# FPGA Based Compact and Efficient Full Image Buffering for Neighborhood Operations

Majida KAZMI, Arshad AZIZ, Pervez AKHTAR, Dur-e-Shahwar KUNDI
*Department of Electrical Engineering (PNEC),*
*National University of Sciences and Technology (NUST), H-12, Islamabad, 46000, Pakistan*
*majida.kazmi@pnec.nust.edu.pk*

***Abstract*—Image processing systems based on neighborhood operations i.e. Neighborhood Processing Systems (NPSs) are computationally expensive and memory intensive. Field Programmable Gate Array (FPGA) based parallel processing architectures accelerate calculations of NPS provided if they have fast external-memory data access by using on-chip data buffers. The conventional data buffers namely full Row Buffers (RBs) implemented with FPGA embedded memory resources i.e. Block RAMs (BRAMs) are resource inefficient. It makes overall NPS implementation on FPGA expensive and infeasible especially for resource-constraint environment. This paper presents compact and efficient image buffering architecture with an additional feature of pre-fetching. Proposed design fits in minimal BRAMs by using small yet efficient Main Control Unit (MCU). Its optimal multi-rated BRAM data accessing technique reduces BRAM cost to provide multiple pixels of pre-fetched data/clock to NPS in a fixed pattern. It controls and synchronizes BRAMs operations to attain throughput of 1 clock/pixel. Thus our buffer architecture with 66% reduction in BRAM requirement as compared to conventional RBs is capable to support buffering for real time systems with high resolution (1080x1920@62fps). Therefore proposed buffer architecture can suitably replace conventional RB in any real time NPS application**.

***Index Terms*—Buffer Storage, Convolver, Field Programmable Gate Array, Image processing, Image storage.**

## I. INTRODUCTION

Neighborhood operations based image pre-processing is frequently used to obtain high quality output images. These neighborhood operations use two dimensional (2D) structuring element i.e. filter kernel which is shifted pixel-wise over the entire image. For every shift operation, the input image pixels covered by the filter kernel are processed by mathematical operation to calculate an output pixel. The overall operation requires high computational power and extensive external memory bandwidth for real time image processing [1].

FPGA based parallel processing architectures exploit data and instruction level parallelism of these neighborhood operations to accelerate its calculation [2]. However while accelerating performance of NPS on FPGA platform, high data transfer rate between NPS and external memory becomes bottleneck as fetching redundant neighborhood data from external memory onto the FPGA is relatively a slow process [3].

For this reason, it is essential to limit the data traffic between FPGA based NPS and external memory by buffering image data on chip (FPGA). Buffering provides NPS a rapid access to on-chip buffered data and to re-use this data as many times as required by NPS without using external memory bandwidth [4,5]. Therefore on chip image data buffering systems become indispensable for high performance FPGA based NPS.

On chip image buffering schemes can be categorized as Partial Buffering (PB) schemes and Full Buffering (FB) schemes. PB stores only partial input image rows involved in current neighborhood operation in its shift registers to calculate current output pixel. It requires multiple pixels from external memory for every next output pixel calculation, to keep pixel throughput rate of 1 clock/pixel [5,6]. This scheme occupies few FPGA resources for buffering partial image data at the expense of increased external memory bandwidth requirement.

Second category of image buffering scheme is FB. It stores full rows of input image involved in current neighborhood operation in its RBs to calculate current output pixel [7]. A single pixel is required from external memory for every next pixel calculation to keep pixel throughput rate of 1clock/pixel. With efficiency of 1 clock/pixel and external memory bandwidth requirement of 1, FB is more appropriate for real time applications at the expense of large amount of FPGA resources required for full rows buffering in its RBs. These FPGA resources for implementing RBs are even higher than resources for implementing rest of the system (i.e. arithmetic unit for neighborhood operation) [7], which makes overall NPS implementation on FPGA very expensive.

Therefore reducing implementation cost of RB in terms of FPGA area is of great interest to reduce NPS overall implementation cost for compact yet high performance FPGA based NPS. In this paper, we present a compact and efficient image buffering scheme, which uses lesser FPGA embedded memory resources as compared to conventional RBs to keep pixel throughput rate of 1clock/pixel. The good balance between on-chip memory resource utilization and overall system performance makes it suitable for any real time NPS application with high spatial and temporal resolution (1080x1920@62 fps).

Rest of the paper is organized as follows. Section II discusses related work. In Section III we present our proposed efficient full image buffering architecture for NPS. Section IV declares results and comparison with up to date work. Performance analysis of our design is presented in Section V. Section VI concludes the work.

Digital Object Identifier 10.4316/AECE.2015.01014

## II. Related Work

On-chip (FPGA) data buffering is essentially required along with NPS to accelerate its performance on FPGA platform. A FPGA based PB scheme for single window processing was proposed by Bosi *et al* [5]. While increasing NPS kernel size, it costs a sharp increase in external memory bus bandwidth requirement in order to keep the 1 clock/pixel throughput rate. Also, excessive data transfer from external memory per output pixel is a slow process that limits overall operating frequency of NPS. To improve data reuse capability and lowering bandwidth requirement of PB, different variants of PB were proposed [2,8]. However to scale PB for large kernel size or image data, fixed and limited bandwidth of external memory device becomes a bottleneck [9]. Due to above mentioned limitations, PB is not recommended for real time image processing applications [9,10].

For real time applications, FB is more appropriate buffering scheme but at the cost of additional memory resources. Its external memory bandwidth requirement is 1 pixel/clock to keep the throughput rate at 1clock/pixel. For a RxC NPS, it buffers R-1 full rows of input image in RBs by using Configurable Logic Blocks (CLBs) as chain of shift registers [5]. This results in wastage of considerable amount of CLBs to implement large memory functions and degrades overall system performance. Liang *et al* [6] suggested using BRAMs instead of CLBs for implementing RBs to spare significant number of CLBs for rest of the logic circuitry. Wiatr *et al* [7] implemented FPGA based convolver and calculated that number of CLBs required to implement RBs is even more than CLBs required for implementing rest of the system i.e. convolver (NPS). Therefore they recommended BRAM based RBs. Moore *et al* [11] strengthened the concept of [6,7] by investigating performance and area requirement of FB scheme using combination of CLBs (as Shift registers) and BRAMs. They concluded that BRAM is more suitable to implement RBs in FB instead of CLBs based shift registers.

All the latest implementations of NPS based real time systems follow the same trend for row buffering by using BRAMs as standard FPGA primitive. These implementations range from medical image processing [12-16] to smart cameras [10,17] and stereo vision [18] etc. In all of these RB implementations, BRAMs requirement increase linearly with increasing NPS kernel size, where each input image row requires a separate BRAM for buffering. Secondly most frequently used image sizes are ranging from 320x256 to 640x480 [15,17,18]. Thus each row for buffering occupies only (320*8) =2.5kbits to (640*8) 5kbits of BRAM, which is quite less than actual capacity of latest available BRAM [19]. Therefore in most of these implementations, almost 80% capacity of BRAM remains unutilized. This inefficient BRAM utilization is a major contributing factor for increasing implementation cost of NPS on FPGAs as 70% to 60 % of total resources are required for implementing its RBs and only 30% to 40% of total resources are required for its arithmetic unit (to perform neighborhood operation) [7].

