IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received August 20, 2020, accepted September 9, 2020, date of publication September 21, 2020, date of current version October 15, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.3025344

Resource-Efficient Image Buffer Architecture for Neighborhood Processors

MAJIDA KAZMI[1], ARSHAD AZIZ[2], HASHIM RAZA KHAN[1,4], SAAD AHMED QAZI[1,4], AND LAMPROS K. STERGIOULAS[3], (Member, IEEE)

[1]Faculty of Electrical and Computer Engineering, NED University of Engineering and Technology, Karachi 75270, Pakistan
[2]Department of Electrical Engineering, PNEC, National University of Sciences and Technology (NUST), Karachi 75350, Pakistan
[3]Faculty of IT and Design, The Hague University of Applied Sciences, 2521 EN The Hague, The Netherlands
[4]Neurocomputation Lab, National Center of Artificial Intelligence, Karachi 75270, Pakistan

Corresponding author: Lampros K. Stergioulas (l.stergioulas@hhs.nl)

This research received funding from the Neuro-Computation Lab, National Centre of Artificial Intelligence, NED University of Engineering and Technology, Karachi, Pakistan (PSDP.263/2017-18).

ABSTRACT Neighborhood image processing operations on Field Programmable Gate Array (FPGA) are considered as memory intensive operations. A large memory bandwidth is required to transfer the required pixel data from external memory to the processing unit. On-chip image buffers are employed to reduce this data transfer rate. Conventional image buffers, implemented either by using FPGA logic resources or embedded memories are resource inefficient. They exhaust the limited FPGA resources quickly. Consequently, hardware implementation of neighborhood operations becomes expensive, and integrating them in resource constrained devices becomes unfeasible. This paper presents a resource efficient FPGA based on-chip buffer architecture. The proposed architecture utilizes full capacity of a single Xilinx Block-RAM (BRAM36 primitive) for storing multiple rows of input image. To get multiple pixels/clock in a user defined scan order, an efficient duty-cycle based memory accessing technique is coupled with a customized addressing circuitry. This accessing technique exploits switching capabilities of BRAM to read 4 pixels in a single clock cycle without degrading system frequency. The addressing circuitry provides multiple pixels/clock in any user defined scan order to implement a wide range of neighborhood operations. With the saving of 83% BRAM resources, the buffer architecture operates at 278 MHz on Xilinx Artix-7 FPGA with an efficiency of 1.3 clock/pixel. It is thus capable to fulfill real time image processing requirements for HD image resolution (1080 × 1920) @103 fps.

INDEX TERMS Image buffering, neighbourhood operations, FPGA, embedded block RAM (BRAM), memory accessing technique, resource constrained applications.

I. INTRODUCTION

Recent technological advancements open new avenues in mobile and portable imaging devices. These devices are progressively becoming more compact with emphasis on autonomous power supply [1]. In such resource (area, power) constrained devices, it becomes very challenging to integrate various computationally intensive low-level image processing units, which are indispensable for improving the quality of captured images. The neighborhood operations are widely used low level image processing operations, to enhance the visual quality of captured images [2], [3]. These operations use a two-dimensional window that shifts pixel by pixel, in a fixed scan order over the complete image. At each shift operation, image pixels within the window are processed either by a linear or non-linear operation [4]. Although these neighborhood operations are theoretically simple, their hardware implementation is computationally expensive and memory intensive [5]. A (N×N) window processes (N×N) neighborhood input pixels per output pixel, thus leading to high pixel data transfer rate of N^2 pixels, between external memory and processor unit that eventually requires a large memory bandwidth.

For the parallel hardware implementation of such Neighborhood Image Processors (NIPs), the use of hardware devices such as FPGAs is gaining momentum due to their

The associate editor coordinating the review of this manuscript and approving it for publication was Mehul S Raval.

parallelism, power efficiency and re-configurability [5]. Inherently, the parallel execution capabilities of FPGAs are particularly suitable to exploit the tempo-spatial parallelism of NIPs. Therefore, image processing applications can be greatly accelerated by placing its computationally expensive neighborhood operations on FPGA. However, the high data transfer rate over the limited memory bandwidth (usually 32 bits per clock [6]) is a main barrier in accelerating the performance of parallel NIP on FPGA. For this reason, the frequently accessed pixel data is temporary buffered on FPGA to reduce the data transfer rate and thus external memory bandwidth requirements [7]–[10].

The conventional on-chip data buffers can be divided into two main groups: Partial Buffers (PBs) [7]–[9], [9]–[11] and Full Buffers (FBs) [5], [7], [12], [20], [21]. The PBs [7]–[11] store only partial input image pixels, and therefore they are resource efficient but their bandwidth requirement increases proportionally with the window (kernel) size. Therefore, fixed and limited memory bandwidth restricts their capacity to up-scale for larger NIPs [6]. On the other hand, the FBs store full image rows which are required for calculating an output pixel. It provides an efficiency of 1 clock/pixel (in row scan order only) at a constant external memory bandwidth of 1 pixel/clock. However for buffering the full rows, it utilizes considerable FPGA memory resources in the form of FPGA Configurable Logic Blocks (CLBs) [7]–[10] or embedded Block RAMs (BRAMs) [13]. Therefore, unlike these conventional FB and PB schemes, it is of great interest to provide a low cost image buffering solution without elevating the permissible memory bandwidth limit for implementing parallel NIPs in a resource constrained environment such as portable and mobile imaging devices [14]–[17].

The rest of the paper is organized as follows. Section II discusses existing relevant work. Section III explains the methodology. Section IV presents the timing analysis and Section V demonstrate the implications of our work on practical image processing applications. Section VI discusses the results and makes a comparison with existing methods. Section VII summarises the conclusion of this work.

II. RELEVANT WORK

A comprehensive literature review on FPGA based image buffers for NIP, revealed that most of the reported full buffers consume significant resources in terms of CLB based Distributed RAM (DRAM) [7]–[10] or sequential BRAMs [12], [13], [18] to fulfill the memory requirement of NIP. Bosi *et al.* [7] have proposed high speed convolver processors on FPGA. In order to execute parallel computations, they employed on-chip data buffering. They proposed two buffering schemes, i.e. FBs and Single Window Partial Buffers (SWPBs). In FB, N-1 full rows of input image were stored in row buffers for (N×N) convolver. They configured LUTs of Xilinx CLBs to implement these row buffers. It consumed a lot of logic resources, especially for higher image resolutions. Alternately, they implemented a SWPB scheme which consumes fewer Xilinx CLBs but at the cost of extensive utilization of external memory bandwidth. The resulting external memory bandwidth requirement for SWPB is N pixels/clock for a (N×N) convolver, leading to sharply raised external memory bandwidth for larger kernels.

Cardells-Tormo and P.-L [8] have proposed three variants of the conventional PB method and compared them with other available existing methods for FPGA resource utilization against throughput (clock/pixel). The three proposed PB schemes were Column Major Moving Window, Row Major Moving Window and Moving Window with Rotation Stage. These three architectures were realized by using Shift Registers and on-chip SRAM. The major drawback of these architectures was their extensive memory bandwidth utilization. Also, they occupied more resources as compared to the SWPB method. Only the third architecture i.e. Moving Window with Rotation Stage achieved 1, the ideal clock/pixel rate [8]. Joginipelly and Charalampidis [9] proposed a variant of [8] for separable convolver kernel with a reduction in external memory bandwidth requirement. Zhang *et al.* [10] have proposed a hybrid buffering approach termed as Multi Window Partial Buffering (MWFB). It was based on shift registers matrix of sizes larger than the convolver kernel. The same number of input pixels were shared among adjacent convolvers, which results in lowering the memory bandwidth requirement. This method was a compromise between the SWPB and FB methods described in [7]. The reported throughput was 1 clock per pixel at reduced FPGA resource utilization as compared to the FB method. However, this buffering scheme may not be feasible for larger NIPs in case of limited available bandwidth.

