Hardware Design of a Binary Integer Decimal-based Floating-point Adder

Charles Tsen
University of Wisconsin
stsen@wisc.edu

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

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

Abstract

Because of the growing importance of decimal floating-point (DFP) arithmetic, specifications for it are included in the IEEE Draft Standard for Floating-point Arithmetic (IEEE P754). In this paper, we present a novel algorithm and hardware design for a DFP adder. The adder performs addition and subtraction on 64-bit operands that use the IEEE P754 binary encoding of DFP numbers, widely known as the Binary Integer Decimal (BID) encoding. The BID adder uses a novel hardware component for decimal digit counting and an enhanced version of a previously published BID rounding unit. By adding more sophisticated control, operations are performed with variable latency to optimize for common cases. We show that a BID-based DFP adder design can be achieved with a modest area increase compared to a single 2-stage pipelined 64-bit fixed-point multiplier. Over 70% of the BID adder's area is due the 64-bit fixed-point multiplier, which can be shared with a binary floating-point multiplier and hardware for other DFP operations. To our knowledge, this is the first hardware design for adding and subtracting 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 (BFP) arithmetic neither provides correct decimal rounding nor exactly represents many decimal fractions, such as 0.01, 0.0475, and 10^{-35} [1], numerous applications require DFP arithmetic. Such applications include currency conversion, insurance, tax calculations, billing, and banking. One study estimates that BFP arithmetic errors can accumulate to an annual error of over \$5 million for large billing systems [2].

Applications that cannot tolerate errors due to BFP arithmetic often use software to perform DFP arithmetic [1]. Software packages for DFP arithmetic include IBM's decNumber library [3] and the Java BigDecimal library [4]. Intel recently published results for a Binary Integer Decimal (BID) software library [5, 6]. These software packages are adequate for many applications, but as globalization and e-commerce grow, software performance for DFP arithmetic may not suffice.

Several hardware designs for DFP arithmetic have been developed using encodings other than BID [7, 8, 9, 10, 11]. Recently, IBM announced DFP hardware on its Power6 server processor [12] and System z9 processor [13] 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.

Addition and subtraction operations occur frequently in DFP applications, so the design of a DFP adder with correct rounding is very important. In this paper, we present a hardware design that correctly performs addition and subtraction on 64-bit BID-encoded DFP numbers for all IEEE P754 rounding modes. We do not address exceptions or special values such as NaN and infinity. Our BID adder differs from previous DFP adder designs, which operate on DPD-encoded numbers [14, 15]. It uses a novel hardware component for decimal digit counting and leverages an enhanced version of a previously published BID rounding unit [16]. Furthermore, operations are performed with variable latency to optimize common cases.

We believe ours is the first hardware design for adding and subtracting BID-encoded floating-point numbers. This may be due to a perception that the BID format is more appropriate for software rather than hardware. Contrarily, we argue that BID is well suited for hardware implementations, since it can share hardware with binary arithmetic units. For example, a 64-bit fixed-point multiplier occupies a large percentage of the area of our BID adder, can also be used to perform BFP multiplication, and BID multiplication, comparison, minimum, maximum, quantize, and toIntergalValue.

The remainder of this paper is organized as follows. Section 2 discusses DFP numbers in IEEE P754.

Section 3 discusses the challenge of adding BID-encoded numbers, and presents the technique and theory for BID addition and subtraction. Section 4 combines the concepts from Section 3 to show a compact DFP adder design. Section 5 provides preliminary synthesis results. Section 6 presents our conclusions.

2. Decimal Numbers in IEEE P754

Due to the importance of DFP arithmetic, the IEEE P754 Draft Standard for Floating-Point Arithmetic includes specifications for DFP formats and operations [17]. 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) [18] and Densely Packed Decimal (DPD) [19]. 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 [19]. 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 use either the BID or DPD encoding.

The significand of a DFP number is not normalized, meaning that a single DFP number may have multiple representations. For example, 3×10^{-1}, 30×10^{-2}, and 300×10^{-3} all have the same numeric value, but they have different IEEE P754 representations. Because of this characteristic, IEEE P754 defines the Preferred Representation Exponent, which specifies a required exponent, and implicitly the significand, after each decimal operation. For example, with decimal addition and subtraction, the exponent of the result equals the smaller exponent of the two input operands if the result is exact. If it is not exact, the exponent is selected to maximize the number of significant digits in the rounded result. IEEE P754 specifies five rounding modes for DFP arithmetic: `roundTiesToEven (RTE)`, `roundTiesToAway (RTA)`, `roundTowardZero (RTZ)`, `roundTowardNegative (RTN)`, and `roundTowardPositive (RTP)`.