Up till date very little work is reported in open literature to optimize RB implementation on FPGA. Bailey [20] proposed a pipelined based BRAM architecture for RB to enhance operating frequency of their NPS architecture. They use register as pipeline stage at the input and output in their BRAM based RBs. This results in improving the impact of its clock to out timings which in turns increases the overall frequency of their design to 923 MHz on Virtex-5. However it is evident from Virtex-5 data sheets [21] that frequency of BRAM cannot exceed 550 MHz therefore reported frequency of BRAM based design is practically not possible. This optimization has still underutilized the potential of BRAMs. Thus using these conventional inefficient BRAM based RBs [10,12-14,18] for NPS not only result in poor resource utilization but also increase power consumption. Therefore optimizing resource (BRAM) requirement for RBs is of great interest to reduce NPS overall implementation cost for compact yet high performance FPGA based NPS.

## III. Our FPGA Based Efficient Image Buffering Architecture

This work proposes a compact and efficient image buffering architecture with an additional feature of pre-fetching data. We have chosen 7x7 convolution filter (as NPS) and 8 bit precise grey scale image of size 128x128 as a case study to explain the concept of our buffering architecture. The design of proposed buffering architecture is shown in Figure 1. It comprises of two main parts i.e. BRAMs (BRAM1 and BRAM2) as memory elements and MAIN CONTROL UNIT (MCU) as buffer and pre-fetch controller unit. Both BRAMs are working alternately for image pre-fetching and image buffering operations while MCU serves to control both BRAMs and provides multiple pixels of pre-fetched data per clock to NPS in a fixed pre-defined pattern with an efficiency of 1 clock/pixel.

For simplicity, MCU is further divided into four sub modules. PREFETCH CONTROL (PRC), BUFFER CONTROL (BUC), DATA SYNCHRONIZATION (DAS) and BRAM SELECTOR (BRS). Each sub module performs a specific task. PRC module controls the pre-fetching of image data from external memory (i.e. BRAM write operation) while BUC module reads buffered data (i.e. BRAM read operation). DAS synchronizes and regulates the flow of buffered data to convolution filter and BRS alternately selects BRAM 1 and BRAM2 to maintain the throughput of 1.

The design is initiated by *Start* signal which enables PRC module and buffer architecture starts fetching data from external memory in segments of 32 Kb (to fully utilize single 32 Kb BRAM [19]) and writes it onto BRAM1. Once first segment of 32Kb image data is completely written on BRAM1, it generates a signal at point *1* to enable BUC module as shown in Figure1. BUC module starts reading data from BRAM1 at four times higher clock speed than rest of the system clock *CLK* in column scan order. It reads total 8 pixels from true dual port BRAM1 per *CLK* cycle and temporarily stores these premature output pixels to DAS module. It comprises of eight output registers (R0-R7) operating at twice the system clock speed to hold premature output data of BRAM 1 and delivers 7 out of 8 valid data pixels to convolution filter simultaneously per *CLK* cycle as shown in Figure 1. At the same time when BUC is reading data from BRAM1, PRC module starts fetching second

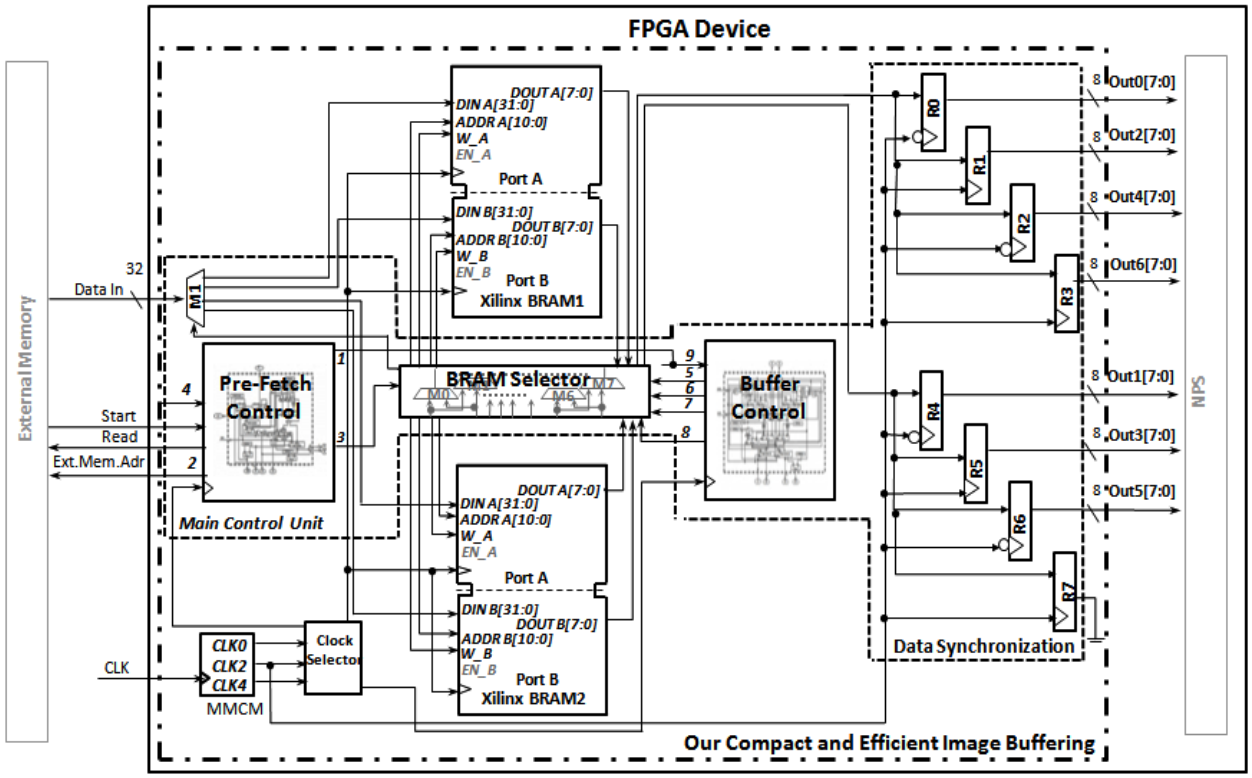


Figure 1. Our Compact and Efficient Image Buffering Architecture on FPGA

segment of 32 Kb image data and writes on BRAM2. Alternate selection of BRAM1 and BRAM2 for data fetching and buffering operations is done by BRS module for a seamless data movement to NPS, keeping pixel throughput rate of 1. The process continues until entire image is accessed by convolution filter for processing.

In order to further elaborate working of our buffering architecture, in next section we will individually discuss the working of each sub module of MCU in more detail.

### *A. PRE-FETCH CONTROL (PRC)*

PRC module fetches packed data [6] from external memory (i.e. generally available in 32 bit wide data port [9]) and writes on the BRAM. With packed data fetching (should be ≥2 pixels), latency of data fetching operation remains less than latency of data buffering operation which in turn results seamless data movement to convolution filter at the throughput of 1 clock/pixel.

