ALOHA-FP2I: Efficient Algorithms and Hardware for Multi-Mode Rounding of Floating Point to Integer

MAHENDRA RATHOR

School of Instrumentation, Devi Ahilya University, Indore, India, mahendra7790@gmail.com

Modern technology is relying on hardware accelerators to achieve enhanced performance of computing systems. In the modern computing paradigm, floating point representation of numbers has gained popularity owing to its wide dynamic range. Rounding of floating point numbers to integer is used in modern processor architectures e.g. ARM and Intel's architecture (IA) as well as in specific applications such as multimedia. However, the academic literature lacks discussion on hardware designs for rounding binary floating point numbers to integer in different rounding modes. This paper presents novel efficient algorithms and hardware architecture designs for rounding binary floating point numbers to the integer for the following rounding modes: round towards zero, round up (towards positive infinity), round down (towards negative infinity), round to the nearest integer, and round to nearest even. The paper also proposes an integrated multi-mode rounding (IMR) algorithm and hardware design which can be configured to a specific rounding mode among the above mentioned five modes. This paper proposes a mantissa bit of rounding (MBR) to determine the condition of rounding for the various modes. The MBR is identified on the basis of the dynamic range and precision features of floating point representation. To the best of our knowledge, we present the individual as well as an integrated hardware design for the various rounding modes for the first time in the literature. The proposed designs have been implemented on an FPGA platform to analyze the design metrics such as area, delay and power. The results imply that the proposed designs are suitable to aid the intended hardware accelerators as they are efficient in terms of the design parameters. Moreover, this paper presents the integration of the proposed rounding hardware design with the compression processor and evaluates the integration overhead which is found to be nominal (<1%).

Additional Key Words and Phrases: Binary floating point, rounding modes, hardware design

1 INTRODUCTION

Hardware acceleration has fulfilled a pivotal role in realizing fast and high throughput computing systems of modern age. Performing data and computationally intensive functions through a dedicated hardware counterpart not only accelerates the computing task to offer the higher performance but also saves the power [1]. For instance, multimedia and machine learning applications perform large computations on ample amount of data. This entails employing dedicated hardware to accelerate their processing [2]-[5]. The computations in the multimedia and machine learning applications are largely performed on real numbers.

The real numbers are commonly represented in floating point representation in the computers because of their capability of offering wide dynamic range and dynamic precision. According to the IEEE-754 standard, there exist various rounding modes such as round towards zero, round up (round towards positive infinity), round down (round towards negative infinity), round to nearest (where ties break-up with round away from zero) and round to even (where ties break-up with the nearest even integer) to round the results obtained from the elementary operations such as multiplication, division, square root, and so on [6], [7]. Further, in some specific applications also, the floating point representation is desirable over the integer representation such as in High Dynamic Range (HDR) photography and videography. To support the HDR, the sample values are represented in floating point numbers [8]-[10]. For example, half precision floating point number representation is used by the EXR format which is employed in post-

ACM 1539-9087/2024/10-ART
http://dx.doi.org/10.1145/3701560

processing of cinematic material. It can cover dynamic ranges up to 10.7 magnitudes in luminance [8]. Additionally, modern shading units in GPUs denote the pixel information in floating point [11]. Further, floating point samples are required to be converted to integer samples in the most audio programs/processing [12]. Intel provides an optimized signal processing library which has fast rounding and conversion functions [12].

Floating point rounding to integer (or whole number) is also useful in various problems such as polymatroid and branching optimization problems [13], GPS integer ambiguity resolution [14], integer optimization and relaxed integer programming problems [15] etc. Apart from this and more popularly, specific applications such as multimedia and machine learning require rounding of fractional values to the integer as intermediate data or final result. For instance, multimedia applications such as JPEG image compression and MPEG video compression require rounding to integer value in the quantization step. More explicitly, the lossy compression processor performs discrete cosine transformation (DCT) followed by quantization step wherein the rounding to integer is performed after the multiplication/division operation [16], [17]. Further in the machine learning (ML) applications such as K-means clustering, the numbers encountered during the computations are quantized into the corresponding integer value using the rounding operation. Here, this rounding to integer operation enables the computation of the Euclidean distances using simple integer arithmetic [18]. Additionally, ML or deep learning based regression models for user ratings prediction or recommendation systems also require rounding of decimal data to integer [19]. Thereby, efficient methods of rounding of floating point numbers to integer using the desired rounding mode are required.

In neural network training, some portion of the process such as accumulators, weights, gradients, error and activation computation during back-propagation rely on floating-point arithmetic to ensure successful training (high numerical accuracy) [20]. However, conversion of floating point numbers into integer (fixed-point format) is required to store data into desired format efficiently [20]. The proposed approach converts the floating point numbers into integer in the floating-point format. However, there are methods and hardware designs that can translate the floating point format to fixed point format. Therefore, the hardware for floating point to fixed point conversion can be used in conjunction with the proposed rounding hardware to achieve the result in the desired format. To the best of our knowledge, there is no such design available that directly rounds the floating point numbers into integer in the fixed-point format.

1.1 Motivation and Novel Contributions

Floating point rounding to integer operations are employed in both general purpose processor and application specific processors. The general purpose processor or instruction set architecture of Intel and ARM use some specific instructions for floating point rounding [21], [22]. For example, Intel-64 or IA (Intel Architecture)-32 use instructions such as `ROUNDSD' and `ROUNDSS' for rounding scalar double precision and scalar single precision floating-Point values respectively [21]. The rounding process of `ROUNDSD' and `ROUNDSS' instructions returns the integer result as a double and single precision floating-point value respectively. There is an immediate operand field in these instructions which specifies the control fields for the rounding operation. The particular two bits i.e. the Rounding Control (RC) bits (b1b0) of the immediate operand field help select the specific rounding-mode [21]. The following rounding modes are supported in `ROUNDSD' and `ROUNDSS': round to nearest even (using b1b0=00), round down (using b1b0=01), round up (using b1b0=10), round towards zero (using b1b0=11). Similarly, ARM instructions such as `VCVT' and `VCVTR' are used to convert between floating-point and integer [22]. Although the Intel and ARM architectures are facilitating the various floating point rounding to integer operations, however the rounding hardware designs have not been explicitly discussed yet and are not publicly available for the academic research.

On the other hand, the need for rounding hardware is also escalating in the application specific processors such as multimedia and ML processors. Since the frameworks of multimedia and ML applications are computationally intensive, therefore they are efficient to be executed using dedicated hardware for achieving acceleration. We can achieve the enhanced acceleration and higher performance by executing the desired rounding operations in the multimedia and ML applications through a dedicated hardware. Hence, designing area-power efficient and high performance hardware architectures for rounding of the floating point numbers to the integer can play a crucial role in aiding multimedia and ML accelerators.

In the literature, Tsen et al. [25] have proposed rounding algorithm and corresponding hardware unit in case of decimal floating point arithmetic (binary integer decimal) for the various rounding modes. However, the rounding algorithm and hardware designs for binary floating point have not been presented or discussed in [25] unlike the proposed approach. Rathor et al. [26] discussed rounding of floating point numbers in round to nearest mode, however they did not analyzed the other rounding modes. Further, algorithms for the stochastic rounding of elementary arithmetic operations in the floating-point arithmetic have been proposed by Fasi and Mikaitis [23]. However, they neither specifically target rounding to the whole number nor the hardware designs for all the conventional rounding modes. Furthermore, Mikaitis [24] have proposed algorithm and hardware design for the stochastic rounding of fixed-point arithmetic. However, it [26] lacks discussion on hardware designs for the binary floating point rounding to integer. Apart from these, rounding using the round-to-nearest (RN) coding [27] based number representation has been proposed. This coding style leverages the property of signed-digit representation wherein the rounding to the nearest is identical to truncation. The formats or methods for the arithmetic operations under round-to-nearest were proposed [28], [29]. However, the focus of our work is on the conventional floating point representations of numbers unlike [28], [29]. More precisely, we mainly focus on the rounding to the integer for the various rounding modes.

To the best of our knowledge, the hardware designs for rounding of the binary floating point numbers to the nearest integer (in floating point type) for the various rounding modes have not been explicitly discussed in the literature so far. Taking into account the significance of the rounding hardware design in the instruction set architectures and to aid ML and multimedia accelerators in modern computing paradigm, this paper presents the following main contributions as below:

- This paper presents algorithms of rounding a given binary floating point number to the integer for the various rounding modes viz. round to nearest integer (with the halfway cases rounded away from zero), round up, round down, round towards zero and round to even (with the halfway cases rounded to nearest even). The returned integer value of the proposed rounding designs have the floating-point type. The proposed idea of rounding is presented for the multi-precision (32-bit and 64-bit) floating point numbers and is extendable to the rounding of 128-bit precision as well, without making considerable changes in the proposed algorithm.

- This paper proposes a *mantissa bit of rounding (MBR)* within the mantissa field to make the rounding decisions for the various rounding modes.
- This paper presents integrated multi-mode rounding (IMR) algorithm and hardware design which can be configured to a particular mode out of the above mentioned five modes. The proposed IMR design achieves a good savings in resources at the cost of slight increase in propagation delay compared to the five individual rounding modes.
- This paper also proposes low-cost register transfer level (RTL) hardware designs for rounding the floating point number to integer value for the above mentioned rounding modes. The rounding hardware have been implemented and functionally validated on an Intel's FPGA to analyze of the design metrics of the proposed rounding hardware units.
- We integrate the proposed rounding modules with the compression processor to analyze the integration overhead which is found to be nominal (less than 1%).

The rest of the paper is organized as follows. Section 2 presents the intuition behind the proposed idea of binary floating point rounding algorithm based on MBR. Further, the proposed algorithms and the hardware designs for the round to nearest, round towards zero, round up, round down and round to even modes are presented in Section 3. The results and analysis of the proposed hardware designs are discussed in Section 4. Finally, we conclude the paper in Section 5.

2 PLOT OF THE IDEA OF PROPOSED BINARY FLOATING POINT ROUNDING TO INTEGER

The proposed work targets rounding of the multi-precision floating point numbers to integer (the whole number), in contrast to the rounding with the desired precision. The definitions of rounding a floating point number [I.F] to integer for the various rounding modes is shown in Table 1, where I represents the integer part and F the fractional part. Let's assume b^s, $b^e[7:0]$ and $b^m[22:0]$ denote the sign, exponent and mantissa part of the input 32-bit binary floating point number; c^s, $c^e[7:0]$ and $c^m[22:0]$ denote the sign, exponent and mantissa part of the rounded output. Similarly, b^s, $b^e[10:0]$ and $b^m[51:0]$ denote the sign, exponent and mantissa part of the input 64-bit binary floating point number; c^s, $c^e[10:0]$ and $c^m[51:0]$ denote the sign, exponent and mantissa part of the rounded output. The basic intuition behind the proposed approach of rounding a binary floating point number to the integer value is discussed as follows. The proposed idea uses the dynamic range and dynamic precision features of the binary floating point representation to perform the rounding in the various modes.

