Aiding to Multimedia Accelerators: A Hardware Design for Efficient Rounding of Binary Floating Point Numbers

Mahendra Rathor
Department of CSE
Indian Institute of Technology Kanpur
rmahendra@cse.iitk.ac.in

Vishesh Mishra
Department of CSE
Indian Institute of Technology Kanpur
vishesh@cse.iitk.ac.in

Urbi Chatterjee
Department of CSE
Indian Institute of Technology Kanpur
urbic@cse.iitk.ac.in

***Abstract*—Hardware accelerators for multimedia applications such as JPEG image compression and video compression are quite popular due to their capability of enhancing overall performance and system throughput. The core of essentially all lossy compression techniques is the quantization process. In the quantization process, rounding is performed to obtain integer values for the compressed images and video frames. The recent studies in the photo forensic research has revealed that the direct rounding e.g. round up or round down of floating point numbers results into some compression artifacts such as 'JPEG dimples'. Therefore in the compression process, performing rounding to the nearest integer value is important especially for High Dynamic Range (HDR) photography and videography. Since rounding to the nearest integer is a data-intensive process, hence its realization as a dedicated hardware is imperative to enhance overall performance. This paper presents a novel high performance hardware architecture for performing rounding of binary floating point numbers to the nearest integer. Additionally, an optimized version of the basic hardware design is also proposed. The proposed optimized version provides 6.7% reduction in area and 7.4% reduction in power consumption in comparison to the proposed basic architecture. Furthermore, the integration of the proposed floating point rounding hardware with the design flow of the computing kernel of the compression processor is also discussed in the paper. The proposed rounding hardware architecture and the integrated design with the computing kernel of compression process have been implemented on an Intel FPGA. The average resource overhead due to this integration is reported to be less than 1%.**

***Index Terms*—Binary Floating point, rounding to nearest integer, hardware acceleration.**

I. Introduction

Algorithms used in image and video processing are rather computationally expensive than the trivial computer applications that could be easily catered through system ALUs. Catering such applications often require a dedicated hardware commonly known as hardware accelerator for enhancing the system performance. Therefore, the high performance requirements entail the integrating hardware accelerators in a system-on-chip (SoC) employed in electronic systems [1].

Nowadays, multi-exposure High Dynamic Range (HDR) capture is becoming popular in photography and videography. The multi-exposure HDR capture technique allows to take high dynamic range images by capturing and then combining different exposures of the same subject matter. To support HDR in photography and videography, it is desirable to represent the sample values in floating point (FP) numbers [2]. For example, the EXR format employed in post-processing of cinematic material uses a 16-bit FP number representation that can cover dynamic ranges up to 10.7 magnitudes in luminance [2]. Further, modern shading units in GPUs represent pixel data in floating point [3]. Thereby, the application of FP numbers in photography and videography is well acknowledged.

Recently, compression of HDR photography [2] and HDR videography [4] have gained attention. The lossy compression of HDR images and videos is vital to satisfy the less storage and fast transmission requirements [5]. Typically, a quantization step in the compression techniques consists of truncation followed by rounding operation. However, recently, photo forensic of JPEG images [6] revealed that the compression artifacts may incur if the appropriate FP rounding mode is not chosen. For example, using the round up or round down modes incur a single darker or brighter pixel in 8×8 pixel blocks, which is termed as a dimple. This kind of compression artifact is also prevalent in commercial cameras [6]. Therefore, rounding of FP numbers to the nearest integer is desirable in the compression process.

A typical compression kernel performs DCT transformation and quantization steps. In the quantization stage, FP multiplication/division is followed by rounding to the closest integer. Since rounding is a data-intensive task which is performed on each quantized pixel, the hardware realization of FP rounding to the closest integer is important to achieve enhanced performance. Further, area and power consumption are also critical parameters of a hardware accelerator. Hence, designing an area-power efficient and low-latency architecture generalized for rounding of FP numbers to the nearest integer is vital.