In order to write the packed data (4 pixels per *CLK*) on BRAM, we configure it with different write and read port widths [19] i.e. 32-bits width for write and 8-bits for read port. With this BRAM configuration, the PRC module writes same amount of data from external memory in lesser clock cycles as compared to conventional fixed port widths approach which results in data fetching latency less than data buffering latency without using any extra logic. Detailed working of PRC is shown in Figure 2. It mainly consists of two components, E_MEM_ADDR and W_ADDR_BRAM. They generate the external memory addresses and the BRAM write addresses respectively.

Upon initialization by *Start* signal, E_MEM_ADDR generates external memory addresses starting from its first location at point *2*. At the same time, W_ADDR_BRAM generates consecutive addresses for port A of BRAM at point *3*. Once 32 rows are completely fetched from the external memory, it stops first pre-fetching operation for BRAM1 and will start the second pre-fetching operation for BRAM2.

In first pre-fetch operation, PRC module loads BRAM 1 with 32Kb image pixel data (4096 pixels i.e. $X_1$ to $X_{4096}$) starting from 1$^{st}$row (pixel $X_1$-$X_{128}$) to 32$^{nd}$row (Pixel $X_{3969}$-$X_{4096}$) of the input image. Pixels processing requirement of convolution filter is shown in Figure 3. It shows that for processing the pixel $X_{3712}$ (i.e. last pixel of 29$^{th}$ row) it requires its total 49 neighborhood pixels that are available from the 26$^{th}$ row ($X_{3201}$ to $X_{3328}$) till 32$^{nd}$row of image. This data is already available in BRAM1. Now for processing the next pixel i.e. $X_{3713}$(i.e. first pixel of 30$^{th}$ row), it requires it's another 49 neighborhood pixels from 27$^{th}$ row ($X_{3328}$ to $X_{3456}$) till 33$^{nd}$($X_{4097}$ to $X_{4224}$) row of image as shown in Figure 3. However 33$^{rd}$ row of the image data is not available in first 32 Kb data segment stored in BRAM 1 so at this point PRC module performs the second pre-fetch operation to acquire the next segment of input image which can provide neighborhood pixels required for processing of the pixel $X_{3713}$ and onwards.

Therefore for second pre-fetch operation, initial value of the E_MEM_ADDR is tracked and resets to first memory location of 27$^{th}$ row of input image to pre-fetch external memory data from 27$^{th}$ row till 58$^{th}$ row ($X_{7297}$ to $X_{7424}$) and writes on BRAM2. With E_MEM_ADDR, W_ADDR_BRAM and supporting circuitry, the PRC module continues pre-fetch operation alternately for both BRAMs (1 and 2) up till the last pixel of input image.

### *B. BUFFER CONTROL MODULE (BUC)*

After writing image data to BRAM1 by PRC module as discussed in previous section, now the BUC module reads this stored data in a pre-determined pattern from each port of BRAM1. The BUC module reads BRAM1 at *CLK4 (4*CLK)* and output 7 buffered image pixels per *CLK* in column wise pattern.

This module has a symmetrical circuitry to generate read addresses for both ports of BRAM1 (i.e. A and B). Its left portion generates addresses for port A while right portion generates addresses for port B. Right portion of symmetrical circuitry mainly comprises of the read address generator RD_ADDR_B for port B of BRAM1 controlled by COL_SEL_2 to read pixels in a required pattern from BRAM1. Its detailed working is shown in Figure 4.

Upon initialization, both the left and right portions start working simultaneously, RD_ADDR_A of the left portion starts reading BRAM1 data as a set of four stored rows (Row 1,3,5,7 of input image) from port A in column scan order and delivers this premature data to output registers R0-R3. At the same time in right portion, RD_ADDR_B starts reading BRAM1 data as a set of four stored rows (Row 2,4,6,8 of input image) from port B in column scan order and delivers this premature data to output registers R4-R7. In right half of symmetrical circuitry, RD_ADDR_B is controlled by COL_SEL_2 to generate addresses of each set of rows exactly twice in column scan order. It is clear from Figure 5 that by reading each set of rows exactly twice and swapping the output by using SWP_DATA among output registers R0-R3 and R4-R7; BUC module delivers every input row, pixel by pixel to each 1D convolution filter and accomplishes the convolution operation successfully. After completely reading the first set of rows (Row 2,4,6,8 of input image) twice, RD_ADDR_B starts reading the second set of rows (Row 4, 6,8,10 of input image) and continues the same process till last set of rows.

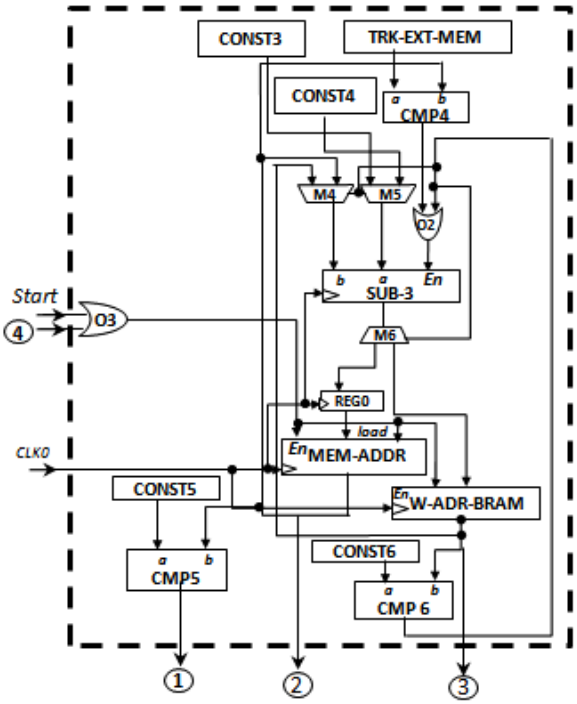


Figure 2. Pre-Fetch Control module (PRC)

