A Novel Separable Convolution Architecture for Image Processing Applications

Jaiza Hassan and Burhan Khurshid

***Abstract*—Two-dimensional (2D) Convolution is frequently used in many image processing applications like image smoothening, image sharpening, feature extraction, image enhancement, object recognition, etc. Although the operation of 2D Convolution is simple, its hardware implementation is quite challenging due to enormous computational and memory costs. Various 2D convolution implementations have been proposed in the literature. Among them Separable architectures stand out in terms of the reduction in the computational complexity. However, these have not been extensively explored in the literature, and more research needs to be done, especially for applications that are constrained in resources. This paper presents a novel separable Convolution architecture based on the folding transformation. The application of the folding transformation technique yields a resource-efficient architecture by time multiplexing the different functional units within the separable Convolution operation. The hardware implementation is done using Xilinx Artix-7 xc7a35tcpg236-1 FPGA device. The proposed architecture offers benefits over the existing architectures regarding on-chip resource utilization, power consumption, critical path delay, and external memory bandwidth (EMB).**

***Index Terms*—2D convolution, Folding transformation, FPGA, Partial buffering scheme, Separable filters.**

Original Research Paper
DOI: 10.53314/ELS2428012H

I. Introduction

In many image-processing applications, 2D Convolution is one of the basic operations that is frequently used. Discrete wavelet transform, object recognition, restoration, template matching, feature extraction, denoising, edge detection, enhancement, high-pass filtering (sharpening), low-pass filtering (blurring), and other image processing applications all use convolution operation [1-5]. The implementation of 2D convolution is complex, even with its straightforward form, since it incurs high computational costs and is also memory-expensive. Calculating a single output pixel using an $n \times n$ kernel necessitates n^2 image pixels per clock cycle, n^2 multiplications, and n^2-1 additions. It is evident that as the kernel size increases, so do these challenges.

Numerous architectural approaches have been suggested in the literature to overcome the aforementioned issues. These architectures primarily focus on optimizing the convolution kernel module and designing an efficient buffering scheme. Reducing on-chip resources based on a suitable EMB is achieved by using an effective buffering scheme. Full Buffering (FB) and Single Window Partial Buffering (SWPB) schemes are proposed in [6]. FB scheme stores entire *n-1* raster rows of an input image for a kernel of size $n \times n$ in row buffers, thereby achieving an EMB of one pixel/cycle but requiring a large amount of on-chip resources. SWPB scheme reduces resource utilization by storing only a small portion of the image at the cost of increased bandwidth. Multi-Window Partial Buffering (MWPB) scheme addresses these issues by reusing shared data between windows, minimizing external memory bandwidth, and reducing hardware utilization [7]. Nevertheless, this buffering strategy might not be feasible for larger kernel sizes if bandwidth is limited [8].

To optimize the Convolution unit, several approaches have been suggested in the literature. These include time-sharing [9-10], multiplier-less [5, 11-12], decomposition of the kernel into smaller parts [12-18], using structural properties of the kernel [19], pipelining [20], reconfigurable architectures [1, 21-23], and polyphase decomposition [14, 24]. Among these approaches, spatial separable convolution is one effective method for minimizing hardware utilization [12-13, 17-18, 21, 25-27], which transforms 2D convolution into the product of two one-dimensional (1D) convolutions. Separable convolution yields significant computational resource savings as the kernel size increases. Further, most 2D kernels are strictly separable; thus, convolution kernel separability is a considerable topic. However, limited study has been done on the architectural optimization of spatial separable convolutions.

Distributed arithmetic (DA) based separable Convolution FPGA architectures viz. Multi-Window Separable Convolution (MWSC) architecture and Single Window Separable Convolution (SWSC) architecture are presented in [12]. Compared to MWSC architecture, SWSC architecture offers better processing time and resource usage at the cost of throughput. In [5], two architectures viz. Separable Filter Architecture (SFA) and Generalized Filter Architecture (GFA) based on DA are proposed and compared. Non-pipelined separable Convolution architecture based on the MWPB scheme (NPSCA) is proposed in [25]. The architecture reuses the same multipliers and adders for horizontal and vertical convolution. In [27], resched-

Manuscript received on May 3rd, 2024. Received in revised form on July 3rd, 2024. Accepted for publication on July 4th, 2024.

Jaiza Hassan is a PhD student in the Department of Electronics and Communication Engineering, National Institute of Technology, Srinagar, India (e-mail: jaiza_2021phaece006@nitsri.ac.in).

Burhan Khurshid is an Assistant Professor in the Department of Electronics and Communication Engineering, National Institute of Technology, Srinagar, India (e-mail: burhan@nitsri.ac.in).

uled separable convolution architecture (RSCA) is proposed in which only one multiplier is used sequentially in the vertical convolution, reducing the overall multiplier complexity.

The abovementioned methods presented in the literature minimize resource utilization, which can be further reduced by sacrificing the throughput by a small amount. Further, with the increase in kernel size, the computational complexity and the critical path of the existing designs increase. The aim is, thus, to design a resource-efficient separable convolution architecture wherein the computational complexity and critical path are independent of kernel size. This is achieved by appropriately time-multiplexing the functional units within the separable convolution algorithm using the folding transformation [28, 29]. The folding technique is a systematic approach to designing control circuits in digital signal processing (DSP) architectures that reduce the number of functional units. Further, as mentioned earlier, many buffering schemes have been proposed to address the EMB issues. The PB scheme [6] is the most resource-efficient of the existing buffering schemes [7, 15, 25]. As designing the resource-efficient convolution architecture is the primary goal of the study, separable convolution architecture based on the PB scheme (SCA) [25] is used as a base architecture for folding transformation.