Using dynamic range feature to determine the different range of input floating point number for which the rounding to be performed: With the increment of one in the exponent field of binary floating point representation, the range of the integer part of corresponding input numbers exponentially increases. The dynamic range attribute of floating point representation is also depicted in Table 2 and Table 3 for the single and double precision respectively. For example, in case of single precision, the exponent field $b^e[7:0]$ = "01111111" corresponds to only one integer value i.e. '1' whereas $b^e[7:0]$= "10000100" corresponds to 32 different integer values. Therefore, for a range of the numbers, the rounding to integer can be performed based a particular value of the exponent field. It is noteworthy that the $b^e[7:0]$ in between the range "00000000" and "01111110" corresponds to the values in between 0 and 1. Similarly for the double precision floating point numbers, a particular value of the exponent field helps determine the corresponding range of input numbers to be rounded as shown in Table 3.

Table 1. Definitions of rounding an input binary floating point number [I.F] to integer T for various modes, where I and F denote the integer and fractional part respectively of the input number, and T denotes the rounded result (integer)

Round to Integer Modes	Condition 'X' is	Positive numbers		Negative numbers	
		'X' is true	'X' is NOT true	'X' is true	'X' is NOT true
Round up	F=0	T:= I	T:= I+1	T:= I	T:= I
Round down	F=0	T:= I	T:= I	T:= I	T:= I+1
Round to nearest (with halfway cases rounded away from zero)	$0 \le F < 0.5$	T:= I	T:= I+1	T:= I	T:= I+1
Round towards zero	F=0	T:= I	T:= I	T:= I	T:= I
Round to even (with halfway cases rounded to nearest even)	$0 \le F < 0.5$	T:= I	If 'I' is even then T:= I else T:= I+1	T:= I	If 'I' is even then T:= I else T:= I+1

Table 2. Exponent bits and mantissa bit of rounding (MBR) of input floating point number used for rounding in case of single precision

The range of integer part of input number to be rounded (total values= 2^{22-n} integer values for a particular n)	b^e [7 : 5]	b^e [4 : 0]	MBR (b^m [n] bit of mantissa field)
1	011	11111	b^m [22]
2, 3	100	00000	b^m [21]
4 to 7	100	00001	b^m [20]
8 to 15	100	00010	b^m [19]
16 to 31	100	00011	b^m [18]
32 to 63	100	00100	b^m [17]
…	…	…	…
1048576 to 2097151	100	10011	b^m [2]

Table 3. Exponent bits and MBR of input floating point number used for rounding in case of double precision

The range of integer part of input number to be rounded (total values= 2^{51-n} integer values for a particular n)	b^e [10 : 6]	b^e [5 : 0]	MBR (b^m [n] bit of mantissa field)
1	01111	111111	b^m [51]
2, 3	10000	000000	b^m [50]
4 to 7	10000	000001	b^m [49]
8 to 15	10000	000010	b^m [48]
16 to 31	10000	000011	b^m [47]
32 to 63	10000	000100	b^m [46]
…	…	…	…
562949953421312 to 1125899906842623	10000	110000	b^m [2]

Using dynamic precision feature to determine the condition of rounding: The floating point representation of numbers exhibit the dynamic precision as the small numbers can be represented with higher precision in comparison to the larger numbers. For example, in case of single precision, the numbers 1.0 and 1.5 are represented as “0 01111111 **0**0000000000000000000000” and “0 01111111 **1**0000000000000000000000” respectively. This implies that the mantissa field can accommodate total 4,194,304 different numbers in-between 1.0 and 1.5. On the contrary, the larger numbers are represented with relatively less precision. For example, in case of single precision, the numbers 1048576.0 and 1048576.5 are represented as “0 10010011 **000000000000000000000**00” and “0 10010011 **000000000000000000001**00” respectively. This implies that the mantissa field can accommodate total only 3 different numbers in-between 1048576.0 and 1048576.5. Hence, it can be inferred that the effective part of mantissa (highlighted in the bold) expands towards the least significant bit (LSB) as we move from smaller to larger numbers. We refer to this effective part of mantissa as our *mantissa field of interest*. This mantissa field of interest is of fixed length for a particular range of the integer part of input number. Here, the least significant bit of the `mantissa field of interest' is of our interest as it helps make the decisions of rounding to integer in case of the various modes.

More precisely, within this mantissa field of interest, *we propose a mantissa bit of rounding (MBR) to determine the condition of rounding to the integer. The MBR can be defined as that bit of the mantissa field of interest which undergoes low-to-high transition when the fractional part F becomes equal to 0.5 for a particular value of exponent field.* This bit is also the least significant bit of the mantissa field of interest. The mantissa field of interest can also be defined in terms of MBR as follows. It is that part of mantissa whose most significant bit (MSB) is the MSB of mantissa but the LSB is the MBR. In case of single precision, the mantissa field of interest is represented by $b^m[22:n]$ where n denotes the index of the mantissa signifying the MBR. Similarly, $b^m[51:n]$ represents the mantissa field of interest for the double precision floating point. The index of the MBR within the mantissa is always fixed for a particular range of the integer part of input number. Table 2 and Table 3 show the MBR (n^{th} index of mantissa) for the different range of integer part of the input numbers, for the single and double precision respectively.

It is evident from the tables that as we move from smaller to larger range of numbers, the respective index of MBR propagates from MSB to LSB. For instance, the index of MBR is the 22nd bit of mantissa in case of the integer part is 1, whereas it is the 2nd bit for the range of integers from 1048576 and 2097151 as shown in Table 2. In case of single precision, Table 2 shows that total 2^{22-n} different integer values (range) can be rounded for the MBR being at n^{th} index of mantissa field which can vary from 22 down to 2. Similarly in case of double precision, Table 3 shows that 2^{51-n} different integer values can be rounded for the MBR being at a particular index of mantissa field which varies from 51 down to 2. It is to be noted that when the MBR reaches at index ‘1’, there will only be the LSB of mantissa i.e. $b^m[0]$ that can take value of either 0 or 1. In this case, only two different values can be represented in floating point representation for the fractional part varying in-between 0 to 0.5. For example, for the real values in between 2097152.0 and 2097152.5, there can have only two possible single precision floating point number representation. Similarly, when the MBR reaches at index ‘0’, then there is no range of values that can be taken by the fractional part in floating point representation for a particular real input value. Therefore, if the integer part is very large then the fractional part is insignificant. Hence, applying rounding rule is not so required.

For the larger real values that have very large integer part (for which the MBR reaches either at index 1 or 0 in floating point representation) can simply be truncated to the integer.

Finally, the plot behind our basic idea of rounding to integer for various modes can be summarized as follows:

- Using the exponent field value of input binary floating point number, we determine (i) the range of integer part for which the rounding condition is applied and (ii) the index of MBR.
- Using the mantissa field of interest and MBR, we determine the condition of rounding to the integer for the different range of integer part.

Based on the above discussed plot of our idea, the proposed algorithms and hardware designs for rounding the single and double precision floating point numbers to the integer value in the five different rounding modes are discussed in the following section.

3 ALOHA-FP2I: PROPOSED ALGORITHMS AND HARDWARE DESIGNS FOR FLOATING POINT ROUNDING TO INTEGER

This section presents the proposed algorithms and hardware designs for rounding the single and double precision floating point numbers to the integer for the following rounding modes: round to nearest, round towards zero, round towards positive infinity, round towards negative infinity and round to even. For each rounding mode, the proposed work is presented in the following sub-sections.

3.1 Rounding to Nearest Integer

In case of rounding to the nearest integer, if an input number b is in between the range $-0.5< b < 0.5$, it must be rounded to 0. Hence, the sign bit of rounded output c^s will be zero irrespective of the b^s bit. However for other cases, the c^s will remain same as that of b^s. The algorithmic flow of performing this method of determining the sign bit for the single and double precision are shown in Fig. 1(a) and (b) respectively.

Further, exponent and mantissa field of the rounded output are determined based on the mantissa field of interest and MBR of input floating point number as discussed in section 2. For the fractional part being greater than or equal to five for a particular range of the integer part (as shown in the Table 2 and Table 3), the MBR sets to '1' which becomes the basis of making the decision of rounding to the nearest whole number. In general, the following steps are performed for rounding to nearest in case of single precision.

- First, the exponent field of input floating point number is concatenated with the mantissa field of interest. Let us define it as: $B_{23-n}[30-n:0] = b^e[7:0]||b^m[22:n]$, where n denotes the MBR.
- If the MBR is 0, $c^e= B_{23-n}[30-n:23-n]$ and $c^m=B_{23-n}[22-n : 0] ||\{0\}^n$.
- If the MBR is 1, the $B_{23-n}[30-n : 0]$ is incremented by one and it is denoted using $B'_{23-n}[30-n : 0]$. Now,

$c^e= B'_{23-n}[30-n:23-n]$ and $c^m=B'_{23-n}[22-n : 0] ||\{0\}^n$.

Above, the operators '||' concatenates bit strings and $\{0\}^n$ gives a string with *n* zeros. Similarly, the steps of rounding a 64-bit floating point number to the nearest integer are as follows.

• The concatenation of the exponent field and the mantissa field of interest is defined as follows:

$B_{52-n}[62-n : 0] = b^e[10:0] || b^m[51 : n]$, where n denotes the MBR.

• If the MBR is 0, $c^e = B_{52-n}[62-n : 52-n]$ and $c^m = B_{52-n}[51-n : 0] || \{0\}^n$.
• If the MBR is 1, the $B_{52-n}[62-n : 0]$ is incremented by one and it is denoted using $B'_{52-n}[62-n : 0]$. Now,

$c^e = B'_{52-n}[62-n : 52-n]$ and $c^m = B'_{52-n}[51-n : 0] || \{0\}^n$.