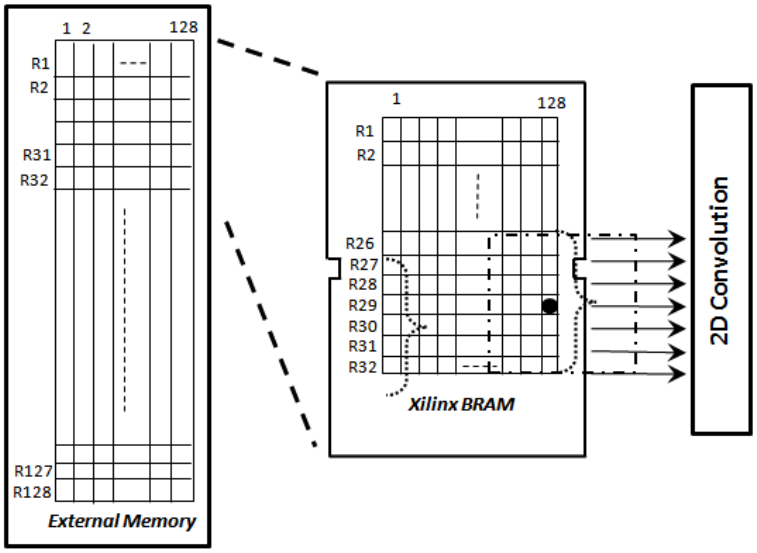


Figure 3. A 32 Kb image segment buffered in a single BRAM

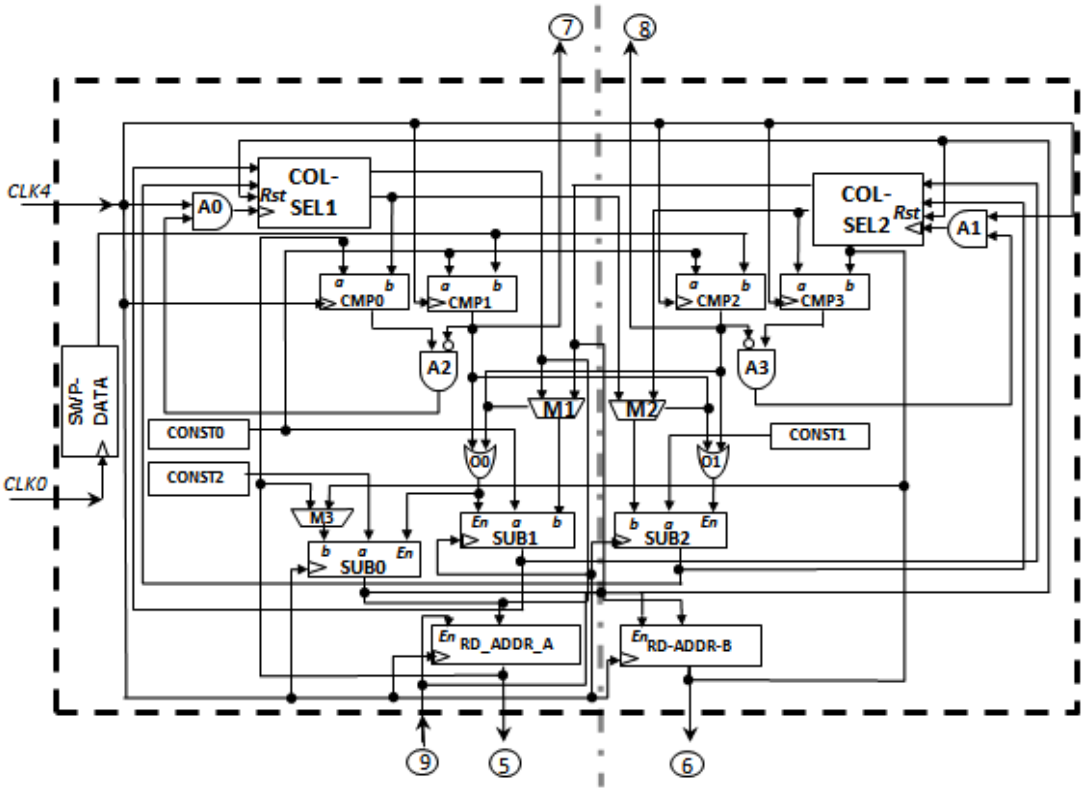


Figure 4. Buffer Control Module (BUC)

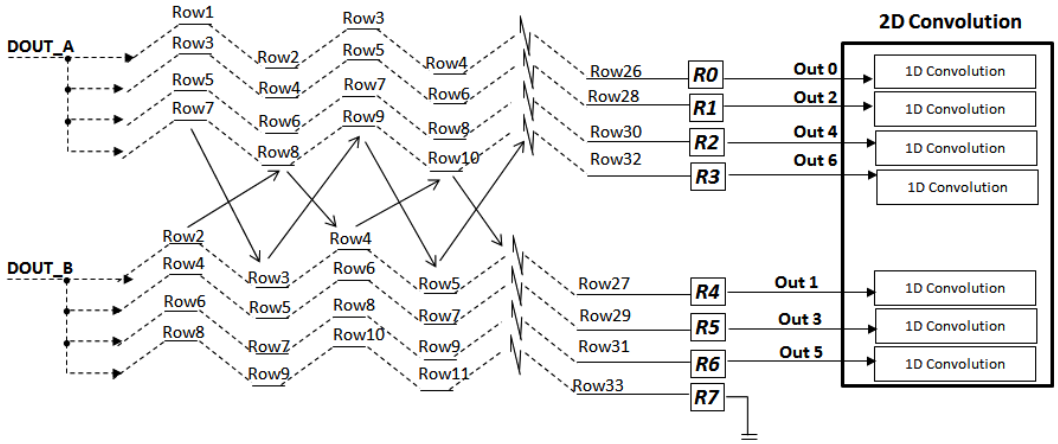


Figure 5. BRAM data output flow to 2D Convolution filter

Simultaneously left portion of symmetrical circuit performs the similar operation in parallel for port A of the BRAM1 by using COL_SEL_1, RD_ADDR_A and SWP_DATA. In this way BUC module reads all the pixel data from BRAM1 at *CLK4* in column scan order and reuses this data as many times as required by the 2D convolution filter without using external memory bandwidth.

### *C. DATA SYNCHRONIZATION (DAS)*

As discussed in above section, BUC reads the buffered pixels from BRAM at 4 times higher clock speed than the system clock *CLK* i.e. *CLK4* and delivers the output pixels to convolution filter which is operating at a different clock speed i.e. at system *CLK* speed. The output pixels data of BRAM at *CLK4* is premature with respect to *CLK*. Therefore it is necessary to properly hold and synchronize BRAM1 output for its validity for a complete *CLK* cycle and it's on time availability to convolution filter. For this reason, we deploy DAS module to balance the system. It is comprised of eight intermediate registers R0-R7 as shown in Figure 1. These registers are working at intermediate clock speed i.e. *CLK2* (2x of *CLK*). They ensure pixels validity for a complete *CLK* cycle and it's on time availability to convolution filter.

Table I clearly shows that DAS completely synchronizes timings of output pixels from BUC module and keeps it valid for a complete *CLK* cycle. At first positive edge of *CLK4*, first output pixel $P_{n\text{-}n}$ of port A is stored in R0 at positive *CLK2* cycle and negative *CLK* cycle. At second positive edge of *CLK4*, second output pixel $P_{n\text{-}n+1}$ of port A is stored in R1 at negative clock cycle of both *CLK2* and *CLK* . At third positive edge of *CLK4*, third output pixel $P_{n\text{-}n+2}$ of port A is stored in R2 at positive clock cycle of both *CLK2* and *CLK* similarly at forth positive edge of *CLK4*,

TABLE I. OUTPUT PIXELS SYNCHRONIZATION

| BRAM Port | Output Registers | CLK (Convolution Filter Clock) | CLK2 (Output Register Clock) | CLK4 (BRAM Read Clock) | Valid Output Pixel $_{Column - Row}$ |
|---|---|---|---|---|---|
| Port A | R0 | 0 | 1 | +ve edge | Pixel $_{n-n}$ |
| | R1 | 0 | 0 | +ve edge | Pixel $_{n-n+1}$ |
| | R2 | 1 | 1 | +ve edge | Pixel $_{n-n+2}$ |
| | R3 | 1 | 0 | +ve edge | Pixel $_{n-n+3}$ |
| Port B | R4 | 0 | 1 | +ve edge | Pixel $_{n-n+4}$ |
| | R5 | 0 | 0 | +ve edge | Pixel $_{n-n+5}$ |
| | R6 | 1 | 1 | +ve edge | Pixel $_{n-n+6}$ |
| | R7 | 1 | 0 | +ve edge | Pixel $_{n-n+7}$ |

