High Throughput Spatial Convolution Filters on FPGAs

Lenos Ioannou, *Student Member, IEEE,* Abdullah Al-Dujaili, *Member, IEEE,* and Suhaib A. Fahmy, *Senior Member, IEEE*

***Abstract*—Digital signal processing (DSP) on field-programmable gate arrays (FPGAs) has long been appealing because of the inherent parallelism in these computations that can be easily exploited to accelerate such algorithms. FPGAs have evolved significantly to further enhance the mapping of these algorithms, included additional hard blocks, such as the DSP blocks found in modern FPGAs. Although these DSP blocks can offer more efficient mapping of DSP computations, they are primarily designed for 1-D filter structures. We present a study on spatial convolutional filter implementations on FPGAs, optimizing around the structure of the DSP blocks to offer high throughput while maintaining the coefficient flexibility that other published architectures usually sacrifice. We show that it is possible to implement large filters for large 4K resolution image frames at frame rates of 30–60 FPS, while maintaining functional flexibility.**

***Index Terms*—Convolution, digital signal processing (DSP), field programmable gate arrays (FPGAs), image processing.**

I. Introduction

Image and, by extension, video processing entail intensive computations on a large stream of input pixels. A full HD color video streaming at 60 frames per second (FPS) requires a processing throughput of over 124 million pixels per second for each channel. This rate, coupled with the numerous operations required per pixel in a typical vision flow, result in many GOPS for real-time processing. Exploiting parallelism is therefore paramount to achieve real-time system implementation [1]. Spatial filtering, or 2-D convolution, is a fundamental operation used in the initial stages of many vision applications, and as a result, its efficiency significantly impacts higher layers in these applications. Meanwhile, the resolution of images and videos is increasing, with 4K video now commonplace, quadrupling the computational requirements compared to full HD. In addition, the increasing popularity and wider use of convolutional neural networks (CNNs) in a plethora of applications [2] makes spatial convolution even more important. As deeper CNNs with more neurons per layer are developed, the memory required to store weights and biases grows significantly. Hence, most CNN acceleration architectures buffer pixel and weight data in off-chip memory, which breaks the streaming model that is more relevant for real-time streaming video applications. Moreover, some CNNs make use of varying convolution window strides, which reduces computational complexity compared to a streaming filter that processes overlapping windows. Hence, optimizations applied in CNN implementation do not typically apply to streaming video processing, and yet the performance requirements continue to scale.

High performance image-processing on field programmable gate arrays (FPGAs) has been an active field of research, mainly due to the ability of FPGAs to exploit fine and coarse grained parallelism, allowing for tradeoffs between performance and area [3]. The reconfigurability of FPGAs also means that they can provide the flexibility often desired in vision systems. Their high throughput processing, ability to exploit parallelism, and flexibility have led to the wide use of FPGAs in real time vision systems [4]–[6] and to implement a variety of filter structures [7], [8].

A typical vision processing flow moves from pixel-level operations to more abstracted algorithms on less dense and structured data, where software implementations can be a better fit due to ease of programming and irregular data access patterns. Ideally a real-time vision system would therefore couple the high performance of a hardware accelerator, to take advantage of massive parallelism in low level operations, with the programmability of a processor for higher level operations. FPGA SoCs, like the Xilinx Zynq, couple an embedded processor with flexible reconfigurable fabric on the same silicon, with high throughput connectivity between them. FPGA platforms with PCIe connectivity can also be used within a workstation environment alongside a more capable CPU. The reconfigurability of FPGAs, including partial reconfiguration, also allows them to support dynamic vision systems where the hardware can adapt at runtime to changing conditions [9]. Hence, FPGAs are ideal for implementing the full computer vision stack including higher level software and low-level hardware in a broad range of domains, from distributed embedded computing to high performance servers. Within this context, we explore generalized convolution architectures to maximize the throughput of low-level operations within a typical vision flow for high bandwidth video streams.

Convolution, or 2-D spatial filtering, is computed by initially performing a pixelwise multiplication of each pixel within a window with a corresponding coefficient, followed by a function that aggregates these products to produce a single output [10]. Both of these functions may vary for different filter applications. The coefficients used in the pixelwise multiplication define the filter's operation, which can be, for ex-

Manuscript received November 26, 2019; revised March 10, 2020; accepted March 25, 2020. This work was supported in part by the U.K. Engineering and Physical Sciences Research Council (EPSRC) under grant EP/N509796/1.

Lenos Ioannou and Suhaib A. Fahmy are with the School of Engineering, University of Warwick, Coventry CV4 7AL, U.K. (e-mail:l.ioannou@warwick.ac.uk; s.fahmy@warwick.ac.uk).

Abdullah Al-Dujaili is with Analog Devices, Norwood, MA 02062 USA (e-mail: ash.aldujaili@analog.com).

ample, noise removal, image sharpening, blurring/smoothing, or feature extraction. With real-time vision systems typically deployed on streaming images, input data flow becomes simpler, not requiring storage of complete frames. It is, however, important that the computations within the convolution meet the real time constraints to avoid becoming a bottleneck.

Increasing image resolution in mainstream use means that the dimensions of convolution windows must also scale in order to maintain their effectiveness. For example, a 3×3 filter applied to a 1280 × 720 image is equivalent to a 9 × 9 filter for a 3840 × 2160 image if spatial equivalence is required. This, consequently, results in increased workload on more input pixels that in turn requires more demanding processing in order to maintain the same frame rate.

Most previous work on FPGA-based spatial filters has focused on optimizations based on the use of fixed coefficients or coefficients constrained to a specific range [11]. Such optimizations are most effective for filters in which the coefficients consist of zeros or ones, or other powers of two, since each multiplication can be replaced with a shift, resulting in no use of multipliers [12], and much improved hardware area efficiency. This comes at the cost of flexibility as those systems are fixed to a single purpose. Fixed filter implementations, however, are not ideal for smart vision systems, in which filters at the lower layers should be flexible enough to dynamically adapt to different requirements. Flexible convolution architectures use generic multipliers while providing external access to the coefficient registers to support dynamic adaptability.

In this article, we show how the considered use of high performance DSP blocks in generalized filter architectures can achieve high throughput that meets real-time constraints. Compared to previous work, this article focuses on maximizing the throughput of spatial filtering on FPGAs while maintaining dynamic coefficient adaptability though external register access. More specifically, the proposed architecture is built to fully exploit the DSP block resources on modern FPGAs while managing the required data buffering and architectural pipeline for 2-D filtering, and being scalable to large filter and frame sizes. Optimization around the FPGA architecture enables the proposed filters to achieve higher operating frequency and, as a result, higher throughput than the previous work.

A. Generic Filter Architecture

A typical filter architecture and its functional blocks are depicted in Fig. 1. It operates in streaming mode, receiving a new pixel from the source image in each clock cycle, in raster scan order. To compute an output pixel, all pixels within the corresponding input window must be available. For $w \times w$ filter sizes, where w is an odd number, pixels from w rows are required to compute each output pixel. This requires a row buffer with the ability to store $w-1$ rows plus w pixels (since only w pixels are needed from the last row). Full frame buffering is, therefore, not needed for streaming images, something that would consume significant area for large frame sizes, and possibly require frequent off-chip memory accesses, which can have a significant performance and power consumption overhead.

