A High-Performance Accelerator for Real-Time Super-Resolution on Edge FPGAs

HONGDUO LIU, The Chinese University of Hong Kong, Hong Kong, China
YIJIAN QIAN, SmartMore, Shenzhen, China
YOUQIANG LIANG, SmartMore, Shenzhen, China
BIN ZHANG, SmartMore, Shenzhen, China
ZHAOHAN LIU, SmartMore, Shenzhen, China
TAO HE, SmartMore, Shenzhen, China
WENQIAN ZHAO, The Chinese University of Hong Kong, Hong Kong, China
JIANGBO LU, SmartMore, Shenzhen, China
BEI YU, The Chinese University of Hong Kong, Hong Kong, China

In the digital era, the prevalence of low-quality images contrasts with the widespread use of high-definition displays, primarily due to low-resolution cameras and compression technologies. Image super-resolution (SR) techniques, particularly those leveraging deep learning, aim to enhance these images for high-definition presentation. However, real-time execution of deep neural network (DNN)-based SR methods at the edge poses challenges due to their high computational and storage requirements. To address this, field-programmable gate arrays (FPGAs) have emerged as a promising platform, offering flexibility, programmability, and adaptability to evolving models. Previous FPGA-based SR solutions have focused on reducing computational and memory costs through aggressive simplification techniques, often sacrificing the quality of the reconstructed images. This paper introduces a novel SR network specifically designed for edge applications, which maintains reconstruction performance while managing computation costs effectively. Additionally, we propose an architectural design that enables the real-time and end-to-end inference of the proposed SR network on embedded FPGAs. Our key contributions include a tailored SR algorithm optimized for embedded FPGAs, a DSP-enhanced design that achieves a significant four-fold speedup, a novel scalable cache strategy for handling large feature maps, optimization of DSP cascade consumption, and a constraint optimization approach for resource allocation. Experimental results demonstrate that our FPGA-specific accelerator surpasses existing solutions, delivering superior throughput, energy efficiency, and image quality.

CCS Concepts: • **Computer systems organization → Embedded hardware;**

Additional Key Words and Phrases: FPGA, Super-Resolution, Real-time System

This work is supported in part by Shenzhen Science and Technology Program (No. KQTD20210811090149095).
Authors' addresses: H. Liu and B. Yu (Corresponding author), Rm 905, Ho Sin Hang Engineering Building, The Chinese University of Hong Kong, Shatin, Hong Kong SAR; e-mails: hdliu21@cse.cuhk.edu.hk, byu@cse.cuhk.edu.hk; W. Zhao, Rm 122, Ho Sin Hang Engineering Building, The Chinese University of Hong Kong, Shatin, Hong Kong SAR; e-mail: wqzhao@cse.cuhk.edu.hk; Y. Qian, Y. Liang, B. Zhang, Z. Liu, T. He, and J. Lu (Corresponding author), Rm 2201, Tower 2, Qianhai Kerry Center, Nanshan District, Shenzhen, Guangdong Province, China; e-mails: yijian.qian@smartmore.com, youqiang.liang@smartmore.com, bin.zhang@smartmore.com, zhaohan.liu@smartmore.com, tao.he@smartmore.com, jiangbo@smartmore.com.

ACM 1084-4309/2024/05-ART53
https://doi.org/10.1145/3652855

ACM Reference Format:
Hongduo Liu, Yijian Qian, Youqiang Liang, Bin Zhang, Zhaohan Liu, Tao He, Wenqian Zhao, Jiangbo Lu, and Bei Yu. 2024. A High-Performance Accelerator for Real-Time Super-Resolution on Edge FPGAs. *ACM Trans. Des. Autom. Electron. Syst.* 29, 3, Article 53 (May 2024), 25 pages. https://doi.org/10.1145/3652855

1 INTRODUCTION

Since the onset of the digital era, high-definition display panels have become increasingly prevalent in our daily lives, encompassing TV sets and PC monitors. Unfortunately, the usage of low-resolution cameras and compression technologies still results in a large amount of poor-quality images. To address this issue, image **super-resolution (SR)** techniques have been developed to enhance **low-resolution (LR)** images and make them better suited for high-definition displays. In recent years, the emergence of deep learning has further propelled the advancements in SR, with **Deep Neural Network (DNN)**-based methods [1–5] achieving superior reconstruction performance on various SR benchmarks. However, these methods often entail high computational and storage complexity. While real-time SR can be readily achieved in data centers equipped with high-performance cloud GPUs, performing SR at the edge presents significant challenges due to constrained hardware resources and power limitations.

