimal (DPD) [7]. With BID, each significand can be viewed as an unsigned binary integer. With DPD, each significand can be viewed as an unsigned decimal integer, in which groups of 10 bits represent three decimal digits [7]. In IEEE P754, the BID encoding is called the binary encoding and the DPD encoding is called the decimal encoding, but either encoding can be used to represent DFP significands. For example, 5.43 is represented as 543×10^{-2}, where the significand, 543, can be encoded using either BID or DPD.

For 64-bit DFP numbers, of decimal64 type, the supported precision is 16 decimal digits. The BID significand is 54-bits because the maximum significand supported by decimal64 is 10^{16}-1, which is less than 2^{54}. BID lends itself to high-performance binary circuits, since the significand is a binary number. However, a challenge to implementing the BID encoding is performing efficient rounding, which we address in this paper.

To illustrate the challenge of BID rounding, we give an example of the IEEE P754 *roundToIntegral* operation. This operation's input is a floating-point number, and its output is the input rounded to one of its two closest integers, depending on the rounding mode. Consider an input of 1234×10^{-2} to the *roundToIntegral* operation. The resulting value is either 13×10^{0} (in the `RTP` mode) or 12 x 10^{0} (in all other rounding modes). The BID significand of the input is $1234_{10} = 10011010010_2$, where the subscript indicates the base of the number and leading zeros are not shown. After it is rounded, the BID significand is either $13_{10} = 1101_2$ or $12_{10} = 1100_2$. The challenge is to achieve correct and efficient rounding of a specified number of decimal digits from a binary number.

BID rounding can be abstracted to a division by 10^d to truncate d digits, combined with a decision to increment based on the value of the digits that are truncated. We refer to the intermediate truncated significand and its rounding information as the pre-rounded result. Each rounding mode examines different attributes of the truncated digits to determine if an increment is needed or not. A straight-forward method to round off the d least significant digits of a BID number is to use a binary integer division circuit to divide the number by 10^d and to use a remainder operation to determine the rounding information. However, these two operations are traditionally costly in terms of area and delay. Instead, our rounding unit uses reciprocal multiplication to avoid division by 10^d and an innovative approach to ensure that results are correctly rounded in spite of errors due to reciprocal multiplication.

3. Rounding Technique

The technique we use to effectively divide by $m = 10^d$ and keep useful rounding information extends an already known technique, reciprocal multiplication [9, 21], with a novel theorem (Theorem 1), stated on the following page. The reciprocal multiplication technique can be viewed as multiplication by a precalculated approximation of $w_d \approx 10^{-d}$ to effectively achieve division by $m = 10^d$.

Reciprocal multiplication is well suited for division in cases that the divisors are known and few, which is true with the BID format for DFP numbers. We need no more than p divisors in a format with a precision of p decimal digits. The IEEE P754 decimal64 format has a precision of p = 16 decimal digits, so when multiplying by 10^{-d}, the values of d span the range of integers from 1 to 16.

To perform correct rounding, it is necessary to determine if the pre-rounded result lies exactly halfway between two consecutive floating-point numbers (i.e. it is a midpoint) or if it can be exactly represented as a floating-point number with the desired exponent (i.e. it is exact). As an example of why midpoint determination is important, consider an input of 7654500 × 10^{-3} to *roundToIntegral.* In this case, C_i is 7654500, d is 3, and the pre-rounded result is 7654.500. The correctly rounded result is 7654 × 10^0 for some rounding modes (RTE, RTZ, RTN*),* but 7655 × 10^0 for others (RTA, RTP). The importance of midpoint determination is demonstrated by the fact that a variation in the input significand by ±1 unit in the last place (ulp) changes rounding results for the RTE and RTA modes. Thus, a functionally correct rounder must be able to precisely detect midpoints. A similar example for determining exactness can be followed using 7654000 × 10^{-3} as the input to *roundToIntegral.*

To achieve correct rounding when using reciprocal multiplication, we apply Theorem 1, where C_i is a u-bit input significand, $m = 10^d$ is a v-bit unsigned integer, w_d is a (u+1)-bit unsigned integer that approximates 10^{-d}, left-shifted by u+v bits, and P is the (2u+1)-bit product of $C_i \times w_d$. P is partitioned into three fields, Q, R, and D, where Q gives the truncated quotient (i.e, floor($C_i/10^d$)), R provides rounding information, and D is discarded. Theorem 1 and its proof are part of a body of work developed by Intel engineers including Peter Tang, Marius Cornea, and John Harrison [16].