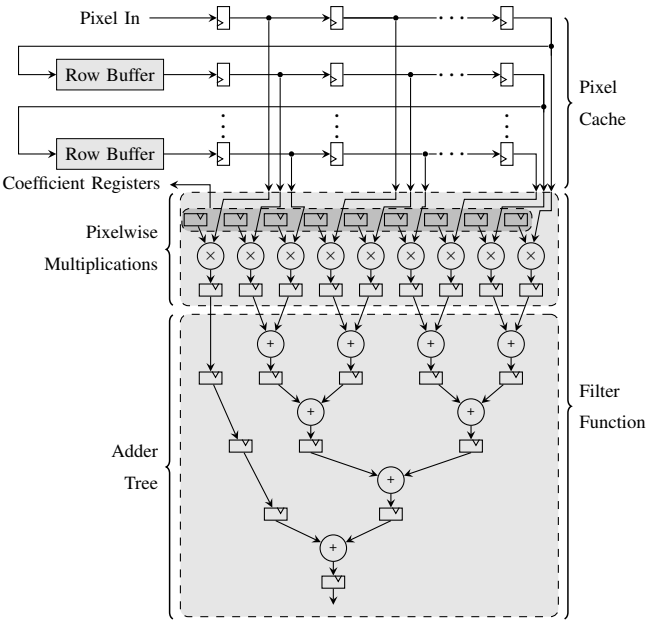


Fig. 1: Filter architecture block diagram.

The pixels in the $w \times w$ filter window from the input image, formed around the calculated pixel, are stored in the pixel cache block as shown in Fig. 1. The pixel values are then fed to the pixelwise multiplications block within which they are multiplied in parallel with the corresponding coefficients, which are stored in registers and can be configured at runtime. This enables modification from the higher layers of a complete vision stack as described previously. The filter's operating mode is controlled by a state machine that cycles between idle, coefficient update, priming, processing, and flushing modes. Output data are streamed at the same rate as the input pixels, maintaining simple data movement with no need for storing complete frames.

Numerous approaches toward more efficient filter designs have been proposed in the literature. These mainly target the core underlying multiply-accumulate (MAC) operations and are generally divided into two categories, multiplier-based and multiplierless filters. Multiplier-based filters directly map the multiplication to hardware multipliers. Park et al. [13] proposed a sharing scheme targeting vector–scalar multiplications through decomposing FIR filters. Bougas et al. [14] use internal pipelining in multiplier arrays to fold FIR filters. Ma et al. [15] reduce the computational complexity of 2-D convolutions by splitting large filter windows to a sequence of convolutions with smaller window sizes.

Multiplierless filters avoid the use of multipliers through various arithmetic transformations and representations. These include programmable canonic signed-digit (CSD) representation in [16], and distributed arithmetic (DA) in [17] and [18] that replaces multiplications with lookup table memories and adders. The Bachet weight decomposition theorem is used in [11] to similarly replace multipliers with ROMs and adders. Other multiplierless methods tailor their architecture to the filter function with hardwired shifts [19] or make use of powers-of-two weights for multiplierless CNN inference [12].

B. FPGA DSP Block Architecture

FPGA DSP blocks have evolved significantly since their introduction into FPGA architectures, from simple multiplier blocks to fixed MAC capability, to programmable multiplication and arithmetic/logic in modern devices. While DSP block availability was limited in older FPGA platforms, even low end FPGAs today include sufficient DSP blocks to implement large filters. DSP blocks have also evolved to support wider input lengths, additional input ports, incorporation of a pre-adder, pattern detection capability and SIMD support. For instance, the DSP48E1 block on Xilinx 7 Series FPGAs supports multiplication, MAC, multiply-add, add-MAC, and three-input-add functions, among others. Interconnect between DSP blocks has also been improved, with modern FPGAs now offering dedicated cascade interconnect between DSP blocks that allows wider computations and chaining of DSP blocks to form 1-D FIR filters without using the logic fabric, hence achieving higher performance. Although connectivity enhancements benefit 1-D FIR filter implementations, 2-D structures cannot fully exploit these features, so data movement and buffering must be done manually in the logic fabric. Dynamic programmability is another feature of the DSP48E1, where it is possible to adjust ALU function (*ALUMODE*), operation mode (*OPMODE*) and input selection (*INMODE*) dynamically at runtime. This has led to their use in lightweight soft processors [20] and flexible overlay architectures [21], [22] where their functionality can be modified on a cycle-by-cycle basis or to reconfigure functional units, all while achieving high throughput.

However, to exploit these features and achieve high performance, designs must be optimized at a low level as synthesis tools are often unable to infer the best structures for complex designs [23]. Indeed, since DSP blocks can be clocked significantly higher than the typically achievable frequency in complex designs, it is possible to share them in a time-multiplexed manner through multipumping, where they are clocked at a multiple of the surrounding logic and multiple operations are mapped to a single DSP block per cycle with suitable buffering [24].

C. Boundary Handling

While the convolution window scans the input image, its computation becomes more complex when it targets pixels at the edges. This calls for particular handling or padding as the convolution window requires pixels that do not exist, as shown in Fig. 2. An alternative is to restrict the sliding window within the valid region of the input frame, which results in an output frame of reduced size compared to the input frame. A scalable architecture should be able to support the addition of border management schemes without a significant impact on performance.

This article proposes a flexible design that uses DSP blocks for the pixelwise multiplication, alongside various design options for the adder tree. In order to achieve high throughput, all filter architectures are extensively pipelined. We make comparisons between direct and transposed form filters and also investigate the impact of border management. The proposed architectures achieve high throughput by operating at near the DSP block theoretical maximum frequency, while also maintaining an adaptable convolution architecture with coefficients that can be updated at runtime. We show how the baseline architecture scales to three widely used video resolutions: HD (1280×720), full HD (1920×1080) and 4K (3840×2160), on a range of filter sizes that span from 5×5 to 25×25. Lastly, we compare our design with equivalent filters generated using high level synthesis (HLS) tools and with previous work found in the literature.

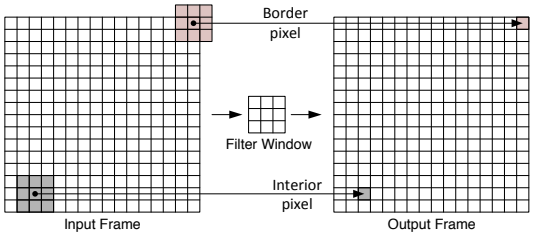


Fig. 2: Filtering for interior and border pixels.

II. Filter Architecture

The proposed filter architectures reflect the general architecture described in Section I-A, with a variety of architectural optimizations to achieve high throughput. Here, we discuss the details of the design.

The general structure of a filter, comprising pixel cache, coefficient multiplication, and adder tree was shown in Fig. 1. Filters can also be implemented in transposed form, in which the incoming sample is multiplied by all coefficients and products are summed serially through the delay line. Transposed form architectures have the advantage of being pipelined by default [10] compared to the manual pipelining required in direct form. FPGA DSP blocks have the required functionality and connectivity to enable 1-D transposed form filters to be implemented using only DSP blocks with no external logic. And hence, these designs have reduced resource utilization and improved performance. For 2-D filters, however, data is buffered across multiple rows and as a result buffering is more complex and cannot be implemented directly using DSP blocks. Direct form architectures require a separate adder tree, which consumes additional resources and power. The adder tree depth depends on the filter size and scales by log_2 of the filter size w. Although our proposed architecture is in direct form, we present a comparison against a transposed form implementation for completeness.

A. Filter Function

The utilization of DSP blocks in our proposed architecture has been made through direct instantiation. Although FPGA vendor tools are able to infer the use of DSP blocks from RTL code, their efficiency decreases for more complex structures as this automated inference does not fully exploit all DSP block features or always suitably pipeline them [24]. Through direct instantiation, we are able to control low-level DSP block mapping and ensure high throughput.