The rest of the paper is organized as follows. Section II illustrates separable convolution architecture, and Section III presents proposed area-efficient architecture. Section IV discusses hardware and timing complexities, and Section V discusses the performance analysis. Finally, the paper concludes the work in Section VI.

II. Separable Convolution

The concept behind separable convolution is that a 2D separable $n \times n$ kernel can be spatially divided into two 1D kernels: an $n \times 1$ vertical kernel v and a $1 \times n$ horizontal kernel h. Thus, just $2n\text{-}2$ additions and $2n$ multiplications are needed per pixel. The architecture of the separable convolution based on the PB scheme [25] is presented in Figure 1. For an $n \times n$ kernel, n input pixels are accessed simultaneously from the memory into n buffers. Based on whether vertical convolution is performed first or horizontal convolution, the n pixels belong to either $n \times 1$ or $1 \times n$ window. In this study, vertical convolution is performed first, followed by horizontal convolution; however, horizontal followed by vertical can be performed equivalently. The n pixels in the n buffers belonging to the $n \times 1$ window are multiplied by the corresponding vertical coefficients. The results are then added and stored in the registers. Once n intermediate results are available, they are multiplied by the n corresponding horizontal coefficients. The products are then added, and the final result is obtained. In the same manner, all output pixels are computed. The study aims to present the resource-efficient architecture obtained by applying the folding transformation [28] to the above-discussed architecture, which will be discussed in the next section.

III. Proposed Folded Separable Convolution Architecture

Figure 2 presents the detailed schematic of the proposed Folded Separable Convolution Architecture (FSCA). The proposed architecture is obtained by time-multiplexing $2n$ multipliers and $2n\text{-}2$ adders in the separable convolution architecture by two pipelined multipliers (M_V and M_H) and two pipelined adders (A_V and A_H). In other words, separable convolution architecture is folded by folding factor n for kernel of size $n \times n$. Multiplier M_V and adder A_V perform vertical convolution, and multiplier M_H and adder A_H perform horizontal convolution. Two pipelined stages are used for multiplication and one pipelined stage for addition. The decision is based on the latency configurations of the Xilinx IP adders and multipliers in Artix-7 FPGAs that implement these functional units. Instead of n input pixels, only one pixel located in the $n \times 1$ window is accessed from the external memory at a time. This reduces the EMB to *one pixel per clock cycle.*

At every clock cycle, the input pixel is accessed from memory, and the appropriate vertical coefficient is assigned to the multiplier M_V. The product outputs are stored in the registers. A total of $n\text{-}2$ registers are used between the multiplier M_V and ad-

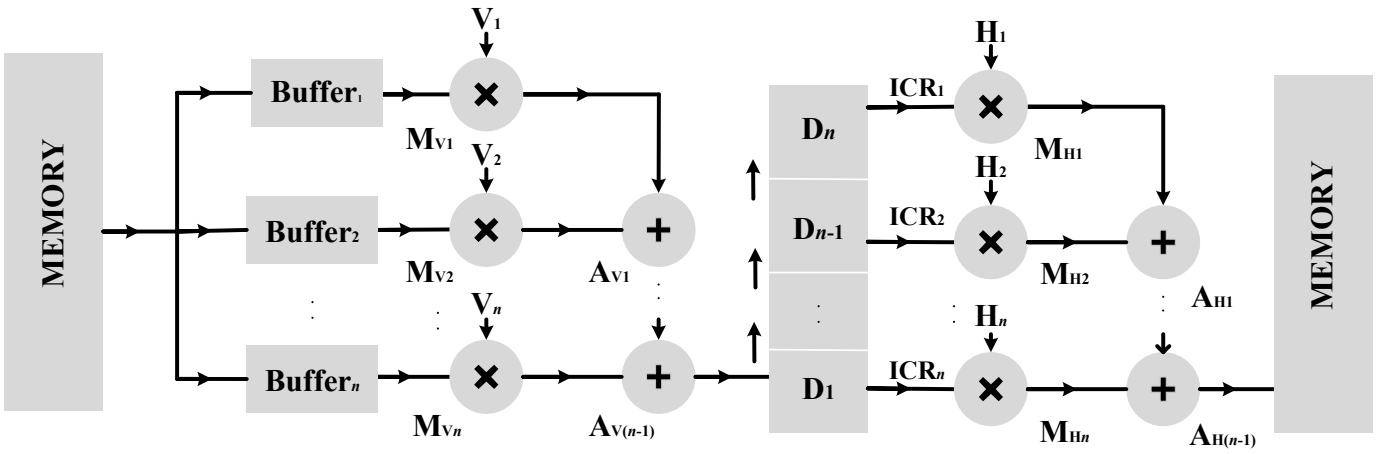

Fig. 1. Separable Convolution architecture based on PB scheme [25].