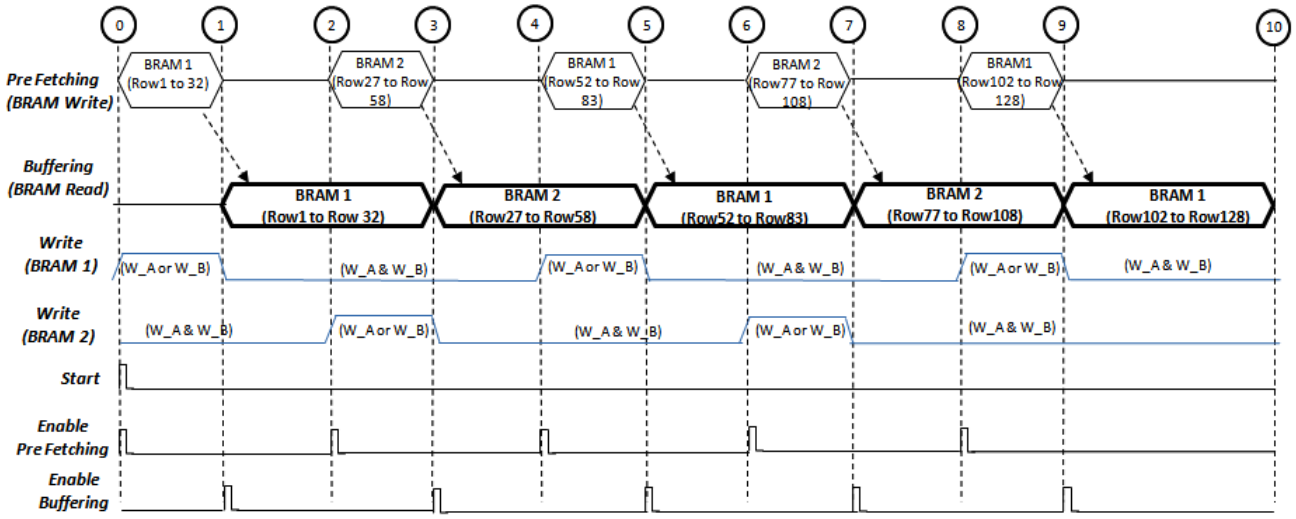


Figure 6. Selection of BRAMs (1 and 2) for Pre-fetching and Buffering Operations

forth output pixel $P_{n-n+3}$ of port A is stored in R3 at negative *CLK2* cycle and positive *CLK* cycle. The same pattern for four output pixels $P_{n-n+4}$ to $P_{n-n+7}$ from port B will be followed for registers R4-R7. In this way, 7 out of 8 valid pixels data are delivered to the convolution filter and this sequence continues for all output pixels from BUC module.

### *D. BRAM SELECTOR (BRS)*

Our compact and efficient buffering design successfully performs pre-fetching and buffering operations for BRAM1 and BRAM2 alternately as mentioned in previous sections. As soon as BRAM1 completes first pre-fetching operation, it starts first buffering operation to deliver 7 valid pixels data per *CLK* to the convolution filter. Meanwhile BRAM2 pre-fetches data for next buffering operation which will be available to convolution filter without any delay to attain throughput of 1 clock/pixel.

We deploy BRS module that balance both pre-fetching and buffering operations between BRAM1 and BRAM2 to continuously deliver pixel data to convolution filter without any delay as shown in Figure 6. While BRAM1 delivers 7 pixel data per *CLK* to convolution filter, it selects BRAM2 for the second pre-fetching operation. When the convolution filter completely process buffered data of BRAM1, it selects BRAM2 for second buffering operation which further continues data delivery to convolution filter without second pre-fetching delay. Meanwhile it selects BRAM1 for third pre-fetching operation as shown in Figure 6. In this way our system completely hides pre-fetching latency and provides seamless data to 2D convolution filter to attain an ideal throughput of 1 clock/pixel.

## IV. RESULTS AND COMPARISON

This work has presented a compact and efficient buffering architecture for real time NPS. A 128x128 image and a 7×7 window were considered as a case study to describe effectiveness and efficiency of our buffer design concept. The design was tested and evaluated on a new generation of low power Atrix-7 device (Xc7A200t-3fbg484). All the coding, testing and validation has been done using Xilinx Integrated Software Environment (ISE) 14.6.

TABLE II. IMPLEMENTATION RESULTS OF CASE STUDY ON ATRIX7

| Resources | Used |
|---|---|
| Area (Slices + BRAM) | 131+2 |
| Total Power/Frame (m Watt) | 0.038 |
| Dynamic Power/Frame (m Watt) | 0.019 |
| Frequency (MHz) | 127 |
| Frame Rate (fps) | 7751 |

Table II shows the results of our efficient buffer design. The design utilizes only 2 BRAMs (using it to its full capacity of 32Kb) with minimal overhead of supporting circuitry i.e. Main Control Unit (MCU) of just 131 Slices. With reasonable operating frequency of 127 MHz, it occupies minimal memory resources for RBs reported to date with an additional advantage of very low power consumption of 0.038 m W per frame @ 7751 fps.

TABLE III. COMPARISON OF OUR WORK WITH ROW BUFFER (RB) IMPLEMENTATIONS IN NPS

| Architecture | Work | Kernel Size | Buffering Scheme | BRAM | Frequency (MHz) |
|---|---|---|---|---|---|
| **2D Convolver** | [22] | 3x3 | RB | 2 | 118.5 |
| **Morphology** | [16] | 7x7 | RB | 6 | 216 |
| | [20] | 7x7 | RB | 6 | 923** |
| **2D Convolver** | **Proposed** | **7x7** | **C&EB** | **2** | **127** |

TABLE IV. COMPARISON OF BUFFERING SCHEMES ON ATRIX7 FOR OUR CASE STUDY

| | Standard Parameters | | | Additional Parameters | |
|---|---|---|---|---|---|
| | **Throughput (clock/pixel)** | **Bandwidth (pixel/clock)** | **Memory Resources (BRAMs)** | **Frequency (MHz)** | **Power Consumption (mW)** |
| **Row Buffers** | 1 | 1 | 6 | 397 | 15.8*6=94.8 |
| **Our Design** | 1 | ≥2 | 2 | 127 | 15.8*2=31.6 |
| **%Reduction** | - | - | 66% | - | 66% |

Table III compares our results with recent reported implementations of RBs for NPS used in different real time image processing applications [16,20,22]. It shows that all reported implementations follow the same conventional trend of R-1 rows buffering for RxC NPS, where each row was buffered in a separate BRAM. Our efficient buffer design significantly reduces linear requirement of BRAMs for full image buffering as compared to conventional RB implementations reported up till date.

In addition to memory resource utilization in terms of number of BRAMs, few more standard parameters are also important for complete and true performance evaluation and comparison of buffer designs such as system throughput in terms of clock/pixel and external memory bandwidth in terms of pixels/clock [2,23]. Also, to evaluate feasibility of these buffer designs in real time systems especially in emerging power constraint applications such as battery operated imaging devices, few additional parameters are also critical such as operating frequency [24] and power consumption [25] to determine fps and battery life respectively.