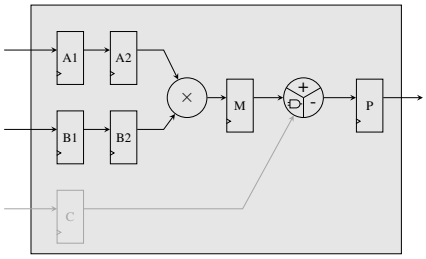


Fig. 3: DSP48E1 block diagram for multiplication.

B. Pixel Cache

The filter cache consists of row buffers and individual registers for the pixels in the active window, which are connected to the coefficient multipliers. The number of row buffer units and individual registers depends on the filter size while the length of the row buffers depends on frame width. Hence, higher frame resolution is more demanding in terms of buffering, utilizing more memory elements. Shift register look-up-tables (SRLs) are one way of more efficiently implementing line buffers on FPGAs, since pixels in the line buffers do not need to be accessed until they reach the filter window. By utilizing a suitable coding style, it is possible to ensure the line buffers are implemented using SRLs [25]. To demonstrate the impact of this optimization, a six-row buffer (for a 7×7 filter size and 1280 wide frame) utilizes over 61 000 flip-flops or only 110 flip-flops and 1920 LUTs when implemented using SRLs. The savings introduced through use of SRLs contributes to the scalability of the proposed architecture. SRLs do impact achievable frequency, lowering it from 700 to 600 MHz for this isolated experiment, but this is in line with the capabilities of the DSP block and so not a limiting factor in the proposed architecture, while offering a significant area saving. Moreover, the higher resource utilization of the register based implementation can have an adverse impact on the placement and routing process in a larger design, resulting in longer routing delays for other parts of a design [25].

C. Adder Tree

In a direct form filter implementation, DSP blocks configured as multipliers, as shown in Fig. 3, are used to calculate the pixelwise products while a separate adder tree follows to sum these products up and generate an output pixel value. We explore three different types of adder trees, as shown in Fig. 4. More specifically, the three layouts are as follows.

1) *DSP Layout:* The adder tree comprises directly instantiated DSP blocks configured as wide adders. Since this operation is mapped directly to silicon, we use this as a baseline for maximum performance. In this case only the postadder in each DSP block is used.
2) *LOG Layout:* The adder tree is mapped to the FPGA logic fabric. This results in a more balanced utilization of the device resources. Each adder in this layout is followed by a register, resulting in a pipelined architecture that is mapped to LUTs and registers.
3) *DSPCOMP Layout:* A compression component, mapped to the FPGA logic fabric, is used to reduce the depth of the adder tree while also using fewer DSP blocks. More specifically, the logic-based compressor (6:3) takes six operands and generates three partial sums, which are then summed using two DSP blocks.

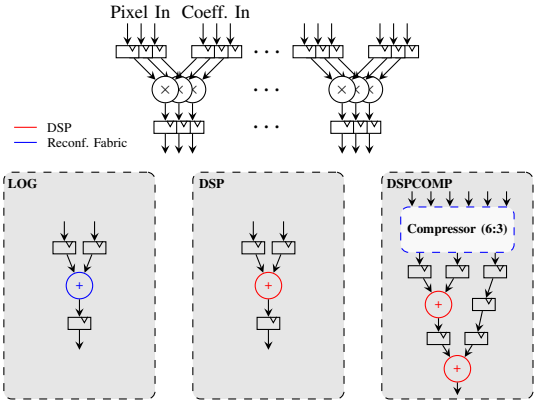


Fig. 4: Alternative adder tree layouts: LOG, DSP, and DSPCOMP.

Using hardened DSP blocks for the multiplier means that wordlength can be chosen to use the maximum available within the structure without impacting area significantly. The filter architecture uses 14 fractional bits. For the pixelwise multiplications, the input pixels are mapped to the 25-bit inputs of the DSP48E1 blocks while the coefficients are mapped to their 18-bit inputs. In every case, the output of each multiplication is reduced from 48 to 24 bits and then fed to the adder tree. In the first stage of LOG- and DSP-based adder trees, 24-bit additions are performed, while in the following stages, 48-bit additions are performed. The reduced input wordlength at the first stage is taken advantage of by adjusting the adders' wordlength in the LOG filter, and by mapping two 24-bit additions to a single DSP block in the DSP filter, reducing the number of DSP blocks utilized. In the DSPCOMP, 45-bit additions are performed throughout the adder tree due to the fixed wordlengths. The 24-bit inputs are sign extended accordingly before being fed to the DSPCOMP adder tree.

Information about the DSP block utilization and latency for each adder tree layout is summarized in Table I, and dataflow

TABLE I: Adder tree layout resource consumption.

	Architecture		
	DSP	**LOG**	**DSPCOMP**
Number of Inputs	2	2	6
Basic Units	1 DSP48E1	LUTs	2 DSP48E1s LUTs
Latency	3	1	10
Number of adders for $w = 7$	48/36*	48	10
Number of stages for $w = 7$	5	5	3

*without/with SIMD mode

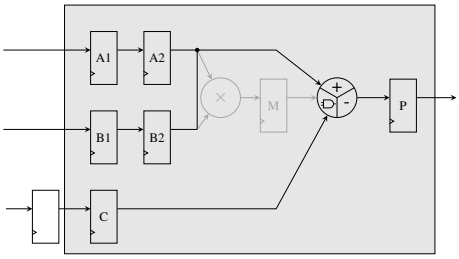


Fig. 5: DSP48E1 block diagram for addition.

is shown in Fig. 4. To achieve high throughput, each adder tree is extensively pipelined. The adders implemented in DSP48E1 blocks have a latency of three clock cycles, as shown in Fig. 5, the compression logic requires two additional clock cycles to generate its partial sums while adders mapped on the FPGA fabric have a latency of a single clock cycle.

Transposed form filters are formed by reversing the signal flow while rearranging the building blocks of direct form filters accordingly. This results in an inherently pipelined adder tree [10]. Transposed form filters in 1-D are fully supported by the DSP blocks, since an MAC operation can be mapped to a single block, and therefore requires no external logic. Although in 2-D structures some logic is required, this does not fully nullify the savings introduced.

The estimates of DSP block utilization in a transposed form filter function for a $w \times w$ convolution window are outlined in TableII. The DSP block utilization shows those used in the individual pixel multiplications and the adder tree separately. In the direct form design with DSP adder tree, the dual 24-bit SIMD mode (two two-input adders) is used at the first stage of the adder tree to pack two additions in a single DSP block, as described earlier. Although the 25-bit preadders in DSP blocks usually offer a more efficient utilization of the DSP resources, they can be used only in the first stage of the adder tree where 24-bit additions take place, so mapping two 24-bit additions to the 48-bit postadder was preferred. This leads to the same DSP block utilization, while maintaining a more straightforward interconnect with no need for delay buffers. In the direct form design with LOG adder tree, the adder tree is mapped to the FPGA fabric, using no DSP blocks. The compressor (6:3) in the direct form design with DSPCOMP adder tree generates three partial results that are summed up using two DSP blocks, at the expense of some logic utilization, compared to the five DSP blocks required in the direct DSP. The proposed architecture does not include any mechanism to handle overflow and as a result inaccuracies may occur in cases where the intermediate results overflow.

TABLE II: DSP Block usage for different configurations for a filter size of $w \times w$.

