IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received September 19, 2018, accepted November 3, 2018, date of publication November 12, 2018,
date of current version December 18, 2018.

Digital Object Identifier 10.1109/ACCESS.2018.2880480

High-Speed Fractal Image Compression Featuring Deep Data Pipelining Strategy

ABDUL-MALIK H. Y. SAAD[1] AND MOHD Z. ABDULLAH[1,2], (Member, IEEE)
[1]School of Electrical and Electronic Engineering, Universiti Sains Malaysia, Engineering Campus, Nibong Tebal 14300, Malaysia
[2]Collaborative Microelectronic Design Excellence Centre, Universiti Sains Malaysia, Engineering Campus, Nibong Tebal 14300, Malaysia

Corresponding author: Mohd Z. Abdullah (mza@usm.my)

The work of A.-M. H. Y. Saad was supported by Hodeidah University, Yemen. The work of M. Z. Abdullah was supported by Research University under Grant 1001/PCEDEC/6740037.

ABSTRACT A new architecture based on deep data pipelining is proposed for implementing fractal compression of high-resolution images in real time. The general idea is to partition an image into overlapping range and domain blocks in which four range blocks constitute one domain block. In this way, two matching operations can be performed simultaneously using two processor units. Further reduction in the encoding time is achieved by exploiting the inherently high degree of correlation among pixels in the neighborhood areas and restricting the search in the neighboring blocks only. The design is synthesized on Altera Stratix IV field-programmable gate array and optimized at circuit level in order to achieve a high-speed implementation. The proposed architecture is evaluated in terms of the peak signal-to-noise ratio (PSNR), the runtime, the memory utilization, and the compression ratio (CR). Experimental results suggest that the proposed architecture is able to encode a 1024×1024 size image in 10.8 ms with PSNR and CR averaging at 27 dB and 34:1, respectively. Meanwhile, the energy dissipation is approximately 0.5 W which is comparable to the state-of-the-art fractal processors.

INDEX TERMS Digital circuits, field programmable gate arrays, fractal image compression, parallel architectures, pipeline processing.

I. INTRODUCTION

Many organizations are now requiring video images from closed-circuit television (CCTV) are recorded and archived continuously for two weeks or more. For this purpose the wireless-based high-resolution camera are becoming more popular for CCTV application since this type of device offers greater flexibility, better performance and easier installation. This flexibility comes with three major setbacks: huge demand for storage, limited bandwidth and limited power supply. Therefore image compression is an integral part of the overall CCTV network. In this respect fractal image compression (FIC) offers an alternative technology because of its high compression efficiency, superior performance and simplicity. The basic idea is to transform 2-D image into a statistically uncorrelated dataset or fractals using the self-similarity between image blocks prior to transmission or storage [1]. Decoding is performed offline at some other time to reconstruct the original image from its fractals. This method was proposed first by Barnsley and Hurd [2] and further developed by Jacquin [3]. Compared to existing lossy techniques, FIC offers a high CR, especially when applied to images produced from aerial photography or captured by satellite imagery [3]. Owing to its popularity in digital archiving, FIC has found numerous applications in other fields of image processing such as character recognition [4] and watermarking [5]. FIC also possesses other attractive features like good PSNR performance and simple decoding method [6], [7]. However, FIC suffers one major drawback arising from the computational complexity of the algorithm. Typically, FIC requires a very large number of searches in order to find the best fractals which can accurately represent the original image. Therefore, the encoding time is measured in terms of minutes if not hours. Due to this reason, FIC is rarely used for processing high-resolution images especially those that contain small but multiple objects.

In an attempt to increase the speed-up and reduce the runtime, many researchers have developed new methods or improved techniques in implementing the FIC [8]–[17]. In general most of these methods are based in software in which the speed-up is achieved by reducing or limiting the number of search. For instance, Zheng *et al.* [8] and Wu *et al.* [13] used the genetic algorithm,

while Wang *et al.* [14] utilized the particle swarm optimization. In another study the Hadamard transform was investigated as means to speed-up the runtime [15]. In [12] and [16], prediction method is used along with the fractal technique to improve the runtime. Meanwhile, other research such as [9]–[11] implemented the classification scheme to accelerate the coding process by classifying blocks into groups, and applying matching search between blocks in the same class only. Although these methods proved to be useful, however, they suffer from two major weaknesses. First, the quality of the reconstructed image is relatively low because most of these techniques deployed partial search schemes as means for shortening the encoding time. Second, the runtime rarely reached a real-time speed even with a search-less schemes since most of these techniques are implemented in General Purpose Processors (GPPs) which operate in a serial fashion [18]. A slight departure from this trend are the work reported by Haque *et al.* [19], Erra [20], and Ismail *et al.* [21]. These authors targeted Graphic Processor Unit (GPU) for accelerating the algorithm by parallel processing instead of serial computation in the previous studies. Consequently, the encoding speed is significantly improved. However, the best attainable runtime is approximately 1 s which is still too slow for most real-time applications. Solving this problem requires a dedicated hardware which is far more superior in terms of speed compared to software-based solutions. Examples of recent work of FIC targeting the hardware implementation are published elsewhere [22]–[24]. One such design is based on a full-search scheme in which the entire image is examined for a given range block [22], [24]. However, the encoding time is generally slow. A significant improvement in speed is obtained in the design proposed by Samavi *et al.* [23]. These authors proposed an architecture using the classification approach to partially search for similarity. A runtime of approximately 0.8 ms has been reported when encoding a 256×256 greyscale image. Nevertheless, this approach requires many logic elements and memory block units for classification purpose. Hence the complexity of the hardware increases further if it is applied to encode high-quality images.

In this paper, a new high-speed FIC design is proposed for coding a high-resolution image of 1024×1024 pixels in size. This size is chosen since it supports the high end CCTV recorder. There are four major improvements in the design discussed in this paper compared to the work reported in [25] and [26]. First the current design is oriented for compressing high-resolution images compared to 256×256 size image previously. Second and more importantly, the scale value in the present design is estimated heuristically instead of been calculated empirically as in our previous work. This resulted in completely new architecture leading to a much more efficient design with remarkable improvement in the hardware utilization. Third the current architecture implements the partial-search scheme compared to full-search strategy in our previous designs. This leads to a significant speed-up in the runtime which is very important when processing high-resolution images. Finally, the current design exploits inherent parallelism in FIC and the architecture is implemented using deep data-pipelining compared to two-stage pipelining previously. Furthermore the design is optimized at circuit level, thus enabling the hardware to operate at much higher frequency as well as lowering the utilization of logic elements. By so doing the new design can compress much larger image size while maintaining the runtime and power consumption at relatively intact.

In summary, an image is partitioned into few sub-images, where each sub-image comprises numbers of range and domain blocks. For a given range block in each sub-image, a search is performed on all domain blocks in that sub-image only. In this way, a significant reduction can be achieved in memory access and runtime. Also, an image is partitioned in such a way that each range block is exactly one quarter of a domain block. In other words, each domain block contains 4 range blocks. Thus, fetching a domain block also means acquiring 4 range blocks simultaneously, or vice versa. As a result, two matching operations can be performed in parallel. This leads to an additional reduction in memory access, and hence increases the speed-up by the same factor.

As a trade-off between runtime and PSNR, this paper presents a new architecture based on the hardware-friendly partial search scheme together with deep-data pipelining. The new architecture is also optimized at gate level, thus ensuring the efficient use of hardware resources as well as achieving higher clock speed. These are two major contributions which we wish to highlight in his paper.

This paper is organized as follows. Section II contains a comprehensive review of the related hardware-based works. Section III explains the encoding and decoding approach of FIC method. Thereafter, the methodology applied in this research is described in Section IV. Then, Section V describes the proposed hardware architecture. This includes the description of the memory organization, the design units' functions and the data flow for the encoding process. The performance of this proposed architecture is then discussed in Section VI, including the comparison with the previous works. Finally, the paper is concluded in Section VII.

II. RELATED WORKS

To date, there are few designs proposed to speed-up the fractal calculations in hardware. Almost all the designs exploit the inherent parallelism in FIC as means for speed-up. Few designs minimize the computational complexity existing in the coding operations by avoiding use of the high complexity operations or reducing the operation precision or both. An example is the design reported by Vidya *et al.* [22]. These authors developed a parallel architecture based on 3-stage pipeline strategy. Comprising of eight symmetrical elements and a total of 434 bits of the functional units, such a design improves the runtime as it takes less 14.15 s to encode 128×128 size image. Nevertheless this speed is far too slow for high-speed applications.

Recently, Panigraphy *et al.* [24] introduced another FIC hardware based on full-search architecture. This design consists of many working units such as the range and domain control unit, the mean computation unit, and the sum of absolute differences (SAD) unit. In addition, the isometric transformation is used in this architecture to increase the matching probability. Unlike conventional FIC, however, only the range blocks are processed, thereby, reducing the encoding time remarkably since the transformations do not require additional clock cycle. Also, the encoding time is reduced by terminating the search operation whenever the matching error obtained is lower than the predetermined threshold limit. Overall, their hardware resulted in 0.4 s runtime and 31 dB PSNR when tested using 256 × 256 size image. Though the PSNR is adequately acceptable, however, the runtime does not meet the 30 fps real-time requirement.

In an attempt to improve the runtime, Samavi *et al.* [23] present another architecture based on a partial search scheme. These authors incorporated the binary classifier in their design which groups 8 × 8 range and domain blocks into 32 distinct classes. In each class, only eight domain blocks are available. They are compared with the range blocks of the same class. For a given class, the image data from a given domain blocks are fetched pixel-by-pixel, and loaded serially into eight RAM blocks. Then, these data are compared with every range block of the same class in parallel. This resulted in the significant improvement in the speed-up. On average their architecture takes approximately 0.8 ms to encode one 256 × 256 greyscale image. Although this design is very promising, however, the architecture is tested using low-resolution images only. It is expected that the complexity of this hardware in terms of the memory utilization and logic element will increase significantly since the design implements image processing prior to coding.