For 64-bit DFP numbers, of decimal64 type, the precision is $p = 16$ decimal digits. The decimal64 significand is 54 bits, because the maximum significand supported is $10^{16} - 1$, which is less than 2^{54}. The biased exponent is ten bits, and one bit represents the sign.

BID lends itself to high-performance binary circuits, since the significand is a binary integer. However, a challenge is performing efficient significand alignment and rounding. Section 3 describes our technique for BID addition and subtraction and illustrates the challenges that we address in this paper.

3. BID Addition/Subtraction Technique

In the following discussion, let A and B be the DFP operands represented by the triples of (A_{sign}, A_c , A_{exp}) and (B_{sign}, B_c , B_{exp}), respectively. The subscripts *sign, c,* and *exp* represent the sign, significand, and exponent of an operand, respectively. In our design, the inputs may be swapped to enable the simplifying assumption that $A_{exp} \geq B_{exp}$. The swapped operands, A_N and B_N, are represented by the triples (A_{Nsign}, A_{Nc} , A_{Nexp}) and (B_{Nsign}, B_{Nc} , B_{Nexp}), respectively.

To help understand the hardware implementation of our BID adder, we first describe a high-level approach for BID addition. Abstractly, the addition of two DFP numbers can be thought of as an alignment of the significands so that the exponents are equal, followed by adding the aligned significands, followed by rounding the intermediate result to the format's precision. Since DFP numbers have an exponent base of 10, alignment of significands corresponds to multiplication by powers of 10. Rounding DFP numbers by d decimal digits is equivalent to discarding d digits, followed by a possible increment of the truncated significand, depending on the rounding mode, and an increase of the exponent by d.

As a potential technique for implementing BID addition in the decimal64 format with $p = 16$, consider the addition of $A_N = 1{,}234{,}567{,}890{,}123{,}456 \times 10^{17}$ plus $B_N = 6{,}543{,}210{,}987{,}654{,}321 \times 10^{12}$. With this technique, A_{Nc} is first multiplied by $10^{(Aexp\ -\ Bexp)} = 10^5$ to align it with B_{Nc}, having an exponent of 12. After addition, the intermediate significand is $Z_{Ic} = 123{,}463{,}332{,}223{,}333{,}254{,}321$ and the intermediate exponent is $Z_{Iexp} = B_{Nexp} = 12$. The 21-digit intermediate significand is then rounded to fit in 16 digits, so $d = \text{digits}(Z_{Ic}) - p = 21 - 16 = 5$ digits are rounded off and the intermediate exponent is increased by d. Thus, in the `RTZ` rounding mode, the correctly rounded significand and exponent are $Z_c = 1{,}234{,}633{,}322{,}\ 233{,}332$ and $Z_{exp} = B_{Nexp} + d = 12 + 5 = 17$. With unconstrained hardware resources, this approach suffices for all exponent values, but it is not practical. If A and B have the decimal64 format's maximum exponent difference of 767, Z_{Ic} has over 2,500 bits, which is not practical for a rounder to handle.

```
BID ADDITION/SUBTRACTION ALGORITHM
Step1: Compare Exponents $K = |A_{exp} - B_{exp}|$
       Swap operands if $(A_{exp} - B_{exp}) < 0$
       Determine Effective Operation (EOP)
Step2: $Q_a$ = digits($A_{Nc}$)
Step3: Examine $r = Q_a + K$
 Case1: $r \leq 19$ AND $K \neq 0$
   $Z_{Ic} = |10^K A_{Nc} \pm B_{Nc}|$
   $Z_{Isign} = (10^K A_{Nc} \pm B_{Nc} < 0)$
   $Q_I$ = digits($Z_{Ic}$)
   $d1 = \max(0, Q_I - 16)$
   $Z_c$ = round($Z_{Ic}$, d1)
   $Z_{exp} = A_{Nexp} - d1$
   $Z_{sign} = A_{Nsign}$ XOR $Z_{Isign}$
 Case2: K == 0 // thus $r \leq 16$
   $Z_{Ic} = |A_{Nc} \pm B_{Nc}|$
   $Z_{Isign} = (A_{Nc} \pm B_{Nc} < 0)$
   $Z_{sign} = A_{Nsign}$ XOR $Z_{Isign}$
   If ($Z_{Ic} < 10^{16}$)
        $Z_c = Z_{Ic}$
        $Z_{exp} = A_{Nexp}$
   Else
        $Z_c$ = round($Z_{Ic}$, 1)
        $Z_{exp} = A_{Nexp} + 1$
 Case3: $r > 19$ // thus $K \geq 4$
   $g = 16 - Q_a$
   $d3 = K - g$
   $Z_{IC} = 10^g A_{Nc} \pm$ round($B_{Nc}$, d3)
   $Z_{Iexp} = A_{Nexp} - g$ // $= B_{Nexp} + d3$
   $Z_{sign} = A_{Nsign}$
   If ($Z_{IC} \geq 10^{16}$)
      // second round needed:
      $Z_c$ = round($Z_{IC}$, 1)
      $Z_{exp} = Z_{Iexp} + 1$
   Elsif ($Z_{IC} < 10^{15}$)
      // recalculate:
      $Z_c = 10^{g+1} A_{Nc}$ - round($B_{Nc}$, d3-1)
      $Z_{exp} = Z_{Iexp} - 1$
   Else
      $Z_c = Z_{IC}$
```