	DSP Block Usage		DSP blocks for
	Mult. Block	Adder Tree	$w = 7$
Direct			
DSP	w^2	$\frac{w^2-1}{4} + \frac{w^2-1}{2}$	85
LOG	w^2	-	49
DSPCOMP	w^2	$\lceil 2\times \frac{w^2-1}{5} \rceil$	69
Transposed	w^2		49

III. Border Management Techniques

Border management techniques handle the undefined regions of an input pixel stream to produce an output of the same size. Spatial filters can be implemented without border management, generating output images with reduced size. In 1-D filters, this affects only the very beginning of the input signal, and the first outputs can be ignored. In two dimensions, however, this affects every output frame, reducing frame size. Although this may not be an issue for all applications, there are occasions where this can be problematic, such as when a sequence of filters is used to process an image. In CNNs for machine learning, border management is not usually required, since as data propagate through the neural network, the convolutions are applied to more abstracted features and these edge pixels have little or no impact. Filters with no border management have simpler control logic and data flow, allowing a straightforward implementation of a transposed form filter.

The complexity introduced by border management has resulted in a body of work on mitigating its effects. A review of 2-D border handling methods on FPGAs is presented in [26], where the authors also introduce a novel border handling management scheme with overlapped priming and flushing. In this method, registers acting as temporary pixel buffers and multiplexers are used to reduce the time overhead in handling border pixels. Bailey and Ambikumar [10] proposed two novel border handling mechanisms, transformation coalescing, and combination chain modification, that reduce the complexity of border handling in transposed form filters while taking advantage of the inherent pipelining of the transposed form structure. Another approach in [27] considers symmetric extension for 1-D signal border management by exploiting the SRL16 shift register primitives in Xilinx FPGAs to skew data. This technique, however, is not ideal for DSP block based filters as it introduces shift registers between the multiplication and addition, preventing efficient mapping to DSP blocks.

Table III summarizes some of the methods used for border handling along with notes on their advantages and disadvantages. Further details on these methods can be found in [1]. The techniques in the last three rows of Table III are graphically illustrated in Fig. 6. These approaches are more widely used compared to others. Mirroring, for instance, is used in [26]–[28].

These techniques can be implemented in hardware in a number of ways, as briefly described in Table IV, while [26] offers a more detailed discussion. Direct window input and cached priming add extra stalling cycles when processing border pixels, reducing their efficiency. This also complicates the streaming data flow and, as a result, the data-path control logic in real-time systems. The overlapped priming and flushing schemes on the other hand, both naive and the scheme pro-

TABLE III: Image border management techniques.

Advantages	Disadvantages
Border Neglecting	
No additional control logic	Reduced frame size which can be problematic for small frames or when cascading filters
Wrapping	
Small control logic, Same frame size	Possible discontinuity and artefacts
Function Change	
Same frame size	Complex control logic, difficult to generalize to all filters
Constant Extension	
Same frame size	Discontinuity, artefacts, additional control logic
Border Duplication	
Same frame size	Discontinuity, artefacts, additional control logic
Mirroring with/without Duplication	
Same frame size	Additional control logic

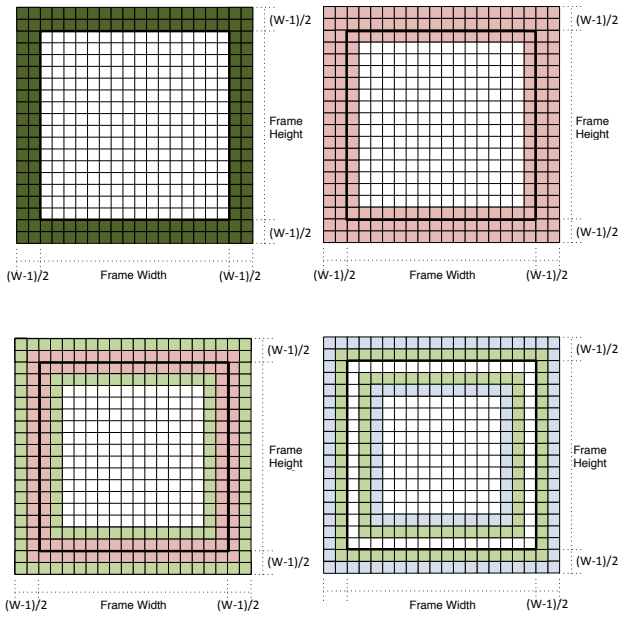


Fig. 6: Border management techniques (top left: constant extension, top right: border extension, bottom left: mirroring with duplication, bottom right: mirroring without duplication).

posed in [26], preserve the regular streaming flow at the cost of additional logic. More specifically, additional multiplexers are required to make the replacement values immediately available. The naive scheme uses extra row buffers along with the additional temporary registers within the window pixel cache, requiring additional memory components. All these techniques are modifications to the pixel cache block in the filter architecture.

This article does not focus on border handling techniques and their optimization, but we demonstrate that our proposed architecture can be extended with these techniques without significant impact on performance or efficiency. We have considered border extension using the overlapped priming and flushing scheme for comparison.

TABLE IV: Image border handling technique implementations in hardware for a filter size of $w \times w$.

Advantages	Disadvantages
Direct Window Input	
No modifications to pixel cache	Complex address generation logic, stalling input stream when processing border pixels by $(w-1)$ between rows and frames for priming and flushing
Cached Priming	
No complex address generation logic	Stalling input stream when processing border pixels by $(w-1)/2$ between rows and frames for flushing, requires extra multiplexers
Naive Overlapped Priming & Flushing	
No stalling, no complex address generation logic	Extra multiplexers, extra temporary pixel buffers within pixel cache, and extra temporary row buffers
Overlapped Priming & Flushing (proposed in [26])	
No stalling, no complex address generation logic, no extra temporary row buffers	Extra multiplexers, extra temporary buffers within pixel cache

IV. Proposed Architecture Results

This section presents the implementation results of the proposed filter architecture, showing operating frequency and throughput as well as area and latency (which represents the number of clock cycles required for the first output pixel to be generated), for different design parameters. We first investigate our proposed architecture in detail using an indicative filter with a 7×7 window for 1280×720 frames, which we use to make comparisons between the adder tree types in direct form, the direct and transposed forms, against an HLS equivalent, and the impact of border management on area and performance. We then show how the architecture scales on three frame sizes, 1280×720, 1920×1080 and 3840×2160, for 11 different filter sizes, ranging from 5×5 to 25×25. Finally, we compare the proposed architecture against published work in the literature. All our designs were implemented in Verilog HDL, using Vivado 2018.2, targeting the Xilinx Virtex 7 XC7VX690 on the VC709 development board and the results presented in Sections IV-A– IV-F are post place and route.

A. Adder Tree Designs in Direct Filter Structure

We explore the effect of three different adder trees, **DSP**, **LOG**, and **DSPCOMP**, as described in Section II without considering border management. Table V summarizes the operating frequency and latency for the three designs, all offering high throughput at similar frequencies. The LOG filter is slightly slower with marginally improved latency.

Table VI shows the resource utilization for all designs. All filters use 49 DSPs for the pixelwise multiplication, as shown earlier in Fig. 1. The LOG design does not use any

TABLE V: Frequency and latency of direct form filter implementations with different adder tree designs for 1280×720 frame, 7×7 filter and no border management.

Adder Tree Design	Freq. (MHz)	Latency (Cycles)
DSP	535	7713
LOG	525	7700
DSPCOMP	530	7725

TABLE VI: Resource utilization of direct form filter implementations with different adder tree designs for 1280×720 frame, 7×7 filter and no border management.