In the literature, Tsen et al. [7] have proposed rounding hardware design for decimal FP arithmetic for the conventional rounding modes. However, it does not present the hardware for binary FP numbers rounding unlike the proposed approach. Moreover, the existing compression framework is amenable to the binary FP arithmetic as it supports a larger range of

978-3-9819263-7-8/DATE23/© 2023 EDAA

numbers in contrast to the decimal FP arithmetic. *To the best of our knowledge*, the hardware acceleration of binary FP numbers rounding to the nearest integer value has not been explicitly discussed so far. Therefore, this paper caters two main issues: (1) a low latency and efficient hardware architecture design for the rounding process of FP numbers to the nearest integer value, (2) integration of rounding hardware module with the design process of a compression kernel, to aid multimedia accelerators for achieving enhanced performance.

In summary, the main contributions of this paper are highlighted as follows:

- This paper presents an algorithmic flow of rounding a given FP number to the nearest integer value. The proposed idea of rounding is extendable to rounding of single and double precision FP numbers as well, without making substantial changes in the proposed algorithm.
- This paper also presents a basic low latency register transfer level (RTL) circuit design for rounding the FP number to the nearest integer value.
- Next, we present an optimized architecture to further reduce area and power consumption. The rounding hardware has been implemented and functionally validated on an FPGA.
- Finally, we present an integration flow of the proposed scheme with the computing core of compression process used in image/video compression applications. The integrated design is also implemented on a FPGA platform to analyze the resource overhead which is found to be less than 1%.

The rest of the paper is organized as follows. Section II presents the proposed FP number rounding algorithm, its hardware realization and the integration with the compression kernel design flow. Further, Section III presents experimental results and analysis and finally section IV concludes the paper.

II. Proposed Floating Point Number Rounding Algorithm and its Hardware Realization

This section first discusses the basic intuition behind the proposed approach of rounding a FP number to the nearest integer value. Second, we present the flow of the proposed rounding algorithm with a demonstrating example. Third, we present a hardware architecture for the proposed FP rounding process and a potential interconnect hardware optimization into it. Fig. 1 shows a generic block diagram of the rounding algorithm for the 16-bit binary FP numbers, where b^s, $b^e[4:0]$ and $b^m[9:0]$ denote the sign, exponent and mantissa part of the input FP number; c^s, $c^e[4:0]$ and $c^m[9:0]$ denote the sign, exponent and mantissa part of the output FP number.

A. The Basic Intuition of the Proposed Rounding

Let us assume a floating point number to be represented as $[D.F]$ where D represents the integer part and F the fractional part. As discussed in Section I, for rounding to the nearest integer, if $0 \leq F < 5$, we choose the number to be rounded down and if $F \geq 5$, then it is rounded up. The basic intuition of our proposed FP rounding operation to the nearest integer leverages the dynamic range and precision feature of the FP

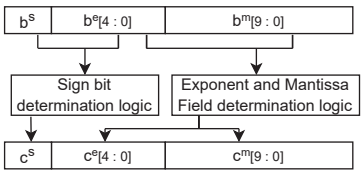

Fig. 1. Generic Logic blocks of proposed FP rounding to nearest integer

TABLE I
Exponent and mantissa bits of input FP number used for rounding

The range of integer part of input FP number to be rounded off (total values= 2^{9-n})	$b^e[4:3]$	$b^e[2:0]$	MBI (b^m_n bit of mantissa field)
1	01	111	b^m_9
2 to 3	10	000	b^m_8
4 to 7	10	001	b^m_7
8 to 15	10	010	b^m_6
...	...	...	...
128 to 255	10	110	b^m_2

numbers, which is discussed as follows.

Leveraging dynamic range feature: Due to the dynamic range feature, the range of the integer part of input numbers exponentially increases with the single increment in the corresponding exponent field of its binary FP representation as shown in Table I. For example, the exponent field $b^e[4:0] =$ "01111" in a 16-bit FP corresponds to only one integer value i.e. '1' whereas $b^e[4:0] =$ "10110" corresponds to 128 different integer values. Therefore, in our approach, the condition of rounding for a range of the numbers is determined using a particular value of the exponent field. It is to be noted that the $b^e[4:0]$ in between the range "00000" and "01110" corresponds to the values in between 0 and 1.