FPGA-based solutions have emerged as an appealing choice when considering the hardware implementation of DNN-based SR algorithms at the edge. FPGAs provide enhanced flexibility in terms of data flow and data types compared to GPUs. Additionally, FPGAs offer greater programmability and shorter development time than ASICs, making them well-suited for handling rapidly evolving models. Previous FPGA-based SR accelerators have employed various techniques to bypass the huge computational and memory costs of DNN-based SR algorithms. He et al. [6] proposed a method that involved partitioning each input frame into blocks and dispatching them to either a neural network or an interpolation module for scaling, depending on the block's variation values. However, since some blocks were only processed using interpolation, the resulting **high-resolution (HR)** images exhibited lower quality than those fully processed by the SR network. Kim et al. [7] introduced a novel SR method that performs LR input frame processing line by line using 1D convolution. While this approach significantly reduced the memory footprint, the limited receptive field of 1D convolution also constrained the overall SR performance. Chang et al. [8] implemented a more lightweight version of FSRCNN on FPGA by transforming the deconvolutional layer into a convolutional layer. However, due to its compressed nature compared to the original FSRCNN, the SR performance suffered a degradation.

In this paper, we have enhanced the reconstruction quality of FPGA-based SR accelerators from two fundamental aspects. First, we introduce a novel SR network capable of generating higher-quality HR images compared to previous lightweight SR networks, maintaining analogous computation costs. Second, rather than adopting aggressive simplification techniques that can severely impact reconstruction quality, we propose a novel architectural design that efficiently handles the immense computation and memory requirements of DNN-based SR algorithms. By tackling these challenges head-on, our work enables high-quality SR without compromising the integrity of the reconstructed images. The contributions of this work are listed as follows.

— We propose a new SR algorithm considering the constraints of embedded FPGAs. It can achieve better reconstruction performance than previous lightweight SR models with similar computation costs.
— We propose a DSP usage to enable 4× speedup compared with traditional design and a novel cache strategy to tackle the extremely large feature maps in DNN-based SR models. The cache strategy is scalable to different input sizes and can effectively handle stride 1 and

stride 2 convolution. Our cache strategy can consistently transfer data to the convolution engine without stalling the computation.
— For the first time, we explore the cascade of DSPs to optimize the consumption of FFs and LUTs. We construct a tree structure to capture the behavior of connected DSP blocks and build an analytical model to inspect the resource utilization of the most basic computation units based on the tree structure. We also propose an efficient search algorithm to discover Pareto optimal designs in an exponentially increasing search space.
— Given the well-defined dataflow, various hardware components' latency and resource requirements can be clearly inferred using the design parameters. Then, we formulate the resource allocation issue as a constraint optimization problem, considering DSP and bandwidth constraints. By solving the optimization problem, we can obtain an optimized hardware configuration to maximize throughput.
— Our accelerator, designed specifically for embedded FPGAs, outperforms commercial GPU, FPGA, and AI chip solutions in terms of both throughput and energy efficiency. Additionally, it surpasses existing SR accelerators on FPGAs by delivering higher-quality HR images.

The remainder of this paper is organized as follows. Section 2 provides some preliminaries regarding SR and the design motivations of our accelerator. Section 3 details the architecture of our SR accelerator. In Section 4, we illustrate the design of building blocks for computation engines. Section 5 presents the resource allocation scheme of our accelerator to maximize throughput. Section 6 gives the evaluation results and comparisons with other **state-of-the-art (SOTA)** implementations. Section 7 concludes the paper and illustrates our future work.

2 BACKGROUND

In this section, we begin by introducing the concept of super-resolution and highlighting its wide range of applications. We then proceed to present our newly proposed super-resolution network, providing an overview of its key components and architecture. The network has been meticulously designed, taking into account both super-resolution quality and hardware resources. Next, we delve into the intricate design challenges that arise when implementing the network end-to-end on an embedded FPGA. We discuss the complexities involved and the considerations that need to be taken into account. Lastly, we summarize the fundamental principles that guide the design of our accelerator. These principles encompass the computation scheme, data communication, and dataflow of our proposed SR network. Through hardware-software co-optimization, we ensure an optimal balance between performance and efficiency.

2.1 Super-Resolution and Its Applications

Image super-resolution (SR) is a fundamental task in computer vision that aims to reconstruct high-resolution (HR) images from their corresponding low-resolution (LR) counterparts. It finds extensive applications in various domains, including satellite imaging [9], surveillance [10, 11], and medical imaging [12–14]. While SR can be performed on devices with rich computing resources, such as desktops or PCs, there is also a growing need for SR at the edge, closer to end users. One such scenario involves embedding super-resolution engines into video players to enhance the quality of low-resolution archives, like old movies and self-filmed videos. Additionally, embedded SR accelerators have the potential to enhance the **Quality of Experience (QoE)** for online video subscribers. SR has been widely adopted in internet video streaming systems, with approaches like NAS [15] and SRAVS [16] integrating SR engines at the client side to improve video quality, particularly in poor network conditions. This paper primarily focuses on facilitating edge-based SR.