Modules	Adder Tree	Resource		
		Regs	LUTs	DSPs
Coeff. File	DSP	735	0	–
	LOG	735	0	–
	DSPCOMP	735	0	–
Control Unit	DSP	49	59	–
	LOG	49	63	–
	DSPCOMP	49	59	–
Pixel Cache	DSP	440	1920	–
	LOG	440	1920	–
	DSPCOMP	104	1920	–
Filter Func.	DSP	4516	12	85
	LOG	4539	1464	49
	DSPCOMP	8954	1120	69
Total	DSP	5768	1991	85
	LOG	5791	3447	49
	DSPCOMP	9870	3099	69

DSPs in the adder tree, while the DSP design uses 36 and the DSPCOMP design 20 DSP blocks when $w = 7$. These results correspond with the estimates in Table II. When comparing the total resource utilization, we see that DSPCOMP utilizes the most registers while ranking second for LUT and DSP utilization. Since all filters operate at almost the same frequency, DSPCOMP can be considered the least efficient when considering utilized area. We observe that LOG utilizes about 73% more LUTs compared to DSP while using 42% fewer DSP blocks and approximately the same number of registers. Considering the availability of such resources in modern FPGAs, DSP blocks are the least abundant, while the register-to-LUT ratio is 2 to 1. Therefore, the resource mix of the LOG filter better mirrors the FPGA architecture and utilizes the fewest DSP blocks. Trading 1456 LUTs and 23 registers for 36 DSP blocks is a net positive in area terms based on the approximate 120:1 ratio of resources on the device. This configuration allows better replication of parallel filters for different streams while not utilizing more DSP blocks than are necessary, and achieving almost identical performance.

B. Direct Versus Transposed Form Architectures

Table VII summarizes the resource utilization, maximum operating frequency, and latency of the direct form (with LOG adder tree) and transposed form architectures. The transposed form filter structure combines the pixel cache and filter function into a single module so separate results are not shown. In terms of performance, both filters have similar latency while the direct form operates at a slightly higher frequency. Direct form uses significantly more registers than the transposed form while using about half as many LUTs. The majority of LUTs in the transposed form filter are utilized by the combined filter function and pixel cache modules. The LUTs in this case are solely used as shift registers (SRLs) for buffering, affected primarily by image width. The transposed form architecture does not require an adder tree, instead using an adder chain that can be packed into the same DSP blocks that implement the individual multipliers. As a result, the DSP utilization of both filters is the same. Additional logic is still required, however, for the 2-D transposed form filter as the dedicated cascade wires offered by the DSP blocks are only suitable for 1-D structures. The direct form LOG design implements the adder tree in the FPGA logic fabric.

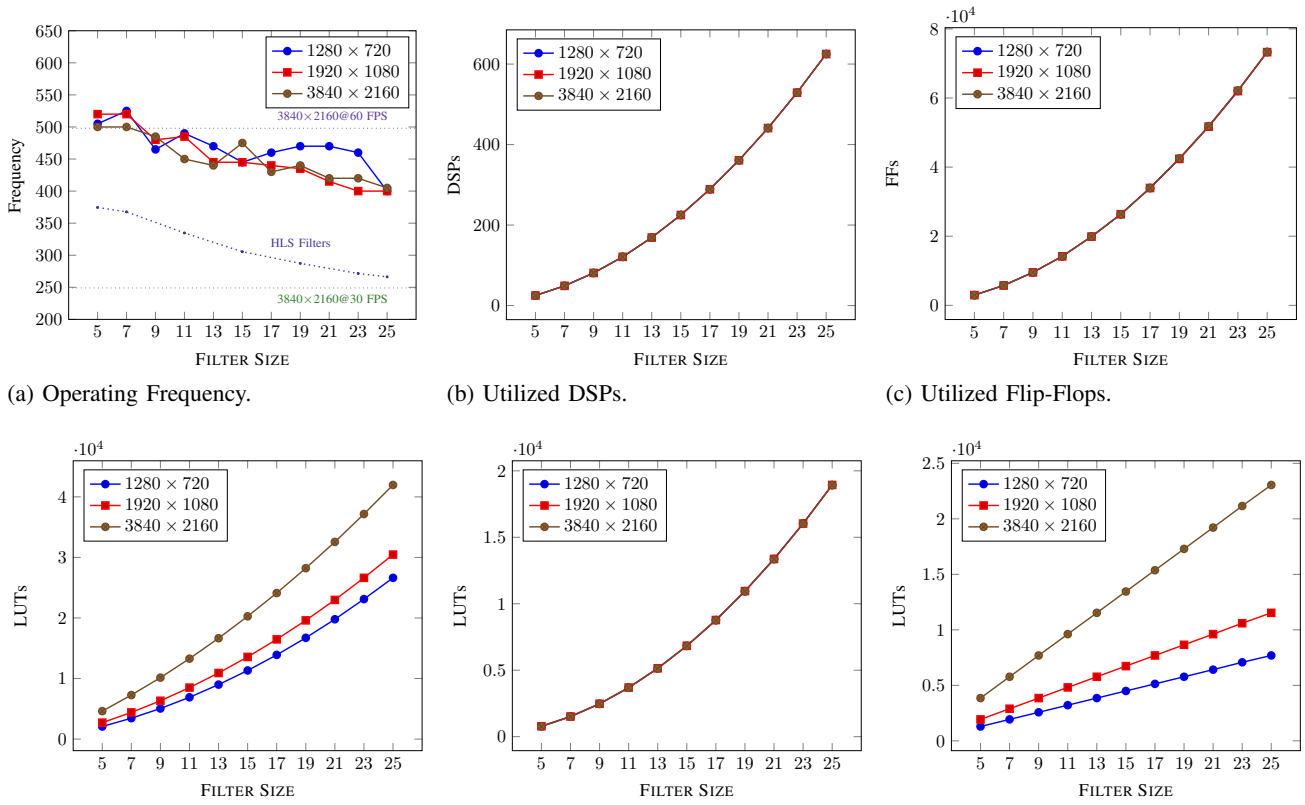
TABLE VII: Direct and transposed form implementation summary with 1280×720 frame and 7×7 filter.

Module	Direct LOG			Transposed		
	Regs	LUTs	DSPs	Regs	LUTs	DSPs
Coef. File	735	0	-	735	0	-
Control Unit	49	63	-	83	42	-
Pixel Cache	440	1920	-	918	7200	49
Filter Func.	4539	1464	49			
Total	5791	3447	49	1763	7242	49
Freq. (MHz)	525			505		
Latency (Cyc)	7700			7691		

While both filters have similar performance, with the direct form operating at slightly higher frequency, their resource utilization varies significantly. The resource utilization of LOG has a better register to LUT ratio as discussed previously. The transposed form filter architecture is also less extensible to support border handling as pixel values within the window are already accumulated with other pixels. This issue has been discussed in [27], where the use of shift registers is proposed as a solution. This approach however separates the multiplication and addition, making the resource utilization similar to the direct form. Two novel border handling techniques are proposed in [10] to reduce the border handling complexity in transposed form filters. These methods result in designs of similar complexity to the direct form ones. Hence, transposed form filters must sacrifice their efficiency to offer scalability and support for border handling. We show that direct form filters can be suitably pipelined to achieve equivalent performance. Finally, as discussed earlier, the resource utilization mix of the direct form filter better mirrors the resource availability on modern FPGAs.

C. Direct Filter Structure With Border Management

Although border management is not the focus of this article, we explore the use of the overlapped priming and flushing



(a) Operating Frequency. (b) Utilized DSPs. (c) Utilized Flip-Flops.

(d) Utilized LUTs. (e) Utilized LUTs as Logic. (f) Utilized LUTs as shift registers.