Figure 1: BID Addition/Subtraction Algorithm

Our design uses a BID rounder that can handle inputs of up to 64-bits, such that intermediate results can have values up to 2^{64}-1 = 18,446,744,073,709,551,615, which is a 20 digit number. This size is chosen because the critical operation in BID-based rounding is multiplication [20], and many processors include a 64-bit by 64-bit multiplier. With a rounder of this size, we divide the problem space into three cases. In the first case, Z_{Ic} is guaranteed to fit into the 64-bit rounder, and an approach similar to that shown in the previous example is used. The second case is a special case of the first case, where $A_{exp} = B_{exp}$. This case occurs frequently in DFP applications [21], does not require significant alignment, and requires rounding of at most one digit. In the third case, the intermediate result is too large for the rounder and an alternative approach is necessary.

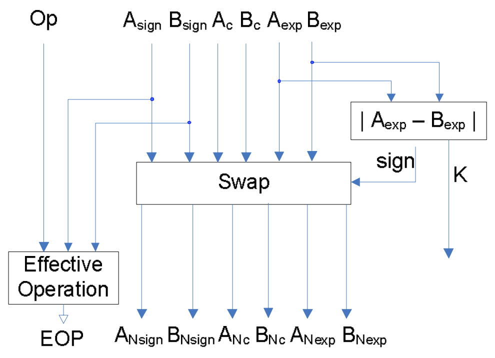
3.1 BID Addition/Subtraction Algorithm

In this section we present our proposed BID addition/subtraction algorithm and implementation. The general algorithm is given in Figure 1. For simplicity, the algorithm does not address exceptions or special case handling for NaN and Infinity. We describe sections of the algorithm in general terms and the modules used to implement them. In Section 4, we present the complete BID adder design.

In the algorithm, three functions are used, with the following definitions:

digits(n1) -	the number of decimal digits in *n1*
max(n1, n2) -	the greater of *n1* or *n2*
round(n1,d) -	the value of *n1* after rounding off *d* digits in the prevailing rounding mode.

Step 1 of the BID Addition/Subtraction Algorithm consists of some initial computations to prepare the operands for further processing. First, the operands may be swapped, as shown in Figure 2, so that A_N is the operand with the larger exponent in the rest of the circuit. This is similar to a technique used in BFP addition [20]. Also, the effective operation (*EOP*) is computed, based on the input operation (*OP*) and the signs of the operands, A_{sign} and B_{sign}, as $EOP = OP$ xor A_{sign} xor B_{sign}, where *OP* is zero for addition and one for subtraction.



Op
A_{sign} B_{sign} A_c B_c A_{exp} B_{exp}
$|A_{exp} - B_{exp}|$
sign
K
Swap
Effective Operation
EOP
A_{Nsign} B_{Nsign} A_{Nc} B_{Nc} A_{Nexp} B_{Nexp}

Figure 2: Hardware for Algorithm Step1

Step 2 of the BID Addition/Subtraction Algorithm consists of calculating the number of decimal digits in A_{Nc}, which is referred to as Q_a. Counting decimal digits is important in BID-based DFP hardware. For example, it is also needed in Case 1, which is described later.

Furthermore, counting decimal digits is a challenge since BID significands are represented in binary. To resolve this challenge, we present a novel hardware component to calculate the number of decimal digits of a BID significand.

A general top-level design of this digit-counter is shown in Figure 3, whose input to x in Step 2 of the BID Addition/Subtraction Algorithm is A_{Nc}. The module uses a binary leading-1 detector to determine the bit position, m, which indexes into a lookup table (LUT) to estimate the number of decimal digits in x. This estimate is denoted as n and may be one digit less than the actual number of decimal digits. If m is the position of the most significant bit, the decimal value range is $[2^m, 2^{m+1} - 1]$. Table 1 illustrates the relationship between m and the number of decimal digits, and shows that there is an uncertainty in roughly 1 out of $\log_2 10$ entries, based on where binades and decades overlap. The idea is simple, but the table helps illustrate the motivation behind the design of the Decimal Digit Counter. In Table 1, the lower number of the rightmost column is bolded and is the value of n for the LUTs. Table 1 shows m up to 63, the largest value needed in our design. In Step 2, m up to 53 suffices.