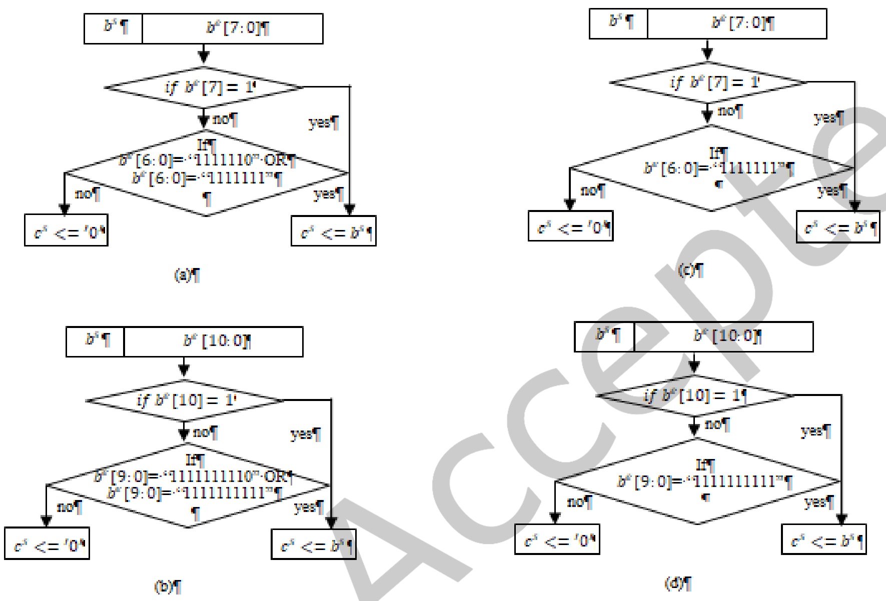


Fig. 1. Algorithmic flow of sign bit generation of rounded floating point output for the (a) round to nearest mode for 32-bit (b) round to nearest mode for 64-bit (c) round towards zero, round up and round to even modes for 32-bit (d) round towards zero, round up and round to even modes for 64-bit; Note: In case of rounding down, the sign bit is not altered.

Special Case: For the numbers that are in the range $0 \leq b < 1$, the exponent part is not constant but varies from “00000000” to “01111110” for the single precision. Similarly, it varies from “00000000000” to “01111111110” for the double precision. In this scenario, round to nearest integer is performed as follows. (i) for the range $0 \leq b < 0.5$, the output is rounded to 0, and (ii) for the range $0.5 \leq b < 1$, the exponent field is incremented by one and the mantissa field is set to 0.

Example: Let us assume an input number is 7.5 whose binary 32-bit floating point representation is “0 10000001 11100000000000000000000”. For this example, $b^e[7:5]$= “100” and $b^e[4:0]$= “00001”. In this case, the corresponding MBR is $b^m[20]$ as per the Table 2 and hence the value of n is 20. According to the proposed algorithm, firstly $b^e[7:0]$ are concatenated with $b^m[22:20]$ which results in $B_3[10:0]$= “10000001111”.

Secondly, n=20th bit of mantissa part is checked. Since it is '1', $B_3[10{:}0]$= is incremented by one resulting into $B'_3[10{:}0]$= "10000010000". Further, $B'_3[10{:}3]$ is assigned to $c^e[7{:}\ 0]$ and B'3[2:0]||"00000000000000000000" is assigned to $c^m[22{:}\ 0]$ to produce the exponent and mantissa bits of the rounded output. This produces the number "0 10000010 00000000000000000000000" which is equivalent to 8.0.

Proposed hardware design: The proposed register transfer level (RTL) hardware design of rounding 32-bit binary floating point to the nearest integer is shown in Fig. A.1 in Appendix-A, where the blocks 1, 2 and 3 corresponds to the sign bit, exponent field and mantissa field generation logic. This design first divides the [30-n : 0] into [30-n : 23-n] and [22-n : 0] for generating the exponent and mantissa part respectively of the rounded output. Further, based on the MBR, it selects either $B_{23-n}[30\text{-}n : 23\text{-}n]$ or B'_{23-n} [30-n : 23-n] for the exponent part using the multiplexers. Similarly, it selects either B_{23-n} [22-n : 0] or B'_{23-n} [22-n : 0] for the mantissa part.

3.2. Rounding towards Zero

In case of rounding towards Zero, if an input number b is in between the range $-1.0< b < 1.0$, it must be rounded to 0. Hence, the sign bit of rounded output c^s will be zero irrespective of the b^s bit. The algorithmic flows of generating the sign bit for the single and double precision are shown in Fig. 1(c) and (d) respectively. Further, mantissa field of interest plays an important role in determining the mantissa of the rounded output. For a particular range of the integer part (shown in the Table 2 and Table 3), the mantissa field of interest excluding the MBR is concatenated with a sequence of zeros to perform the rounding towards zero. The algorithm for determining the exponent and mantissa field of the rounded output, in case of single precision, is presented as follows.

- $c^e[7 : 0]= b^e[7 : 0]$.
- $c^m = b^m[22 : n+1] \,||\, \{0\}^{n+1}$, for $2 \le n \le 21$, where n is the index of mantissa signifying the MBR and $b^m[22 : n+1]$ is the mantissa field of interest excluding the MBR.
- $c^m = \{0\}^{n+1}$, for n = 22.

Similarly, rounding of a 64-bit floating point number towards zero is performed as follows.

- $c^e[10 : 0]= b^e[10{:}\ 0]$.
- $c^m = b^m[51 : n+1] \,||\, \{0\}^{n+1}$, for $2 \le n \le 50$.
- $c^m = \{0\}^{n+1}$, for n = 51.

Example: To round the number 7.5 towards zero, we find the corresponding MBR which is $b^m[20]$ as per the Table 2 and hence the value of n is 20. According to the proposed algorithm, firstly $b^m[22{:}21]$ is concatenated with "000000000000000000000" and assigned to $c^m[22{:}\ 0]$ to produce the mantissa bits of the rounded output. On the other hand, the exponent part remains same as that of input. Finally, this produces the number "0 10000001 11000000000000000000000" which is equivalent to 7.0.

Proposed hardware design: The proposed RTL design of rounding 32-bit binary floating point towards zero is shown in Fig. A.2 in Appendix-A, where the blocks 1, 2 and 3 corresponds to the sign bit, exponent field and mantissa field generation logic. As shown, the exponent field generation logic does not require any additional hardware, whereas mantissa field of rounded output can be generated using the mantissa field of interest (except the MBR) and the exponent field (acting as the select line) of input number and multiplexers.

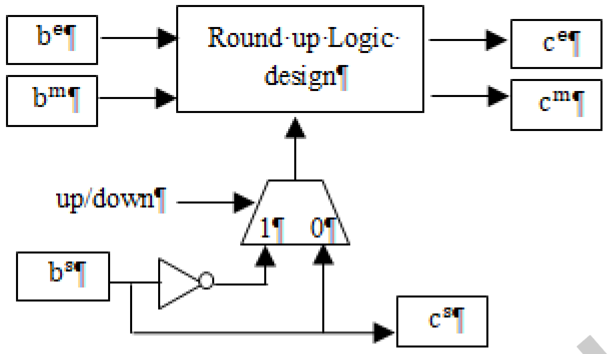


Fig. 2. RTL hardware design for rounding 32-bit floating point number towards negative infinity (Round-Down)

3.3. Rounding towards Positive Infinity

In case of rounding towards positive infinity, the sign bit of rounded output is altered only if the input number b is in between the range -1.0< b < 0. In this case, the number must be rounded to 0 hence the sign bit of rounded output c^s will be zero. The algorithmic flows of determining the sign bit for the single and double precision are shown in Fig. 1(c) and (d) respectively. In this case of rounding, positive and negative numbers require separate procedure. While the negative numbers are rounded towards zero only, the positives numbers are rounded to the next integer value for the fractional part being not equal to zero as defined in the Table 1. For determining the exponent and mantissa field of the rounded output, the following steps are performed for the single precision numbers.

If numbers are positive (b^s=0):

- First, $b^e[7 : 0]$ is concatenated with the $b^m[22 : n]$. Let us define it as: $B_{EM}[29 - n : 0] = b^e[7 : 0] \| b^m[22 : n+1]$, for $2 \le n \le 21$
- If the $b^m[n : 0]$ is 0, $c^e[7 : 0]= b^e[7 : 0]$ and $c^m[22 : 0]=b^m[22 : 0]$.
- Else, if the $b^m[n : 0]$ is not 0, the $B_{EM}[29 - n : 0]$ is incremented by one and it is denoted using $B'_{EM}[29-n : 0]$. Now, $c^e = B'_{EM}[29-n : 22-n]$ and $c^m = B'_{EM}[21- n : 0] \| \{0\}^{n+1}$

If numbers are positive (b^s=1):

- $c^e[7 : 0]= b^e[7 : 0]$.
- $c^m = b^m[22 : n + 1] \| \{0\}^{n+1}$, for $2 \le n \le 21$.
- $c^m = \{0\}^{n+1}$, for n = 22.

For the double precision numbers, the steps are as follows.

If numbers are positive (b^s=0):

- First, $b^e[10 : 0]$ is concatenated with the $b^m[51 : n+1]$. Let us define it as:
  $B_{EM}[61 - n : 0] = b^e[10 : 0] \| b^m[51 : n+1]$,
- If the $b^m[n : 0]$ is 0, $c^e[10 : 0]= b^e[10 : 0]$ and $c^m[51 : 0]=b^m[51 : 0]$.
- Else, if the $b^m[n : 0]$ is not 0, the $B_{EM}[61 - n : 0]$ is incremented by one and it is denoted using $B'_{EM}[61-n : 0]$. Now, $c^e= B'_{EM}[61-n : 51-n]$ and $c^m = B'_{EM}[50- n : 0] \| \{0\}^{n+1}$

If numbers are positive (b^s=1):

- $c^e[10 : 0]= b^e[10 : 0]$.
- $c^m = b^m[51 : n + 1] \| \{0\}^{n+1}$, for $2 \leq n \leq 50$.
- $c^m = \{0\}^{n+1}$, for n = 51.

Example: Let us assume the number to be rounded towards positive infinity is 7.4 whose 32-bit binary floating point representation is "0 10000001 11011001100110011001101". In this example, the corresponding MBR is $b^m[20]$ as per the Table 2 and $B_{EM}[9 : 0]$ is "1000000111" as per the algorithm. Since the $b^m[20 : 0]$ is not 0, hence $B_{EM}[9 : 0]$ is incremented by one which results into "1000001000". Further, the first eight bits from MSB (i.e. "10000010") are assigned to the $c^e[7: 0]$ and reaming two bits (i.e. "00") are concatenated with "000000000000000000000" and assigned to $c^m[22: 0]$ to produce the mantissa bits of the rounded output. Finally, this produces the number "0 10000010 00000000000000000000000" which is equivalent to 8.0. However, in case the number is negative; the example remains same as shown for the round towards zero.