Besides the above discussed CLB based image buffering schemes, a few other embedded BRAM based image buffering schemes have also been reported in literature [6], [12], [13], [18], [20], [21]. Liang *et al.* [12] theoretically proposed a full parallel buffering scheme by using BRAMs. However, in this work BRAM has been employed as FIFOs and in this way, it was not able to utilize the full storage capacity of BRAM. Schmidt *et al.* [13], Moore *et al.* [18], Licciardo *et al.* [20], [21] have also implemented FB by using sequential BRAMs. Cao *et al.* [6] proposed a fast buffering scheme which is based on a horizontal buffer and a vertical buffer, equivalent to the BRAM based row buffers and shift registers of FB scheme respectively. All of these BRAM based approaches result in inefficient utilization of a large number of partially filled BRAMs.

All of the above discussed conventional on-chip buffers were either configured CLBs as DRAM or used dedicated embedded BRAMs sequentially. The DRAM based buffering architectures require large number of CLBs and the interconnection requires significant routing resources. Therefore, overall resource consumption of these buffering schemes is considerably high. On the other hand, the BRAM based buffers [6], [12], [13], [18], [21] are also resource inefficient as they occupy one BRAM per image row. For example, the frequently used 320×256 gray scale images require only

2.5 Kbits space to store a single row of 320 gray scale pixels (320∗8 bits = 2.5 Kbits) which is just 6.9 % capacity of latest Xilinx BRAM whereas 93% BRAM capacity remains underutilized. Similarly 640 × 480 gray scale image requires 5 Kbits space to store a single row of 640 gray scale pixels (640∗8 bits = 2.5 Kbits) which is just 13.8 % capacity of latest Xilinx BRAM whereas 86 % capacity remains underutilized. Therefore, in most of these buffers, a major percentage of BRAM capacity remains underutilized. All of these resource inefficient implementations exhaust limited hardware resources on a FPGA device quickly, making NIP expensive and thus unfeasible for resource constrained applications. Moreover, all of the above reported implementations are restricted to provide pixels in only row scan order which is required by limited NIPs. To overcome these limitations, this work proposes a FPGA based image buffering solution with saving of up to 83% valuable BRAM resources. Operating at high speed of 278 MHz, it meets real time image processing requirements for HD image resolution (1080 × 1920) @103 fps. At the same time, the proposed architecture has an added advantage of accessing pixels in different user defined scan orders for implementing a wide range of parallel NIP in resource constrained applications.

III. METHODOLOGY

This work presents an efficient BRAM based image buffering architecture on FPGA with an additional pre-fetching attribute. For proof of concept, a 7 × 7 NIP has been selected to process an 8-bits gray scale image of size 128 × 128. The 7 × 7 NIP is chosen because mostly up to a 7 × 7 kernel provides a reasonable tradeoff between noise removal and feature preservation while enhancing the quality of images.

The block diagram of the proposed buffer architecture is depicted in Fig. 1. It is comprised of one BRAM [27], Address Generator Module (AGM), Clocking Circuitry (CC), Clock Selector (CLK SEL) and a set of eight Registers (R0-R7). The design is initialized by a ''START'' signal. It enables AGM for fetching 32 Kbits packed image pixels from external memory and writing it on a single BRAM. The proposed 32 Kbits size of buffer fulfills requirement of NIP as well as fully occupies a single BRAM36 primitive. Write operation is performed via 32-bit wide data input bus of port A i.e. DI-A [31: 0]. Once the pixels data is completely written on BRAM, the AGM starts reading data from BRAM via 8-bit wide data output bus of both ports A and B i.e. DO-A/B [7: 0] in a user defined scan order. It accesses 4 pixels/CLK from each port (A and B) by using an efficient duty-cycle based pixels accessing technique. Therefore, it reads altogether 8 pixels/CLK from both ports (A and B) and temporarily stores them in eight Registers (R0-R7) for synchronization purpose. These registers are operating at ×2 CLK to deliver 7 out of 8 valid pixels to the 7 × 7 NIP simultaneously per CLK cycle. After completely reading and reusing the stored input pixels data by NIP, the BRAM is re-loaded with the next 32 Kbits pixels data, and continues further processing. Our main contributions in this work include:

- Utilize full storage capacity of a single 32 Kbits BRAM for storing multiple rows of input image. It eliminates the requirement of partially filled multiple BRAMs.
- Read 8 pixels from both ports (A and B) of a true dual port BRAM within a single clock cycle without using ×4 over clocking. A mechanism is devised to exploit switching capabilities of BRAM. It effectually utilizes a combination of two 25% duty-cycle clocks with reduced ON time, for reading 4 times per clock (CLK) from each port of a single BRAM with conforming its timing constraints.
- Access pixels in a user defined scan order from different BRAM locations. An efficient addressing circuitry is devised to provide strongly patterned pixels for implementing a wide range of NIPs.

A. PIXELS FETCHING AND BUFFERING IN BRAM WITH DIFFERENT PORT ASPECT RATIO

The proposed design fetches input image pixels from secondary memory elements available on FPGA boards / imaging devices as an external resource in form of SRAM or DRAM. These external memories are generally available with 32-bit bandwidth [6]. For minimizing the latency of data fetching operation, the design fetches the data from external memory in standard packed pixels format. It writes four packed 8-bit gray scale pixels (4*8 =32 bits) per clock cycle on consecutive BRAM locations, completely utilizing the standard 32-bit external memory bandwidth. For accessing multiple buffered pixels per clock cycle in a requisite scan order, the design reads pixel wise data (8 bits) per clock cycle at a higher clock rate from multiple BRAM locations.

The desired design functionality is achieved by using BRAM as a True Dual Port (TDP) RAM and configured with different port aspect ratio for read and write operations. This configuration allows to control the BRAM access for read and write operations independently. For writing packed pixels (4 pixels/clock), the write width of BRAM for port A is set to 32 bits i.e. DI-A [31: 0] and for reading pixel wise data from both ports, the read width is set to 8-bit i.e. DO-A/B [7: 0] for port A and B respectively. Since, the storage capacity of BRAM remains same i.e. 32 K bits for both write and read port as shown in Fig. 2, therefore depth of BRAM is to be different with respect to the reads and writes width of ports. From perspective of write port, BRAM is 32 bit wide and 1024 locations deep (total capacity is 32*1024=32,684 bits) and from perspective of read port, BRAM is 8 bit wide and 4096 locations deep (total capacity is still $8 \times 4096 = 32,684$ bits). Width of address bus ADDR for both ports is selected as 12 bits so that it can support wider of the two ports depths, i.e. ADDR A/B [11: 0] is used for read addresses and the same is partially used ADDR A/B [11: 2] for write addresses.

B. PIXELS SCANNING IN DIFFERENT SCAN ORDERS

The parallel implementation of NIP requires multiple pixels concurrently in a fixed scan order. The pattern of pixels