Theorem 1: *Let C_i and $m = 10^d$ be positive integers such that $0 < C_i < 2^u$ and $0 < m < 2^v$. Let q and r be the integer quotient and remainder of C_i divided by m. Thus, $C_i = q \cdot m + r$, $0 \le r \le m-1$. Define $w_d = ceil(2^{u+v}/m)$ and let $P = C_i \times w_d$ be expressed in the form*

$$P = 2^{u+v} Q + 2^u R + D$$

where Q, R, and D are non-negative integers, $R < 2^v$, and $D < 2^u$. Then Q = q. Furthermore $r = 0$ iff $R = 0$; $r \le (m/2)-1$ iff $R \le 2^{v-1}-1$; $r = m/2$ iff $R = 2^{v-1}$; and $r \ge (m/2)+1$ iff $R \ge 2^{v-1}+1$.

Based on Theorem 1, Q provides the truncated quotient, q = floor($C_i/10^d$), and R provides all information needed to determine the correctly rounded result.

We elaborate on how this theorem helps to detect midpoints and exactness by focusing on the calculation of the two parameters u and v. The number of bits in the input significand, C_i, is referred to as u. For decimal64 , u is fixed at 54 bits, since the maximum significand supported, 10^{16}-1, fits within 54 bits. The number of bits needed to represent 10^d, is referred to as v, where

$$v = \mathrm{ceil}(\log_2(10^d)) = \mathrm{ceil}(d \cdot \log_2(10))$$

For example, 10^6 is represented in 20 bits, so v is 20 when rounding off d = 6 decimal digits. It is important to note that v varies with d, whereas u is fixed for a given significand precision, p.

The (2u+1)-bit product, $P = C_i \times w_d$ is shown in Figure 1, where P is partitioned into three fields: Q, R, and D. D consists of the u least significant product bits, which contain no useful information and are discarded. The Q and R fields occupy the remaining (u+1)-bits and vary in width depending on v. The most significant (u+1-v)-bits, Q, correspond to the truncated quotient. The next lowest v bits, R, are inspected to determine if the truncated decimal digits, when viewed as a fraction, represent exactly zero, exactly one half, between zero and one half, or above one half.

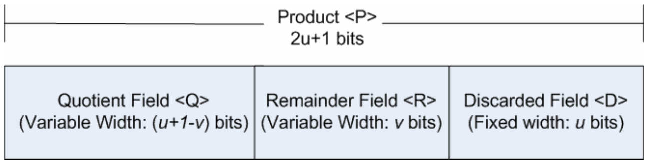

Figure 1. Product Fields

To determine whether the pre-rounded result is exact, less than a midpoint, a midpoint, or greater than a midpoint, we use the properties described in Table 1. The round bit, r^*, is the most significant bit of the R field, and the sticky bit, s^*, is set if any of the remaining bits of the R field are 1. As shown in Table 1, these two bits keep enough information to know if the pre-rounded result is exact (r^*=0, s^*=0), less than a midpoint (r^*=0, s^*=1), a midpoint (r^*=1, s^*=0), or greater than a midpoint (r^*=1, s^*= 1).

Table 1: Midpoints and Exact Results

Conditions	Values of R	r*	s*
Pre-rounded result is exact	$R == 0$	0	0
Pre-rounded result less than midpoint	$R < 2^{v-1}$	0	1
Pre-rounded result exactly midpoint	$R == 2^{v-1}$	1	0
Pre-rounded result greater than midpoint	$R > 2^{v-1}$	1	1

4. Rounding Unit Design and Testing

In our rounding unit for BID-encoded IEEE P754 decimal64 numbers, we pre-calculate sixteen values of w_d and store them in a lookup table. The table entry used to truncate d digits is referred to as w_d and is equal to ceil(2^{u+v} / 10^d). The pre-calculated values of w_d used in this design are shown in Table 2. Since $u=54$, each table entry is u+1 = 55 bits (Th.1). Though 55 bits are needed for the entries, each value's most significant bit is always 1. Thus, only 54 bits are stored per entry, and the most significant bit of w_d is hardwired to 1. A value of w_d is selected from the table, based on the number of digits to round off. The 54-bit input significand, C_i, is then multiplied by w_d to produce the result P, a 109-bit intermediate value.