It is evident from up to date literature review [16,20,22] that all of these performance parameters except number of BRAMs, are either not reported or specific to their overall image processing architecture and not separately mentioned for row buffering unit only. Secondly performance of few parameters varies from one FPGA device to another and it's not fair to compare them directly for different FPGA devices. Therefore to ensure equivalent functionality for a fair comparison of theses performance parameters, we implemented conventional RBs for our case study along with our design on same targeted device i.e. Atrix-7 (Xc7A200t-3fbg484) and compare both results in Table IV.

First three columns of Table IV compare standard parameters. For our case study (i.e. 7x7 NPS and a 128x128 image) conventional RBs use 6 BRAMs to buffer 6 rows with partial utilization of each BRAM but our buffer design utilize only 2 BRAMs with 100 % utilization of each BRAM. It clearly shows that effective utilization of BRAM in our buffer design saves 66% BRAMs and at the same time maintains an ideal throughput of 1 cycle/pixel.

Additional parameters shown in last two columns of Table IV are critical for real time systems in power constraint environment where we have to keep the balance between power and frequency. They should remain within a permissible range to deploy the design in real time, battery operated and high frame rate systems. Therefore these additional parameters need extra optimization efforts. Unlike to previously reported results without or with very little effort [20] to optimize buffer designs for these additional parameters i.e. frequency or power, we optimally implemented buffer design for these parameters by configuring embedded memory resources (BRAM) with their maximum performance and also by efficiently map the design onto the FPGA device to reduce critical paths of overall design.

Performance in terms of frequency of our buffer design is enhanced by improving the impact of clock-to-out timings of output path of BRAM by using its output primitive registers. These embedded primitive registers are free of hardware cost to hold output data of BRAM in pipeline manner which in turns improves its timing [26] at the expense of an extra clock cycle. This extra clock cycle is added as an initial latency of our design. With these embedded primitive registers, BRAMs can be operated at their maximum frequencies which in turns enhance performance of overall buffer design.

To further enhance design performance, we efficiently map the design onto the FPGA device to reduce critical paths. Conventionally synthesis tool uses Computer-Aided Design (CAD) algorithms that place and route the design onto the FPGA device. These mapping algorithms are heuristic in nature that shows good results but not guarantee the optimal solution [27]. Therefore for optimal solution with reduced critical paths, besides using tool efficiently by choosing optimal mapping strategy for our design, we also manually pack associated logic across utilized dedicated resources of device. This optimization in design placement further reduces critical paths of overall design to enhance performance. It also reduce occupied slices of our design with reduced interconnect usage, which results in further reducing its overall power consumption.

Two above mentioned optimization were applied to both conventional and our proposed buffering design for fair comparison and results are declared in table IV. It clearly shows that our design with minimum BRAM resources operates at a frequency of 127 MHz and meets the requirement of full buffering for any real-time high frame rate (up to HD @ 62 fps) imaging system. This significant reduction in BRAM requirement also reduces down over all

power consumption of design which is a main contributing factor for any battery operated device. As in our design, each BRAM is consuming 15.8mW, so conventional RBs with 6 BRAMs consume 94.8 mW, while our design with 2 BRAMs consumes only 31.6 mW. Hence our efficient buffer design suitably replaces conventional RBs in NPS systems for real time image processing applications especially in battery operated portable devices to increase their performance, portability and battery life.

## V. Discussion

We have presented a compact buffering solution that can replace conventional RBs in any NPS. It uses multi rated clocking in conjunction with an efficient addressing technique to access fixed pattern of multiple neighborhood pixels/clock using single BRAM. This multi rated clocking economizes the buffer design on number of BRAMs, at the cost of limiting overall system performance. Despite of this fact, our design for the case study with performance efficiency of HD@62 fps meets the buffering requirement of high frame rate systems along with minimum BRAMs utilization and the same performance trend applies for any NPS with any image and kernel sizes.

Up to 7x7 kernel size, design does not require any modification, but to extend the design for kernel sizes above 7x7, either we go for higher multiple of clock with same BRAM requirement to cater real time systems with low frame rate requirement (HD @ < 60 fps) or use small multiple of clock with additional BRAMs requirement to cater real time systems with high frame rate requirement (HD @ > 60fps) as shown in Figure 7. A balance between resources (in terms of BRAMs) and overall design performance is obtained at 3x and 4x clock rates as shown in Figure 7, else these parameters can be selected upon demand of targeted image processing application. By keeping optimal choice of clock multiple (i.e. 3x or 4x) for a balanced system as discussed above, BRAMs requirement for our efficient design remains constant for four consecutive kernel sizes and subsequently increases by a factor of 2 for very next four filter sizes, while BRAM requirement for RBs increases linearly with increase in kernel size as shown in Figure 8. It clearly shows that our efficient buffer design reduces BRAMs requirement as compared to RB up to 66% for said case study. This percentage of reduction in BRAMs requirement is further increased with increase in kernel size which verifies effectiveness of our design for larger kernel sizes.

In addition to kernel size, the proposed buffer design is also capable to support any image size for a variety of image processing applications. For 128x128 image size (case study), the design with single BRAM buffers 32 rows (32*128*8 = 32,768bits) at a time and can support up to 7x7 kernel size using optimal multi rated clock. Increasing image size up to 512x512, it can buffer 8 rows (8*512*8 = 32,768bits) at a time and can support up to 7x7 kernel size without any modification. However to extend the design for image size above 512x512, we partition the image into vertical bands [4,5,28] of *W* width (*W<512*) and buffer these vertical bands by using same number of BRAMs as a narrow but complete image at a time. In this way, memory requirement of proposed design will remain same as for smaller images and intact its compactness. However few additional columns with overlap width *'OW'* (*OW= Kernel size - 1*) are required to be fetched on border of each vertical band for the complete neighborhood operation, as explained in Figure 9. Fetching additional columns will not degrade our design performance due to executing buffering and pre-fetching operations separately by using separate BRAMs. Therefore by partitioning the larger images into vertical bands of *128* width (*W*=128); BRAM requirement and system performance in terms of operating frequency of proposed buffer design will remain almost same for any larger image size and consequently frame rate will decrease with increase in image size as shown in Table V.