Table 1: Bit position in unsigned binary number versus number of decimal digits

Leading 1 bitpos (m)	Decimal Value Range	Decimal Digits
0	1	**1**
1	2-3	**1**
2	4-7	**1**
3	8-15	**1** or 2
4	16-31	**2**
5	32-63	**2**
6	64-127	**2** or 3
7	128-255	**3**
8	256-511	**3**
9	512-1,023	**3** or 4
10	1,024-2,047	**4**
…	…	**…**
63	9,223,372,036,854,775,808-18,446,744,073,709,551,615	**19** or 20

In our implementation, each entry is indexed by m and each entry of the LUT contains n, the minimum number of decimal digits for a given value of m. For example, if the leading one of the BID significand x is in bit position m = 3, then x can be between 8 and 15. As shown in Table 1, the minimum number of decimal digits that x can have in this case is n = 1. Thus, this lookup table provides the number of decimal digits in x, with an error of at most one. In this example, if the input significant is 12, the number of decimal digits is 2.

Another lookup table, indexed by m, stores pre-calculated values of 10^n, such that 10^n is the smallest power of ten greater than 2^m. Following the earlier example, the power of ten stored in index m = 3 of this second lookup table is 10 (since $10 > 2^3$). Finally, if $x < 10^n$ then n is chosen as the output of the digit counter; otherwise $n+1$ is chosen.

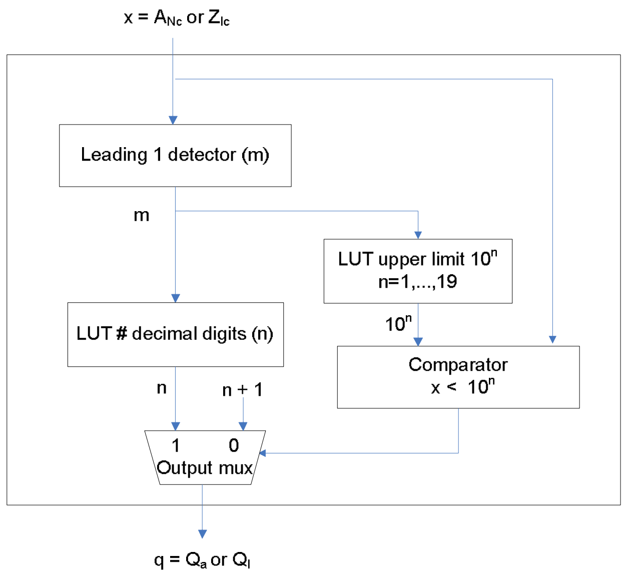


Figure 3: Decimal digit counter

Due to using a rounder that handles inputs with at most 20 digits, we characterize the input as one of three cases: (1) $A_{Nc} \times 10^K < 10^{19}$ and $A_{Nexp} \neq B_{Nexp}$, (2) $A_{Nexp} = B_{Nexp}$ (thus $A_{Nc} \times 10^K < 10^{16}$), and (3) $A_{Nc}\ 10^K > 10^{19}$.

In Case 1, $Z_{Ic} = |10^K \times A_{Nc} \pm B_{Nc}|$ can be handled by our rounder, and several computations are performed. First, A_{Nc} is multiplied by 10^K, which is obtained from a LUT indexed by K. The hardware for the LUT and multiply is shown at the top of Figure 4. Next, the Add/Subtract and Absolute Value Unit computes $Z_{Ic} = |10^K \times A_{Nc} \pm B_{Nc}|$ and $Z_{Isign} = (10^K \times A_{Nc} \pm B_{Nc} < 0)$ based on EOP. Z_{Isign} is used to help determine the sign of the final result. Once Z_{Ic} is computed, it is sent to the rounder. To determine the number of digits to round off, a digit counter is used. The number of digits to round off is $d1 = \max(Q_I - 16, 0)$, where $Q_I = \text{digits}(Z_{Ic})$.

Case 2, in which $A_{exp} = B_{exp}$, represents an optimized Case 1. A previous study [21], found that a large percentage of DFP addition operations have operands with identical exponents. In some DFP applications, $A_{exp} = B_{exp}$ in over 90% of the addition operations [21]. Because this case is so common, Amdahl's Law suggests that it is worth optimizing.