Table 2: Pre-calculated values of w_d

d	LUT Position	w_d
1	0001	55'h66666666666667
2	0010	55'h51eb851eb851ec
3	0011	55'h4189374bc6a7f0
4	0100	55'h68db8bac710cb3
5	0101	55'h53e2d6238da3c3
6	0110	55'h431bde82d7b635
7	0111	55'h6b5fca6af2bd22
8	1000	55'h55e63b88c230e8
9	1001	55'h44b82fa09b5a53
10	1010	55'h6df37f675ef6eb
11	1011	55'h57f5ff85e59256
12	1100	55'h465e6604b7a845
13	1101	55'h709709a125da08
14	1110	55'h5a126e1a84ae6d
15	1111	55'h480ebe7b9d5857
16	0000	55'h734aca5f6226f1

The top-level design of our rounding unit is presented in Figure 2. The design takes as inputs, the 54-bit input significand, C_i, an 11-bit value, d, that indicates the number of digits to be rounded off, the sign of the input operand, S_i, and the 3-bit Rounding Mode. The d input was chosen to be 11 bits to handle the maximum difference between two exponents, which is used when implementing the IEEE P754 *quantize* operation. The output is a 54-bit rounded significand, C_o. The majority of the diagram, through the center and left, illustrates the data path. The right side of the diagram shows blocks for basic control. The rounder handles the rounding of d > 16 digits by using a bypass path, controlled by the bypass logic, skipping the multiplier altogether. Division by powers of 10^d with d > 16 results in a value less than one tenth, which rounds to either 0 or 1, depending on the rounding mode. In this case, the bypass logic chooses one of these two values to route to the output. The bypass logic also handles the case where the number of digits rounded off, d, equals 0. In this case, $C_i[53:0]$ is routed along the bypass path to the output. The unit assumes other IEEE P754 DFP computations such as sign and exponent calculation, detection of Not-a-Number (NaN), infinities, overflow, and inexact occur outside this block. The 55-bit by 54-bit multiplier dominates the design area, whereas the logic for multiplexer control is compact. More details on the rounding unit's design are given throughout the rest of the section.

The extract significant, round and sticky unit following the multiplier, which is also shown in Figure 3, is used to extract Q, r^*, and s^*. Q is prepended with $(v-1)$ leading zeros, to produce a 54-bit temporary significand, C_{tmp}. The Extract Significand and Round module can be considered a "specialized shifter" which shifts $P[108:54]$ right by $v-1$ bits, so r^* is in its least significant bit position. The extract sticky unit uses a mask obtained from a lookup table on $d[3:0]$ to keep only relevant bits of $P[105:54]$. It then OR's all these bits together to compute s^*. We know that $P[108:106]$ can never contain sticky information, since *(1)* $P[108]$ is always clear due to C_i being restricted to values less than 10^{16}, *(2)* $P[107]$ always contains information about Q since Q is non-zero (i.e. it must be at least one bit), and (3) r^* is the most significant bit of the R field, which prevents P[106] from containing sticky information. Note that due to *(1)* the multiplier may be modified to omit the computation of $P[108]$.

The value of C_{tmp} extracted from the Q field of the product is the truncated result. In many cases, this is the value for C_o, but in other cases it must be incremented. For this reason, an increment unit is included in the design near the output.