2.2 Adaptive Filter-Based Super-Resolution

Super-resolution has been a prominent area of research in computer vision and signal processing for several decades. While traditional interpolation-based techniques such as bicubic interpolation are simple and efficient, they struggle to restore fine details in images. In recent years, deep learning approaches have gained traction by leveraging the statistical relationships between LR and HR images. One notable deep learning model, SRCNN [17], employs a deep convolutional neural network to learn an end-to-end mapping between bicubic interpolated images and HR images. Another model, FSRCNN [18], introduces a deconvolution layer as the final layer, enabling direct upsampling from the original low-resolution image. To further address the computational and memory overhead of deconvolution, ESPCN introduced a sub-pixel convolution technique, specifically pixelshuffle, as the primary upsampling method. ESPCN utilizes convolutional layers to extract features and then applies sub-pixel convolution to increase spatial resolution. Additionally, VSDR [19] demonstrates that increasing network depth significantly improves SR quality. Furthermore, SRGAN [20] treats SR as an image generation problem and leverages adversarial and content loss within a generative adversarial network to produce highly realistic images.

Although deep learning-based SR achieves superior performance than conventional algorithms. Stacking deeper layers often demands substantial computational resources, making them impractical for embedded applications. To address this, adaptive filter-based methods, as suggested by [21, 22], offer a solution. These methods reduce computational demands while still providing high-quality SR results. They involve three main stages: interpolation, filter generation, and filtering. In the interpolation phase, bilinear or bicubic interpolation techniques are employed to obtain up-sampled images. The filter generation phase focuses on learning the relative importance of neighboring elements. Finally, the filtering stage enhances the quality of cost-effective up-sampled images.

2.3 Network Architecture of Proposed SR Network

As adaptive filter-based methods demonstrate a balance between computation demands and super-resolution performance. Our proposed super-resolution network builds upon the foundations of LAPAR [24], one of the leading methods in this field. Despite LAPAR's impressive state-of-the-art performance, even its most lightweight version (LAPAR-C) falls short of achieving real-time performance (> 25fps) on an embedded FPGA. To address this limitation, we have simplified LAPAR to make it highly suitable for embedded applications. Initially, our approach involves selectively retaining essential operators such as convolution, pixelshuffle, and **Einstein summation (einsum)**. This strategy helps us to minimize design efforts and enables maximal hardware component reuse, thereby enhancing efficiency. Secondly, we strategically omit certain layers from our model to significantly reduce computational demand. This step is critical in optimizing performance without compromising the core functionality of the network. Furthermore, we streamline the entire network architecture by eliminating some residual connections. This simplification not only reduces complexity but also aids in unifying the data flow within the accelerator. These modifications collectively ensure a more efficient and cohesive operation of our super-resolution network, striking a balance between performance and hardware resource utilization. In Figure 1, we present performance comparisons of various DNN-based SR models. LAPAR-A, LAPAR-B, and LAPAR-C correspond to different versions of LAPAR with varying network sizes and complexities, as proposed in the original paper. Additionally, we introduce "Ours," which represents our newly developed SR network based on LAPAR. The results demonstrate that LAPAR networks achieve superior SR quality with less computational cost. Notably, our SR network produces higher quality HR images with similar computation overhead compared to FSRCNN [18] and MoreMNAS-C [25].

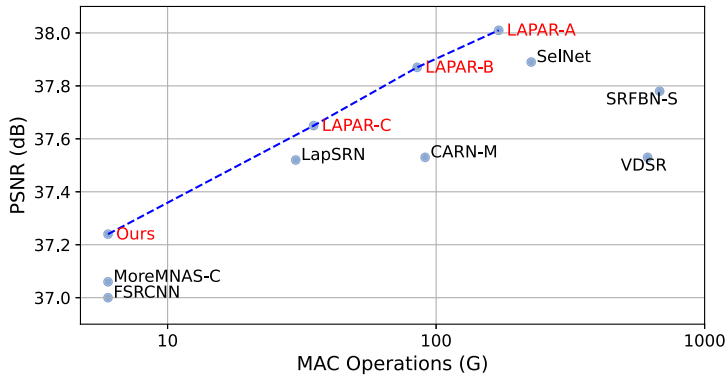

Fig. 1. Comparison between different DNN-based SR networks on Set5 [23] for ×2 setting. MAC represents the count of multiply-accumulate operations required to obtain the HR images. **Peak signal-to-noise Ratio (PSNR)** is a commonly used metric to evaluate the quality of generated HR images.