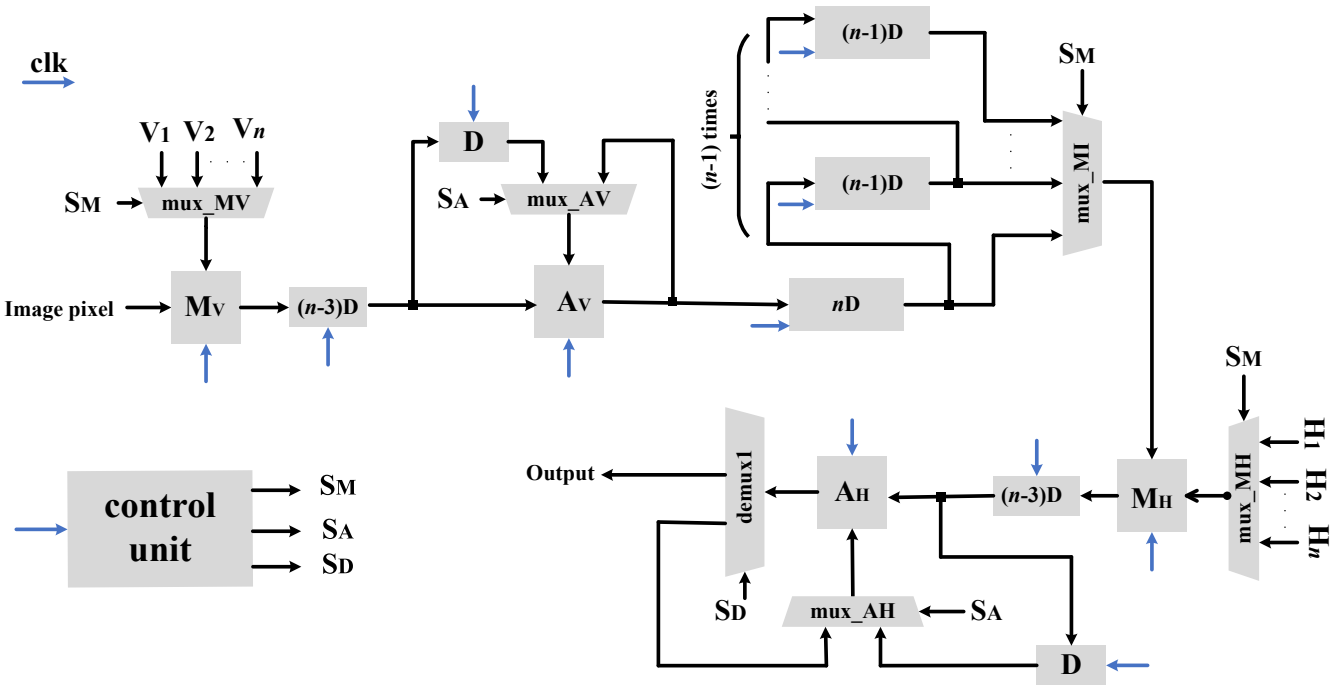

Fig. 2. Folded Separable Convolution architecture (FSCA).

der A_V. The product output is then added either to the previous product output or to the previous adder output using the adder A_V. After every n clock cycles, the intermediate result, i.e., vertical convolution output, is generated and stored in the registers. n shift registers (one shift register of size n and n-1 shift registers of size n-1) are used to store intermediate results. Among the n intermediate results from n shift registers and n horizontal coefficients, appropriate values are assigned to the multiplier M_H, and product results are stored in registers. Like vertical convolution, n-2 registers are used between the multiplier M_H and adder A_H. Adder A_H then adds product output either to the previous product output or to the previous adder output, like the adder A_V. The final output is produced after n clock cycles by the adder A_H through demultiplexer *demux1*. Each output pixel is produced similarly after every *n clock cycles.*

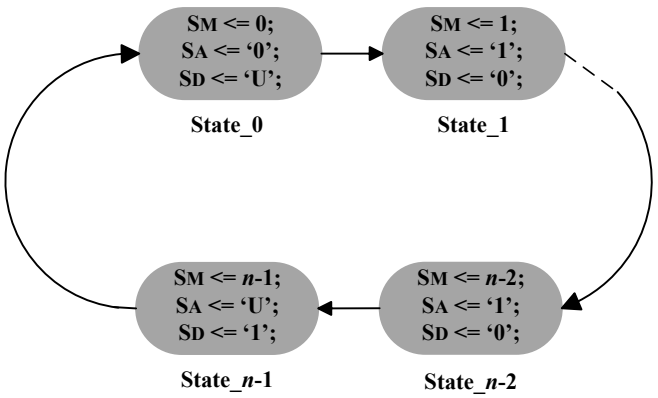

Fig. 3. Control Unit FSM.

Five multiplexers, *mux_MV*, *mux_MH*, *mux_MI*, *mux_AV*, *mux_AH,* and one demultiplexer, *demux1,* are required to perform the time multiplexing for the proposed architecture based on the control signals. Control signals, S_M, S_A, and S_D, are generated by the control unit, a modulo *n-counter* incremented for each clock cycle, where n is the size of the kernel. The control unit is designed by using the finite state machine (FSM), as shown in Figure 3. For kernel of size $n \times n$, the FSM consists of n states, *state_0* to *state_n-1*. The FSM resets to *state_0* after reaching *state_n-1*. The transition from one state to the next will occur at every clock cycle. The values on the selection lines are set according to the FSM's different states, which correspond to the different time instants. A timing diagram for the selection signals for a kernel of size 7×7 is shown in Figure 4.

Signal S_M is sent to the multiplexers, *mux_MV*, *mux_MH*, and *mux_MI,* signal S_A to multiplexers, *mux_AV,* and *mux_AH*, and signal S_D to demultiplexer *demux1*. The value of the signal S_M varies from 0 to *n-1,* incremented for each clock cycle (state). At each clock cycle based on the signal S_M, *mux_MH* and *mux_MV* decide on the n horizontal and vertical filter coefficients, respectively, while *mux_MI* decides on the n intermediate results from the vertical Convolution. Based on signal S_A, multiplexers, *mux_AV*, and *mux_AH*, select one of the inputs to adder A_V and A_H, respectively. For one clock cycle, S_A is low, and for *1* to *n-2* clock cycles, S_A is high, as described in Figure 4. When S_A is low, both inputs to the adders are products stored in registers. When S_A is high, one input to the adders is a product result, and the other is its output generated in the previous clock cycle. Module demultiplexer, *demux1,* sends data to the *mux_AH* when S_D is low and provides the final output when S_D is high. As shown in Figure 4, for *1* to *n-2* clock cycles, signal S_D is low, and at *n-1* clock cycle S_D is high. From Figure 3 and Figure 4, it can be seen that signal S_A and S_D are shown as uninitialized *'U'* (i.e., set to a "don't care" condition) at *state_n-1* and *state_0*, respectively. This is because adders, A_V and A_H, and

demultiplexer, *demux1* are not being used during these clock cycles, and it doesn't matter what value the selection line takes because the output of adders and demultiplexer is not being used at these particular instants of time.