Proposed hardware design: The proposed RTL design of rounding 32-bit binary floating point towards positive infinity is shown in Fig. A.3 in Appendix-A, where the blocks 1, 2 and 3 corresponds to the sign bit, exponent field and mantissa field generation logic. As shown, 2:1 multiplexers are used to select between the logic for the positive and negative numbers. Further, the outputs of multi-bit OR gates are used to generate the select line of the multiplexers that decide between the logic whether to increment the integer part for rounding up or not.

3.4. Rounding towards Negative Infinity

Proposed algorithm and hardware design: In case of rounding towards negative infinity, the sign bit is not altered in any case. Further, for producing the exponent and mantissa field of the rounded output, the same logic as described for the rounding towards positive infinity can be applied with a slight modification. In this case, the positive numbers are rounded towards zero only. On the other hand, the negative numbers are rounded to the next integer value for the fractional part being not equal to zero as shown in the Table 1. This condition is just opposite to that of rounding towards positive infinity. The hardware design for rounding to negative infinity is shown in Fig. 2. As shown, the hardware design for the round up can be made work in round down mode using a 2:1 multiplexer and a NOT gate. If the select line `up/down' is '0', it acts as in the round up mode and if it is '1', it acts as in the round down mode.

Example: Let us assume the number to be rounded towards negative infinity is -7.4 whose 32-bit binary floating point representation is "1 10000001 11011001100110011001101". In this example, the sign bit is '1' therefore it is inverted to '0' in order to utilize the round up hardware design to perform in the round down mode as shown in Fig. 2. Now, the process of rounding the number down remains same as that of rounding up. The exponent and mantissa field produced by applying rounding up with sign bit ‘0’ (which is same as rounding down with sign bit ‘1’) are “10000010” and “00000000000000000000000” respectively, while keeping the sign bit as it is. This produces the binary floating point number equivalent to 8.0. However, in case the number is positive; the example remains same as shown for the round towards zero.

3.5. Rounding to Even

In this rounding mode, the algorithm of determining the sign bit of rounded output is same as that of rounding towards Zero which is shown in Fig. 1(c) and (d) for the single and double precision respectively. Further, based on the definition of rounding to even mentioned in Table 1, the algorithm of producing exponent and mantissa field of rounded output is given below. In this case, the bit preceding to MBR (i.e. the bit at $(n+1)^{th}$ index of mantissa or at the second place in the mantissa field of interest from its LSB) also plays an important role in determining the condition of round to even because of the following reason. For the numbers in which the integer part is even, the bit preceding to MBR is always ‘0’. On the contrary when the integer part is odd then this bit is always ‘1’. Thus, we leveraged the capability of the bit preceding to MBR of distinguishing between the even and odd integer part to make the decision of rounding to even.

- First, the exponent field of input floating point number is concatenated with the mantissa field of interest as shown below: $B_{23-n}[30 - n : 0] = b^e[7 : 0] \| b^m[22 : n]$,
- If the MBR is ‘0′,
  - $c^e = B_{23-n}[30 - n : 23 - n]$ and $c^m = B_{23-n}[22 - n : 0] \| \{0\}^n$.
- If the MBR is ‘1′, then
  - if the bit preceding to MBR is ‘0′ then $c^e[7 : 0] = b^e[7 : 0]$, and $c^m = b^m[22 : n + 1] \| \{0\}^{n+1}$, for $2 \leq n \leq 21$, and $c^m = \{0\}^{n+1}$, for $n = 22$.
  - if the bit preceding to MBR is ‘1′ then the $B_{23-n}[30 - n : 0]$ is incremented by one and it is denoted using $B'_{23-n}[30 - n : 0]$. Now, $c^e = B'_{23-n}[30 - n : 23 - n]$ and $c^m = B'_{23-n}[22 - n : 0] \| \{0\}^n$.

For the double precision numbers, the steps are as follows.

- $B_{52-n}[62 - n : 0] = b^e[10 : 0] \| b^m[51 : n]$,
- If the MBR is ‘0′,
  - $c^e = B_{52-n}[62 - n : 52 - n]$ and $c^m = B_{52-n}[51 - n : 0] \| \{0\}^n$.
- If the MBR is ‘1′, then
  - if the bit preceding to MBR is ‘0′ then $c^e[10 : 0] = b^e[10 : 0]$, and $c^m = b^m[51 : n + 1] \| \{0\}^{n+1}$, for $2 \leq n \leq 50$, and $c^m = \{0\}^{n+1}$, for $n = 51$.
  - if the bit preceding to MBR is ‘1′ then the $B_{52-n}[62 - n : 0]$ is incremented by one and it is denoted using $B'_{52-n}[62 - n : 0]$. Now, $c^e = B'_{52-n}[62 - n : 52 - n]$ and $c^m = B'_{52-n}[51 - n : 0] \| \{0\}^n$.

Example: Let us illustrate the example for two different numbers viz. 2.5 and 3.5. In case of 2.5 whose 32-bit binary floating point representation is “0 10000000 01000000000000000000000”, the illustration is as follows. In this example, the corresponding MBR is $b^m[21]$ as per the Table 1 and hence the value of n is 21. Here, the MBR is ‘1’ and the bit preceding to MBR (i.e. at $(n+1)^{th}$ index) is ‘0’ therefore the exponent part remains same as that of input according to the algorithm. To produce the mantissa bits of the rounded output, firstly $b^m[22]$ is concatenated with "0000000000000000000000" and assigned to $c^m[22: 0]$. Finally, this produces the number "0 10000000 00000000000000000000000" which is equivalent to 2.0.

In case of 3.5 whose 32-bit binary floating point representation is "0 10000000 11000000000000000000000", the corresponding MBR is also $b^m[21]$ and hence the value of n is 21. Here, the MBR is '1' and the bit preceding to MBR is also '1' therefore, firstly, $b^e[7:0]$ is concatenated with $b^m[22:21]$ which results in $B_2[9:0]$. Secondly, $B_2[9:0]$ is incremented by one resulting into $B'_2[9:0]$. Further, $B'_2[9:2]$ is assigned to $c^e[7: 0]$ and $B'_2[1:0]||$"000000000000000000000" is assigned to $c^m[22: 0]$ to produce the exponent and mantissa bits of the rounded output. Finally, this produces the number "0 10000001 00000000000000000000000" which is equivalent to 4.0.

Proposed hardware design: The proposed RTL design of rounding 32-bit binary floating point to even is shown in Fig. A.4 in Appendix-A, where the blocks 1, 2 and 3 corresponds to the sign bit, exponent field and mantissa field generation logic. As shown, the logic of performing the rounding decisions based on the MBR and its preceding bit are realized using the multiplexers in the hardware.

To summarize, the hardware designs for rounding towards positive infinity and rounding towards negative infinity share logic as explained in section 3.4 and Fig. 2. The logic of rounding towards positive infinity (round up) can be reused to generate the logic of rounding towards negative infinity as shown in Fig. 2. Other proposed designs require separate logic. Note that in case of special input values, the rounding cannot be performed. Therefore, Zero is represented as Zero, NaN is represented as NaN and +/-INF is represented as +/-INF at the output in the proposed algorithms.

3.6. Proposed Integrated Multi-mode Rounding (IMR) and Hardware Design

As discussed earlier, the proposed algorithms for all the different rounding mode are based on the idea of MBR and MFI. Therefore, it becomes feasible to integrate the five different rounding algorithm and design an integrated multi-mode rounding (IMR) hardware. It helps achieve a good improvement in area (hardware resources) at the cost of slight increase in power and delay in comparison to the cumulative area, power and delay of individual hardware designs for the various modes. The proposed IMR hardware design is configured to a specific rounding mode based on the select bus S[2:0], where S[2:0]={000, 001,010,011,100} indicates {round to even, round to nearest integer, round up, round down, round towards zero} modes respectively. The algorithm of the IMR for 32-bit floating point numbers is described as follows:

- If S[2]= '1' then
    - $c^e[7 : 0]= b^e[7 : 0]$.
    - $c^m = b^m[22 : n+1] \,||\, \{0\}^{n+1}$, for $2 \le n \le 21$.
    - $c^m = \{0\}^{n+1}$, for $n = 22$.
- Else if S[2]= '0' then
    - if S[1]= '0' then
    - $B_{23-n}[30 - n : 0] = b^e[7 : 0] \,||b^m[22 : n]$,
    - $B'_{23-n}[30 - n : 0]= B_{23-n}[30 - n : 0]+1$
        - If the MBR is '0', $c^e= B_{23-n}[30 - n : 23 - n]$ and $c^m=B_{23-n}[22 - n : 0] \,||\, \{0\}^n$.

    - If the MBR is ‘1′, then
    - if S[0]= ‘0’ then
        - if the bit preceding to MBR is ‘1′ then
        - c^e=$B'_{23-n}[30 - n : 23 - n]$ and $c^m = B'_{23-n}[22 - n : 0] \,||\, \{0\}^n$.
        - if the bit preceding to MBR is ‘0′ then
        - $c^e[7 : 0]= b^e[7 : 0]$, and $c^m = b^m[22 : n + 1] \,||\, \{0\}^{n+1}$, for $2 \le n \le 21$, and $c^m = \{0\}^{n+1}$, for n = 22.
    - if S[0]= ‘1’ then
        - c^e=$B'_{23-n}[30 - n : 23 - n]$ and $c^m = B'_{23-n}[22 - n : 0] \,||\, \{0\}^n$.
- if (S[1:0]= ‘10’ and if b^s= ‘0’) or (S[1:0]= ‘11’ and if b^s= ‘1’) then
    - If the $b^m[n : 0]$ is 0, then $c^e[7 : 0]= b^e[7 : 0]$ and $c^m[22 : 0]=b^m[22 : 0]$.
    - Else, if the $b^m[n : 0]$ is not 0, then
        - $B_{22-n}[29 - n : 0] = b^e[7 : 0] \,||b^m[22 : n+1]$, for $2 \le n \le 21$
        - $B'_{22-n}[29 - n : 0]= B_{22-n}[29- n : 0]+1$
        - c^e=$B'_{22-n}[29 - n : 22 - n]$ and $c^m = B'_{22-n}[21 - n : 0] \,||\, \{0\}^{n+1}$.
- if (S[1:0]= ‘10’ and if b^s= ‘1’) or (S[1:0]= ‘11’ and if b^s= ‘0’) then
    - $c^e[7 : 0]= b^e[7 : 0]$.
    - $c^m = b^m[22 : n+1] \,||\, \{0\}^{n+1}$, for $2 \le n \le 21$.
    - $c^m = \{0\}^{n+1}$, for n = 22.