Leveraging dynamic precision feature: Due to the dynamic precision feature of FP numbers, small numbers can be represented with higher precision in comparison to the larger numbers. For example, the 16-bit FP representation of 1.0 and 1.5 are "0 01111 **0000000000**" and "0 01111 **1000000000**" respectively. Here, the mantissa part can accommodate total 511 different numbers in-between 1.0 and 1.5. However, in case of larger numbers, the precision is relatively less. For example, the 16-bit FP representation of 255.0 and 255.5 are "0 10110 1111111**000**" and "0 10110 1111111**100**" respectively. Here, the mantissa part can accommodate only 3 different numbers in-between 255.0 and 255.5. This observation indicates that as the numbers become larger, the effective part of mantissa (highlighted in the bold) shrinks toward the least significant bit (LSB).

In this effective part of mantissa, *we propose a mantissa bit of interest (MBI) to determine the condition of rounding to the nearest integer. The MBI is defined as that bit of mantissa which undergoes low-to-high transition when the fractional part 'F' becomes equal to* 5. Table I shows the MBI for the different range of integer part of the FP numbers. As we move from smaller to larger range of numbers, the respective MBI propagates from MSB to LSB. The MBI is the 9^{th} bit of mantissa in case of the integer part is 1, whereas it is the 2^{nd}

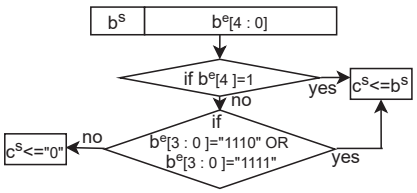

Fig. 2. Algorithmic flow of sign bit generation of rounded FP output

bit for the range of integers from 128 to 255.

We summarize our basic idea of rounding as follows: (i) with the help of the exponent field value, we determine the range of integer part for which the rounding condition is applied, (ii) with the help of MBI, we determine the condition of rounding to the nearest integer value for a particular range of integer part. In the following subsections, we discuss the proposed algorithm for the FP numbers rounding to the nearest integer value.

B. Algorithmic Flow for Generating Rounded Floating Point Output

The proposed idea of rounding FP numbers is divided in two parts: (i) determination of the sign bit of the rounded output, (ii) determination of the exponent and mantissa fields.

First, the algorithmic flow chart of *generating the sign bit (c^s)* of rounded FP output is shown in Fig. 2. As shown, the c^s bit can be determined using b^s and $b^e[4:0]$. Since if a FP number q is in between the range $-0.5 < q < 0.5$, it would be rounded to 0. Hence 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.

Next, as discussed in section II.A, the condition for the rounding can be determined by observing the exponent field and the MBI of the input FP number. For the MBI being at n^{th} index of mantissa field (where n varies from 9 down to 0), it can round off the values for total 2^{9-n} different integer values (range) as shown in Table I. In general, the following steps are performed for the rounding:

- First, $b^e[4:0]$ is concatenated with $b^m[9:n]$, where n is the index of mantissa acting as MBI. Let us define it as: $B_{10-n}[14-n:0] = b^e[4:0] \parallel b^m[9:n]$
- If the MBI is "0", $c^e = B_{10-n}[14-n : 10-n]$ and $c^m = B_{10-n}[9-n:0] \parallel \{0\}^n$.
- If the MBI is "1", the $B_{10-n}[14-n:0]$ is incremented by one and it is denoted using $B'_{10-n}[14-n:0]$. And $c^e = B'_{10-n}[14-n:10-n]$ and $c^m = B'_{10-n}[9-n:0] \parallel \{0\}^n$.

Special Case: For the numbers that in the range $0 \le q < 1$, the exponent part is not constant but varies from "00000" to "01110". Therefore, this case is handled as follows. (i) for $0 \le q < 0.5$, the output is rounded to 0, and (ii) for $0.5 \le q < 1$, the $c^e[4:0]$ is incremented by one and $c^m[9:0]$ is made 0 to determine the rounded FP output.