Generally, the majority of the previous designs targeted low-resolution applications with image size of 256 × 256 pixels maximum. Therefore, their performance for applications involving high-resolution images remains relatively unknown. One of the factors which hinders compressing high-resolution image arises from the complexity of the architecture especially when an existing parallelism in the algorithm is not amenably exploited. Therefore, in this paper, we propose a novel architecture using efficient data pipelining strategy and inherent parallelism in both FIC and FPGA, resulting in a significant improvement in the performance compared to the previous designs.

III. FRACTAL IMAGE COMPRESSION

A. BACKGROUND

The basic idea of FIC is to arrive at a set of transformations that can map an image into itself. It relies on the existence of local self-similarities between sub-images and other parts of the same image. In coding, an image is partitioned into a set of small non-overlapping blocks of size $n \times n$, known as range blocks. Another set is produced by partitioning the same image into larger $c \times c$ blocks referred to as domain blocks. The domain blocks are usually overlapped and having a double size of the range block, i.e., $c = 2n$. For a range block, the algorithm searches the entire or part of the domain pool until the best match is obtained. In searching, the affine transformation is applied to each domain block, thus producing the fractals which best describe the original image. In general, the affine transformation involves a combination of different types of transformations. First, the contracting transformation shrinks the domain block to the same size of the range block. This is achieved by averaging the intensity values of each non-overlapping square block of 2 × 2 pixels in the domain pool. Second, the geometric transformation changes the pixels position in the contracted domain block for producing eight symmetries. Third and last, the intensity value of the contracted domain block is modified via an affine transformation. For simplicity, we only consider the contrast scaling and brightness shifting, while the geometric transformation is discarded. Therefore the affine transformation is expressed as:

$$T(D) = s \cdot D + g \cdot I \tag{1}$$

where D is the contracted domain block, s and g are the scaling and offset values, respectively, and I is the $n \times n$ matrix whose entries are equal to 1. Meanwhile The scaling factor lies between -1 and 1 [27].

The similarity between two blocks can be measured by different metrics such as the Mean Square Error (MSE), the Sum of Absolute Differences (SAD), and the Least Square Error (LSE). From hardware point of view, SAD is relatively less complex since it uses simple operations compared to MSE and LSE which require a number of multiplication operations. In this case, SAD measures the distortion between range and domain blocks defined as follows:

$$SAD(R, D) = \sum_{i=1}^{N} |R_i - T(D_i)| = \sum_{i=1}^{N} |R_i - sD_i - gI| \tag{2}$$

where R_i and D_i are, respectively, the greyscale values of the range block and the contracted domain block at i_{th} location, and N is the total number of pixels in the range block, i.e., $N = n \times n$. The optimal values of s and g are given by:

$$s = \frac{N \sum_i^N R_i D_i - \sum_i^N R_i \sum_i^N D_i}{N \sum_i^N D_i^2 - \sum_i^N D_i \sum_i^N D_i} \tag{3}$$

and,

$$g = \mu(R) - s \cdot \mu(D) \tag{4}$$

where $\mu(R)$ and $\mu(D)$ are, respectively, the mean values of range and domain block. Since s is always quantized into a few bits, in this case 2 to 5 bits [28], [29], therefore, examining all possible combinations of s concurrently is more preferred scheme instead of using (3) directly. Furthermore, the implementation of Equation (3) requires a divider IP core since the divisional operation is not directly synthesizable

in FPGA. In most applications, it is sufficient to represent s in 2 bits. Therefore in this design, the value of s is one chosen from four available options, i.e., 0.25, 0.5, −0.5 and 1. As a result, scaling operation can be performed simply with right shifting.

For each range block R, the algorithm searches all or selected domain blocks until SAD is minimized. The s and g values, i.e. Fractal Code (FC), corresponding to minimum SAD are store together with the spatial information. Here, g is quantized into 7 bits prior to storage.

B. DECODING PROCEDURE

The decoding method is an iterative process, starting from any arbitrary image and then updating it until a convergence is achieved. In the process, each range block is reconstructed as follows:

$$R = sD_{x,y} + gI \tag{5}$$

where R is the reconstructed range block, $D_{x,y}$ is the corresponding domain block at (x, y) coordinate, s and g are, respectively, the stored scale and offset values. After each iteration, the resulted image is used as an input for the next iteration. Generally, the decoding requires not more than 10 iterations to faithfully reconstruct an image. The fidelity of the reconstruction is calculated in terms of PSNR which is defined as:

$$PSNR = 10\log_{10}\frac{255 \times 255}{\frac{1}{W \times H}\sum\left(f - \tilde{f}\right)^2} \tag{6}$$

where f and $\tilde{f}$ are the original and reconstructed images, respectively, and $W \times H$ is the image size.

IV. METHODOLOGY

As previously discussed, the essence of FIC is the pairing of each range block to a domain block so that the error under affine transformation defined in (2) is minimum. This procedure is very time consuming since it entails searching all blocks in the domain pool and repeatedly performing the affine calculations. Based on the fact that the correlation degree among adjacent blocks is generally higher for images captured from natural scenes [1], [30], the search can be restricted to the neighboring area of the corresponding range block only. This would lead to a significant speed-up in the encoding time.

Exploiting the similarity between a set of pixels in the image and an area surrounding it, an FIC search scheme is proposed by partitioning an image in such a way that all range blocks in $K \times K$ window have the same domain pool. This allows matching of multiple range blocks in the same set with their domain counterparts be performed in parallel. This proposed scheme is different than the search scheme used elsewhere [31], [32], where each range block has a distinct domain pool.

In so doing, the image is segmented into sub-images of size $K \times K$ pixels, where each sub-image is partitioned into

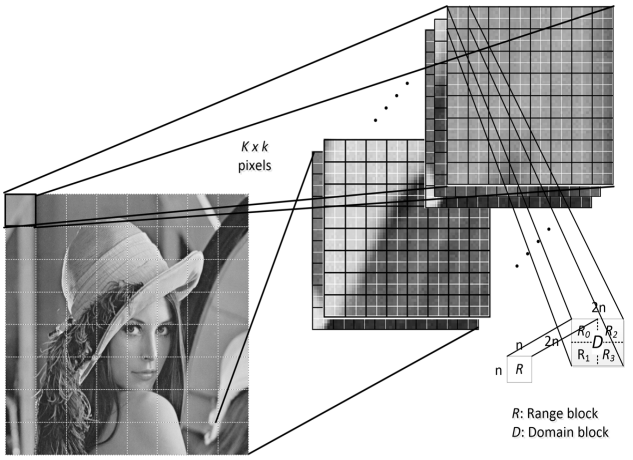

FIGURE 1. Image partitioning structure.

groups of non-overlapping range and domain blocks of $n \times n$ and $2n \times 2n$ pixels, respectively, as shown in Fig. 1. In this case, each sub-image comprises $K/n \times K/n$ range blocks and $K/2n \times K/2n$ domain blocks. Here, the range block size is rigidly fixed to an appropriate size. Larger size leads to higher degradation because of the lowering in the degree of matching, while smaller size reduces the compression efficiency. As a trade-off, the 8×8 range size is selected for this work. This corresponds to a compression efficiency of at least 25, which is desirable especially when coding high-resolution images.

Referring again to Fig. 1, each domain block comprises of four non-overlapping range blocks. In other words, each range block is exactly a quarter in size of a domain block. Thus, reading 4 range blocks indirectly means acquiring 1 domain block. Similarly, reading a single domain block corresponds to fetching 4 range blocks. Since these blocks are already available in the system, therefore, they can also be utilized for pairing and matching, thus saving the clock's cycles required for fetching them individually from memory. This leads to a reduction in memory access and increases the speed of compression.

The proposed algorithm is firstly investigated in software in terms of $K \times K$ window size, starting from the smallest window of $K = 32$ to the largest window of $K = 1024$ which is equivalent to full search. Since each domain block is rigidly fixed to 16×16 pixels, therefore, the number of domain blocks in each pool is $(K/16)^2$, which is equivalent to the number of matching operations required for each range block. For smaller window size, the matching number is reduced significantly. For instance the reduction in matching increases sharply from 0 % to 74 % when K decreases from 1024 to 512 respectively. This trend continues in almost linear fashion with decreasing K, and reaching almost 98 % when K reaches 128. Further reduction in K results in no significant change in the number of matching as evident from Fig. 2(a). Since the runtime is directly proportional to the number of matching, therefore, the runtime also decreases by the same amount. Fig. 2(b) and 2(c) show the effect of