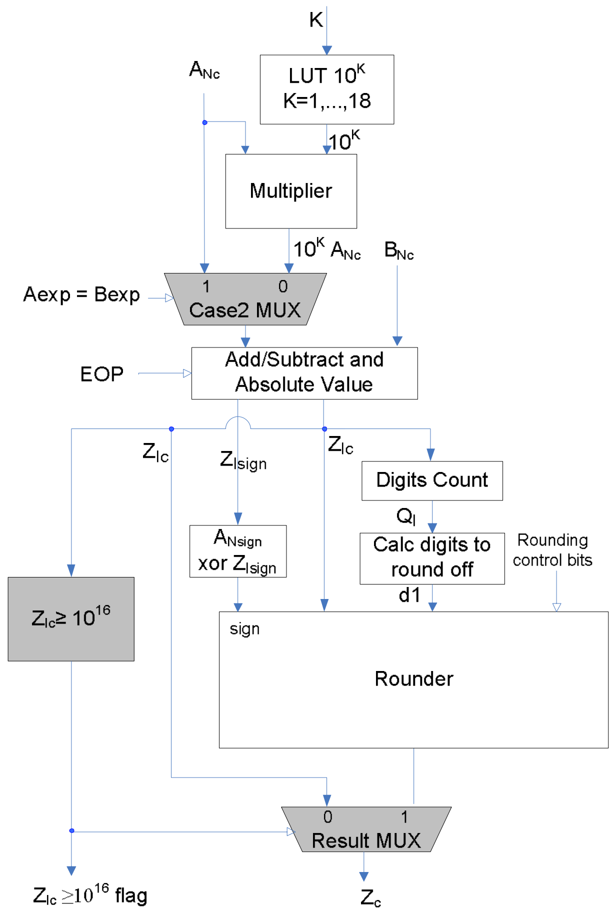


Figure 4: Direct hardware for Cases 1 and 2

In Case 2, the input exponents are equal and thus significand alignment is not needed, saving a multiply. Also, rounding is only needed when $Z_{Ic} \geq 10^{16}$. Consequently, the rounder may be skipped, with detection of the case where the result does not fit in the format's precision. Handling Case 2 with the ability to bypass the rounder comes at a small incremental hardware cost, requiring just two multiplexers and logic to detect if $Z_{Ic} \geq 10^{16}$, as shown in gray in Figure 4. This detection logic sets a flag when $Z_{Ic} \geq 10^{16}$ to indicate that the final result should be taken from the rounder. When $Z_{Ic} \geq 10^{16}$, the rounder rounds one digit from Z_{Ic} and the exponent logic sets Z_{exp} to $A_{Nexp} + 1$.

The same hardware used in Case 2 can also improve the latency of Case 1, when $Z_{Ic} < 10^{16}$ and $A_{exp} \neq B_{exp}$. Since rounding is not needed when $Z_{Ic} < 10^{16}$, the result may come directly from the Add/Subtract and Absolute Value Unit, bypassing the rounder. This improvement is not shown in Case 1 of Figure 1 and is not implemented in our BID adder, but it can easily be added.

Case 3 handles the situation when $Z_{Ic} = |10^K \times A_{Nc} \pm B_{Nc}|$ is too large to be handled by the rounder. The hardware shown in Figure 5 follows the BID Addition/Subtraction Algorithm by rounding B_{Nc} to give B', which is then added to or subtracting from $10^g \times A_{Nc}$. As shown in Figure 5, the hardware to handle this case consists of similar components to Cases 1 and 2, but the ordering differs. The first step is to determine how many digits need to be rounded from B_{Nc}. B_{Nc} is rounded by $d3 = K - g$ digits, where $g = 16 - Q_a$ and $K = | A_{exp} - B_{exp} |$. The main idea here is to compute the number of digits in B_{Nc} that do not overlap with an aligned $10^g \times A_{Nc}$ that occupies the full precision of 16 digits. A_{Nc} is multiplied by 10^g to ensure its most significant digit is in the format's most significant digit position. As in Case 1, this is accomplished with a LUT before the multiplier, but in this case the index is g. B' is then added to or subtracted from $10^g \times A_{Nc}$, depending on EOP. The multiplier may be bypassed if $g = 0$.

The algorithm, as we have discussed up until now provides correctly rounded results for the overwhelming majority of inputs. However, Case 3 leaves two situations in which results need to be adjusted to comply with IEEE P754. The first is when the EOP is addition and the number of digits in the intermediate result Z_{Ic} exceeds the format precision ($Z_{Ic} \geq 10^{16}$). The second occurs when the EOP is subtraction and the intermediate result has too few digits of precision ($Z_{Ic} < 10^{15}$). To illustrate these two cases, we show two examples.