Fig. 7: Implementation results of the proposed filter architecture on three image resolutions, each on 11 filter sizes.

TABLE VIII: Direct LOG architecture for 1280 × 720 frame and 7 × 7 filter with border policy from [26].

Modules	Direct LOG		
	Regs	LUTs	DSPs
Coef. File	735	0	–
Control Unit	63	65	–
Pixel Cache	652	1657	–
Filter Func.-LOG	4542	1464	49
Total	6020	3186	49
Freq. (MHz)	515		
Latency (Cycles)	3860		

scheme presented in [26]. As some applications may require the use of border management, this section demonstrates the extensibility of the proposed architecture to support this feature. Moreover, we quantify the overhead introduced, as a result of the increased design complexity. Table VIII summarizes the implementation results of the direct form LOG filter with border management on a 7 × 7 filter for a 1280 × 720 frame size.

Compared to the direct form LOG design without border management, the border extension architecture uses fewer LUTs and more registers, while DSP block utilization remains the same. Of particular interest is the LUT reduction in thePixel Cache module within the border management architecture. Its LUT utilization amounts to 376 LUTs for logic and 1281 LUTs for SRLs, meanwhile the same module without border extension uses all 1920 LUTs as SRLs. The straightforward flow of data without border management enables the synthesis tool to map the Pixel Cache into SRLs. In contrast, border management requires additional logic for its implementation and its more complex data flow is mapped to both SRLs and registers, contributing to higher utilization of slice registers. Border management reduces frequency marginally due to the increased complexity of the design and increased routing congestion. The latency of the first output pixel is decreased, as expected, since required pixels are present in about half as many clock cycles as in the baseline design. For example, when a 7×7 filter is used, only four of the first seven lines need to be buffered for the computation to start, since the out of frame pixels in the window are replicated from those pixels. Without border management, no output pixel is produced until seven lines are buffered.

D. Comparison With Vivado HLS Filters

HLS is increasingly gaining popularity due to its higher level design abstraction compared to HDL, enabling faster design time and functional verification of hardware accelerators. Ease of implementation in HLS, however, can come at the cost of reduced performance and possibly poorer resource efficiency, especially when considering processing patterns that map well to low-level architectural features like the DSP

blocks. We used Vivado HLS 2018.2 with the image processing libraries provided by Xilinx to explore how resource utilization and throughput scales for filters generated from high level code, assuming a 1280×720 frame size. Pragmas were used to unroll and pipeline the computation of the HLS filters in order to enable streaming processing, reading, and outputting a pixel in each clock cycle. The achieved frequency for these designs is plotted with a dotted line in Fig. 7a. All filter coefficients are configurable, resulting in the same functionality and DSP block utilization as the proposed filter architecture. Coefficient wordlength was set to 18 bits, and inputs and outputs were set to 8 bits with 16-bit intermediate results. The reduced wordlength compared to our proposed architecture as described in Section II, reduces area somewhat, but allows the tool to generate high throughput filters for more competitive comparison. In Table IX, we summarize the relative change in resource utilization and frequency for the HLS filters compared to our proposed architecture, for the same parameters, compared to our filter architecture as the baseline. The reduced wordlengths result in reduced resource utilization compared to the proposed filter architectures. This becomes more apparent as the filter size increases, resulting in deeper and wider adder trees, which in turn increases the difference in overall resource utilization. The purpose of this comparison, however, is primarily throughput and we see that the HLS filters have at best 25% lower throughput and up to 40% less for larger filters. This demonstrates the effectiveness of our proposed architecture for high throughput applications, where some area overhead can be tolerated.

TABLE IX: Relative resource utilization and frequency for Vivado HLS filters.

	Filter Size ($w \times w$)						
	5	7	11	15	19	23	25
Regs	−36.83	−41.29	−59.71	−65.07	−66.04	−67.54	−68.20
LUTs	−15.79	−22.74	−22.72	−28.71	−34.19	−39.91	−42.86
DSPs	0	0	0	0	0	0	0
Freq. (MHz)	−25.84	−29.97	−31.68	−31.34	−38.86	−40.96	−33.42

E. Scalability Analysis

Spatial filters are widely used in configurations with different filter and frame sizes as required for a variety of vision applications. We therefore explore performance and resource utilization when scaling the proposed architecture to three frame sizes, 1280×720, 1920×1080 and 3840×2160, with 11 filter sizes ranging from 5×5 to 25×25. Results are illustrated in Fig. 7a– 7f. Operating frequency varies from 525 to 400 MHz, decreasing as the filter size increases, due to the critical path resulting from a wider adder tree and routing of more coefficient products. The frequency fluctuations are a result of the critical path moving through various parts of the adder tree as the architecture grows, leading to routing congestion. Meanwhile, frame size has minimal impact on operating frequency as it primarily affects the size of the line buffers, which are not in the critical path. These results compare favorably with the DSP theoretical maximum frequency of 650 MHz on this Xilinx Virtex 7 device [29].

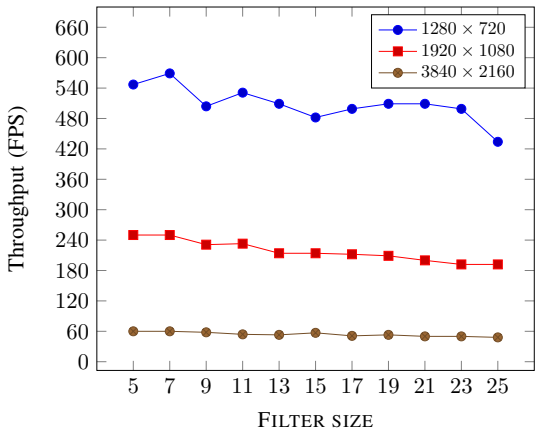


Fig. 8: Achievable frame rates for varying filter and frame sizes.

DSP block and flip-flop utilization are dependent on filter size rather than frame size. DSP blocks are explicitly instantiated for the multiplication of window pixels with the filter coefficients, and hence are filter size dependent. Flip-flops are used mainly for pipelining the computational datapath, which in turn depends on the filter size. LUT utilization is more complex, depending on both the filter and frame size. To further analyze the scaling pattern, we show the LUTs utilized as logic in Fig. 7e and the LUTs utilized as SRLs in Fig. 7f. LUTs as logic are primarily in the adder tree and additionally in the control logic. LUTs as shift registers are used primarily in the line buffers and as a result their utilization depends primarily on the number of lines that need to be buffered and also on the width of those lines.

We finally show how the operating frequency translates to throughput in FPS in Fig. 8. All designs perform well over the 30 FPS required for real-time processing, even on 4K videos. 60 FPS is achieved by the majority of designs, except 9×9 and larger filters on 4K frames, achieving 58 FPS for the 9×9 filter, and as low as 48 FPS for the 25×25 filter. To determine whether newer FPGAs would allow processing at 60 FPS, we implemented the 25×25 filter for 4K frames on a Zynq Ultrascale+ ZCU102, successfully satisfying the 500-MHz constraint for 60 FPS, suggesting that the performance of this architecture scales well with newer FPGA devices.

F. Comparisons With Previous Work

We now evaluate the proposed architecture against other relevant work in the literature, with results summarized in Table X. We extrapolate the achieved FPS based on the assumption that these architectures offer a one pixel per cycle streaming processing flow where this is not reported.