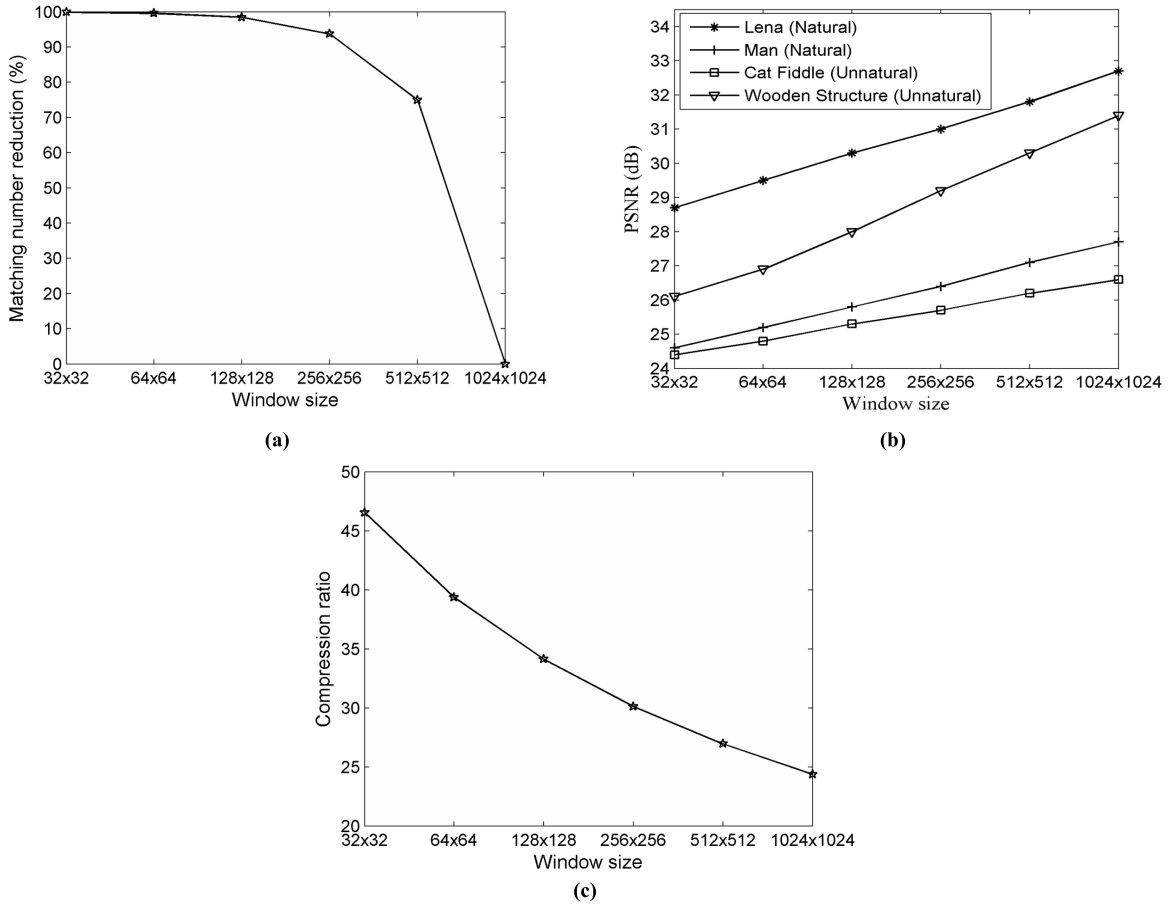

FIGURE 2. The effect of varying window size on compression performance: (a) Window size vs number of matching, (b) Window size vs PSNR, (c) Window size vs compression ratio.

varying the window size on compression performance using four images, including natural and unnatural textured images of size 1024×1024 each. Lena and Man images in Fig. 11 represent natural images, while Cat Fiddle and Wooden Structure images shown in Fig. 12 represent unnatural textured images. Clearly from Fig. 2(b), the PSNR increases when K is increased. In this case, the larger the window size the more the information that is used in the compression as depicted in Fig. 2(c), leading to the increased in number of matching, and subsequently PSNR. As a trade-off between runtime and accuracy, the window size is rigidly fixed to 128×128 pixels, yielding PSNR values which lie between 25 dB to 30 dB, and averaging at 27 dB for four given images.

In performing the compression, an image is first partitioned into 8×8 non-overlapping sub-images, sub-img_k, $k = 0 \cdots 63$. Shown in Fig. 3, each sub-image is 128×128 pixels in size. Altogether, there are 64 sub-images for a 1024×1024 size image. Second, each sub-image is further divided into 8×8 non-overlapping domain blocks of 16×16 pixels size each, i.e. $D_j, j = 0 \cdots 63$. The assembly of all domain blocks in one sub-image constitutes one domain pool. Therefore, the total number of domain blocks in each domain pool is 64 or 4096 for the entire image. Finally, each domain block D_j is divided into four non-overlapping range blocks, $R_{j,p} : p = 0, 1, 2, 3$ of 8×8 pixels size each. Each four adjacent $R_{i,p}, i = 0 \cdots 63$ is compared serially with all Ds available in the sub-image pool. At each comparison between $R_{i,p}$ and D_j, the corresponding $R_{j,p}$ and D_i are also available and subsequently another matching operation can thus be performed. Thus both $R_{j,p} - D_i$ and $R_{i,p} - D_j$ matchings can be performed in parallel in hardware using one processor each. From each matching, the fractal codes corresponding to the best match are stored. In order to avoid repetition, the $R_{i,p} - D_j$ matching is performed in the first processor and for $j \geq i$ while similar matching but for $j < i$ is performed in the second processor. For instance, in the first cycle (i.e. $i = 0$) the first 4-range block from the first domain $R_{0,p}$ is matched with all 64 domain blocks $D_j : j = 0 \cdots 63$ by the first processor sequentially. At the same time $R_{j,p} : j = 1 \cdots 63$ is compared with D_0 by the second processor. In the second cycle, i.e. $i = 1$, the next 4-range block $R_{1,p}$ is compared with $D_j : j = 1 \cdots 63$ only because the $R_{1,p} - D_0$ matching has already been performed in the first cycle. Also in this cycle the $R_{j,p}$;$j = 2 \cdots 63$ is compared with D_1 by the second processor.

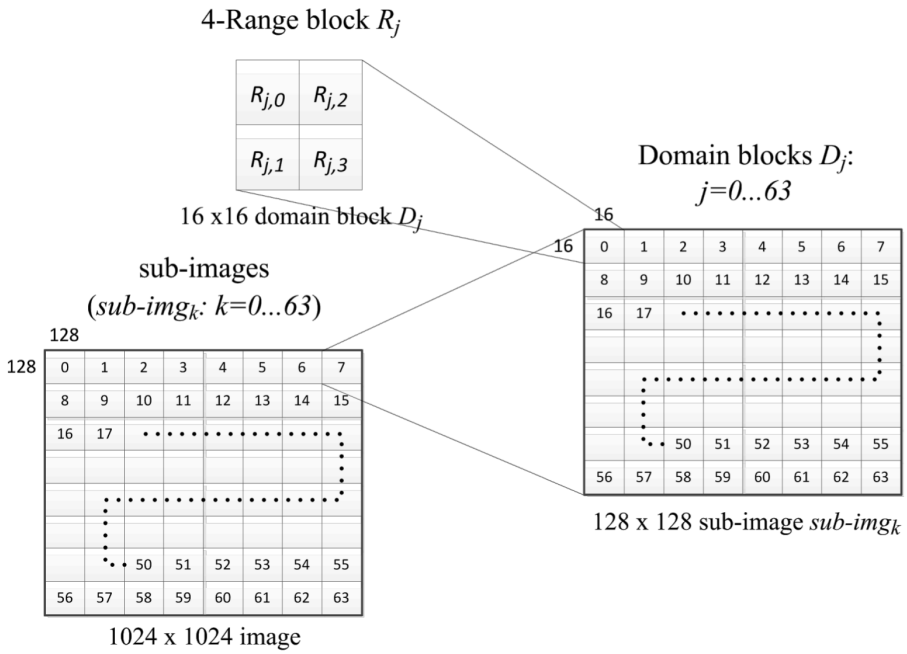

FIGURE 3. Steps in image partitioning.

Therefore, for each new cycle, the number of domain blocks need to be matched is $64 - i$. Clearly in this case the number of matching is reduced by one at each matching cycle, resulting in a significant reduction in search time. For clearer understanding, a flowchart of this proposed algorithm is given in Fig. 4.

V. PROPOSED HARDWARE ARCHITECTURE

A new hardware architecture depicted in Fig. 5 is proposed to implement the methods and procedures previously discussed. The proposed design is optimized to work at higher frequency. Parallelism and pipelining techniques are also exploited in designing the principal units of the proposed architecture. Essentially the system comprises of: (i) Address Generation Unit (AGU), (ii) the Memory Control Unit (MCU), (iii) the Mean and Contracted Domain Computation Unit (MCDCU) and its control unit MCDCU-Ctrl, (iv) the Offset Computation Unit (OCU), (v) the Sum of Absolute Differences Computation Unit (SADCU) and its control unit SADCU-Ctrl, (vi) the Storing Control unit (SCtrl), and (vii) the Fractal Codes and SAD Storing Control unit (FC-SAD SCtrl). In this case the AGU is used to generate the required addresses. Meanwhile the MCU is used to control the main memory while the MCDCU, OCU and SADCU are used to compute the mean, offset and sum of absolute differences values respectively. The MCDCU-Ctrl, SADCU-Ctrl, SCtrl and FC-SAD SCtrl are used to control the overall operation of the hardware. More details are discussed in the following sub-sections.

A. MEMORY ORGANIZATION

Altogether, 1 megabyte of memory is needed for buffering 8-bit and 1024×1024 size image. This is equivalent to 2^{17} memory locations of 8-byte size each. The image is stored in the memory in a row-wise order. Altogether 17-bit addresses are required to address all memory locations. In order to read a particular domain or four adjacent range blocks, both the block and the sub-image numbers are required to generate the correct addresses. As explained earlier, the image is partitioned into 64 sub-images, and each sub-image contains 64 domain blocks. Therefore, addressing each sub-image and domain require 6-bit address, resulting-in a 12-bit long address. The remaining 5 bits are used to address the domain block pixels. The 17-bit long address ($Addr$) is formatted as follows: bits 16-14 ($Addr_{16...14}$) and bits 13-11 ($Addr_{13...11}$) correspond to 3 most significant bits of sub-image and domain block addresses respectively; bits 6-4 ($Addr_{6...4}$) and bits 3-1 ($Addr_{3...1}$) correspond to 3 least significant bits of sub-image and domain block addresses respectively; bits 10-7 ($Addr_{10...7}$) and bit 0 ($Addr_0$) correspond to an address of domain pixel. Respectively, the base addresses of each sub-image and domain block are xxx-000-0000-xxx-000-0 and xxx-xxx-0000-xxx-xxx-0.