Example: Let us assume an input number is 2.5 whose binary 16-bit FP representation is "0 10000 0100000000". In this case, $b^e[4:3]$ = "10" and $b^e[2:0]$ = "000". Therefore, the corresponding MBI is b^m_8 as per the Table I and the value of n is 8. Further, according to the proposed algorithm, firstly $b^e[4:0]$ are concatenated with $b^m[9:8]$ which results in $B_2[6:0]$. Secondly, $n = 8^{th}$ bit of mantissa part is checked according to the algorithm. Since it is '1', $B_2[6:0]$ is incremented by one forming $B'_2[6:0]$. In order to generate the exponent and mantissa bits of the rounded output, $B'_2[6:2]$ is assigned to $c^e[4:0]$ and $B'_2[1:0] \parallel$ "00000000" is assigned to $c^m[9:0]$. Finally, this forms the number "0 10000 1000000000" which is equivalent to 3.0.

C. Proposed Architecture of FP Rounding Hardware

The proposed basic architecture of 16-bit FP rounding hardware is shown in Fig. 3. The blocks 1, 2 and 3 show sign bit, exponent part and mantissa part generation logic respectively and the hardware works as follows:

- It first divides $[14-n:0]$ into $[14-n:10-n]$ and $[9-n:0]$ for the exponent and mantissa part respectively of the output FP number.
- Now, based on the MBI, it chooses either $B_{10-n}[14-n : 10-n]$ or $B'_{10-n}[14-n:10-n]$ for the exponent part. Similarly, it chooses either $B_{10-n}[9-n:0]$ or $B'_{10-n}[9-n:0]$ for the mantissa part.

However, we explore an interconnect hardware optimization in order to achieve area and power efficacy. We perform the optimization by reordering the operations as follows:

- First it makes a decision of choosing either $B_{10-n}[14-n:0]$ or $B'_{10-n}[14-n:0]$ based on the MBI.
- It then divides the $[14-n:0]$ into $[14-n:10-n]$ and $[9-n:0]$ for the exponent and mantissa part respectively.

This re-ordering based optimization eliminates the need of some additional interconnect hardware (Muxes) in the mantissa field determination architecture. An excerpt of thus obtained optimized architecture is shown in Fig. 4.

D. Extension of Proposed Idea of Rounding for Single and Double Precision Floating Point

The proposed idea of rounding FP numbers to the nearest integer is also applicable for single (32-bit) and double precision (64-bit) values. In case of single and double precision, the mantissa part is 23 and 52 bits long, hence the MBI (the n^{th} index of mantissa field) is varied for $[22:0]$ and $[51:0]$ respectively to apply the condition of rounding. Similar to the half precision, the following concatenation operation is performed in case of single precision and double precision.

- In case of single precision, the exponent field $b^e[7:0]$ of input FP number is concatenated with the mantissa bits $b^m[22:n]$ to generate $B_{23-n}[30-n:0]$.
- In case of double precision, the exponent field $b^e[10:0]$ is concatenated with the mantissa bits $b^m[51:n]$ to generate $B_{52-n}[62-n:0]$.

Thus generated concatenated output is divided into two parts based on the MBI to assign to the exponent and mantissa filed of output number, as similar to the proposed idea for half precision.