Licciardo et al. [11] implement a multiplierless design that emulates the IEEE-754 floating point standard through the use of fixed point adders and additional logic to manage exponent alignment. Their proposed filter architecture is tailored to a fixed set of coefficients and a fixed range of input values using the Bachet weight decomposition theorem. Ortega-Cisneros et al. [30] present a 3×3 filter with fixed coefficients obtaining at best 318-MHz frequency. The work in [18] presents a

multiplierless, coefficient independent filter that also utilizes a mechanism for zero padding at the borders. They also compare against a baseline architecture that uses embedded multipliers and other work in the literature, with the most relevant to this article being [31]–[33]. Bailey and Ambikumar [10] explored border handling in transposed form filters. More specifically, they explored the additional cost of border management, the cost of different border extension methods in transposed form filters and the scaling of their proposed optimal border extension mechanism. The scaling exploration takes place on a morphological filter at seven filter sizes, ranging from 3×3 to 15×15, for a frame size of 1024×768. They also provide implementation results for the overlapped priming and flushing method presented in [26] on a direct form filter, a method used in this article for comparison in Section IV-C. In this particular case, the authors use a two-stage pipeline on the combination tree and further improvements can be obtained with a more heavily pipelined architecture. Meanwhile, our proposed designs, fully pipeline the adder tree. The work in [34] explores the performance and energy consumption of FPGAs, GPUs, and multicore processors for sliding window applications. The authors in this case implement three applications, sum of absolute differences (SAD), 2-D convolution, and correntropy, on all platforms. They explore how their architectures scale on a range of filter sizes, each for three frame sizes: 640×480, 1280×720, and 1920×1080. For the FPGA analysis, an Altera Stratix III E260 on a GiDEL ProcStar III board was used. Their 2-D convolution architectures use 16-bit fixed point or 32-bit floating point representations, obtaining operating frequencies of 104–115 and 103–114 MHz respectively. That work concludes that FPGAs are more power efficient, compared to GPUs and multicores, while providing significant performance improvement for large input sizes.

The implementation results of the filter architectures in this article demonstrate significant throughput improvement compared to previous work in the literature, while being flexible to adapt to varying coefficients dynamically. It is also worth noting that the architectures we have presented for comparison in this article, including the transposed form filter and border management enabled design, also demonstrate substantial improvements compared to previous work.

TABLE X: Comparisons with related work.

	Kernel Size	FPGA Platform	Freq. (MHz)	Resolution	FPS	Fixed Kernel	Notes
[10]	5×5	Cyclone V	175	1024×768	223		Transposed form, DSP-based, No border ext.
[10]	5×5	Cyclone V	152	1024×768	193		Direct form, Overlap prime and flush [26], Two-stage pipeline
[10]	5×5	Cyclone V	174	1024×768	221		Transposed form, Zero ext. using Transform Coalescing
[10]	5×5	Cyclone V	185	1024×768	235		Transposed form, Zero ext. using Combination Chain
[10]	5×5	Cyclone V	173	1024×768	219		Transposed form, Constant ext.
[10]	5×5	Cyclone V	159	1024×768	202		Transposed form, Duplication
[10]	5×5	Cyclone V	188	1024×768	239		Transposed form, Two-Phase duplication
[10]	5×5	Cyclone V	180	1024×768	229		Transposed form, Mirroring
[10]	5×5	Cyclone V	178	1024×768	226		Transposed form, Mirroring with duplication
[11]	3×3	XC7V	213	640×480 1920×1080 3840×2160	692 102 25	× × ×	Multiplierless, Emulates IEEE-754, Optimized for fixed set of coeffs. with fixed input range
[18]	7×7 11×11 22×22 30×30	V4LX160 V4LX160 V4LX160 V4LX160	175 181 177 171	1920×1080 1920×1080 1920×1080 1920×1080	84 87 85 82		Multiplierless, Zero-padding, Flexible Coefficients
[18]	7×7 11×11 22×22	V4LX160 V4LX160 V4LX160	183 142 149	1920×1080 1920×1080 1920×1080	88 68 72		Use of Multipliers
[30]	3×3	Stratix V	318	1024×720	410	×	Fixed Kernel
[31]	5×5	V5LX330	115	-	-	-	Neural Network related
[32]	7×7	V4SX35	200	-	-	-	Neural Network related
[33]	13×13	V4LX25	50	-	-	-	-
[34]	4×4–25×25	Stratix III	≤115	1920×1080	≤55		Fixed point
[34]	4×4–13×13	Stratix III	≤114	1920×1080	≤55		Floating point

V. Conclusion

This article presented a detailed discussion of 2-D spatial convolution filter design for FPGAs, proposing a scalable direct form architecture that can be extended to support border management, while offering high throughput through architectural optimizations driven by the underlying FPGA architecture, specifically the DSP blocks. Alternative adder tree designs were compared, alongside a comparison against transposed form implementations and an extended design with border management using the overlapped priming and flushing scheme. We demonstrated that the proposed architecture scales to a wide range of filter sizes, and frame sizes up to 4K, offering throughput of over 60 FPS in most cases, while further enhancements can be achieved on more recent FPGA devices. The proposed designs are optimized around the features of modern FPGA DSP blocks, used through explicit instantiation to achieve high throughput.

References

[1] D. G. Bailey, *Design for Embedded Image Processing on FPGAs*. Singapore : John Wiley & Sons (Asia), 2011.

[2] J. Gu, Z. Wang, J. Kuen, L. Ma, A. Shahroudy, B. Shuai, T. Liu, X. Wang, G. Wang, J. Cai, and T. Chen, "Recent advances in convolutional neural networks," *Pattern Recogn.*, vol. 77, no. C, pp. 354–377, May 2018.

[3] S. Asano, T. Maruyama, and Y. Yamaguchi, "Performance comparison of FPGA, GPU and CPU in image processing," in *Proceedings of International Conference on Field Programmable Logic and Applications*, 2009, pp. 126–131.

[4] D. Crookes, "Architectures for high performance image processing: The future," *Journal of Systems Architecture*, vol. 45, pp. 739–748, 1999.

[5] F. Schwiegelshohn, L. Gierke, and M. Hübner, "FPGA based traffic sign detection for automotive camera systems," in *International Symposium on Reconfigurable Communication-centric Systems-on-Chip (ReCoSoC)*, June 2015.

[6] S. Wang, C. Zhang, Y. Shu, and Y. Liu, "Live video analytics with FPGA-based smart cameras," in *Workshop on Hot Topics in Video Analytics and Intelligent Edges*, 2019, pp. 9–14.

[7] S. A. Fahmy, P. Y. K. Cheung, and W. Luk, "High-throughput one-dimensional median and weighted median filters on FPGA," in *IET Computers and Digital Techniques*, vol. 3, no. 4, 2009, pp. 384–394.