The corresponding RTL hardware design of IMR is shown in Fig. 3, where block-1 generates the sign bit, block-2 the exponent field and block-3 the mantissa field for the desired rounding mode. The select line S[2:0] is used to run the desired rounding mode. The proposed hardware design primarily uses multiplexing logic; hence it achieves a considerably good latency (fast execution). The reusing of common logic of the different rounding modes saves significant amount of resources. The impact on overall hardware resources, propagation delay and power due to the integration of different rounding modes is discussed in the results section 4.2.

3.7. Extension of Proposed Idea of Rounding for Quadruple (128-bit) Precision

The proposed idea of rounding binary floating point numbers to the integer can seamlessly be applied for quadruple precision. In this case, the mantissa part is 112 bits long, therefore the MBR (the n^{th} index of mantissa field) can vary in between [111 : 0] to realize the condition of rounding for the different range of integer parts of input number. In this case, the mantissa field of interest is represented by $b^m[111 : n]$. Similar to the single and double precision, the following concatenation operation is performed in case of the quadruple precision. The exponent field $b^e[14 : 0]$ is concatenated with the mantissa field of interest $b^m[111 : n]$ to generate $B_{112-n}[126\text{-}n : 0]$. Thus produced concatenated output can be partitioned into two parts based on the MBR in order to assign to the exponent and mantissa field to produce the desired rounded output, similar to the proposed idea for single and double precision.

3.8. Advantages of Proposed Rounding Hardware and its Application in Industrial Setting

Both the individual rounding hardware designs and the IMR hardware design have advantages. The IMR hardware design can be used in general purpose processor architecture to execute the ROUND instruction in various modes where a specific rounding mode can be selected using the select bus S[2:0]. Compared to the separately implemented rounding hardware for different modes, the integrated rounding hardware is more advantageous in terms of wide applicability and design metrics as the same hardware can execute different rounding modes (achieving savings in area). However, each individual rounding mode hardware can also be beneficial especially in case of application specific hardware design. If an application specific processor is customized to cater a specific function with a particular rounding mode then using only the desired individual rounding hardware can be useful. It will save area, power and latency compared to the integrated design. For example, an image compression processor hardware can be designed to use only 'round to nearest integer' rounding mode. It can also be used in machine learning accelerators of deep learning and k-means clustering. Further, rounding hardware design in 'round down' mode can be used in integer division hardware and executing the `floor' operation. Further, 'round up' design can be used in calculating the least integer greater than or equal to a given number and executing the 'ceil' operation. 'Round towards zero' design can be used for truncating a real number to its integer value. The round towards zero hardware design is useful in stochastic rounding hardware and machine learning accelerators. 'Rounding to even' mode avoids the statistical bias (important for addition and subtraction) because it rounds upward about half of the time and downward about half of the time. The applicability of proposed rounding hardware design in an industrial setting can be seen in terms of both application specific processor and general purpose processor/co-processor design. For example, the proposed rounding hardware can be integrated to an image compression processor using the following steps: (i) the computationally intensive task (such as DCT transformation and quantization) of the image compression algorithm is converted to RTL using high level synthesis process; thus obtaining RTL of the computational kernel (ii) next, the RTL of the computational kernel of image compression processor is integrated with the RTL of the proposed rounding hardware (iii) further, logic synthesis process is performed to obtained the gate level netlist of the image compression processor with rounding hardware.

In case of general purposes processor architecture, the proposed rounding hardware can be used as follows: (i) proposed IMR hardware is integrated with the floating point co-processor datapath (ii) the selection bits S[2:0] can be used as a portion of a `ROUND' instruction that can perform various rounding operations (iii) during execution, the `ROUND' instruction is decoded to perform rounding in the desired mode among the five modes, based on the particular value of S[2:0].

4. RESULTS AND ANALYSIS

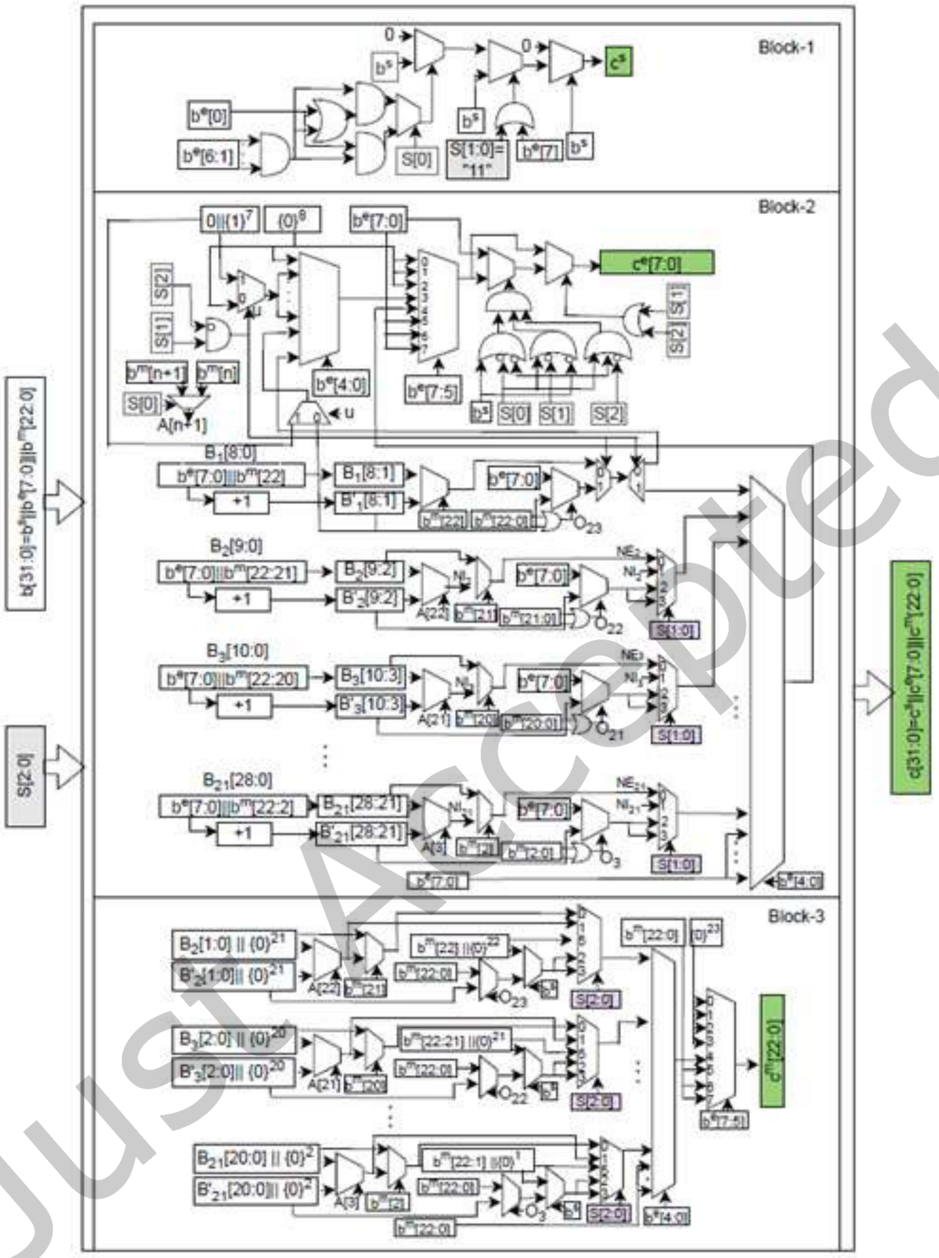
This section presents results of the proposed hardware designs of rounding 32-bit and 64-bit binary floating point numbers to integer and analyses in terms of the design metrics such as area, power and propagation delay. The results of proposed IMR hardware design have also been analyzed and compared with the design metrics of individual implemented rounding modes. Further, this paper also presents the results of integration of the proposed rounding hardware unit with the computing core of image/video compression processor. The proposed hardware designs of five different rounding to integer modes viz.

round to nearest, round towards zero, round up, round down and round to even have been implemented on Intel's FPGA of Cyclone-II family using the Quartus II tool. The results of the rounding for all the five modes have been functionally validated using the Quartus simulator. The area is assessed in terms of FPGA resource consumption, whereas delay and power of the proposed designs are assessed using the 'TimeQuest Timing Analyzer' and 'PowerPlay Power Analyzer' tool respectively of Quartus II. The proposed rounding hardware are applied for JPEG image compression application to assess the image quality. Further, we also compare the proposed work with the existing related work.

4.1. Area, Delay and Power Analysis of Individual Rounding Modes

The area analysis of the proposed rounding hardware designs in terms of the FPGA resource utilization is presented in Table 4. The FPGA resource usage has been measured in terms of total logic elements (cells) consumed by the proposed designs. As shown in the table, the total logic elements for the 4-input, 3-input and less than equal to 2-input look-up-tables (LUTs) for the rounding modes viz. round towards zero, round to nearest, round to even, round down and round up are around 0.15K, 1K, 1K, 1.2K and 1.3K respectively for the binary single precision floating point. However in case of binary double precision floating point, the count of logic cells are around 0.3K, 4.5K, 4.6K, 5K and 5K respectively which are higher than single precision as intuitive. Further, it is noteworthy that the proposed rounding hardware designs do not consume dedicated logic registers, DSP elements and embedded multipliers. Hence, the proposed designs are efficient in terms of resource utilization because their implementation purely rely on the logic cells or LUTs.

The delay of the proposed rounding hardware designs has been analyzed in terms of worst-case propagation delay for the various round to integer modes. Table 5 presents the worst-case propagation delay for proposed designs in case of both the single and double precision binary floating point. The worst-case propagation delay is defined as the longest delay between the edges of a signal propagating from an input port to an output port. For the different combinations of edges (such as rising and falling edge) of signals between the input and output port, this delay can vary as shown in the table. Where, 'RR', 'RF', 'FR' and 'FF' denote the highest delay measured from rising edge to rising edge, rising edge to falling edge, falling edge to rising edge and falling edge to falling edge respectively. The proposed designs do not require clock signal, however analyzing the worst-case propagation delay is significant in order to determine how fast the clock of the bigger entity (in which the proposed designs to be used) can work. Further, as evident from the Table 5, the delay for the various rounding modes varies from 13ns to 24ns in case of single precision and 15ns to 45ns in case of double precision. In all the case, the delay is on the order of tens and hence trivial. This implies that the proposed rounding hardware designs can offer good performance in case of the different rounding modes and the different precision floating point numbers.