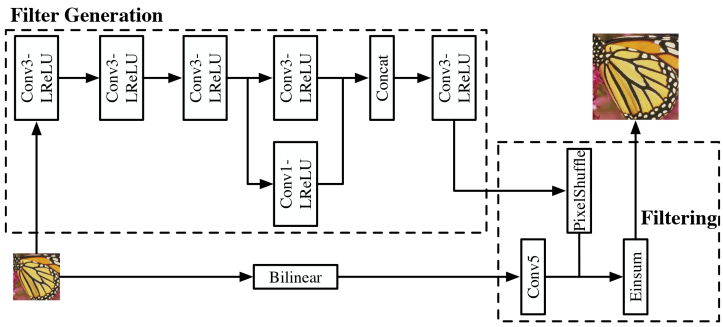

Fig. 2. Adaptive filter based SR.

Figure 2 illustrates the architecture of our proposed SR network, which is an adaptive filter-based SR algorithm that has been refined from LAPAR [24]. The filter generation phase comprises multiple convolutional layers, with the first 3×3 convolution having a stride of 2, while the subsequent convolutions employ a stride of 1. In the filtering phase, we incorporate a convolutional layer to extract features from the interpolated image, and we utilize an einsum operator for calibration, which involves reshaping the filters prior to the calibration process.

2.4 Design Challenges

In recent years, we have witnessed significant improvements on accelerating image classification [26–29], object detection [30–33] and speech recognition [34–36] tasks on FPGAs. However, few works are targeting the SR problem. SR models are very different from the networks mentioned above. They take in a high-dimensional input and give out a larger feature map. This upsampling procedure indicates a larger memory footprint and more intensive computation. Figure 3 illustrates the computation complexity and memory footprint comparison between our target SR network and some famous classification networks. We use **Giga Operations (GOP)** to evaluate the required operations and the sum of feature map size to evaluate the memory footprint of a network. The input size of our SR network is 540×960, while the input size for other networks is 224×224. Although our SR network is no deeper than most classification networks, much larger

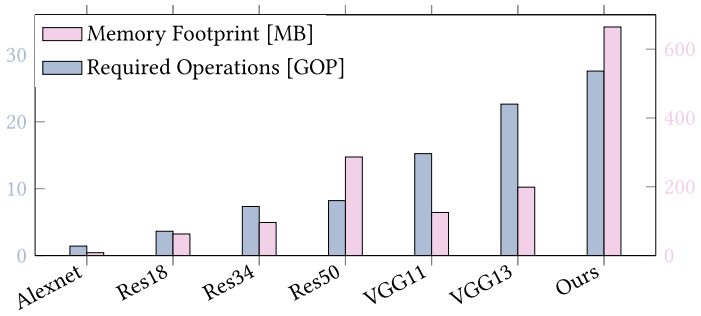

Fig. 3. Computation complexity and memory footprint comparison between our SR network and some famous classification networks.

feature maps inevitably incur higher total computation and communication complexity. In classification and detection tasks, rescaling the input images does not harm the accuracy much when dealing with high-resolution images. While in SR tasks, it is not practicable to reduce computation by resizing. Moreover, the model structure, dataflow, and operators of SR models are distinct from well-studied neural networks. We need specialized optimization to accomplish the objective of real-time inference, especially in an embedded system.

2.5 Design Principles

The roofline model [37, 38] suggests that the actual performance of a system can be constrained by either the computation roof or the communication to computation ratio. Regarding computation, the number of DSPs on embedded FPGAs is severely limited, posing difficulties in constructing high-performance compute engines since most computations heavily rely on DSPs. In terms of communication, managing large feature maps becomes challenging due to restricted on-chip memory and limited bandwidth. These factors can impede efficient data transfer and exacerbate the communication bottleneck. To address the demands of handling large feature maps and intensive computation workloads, we optimize our accelerator from the following three aspects.

Computation. Firstly, we consider improving the maximum attainable performance. Generally, the attainable performance of each DSP depends on the number of operations finished per cycle and working frequency [39], as shown in the following equation:

$$\text{Performance Per DSP} = \frac{\#\text{Operations}}{\#\text{Cycles}} \times \text{Frequency}. \tag{1}$$

Therefore, we should both consider fully leveraging the potential of DSPs and improving the working frequency. To increase operations finished per cycle, we refer to the DSP usage listed in [40] with a slight modification. For the purpose of improving frequency, we consider two policies. One is to let DSPs run at a clock rate that is two times higher than the system clock. Another is to optimize resource usage to ease timing closure. The compute engines are, in fact, the cascade of DSPs. However, we need to incorporate LUTs and FFs to ensure accurate timing when designing the compute engines. There are different ways to organize the connection of DSPs. As we use most of the DSPs to meet the real-time requirement, minimizing the usage of FFs and LUTs is essential to guarantee high frequency. It is tedious and time-consuming to search for the optimal design by trial and error. Therefore, we propose an analytical model to search for the Pareto frontier designs.