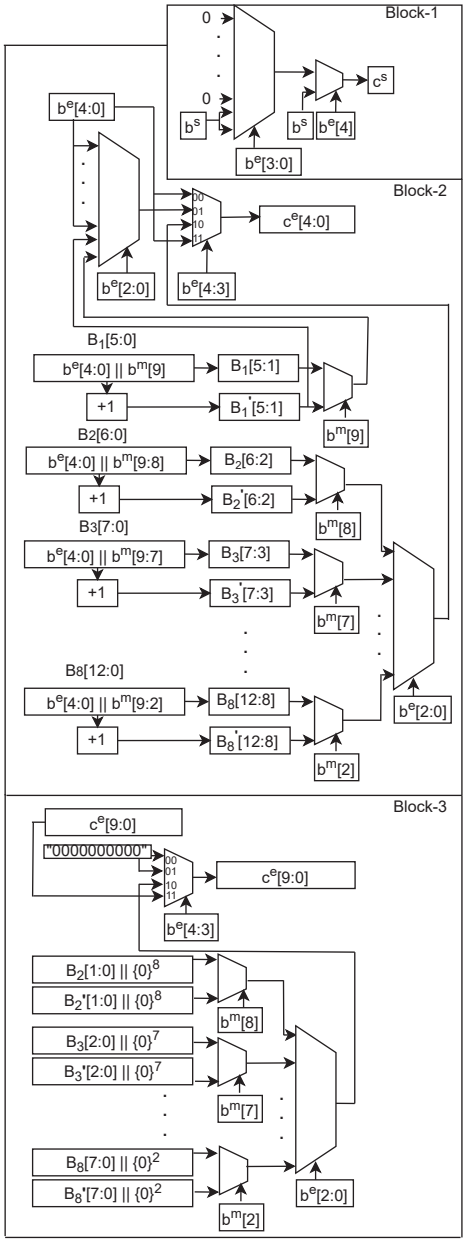

Fig. 3. Proposed basic architecture of 16-bit FP rounding hardware

E. Integration with the Computing Core of Compression Processor

This section discuses the integration flow of the proposed rounding hardware with the design flow of computing core of compression processor used in multimedia applications such as image and video coding. In these applications, the discrete cosine transform (DCT) computation and quantization are the computing core of the compression processor [8] and also computationally intensive processes. In the design process of the hardware of DCT computation and quantization, their algorithmic description is represented in terms of mathematical transfer function. Next, the algorithmic description is converted into the corresponding RTL description using the behavioral synthesis process. Further, the RTL of the proposed FP rounding hardware can be integrated with the RTL of the DCT transformation and quantization steps of the compression process. The proposed rounding hardware takes a FP number generated from the quantization step as input and generates the output rounded to the nearest integer value. The integration flow of the proposed rounding hardware with the computing core of the compression process is depicted in Fig. 5. Post integration, logic synthesis step can be performed to generate a gate level netlist (firm IP core) of the compression processor. The proposed rounding hardware module can serve as an application specific add-on to the multimedia accelerators.

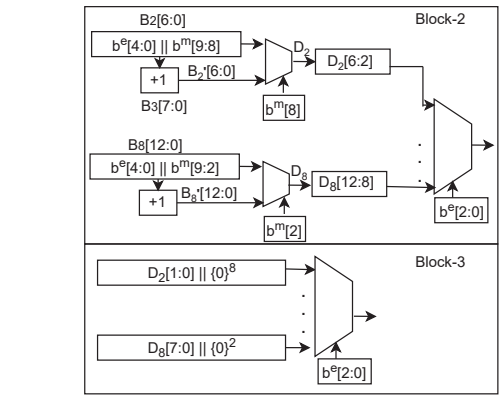

Fig. 4. An excerpt of optimized architecture of 16-bit FP rounding hardware

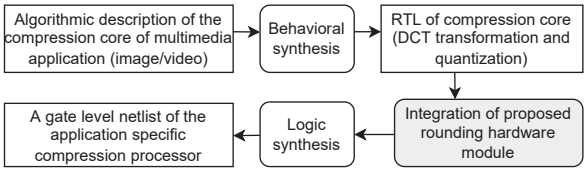

Fig. 5. The integration flow of proposed FP rounding hardware with the compression processor design

III. Experimental Results and Analysis

This section first discusses the *area-power-delay* analysis of the proposed rounding hardware using 15 nm technology based open-cell library [9]. Second, the hardware implementation and functional validation of the proposed mechanism has been done in Intel cycloneII FPGA using Quartus-II tool.

The proposed basic (X) and optimized (Y) architecture of the 16-bit binary FP rounding hardware are analyzed in terms of area, power and latency as shown in Fig. 6. Clearly, the optimized design (Y) achieves around 6.7% reduction in the area and 7.4% reduction in power when compared with basic design (X). Further, the latency of the proposed architecture is estimated to be 110 ps, which signifies high performance feature of the design.

The proposed rounding architecture is implemented for 16-bit and 32-Bit FP numbers. The resource utilization on the FPGA are presented in Table II. The resources used, in terms of look-up-tables (LUTs), by the proposed 16-bit and 32-bit FP

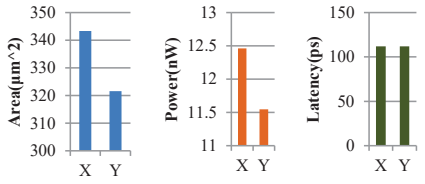

Fig. 6. Area, power and latency comparison of proposed basic (X) and optimized (Y) architecture of 16-bit FP rounding hardware