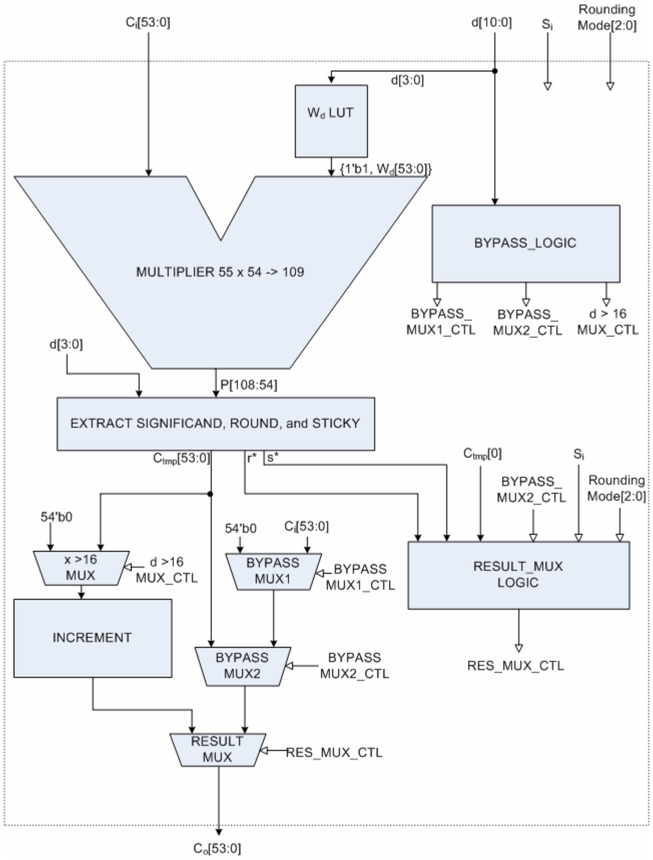

Figure 2. Top Level Design of BID Rounding Unit

Table 3 summarizes the control logic used to determine if an increment is needed, based on S_i, r^*, s^*, and $C_{tmp}[0]$. The simplest rounding mode is RTZ, where C_{tmp} can always be taken as the final significand. The r^* bit is set if the pre-rounded result is a midpoint or above, which is sufficient knowledge for an increment in RTA mode. An additional check of s^* and $C_{tmp}[0]$ is needed in the RTE mode, because an exact midpoint with an even truncated significand is not incremented. The sign must be checked to determine if an increment is needed in the RTP and RTN modes. If the sign matches the direction of the rounding mode and if the remainder is non-zero, C_{tmp} is incremented. Any non-zero remainder has either $r*$ or $s*$ set.

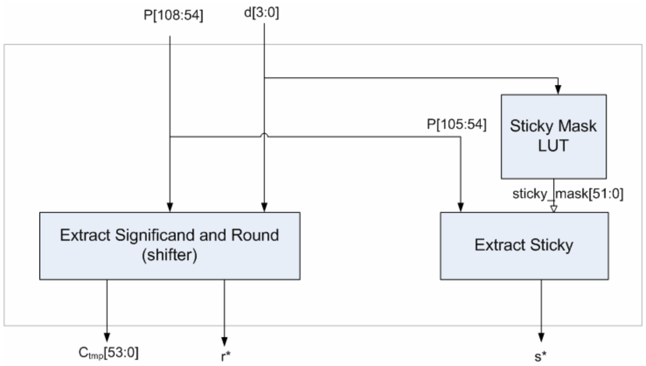

Figure 3. Significand, Round, and Sticky Extraction Unit

Revisiting an earlier example, in the context of this design, consider the input 7654500×10^{-3}, to the *roundToIntegral* function in the `RTE` mode. From Table 2, the precalculated w_d for $d = 3$ is $4189374bc6a7f0_{16}$, which is multiplied by the input significand, $C_i = 7654500_{10}$, giving a product, $P = 000000001de680000000002cd9c0_{16}$. Since $v = 10$ and $u = 54$, the quotient is $Q = P[108:64] = 000000001de6_{16} = 7654_{10}$, $C_{tmp}[53:0] = 00000000001de6_{16}$, and $R = P[63:54] = 200_{16}$, which gives $r* = 1$ and $s* = 0$. This means that the pre-rounded result is an exact midpoint, so for `roundTiesToEven`, C_{tmp} is incremented if $C_{tmp}[0] = 1$. In this example, it is not, so the control chooses the multiplexer path that does not include the increment unit and the expected result of 7654×10^0 is produced.

Table 3: Logic Equations for Increment Control

Rounding Mode	Increment Condition
`roundTiesToAway`	$r*$
`roundTiesToEven`	$r* \ \& \ (s* \mid C_{tmp}[0])$
`roundTowardZero`	0
`roundTowardPositive`	$\sim S_i \ \& \ (r* \mid s*)$
`roundTowardNegative`	$S_i \ \& (r* \mid s*)$

We developed a directed random test generator to enhance our test coverage and increase our confidence in the design methodology. The test generator is written in Perl, and it produces a significand and the number of digits to round off. Then, it computes the expected result, against which it compares the output of the rounder. The directed random test generator produces a user-controllable mix of tests that fall into five categories: *vanilla, midpoint, exact, near-midpoint*, and *near-exact*. The *vanilla* tests can give any significand between 1 and 9999999999999999 with a uniform distribution. The *midpoint* and *exact* categories exercise the boundary conditions by ensuring the digits cut off are 5 followed by all zeros or all zeros, respectively. The *near-midpoint* and *near-exact* categories are similar to the midpoint and exact cases, with the input significand incremented or decremented before rounding. The user can control the percentage of tests in each category and generate millions of tests very quickly. This directed random test generator aided in our verification of the design, allowing us to quickly test many iterations of the design and to ensure correct functionality.