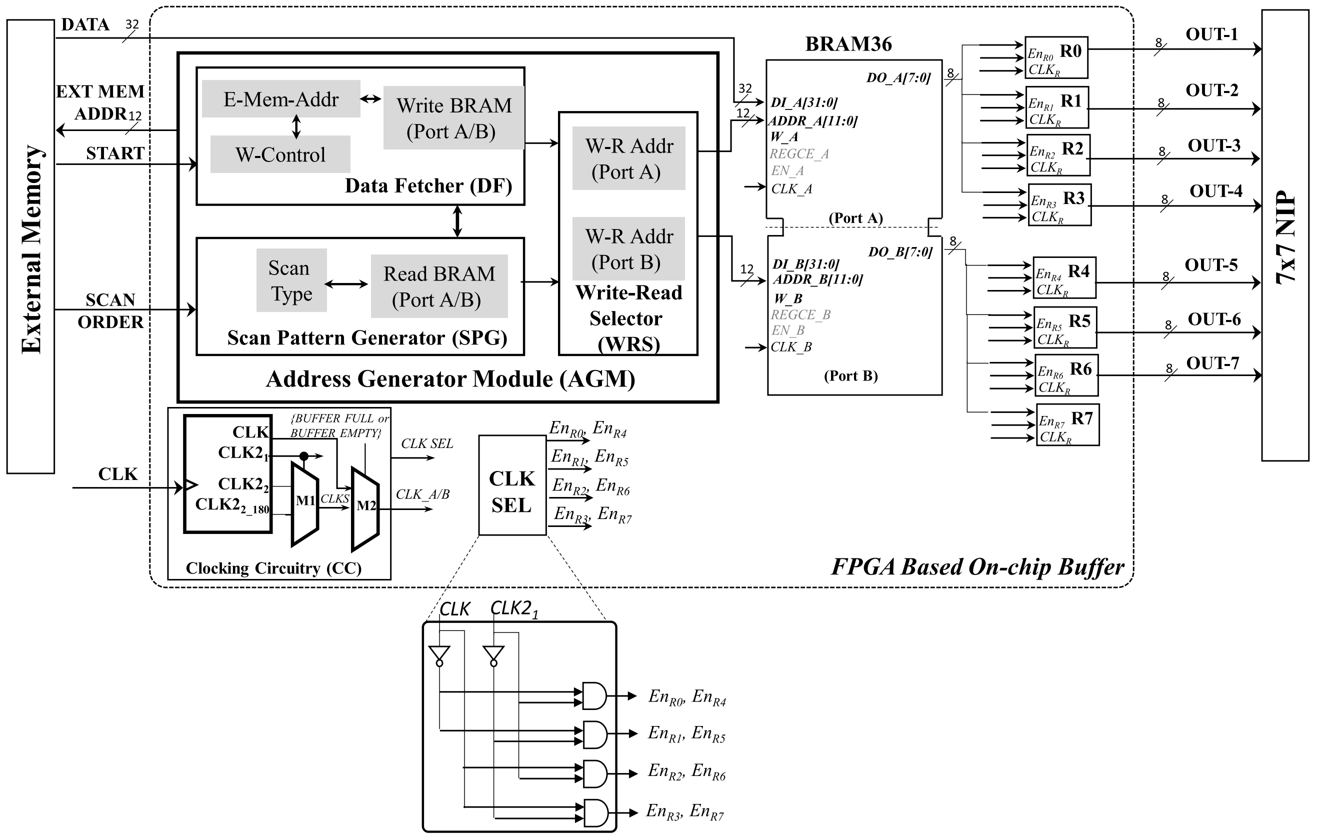

FIGURE 1. Proposed on-chip buffer architecture on FPGA.

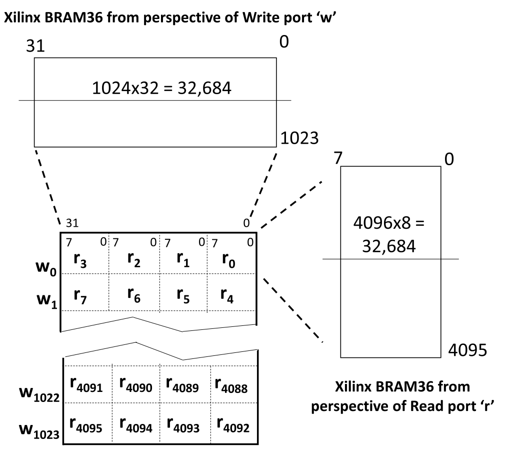
Xilinx BRAM36 from perspective of Write port 'w'
1024x32 = 32,684
4096x8 = 32,684
Xilinx BRAM36 from perspective of Read port 'r'

FIGURE 2. Memory mapping of BRAM36 with different W/R port widths.

depends on the NIP. Fig. 3 illustrates scan patterns such as row scan, column scan, row prime scan, or column prime scan, which are widely used by different NIP applications [32]. Our proposed design is capable to deliver pixels in any of the above-mentioned scan patterns. A customized addressing module i.e. AGM is devised. It fetches 4 packed pixels per clock from consecutive rows of input image, and write these packed pixels on consecutive write locations of BRAM (w_0, $w_1 \cdots w_{1023}$). Once, BRAM is filled completely, it starts reading pixel wise data in any of the above-mentioned scan order from corresponding read locations of BRAM (r_0, r_1, $\cdots$ r_{4095}). The AGM comprises of three sub-modules, Data Fetcher (DF), Scan Pattern Generator (SPG) and Write Read Selector (WRS). The DF fetches 32 Kbits pixels data in terms of 4 packed pixels per clock from consecutive input image rows and stores it in 32 Kbits BRAM. It also keeps a track of address location for next pre-fetching operation. Once 32 Kbits image data is fully buffered in BRAM, it becomes available to NIP for processing. The design reads these buffered pixels data in a user defined scan order by using SPG. This module is flexible to support different scan patterns selected by user at SCAN ORDER pin. Pseudo code for SPG for an exemplary n×n block of pixels data is given below.

C. MULTIPLE PIXELS ACCESSING

The chosen size of NIP is 7×7 that requires 7 pixels per CLK. However, due to the synchronous behavior of BRAM, the design can access each of its port only once per CLK. Therefore, to get 7 pixels/CLK, conventionally multi-rated memory accessing technique is used to read from each port of TDP BRAM at 4 times higher clock speed (×4 CLK) than rest of the system [23]. Though this method can save the number of BRAMs but at the expense of reducing overall

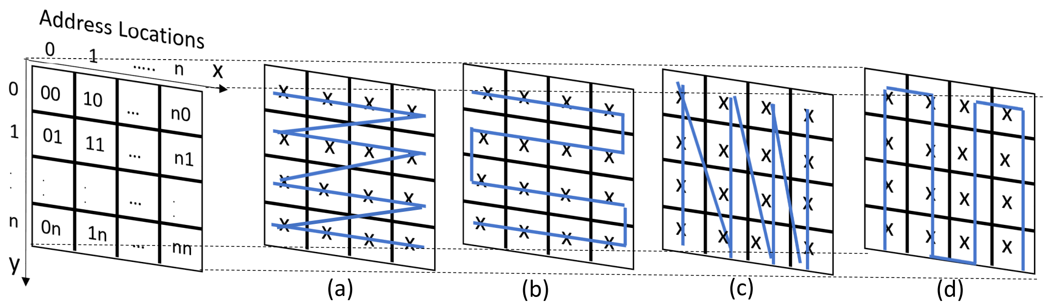

FIGURE 3. Scan Orders: (a) Row Scan, (b) Row Prime Scan, (c) Column Scan, (d) Column Prime Scan.

```
Procedure Pseudo code for Scan orders
Input: Operation, Scan order
Output: ADDR
        For Column, Column Prime: a=x; b=y:
        For Row, Row Prime: a=y; b=x :
 1: if (Pre − fetching) then
 2:     for $y = 0 ; y \leq (n-1) ; y = y+1$ do
 3:         ADDR<=y
 4:     end for
 5: else if (Buffering) then
 6:     for $a = 0 ; a \leq (n-1) ; a = a+1$ do
 7:         If ($a = odd$) then
 8:             for $b = 0 ; b \leq (n-1) ; b = b+1$ do
 9:                 ADDR <=ab
10:             end for
11:         else if ($a = even$) then
12:             for $b = (n-1) ; b \leq 0 ; b = b-1$ do
13:                 ADDR <=ab
14:             end for
15:         end for
16:     end if
17: end if
```

system frequency by the same factor of $\frac{1}{4}$. Therefore, instead of $\times 4$ CLK, we used a combination of two $\times 2$ CLKs with 25% duty cycle to switch the TDP BRAM, 4 times in a single CLK. It results in improving frequency twice as compared to the conventional multi rated technique.