IV. Hardware And Timing Complexity

Table I presents the hardware and timing complexities for the proposed FSCA, SCA [25], RSCA [27], and NPSCA [25] for the $n \times n$ kernel size. Hardware complexity is calculated in terms of the number of multipliers and adders and timing complexity in terms of critical path delay (CP). The proposed architecture requires only two adders and two multipliers, much less than existing architectures. Further, the number of multipliers and adders needed in the proposed design does not change with the kernel size. Thus, the number of multipliers and adders is significantly reduced compared to the existing architectures. Table II presents the resource estimation in terms of multipliers and adders for different kernel sizes for the proposed architecture, SCA [25], RSCA [27], and NPSCA [25], based on the hardware complexity presented in Table I. For a kernel size of 15×15, our proposed design requires 93.33%, 87.5%, and 86.66% fewer multipliers, and 92.85%, 92.85%, and 85.71% fewer adders when compared to SCA [25], RSCA [27], and NPSCA [25], respectively. From Figure 2, it is clear that the critical path in the proposed design will be the time required to complete the multiplication operation (T_m); however, in the proposed design, the multiplier is pipelined by two stages, thereby reducing the critical path to $T_m/2$ u.t. Compared to existing designs, the proposed design has a lesser critical path and is independent of kernel size. Even though our proposed design provides the expected performance results, the actual implementation is necessary to verify these results, as detailed in the next section.

V. Implementation Results

The proposed design is modeled using VHDL language and implemented on Xilinx Artix-7 xc7a35tcpg236-1 FPGA. All hardware modules are synthesized using Xilinx VIVADO 2023.1, and performance metrics are analyzed. Performance is

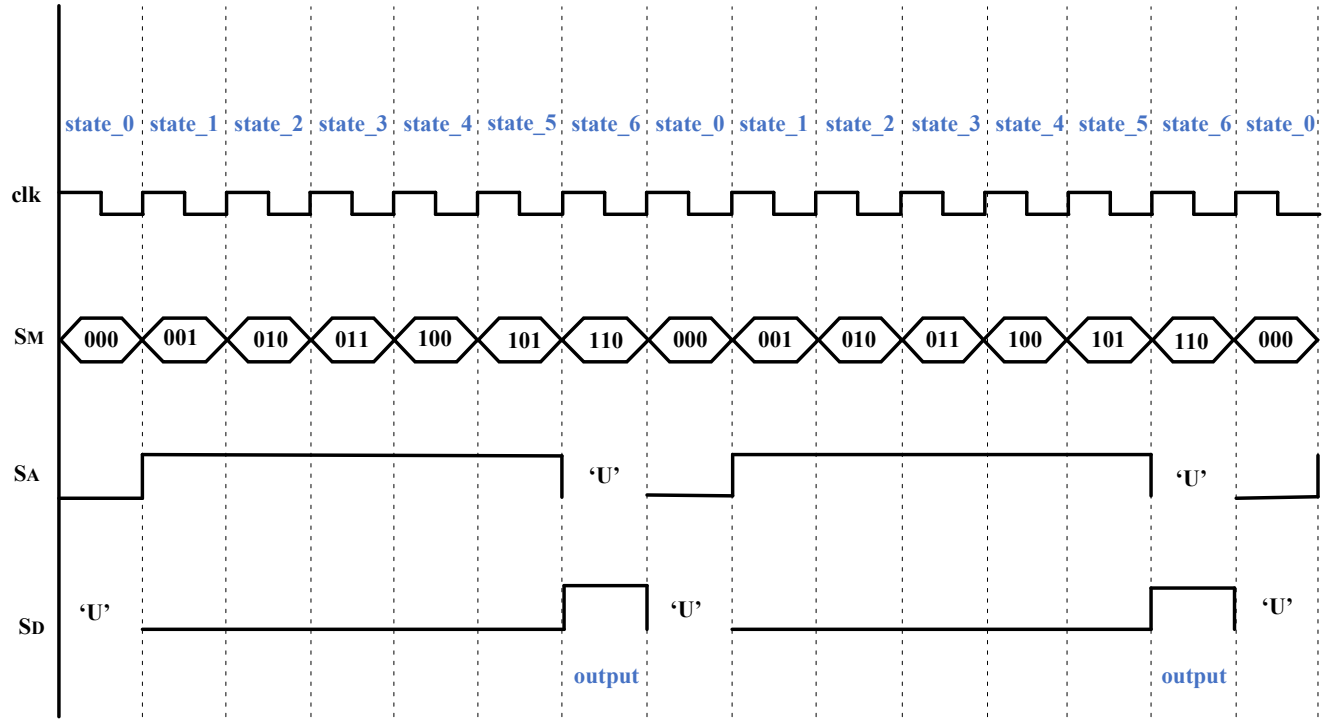

Fig. 4. Timing diagram for a kernel of size 7 x 7.

Table I
Hardware and timing complexity for proposed and existing separable convolution architectures for N×N kernel size.