5. Results

We have initially synthesized, tested, and evaluated our BID rounding unit using Mentor Graphics ModelSim, Synopsys Design Compiler, and the LSI Logic Gflxp 0.11 micron CMOS standard cell library. Our testing consisted of 1,000,000 directed random test vectors in addition to 500 corner-case test vectors. We have evaluated several pipelined implementations of our BID rounding unit, all of which are optimized for delay. A purely combinational design has been synthesized as well as pipelined designs with four to six stages. We examine the tradeoffs between area, critical path delay, and latency across the designs. The results are summarized in Table 4, where the total area includes logic, wire routing, and registers. Each design has registered inputs and outputs.

Table 4: BID Rounding Unit Synthesis Results as a Function of Pipeline Depth

Pipeline Depth	Total Area (μm^2)	Delay (ns)	Latency (ns)
Unpipelined	343612	2.90	2.90
4	612908	0.87	3.48
5	662503	0.81	4.05
6	726430	0.69	4.14

In this technology, an inverter with fan-out of 4 (FO4) has a delay of 0.055 ns, and a 2-input NAND gate has an area of 8.08 μm^2. Thus, the clock cycle for the decimal64 BID rounder design ranges from 13 to 53 FO4 delays. Furthermore, the total area ranges from 0.344 to 0.726 mm^2. The 4-stage pipelined rounder consists of a 2-stage multiplier with 1 stage before and 1 stage after. The 5-stage pipelined rounder uses a 3-stage multiplier and is otherwise the same. The 6-stage rounder is similar to the 5-stage rounder, except it adds an additional pipeline stage after the multiplier.

To put the rounding unit results in perspective, we synthesized an unpipelined Synopsys DesignWare 54-bit by 55-bit multiplier, named DW02_mult. We constrained the synthesis to have the same critical path delay as the multiply portion of the rounder design (1.57ns), and we used the same technology. The total area of the DesignWare multiplier was 329735 μm^2. Thus, by this measure, the multiplier comprises 96% of the total area of the unpipelined rounder design. Though this comparison serves as a good first-order approximation, we believe that the synthesizer had more liberty to optimize the delay at the cost of area in the standalone multiplier design. Thus, it is possible that the area of the standalone multiplier is slightly inflated.

To corroborate the above area measurement, we conducted a separate analysis by invoking the Synopsys `profile_area` command. The tool reported the combinational area of the 55-bit by 54-bit multiplier as ranging from 86% to 88% of the combinational area of the total rounder design, depending on the number of pipeline stages.

These results are promising, as they indicate that BID rounding can be achieved with only a small increase in area when hardware is shared with an existing double-precision floating-point multiplier. Since we are using a synthesized multiplier, we are further encouraged that efforts to improve the multiplier have the potential to also enhance the rounding unit.

In the unpipelined rounding unit design, the critical path delay from inputs to outputs is 2.90ns. The critical path is through the w_d lookup table, 55-bit by 54-bit Multiplier, Extract Significand Unit, $d > 16$ Multiplexer, Increment Unit, and finally the Result Multiplexer. A large portion of the delay is due to the 55-bit by 54-bit Multiplier, which has a delay of 1.57ns, or 54.1 percent of the total delay. The delay of the other units on the critical delay path is as follows: the w_d lookup table delay is 0.47ns, the Extract Significand Unit delay is 0.47ns, the d > 16 Multiplexer delay is 0.08ns, the Increment Unit delay is 0.23ns, and the Result Multiplexer delay is 0.08ns. These results show that a BID hardware rounder can be implemented with a reasonable delay, and its delay is largely due to the delay of the multiplier. The remaining circuitry has relatively low delay.

6. Conclusion

We have presented the first design of a BID-based DFP rounding unit. This design correctly rounds the significand of an IEEE P754 BID decimal64 number. The techniques and theory of the design can be adapted to support the IEEE P754 operations of *addition, subtraction, multiplication, roundToIntegral,* and *quantize,* and can be adapted for other operand sizes.

The hardware design is promising in terms of area, latency, and potential hardware reuse. The design can be pipelined to achieve a cycle time equal to 13 FO4 inverter delays. Over 86% of the rounding unit's area is due to a 55-bit by 54-bit binary multiplier, which can be shared with a double-precision binary floating-point multiplier. The design illustrates that BID rounding can be effectively achieved in hardware.

7. Acknowledgements

The research presented in this paper is supported in part by a grant from Intel Corporation. The authors are indebted to Peter Tang, Marius Cornea, John Crawford, and John Harrison for theoretical work supporting the rounder design.