B. ADDRESS GENERATION UNIT (AGU)

The AGU is responsible for generating the correct addresses of range and domain blocks. The range and domain blocks should be read in orderly fashion as explained in Section IV. Therefore, AGU employs four counters to generate the required addresses as shown in Fig. 6. These counters are: sub-image address counter S_Cr, 4-range block address counter R_Cr, block address counter B_Cr, and block pixel address counter P_Cr. The P_Cr generates the addresses needed to read all pixels in each domain or 4-range block. The vertical scanning mode is used in reading block pixels,

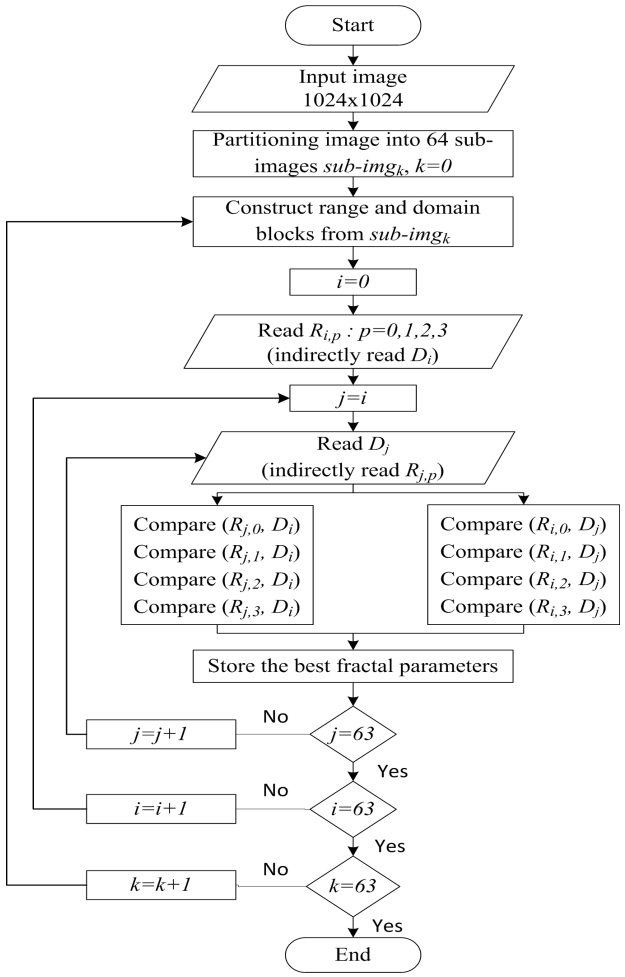

FIGURE 4. Flowchart of the proposed algorithm.

starting from the top-left pixel and ending with bottom-right pixel. In this case, the *P_Cr* is a 5-bit counter whose registers are advanced by one step for each clock as long as the *GenerateAddEn* remains high. Meanwhile, *B_Cr* generates the base address of the required 4-range and domain blocks. Once the *P_Cr* reaches the final count (*P_Cr* = 11111) then *B_Cr* is either loaded with *R_Cr* value plus 1 corresponding to the base address of the subsequent 4-range block or enabled to generate the base address of the subsequent domain block as long as *B_Cr* $\neq$ 63. The value 63 signals the end of the search for a specific 4-range block. The R_Cr is also enabled concurrently while *B_Cr* is being loaded. The encoding of the corresponding sub-image is completed when *R_Cr*, *B_Cr* and *P_Cr* reach the maximum count. The *S_Cr* is then enabled to generate the base address of the subsequent sub-image. Hence, the fetching process of the range and domain blocks continues in the same manner until all sub-images have been processed.

C. MEAN AND CONTRACTED DOMAIN COMPUTATION UNIT (MCDCU)

This MCDCU unit is designed to complete two tasks. The first is to compute the mean value of the range and domain blocks, and the second is to perform a geometric contraction of the domain blocks needed in fractal calculations. The structure of this unit is depicted in Fig. 7. These two tasks are grouped together so that the hardware can be shared, thus resulting in the reduction of logic elements. Mean value computation is performed in 3-stage pipelining to increase the maximum clock frequency and throughput. Pipeline registers and multiplexers existing in the unit are controlled by MCDCU-Ctrl unit.

Once the block's data, i.e. 8 pixels vector $P_n{:}n = 0 \cdots 7$, are available in the databus, they are added together producing the sum values of range and domain blocks. The sum values are then shifted to the right by 6 bits, producing the average value of the range and the contracted domain block, i.e. *MeanR* and *MeanD* respectively. These values are available 3 clock cycles later owing to 3-stage pipelining, as evident from Fig. 7. Altogether, 8 and 32 clock cycles are needed to fetch each range and domain block, respectively. Therefore, the *MeanR* and *MeanD* values are available in 11th and 35th clock cycles, respectively.

A domain block is contracted in MCDCU by means of 2×2 window averaging. For passing 2×2 pixels window into MCDCU unit, two consecutive rows of 8-pixels (2×8 pixels) need to be read. This enables four window averaging be performed and the whole process requires 2 clock cycles. Resultantly four contracted domain block values, D_4Byte are produced in every two cycles. These values are then used to calculate the mean value of the contracted domain block, *MeanD*. Also the same values are stored in a dual-port RAM for computing the matching error in the later stage of image coding.

D. OFFSET COMPUTATION UNIT (OCU)

The OCU is responsible for computing four offset values g_u corresponding to the four predefined scale values s_u ($s_0 = 0.25, s_1 = 0.5, s_2 = -0.5$ and $s_3 = 1$). Both *MeanR* and *MeanD* values are needed in the computation. The OCU is only enabled after both *MeanR* and *MeanD* values are available in the memory. Since s values are represented in powers of 2, therefore, $s \times$ *MeanD* operand in (4) can be performed easily by right-shifting *MeanD*. This improves the performance in terms of speed, logic elements and power. In the implementation, the OCU needs four adder or subtractor elements.

E. SAD COMPUTATION UNIT (SCU)

The SCU is shown in Fig. 8 with 4-pipeline stages. In the computation, the affine transformation described in (1) is performed first to obtain $T(D)$ and then subtracted from R to produce the $R - T(D)$ value in the first pipeline stage. Since s has four values, therefore four $R - T(D)$ values are generated for each pixel in the range and domain blocks. As shown in Fig. 8, altogether every 8 pixels in each range ($P_{xr}{:}x = 0 \cdots 7$) and domain blocks ($P_{xd}{:}x = 0 \cdots 7$) are passed to SCU along with the corresponding g_u values in every cycle of Stage 1 in the pipeline. This means

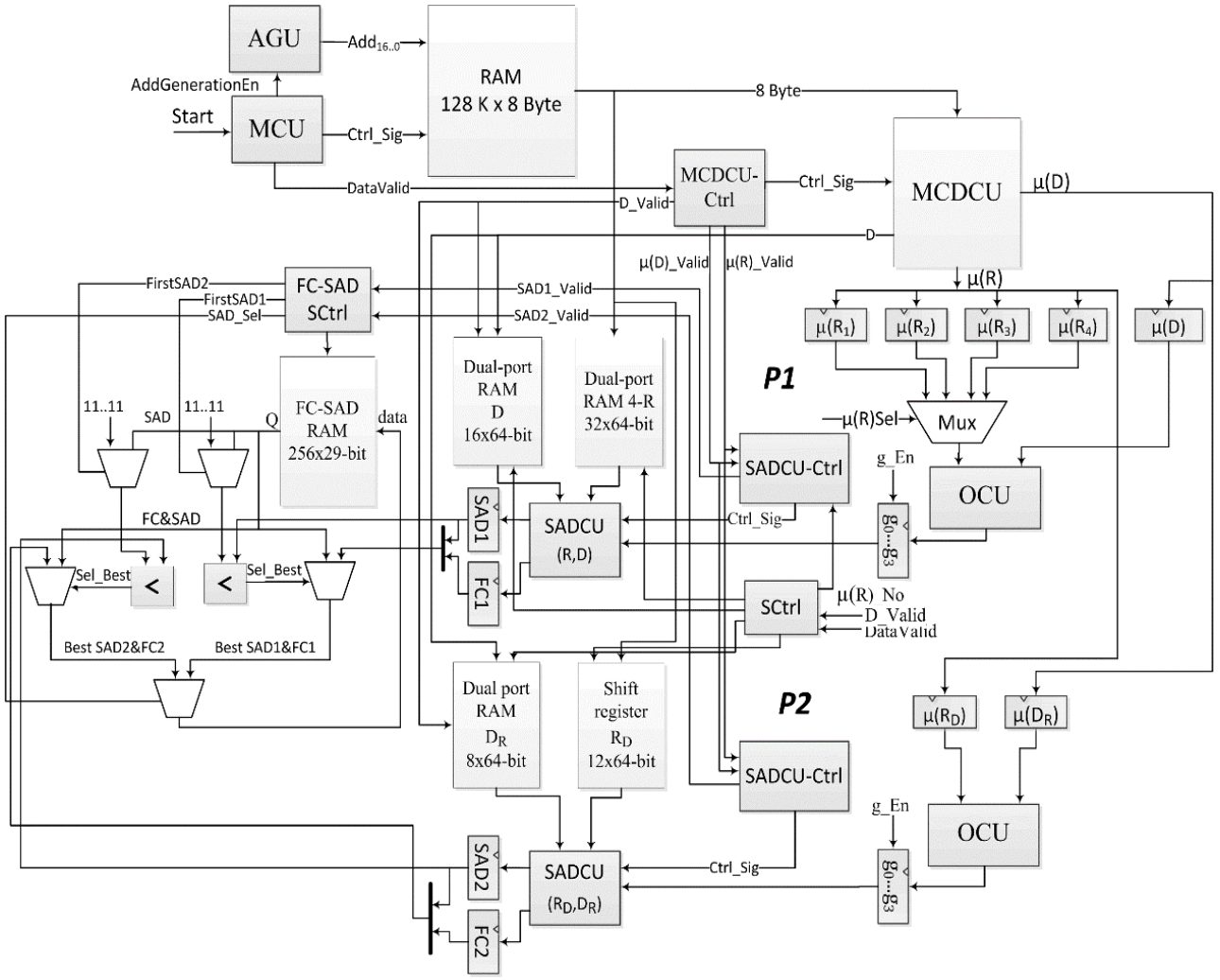

FIGURE 5. Overall hardware architecture of the proposed design.

32 difference values are produced in each cycle. Then, the *SAD* value is computed according to (2). This is performed in the following two cycles, corresponding to Stage 2 and Stage 3 in the pipeline. Four *SAD* values, SAD_u:$u = 0, 1, 2, 3$, are compared in order to select the minimum *SAD*, *MinSAD*. The comparison is performed in Stage 4. This value together with the corresponding s and g values are stored for further processing.

F. DATA FLOW FOR THE ENCODING PROCESS

Fetching process of the image pixels is initiated by the MCU once the start signal, start, is asserted. Simultaneously, the AGU is enabled to generate the required addresses of the range and domain blocks by asserting the control signal, *GenerateAddEn*. MCU unit provides a control signal *DataValid* which is asserted when the pixels data is available in the data bus. This *DataValid* signal gives an indication to other computational and control units to start receiving or processing the data.