1) DUTY CYCLE BASED BRAM ACCESSING TECHNIQUE

Timing diagram of the proposed design with the proposed duty cycle-based memory accessing technique is shown in Fig.4. It is comprised of a system clock CLK, and clean daughter clocks ($\times 2$ variants i.e. $CLK2_1$, $CLK2_2$ and $CLK2_{2-180}$). The daughter clocks are generated by using Clock Management Tile (CMT) as Phase Lock Loop (PLL). PLL uses CLK as input and efficiently generates three glitch free clock variants having different duty cycles and phase shifts. The clock CLK and $CLK2_1$ has 50% duty cycle whereas $CLK2_2$ and $CLK2_{2-180}$ clocks has 25% duty-cycle. Usually, BRAM is triggered at positive edge of 50% duty cycle clock i.e. $CLK2_1$. During OFF time of $CLK2_1$, it remains idle. This OFF time of $CLK2_1$ is much higher than the hold time of BRAM. We devised a 25% duty cycle-based memory accessing technique [30] to utilize this OFF time. Combination of two 25% duty cycle-based clocks i.e. $CLK2_2$ and $CLK2_{2-180}$ are deployed with 0^o and 180^o phase shifts respectively. These clocks have reduced 25% ON time and prolonged 75% OFF time. First, BRAM is switched at positive edge of first clock $CLK2_2$. During its prolonged OFF time, BRAM is switched once again at positive edge of second clock $CLK2_{2-180}$. A multiplexer (M1) selects between $CLK2_2$ and $CLK2_{2-180}$ to create a switching clock CLKS. This CLKS switches BRAM twice in a single clock period of $CLK2_1$, to get an additional output of BRAM within the rated time limits.

For better understanding, we numerate our memory accessing technique on our target Xilinx Artix-7 (XC7A35T) FPGA device with speed grade -3 [34]. Based on operating frequency of our design i.e. 278 MHz, time period of the CLK is 3.597 ns (1/278 MHz) whereas time period of other variants of CLK i.e. $CLK2_1$, $CLK2_2$ and $CLK2_{2-180}$ is 1.7985ns. The clock CLK21 with 50% duty cycle has 0.89925 ns ON time and 0.89925 ns OFF time whereas $CLK2_2$ and $CLK2_{2-180}$ clocks with 25% duty-cycle have 0.45 ns ON time and 1.349 ns OFF time. $CLK2_{2-180}$ is 180^o phase shifted with respect to the $CLK2_1$ for providing an additional positive edge of clock during the OFF time of $CLK2_2$. The time difference between the two consecutive positive edges of $CLK2_2$ and $CLK2_{2-180}$ is 0.899 ns.

For proper switching of BRAM, the above-mentioned clock timings must conform BRAM timing constraints imposed by three parameters: Setup time (T_{setup}), hold time (T_{hold}) and clock to out time ($T_{clock-to-out}$) [24]. The T_{setup}, T_{hold}, and $T_{clock-to-out}$ of BRAM for Artix-7 is 0.45 ns, 0.31 ns and 0.64 ns respectively [34]. Following conditions must be fulfilled for proper switching:

- The 0.45ns ON time of $CLK2_2$ and $CLK2_{2-180}$ clocks must remain greater than both the T_{setup} and T_{hold} i.e. 0.45 ns and 0.31 ns respectively.
- The 0.899 ns time difference between the two consecutive positive clock edges should remain higher than $T_{clock-to-out}$ i.e. 0.64 ns.

- The resultant output of CLKS has a delay (M1 internal delay) to meet the T_{setup} time for a valid input data.

This way, by using CLKS, BRAM is switched 4 times per CLK to read 8 pixels from both ports A and B (4 pixels per port) and at the same time meet all the timing requirements, without using conventional $\times 4$ multi-rated clocking.

IV. TIMING ANALYSIS

To systematically understand the overall operation and synchronization of the proposed design, we elaborate the timing analysis of each operation.

A. FETCHING PACKED PIXELS DATA (BRAM DATA WRITING)

The timing analysis for fetching packed pixels data is illustrated in Fig. 5. It reads 4 packed pixels wise data from external memory and writes it on BRAM i.e. 4 gray scale pixels per CLK. The EXT MEM ADDR [12: 0] is address bus for external memory whereas W-A, ADDR A [11: 0] and DI-A [31: 0] are the write bus, address bus and input data bus of BRAM for port A. The design reads consecutive image rows from external memory, starting from first packed pixel of first row $R_{1-(1to4)}$ (i.e. R_{1-1} R_{1-2} R_{1-3} R_{1-4} where R_{r-p} represents pth pixel of row rth row) at point 0 and write this data on BRAM via port A at point 1. It continues to write till last packed pixel of Row 32 $R_{32-(125to128)}$ (i.e. R_{32-125} R_{32-126} R_{32-127} R_{32-128} registered in port A at point 1024 (in 1024 clock cycles) as shown in Fig.5.

B. ACCESSING PIXEL WISE DATA IN COLUMN-SCAN PATTERN (BRAM DATA READING)

The timing analysis for data accessing (from port A) by NIP is illustrated in Fig. 6. It shows that how the proposed buffer design delivers 7 valid output pixels/CLK from a single BRAM. Once the first data fetching cycle is completed, the design starts reading buffered image pixels from both ports of BRAM at CLKS and thus delivers 8 output pixels in one CLK cycle. For reading pixels in column scan order, address of the first pixel i.e. R_{1-1} (where R_{r-p} represents pth pixel of row rth row) is applied to address bus ADDR-A [11: 0] of port A at the positive edge of CLKS at point 0. This pixel is delivered via DO-A [7: 0] at the next positive edge of CLKS at point 1. This output pixel is registered in R0 at point 2. The R0 is enabled by En_{R0} signal at positive $CLK2_1$ cycle and negative CLK cycle. This pixel data remains valid in R0 for a complete CLK cycle as shown in Fig. 7. At the same time at point 1, next address is applied to port A i.e. address of R_{2-1} and it is delivered at point 2. This output is registered in R1 at point 3. The R1 is enabled by En_{R1} signal at negative $CLK2_1$ and CLK cycle. This pixel data remains valid in R1 for a complete CLK cycle. Similarly, next pixel R_{3-1} is delivered at the next positive edge of CLKS at point 3. This output is registered in R2 at point 4. The R2 is enabled by En_{R2} signal at positive $CLK2_1$ and CLK cycle. This pixel data also remains valid in R2 for a complete CLK cycle. Then, the next pixel R_{4-1} is delivered at the next positive edge of CLKS at point 4. This output is registered in R3 at point 5. The R3 is enabled by En_{R3} signal at negative $CLK2_1$ cycle and positive CLK cycle. This pixel data also remains valid in R3 for a complete CLK cycle as shown in Fig. 6.

The data flow for port B is similar to port A. The first output pixel R_{5-1} of port B is registered in R4 at positive $CLK2_1$ cycle and negative CLK cycle at point 2. Second output pixel R_{6-1} is registered in R5 at negative clock cycle of both $CLK2_1$ and CLK at point 3. Third output pixel R_{7-1} is registered in R6 at positive clock cycle of both $CLK2_1$ and CLK at point 4. Forth output pixel R_{8-1} is registered in R7 at negative $CLK2_1$ cycle and positive CLK cycle at point 5. The flow of all signals which systematically enables these eight registers (R0-R7) are tabulated in Table 1.

TABLE 1. BRAM output pixels read by Registers (R0-R7).

BRAM Port	CLK	$CLK2_1$	CLKS	Active Enable Signal	Active Register	Pixels Data
Port A	0	1	1	En_{R0}	R0	R_{1-1}
	0	0	1	En_{R1}	R1	R_{2-1}
	1	1	1	En_{R2}	R2	R_{3-1}
	1	0	1	En_{R3}	R3	R_{4-1}
Port B	0	1	1	En_{R4}	R4	R_{5-1}
	0	0	1	En_{R5}	R5	R_{6-1}
	1	1	1	En_{R6}	R6	R_{7-1}
	1	0	1	En_{R7}	R7	R_{8-1}

In this way the NIP gets all 7 valid pixel (our design produce 8 outputs per clock but NIP uses only 7 pixels data) simultaneously in one complete system clock CLK from both ports of BRAM.

The complete pre-fetching and pixel accessing operations for a 128×128 input image is shown in Fig. 7. At point 0 our system is initialized with a valid START signal and starts fetching first 32 Kbits image data from external memory into BRAM. When BRAM is completely filled with pixels data, it generates a BRAM FULL signal. At point 1, the system starts delivering buffered data to NIP in a user-defined scan order until BRAM is empty. At point 2 it generates a BRAM EMPTY signal which restarts fetching process again for the next 32 Kbits chunk of image data. This process continues until the complete image is fetched from external memory and provides a complete buffered output to NIP at point 10. Once the complete processing is done it then restarts the system for the next image available in the external memory.