The problem of too many digits in Z_{Ic} occurs, for example, when adding $A = 9{,}999{,}999{,}999{,}995{,}555 \times 10^{11}$ plus $B = 5{,}555{,}400{,}000{,}000{,}001 \times 10^0$. To realize the addition using our BID adder, the operands follow the hardware path shown in Figure 5 for Case 3. Since $Q_a = 16$ and $g = 0$, operand A need not be pushed, but operand B is rounded off by $d3 = K - g = 11$ decimal digits. If we consider the RTZ rounding mode, this gives us the addition of operand $A_{Nc} = 9{,}999{,}999{,}999{,}995{,}555$ and the rounded operand $B' = 55{,}554$ to produce the intermediate significand $Z_{Ic} = 10{,}000{,}000{,}000{,}051{,}109$, having 17 digits, which is one digit more than the format precision. To obtain the correctly-rounded 16-digit result, the least significant digit of Z_{Ic} is rounded to produce $Z_c = 1{,}000{,}000{,}000{,}005{,}110$ and the inter-mediate exponent is incremented to produce $Z_{exp} = 12$.

The problem of too few digits in Z_{Ic} is shown with the subtraction of $B = 1{,}111{,}222{,}233{,}340{,}000 \times 10^7$ from $A = 1{,}000{,}111{,}122{,}223{,}333 \times 10^{11}$. As in the previous example, to realize the subtraction the operands follow the hardware path shown in Figure 5. In this case, the number of digits rounded off from the operand B is the exponent difference of K = 4. Thus, the operands for subtraction are $A_{Nc} = 1{,}000{,}111{,}122{,}223{,}333$ and $B' = 111{,}122{,}223{,}334$ whose resultant significand is $Z_{Ic} =$

999,999,999,999,999, which has 15 digits, one digit less than the format precision. To obtain the correctly rounded 16-digit result, the alignment and rounding are recalculated, with g increased by 1 and $d3$ decreased by 1, so that one less digit is rounded off. This gives Z_c = 9,999,999,999,999,990 and Z_{exp} = 10.

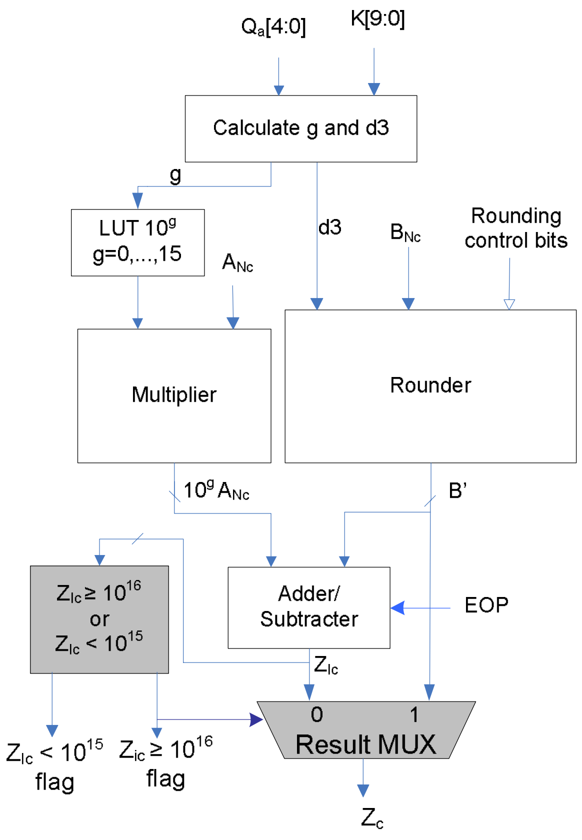


Figure 5: Direct hardware for Case 3

In Case 3, the algorithm detects the occurrence of either of these situations. The detection can be enhanced by restricting it to test for greater than or equal to 10^{16} if the operation is addition and less than 10^{15} if the operation is subtraction, as shown in Figure 5. If either of these situations is detected, the results can be fed back through the rounder or recalculated. This technique provides variable latency results, and it is the approach used in our design. The need to feed results back through the rounder or to recalculate results should be rare, as it only occurs in Case 3 when the four most significant digits of $10^g \times A_{Nc}$ are 9999 for addition or 1000 for subtraction.

3.2 BID Rounder Enhancements

The rounder design is an important component of the overall BID adder. We use a rounder that has been enhanced from the BID rounder design presented in [16]. The design presented in [16] only performs rounding up to 16 decimal digits and cannot be used to implement our algorithm. There are two major enhancements to this design. First, it has been expanded to use a 64-bit multiply as necessary in our algorithm. Second, it has more control bits to allow the rounding direction to be determined based on external information. The added control bits and their function are listed below.

a_odd_even: indicates whether A is odd or even
override_active_in: indicates whether in Case3
sub_rnd_mode: indicates if the EOP is subtraction;, which affects rounding decisions, as shown in Table 2
rnd2_active: indicates second pass through the rounder to avoid double rounding errors
rnd2_active_prev_dir: on the second pass through the rounder, indicates the direction of the first pass to avoid double rounding errors

In Case 3, the *EOP* affects the rounding direction, as shown in Table 2, where | denotes logical OR and & denotes logical AND. In this table, *f* is the fraction being rounded off, and *odd* is set if the truncated value of $10^g \times A_{Nc} \pm B'$ is odd. An increment is defined as adding 1 to the value of the rounder input after truncating *d* digits.