Data Communication. A well-designed computing engine is not sufficient because the actual performance can also be bounded by data communication. Given restricted on-chip memory and bandwidth, we propose a novel cache strategy to handle intensive memory traffic. Our cache

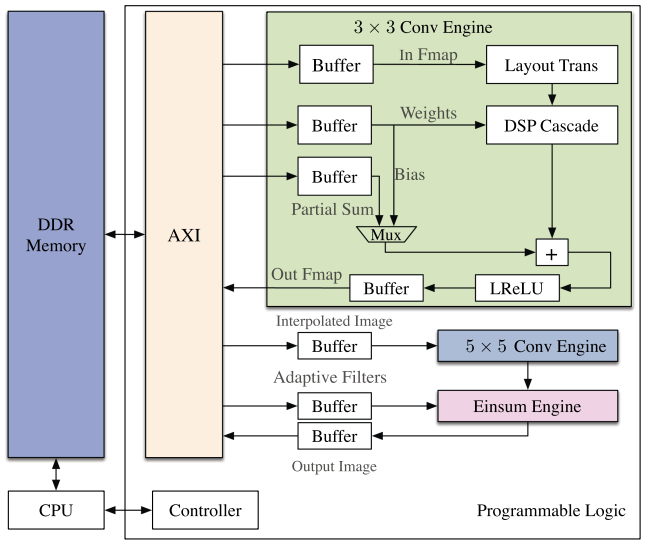

Fig. 4. Architecture of our SR accelerator.

strategy can deal with different strides and is scalable to very large input images. Also, the memory hierarchy efficiently explores the data reuse of input feature maps, convolution kernels, and partial sums to save power. Moreover, the cache strategy can continuously provide data for computation engines without stalling arithmetic operations.

System Level Optimization. We have designed the architecture of our SR network to suit its unique operations and data flow, which differ significantly from those of conventional classification and detection networks. As a result, we can fully exploit the computation and data reuse patterns specific to our network. In order to make the most of limited resources, we have developed a resource allocation scheme that involves solving a constraint optimization problem.

3 ACCELERATOR ARCHITECTURE

Figure 4 provides the sketch of our SR accelerator. The architecture is consistent with the network structure in Figure 2. We only show the main components of the 3×3 convolution engine because the structure of the 5×5 convolution engine is similar. For the 1×1 convolution, the kernel is zero-padded to 3×3 to eliminate additional hardware support. Before convolution, the weights are prefetched to on-chip buffers and stay stationary for further access. Layout trans unit is responsible for preparing data for convolution, and we will illustrate the details of its mechanism in Section 3.2. If a large feature map is split into multiple chunks, storing partial sums to DRAM temporarily and reading them back for accumulation is needed. The controller will decide to accumulate partial sums or biases through a mux.

Firstly, we rely on the 3×3 convolution engine to generate adaptive filters. Then, 5×5 convolution engine and einsum engine work simultaneously to generate the final HR images. To be more specific, the output of 5×5 convolution engine is fed into the einsum engine directly. Once the einsum engine receives data from the 5×5 convolution engine, filters generated in the first phase are read from DRAM to perform computation, and the HR image is generated gradually. Although there is no data dependency between 3×3 convolution and 5×5 convolution, they are processed separately because the limited bandwidth cannot support concurrent computing.

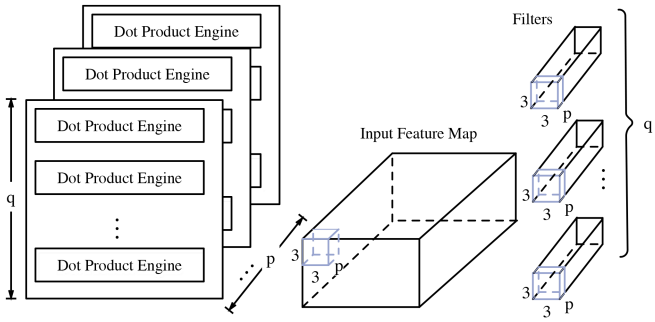

Fig. 5. Stack of DPEs in 3×3 convolution engine.

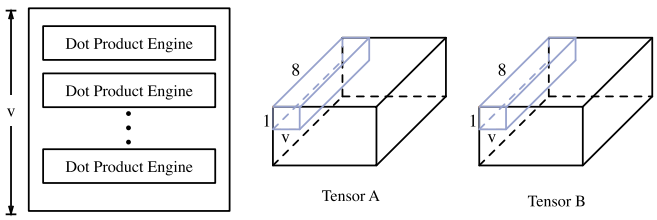

Fig. 6. Stack of DPEs in Einsum Engine.