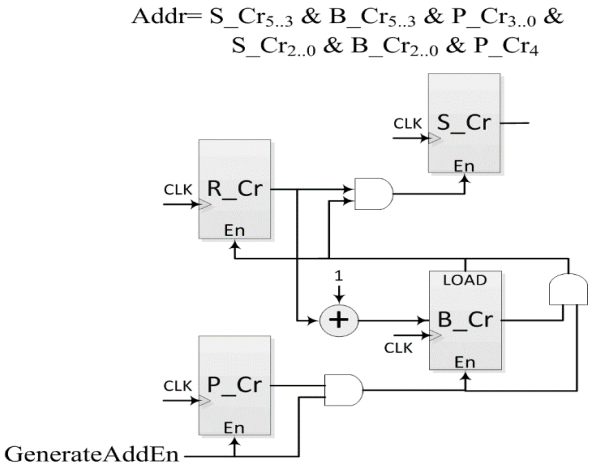

FIGURE 6. The architecture of the Address Generation Unit.

In the proposed architecture, there are two matching processors, *P1* and *P2* working in parallel. Each processor contains separate RAMs, OCU, SADCU and SADCU-Ctrl

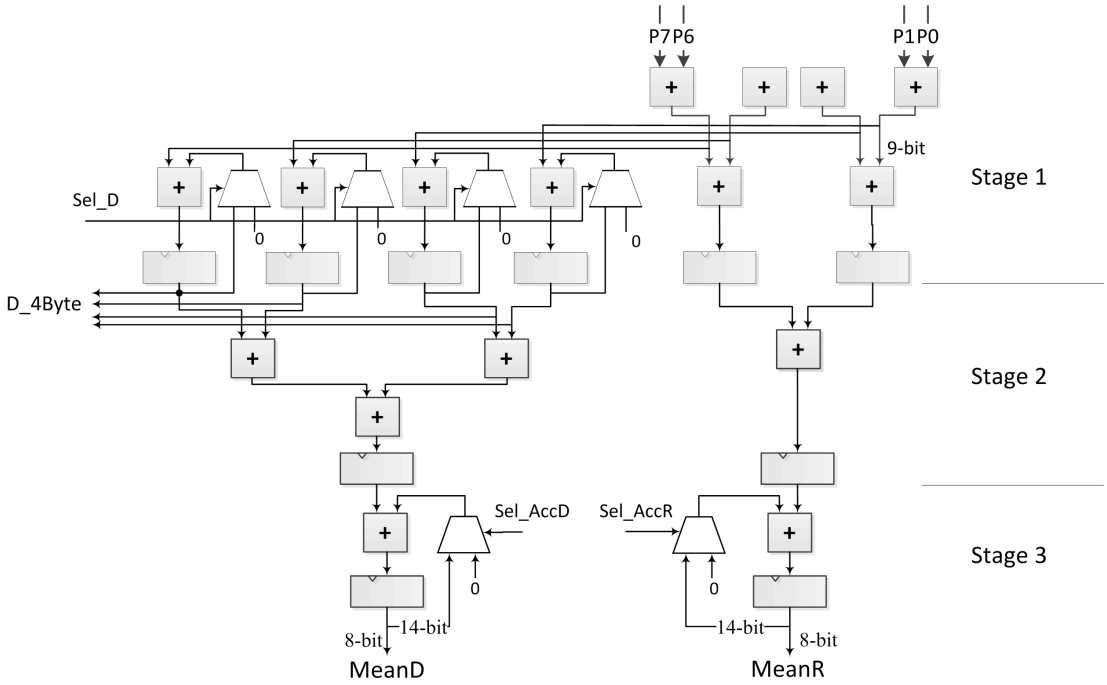

FIGURE 7. The architecture of the Mean and Contracted Domain Computation Unit.

units. Other units like MCDCU, MCDCU-Ctrl, SCtrl and FC-SAD SCtrl are shared by both processors. During operation, *P1* performs the matching operation between each 4-range block R_i and domain block D_j:$j = i \cdots 63$, while P2 performs the matching operation between each domain block D_i with every 4-range block R_j:$j = i{+}1 \cdots 63$. Both D_i and R_j are constructed, respectively, from the 4-range block R_i and the domain block D_j used in P1. D_i and R_j are denoted as D_R and R_D in the Fig. 5, respectively. Thus, the second matching operation does not need any additional clock cycle since all the information needed for computation are already available in the memory. In each matching operation, the corresponding processor calculates g and SAD values for a given value of s. The minimum SAD value along with the corresponding fractal codes, s and g values, are stored for further processing.

In order to match $R_{\mathrm{i,p=0,1,2,3}}$ with $D_{\mathrm{j=i\cdots 63}}$ in *P1*, four adjacent range blocks $R_{\mathrm{i,p=0,1,2,3}}$ are fetched sequentially in the following order - $R_{\mathrm{i,0}}$, $R_{\mathrm{i,1}}$, $R_{\mathrm{i,2}}$ and $R_{\mathrm{i,3}}$. Then the domain blocks $D_{\mathrm{j=i\cdots 63}}$ are fetched in the following cycles. Since the first domain block that needs to be compared is D_i, which is also equivalent to 4-range block R_i has already been fetched, therefore reading the subsequent domain blocks starts from $i + 1$. The mean values of $R_{\mathrm{i,p=0,1,2,3}}$ and $D_{\mathrm{j=i\cdots 63}}$ are computed by MCDCU and the results are presented in $\mu\,(R)$ and $\mu\,(D)$ outputs respectively. As previously explained in the Sub-section C, MCDCU needs 3 clock cycles to calculate the mean, due to 3-stage pipeline strategy. In this case, $\mu\left(R_{i,p}\right)$ and$\mu(D_j)$ are available after ($\mathrm{N}_R \times p{+}3$)th and ($\mathrm{N}_D{+}3$)th clock cycle respectively. N_R and N_D are the number of clock cycles required to fetch the range and domain block respectively, i.e. $N_R = 8$ and $N_D = 32$. The $\mu\left(R_{i,p=0,1,2,3}\right)$are then stored in four registers $\mu(R_p)$, while $\mu(D_j)$ is stored in μ (D) for offset calculation. Timing diagram in Fig. 9 summarizes the overall sequence of these operations.

During the computation, both R_i and contracted domain block D_j are stored in two dual-port RAMs with an 8-byte word size, and named as 4-R and D, respectively. These information are later used to calculate the SAD value. Altogether 16×64 bits are needed to store two consecutive domain blocks since the contracted D_j needs to be read four times for computing SAD($R_{i,0}$, D_j), SAD($R_{i,1}$, D_j), SAD($R_{i,2}$, D_j) and SAD($R_{i,3}$, D_j) sequentially, while the next domain block $D_{j+1}$is being written. Once $\mu\,(D_i)$ is stored in $\mu(D)$ register, *P1* is enabled to start matching operations. As seen from Fig.9, $\mu\,(D_i)$ is only available at 36th clock cycle. The matching process starts by firstly computing the offset values g_u of $R_{i,0}$ and D_i, and secondly passing the results together with the blocks' pixels stored in dual port RAM to SADCU unit for calculating the SAD values. SADCU needs 4 cycles to produce the minimum value of SAD_u, MinSAD. This value is then stored with its corresponding fractal codes in two registers, i.e. SAD1 and FC1. This operation is repeated for every 8 clock cycles for each one of $R_{i,1}$, $R_{i,2}$ and $R_{i,3}$. Every time the SAD1 value is compared with the previously stored SAD, i.e. the minimum SAD obtained from the previous matching operations for the same range block. The result from this comparison is written into FC-SAD RAM of size 256×29 bits. The FC-SAD SCtrl unit controls the reading

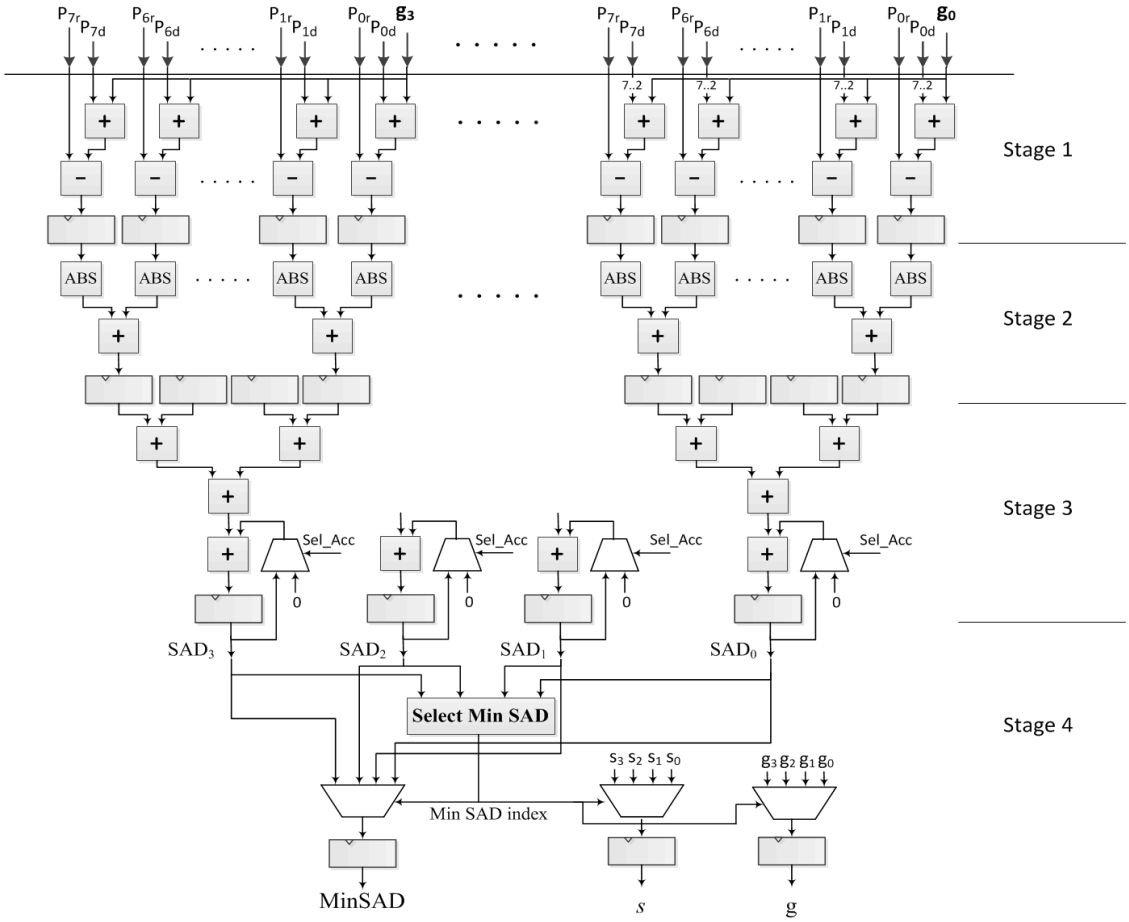

FIGURE 8. The architecture of the Sum of Absolute Difference Computation Unit.

and writing operations of the FC-SAD memory. In the same way, $R_{i,\mathrm{p}=0,1,2,3}$ are also compared with each D_j as soon as $\mu\left(D_j\right)$ is available. The timing diagram in Fig. 10 (a) shows important sequences needed in the computation of minimum SAD value by *P1*.