Block-1
Block-2
Block-3
c^s
c^e[7:0]
c^m[22:0]
S[1:0]="11"
b[31:0]=b^s||b^e[7:0]||b^m[22:0]
S[2:0]
c[31:0]=c^s||c^e[7:0]||c^m[22:0]

Fig. 3. Proposed RTL of integrated multi-mode rounding (IMR) hardware design for rounding 32-bit floating point number to integer in five different rounding modes, where S[2:0]={000, 001,010,011,100} indicates {round to even, round to nearest integer, round up, round down, round towards zero} modes respectively

Table 4. FPGA Resource Utilization of Round to Integer Hardware Designs in Various Modes for 32-Bit and 64-Bit Binary Floating Point Numbers

Rounding Modes	Logic elements usage by number of LUT inputs							
	32-bit binary floating point				64-bit binary floating point			
	4 input functions	3 input functions	≤2 input functions	Total logic elements	4 input functions	3 input functions	≤2 input functions	Total logic elements
Round towards zero	86	37	5	128	240	90	46	376
Round to nearest	378	218	392	988	1513	1213	1789	4515
Round up	529	159	421	1109	2209	923	1945	5077
Round down	527	157	426	1110	2214	912	1953	5079
Round to even	549	60	406	1015	2647	150	1837	4634

Table 5. Propagation Delay of 32-Bit and 64-Bit Binary Floating Point Round to Integer Hardware Designs for the Various Modes Implemented in FPGA

Rounding Modes	Worst case propagation delay in ns							
	32-bit binary floating point				64-bit binary floating point			
	RR	RF	FR	FF	RR	RF	FR	FF
Round towards zero	13.45	13.66	13.66	13.45	15.31	15.81	15.81	15.31
Round to nearest	19.13	19.13	19.13	19.13	27.18	27.18	27.18	27.18
Round up	23.43	23.15	23.15	23.43	40.45	40.45	40.45	40.45
Round down	23.92	26.40	26.40	23.93	42.56	44.47	44.47	42.56
Round to even	19.27	19.27	19.27	19.27	26.13	26.13	26.13	26.13

Table 6. Power Dissipation of 32-Bit and 64-Bit Binary Floating Point Round to Integer Hardware Designs for the Various Modes Implemented in FPGA

Rounding Modes	Power dissipation in mW	
	32-bit binary floating point	64-bit binary floating point
Round towards zero	200.68	208.18
Round to nearest	200.72	208.07
Round up	200.71	208.73
Round down	200.89	208.24
Round to even	200.61	208.83

Table 7. Results of Integrated Multi-mode Rounding Hardware Design Implemented in FPGA

Hardware Design	LUTs	Propagation delay (ns)	Power (mW)
Integrated multimode rounding (IMR) hardware	1760	28.305	201.17
Individual implemented rounding hardware for different modes	4,350	26.400	200.89
Impact on metrics using proposed IMR hardware	×2.47 reduction	7.2% increase	0.1% increase

The power analysis of the proposed rounding hardware designs has been reported in Table 6. The reported power dissipation in milliwatt (mW) is mainly due to the core static thermal power dissipation and the input/output thermal power dissipation. Since the proposed designs run independent of clocks, the activity factor or toggle rate does not affect the power dissipation and hence the core dynamic power dissipation remains zero. Further, as evident from the Table 6, the differences in the power for the various rounding modes and also between the single and double precision floating point for a particular rounding mode are very nominal. This implies that the proposed rounding hardware designs are power efficient in case of the different rounding modes and the different precision floating point numbers.

4.2. Impact of Integrated Multi-mode Rounding (IMR) on Area, Power and Delay

The results of the proposed IMR hardware design, implemented in a Cyclone II FPGA platform, is shown in Table 7. We compare the resources (in terms of LUTs), propagation delay and power of the IMR hardware with that of individual implemented different rounding modes. As evident, the integration of various rounding modes in a single hardware reduces LUTs requirement around 2.5 times at the cost of slight increase in propagation delay (~7.2%) and power (~0.1%).

4.3. Case Study: Integration of Rounding Modules with a Compression Processor

The proposed rounding hardware designs for the various rounding modes have been integrated with the computing core of image/video compression processor. The computing core of a compression processor performs the DCT transformation on pixel values followed by the quantization. In the quantization process, the operation of rounding to integer is performed. We implemented the computing core of the compression processor on FPGA platform with the following cases: (i) without rounding hardware integrated (ii) with the proposed rounding hardware integrated in the various modes. Table 8 shows the results in terms of FPGA utilization of logic cells, dedicated logic registers and embedded multipliers for the different cases. Further, Table 9 presents the average resource overhead (in %) due to integration of the proposed rounding hardware with the compression processor. As shown, the integration of the proposed rounding hardware results in nominal resource overhead. The average resource overhead is observed to be less than 1% for all the five rounding modules.

Table 8. FPGA Resource Utilization of the Proposed Rounding Hardware Integrated with the Computing Kernel of a Compression Processor

Compression processor in multimedia applications		Logic elements in terms of LUTs	Dedicated logic registers	Embedded Multipliers (9-bit)
Without rounding hardware		36,194	8192	63
With rounding hardware integrated for various modes	Round to nearest integer	37,184	8192	63
	Round up	37,253	8192	63
	Round down	37,272	8192	63
	Round towards zero	36,255	8192	63
	Round to even	37,201	8192	63

Table 9. Estimated FPGA Resource Overhead of Integration of the Proposed Rounding Hardware with the Compression Processor

Rounding modes	Round to nearest integer	Round up/down	Round to even	Round towards zero
Average %overhead	0.90%	0.98%	0.92%	0.03%

JPEG Image compression using various rounding modes: The JPEG image compression relies on rounding to integers (with halfway cases rounded away from zero) during the quantization step. Hence, integrating the proposed hardware of rounding to the integer with the compression kernel of JPEG compression processor enhances the overall performance. We analyze the five different round to integer modes in JPEG image compression on a standard image 'lena'. We present the original image and the compressed images produced using the proposed floating point rounding hardware for quality factor (QF) of 50 in Fig. 4. The quality of compressed images is depicted in terms of peak signal to noise ratio (PSNR) in Table 10. We have evaluated this metrics using *MATLAB* 2021a running on system with 8 GB of RAM. As shown in Table 10, the PSNR for round to nearest, round to zero and round to even modes are almost same, resulting into compressed images of same quality. However, round up and round down modes introduce some compression artifacts (also known as `JPEG dimples') in the compressed images as shown in Fig. 4 and results into relatively lower PSNR.

Table 10. PSNR of Compressed Images for Different Rounding Modes

Rounding modes	Round to nearest Integer [26]	Round up	Round down	Round towards zero	Round to even
PSNR	18.3632	15.5849	15.5914	18.5482	18.3630

A co-processor, in which the rounding units are integrated, can select the specific rounding unit for the execution using multiplexers (Muxes). The five rounding modes can be driven using a 3-bit select line S[2:0] of the Muxes. A specific combination of the select line will choose the corresponding rounding unit for the execution.

4.4. Comparison with Existing Work

In the literature, the related works apply rounding of floating point numbers to round the result of the specific floating point arithmetic such as floating point addition or multiplication etc. For example, Even and Seidel [30] present floating point rounding customized for the multiplication only. Similarly, Jaberipur *et al.* [31] perform rounding to round the result of the floating point addition. The barrel shifter is used in the conventional floating point adder for the alignment on the mantissa of the smallest floating point number and normalizing the added mantissa [32]. However, the proposed approach is a generic one which can be applied to round any input number (or result of any arithmetic operation) to corresponding integer in the desired rounding mode irrespective of the arithmetic operation that generated the result. To the best of our knowledge, binary floating point rounding hardware designs for performing rounding to integer (irrespective of the arithmetic operation type) in various modes is not available in the literature. A binary floating point rounding hardware for rounding to nearest integer mode (with halfway cases rounded away from zero) was presented in [26]. However, it [26] does not present rounding hardware for other rounding modes viz. round to nearest even, round up, round down and round towards zero unlike the proposed approach. Moreover, it [26] does not analyze the impact of aforementioned four rounding modes on the JPEG image compression. However, the proposed approach also presents this analysis as shown in Fig. 4 and Table 10 compared to [26]. A comparison among various floating point rounding algorithms is presented in Table 11. It is to be noted that the proposed work differs from [25] in terms of the base of the floating point representation. This paper discusses the rounding for binary floating point unlike [25] which is based on decimal floating point. The proposed approach is the only work in the literature which presents detailed algorithm and hardware for five different rounding modes for the binary floating point arithmetic.

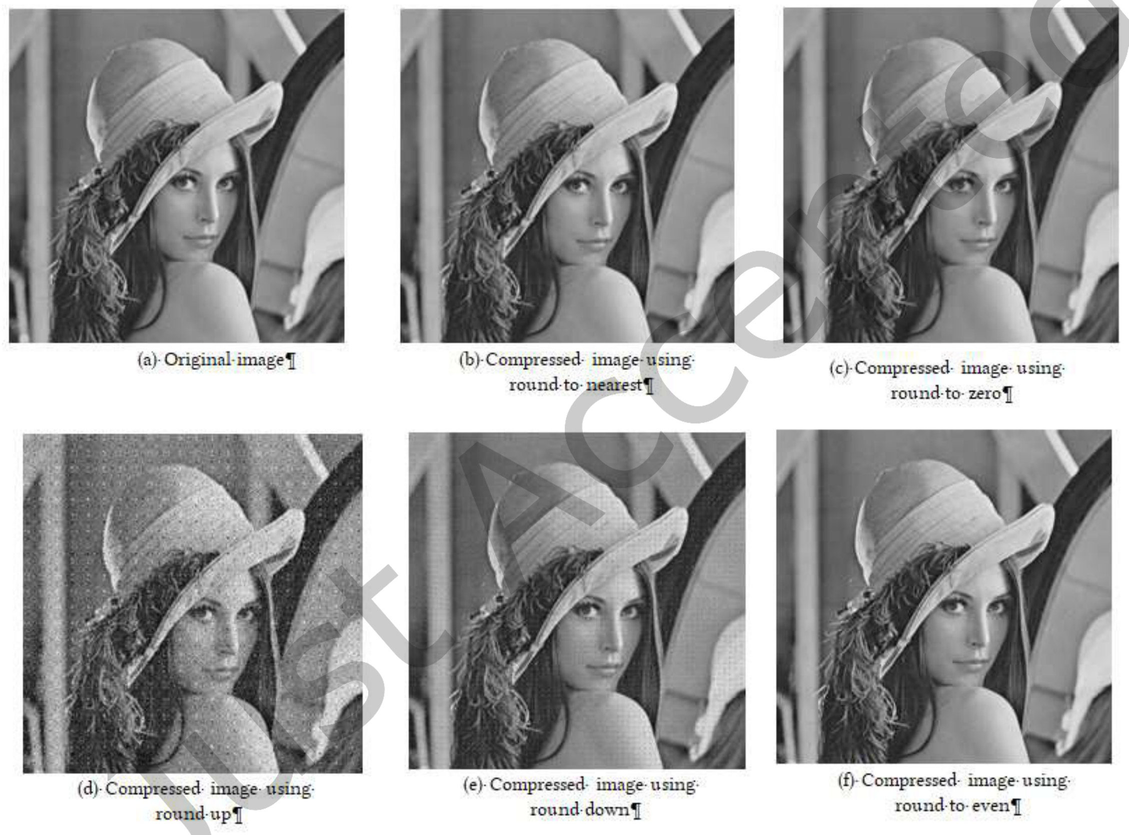

(a) Original image¶
(b) Compressed image using round to nearest¶
(c) Compressed image using round to zero¶
(d) Compressed image using round up¶
(e) Compressed image using round down¶
(f) Compressed image using round to even¶