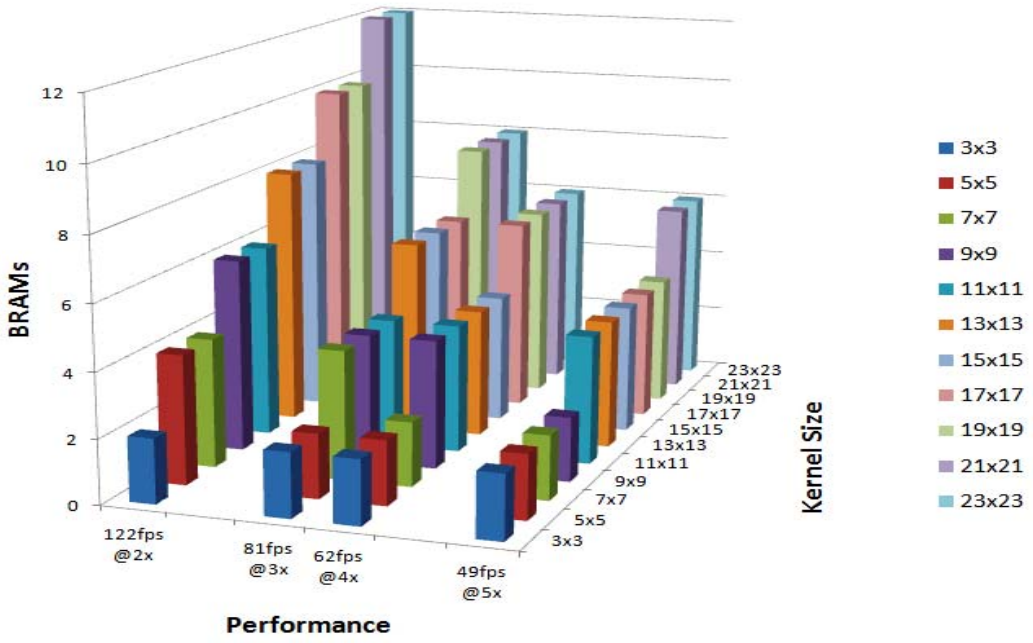


Figure 7. BRAM utilization versus Buffer Performance for different Kernel Sizes

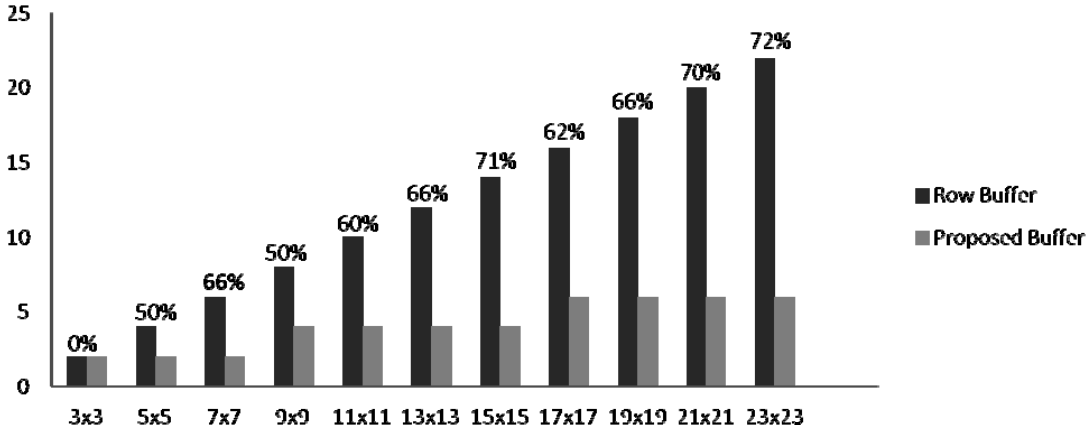


Figure 8. Comparison of BRAM Requirement for different kernel size

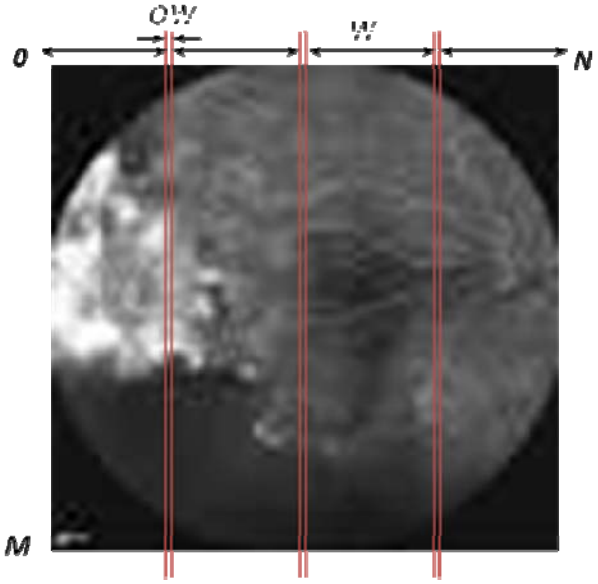


Figure 9. Subdivision of MxN image into 4 vertical bands

TABLE V. SCALABILITY OF OUR DESIGN FOR DIFFERENT IMAGE RESOLUTIONS (KERNEL SIZE IS TAKEN AS 7X7

| Image Resolution | Vertical Band Width 'W' | Overlap Width 'OW' | No. of Vertical Band(VB) | No. of BRAMs | Operating Frequency (MHz) | Frame Rate |
|---|---|---|---|---|---|---|
| 128x128 | 128 | 0 | 1 | 2 | 127 | 7751 |
| 256x256 | 128 | 6 | 2 | 2 | 127 | 1937 |
| 512x512 | 128 | 6 | 4 | 2 | 127 | 484 |
| 1024x1024 | 128 | 6 | 8 | 2 | 127 | 121 |
| 1080x1920(HD) | 128 | 6 | 15 | 2 | 127 | 61 |

## VI. CONCLUSION

In this paper we propose a compact and efficient image buffering architecture for real time NPS with an additional feature of pre-fetching. The buffer design utilizes minimal BRAMs at the expense of small yet efficient Main Control Unit (MCU). The MCU provides multiple pixels of pre-fetched data per clock to NPS in fixed pattern through its optimal multi rated BRAM data accessing technique. It also controls and synchronizes BRAMs operations to maintain an ideal throughput of 1 clock/pixel. The effectiveness of proposed buffering concept is explained with case study (7x7convolver and 128x128 image size) however it is not limited to this case study and efficiently buffer image data in minimum BRAMs for any NPS with any image and kernel size. Our proposed architecture reduces BRAMs requirement as compared to RB up to 66% for said case study. This percentage of reduction in BRAMs requirement is further increased with increase in kernel size along with significant reduction in power consumption. At the same time, it is capable to support buffering for real time systems with high frame rates. Therefore, proposed buffer design suitably replaces conventional RB in any NPS system for real time image processing applications. Additionally, its low power consumption makes it an ideal solution for compact and battery operated portable devices.

## REFERENCES

[1] C.T. Huitzil, M.A. Estrada, "Real-time image processing with a compact FPGA-based systolic architecture," Real-Time Imaging, vol. 10, no. 3, pp. 177-187, June. 2004, http://dx.doi.org/10.1016/j.rti.2004.06.001

[2] H. Zhang, M. Xia, G. Hu, "A Multiwindow Partial Buffering Scheme for FPGA-Based 2-D Convolvers," Circuits and Systems II: Express Briefs, IEEE Transactions on, vol.54, no.2, pp.200-204, Feb. 2007, http://dx.doi.org/ 10.1109/TCSII.2006.886898

[3] Q. Liu, G.A. Constantinides,K. Masselos, P. Cheung, "Combining Data Reuse With Data-Level Parallelization for FPGA-Targeted Hardware Compilation: A Geometric Programming Framework," Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on , vol.28, no.3, pp.305-315, March 2009, http://dx.doi.org/ 10.1109/TCAD.2009.2013541

[4] D. G. Bailey, "Design for embedded image processing on FPGAs", pp. 116, 233, John Wiley & Sons, 2011.