While *P1* is performing the calculation, the matching between $R_{j,p=0,1,2,3}$:$j = i{+}1 \cdots 63$ and D_i is executed in *P2* in parallel, utilizing the range and domain blocks used in P1. In operation, $\mu\left(R_{j,p}\right)$ is computed at the same time with $\mu\left(D_j\right)$ and the results are stored in $\mu\left(D_R\right)$. Similarly $\mu\left(D_i\right)$ is also computed at the same time with $\mu\left(R_{i,p}\right)$ and the results are stored in $\mu\left(R_D\right)$ as shown in Fig. 9. Both computations are performed by MCDCU. Unlike *P1*, however, the range block mean values are calculated after the calculation of mean value of the domain block. The matching starts once $\mu\left(R_{i+1,0}\right)$ is available in $\mu\left(R_D\right)$ register. As shown in the timing diagram in Fig. 9, $\mu\left(R_{i+1,0}\right)$ is stored in $\mu\left(R_D\right)$ register at 44th cycle. at the same time OCU computes g_u of $R_{i+1,0}$ and D_i, and stores the results in their respective register in the following cycles. Following this process the SADCU is enabled to calculate the *SAD* values. In so doing the SADCU requires the corresponding range and domain block, i.e. $R_{i+1,0}$ and D_i. Prior to calculation the $R_{i+1,0}$ is stored in 64 12-bit shift register. Meanwhile the contracted D_i is stored in a dual-port RAM, DR of size 8×64 bits. The outputs from SADCU are stored in two registers, i.e. SAD2 and FC2. These operations are repeated for each $R_{j,p=0,1,2,3}$:$j = i{+}1 \cdots 6$. Both SAD1 and SAD2 are examined together with their corresponding minimum *SAD* value stored in FC-SAD RAM to select the minimum *SAD* value. To avoid reading two individual memory locations at the same time, therefore, the minimum *SAD* value corresponds to SAD1 is read first, and followed by the minimum *SAD* corresponds to SAD2. Thus, SAD2 is examined one cycle later after the examination of SAD1. Fig. 10 (b) shows the overall sequence in SAD calculations by *P2*.

VI. EXPERIMENTAL RESULTS

The proposed architecture is coded in VHDL and compiled by Altera Quartus II software package. Altera Stratix IV (EP4SGX230KF40C2) FPGA device is used to implement the hardware design. Table 1 summarizes the hardware utilizations and other important information. As can be seen from this table, the proposed design consumes 4799 logic

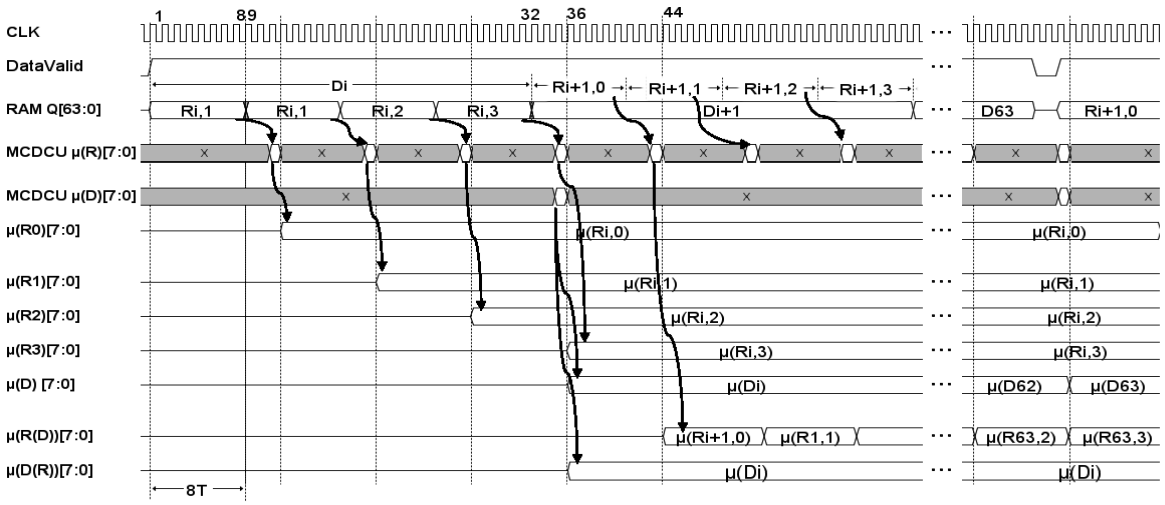

FIGURE 9. Timing diagram for reading range and domain blocks, and calculating the mean values.

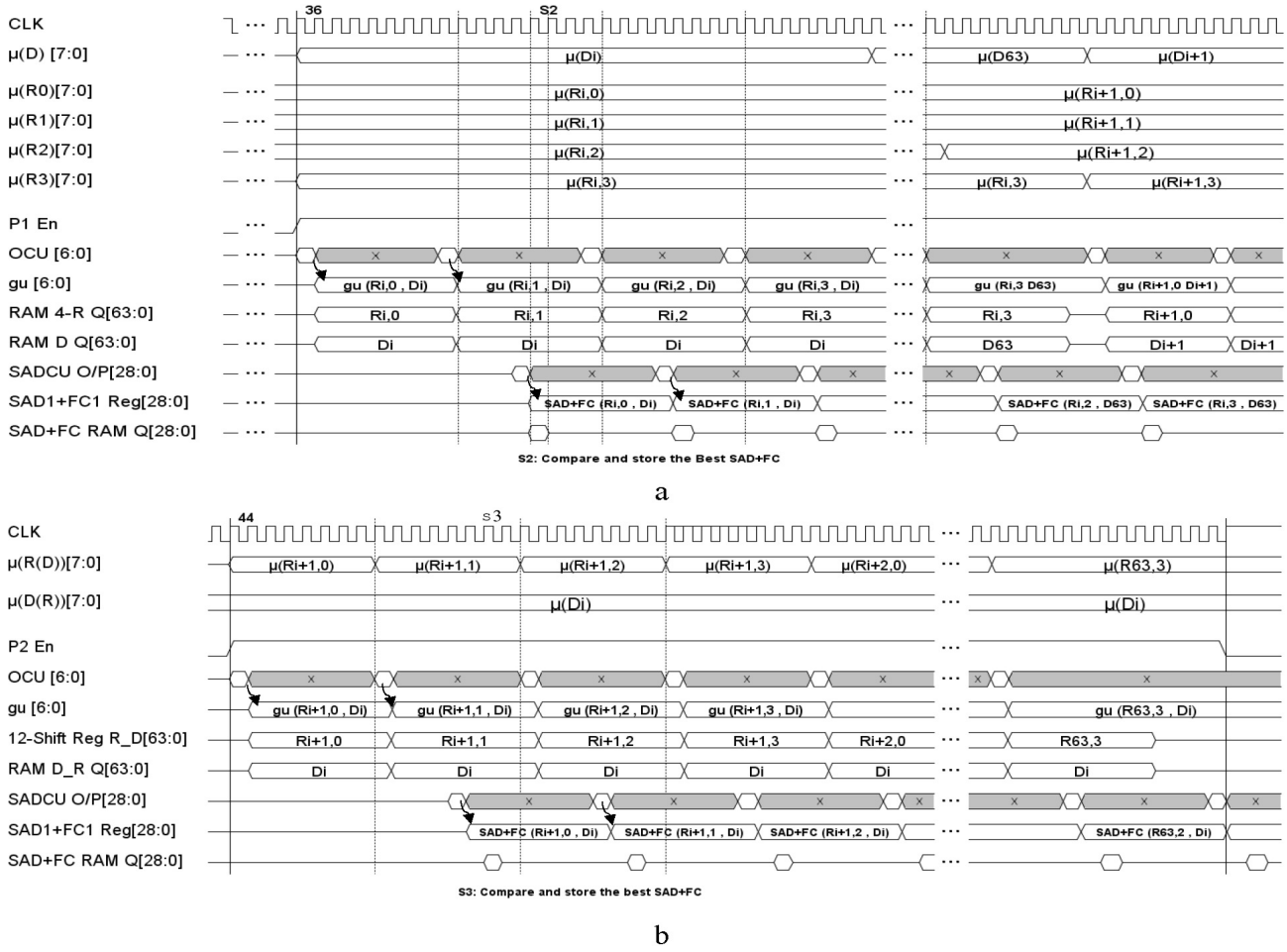

FIGURE 10. Timing diagrams of (a) P1 and (b) P2 processors.

elements (LEs) only. This corresponds to the hardware utilization of about 2%. On average the proposed architecture consumes power less than 0.5 W which makes it a very energy efficient hardware. Also, the design is capable of operating at a clock speed of 395 MHz. This is only possible by partitioning larger operations into smaller tasks via deep

FIGURE 11. Example of encoding 1024 × 1024 size Lena and Man images, where (a), (c) original images, (b), (d) reconstructed images.