V. DEMONSTRATION OF PROPOSED IMAGE BUFFERING ARCHITECTURE FOR PRACTICAL APPLICATIONS

We have proposed a compact image buffering solution on FPGA platform. This solution is suitable for implementing on-chip buffers for several practical applications. To evaluate its effectiveness in practical designs, let us consider three exemplary applications from literature [23], [24]. Table 2 shows the implementation outcomes in terms of number of BRAM required by our design to buffer pixels data for the chosen applications along with its total power

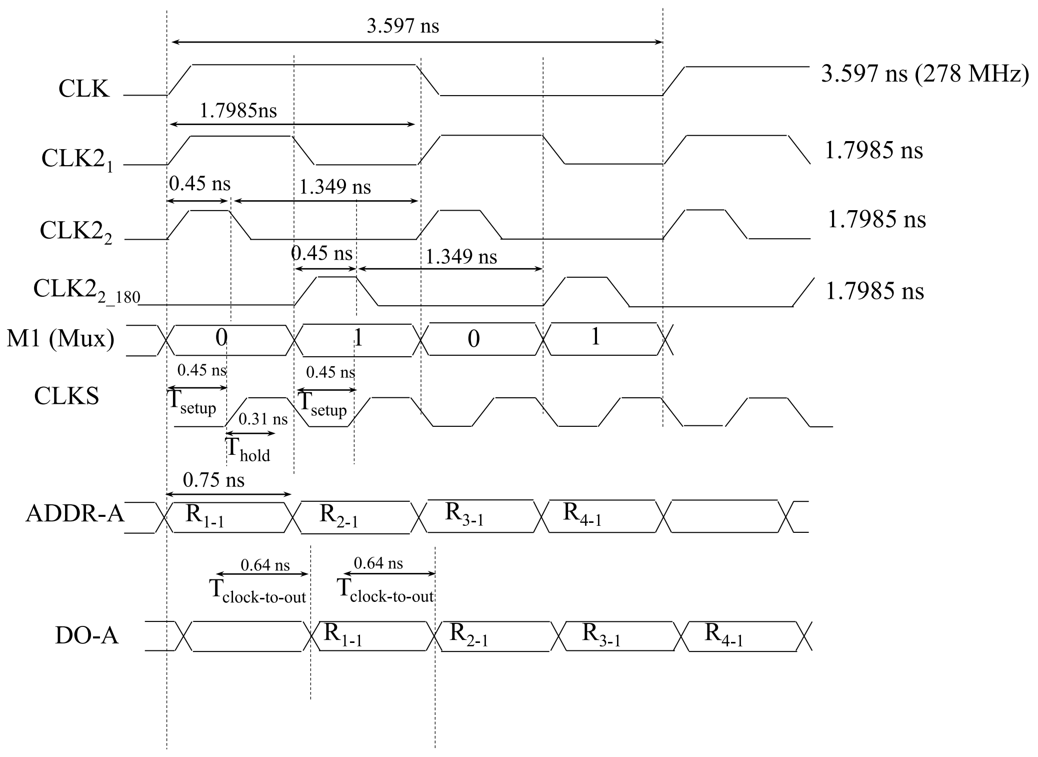

FIGURE 4. Numerating accessing technique for our design on Artix-7.

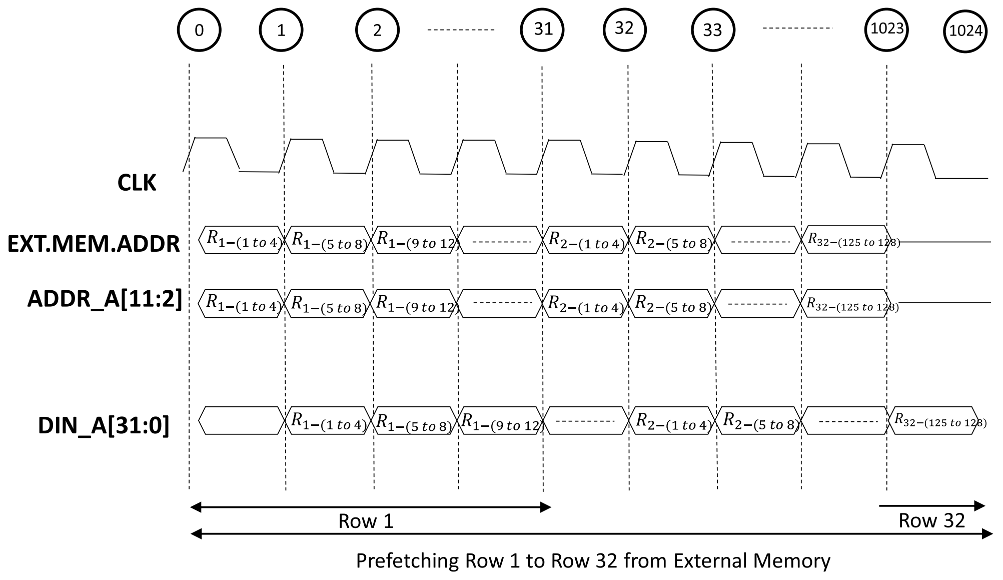

FIGURE 5. Timing analysis of data fetching operation.

consumption, and also compares the result with conventional methods reported in [18], [25].

The first application is taken for improving quality of X-ray images [23]. This task is comprised of implementing a 3 × 3 Laplacian filter, a 5 × 5 image Smoothening filter and two 3 × 3 Sobel filters. The 3 × 3 Laplacian filter requires 2 full row buffers, 5 × 5 image Smoothening filter requires 4 full row buffers whereas two 3 × 3 Sobel filters requires total 4 (i.e. 2∗2) full row buffers. Conventionally, these row buffers are implemented on BRAMs less efficiently by configuring them as FIFOs [18], [25]. It requires 1 BRAM per row buffer therefore 10 BRAMs for 10 row buffers are required altogether as shown in Table 2. However in our proposed buffer architecture, the pixel rows for up to 7 × 7 NIP are confined

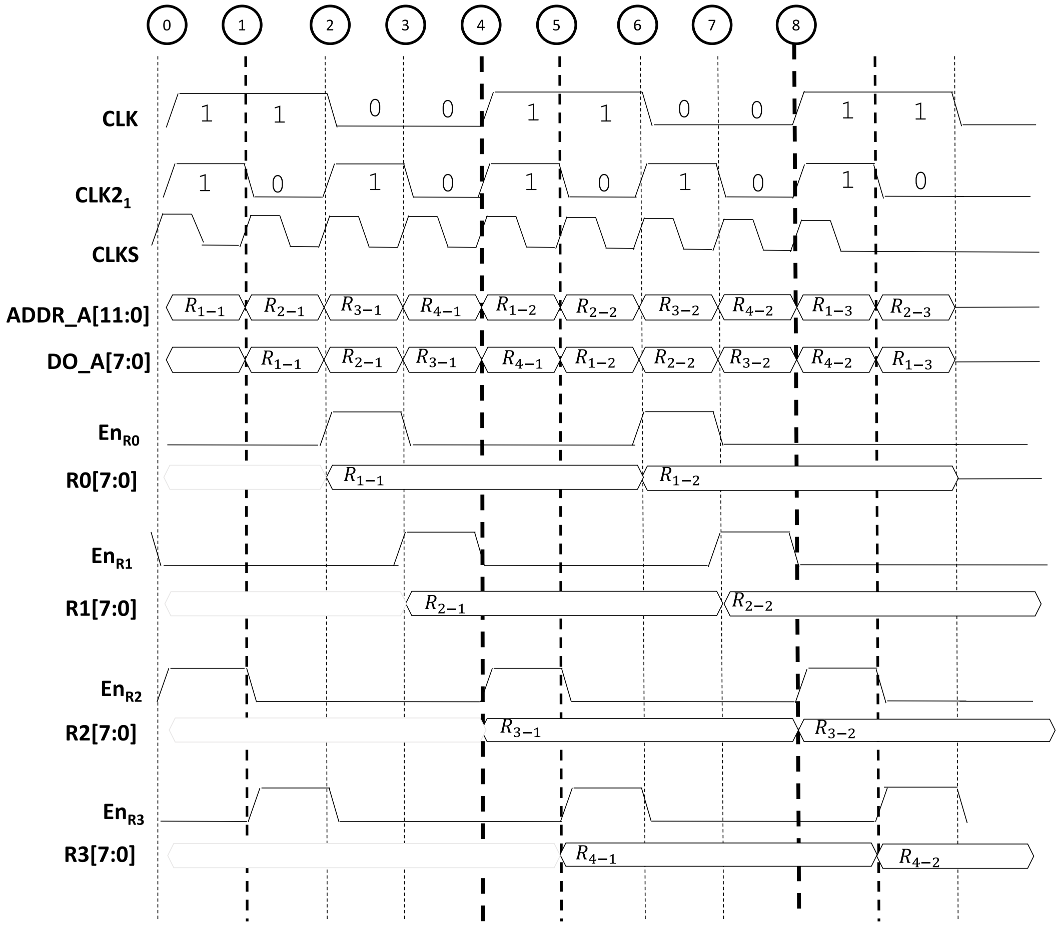

FIGURE 6. Timing analysis of data accessing operation (Port A).