3.1 Convolution Engine and Einsum Engine

In our SR accelerator, we build a **Dot Product Engine (DPE)** to compute the dot product of two vectors $\boldsymbol{a} = [a_1, a_2, \ldots, a_m]$ and $\boldsymbol{b} = [b_1, b_2, \ldots, b_m]$:

$$\boldsymbol{a} \cdot \boldsymbol{b} = \sum_{i=1}^{m} a_i b_i = a_1 b_1 + a_2 b_2 + \cdots + a_m b_m. \tag{2}$$

Consider the convolution operation

$$\mathcal{O}[n, k, h, w] = \sum_{y} \sum_{r,s} \mathcal{I}[n, y, h + r, w + s] \times \mathcal{K}[k, y, r, s], \tag{3}$$

where $\mathcal{O}, \mathcal{I}, \mathcal{K}$ denote the output tensor, input tensor, and convolution kernels, respectively. The einsum operation between two tensors with summation convention "nchw,nchw→nhw" can be expressed as

$$\mathcal{D}[n, h, w] = \sum_{c} \mathcal{A}[n, c, h, w] \times \mathcal{B}[n, c, h, w], \tag{4}$$

where $\mathcal{A}, \mathcal{B}$ are two input tensors and $\mathcal{D}$ is the output tensor. From Equation (3) and Equation (4) we can see that convolution and einsum can be factorized into multiple dot product operations. For example, when $r = s = 3$ in Equation (3), the inner summation can be seen as the dot product between two vectors with size 9. Based on the above analysis, we use DPEs as building blocks to construct the convolution engine and einsum engine.

Figure 5 illustrates how DPE is stacked to form the 3×3 convolution engine. The processing array is divided into p slices, and each slice contains q DPEs. Within a slice, q DPEs handle the convolution between a specific channel of the input feature map with q convolution kernels simultaneously. Similarly, p slices process p input channels at the same time. In other words, p can be seen as the input channel parallel factor and q can be seen as the kernel parallel factor.

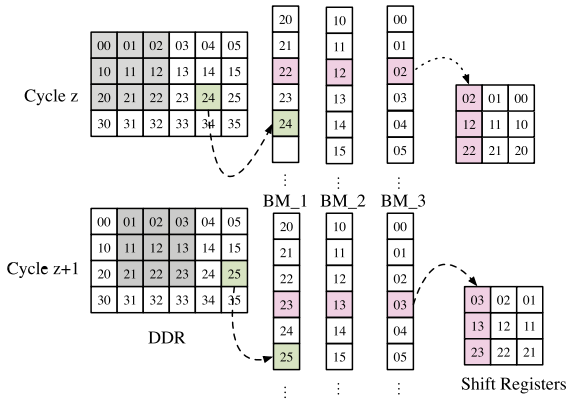

Fig. 7. Cache strategy for stride 1 convolution.

Figure 6 represents the structure of einsum engine when $c = 8$ in Equation (4). v DPEs can consume $8v$ data points from each input tensor and produce v data points every cycle.

3.2 Cache Strategy of Convolution

DNN-based super-resolution networks typically operate on input feature maps that are considerably larger than those used in traditional classification and object detection tasks. Additionally, SR algorithms seek to generate high-resolution images, which results in even larger intermediate feature maps during the model inference process. This can create a bottleneck in the overall system due to limited bandwidth and intensive communication between on-chip and off-chip memory, highlighting the need for an efficient cache strategy to mitigate this issue. To tackle the intense data communication in our SR network, we propose a novel row-buffer-based cache scheme without stalling the convolution engine. Although some previous works also use row/column-based cache strategy, our revised cache scheme can handle different stride sizes effectively. Moreover, we will show that this cache strategy is scalable to large input images given today's BRAM features.

Flexibility. Figure 7 shows the input feature map cache strategy of 3×3 convolution when stride is 1. The three buffers indicate three individual BRAM modules on the FPGA board, and different BRAM models are used to cache different row data from the input feature map. When we fetch data from the three buffers using the same address, acquired data points are exactly from the same column in the feature map. Then, the three data points move to an array of shift registers to rearrange the data layout. This caching scheme enables us to write one data point to the on-chip buffers but read three data points from the on-chip buffers every cycle. After all three BRAMs are filled, the data in the first BRAM becomes out of date, and new data from DRAM can be written into this BRAM. Therefore, when reading from the buffers, there are three states because the latest data can be either in BRAM_1, BRAM_2, or BRAM_3. Thanks to the shift registers, data points can be reused by different sliding windows of convolution. We can significantly reduce the communication traffic between off-chip and on-chip memory. By replacing the oldest data with the newest data, we can utilize on-chip memory effectively. Moreover, as data fetch, computing and data writeback are working simultaneously, the execution time of convolution will overshadow the time spent on data transfer. For 5×5 convolution, we only need to increase the number of BRAMs to five. Figure 8 illustrates the cache strategy of $3\times$ convolution when stride is 2. Instead of placing a whole row of input feature map into a single BRAM, data from one row is put into two BRAMs alternately. For instance, "00, 01" is written into BRAM_1 while "02, 03" is written