Fig. 4. (a) Original and (b) to (f) compressed images with QF=50 after applying different rounding modes

Table 11. Comparison of the proposed work with the related works [21], [25], [26], [30], [31], where (–) denotes 'not available'

Approaches	Round to nearest Integer (break ties with round away)	Round up	Round down	Round towards zero	Round to nearest even	Rounding base		Independence of rounding on arithmetic operation type such as *, + and / etc.	Hardware design
						Binary	Deci-mal		
Proposed	✓	✓	✓	✓	✓	✓	×	✓	✓
[26]	✓	×	×	×	×	✓	×	✓	✓
[25]	✓	✓	✓	✓	✓	×	✓	✓	✓
[30]	×	✓	✓	✓	✓	✓	×	×	✓
[31]	×	×	×	×	✓	✓	×	×	✓
[21]	×	✓	✓	✓	✓	✓	×	✓	--

A quantitative comparison of the proposed integrated rounding hardware design for the binary floating point with respect to the rounding hardware design for the decimal floating point [25] is shown in Table 12. The design metrics such as area, delay and static power of the proposed rounding hardware is calculated using a 15nm open cell library [33]. The design metrics are achieved to be lower because reusing of common logic with multiplexers saves resources (in turn saving area and power) and making quick rounding decisions using MBR enables the fast execution. Further, we also compare with [26] which only implements one rounding mode (i.e. round to nearest integer). On the contrary, we implement all the five rounding modes however incurring only about 1.2 times higher propagation delay and 3 times higher area and power than [26] as shown in Table 12. The quantitative comparison with other related works such as [30] and [31] may not be relevant as their rounding designs are customized to a particular arithmetic operation. On the contrary, the proposed rounding hardware is generalized as its applicability is not dependent on a particular operation type only.

Table 12. Analysis of Design Metrics of the Proposed IMR Hardware with respect to the Related Works [25], [26], where (--) indicates `not available'

Approaches	Technology Node	Area (μm^2)	Propagation delay (ps)	Static Power (μW)	Remarks about implementation
Proposed IMR hardware	15 nm	5327.14	716.82	219.49	All five rounding modes in binary FP
[25]	65 nm	2,09,713.0	2240	--	All five rounding modes in decimal FP
[26]	15 nm	1784.41	602.07	62.33	Only Round to nearest integer mode in binary FP

5. CONCLUSION

The need of high performance and energy efficiency entails employing the hardware accelerators in the modern computing systems. Huge computational and data hungry applications such as image compression, video compression and machine learning etc. require rounding to integer in their processing. To aid the hardware acceleration of multimedia and machine learning applications, performing rounding to integer through a dedicated hardware is important. Hence, in this paper, we propose algorithms and hardware designs for efficient rounding of binary floating point numbers to integer in the following rounding modes: round towards zero, round to nearest, round up, round down and round to even. We have shown that how the proposed MBR and the mantissa field of interest help achieve an efficient rounding to integer operation in the various modes, for both the single and double precision floating point. The proposed designs have been implemented on FPGA platform to analyze the area, power and delay. The results highlight that the proposed designs are efficient in terms of area, power and delay. Thereby, they are amenable to be integrated to aid the hardware acceleration in the modern high performance and energy efficient computing systems. We have also shown the integration of rounding hardware with the computing core of a compression processor used in multimedia applications. The integration overhead is achieved to be negligible.

ACKNOWLEDGMENTS

This work is supported by Devi Ahilya Vishwavidyalaya Indore, India under the grant number Dev-I/Seed Money/2024/820.

REFERENCES

[1] Altera, “Adding Hardware Accelerators to Reduce Power in Embedded Systems," WP-01112-1.0, 2009.

[2] A. Sengupta, M. Rathor, “Securing Hardware Accelerators for CE Systems Using Biometric Fingerprinting”, IEEE Transactions on Very Large Scale Integration (VLSI), vol. 28, no. 9, pp. 1979-1992, 2020.

[3] H. Park, S. Kim, “Chapter Three - Hardware accelerator systems for artificial intelligence and machine learning", Editor(s): Shiho Kim, Ganesh Chandra Deka, Advances in Computers, Elsevier, Volume 122, 2021, Pages 51-95.

[4] J. L. Nunez-Yanez and V. A. Chouliaras, “Design and implementation of a high-performance and silicon efficient arithmetic coding accelerator for the H.264 advanced video codec," IEEE International Conference on Application-Specific Systems, Architecture Processors (ASAP’05), Samos, Greece, 2005, pp. 411-416.

[5] Y.Wang, Y.Wang, H. Li and X. Li, “An Efficient Deep Learning Accelerator Architecture for Compressed Video Analysis," IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 41, no. 9, pp. 2808-2820, Sept. 2022.

[6] ANSI/IEEE 754-1985, “Standard for Binary Floating-Point Arithmetic,” 1985.

[7] D. Piso and J. D. Bruguera, “Variable Latency Goldschmidt Algorithm Based on a New Rounding Method and a Remainder Estimate," IEEE Transactions on Computers, vol. 60, no. 11, pp. 1535-1546, 2011.

[8] T. Richter, “Evaluation of floating point image compression,” in Proc. ICIP, 2009, pp. 1909–1912.

[9] R. Mukherjee, K. Debattista, T.-B. Rogers, M. Bessa, and A. Chalmers, “Uniform color space-based high dynamic range video compression,” IEEE Transactions on Circuits and Systems for Video Technology, vol. 29, no. 7, pp. 2055–2066, 2019.

[10] R. Mantiuk, K. Myszkowski, H.-P. Seidel, B. Rogowitz, T. Pappas, and S. Daly, “Lossy compression of high dynamic range images and video -art. no. 60570v,” Human Vision and Electronic Imaging XI, SPIE, vol. 6057, 02 2006.

[11] K. A. M Segal, “The opengl graphics system: A specification,” available at http://www.opengl.org/registry/ doc/glspec30.20080811.pdf, 2008.

[12] Roger B. Dannenberg, “Danger in Floating-Point-to-Integer Conversion,” (letter to editor), Computer Music Journal, vol. 26, no. 2, Summer 2002, p4.

[13] S. Baum, and L. E. Trotter Jr., “Integer Rounding for Polymatroid and Branching Optimization Problems,” SIAM Journal on Algebraic Discrete Methods,vol. 2, no. 4, pp. 416-425, 1981.

[14] P. Teunissen, “Success probability of integer GPS ambiguity rounding and bootstrapping,” Journal of Geodesy, Springer, vol. 72, pp. 606-612, 1998.

[15] R. Cont and M. Heidari, “Optimal rounding under integer constraints,” ArXiv, vol. abs/1501.00014, 2014.

[16] K. Cabeen and P. Gent, “Image Compression and the Discrete Cosine Transform," college of the Redwoods, Tech. Rep, pp. 1-11, 1998.

[17] A. Sengupta, D. Roy, S. P. Mohanty, and P. Corcoran, “Low-cost obfuscated jpeg codec ip core for secure ce hardware,” IEEE Transactions on Consumer Electronics, vol. 64, no. 3, pp. 365–374, 2018.

[18] A. Soetewey, “The complete guide to clustering analysis: k-means and hierarchical clustering by hand and in R," Stats and R, 2013.

[19] M. Zhu, Q. Zhang, Y. Zhang, T. Shen and B. Zhang, “Investigation of Rounding Algorithms Combining Data Distribution Characteristics," IEEE 17th Conference on Industrial Electronics and Applications (ICIEA), Chengdu, China, 2022, pp. 1668-1672.

[20] M. Wang, S. Rasoulinezhad, P. H. W. Leong and H. K. . -H. So, “NITI: Training Integer Neural Networks Using Integer-Only Arithmetic,” in IEEE Transactions on Parallel and Distributed Systems, vol. 33, no. 11, pp. 3249-3261, 1 Nov. 2022.

[21] Intel® 64 and IA-32 Architectures Software Developer’s Manual, Intel, Order Number: 325462-082US, December 2023.

[22] ARM Architecture Reference Manual ARMv7-A and ARMv7-R edition, ARM DDI 0406C ID112311, 2011.

[23] M. Fasi and M. Mikaitis, “Algorithms for Stochastically Rounded Elementary Arithmetic Operations in IEEE 754 Floating-Point Arithmetic," IEEE Transactions on Emerging Topics in Computing, vol. 9, no. 3, pp. 1451-1466, 1 July-Sept. 2021.

[24] M. Mikaitis, “Stochastic Rounding: Algorithms and Hardware Accelerator," International Joint Conference on Neural Networks (IJCNN), Shenzhen, China, 2021, pp. 1-6.

[25] S. Tsen, S. Gonzalez Navarro, M. J. Schulte and K. Compton, “Hardware designs for binary integer decimal based rounding,” IEEE Transactions on Computers, vol. 60, no. 5, pp. 614-627, 2011.

[26] M. Rathor, V. Mishra and U. Chatterjee, “Aiding to Multimedia Accelerators: A Hardware Design for Efficient Rounding of Binary Floating Point Numbers," Design, Automation & Test in Europe Conference & Exhibition (DATE), Antwerp, Belgium, 2023, pp. 1-6.