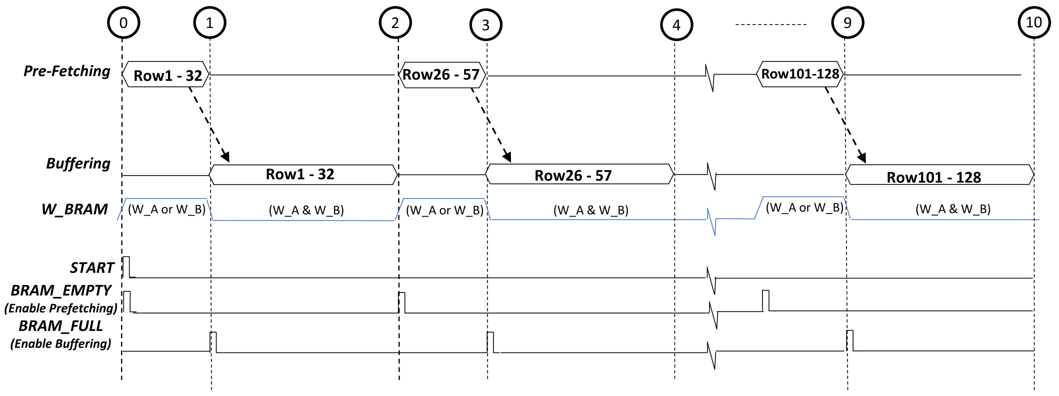

FIGURE 7. Timing analysis of complete operation.

within a single BRAM. Thus for this specific application [23] it requires total 4 BRAMs (i.e. 1∗4; 1 to buffer rows for a 3×3 Laplacian filter, 1 for a 5×5 image Smoothening filter and 2 for two 3×3 Sobel filters). In comparison to conventional approach our design results in saving the number of BRAMs by 60%. Considering the power consumption of each BRAM equals to 15.8mW [26], the conventional full buffer approach consume 158mW (i.e. 15.8 ∗ 10mW) whereas our proposed buffer architecture consume 60 % less power i.e. 63.2mW (i.e. 15.8 ∗ 4mW).

Let us consider another application for implementing a vision pre-processing task [24]. This pre-processing task is comprised of four 7×7 Gabor filters. These four filters require 24 (i.e. 6∗4) full row buffers. Conventional

TABLE 2. Power and resource comparison for practical applications.

Application	Required Row buffers	Conventional [18], [25] BRAMs	Power (mW)	Proposed BRAMs	Power (mW)	Percentage Reduction
Image Enhancement [23]	2+2+2+4=10	10	158	4	63.2	60%
Vision Pre-processing [24]	6+6+6+6=24	24	379.2	4	63.2	83.30%
Image segmentation [24]	4+2+2=8	8	126.4	3	47.4	62.50%

approaches reported in [18], [25] consume 24 BRAMs sequentially for implementing 24 row buffers whereas our proposed compact buffer architecture requires just 4 BRAMs (i.e. 1∗4; 1 for each 7×7 Gabor filter). It results in saving on the number of BRAMs by 83.3% and the power consumption by the same factor as shown in Table 2. Similarly a third application is when implementing image segmentation task as reported in [24]. This task is comprised of a 5×5 Gaussian filter and two 3×3 Sobel filters. The 5×5 image Gaussian filter requires 4 full row buffers whereas two 3×3 Sobel filters require total 4 full row buffers. Conventional approach [18], [25] consumes 8 BRAMs sequentially for implementing 8 row buffers altogether whereas our proposed compact buffer architecture requires just 3 BRAMs (i.e. one for each filter). It results in saving on the number of BRAMs by 62.5% and the power consumption by the same factor.

The above discussed on-chip buffer realization for the three chosen practical applications by using our proposed image buffering architecture as well as conventional full buffer architecture [18] supports the effectiveness of the design proposed in this paper for all such image processing tasks, especially in an area and power constrained environment.

VI. RESULTS AND COMPARISON

This work has presented an efficient on-chip buffering architecture on FPGA for resource-constrained NIP applications. A 7×7 NIP kernel and an 8-bit gray scale 128×128 image was considered as a Case Study for our Proof of Concept. The design was implemented on a low power Xilinx Artix-7 (XC7A35T) FPGA device using Xilinx Integrated Software Environment (ISE) 14.6. Post place and route results of our implemented buffer design and its comparison with the other similar works reported in recent literature are presented in Table 3 and Table 4 respectively. This design occupies only 1 BRAM with minimal overhead of the supporting circuit, i.e. only 113 logic Slices at an operating frequency of 278 MHz.

TABLE 3. Implementation results on Artix-7 (XC7A35T).

Resources	Utilized
BRAM 36	1
Slices	113
CMT	1
Frequency	278 MHz
Frame rate	13052 fps

The work presented in [7]–[10], [9] was based on PB methods, while [7], [13], [18] were based on the FB method. It is evident that all the reported PBs [7]–[10] have a strong reliance on external memory bandwidth usage for achieving throughput of 1 clock/pixel. For a (N×N) NIP, usually they require N pixels/clock from external memory for achieving throughput of 1 clock/pixel, therefore this is not suitable for large windows/kernels due to the standard 32-bit available bandwidth of external memories [6]. The variants of PB design proposed in [10] and [9] perform better than conventional PB design of [7] by the factor of S, only if S windows are computed simultaneously, which is usually not applicable in resource constrained environments. On the other hand, all previously proposed FBs have an efficiency of 1 clock/pixel at the bandwidth usage of 1 pixel/clock. They achieved throughput of 1 clock/pixel at the expense of partially utilizing several FPGA resources. Therefore, these resource inefficient FBs implementations consume hardware resources quickly and make NIP very expensive and unfeasible for resource constrained applications. Moreover, delivering pixels in row scan order is a major shortcoming of these designs which supports only limited NIPs.

As compared to these previously reported buffering schemes, our design provides a compact full buffering solution with an efficiency of 1.3 clock/pixel at a standard bandwidth requirement of 32-bits (i.e. 4 pixels/clock). Unlike to the PBs, which sharply raise memory bandwidth usage for larger kernels, our design uses a fixed memory bandwidth which is within the permissible bandwidth limit. It exploits the fact that when external memory bandwidth (32 bits) is higher than the pixel precision (8 for gray scale image), it can be used to fetch packed image pixels from external memory per clock cycle. This approach results in reducing I/O latency of pre-fetching operation in our design. With the achieved operating frequency of 278 MHz and efficiency of 1.3 clock/pixel, it is capable to support high frame rates for HD images (1080×1920) @103.

Before proceeding to compare the proposed design with the previously reported methods in terms of device utilization, we need to consider the following facts: In most of the conventional implementations, FPGA logic Slices [7]–[10], [9] are configured as large memory elements- this is not an appropriate approach as FPGAs have dedicated memory elements present in the form of BRAM. Hence, this is an inappropriate resource usage for implementing large memory functions when memory elements like BRAM remain unutilized. On the other hand, those implementations [12], [13], [18] that use BRAM as FIFO do not utilize it to its full capacity, which results in partial utilization of several BRAM units.