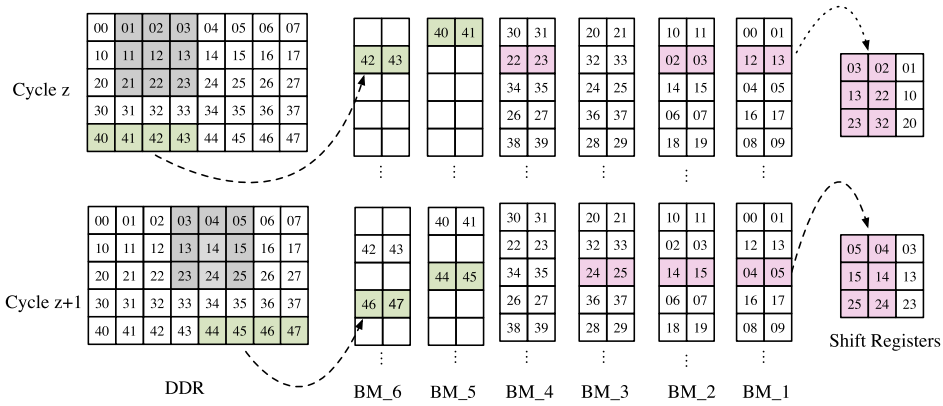

Fig. 8. Cache strategy for stride 2 convolution.

into BRAM_2. When reading from BRAMs, data points from two adjacent columns are pushed into the shift registers every cycle. Similarly, the data out of data can be replaced by new data to save BRAM consumption.

Scalability. We also analyze the scalability of our cache scheme. Considering one BRAM on UltraScale+ architecture, it is capable of storing up to 36K data and can be configured as two independent 18 Kb RAMs. Assuming an input image dimension of $1 \times H \times W$ and a bit width of 8, the BRAM in Figure 8 would require a bit width of 16 and a depth of $W/2$. This means that a single 18 Kb RAM can meet the requirement as long as W is less than or equal to 2048, and a total of three 36 Kb RAMs would be needed. After the stride 2 convolution, the width reduces to $W/2$, allowing one 18 Kb RAM to cache two channels. Therefore, to cache a $C \times \frac{H}{2} \times \frac{W}{2}$ feature map, we would need $\frac{3C}{4}$ 36 Kb RAMs.

4 OPTIMIZATION OF DPE

The efficiency of DPEs is a crucial factor in achieving real-time performance for our SR accelerator, as we stack multiple DPEs to build Convolution Engine and Einsum engine. Moreover, the resource consumption of DPE significantly affects overall resource utilization, which in turn impacts the attainable frequency. This section will explore how to find a DPE topology that minimizes resource utilization and an architectural design methodology that enables high throughput.

4.1 DPE Topology Generation

This subsection will discuss how to organize the connections between DSPs to construct DPE. Firstly, we propose a data structure to characterize the DSP cascade. After that, we present a search algorithm to explore the optimal DPE structure with the least resource consumption.

DPE Topology Abstraction. Equation (2) shows that the dot product of two vectors is composed of a series of **multiply-accumulate (MAC)** operations. By configuration, one DSP on FPGA fabric can be used as a MAC unit, and cascaded DSPs can serve as a DPE. To accumulate partial sum correctly, incorporating LUT-based adders may be needed. Furthermore, registers are indispensable to ensure signals arrive at the input ports of DSP at the right cycle. There are multiple feasible organizations, and different organizations of DSPs may induce different resource utilization of FFs and LUTs. For instance, Figure 9(a) shows two different ways to organize the connection of 5 DSPs. The left-side organization consumes no LUT-based adders but consumes more shift registers to cache the input signal than the right-side organization. It is tedious and error-prone to search

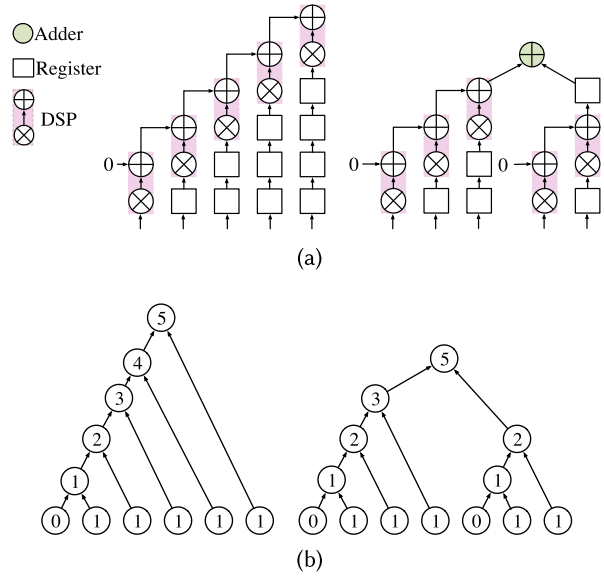

Fig. 9. (a) Two ways to organize DSP cascades; (b) Corresponding ABDTree representations.