8. References

[1] G. Bohlender, T. Teufel, "A Decimal Floating-Point Processor for Optimal Arithmetic," *Computer Arithmetic : Scientific Computation and Programming Languages, ISBN 3-519-02448-9*, B. G. Teubner Stuttgart, 1987, pp. 31-58.

[2] M. S.Cohen, T. E. Hull, V. C. Hamacher, "CADAC: A Controlled-Precision Decimal Arithmetic Unit," *IEEE Transactions on Computers, vol. C-32, no. 4*, April 1983, pp. 370-377.

[3] M. Cornea, C. Anderson, J. Harrison, P. Tang, E. Schneider, C. Tsen, "A Software Implementation of the IEEE 754R Decimal Floating-Point Arithmetic Using the Binary Encoding Format," *to be published in the 18th International Symposium on Computer Arithmetic*, Montpelier, France, June 2007.

[4] M. Cornea, C. Anderson, C. Tsen, "Software Implementation of the IEEE 754R Decimal Floating-Point Arithmetic," *Proceedings of the International Conference on Software and Data Technologies*, Setúbal, Portugal, September 2006.

[5] M. F. Cowlishaw, "Decimal Floating-Point : Algorism for Computers," *Proceedings of the 16th IEEE Symposium on Computer Arithmetic*, June 2003, Santiago de Compostela, Spain, pp. 104-111.

[6] M. F. Cowlishaw, "The decNumber Library," *Available at http://www2.hursley.ibm.com/decimal/decnumber/pdf 2006.*

[7] M. F. Cowlishaw, "Densely Packed Decimal Encoding," *IEE Proceedings – Computers and Digital Techniques, vol. 149*, May 2002, pp. 102-104.

[8] A. Y. Duale, M. H. Decker, H-G. Zipperer, M. Aharoni, T. J. Bohizic, "Decimal floating-point in z9: An implementation and testing perspective," *IBM Journal of Research and Development. vol. 51, no. 1/2*, March 2007.

[9] T. Granlund, P. Montgomery, "Division by Invariant Integers using Multiplication," *Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation*, Orlando, Florida 1994, pp. 61-72.

[10] IBM Corporation, "The 'telco' benchmark," *Available at http://www2.hursley.ibm.com/decimal/ telco.html*, 2002.

[11] Institute of Electrical and Electronic Engineers, "Draft Standard for Floating-Point Arithmetic," *http://754r.ucbtest.org/drafts/754r.pdf,* October, 2006.

[12] H. Nikmehr, B. Phillips, C.-C. Lim, "Fast Decimal Floating-Point Division," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 14, no 9*, September 2006. pp. 951-961.

[13] S. Shankland, "IBM's Power6 Gets Help with Math Multimedia," *Available at http://news.zdnet.com/2100-*

9584_22-6124451.html. Published on ZDNet News, October 10, 2006.

[14] Sun Microsystems, "BigDecimal (Java 2 Platforms SE v1.4.0)," *URL: http://java.sun/com/products,* Sun Microsystems Inc., 2002.

[15] P. Tang, "Binary-Integer Decimal Encoding for Decimal Floating-Point," *Intel Corporation, Available at http://754r.ucbtest.org/issues/decimal/bid_rationale.pdf.*

[16] P. Tang, M. Cornea, J. Harrison, "BID Building Blocks," *Intel Internal Technical Report.* Available Upon Request.

[17] J. Thompson, N. Karra, and M. J. Schulte, "A 64-bit Decimal Floating-Point Adder," *IEEE Computer Society Annual Symposium on VLSI*, Lafayette, Louisiana, February 2004, pp. 297-298.

[18] L.-K. Wang, M. J. Schulte, "Decimal Floating-Point Division Using Newton-Raphson Iteration," *Proceedings of the IEEE International Conference on Application-Specific Systems, Architectures and Processors,* September 2004, pp. 84-95.

[19] L.-K. Wang, M. J. Schulte, "Decimal Floating-Point Square Root Using Newton-Raphson Iteration," *Proceedings of the IEEE International Conference on Application-Specific Systems, Architectures, and Processors*, Samos, Greece, July 2005, pp. 309-315.

[20] L.-K. Wang, C. Tsen, D. Jhalani, M. J. Schulte, "Performance Analysis and Benchmarking for Decimal Floating-Point Applications," *submitted to the International Conference on Computer Design*, Lake Tahoe, California, October 2007.

[21] Warren, Henry S Jr. *Hacker's Delight*, Addison Wesley, Reading Massachusetts, 2003.