[5] B. Bosi, G. Bois, Y. Savaria, "Reconfigurable pipelined 2-D convolvers for fast digital signal processing," Very Large Scale Integration (VLSI) Systems, IEEE Transactions on , vol.7, no.3, pp.299-308, Sept. 1999, http://dx.doi.org/ 10.1109/92.784091

[6] X. Liang, J. Jean, K. Tomko, "Data buffering and allocation in mapping generalized template matching on reconfigurable systems," The Journal of Supercomputing, vol. 19, no. 1, pp. 77-91, 2001, http://dx.doi.org/ 10.1023/A:1011196613858

[7] K. Wiatr, E. Jamro, "Implementation image data convolutions operations in FPGA reconfigurable structures for real-time vision systems," Information Technology: Coding and Computing, 2000. Proceedings. International Conference on, pp.152-157, 2000, http://dx.doi.org/ 10.1109/ITCC.2000.844199

[8] F.C. Tormo, P.L. Molinet, "Area-efficient 2-D shift-variant convolvers for FPGA-based digital image processing," Circuits and Systems II: Express Briefs, IEEE Transactions on, vol.53, no.2, pp.105-109, Feb. 2006, http://dx.doi.org/ 10.1109/TCSII.2005.857091

[9] T.P. Cao, D. Elton, G. Deng, "Fast buffering for FPGA implementation of vision-based object recognition systems," Journal of Real-Time Image Processing, vol. 7, no. 3, pp. 173-183, 2012, http://dx.doi.org/10.1007/s11554-011-0201-1

[10] M. Schmidt, M. Reichenbach, A. Loos, D. Fey, " A smart camera processing pipeline for image applications utilizing Marching Pixels," Signal & Image Processing: An International Journal (SIPIJ), vol. 2, no. 3, pp. 137-156, 2011.

[11] C.T. Moore, H. Devos, D. Stroobandt, " Optimizing the FPGA memory design for a sobel edge detector," in 20th Annual Workshop on Circuits, Systems and Signal Processing (ProRISC 2009), STW Technology Foundation, 2009, http://hdl.handle.net/1854/LU-864518

[12] C.L. Sotiropoulou, L. Voudouris, C. Gentsos,A.M. Demiris,N. Vassiliadis, S. Nikolaidis "Real-Time Machine Vision FPGA Implementation for Microfluidic Monitoring on Lab-on-Chips," Biomedical Circuits and Systems, IEEE Transactions on , vol.8, no.2, pp.268-277, April 2014, http://dx.doi.org/ 10.1109/TBCAS.2013.2260338

[13] D. Koukounis, C. Ttofis, A. Papadopoulos, T. Theocharides, "A high performance hardware architecture for portable, low-power retinal vessel segmentation," Integration, the VLSI Journal, vol. 47, no. 3, pp. 377-386, June. 2014, http://dx.doi.org/10.1016/j.vlsi.2013.11.005

[14] D. Koukounis, C. Tttofis, T. Theocharides,"Hardware acceleration of retinal blood vasculature segmentation," In Proceedings of the 23rd ACM international conference on Great lakes symposium on VLSI (GLSVLSI '13), ACM, New York, NY, USA,pp. 113-118, 2013, http://dx.doi.org/10.1145/2483028.2483073

[15] T.R. Savarimuthu, A. Kjær-Nielsen, A.S. Sørensen," Real-time medical video processing, enabled by hardware accelerated correlations," Journal of Real-Time Image Processing, vol. 6, no. 3, pp. 187-197, 2011, http://dx.doi.org/10.1007/s11554-010-0185-2

[16] R.M. Gibson, A. Ahmadinia, S.G. McMeekin, N.C. Strang, G. Morison, "A reconfigurable real-time morphological system for augmented vision," EURASIP Journal on Advances in Signal Processing, vol. 1, pp. 1-13, 2013, http://dx.doi.org/10.1186/1687-6180-2013-134

[17] M. Imran, K. Khursheed, N. Ahmad, M. O'Nils, N. Lawal,M.A. Waheed, "Complexity Analysis of Vision Functions for Comparison of Wireless Smart Cameras," International Journal of Distributed Sensor Networks, vol. 2014, Article ID 710685, 15 pages, 2014, http://dx.doi.org/10.1155/2014/710685.

[18] S. Jin, J. Cho, X.D. Pham, K.M. Lee, S.K. Park, M. Kim,J.W. Jeon, "FPGA Design and Implementation of a Real-Time Stereo Vision System," Circuits and Systems for Video Technology, IEEE Transactions on, vol.20, no.1, pp.15-26, Jan. 2010, http://dx.doi.org/ 10.1109/TCSVT.2009.2026831

[19] 7 Series FPGAs Memory Resources, User Guide, v1.10 .1, May2014, http://www.xilinx.xom/support/userguide/ug473-7Series FPGAs Memory Resources.pdf.

[20] D.G. Bailey, "Efficient implementation of greyscale morphological filters," Field-Programmable Technology (FPT), 2010 International Conference on, pp.421-424, 8-10 Dec. 2010, http://dx.doi.org/ 10.1109/FPT.2010.5681450

[21] Virtex-5 FPGA, User Guide, v5.4, 2012, http://xilinx.com/support/userguide/ ug190-Virtex-5 FPGA.

[22] S. Singh, A.K. Saini, R. Saini, A.S. Mandal, C. Shekhar, A. Vohra,"A Novel Real-time Resource Efficient Implementation of Sobel Operator based Edge Detection on FPGA,". International Journal of Electronics, pp. 1-11, 2014, http://dx.doi.org/10.1080/00207217.2014.888782

[23] F.C. Tormo, P.L. Molinet, "Area-efficient 2-D shift-variant convolvers for FPGA-based digital image processing," Signal Processing Systems Design and Implementation, 2005. IEEE Workshop on, pp.209-213, 2-4 Nov. 2005, http://dx.doi.org/ 10.1109/SIPS.2005.1579866

[24] S. Singh, S. Saurav, R. Saini, A .K. Saini, C. Shekhar, A. Vohra, "Comprehensive Review and Comparative Analysis of Hardware Architectures for Sobel Edge Detector," ISRN Electronics, vol. 2014, Article ID 857912, 9 pages, 2014, http://dx.doi,.org/10.1155/2014/857912.

[25] P. Turcza,M. Duplaga, "Hardware-Efficient Low-Power Image Processing System for Wireless Capsule Endoscopy," Biomedical and Health Informatics, IEEE Journal of , vol.17, no.6, pp.1046-1056, Nov. 2013, http://dx.doi.org/ 10.1109/JBHI.2013.2266101

[26] Logi CORE IP Block Memory Generator, Xilinx, v7.3. 2012, http://xilinx.com/support/productguide/pg058- LogiCORE IP Block Memory Generator.pdf.

[27] D. Chen, J. Cong, P. Pan, "FPGA Design Automation: A Survey," Found. Trends Electron. Des. Autom, vol. 1, no. 3, pp, 139-169, January. 2006, http://dx.doi,.org/10.1561/1000000003

[28] N.P. Sedcole, "Reconfigurable platform-based design in FPGAs for video image processing." PhD diss., University of London, 2006.