As an example, assume A = 5,000,000,000,000,004 × 10^0, B = 1,500,000,000,000,000 × 10^{-15}, and RTA rounding. The infinitely precise result for addition is 5,000,000,000,000,005.500000000000000 × 10^0, which rounds to 5,000,000,000,000,006 × 10^0. For subtraction, it is 5,000,000,000,000,002.500000000000000 × 10^0, which rounds to 5,000,000,000,000,003 × 10^0. B_{Nc} is rounded by $d3 = K - g = 15$ digits before it is added to or subtracted from $10^g \times A_{Nc}$. If *EOP* is addition, *B'* should be 1, and if *EOP* is subtraction, *B'* should be 2. Thus, the rounding direction for *B'* can vary based on *EOP*.

Table 2: Increment conditions for Case 3

Rounding Mode	Addition Increment Condition	Subtraction Increment Condition
RTZ	Never	$f \neq 0$
RTA	$f \geq ½$	$f > ½$
RTE	$f > ½$ | ($f = ½$ & odd)	$f > ½$ | ($f = ½$ & odd)
RTP	$\sim A_{Nsign}$ and $f \neq 0$	A_{Nsign} and $f \neq 0$
RTN	A_{Nsign} and $f \neq 0$	$\sim A_{Nsign}$ and $f \neq 0$

4. Combined BID Adder Design

Cases 1-3 have been presented to explain the ideas behind the BID adder design. Figure 6 shows our complete BID adder design, which shares components to use less hardware, by adding multiplexers and control logic. With intelligent scheduling and adding an additional path through the rounder, only one multiplier is needed. The multiplication used to push A_{Nc} to the full decimal64 precision of p = 16 digits is performed with the multiplier inside the rounder. To simplify Figure 6, the multiplier is darkened to illustrate this point.

The ability to reuse the multiplier in the DFP unit has important effects. First, by adding logic around the multiplier, many functions may be incrementally added to a DFP solution at a modest area cost. Second, the high utilization of the multiplier requires sophisticated scheduling with several possibilities for optimization including adding buffers, reservation stations, and control logic. These details are not shown in Figure 6.

5. Results

To verify our algorithm, we modeled a BID adder in Verilog with a separate path for each case and additional steps for the cases that require a second pass through the rounder. Though this design is too large to be practical, we have successfully run over 10 million random test vectors and over 300 directed corner testcases.

As a more realistic design, we have pipelined the BID adder and combined the datapaths so that only one 64-bit multiplier is used. In this design, the multiplier has two pipeline stages, the rounder has four pipeline stages, and a flag is set in Case 3 when $Z_{IC} < 10^{15}$ to indicate that the result must be recalculated. Except for infrequent cases that need a second rounding or recalculation, the latency is seven cycles for Cases 1 and 3, and three cycles for Case 2. In comparison, the average latency of a 64-bit DFP addition using a BID software library and executing on an EM64t Xeon 5100 Processor is 71 cycles when function call overhead is not included [5].

We have performed preliminarily synthesis, testing, and evaluation of our BID adder using Mentor Graphics ModelSim, Synopsys Design Compiler, and the LSI Logic Gflxp 0.11 micron CMOS Standard Cell Library. In this technology, a 2-input NAND gate's area is 8.08 μm^2, and a fan-out-of-four (FO4) inverter's delay is 55 ps. Our preliminary synthesis indicates the total area is roughly 0.55 mm^2 (68,459 NAND gate equivalents) and the critical path delay is roughly 2.4 ns (44 F04 inverter delays). We believe the delay can be improved significantly through code rewriting, timing path adjustments, further design optimizations, and deeper pipelining.

For comparison, we synthesized a 2-stage pipelined Synopsys DesignWare 64-bit by 64-bit multiplier, named DW02_mult_2stage. The total area of this multiplier is 0.41 mm^2. By this measure, the multiplier comprises roughly 70% of the total area of the pipelined BID adder design. These results indicate that BID addition and subtraction can be achieved with a modest increase in area when hardware is shared with an existing BFP multiplier. Since we are using a synthesized multiplier, we are encouraged that efforts to improve the multiplier will also likely enhance the BID adder.