[27] P. Kornerup and J.-M. Muller, “RN-Coding of Numbers: Definition and Some Properties,” Proc. Int’l Meeting on Automated Compliance Systems (IMACS ’05), July 2005.

[28] P. Kornerup, J. -M. Muller and A. Panhaleux, “Performing Arithmetic Operations on Round-to-Nearest Representations," IEEE Transactions on Computers, vol. 60, no. 2, pp. 282-291, Feb. 2011.

[29] J. Hormigo and J. Villalba, “New Formats for Computing with Real-Numbers under Round-to-Nearest," IEEE Transactions on Computers, vol. 65, no. 7, pp. 2158-2168, 1 July 2016.

[30] G. Even and P.M. Seidel, "A comparison of three rounding algorithms for IEEE floating-point multiplication," IEEE Transactions on Computers, vol. 49, no. 7, pp. 638-650, July 2000, doi: 10.1109/12.863033.

[31] G. Jaberipur, B. Parhami and S. Gorgin, "Redundant-Digit Floating-Point Addition Scheme Based on a Stored Rounding Value," IEEE Transactions on Computers, vol. 59, no. 5, pp. 694-706, May 2010, doi: 10.1109/TC.2009.152.

[32] S. Pitchai, S. Pitchai, "Area-latency efficient floating point adder using interleaved alignment and normalization,", Microprocessors and Microsystems, vol. 99, 2023, https://doi.org/10.1016/j.micpro.2023.104842.

[33] “Open cell library 15 nm . [online],” Available: https://si2.org/open-celllibrary/, last accessed on, 2020.

APPENDIX-A

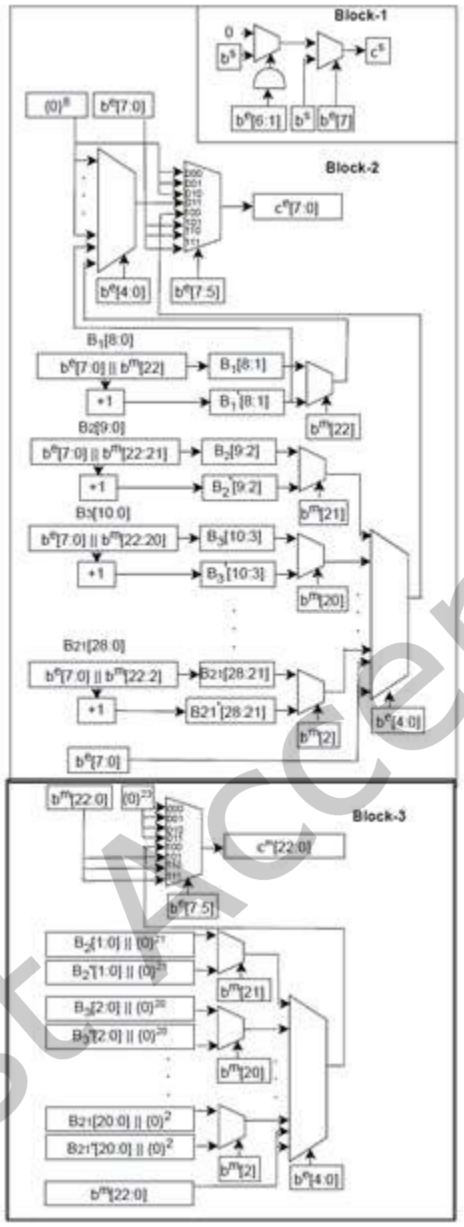


Fig. A.1. RTL hardware module for rounding 32-bit binary FP number to nearest integer

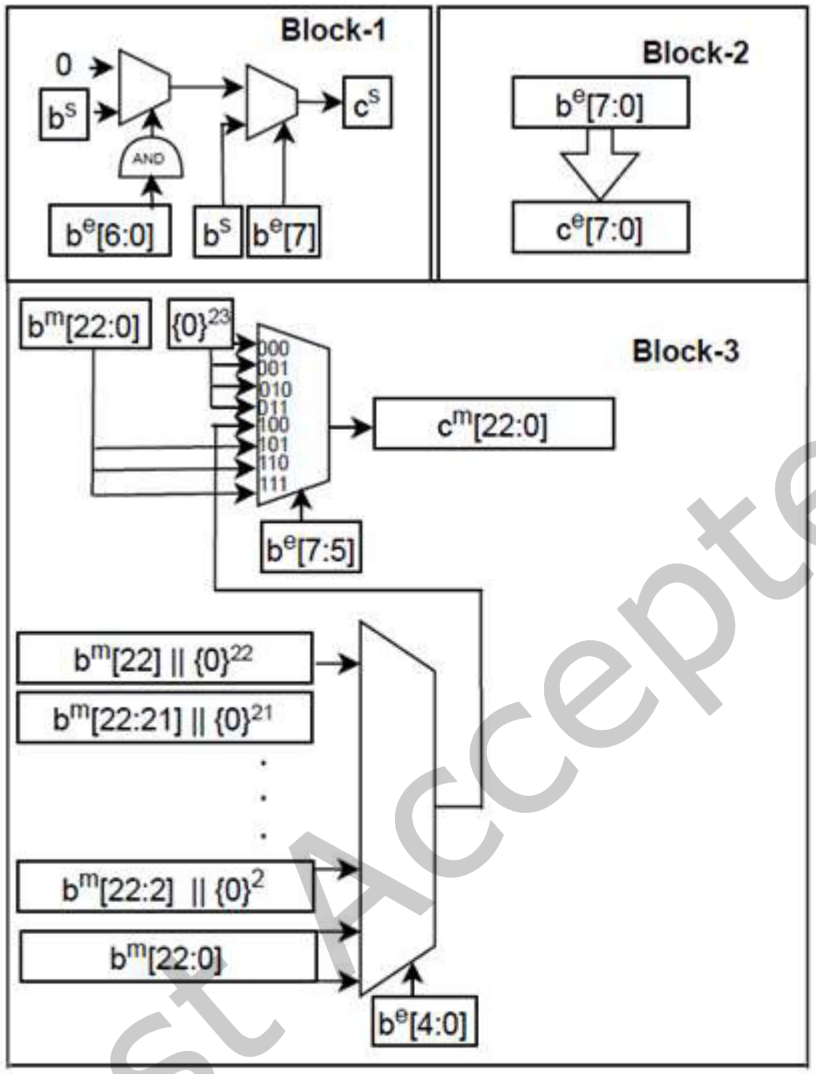


Fig. A.2. RTL hardware design for rounding 32-bit floating point number using round towards zero mode

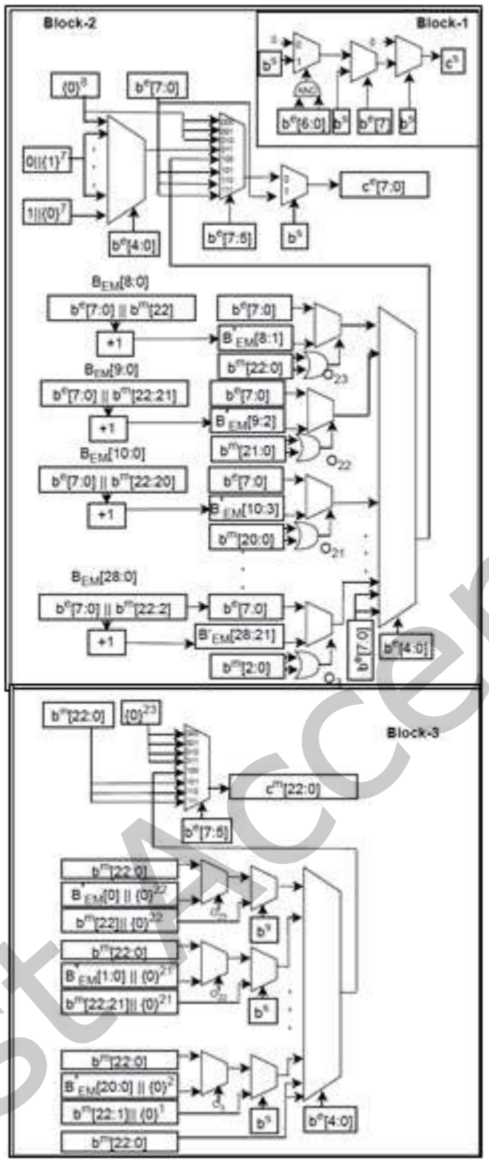


Fig. A.3. RTL hardware design for rounding 32-bit floating point number using round up mode

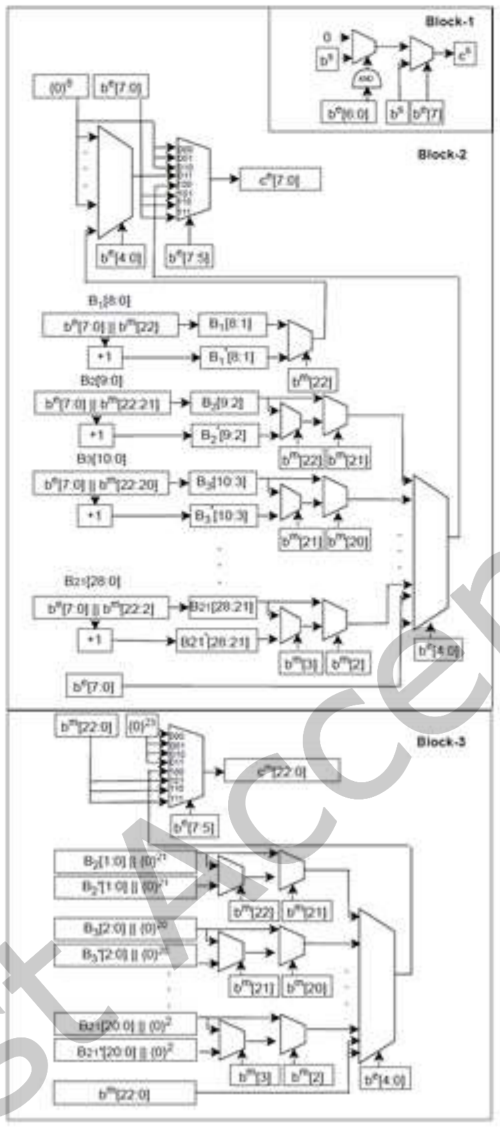


Fig. A.4. RTL hardware design for rounding 32-bit floating point number using round to nearest even mode