Method	Multipliers	Adders	Critical Path
SCA [25]	*2n*	*2n-2*	$T_m + (n-1)\,T_a$
RSCA [27]	*n+1*	*2n-2*	$T_m + (n-1)\,T_a$
NPSCA [25]	*n*	*n-1*	$T_m + (n-1)\,T_a$
Proposed FSCA	*2*	*2*	$T_m/2$

Tm: Computation time for the multiplication operation.
Ta: Computation time for the addition operation.

TABLE II
NUMBER OF ADDERS AND MULTIPLIERS REQUIRED IN PROPOSED AND EXISTING SEPARABLE CONVOLUTION ARCHITECTURES.

Filter size	SCA [25]		RSCA [27]		NPSCA [25]		Proposed FSCA	
	Multiplier	Adder	Multiplier	Adder	Multiplier	Adder	Multiplier	Adder
3 x 3	6	4	4	4	3	2	2	2
5 x 5	10	8	6	8	5	4	2	2
7 x 7	14	12	8	12	7	6	2	2
9 x 9	18	16	10	16	9	8	2	2
11 x 11	22	20	12	20	11	10	2	2
13 x 13	26	24	14	24	13	12	2	2
15 x 15	30	28	16	28	15	14	2	2

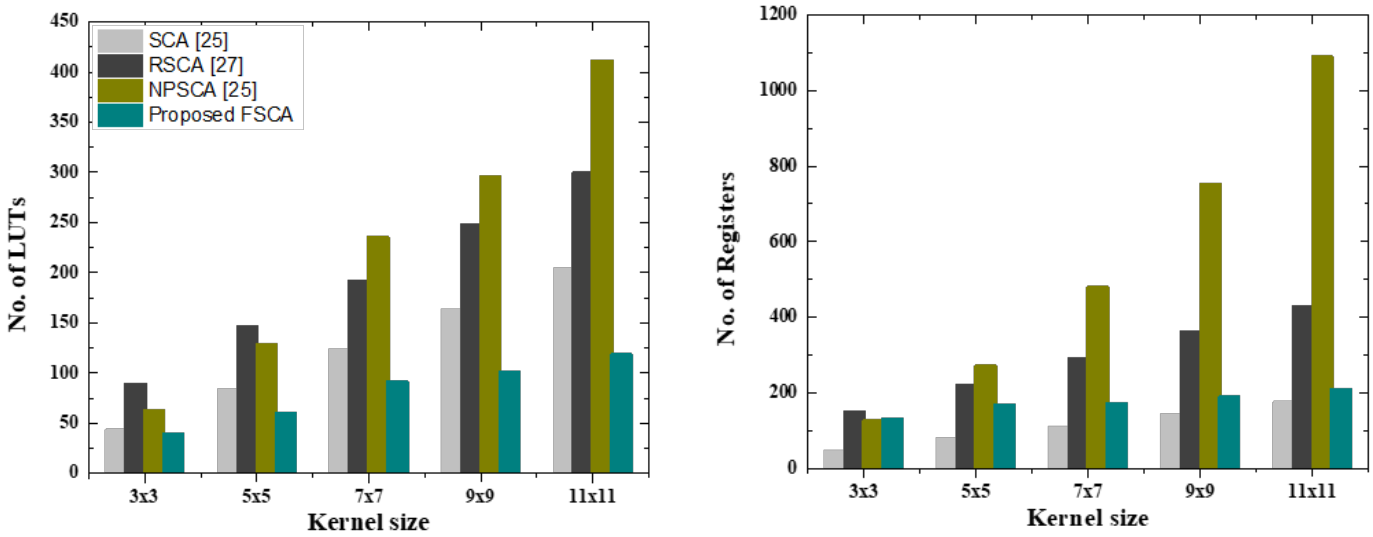

Fig. 5. LUT and register utilization for different architectures for an image size of 128×128 and varied kernel sizes.

compared against SCA [25], RSCA [27], and NPSCA [25] presented in the literature. All the architectures are implemented within the same design environment and using the same set of constraints.

Figure 5 presents the LUT and register utilization for 128 x 128 image and various kernel sizes for different architectures. It can be observed that both the LUT and register utilization in all the designs increase as the kernel size increases. However, the increase in LUT utilization with the kernel size in the proposed design is minimal compared to the other designs. This increase is due to the increase in the number of states in the FSM used to design the control unit and the size of multiplexers as the kernel size increases. For a given kernel size, the proposed design utilizes fewer LUTs than the other designs. For instance, for the kernel size of 11×11, the LUT utilization for the proposed design is 41.66%, 60.33%, and 71.11% lesser when compared to SCA [25], RSCA [27], and NPSCA [25], respectively. Register utilization for the proposed design is comparatively higher than that of the SCA. It is evident that since the proposed design is obtained by time-multiplexing the functional units it requires more registers at different time instants to store the intermediate results. However, an increase in the registers to reduce the functional units, especially the multipliers, is advantageous as multipliers are the main computational burden in the design. Furthermore, register utilization does not result in additional hardware costs since each FPGA logic slice has a large number of registers that will otherwise remain unutilized. For a kernel of size 3×3 and 11×11, our proposed design utilizes 179.16% and 19.3% more registers than SCA [25]. Therefore, as the kernel size increases, the percentage increase in the register utilization decreases. Further, compared to RSCA [27] and NPSCA [25], register utilization for the proposed design is lesser. For a kernel size of 11×11, register utilization for the proposed design is 51.27% and 80.75% less than that of RSCA [27] and NPSCA [25].