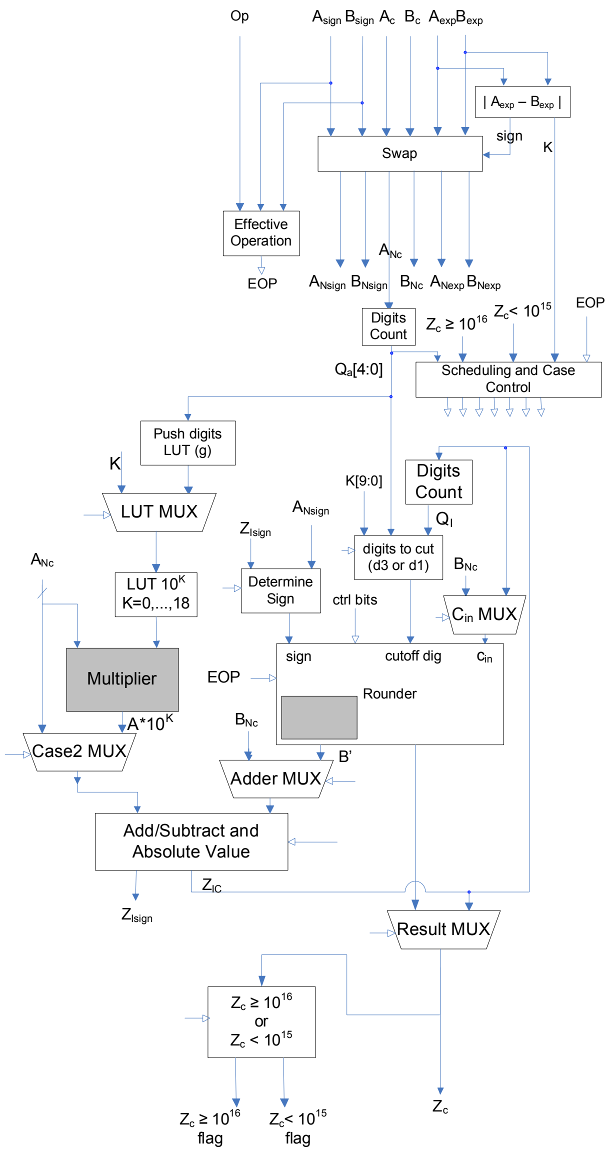


Figure 6: Adder hardware design with combined paths for Cases 1 - 3

6. Conclusion

We have presented the first design of a BID-based DFP adder, which provides correctly rounded results for adding and subtracting IEEE P754 decimal64 numbers. The design demonstrates that BID addition and subtraction can be effectively achieved in hardware. It can be adapted to also support the IEEE P754 operations of comparison, minimum, maximum, quantize, and toIntergalValue, and it can be adapted for other operand sizes. The design is promising in terms of area and potential hardware reuse. Over 70% of the BID adder's area is due to a 64-bit binary multiplier, which can be shared with a BFP multiplier and other BID operations. In future research, we plan to investigate the design of shared IEEE P754 BFP and DFP units.

Acknowledgements

The research presented in this paper is supported 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 design.

References

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

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

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

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

[5] 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", *IEEE International Symposium on Computer Arithmetic*, pp. 29-37, June 2007.

[6] 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*, Portugal, September 2007.

[7] 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, pp. 31-58, 1987.

[8] 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, pp. 370-377 April 1983.

[9] 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, pp. 951-961, September 2006.

[10] 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,* pp. 84-95, September 2004.

[11] 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*, pp. 309-315, July 2005.

[12] 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.

[13] 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.

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

[15] L.-K. Wang and M. J. Schulte, "Decimal Floating-Point Adder and Multifunction Unit with Injection-Based Rounding," *IEEE International Symposium on Computer Arithmetic*, pp. 56-68, June 2007.

[16] C. Tsen, M. J. Schulte, and S. Gonzalez-Navarro, "Hardware Design of a Binary Integer Decimal-based IEEE P754 Rounding Unit," *Proceedings of the IEEE International Conference on Application-Specific Systems, Architectures, and Processors*, pp. 115-121, July 2007.

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

[18] P. Tang, "Binary-Integer Decimal Encoding for Decimal Floating-Point," Intel Corporation, *Available at http://754r.ucbtest.org/issues/decimal/bid_rationale.pdf*, July 2005.

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

[20] M. Ercegovac, T. Lang, *Digital Arithmetic*, "Floating-Point Representations, Algorithms, and Implementations," Morgan Kaufmann Publishers, pp. 397-479, 2004.

[21] L.-K. Wang, C. Tsen, M. J. Schulte, D. Jhalani, "Benchmarks and Performance Analysis for Decimal Floating-Point Applications," *IEEE International Conference on Computer Design*, October 2007.