[8] A. Gabiger-Rose, M. Kube, R. Weigel, and R. Rose, "An FPGA-based fully synchronized design of a bilateral filter for real-time image denoising," *IEEE Transactions on Industrial Electronics*, vol. 61, no. 8, pp. 4093–4104, 2013.
[9] K. Vipin and S. A. Fahmy, "FPGA dynamic and partial reconfiguration: A survey of architectures, methods, and applications," *ACM Computing Surveys*, vol. 51, no. 4, p. 72, 2018.
[10] D. G. Bailey and A. S. Ambikumar, "Border handling for 2D transpose filter structures on an FPGA," *Journal of Imaging*, vol. 4, no. 12, 2018.
[11] G. D. 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, 2017.
[12] B. McDanel, S. Q. Zhang, H. T. Kung, and X. Dong, "Full-stack optimization for accelerating CNNs using powers-of-two weights with FPGA validation," in *ACM International Conference on Supercomputing*, 2019, pp. 449–460.
[13] J. Park, K. Muhammad, and K. Roy, "High-performance FIR filter design based on sharing multiplication," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 11, no. 2, pp. 244 –253, April 2003.
[14] P. Bougas, P. Kalivas, A. Tsirikos, and K. Pekmestzi, "Pipelined array-based FIR filter folding," *IEEE Transactions on Circuits and Systems I: Regular Papers,*, vol. 52, no. 1, pp. 108–118, Jan 2005.
[15] Z. Ma, Y. Yang, Y. Liu, and A. A. Bharath, "Recurrently decomposable 2-D convolvers for FPGA-based digital image processing," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 63, no. 10, pp. 979–983, Oct 2016.
[16] Z. Tang, J. Zhang, and H. Min, "A high-speed, programmable, CSD coefficient FIR filter," *IEEE Transactions on Consumer Electronics*, vol. 48, no. 4, pp. 834–837, nov 2002.
[17] S.-S. Jeng, H.-C. Lin, and S.-M. Chang, "FPGA implementation of FIR filter using m-bit parallel distributed arithmetic," in *IEEE International Symposium on Circuits and Systems*, 2006.
[18] F. J. Toledo-Moreo, J. J. Martínez-Alvarez, J. Garrigós-Guerrero, and J. M. Ferrández-Vicente, "FPGA-based architecture for the real-time computation of 2-D convolution with large kernel size," *Journal of Systems Architecture*, vol. 58, no. 8, pp. 277–285, 2012.
[19] S. Mirzaei, A. Hosangadi, and R. Kastner, "FPGA implementation of high speed FIR filters using add and shift method," in *International Conference on Computer Design*, 2006, pp. 308–313.
[20] H. Y. Cheah, F. Brosser, S. A. Fahmy, and D. L. Maskell, "The iDEA DSP block-based soft processor for FPGAs," *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, vol. 7, no. 3, pp. 19:1–19:23, 2014.
[21] A. K. Jain, D. L. Maskell, and S. A. Fahmy, "Throughput Oriented FPGA Overlays Using DSP Blocks," in *Proceedings of the Design, Automation and Test in Europe Conference (DATE)*, 2016.
[22] A. K. Jain, X. Li, P. Singhai, D. L. Maskell, and S. A. Fahmy, "DeCO: a DSP block based FPGA accelerator overlay with low overhead interconnect," in *IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2016, pp. 1–8.
[23] B. Ronak and S. A. Fahmy, "Mapping for maximum performance on FPGA DSP blocks," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 35, no. 4, pp. 573–585, 2016.
[24] B. Ronak and S. A. Fahmy, "Multipumping flexible DSP blocks for resource reduction on Xilinx FPGAs," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 36, no. 9, pp. 1471–1482, Sep. 2017.
[25] *HDL coding practices to accelerate design performance. Xilinx White Paper (WP231)*, 2006.
[26] D. Bailey, "Image border management for FPGA based filters," in *IEEE International Symposium on Electronic Design, Test and Application (DELTA)*, 2011, pp. 144 –149.
[27] A. Benkrid, K. Benkrid, and D. Crookes, "A novel FIR filter architecture for efficient signal boundary handling on Xilinx Virtex FPGAs," in *IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2003, pp. 273–275.
[28] C. Choo and P. Verma, "A real-time bit-serial rank filter implementation using Xilinx FPGA," in *Proc SPIE. 6811, Real-Time Image Processing*, 2008, p. 68110F.
[29] *Virtex-7 T and XT FPGAs Data Sheet: DC and AC Switching Characteristics (DS183)*, V1.28, March 2019.
[30] S. Ortega-Cisneros, M. A. Carrazco-Díaz, A. Pedroza de-la Crúz, J. J. Raygoza-Panduro, F. Sandoval-Ibarra, and J. Rivera-Domínguez, "Real time hardware accelerator for image filtering," in *Progress in Pattern Recognition, Image Analysis, Computer Vision, and Applications*. Springer International Publishing, 2014, pp. 80–87.
[31] M. Sankaradas, V. Jakkula, S. Cadambi, S. Chakradhar, I. Durdanovic, E. Cosatto, and H. P. Graf, "A massively parallel coprocessor for convolutional neural networks," in *Proceedings of the IEEE International Conference on Application-specific Systems, Architectures and Processors*, July 2009, pp. 53–60.
[32] C. Farabet, C. Poulet, J. Y. Han, and Y. LeCun, "CNP: An FPGA-based processor for convolutional networks," in *Proceedings of the International Conference on Field Programmable Logic and Applications*, Aug 2009, pp. 32–37.
[33] F. Fons, M. Fons, and E. Cantó, "Run-time self-reconfigurable 2D convolver for adaptive image processing," *Microelectronics Journal*, vol. 42, no. 1, pp. 204–217, 2011.
[34] J. Fowers, G. Brown, P. Cooke, and G. Stitt, "A performance and energy comparison of FPGAs, GPUs, and multicores for sliding-window applications," in *Proceedings of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, 2012, pp. 47–56.

Lenos Ioannou (Student Member, IEEE) received the B.Sc. degree in electrical engineering from the Cyprus University of Technology, Limassol, Cyprus, in 2014, and the M.Sc. degree in system on chip from the University of Southampton, Southampton, U.K., in 2015. He is currently working toward the Ph.D. degree at the Adaptive and Reconfigurable Computing Laboratory, University of Warwick, Coventry, U.K.

His research interests include embedded computing architectures and accelerators on reconfigurable platforms.

Abdullah Al-Dujaili (Member, IEEE) received the B.Eng. degree from Universiti Teknologi PETRONAS, Seri Iskandar, Malaysia, in 2012, and the Ph.D. degree from Nanyang Technological University, Singapore, in 2017.

He is a Research Scientist with the Algorithmic Systems Group, Analog Devices, Norwood, MA, USA. He was previously a Postdoctoral Associate with CSAIL, MIT, Cambridge, MA, USA, from 2017 to 2019, and a Data Scientist at Grab, Singapore, solving transportation challenges at the interface of optimization and machine learning techniques. He spent time as a Research Intern at institutes, such as the Upper Austria University of Applied Sciences, Wels, Austria, Aeste Works, Kuala Lumpur, Malaysia, IRISA Laboratory, France, the National University of Singapore, Singapore, and the Institute for Infocomm Research, Singapore

Dr. Al-Dujaili received the PETRONAS Carigali Country Award in 2008, the Singapore International Graduate Award (SINGA) in 2012, the 2nd Runner-Up Award at BBComp at GECCO'15, and the Best Poster Award at the IBM AI Research Week in Cambridge, in 2018.

Suhaib A. Fahmy (Senior Member, IEEE) received the M.Eng. degree in information systems engineering and the Ph.D. degree in electrical and electronic engineering from Imperial College London, London, U.K., in 2003 and 2007, respectively.

From 2007 to 2009, he was a Research Fellow with Trinity College Dublin, Dublin, Ireland, and a Visiting Research Engineer with Xilinx Research Labs, Dublin. From 2009 to 2015, he was an Assistant Professor with the School of Computer Engineering, Nanyang Technological University, Singapore. Since 2015, he has been an Associate Professor and now a Reader in Computer Engineering with the School of Engineering, University of Warwick, Coventry, U.K. His research interests include reconfigurable computing, high-level system design, and computational acceleration of complex algorithms.

Dr. Fahmy is a Senior Member of the ACM and a Chartered Engineer and member of the IET. He was a recipient of the Best Paper Award at the IEEE Conference on Field Programmable Technology in 2012, the IBM Faculty Award in 2013 and 2017, and the ACM TODAES Best Paper Award in 2019.