Figure 6 provides the timing results regarding the minimum attainable clock period (MCP), maximum operating frequency (F), and critical path delay (CPD) for various kernel sizes. As expected, the proposed design, when compared to SCA [25], RSCA [27], and NPSCA [25], has a lesser minimum attainable clock period and critical path. A lower minimum attainable clock period and critical path result from the reduced number of functional units and the usage of pipelined units in the proposed architecture. It can be pointed out that a gradual increase in the minimum attainable clock period and the critical path occurs with the increase in kernel size. This is due to the increased complexity of the design as the kernel size increases. Nevertheless, as anticipated, in SCA [25], RSCA [27], and NPSCA [25], as kernel size increases, a prominent increase in the minimum attainable clock period and critical path is reported. For a kernel

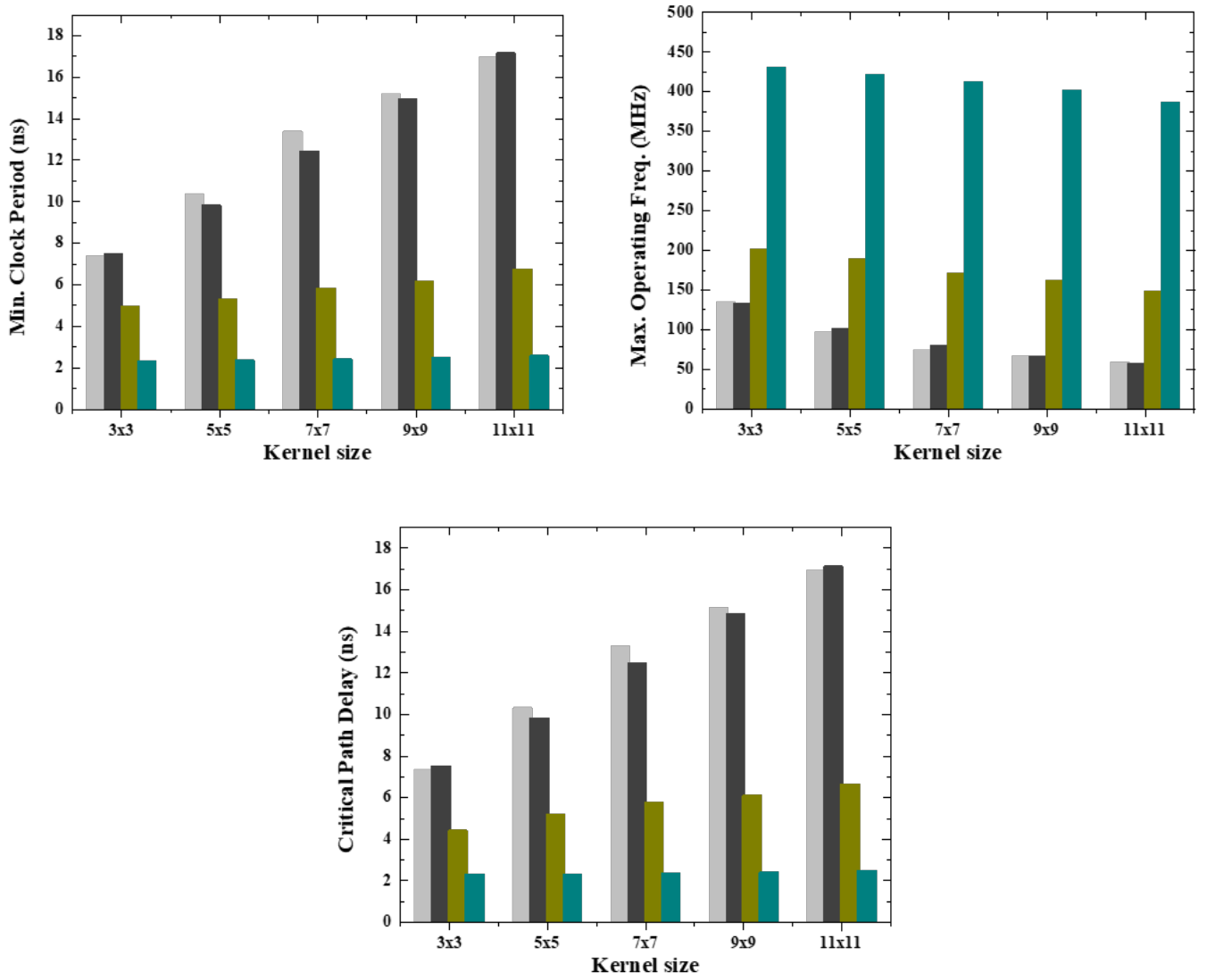

Fig. 6. Timing analysis for different architectures for an image size of 128×128 and varied kernel sizes.