TABLE II
RESOURCE UTILIZATION SUMMARY OF 16-BIT AND 32-BIT FP ROUNDING HARDWARE DESIGN IMPLEMENTED IN INTEL'S CYCLONE II FPGA

FPGA resources		16-bit FP	32-bit FP
Logic elements usage by number of LUT inputs	(4 input functions)	92	378
	(3 input functions)	44	218
	($\leq$ 2 input functions)	75	392
Dedicated logic registers		0	0

rounding hardware designs are 0.2k and 1k respectively. Further, the integration of the proposed 16-bit and 32-bit rounding hardware modules with the hardware of the computing kernel of compression mechanism is also implemented on the same FPGA platform. Table III presents the resource utilization of the proposed rounding hardware integrated with compression kernel. Here, we compare the FPGA resource requirement of the compression kernel hardware implementation with and without the proposed rounding hardware module integrated. Further, the comparison is made in terms of the LUTs, dedicated registers and embedded multipliers. As indicated, the average resource overhead due to the integration of the proposed rounding hardware module is less than 1%.

A. Applications

In this section, we discuss the applicability of our proposed hardware design to improve the overall performance of multimedia accelerators. More specifically, we discuss the JPEG image compression application evaluated using conventional JPEG compressor (integrated with proposed hardware design). Thereafter, we briefly discuss how the proposed work can benefit in the design of popular machine learning (ML) and deep learning (DL) accelerators.

1) JPEG Compression: As discussed earlier, JPEG image compression requires rounding to integers during the quantization step. Therefore, integrating the proposed hardware of rounding to the nearest integer with the compression kernel of JPEG compression processor not only facilitates the performance enhancement but also eliminates the JPEG dimple artifacts. Also, since the JPEG based lossy image compression method allows a trade-off between storage size and the adjustable degree of compression. Thereby, we showcase (depicted in Fig. 7) the original image and the images compressed using the proposed FP rounding hardware for quality factor (QF) 0, 30, and 60 respectively. For this purpose, we have taken 256x256 cameraman image from image processing dataset. Thereafter, the evaluation of images (a), (b) and (c) (shown in Fig.7) is performed using the popular image quality evaluation metrics viz. peak signal to noise ratio (PSNR) and mean square error (MSE) [10] defined as below:

$$MSE = \frac{1}{n*m}\sum_{i=1}^{n}\sum_{j=1}^{m}(x_{i,j} - y_{i,j})^2 \quad (1)$$

$$PSNR = 10log_{10}\Big[\frac{255^2}{MSE}\Big]. \quad (2)$$

Table IV depicts the obtained PSNR and MSE values for images (b), (c), and (d) shown in Fig.7. We have evaluated these metrics using *MATLAB* 2018b running on system with 4GB of RAM. We have also traced the computation time required to compress the images. Clearly, with change in QF, the PSNR values improves by 1.2x times and the MSE reduces by 194.88 times respectively. However, the computation time remains nearly constant due to the algorithm being the same.

2) Scope for Other Applications: The proposed scheme can also be used in the design of ML and DL accelerators to achieve the enhanced performance. For instance, K-Means is a clustering algorithm used extensively in the machine learning. The algorithm clusters a given set of data points into k clusters. In this ML application, we initially quantize any FP number, if encountered during computation, into its nearest integer using the proposed rounding module. Thereafter, we use accurate Integer-ALU to compute euclidean distances. To demonstrate the application, we have used a data-set containing 150 points uniformly distributed into 3 clusters. Fig. 8 shows the outcome of the K-Means algorithm for the following two cases: (i) computation with the conventional FP numbers (the baseline) (ii) computation with the rounded FP numbers to nearest integer. We have used a metric called relative accuracy, defined by the number of data points clustered accurately in comparison to our baseline. While deploying proposed rounding hardware module to cater K-Means, we get a relative accuracy of 96.67% when compared with the baseline case. Thus, computationally expensive yet error-tolerant applications such as ML and DL can benefit greatly from the suggested hardware by replacing the resource consuming FP arithmetic with integer arithmetic respectively.