Secondly, in various previously reported conventional implementations, the memory requirement is calculated in

TABLE 4. Result and comparison.

Work	Device	Size	Application	Buffering method	External Memory Bandwidth	Resource Utilization		Throughput
					Pixel/Clock	Slices	BRAM	Clock/Pixel
[7]	XC4013	5×5	2D convolver	PB	5	376	0	1
[8] I	Spartan 3	7×7	2D convolver	PB	7	120(FF)	-	7
[8] II	Spartan 3	7×7	2D convolver	PB	7	784(FF)	-	1
[8] III	Spartan 3	7×7	2D convolver	PB	7	784(FF)	-	1
[10]	Spartan 3	5×5	2D convolver	PB	1.8	686	0	1
[9]	Virtex 5	7×7	1D convolver	PB	1.86	632 (FF), 328 LUT	0	1
[10]	Spartan 3	5×5	2D convolver	PB	1.8	686	0	1
[13]	Spartan 3	3×3	Morphology	FB	1	857	12	1
[18]	-	N×N	2D convolver	FB	1	Not Given	N-1	1
[7]	XC4013	5×5	2D convolver	FB	1	2129	0	1
Proposed	Artix-7	7×7	Diversified NIPs	FB with Pre-fetching	4	113	1	1.3

TABLE 5. Device utilization comparison.

Work	Buffering method	Equivalent Slices
Bose et al [7]	PB	93
Cardells et al [8] I	PB	15
Cardells et al [8] II	PB	98
Cardells et al [8] III	PB	98
Zhang et al [10]	PB	169
Arjun et al [9]	PB	240
Schmidt et al [13]	FB	1644
Moore et al [18]	FB	768
Bosi et al [7]	FB	583
Proposed	FB with Pre-fetching	**241**

terms of Flip Flops (FFs), which is a conservative approach on actual FPGA devices. The actual parameter to estimate FPGA based system resources is logic Slices rather than FFs. In reality, the Xilinx Synthesis Tool (XST) places and routes the design on target FPGA device by using mapping algorithms [29]. These algorithms partially utilize CLBs to limit the required FFs count. Considering the above findings, in Table 4 we compare our design with previous schemes in terms of Slice count. Moreover, since the reported results are available on different FPGA devices, therefore for a fair comparison of device utilization, we provide results in terms of equivalent Slice count which is calculated by multiplying the reported results with a normalizing factor. A factor of 8 is applied as normalizing factor for comparing Xilinx Spartan-3 and XC4013 with Xilinx Artix-7 [31] i.e. 1 Slice of Artix-7 is equivalent to 8 Slices of Spartan-3 and XC4013; while a factor of 4 is applied as a normalizing factor for comparison of those results which were given in terms of FFs count on Spartan-3 and XC4013. Also, 1 BRAM36 is considered to be equivalent to 128 Slices [32]. By comparing device occupancy in terms of equivalent logic, we eliminated the impact of technology difference and ensured fair comparison results. Finally, since all the reported results are available for different kernel sizes therefore, they are scaled to a 7 × 7 kernel for a fair comparison.

Table 5 shows that the functionally inefficient PBs proposed in [7] and [10] which are considered as low cost approaches have utilized 93 and 169 equivalent Slices for partial pixel data buffering respectively. On the other hand, functionally efficient FBs proposed in [7] consumed 583 equivalent Slices as memory elements and underutilized dedicated memory element (BRAM) which cannot be further used for implementation of other logic functions. However, [13] utilized 1644 equivalent Slices for memory function in an inefficient way, failing to exploit the full potential of BRAM. As compared to all of these proposed buffering approaches, our functionally efficient yet compact design occupies 241 equivalent Slices for full row buffering.

It is evident from these results that the proposed design offers performance comparable to the FB method and utilizes hardware resources comparable to the low cost PB method [7]. Another advantage of the proposed on-chip buffer design is its flexibility to provide pixels in different scan order, so that it can be used to fulfill the buffering requirements of diversified NIPs; while all previously reported buffers have limitations in providing pixels in a fixed row scan order which constrains them to be used only for those specific NIPs which require pixels in that specific scan order.

Our proposed design is able to cater for any image size. Images that are larger than the chosen size of 128×128 can be partitioned into vertical bands [7] of W width (W<128). These vertical bands are buffered by using the same number of BRAMs as a narrow but complete image at a time.

VII. CONCLUSION

This paper proposed an efficient and compact BRAM based on-chip buffer architecture with an additional feature of pre-fetching and user defined scan ordering for a wide range of neighborhood operations. The proposed buffering solution eliminates the linear requirement of BRAM with respect to the NIP size for full rows buffering by confining multiple image rows within a single BRAM without compromising on system performance. It operates at 278 MHz to support real time image processing requirements (1080 × 1920) @103fps. The proposed efficient on-chip buffer methodology out-merits previous conventional implementations in terms of area, making it suitable for portable, low power and mobile imaging devices.

REFERENCES

[1] M. Al Najjar, M. Ghantous, and M. Bayoumi, *Video Surveillance for Sensor Platforms*. New York, NY, USA: Springer-Verlag, 2014.

[2] R. C. Gonzalez and R. E. Woods, *Digital Image Processing*. Upper Saddle River, NJ, USA: Prentice-Hall, 2002.

[3] C. Solomon and T. Breckon, *Fundamentals of Digital Image Processing: A Practical Approach With Examples in MATLAB*. Hoboken, NJ, USA: Wiley, 2011.

[4] M. Holzer, F. Schumacher, I. Flores, T. Greiner, and W. Rosenstiel, "A real time video processing framework for hardware realization of neighborhood operations with FPGAs," in *Proc. 21st Int. Conf. Radioelektronika*, Apr. 2011, pp. 1–4.

[5] D. G. Bailey, *Design for Embedded Image Processing on FPGAs*. Hoboken, NJ, USA: Wiley, 2011.

[6] T. P. Cao, D. Elton, and G. Deng, "Fast buffering for FPGA implementation of vision-based object recognition systems," *J. Real-Time Image Process.*, vol. 7, no. 3, pp. 173–183, Sep. 2012.

[7] B. Bosi, G. Bois, and Y. Savaria, "Reconfigurable pipelined 2-D convolvers for fast digital signal processing," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 7, no. 3, pp. 299–308, Sep. 1999.

[8] F. Cardells-Tormo and P.-L. Molinet, "Area-efficient 2-D shift-variant convolvers for FPGA-based digital image processing," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 53, no. 2, pp. 105–109, Feb. 2006.

[9] A. K. Joginipelly and D. Charalampidis, "Efficient separable convolution using field programmable gate arrays," *Microprocess. Microsyst.*, vol. 71, Nov. 2019, Art. no. 102852.

[10] H. Zhang, M. Xia, and G. Hu, "A multiwindow partial buffering scheme for FPGA-based 2-D convolvers," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 54, no. 2, pp. 200–204, Feb. 2007.

[11] F. Cardells-Tormo and P. Molinet, "Area-efficient 2-D shift-variant convolvers for FPGA-based digital image processing," in *Proc. IEEE Workshop Signal Process. Syst. Design Implement.*, Nov. 2005, pp. 209–213.

[12] X. Liang, J. Jean, and K. Tomko, "Data buffering and allocation in mapping generalized template matching on reconfigurable systems," *J. Supercomput.*, vol. 19, no. 1, pp. 77–91, 2001.

[13] M. Schmidt, M. Reichenbach, A. Loos, and D. Fey, "A smart camera processing pipeline for image applications utilizing marching pixels," *Signal Image Process., Int. J.*, vol. 2, no. 3, pp. 137–156, Sep. 2011.

[14] L. Gasparini, R. Manduchi, M. Gottardi, and D. Petri, "An ultralow-power wireless camera node: Development and performance analysis," *IEEE Trans. Instrum. Meas.*, vol. 60, no. 12, pp. 3824–3832, Dec. 2011.