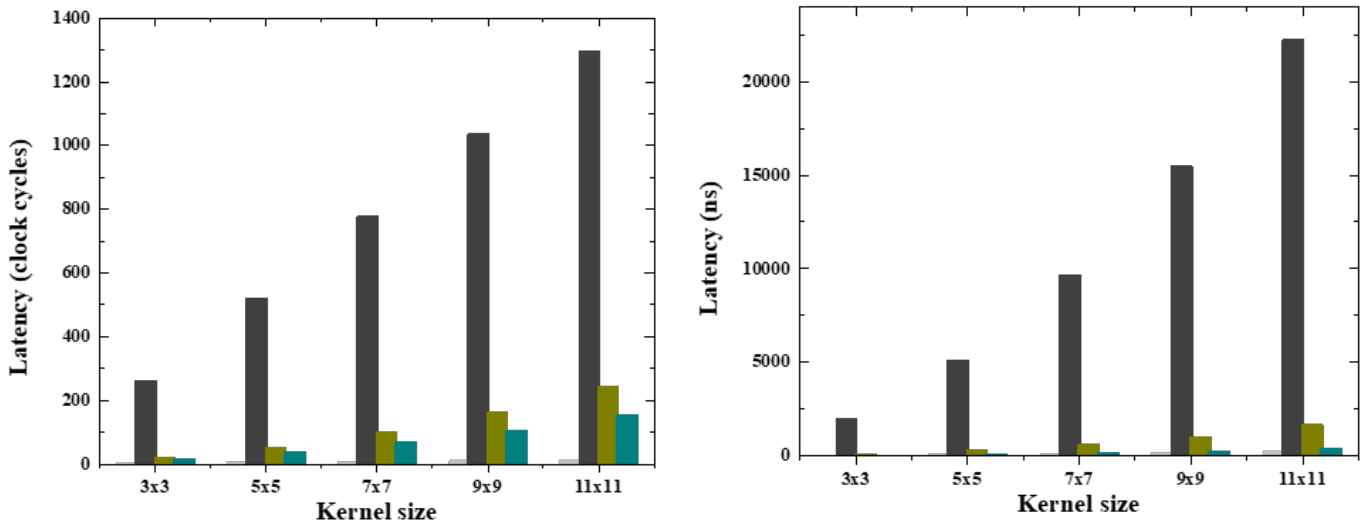

Fig. 7. Latency for different architectures for an image size of 128×128 and varied kernel sizes.

size of 11×11, the minimum achievable clock period and critical path for the proposed design is 84.72%, 84.92%, 61.63%, and 85.17%, 85.34%, 62.37% lesser when compared to SCA [25], RSCA [27], and NPSCA [25], respectively. Additionally, Figure 7 compares the latency of different architectures. Latency is calculated in terms of the clock cycles required to process one output pixel. As the proposed architecture is based on folding transformation, latency will increase as compared to the SCA [25]. This is obvious because the EMB of SCA [25] is *n* times higher than the proposed design. However, higher EMB leads to higher memory requirements and higher power dissipation, limiting the usability of SCA [25] for large kernel sizes. When compared to RSCA [27] and NPSCA [25], the latency of the proposed design is much less. Further, as mentioned earlier, the clock period of the proposed design is less than that of the other designs. Thus, for a fair comparison, latency is also calculated in *ns* by multiplying the MCP by the number of clock cycles. For a kernel size of 11×11, latency for the proposed design is 98.22% and 75.84% less than that of RSCA [27] and NPSCA [25] and 94.70% higher than that of SCA [25].

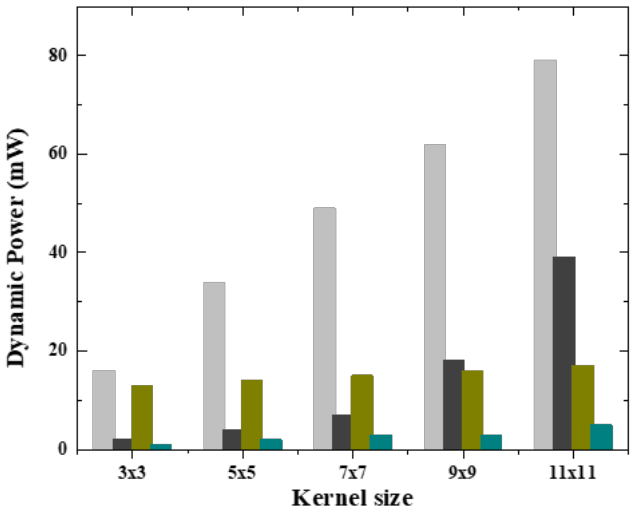

Fig. 8. Power analysis in mW for different architectures for an image size of 128×128 and varied kernel sizes.

Figure 8 presents the power analysis for the different architectures. The proposed design reports a lesser dynamic power dissipation when compared to SCA [25], RSCA [27], and NPSCA [25]. The reduction in power dissipation is expected due to the decreased functional units and the switching activity in the proposed design. Further, the large power dissipation in SCA [25] is also due to multiple pixel access, and in RSCA [27], it is due to the large buffer lines. For a kernel of size 11×11, the power dissipation in the proposed design is 93.67%, 87.17%, and 70.58% less when compared to SCA [25], RSCA [27], and NPSCA [25], respectively.

VI. Conclusion

In this study, the folded separable convolution architecture is proposed that requires only two adders and two multipliers, irrespective of the kernel size. Thus, for a kernel of size $n \times n$, the computational complexity is approximately reduced by factor *n*. In addition to the functional units, since only one pixel is accessed per clock cycle, the EMB of the proposed design is *one pixel/clock cycle*. Experimental results demonstrated that the proposed architecture utilizes fewer LUT slices and has reduced critical path and power dissipation compared to existing designs. The future discourse of the research will focus on designing time-multiplexed architectures for the non-separable convolution architecture.

References

[1] S. Perri, M. Lanuzza, P. Corsonello, G. Cocorullo, "A high-performance fully reconfigurable FPGA-based 2D convolution processor," *Microprocess. Microsy.*, vol. 29, no. 8-9, pp. 381–391, Nov. 2005.

[2] R. C. Gonzalez, "Digital image processing", *Pearson education India*, 2009.

[3] S. Sadangi, S. Baraha, D. K. Satpathy, P. K. Biswal, "FPGA implementation of spatial filtering techniques for 2D images," in *2017 2nd IEEE International Conference on Recent Trends in Electronics, Information & Communication Technology (RTEICT)*, Bangalore, India, 2017, pp. 1213–1217.

[4] J. Lee, H. Tang, J. Park, "Energy efficient canny edge detector for advanced mobile vision applications," *IEEE T. Circ. Syst. Vid.*, vol. 28, no. 4, pp. 1037–1046, Dec. 2018.