IV. CONCLUSION

As rounding operation performed during the quantization step of lossy compression techniques is a data intensive task, its execution through a dedicated hardware is vital to achieve enhanced performance. Additionally, performing rounding to the nearest integer is important to prevent against the potential compression artifacts such as JPEG dimples. Hence, we present an efficient rounding of binary FP numbers to the nearest integer value and its hardware design to aid the multimedia accelerators for the image and video compression applications. The proposed design offers high performance and has been optimized for the area and power consumption. The proposed rounding hardware design has been integrated

TABLE III
RESOURCE UTILIZATION SUMMARY OF PROPOSED ROUNDING HARDWARE INTEGRATED WITH COMPRESSION KERNEL IMPLEMENTED IN INTEL'S CYCLONE-II FPGA

FPGA resources	Compression kernel without integration of rounding hardware		Compression kernel with integration of rounding hardware		Overhead (16-bit FP)	Overhead (32-bit FP)
	16-bit FP	32-bit FP	16-bit FP	32-bit FP		
Logic elements usage in terms of LUTs	17,409	36,194	17,617	37,184	1.2%	2.7%
Dedicated logic registers	4,096	8192	4,096	8192	0%	0%
Embedded Multipliers (9-bit)	18	63	18	63	0%	0%

Fig. 7. JPEG compression perfromed using proposed hardware design. Obtained images: (a) Original Image, (b) Compressed Image (QF = 0), (c) Compressed Image (QF = 30), (d) Compressed Image (QF = 60)

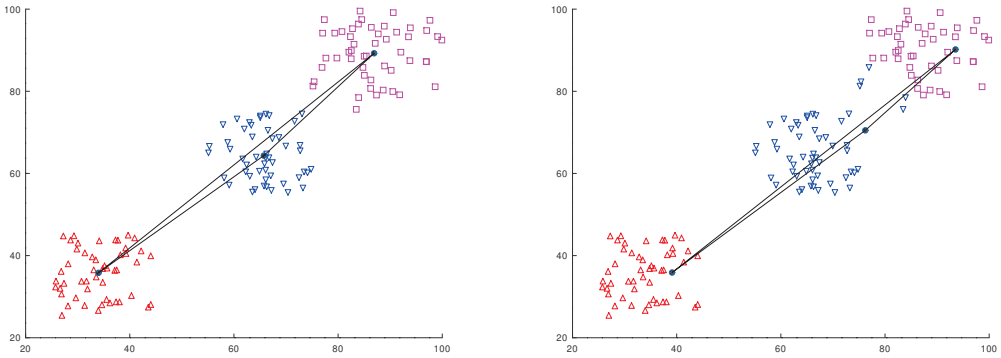

Fig. 8. Clustering results using accurate and rounding hardware for 150 data points, where x and y axis denote the coordinates of a particular data point

TABLE IV
ASSESSMENT OF COMPRESSED IMAGES

Image\Metric	PSNR (in dB)	MSE	Time (in sec)
(b)	18.71	877.16	5.62
(c)	28.36	94.85	5.87
(d)	41.62	4.478	6.21

with the compression kernel of the compression processor and implemented on an FPGA platform for the half and single precision FP numbers. The proposed FP rounding module hardware not only finds utility in the multimedia accelerators but has potential application in the ML and DL accelerators.

REFERENCES

[1] C. Pilato, S. Garg, K. Wu, R. Karri, and F. Regazzoni, "Securing hardware accelerators: A new challenge for high-level synthesis," *IEEE Embedded Systems Letters*, vol. 10, no. 3, pp. 77–80, 2018.

[2] T. Richter, "Evaluation of floating point image compression," in *Proc. ICIP*, 2009, pp. 1909–1912.

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

[4] 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.

[5] 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 (2006)*, vol. 6057, 02 2006.

[6] S. Agarwal and H. Farid, "Photo forensics from jpeg dimples," in *Proc. WIFS*, 2017, pp. 1–6.

[7] S. Tsen, S. González-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.

[8] 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.

[9] "Open cell library 15 nm . [online]," *Available: https://si2.org/open-cell-library/*, last accessed on, 2020.

[10] A. Baviskar, S. Ashtekar, and A. Chintawar, "Performance evaluation of high quality image compression techniques," in *Proc. ICACCI*, 2014, pp. 1986–1990.