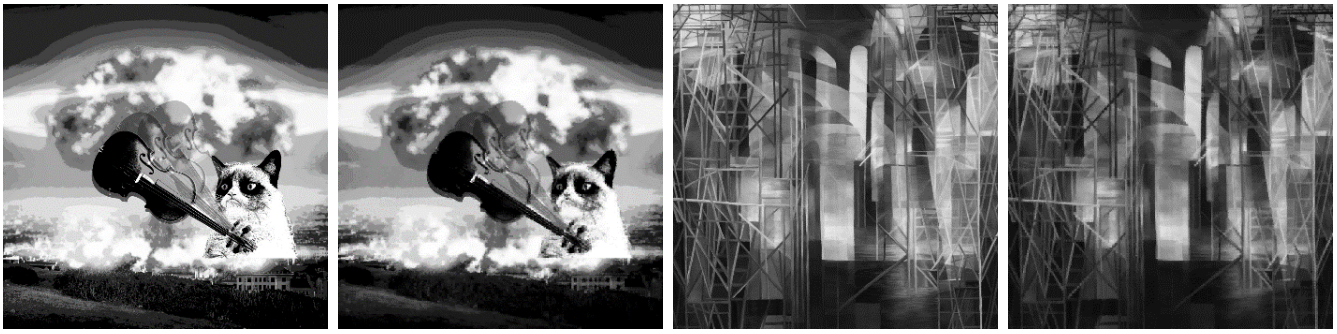

FIGURE 12. Example of encoding 1024 × 1024 size Cat Fiddle and Wooden Structure images, where (a), (c) original images, (b), (d) reconstructed images.

TABLE 1. Hardware utilization.

FPGA chip	Altera Stratix IV
Logic utilization	4799 logic elements
System frequency	395 MHz
Total clock cycles	4272128
Encoding time	10.8 ms (92 fps)
Power consumption	0.5 watt

pipelining as previously explained. Moreover, the use of more complicated operations like multiplication and division are avoided, which results in faster operating speed.

The running time can be also calculated based on the time needed to encode all 64 sub-images. As previously explained, a sub-image contains 64 non-overlapping R_i blocks of 16×16 pixels size each. Each R_i needs to be matched with 64 domain blocks. For each R_i, altogether $(64-i)$ domain blocks are compared in the first processor while the rest are matched concurrently in the second processor without any additional clock cycle. Since the second processor exploits the data fetched for the first processor, therefore, the run-time depends entirely on the time taken by first processor only. Also the number of comparison decreases continuously by 1 as R_i increases by 1. Thus, the total number of domain blocks compared for all $R_i's$ can be simply calculated using the sum of sequence formula $x(x+1)/2$, where x is the number of terms in the series, which is 64 in this case. Hence, the total number comparison for each sub-image is $(64 \times 65)/2$. Each comparison requires 32 clock cycles. Also 2 clock cycles are required before fetching another R_i (see Fig. 9). Therefore, processing one sub-image requires $((64 \times 65)/2) \times 32 + (64 \times 2) = 66688$ clock cycles. This corresponds to $64 \times 66688 \times T = 10.8$ ms encoding time for 1024×1024 size image.

The PSNR is used to evaluate the performance of the hardware. Altogether, four greyscale images, including the famous Lena image are used in the evaluation. These images are encoded in the hardware while the decoding is performed using MATLAB running on 3.4 GHz Intel core i7 processor and 8 GB of RAM. On average it takes 1 s to decode one image on this machine. The results for Lena and Man are shown in Fig. 11, while the results for another two images, Cat Fiddle and Wooden Structure, are shown in Fig. 12. In this case, the Lena image resulted in 30.3 dB PSNR. In all cases, the PSNR values ranged between 25.3 and 30.3 dB, and averaging at 27 dB. These figures are sufficiently adequate, considering that the compression ratio (CR) is well above 30. In this case the CR is calculated as the ratio between the original and compressed size images. In this case each range

FIGURE 13. Compression of Lena image of two different sizes: (a) 256 × 256 original image, (b) 256 × 256 reconstructed image, (c) 512 × 512 original image, (d) 512 × 512 reconstructed image.

TABLE 2. Comparison between the proposed and selected designs.

	Proposed	Vidya et al. [22]	Erra [20]	Samavi et al. [23]	Jackson et al. [33]	Panigrahy et al. [24]	Haque, et al. [19]	da Rosa Righi, et al. [17]	Ismail, et al. [21]
Platform	FPGA	FPGA	GPU	FPGA	APEX20K FPGA	FPGA	GPU	Two dual-core CPUs	GPU
Clock rate (MHz)	395	NA	300	240	32.05	75.52	1301	2900	700
Image size	1024 × 1024	128 × 128	256×256	256 × 256	512 × 512	256 × 256	1024 × 1024	512 × 512	256 × 256
Search type	Partial Search	Full search	Full search	Partial search	Searchless	Full search	Full search	Full search	Partial search
Range block size	Fixed (8× 8)	Fixed (4 × 4)	Fixed (4x4)	Fixed (8 × 8)	Flexible	Fixed (4x4)	Fixed (4x4)	Fixed (8× 8)	Fixed (8 × 8)
PSNR (dB)	30.3(Lena)	28.1(Lena)	-	25.3(Lena)	33.1[a](Lena)	32.7[a](Lena)	-	22 (Coliseum)	28(Lena)
Compression Ratio	34.1	4	5	26	12.8	5.1	4.1	-	-
Coding time	10.8 ms	14.1 s	1 s	0.8 ms	8.4 ms	531 ms	45 s	8.5 s	455 ms

[a] PSNR at threshold 24

block is compressed into 15 bits, comprising of 6-bit address of best domain block, 2-bit scale and 7-bit offset. Since each sub-image has 256 range blocks, therefore, the size of compressed image is 64 × 256 × 15 bits. This is equivalent a CR of 34.1.

For the completeness of the investigation, the performance of the proposed architecture is also compared with eight previous works as shown in Table 2. In addition to FPGA, results obtained from other platforms, namely standard dual-core CPU and GPU are also included in the comparison. With the exception of the work by Vidya *et al.* [22], clearly from this table, the CPU and GPU are generally much slower compared to FPGA. For instance the GPU runtime of 45 s reported by Haque *et al.* [19] is the slowest, and followed 8.5 s da Rosa Righi *et al.* [17], 1 s Erra [20], and 455 ms Ismail *et al.* [21]. In contrast the proposed method takes less than 10.8 ms to encode 1024 × 1024 size image. This corresponds to speed-ups of approximately 4167x, 787x, 93x and 42x compared to Haque *et al.* [19], da Rosa Righi *et al.* [17], Erra [20], and Ismail *et al.* [21] respectively. The significant increased in the speed-up is due to the ability of FPGA to process large amount of data in parallel. Generally this trend holds for most cases except the results published by Vidya *et al.* [22]. The 14 s runtime achieved by these authors is considerably slow by FPGA standard. In fact this runtime is the slowest among the FPGA based designs as evident from Table 2. Among the partial search scheme the fastest architecture is the result from Samavi *et al.* [23]. On average their design takes less than 1 ms compared to 10.8 ms in the proposed architecture. However Samavi *et al.* [23] measured the runtime using 256 × 256 size image compared to 1024 × 1024 in the proposed scheme. Therefore, the runtime from Samavi *et al.*'s [23] design is expected to increase dramatically when encoding a much bigger size image. Moreover the performance of Samavi *et al.*'s [23] design is slightly low

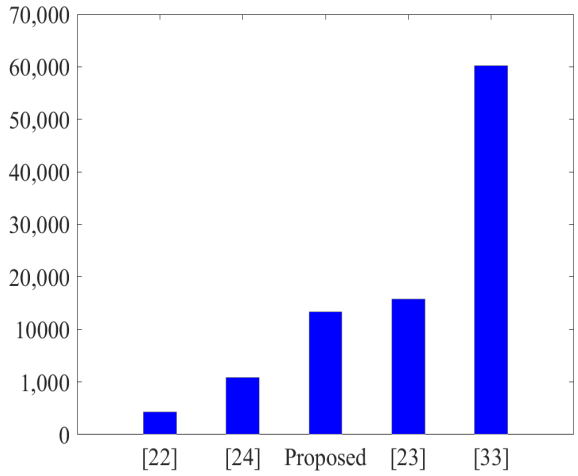

FIGURE 14. Memory usage comparison between proposed and current designs.

in terms of PSNR compared to the proposed scheme. In this case the proposed architecture resulted in PSNR of 30.3 dB compared Samavi *et al.* [23] 25.3 dB.

The same with 34.1 CR resulted by the proposed architecture which is the highest compared to 26 Samavi *et al.* [23], 12.8 Jackson *et al.* [33], 5.1 Panigrahy *et al.* [24], and 4 Vidya *et al.* [22]. The runtime is also computed using different sizes of Lena image in order to make an objective comparison with other designs that encoded different image sizes. The results are shown in Fig. 13. The proposed architecture takes approximately 2.7 ms and 0.67 ms to encode 512×512 and 256×256 size images, respectively. Comparing with existing designs using similar image sizes, these results indicates the proposed architecture is the fastest among the full-search and partial-search designs as evident from Table 2. In terms of PSNR, Samavi *et al.* [23] and Jackson *et al.* [33] registered slightly higher PSNR values compared with the proposed design. However, these designs resulted in much lower CRs. Also, the clock and encoding speeds from their designs are slightly inferior than the proposed hardware. Thus, the proposed architecture is more competitive in terms of image size support, runtime and CR.