[5] D. Mukherjee, S. Mukhopadhyay, "Fast hardware architecture for fixed-point 2D Gaussian filter," *AEU-Int. J. Electron. C.*, vol. 105, pp. 98–105, Jun. 2019.

[6] B. Bosi, G. Bois, Y. Savaria, "Reconfigurable pipelined 2-D convolvers for fast digital signal processing," *IEEE T. VLSI Syst.*, vol. 7, no. 3, pp. 299–308, Sep. 1999.

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

[8] M. Kazmi, A. Aziz, H. R. Khan, S. A. Qazi, and L. K. Stergioulas, "Resource-Efficient Image Buffer Architecture for Neighborhood Processors," *IEEE Access*, vol. 8, pp. 181 964–181 975, Sep. 2020.

[9] V. Sriram, D. Kearney, "A FPGA implementation of variable kernel convolution," in *Eighth International Conference on Parallel and Distributed Computing, Applications and Technologies (PDCAT 2007)*, Adelaide, SA, Australia, 2007, pp. 105–110.

[10] M. Kalbasi, H. Nikmehr, "A classified and comparative study of 2-D convolvers," in *2020 International Conference on Machine Vision and Image Processing (MVIP)*, Iran, 2020, pp. 1–5.

[11] G. D. Licciardo, C. Cappetta, L. Di Benedetto, A. Rubino, R. Liguori, "Multiplier-less stream processor for 2D filtering in visual search applications," *IEEE T. Circ. Syst. Vid.*, vol. 28, no. 1, pp. 267–272, Aug. 2018.

[12] D. Mukherjee, S. Mukhopadhyay, "Fast hardware architecture for 2-D separable convolution operations," *IEEE T. Circuits-II*, vol. 65, no. 12, pp. 2042–2046, Mar. 2018.

[13] F. Mamalet, C. Garcia, "Simplifying convnets for fast learning," in *International Conference on Artificial Neural Networks*, Berlin, Heidelberg, 2012, pp. 58–65.

[14] J. Wang, J. Lin, Z. Wang, "Efficient convolution architectures for convolutional neural network," in *2016 8th International Conference on Wireless Communications & Signal Processing (WCSP)*, Yangzhou, China, 2016, pp. 1–5.

[15] Z.-B. Ma, Y. Yang, Y.-X. Liu, A. A. Bharath, "Recurrently decomposable 2-D convolvers for FPGA-based digital image processing," *IEEE T. Circuits-II*, vol. 63, no. 10, pp. 979–983, Feb. 2016.

[16] J. Chang, J. Sha, "An efficient implementation of 2D convolution in CNN," *IEICE Electron. Expr.*, vol. 14, no. 1, pp. 20161134–20161134, 2017.

[17] S. I. Cho, S.-J. Kang, "Gradient prior-aided CNN denoiser with separable convolution-based optimization of feature dimension," *IEEE T. Multimedia*, vol. 21, no. 2, pp. 484–493, Jul. 2019.

[18] M. Kr´alik, L. Lad´anyi, “Canny edge detector algorithm optimization using 2D spatial separable convolution,” *Acta Electrotech. Inform.*, vol. 21, no. 4, pp. 36–43, 2021.

[19] J. Y. Mori, C. H. Llanos, P. A. Berger, “Kernel analysis for architecture design trade off in convolution-based image filtering,” in *2012 25th Symposium on Integrated Circuits and Systems Design (SBCCI)*, Brasilia, Brazil, 2012, pp. 1–6.

[20] M. Kalbasi and H. Nikmehr, “A Fine-Grained Pipelined 2-D Convolver for High-Performance Applications,” *IEEE T. Circuits-II*, vol. 66, no. 1, pp. 146–150, May 2018.

[21] L. Rao, B. Zhang, J. Zhao, “Hardware implementation of reconfigurable 1D convolution,” *J. Signal Process.*, vol. 82, pp. 1–16, Jan. 2016.

[22] A. Dehghani, A. Kavari, M. Kalbasi, K. RahimiZadeh, “A new approach for design of an efficient FPGA-based reconfigurable convolver for image processing,” *J Supercomput.*, vol. 78, pp. 2597–2615, Feb. 2022.

[23] L. Rao, B. Zhang, J. Zhao, “Hardware implementation of reconfigurable separable convolution,” in *2018 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, Hong Kong, China, 2018, pp. 232–237.

[24] A. Arumalla and M. L. Makkena, “Efficient Architecture for Block Parallel Convolution using Two-Dimensional Polyphase Decomposition,” *Circ. Syst. Signal. Pr.*, vol. 41, no. 2, pp. 1166–1186, Feb. 2022.

[25] A. K. Joginipelly, D. Charalampidis, “Efficient separable convolution using field programmable gate arrays,” *Microprocess. Microsy.*, vol. 71, pp. 102852, Nov. 2019.

[26] Y. Hu, V. K. Prasanna, “Energy-efficient parameterized 2-D separable convolution on FPGA,” in *International Green Computing Conference*, Dallas, TX, USA, 2014, pp. 1–10.

[27] H. Kim, H. Yoo, “Area-efficient two-dimensional separable convolution structure,” *J. Imaging. Sci. Techn.*, vol. 63, no. 5, pp. 50404–1, Sep. 2019.

[28] K. K. Parhi, C.-Y. Wang, A. P. Brown, “Synthesis of control circuits in folded pipelined DSP architectures,” *IEEE J. Solid-St. Circ.*, vol. 27, no. 1, pp. 29–43, Jan. 1992.

[29] K. K. Parhi, “VLSI digital signal processing systems: design and implementation,” *John Wiley & Sons*, 2007.