[15] M. Imran, N. Ahmad, K. Khursheed, M. A. Waheed, N. Lawal, and M. O'Nils, "Implementation of wireless vision sensor node with a lightweight bi-level video coding," *IEEE J. Emerg. Sel. Topics Circuits Syst.*, vol. 3, no. 2, pp. 198–209, Jun. 2013.

[16] M. Imran, K. Shahzad, N. Ahmad, M. O'Nils, N. Lawal, and B. Oelmann, "Energy-efficient SRAM FPGA-based wireless vision sensor node: SENTIOF-CAM," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 24, no. 12, pp. 2132–2143, Dec. 2014.

[17] C.-L. Sotiropoulou, L. Voudouris, C. Gentsos, A. M. Demiris, N. Vassiliadis, and S. Nikolaidis, "Real-time machine vision FPGA implementation for microfluidic monitoring on lab-on-chips," *IEEE Trans. Biomed. Circuits Syst.*, vol. 8, no. 2, pp. 268–277, Apr. 2014.

[18] C. T. Moore, H. Devos, and D. Stroobandt, "Optimizing the FPGA memory design for a sobel edge detector," in *Proc. 20th Annu. Workshop Circuits, Syst. Signal Process. (ProRISC)*, 2009, pp. 496–499.

[19] M. Imran, K. Khursheed, N. Ahmad, M. O'Nils, N. Lawal, and M. A. Waheed, "Complexity analysis of vision functions for comparison of wireless smart cameras," *Int. J. Distrib. Sensor Netw.*, vol. 10, no. 1, Jan. 2014, Art. no. 710685.

[20] G. Licciardo, C. Cappetta, and L. Di Benedetto, "Design of a convolutional two-dimensional filter in FPGA for image processing applications," *Computers*, vol. 6, no. 2, p. 19, May 2017.

[21] G. D. Licciardo, C. Cappetta, L. Di Benedetto, A. Rubino, and R. Liguori, "Multiplier-less stream processor for 2D filtering in visual search applications," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 28, no. 1, pp. 267–272, Jan. 2018.

[22] S. Jin, J. Cho, X. D. Pham, K. M. Lee, S.-K. Park, M. Kim, and J. W. Jeon, "FPGA design and implementation of a real-time stereo vision system," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 20, no. 1, pp. 15–26, Jan. 2010.

[23] S. Perri, M. Lanuzza, P. Corsonello, and G. Cocorullo, "A high-performance fully reconfigurable FPGA-based 2D convolution processor," *Microprocess. Microsyst.*, vol. 29, nos. 8–9, pp. 381–391, Nov. 2005.

[24] B. Zhang, K. Mei, and N. Zheng, "Coarse-grained dynamically reconfigurable processor for vision pre-processing," *J. Signal Process. Syst.*, vol. 79, pp. 45–61, Jul. 2013.

[25] C. Johnston, K. Gribbon, and D. Bailey, "Implementing image processing algorithms on FPGAs," in *Proc. 11th Electron. New Zealand Conf. (ENZCon)*, 2004, pp. 118–123.

[26] A. A. M. Kazmi, P. Akhtar, and D.-E.-S. Kundi, "FPGA based compact and efficient full image buffering for neighborhood operations," *Adv. Electr. Comput. Eng.*, vol. 15, no. 1, p. 10, 2015.

[27] *7 Series FPGAs Memory Resources, User Guide V1.10*, Xilinx, Inc., Albuquerque, NM, USA, 2013.

[28] *LogiCORE IP Block Memory Generator V7.3*, Xilinx, San Jose, CA, USA, 2012.

[29] V. G. Oklobdzija, *The Computer Engineering Handbook*. Boca Raton, FL, USA: CRC Press, 2001.

[30] *7 Series FPGAs Overview, DS180*, Xilinx, San Jose, CA, USA, Jul. 2013.

[31] V. Betz, J. Rose, and A. Marquardt, *Architecture and CAD for Deep-Submicron FPGAs*, vol. 497. New York, NY, USA: Springer, 2012.

[32] S. Drimer, T. Güneysu, and C. Paar, "DSPs, BRAMs, and a pinch of logic: Extended recipes for AES on FPGAs," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 3, no. 1, p. 3, 2010.

[33] D.-E.-S. Kundi, A. Aziz, and M. Kazmi, "An efficient single unit T-box/T-1-box implementation for 128-bit AES on FPGA," *Secur. Commun. Netw.*, vol. 8, no. 9, pp. 1725–1731, Jun. 2015.

[34] *Artix-7 FPGA Data Sheet: DC and Switching Characterstics*, Xilinx, San Jose, CA, USA, 2014.

[35] S. Perri and P. Corsonello, "Efficient memory architecture for image processing," *Int. J. Circuit Theory Appl.*, vol. 39, no. 3, pp. 351–356, Mar. 2011.

MAJIDA KAZMI was born in 1983. She received the B.E. and M.E. degrees in electrical engineering from the NED University of Engineering and Technology, Karachi, Pakistan, in 2009, and the Ph.D. degree in electrical engineering from NUST University, Pakistan, in 2017. She is currently an Assistant Professor with the Department of Computer and Information Systems Engineering, NED University of Engineering and Technology. Her research interests include digital systems design, system on a chip, digital image processing, and computer vision.

ARSHAD AZIZ received the Ph.D. degree in electrical engineering from the National University of Sciences and Technology, Karachi, Pakistan, in 2007. He is currently a Professor with the Electrical Engineering Department, National University of Sciences and Technology and the Director of the Embedded Systems Laboratory. His research interests include digital systems design, network security, and edge computing.

HASHIM RAZA KHAN received the B.E. degree in electrical engineering from the NED University of Engineering and Technology, Karachi, in 2002, the M.Sc. degree in communications engineering from RWTH Aachen, Germany, in 2006, and the Ph.D. degree in electronic engineering from the NED University of Engineering and Technology, in 2014. He was involved in research with Agilent Technologies, Germany, Infineon Technologies, and Linköping University, Sweden. He is currently an Assistant Professor with the NED University of Engineering and Technology, where he is also responsible with the Instrumentation Centre, the RF Laboratory, Electronics Design Centre, and the Neuromorphic Circuit Design Activities with the National Centre of Artificial Intelligence. His research interests include circuits and system design for wide range of applications, including AI, robotics, multistandard transceivers, computer vision, and power electronics.

SAAD AHMED QAZI received the B.E. degree in electrical engineering from the NED University of Engineering and Technology, Karachi, in 2001, the M.S. degree in digital signal processing applications from Lancaster University, U.K., in 2002, and the Ph.D. degree from Brunel University London, U.K., in 2006. He is currently with the NED University of Engineering and Technology as a Meritorious Professor and the Dean with the Faculty of Electrical and Computer Engineering. He is also a Principal Investigator with the Neurocomputation Laboratory, National Centre of Artificial Intelligence. He has several international publications in major areas of technology. He is also working on several national and international research projects. His research interests include digital signal processing, joint time frequency analysis, data analytics, and decision support systems.

LAMPROS K. STERGIOULAS (Member, IEEE) received the M.Sc. (Eng.) and Ph.D. degrees in electrical engineering from the University of Liverpool, U.K. He was a Chaired Professor in business analytics with the Surrey Business School, University of Surrey, and a Chaired Professor in computer science with Brunel University London, U.K. He is currently a Professor of data science with The Hague University of Applied Sciences, where he also leads the Data Science Research Group, Faculty of IT and Design. He is an Expert Evaluator for various programmes sponsored by the European Commission and EU Member States. He has been a Principal Investigator of more than 30 international projects and a Coordinator of four European research projects, in which he collaborated with EU and national public organizations, such as the European Centre for Disease Prevention and Control (ECDC), the European Medicines Agency (EMA), the European Commission, the National Health Service (NHS), U.K., and national and regional authorities around the world. He is also active with the European Commission as an Expert in artificial intelligence, data science, and research ethics. He has written more than 200 scientific publications, supervised, and examined many Ph.D. theses in data science, human-centered computing, health informatics, modeling and simulation, and intelligent systems. His current research interests include applied AI/data science and analytics, health informatics, data-driven management and innovation, system modeling an simulation, and data ethics.