For further evaluation, the proposed architecture is also compared in terms of memory usage, and the results are summarized in Fig. 14. From this figure, it can be seen that the proposed architecture requires reasonable memory size, and averaging at 13 kb compared to 434 bits for Vidya *et al.* [22], 1878 bits for Panigrahy *et al.* [24], 15.5 kb for Samavi *et al.* [23], and 59kb for Jackson *et al.* [33]. In terms of the hardware complexity, clearly, the proposed architecture is the third lowest compared to other designs. Jackson *et al.* [33] present a huge memory usage, since part of the image needs to be stored in internal memory prior to matching. Reference [23] is the second most complicated design that requires more than 15000 bits of memory. Clearly, in terms of the memory utilization, the proposed architecture is significantly less complex compared to Jackson *et al.* [33], but relatively the same compared to Samavi *et al.* [23]. Compared to Samavi *et al.* [23], however, the proposed hardware outperforms this design on both PSNR and CR. Meanwhile, Vidya *et al.* [22] and Panigrahy *et al.* [24] are the most efficient in terms of memory usage. However, the performance in term of speed and CR is the lowest as clearly shown in Table 2. In summary, the proposed design is relatively less complex but offers superiority in term of speed and CR.

In terms of flexibility, the proposed architecture can handle other images with some slight modifications to the hardware as long as their size are divisible by 128×128. For other image sizes the proposed architecture may require substantial modifications especially the memory organization, registers, data structures, and control units. We believe such modifications can easily be accomplished with experience digital designers or competent hardware architects.

VII. CONCLUSION

A new partial search FIC algorithm is presented in the hardware. The new design is evaluated using high resolution images of size 1024×1024. Neighborhood-search scheme is implemented to speed-up the runtime. Fixing the search space for each set of range blocks leads to an efficient parallel operation. The encoding time is reduced by a factor of 2 by performing matching operation using blocks constructed from range and domain blocks available from previous processing. The deep pipelining adopted in the design ensures high-speed operation at sustainable frequency of 395 MHz. On average the hardware takes approximately 10.8 ms to encode one image with PSNR 27 dB and CR 34.1. These promising results lead us to further develop the hardware for compressing color images. This work is actively being pursued in our laboratory and results will be published in our future paper.

ACKNOWLEDGMENT

The authors would like to thank Altera Corporation for donating DE4 boards under the Altera University Program.

REFERENCES

[1] Y. Fisher, *Fractal Image Compression: Theory and Application*. New York, NY, USA: Springer-Verlag, 1995.

[2] M. Barnsley and L. Hurd, *Fractal Image Compression*. Natick, MA, USA: A K Peters, 1993.

[3] A. E. Jacquin, "Image coding based on a fractal theory of iterated contractive image transformations," *IEEE Trans. Image Process.*, vol. 1, no. 1, pp. 18–30, Jan. 1992.

[4] S. Mozaffari, K. Faez, and M. Ziaratban, "Character representation and recognition using quad tree-based fractal encoding scheme," in *Proc. 8th Int. Conf. Document Anal. Recognit. (ICDAR)*, Aug./Sep. 2005, pp. 819–823.

[5] S. Kiani and M. E. Moghaddam, "A multi-purpose digital image watermarking using fractal block coding," *J. Syst. Softw.*, vol. 84, no. 9, pp. 1550–1562, 2011.

[6] C. E. Martin and S. A. Curtis, "Fractal image compression," *J. Funct. Program.*, vol. 23, no. 6, pp. 629–657, 2013.

[7] M. Polvere and M. Nappi, "Speed-up in fractal image coding: Comparison of methods," *IEEE Trans. Image Process.*, vol. 9, no. 6, pp. 1002–1009, Jun. 2000.

[8] Y. Zheng, G. Liu, and X. Niu, "An improved fractal image compression approach by using iterated function system and genetic algorithm," *Comput. Math. Appl.*, vol. 51, pp. 1727–1740, Jun. 2006.
[9] T. Kovács, "A fast classification based method for fractal image encoding," *Image Vis. Comput.*, vol. 26, pp. 1129–1136, Aug. 2008.
[10] N. Rowshanbin, S. Samavi, and S. Shirani, "Acceleration of fractal image compression using characteristic vector classification," in *Proc. Can. Conf. Elect. Comput. Eng.*, May 2006, pp. 2057–2060.
[11] Y.-G. Wu, M.-Z. Huang, and Y.-L. Wen, "Fractal image compression with variance and mean," in *Proc. Int. Conf. Multimedia Expo (ICME)*, vol. 1, Jul. 2003, pp. I-353-1–I-353-6.
[12] S. Zhu and X. Zong, "Fractal lossy hyperspectral image coding algorithm based on prediction," *IEEE Access*, vol. 5, pp. 21250–21257, 2017.
[13] M.-S. Wu, J.-H. Jeng, and J.-G. Hsieh, "Schema genetic algorithm for fractal image compression," *Eng. Appl. Artif. Intell.*, vol. 20, no. 4, pp. 531–538, 2007.
[14] W. Xing-Yuan, Z. Dou-Dou, and W. Na, "Fractal image coding algorithm using particle swarm optimisation and hybrid quadtree partition scheme," *IET Image Process.*, vol. 9, no. 2, pp. 153–161, 2015.
[15] J. H. Jeng, T. K. Truong, and J. R. Sheu, "Fast fractal image compression using the Hadamard transform," *IEE Proc.-Vis., Image Signal Process.*, vol. 147, no. 6, pp. 571–574, Dec. 2000.
[16] S. Zhu, S. Zhang, and C. Ran, "An improved inter-frame prediction algorithm for video coding based on fractal and H.264," *IEEE Access*, vol. 5, pp. 18715–18724, 2017.
[17] R. da Rosa Righi, V. F. Rodrigues, C. A. Costa, and R. Q. Gomes, "Exploiting data-parallelism on multicore and SMT systems for implementing the fractal image compressing problem," *Comput. Inf. Sci.*, vol. 10, no. 1, p. 34, 2016.
[18] X.-Y. Wang, Y.-X. Wang, and J.-J. Yun, "An improved no-search fractal image coding method based on a fitting plane," *Image Vis. Comput.*, vol. 28, no. 8, pp. 1303–1308, 2010.
[19] M. E. Haque, A. A. Kaisan, M. R. Saniat, and A. Rahman, "GPU accelerated fractal image compression for medical imaging in parallel computing platform," *CORR*, vol. abs/1404.0774, Apr. 2014.
[20] U. Erra, "Toward real time fractal image compression using graphics hardware," in *Proc. Int. Symp. Vis. Comput.*, 2005, pp. 723–728.
[21] B. M. Ismail, B. E. Reddy, and T. B. Reddy, "Cuckoo inspired fast search algorithm for fractal image encoding," *J. King Saud Univ.-Comput. Inf. Sci.*, vol. 30, no. 4, pp. 462–469, 2018.
[22] D. Vidya, R. Parthasarathy, T. C. Bina, and N. G. Swaroopa, "Architecture for fractal image compression," *J. Syst. Archit.*, vol. 46, pp. 1275–1291, Dec. 2000.
[23] S. Samavi, M. Habibi, S. Shirani, and N. Rowshanbin, "Real time fractal image coder based on characteristic vector matching," *Image Vis. Comput.*, vol. 28, pp. 1557–1568, Nov. 2010.
[24] M. Panigrahy, I. Chakrabarti, and A. S. Dhar, "Low-delay parallel architecture for fractal image compression," *Circuits, Syst., Signal Process.*, vol. 35, no. 3, pp. 897–917, 2016.
[25] A.-M. H. Y. Saad and M. Z. Abdullah, "Real-time implementation of fractal image compression in low cost FPGA," in *Proc. IEEE Int. Conf. Imag. Syst. Techn. (IST)*, Oct. 2016, pp. 13–18.
[26] A.-M. H. Y. Saad and M. Z. Abdullah, "High-speed implementation of fractal image compression in low cost FPGA," *Microprocess. Microsyst.*, vol. 47, pp. 429–440, Nov. 2016.
[27] R. E. Chaudhari and S. B. Dhok, "Review of fractal transform based image and video compression," *Int. J. Comput. Appl.*, vol. 57, pp. 23–32, Jan. 2012.
[28] K. P. Acken, M. J. Irwin, and R. M. Owens, "A parallel ASIC architecture for efficient fractal image coding," *J. VLSI Process. Syst. Signal Image Video Technol.*, vol. 19, no. 2, pp. 97–113, 1998.
[29] B. Hürtgen, P. Mols, and S. F. Simon, "Fractal transform coding of color images," *Vis. Commun. Image Process.*, vol. 2308, pp. 1683–1691, Sep. 1994.
[30] S. J. Woolley and D. M. Monro, "Optimum parameters for hybrid fractal image coding," in *Proc. Int. Conf. Acoust., Speech, Signal Process. (ICASSP)*, May 1995, pp. 2571–2574.
[31] B. Hürtgen and C. Stiller, "Fast hierarchical codebook search for fractal coding of still images," in *Proc. Berlin-DL Tentative*, 1993, pp. 397–408.
[32] H. T. Chang and C. J. Kuo, "Adaptive schemes for improving fractal block coding of images," *J. Inf. Sci. Eng.*, vol. 15, pp. 11–25, Jan. 1999.
[33] D. J. Jackson, H. Ren, X. Wu, and K. G. Ricks, "A hardware architecture for real-time image compression using a searchless fractal image coding method," *J. Real-Time Image Process.*, vol. 1, no. 3, pp. 225–237, 2007.

ABDUL-MALIK H. Y. SAAD was born in Jeddah, Saudi Arabia, in 1983. He received the B.Sc. degree (Hons.) in computer engineering from Hodeidah University, Hodeidah, Yemen, in 2006, and the M.Sc. degree in electronic systems design engineering and the Ph.D. degree in the digital systems field from Universiti Sains Malaysia (USM) in 2014 and 2018, respectively. He is currently a Lecturer at the School of Electrical and Electronic Engineering, USM.

MOHD Z. ABDULLAH received the B.App.Sc. degree in electronics from Universiti Sains Malaysia (USM) in 1986 and the M.Sc. degree in instrument design and application and the Ph.D. degree in electrical impedance tomography from the University of Manchester Institute of Science and Technology, U.K., in 1990 and 1993, respectively. He was a Test Engineer with Hitachi Semi-Conductor, Malaysia. He is currently a Professor and the Director of the Collaborative Microelectronic Design Excellence Centre, USM. He has published numerous research articles in international journals and conference proceedings. His research interests include microwave tomography, digital image processing, computer vision, and ultra-wideband sensing. One of his papers received the Senior Moulton Medal for the best article published by the Institute of Chemical Engineering in 2002.