Table 1. Notations of ABDTree

Notation	Explanation
t	An ABDTree.
C_i	The set of ABDTrees rooted at i.
$f_o(t)$	The number of leaf nodes noted by 0 in t.
$f_a(t)$	The number of adders in corresponding DPE.
$f_h(t)$	The height of t.
$f_b(t)$	The imbalance degree of ABDTree t.

DSP organizations directly. We need an efficient way to encode DSP connections to search for a DPE structure that minimizes FFs and LUTs usage. A dot product can be seen as the summation of two dot products with more minor scales. Intuitively, we use a binary tree-like data structure to represent the DSP organization. For example, Figure 9(b) shows the tree abstractions of the two DSP organizations in Figure 9(a). Nodes in the tree represent partial sums, and the digit indicates how many products have been accumulated. The leaf node marked by 1 denotes the product $a_i b_i$ for some i, and the root node marked by 5 denotes the result of $\boldsymbol{a} \cdot \boldsymbol{b}$ when $m = 5$ in Equation (2). To better illustrate the abstraction and search process, we provide some definitions regarding this abstraction.

Definition 1 (Adder-Buffer-DSP Tree (ABDTree)). A tree that abstracts the connection among LUT-based adders, registers, and DSPs in a DPE.

Definition 2 (Imbalance degree of an ABDTree). The sum of the height difference between the left child and right child of all none-leaf nodes.

We take the second ABDTree in Figure 9(b) as an example. The height difference between the left child and the right child of node 5 is 1, which is 2 for node 3. Therefore, the imbalance degree of this ABDTree is 1 + 2 + 1 + 1 = 5.

Table 1 lists notations and explanations used in our search procedure. Like a traditional binary tree, we can build up a larger ABDTree by connecting two smaller subtrees. If $t_1 \in C_\eta$ and $t_2 \in C_\gamma$, we can build an ABDTree $t \in C_{\eta+\gamma}$ that satisfies the following equations.

$$f_o(t) = f_o(t_1) + f_o(t_2). \tag{5}$$

$$f_a(t) = f_o(t) - 1. \tag{6}$$

$$f_h(t) = \max(f_h(t_1), f_h(t_2)) + 1. \tag{7}$$

$$f_b(t) = f_b(t_1) + f_b(t_2) + |f_h(t_1) - f_h(t_2)|. \tag{8}$$

Equation (5) shows that the number of leaf nodes marked by 0 in an ABDTree is the sum of such nodes in the left subtree and right subtree. We can obtain the number of LUT-based adders in a DPE by observing the number of nodes marked by 0 in its corresponding ABDTree. As shown in Equation (2), the dot product of two vectors with length n requires n multiplications and $n-1$ additions. Adding a zero to the dot product requires one more adder. As every DSP contributes one adder and one multiplier, adding a zero is penalized with a LUT-based adder except for the first time. That's the intuition that we get Equation (6). Equation (7) means constructing an ABDTree from two subtrees will increase height by 1. Equation (8) indicates that the imbalance degree of an ABDTree comes from the imbalance degree of its left subtree, its right subtree, and the height difference between its left subtree and right subtree. The imbalance degree of ABDTree t reflects the number of registers used in the DSP organization that can be represented by t. By now, we have shown how the resource consumption of a DPE can be inferred from its corresponding ABDTree. To discover DPEs that consume fewer hardware resources, we need to define what kinds of ABDTrees are better than others.

Definition 3 (Pareto Dominance). An ABDTree t_i is said to dominate another ABDTree t_j rooted at same number iif $f_\tau(t_i) \leq f_\tau(t_j)$, $\forall \tau \in \{a, b\}$ and $\exists \tau \in \{a, b\}$ such that $f_\tau(t_i) < f_\tau(t_j)$.

Definition 4 (Pareto Optimality). An ABDTree is said to be Pareto optimal if there is no other ABDTree rooted at the same number that could dominate this ABDTree. Note that we define Pareto Dominance and Pareto Optimality within ABDTrees rooted at the same number.

The height of an ABDTree tree is not taken into account when defining Pareto dominance, despite the fact that the tree's height reflects the end-to-end latency of its corresponding DPE. This is because DPEs operate in a pipelined manner, then latency does not accumulate when the computation is continuously performed in the DPEs.

We perceive that the mapping between a DSP organization and its corresponding ABDTree is invertible. Therefore, the optimal DPE topology generation problem is equal to finding the corresponding ABDTrees that are Pareto Optimal.

Problem 1 (Optimal DPE Topology Generation). Given the number of DSPs in a DPE n, optimal DPE topology generation can be formulated to construct Pareto optimal ABDTrees rooted at n.

DPE Topology Generation Algorithm. Directly exploring ABDTrees is impractical because it requires storing all ABDTrees generated during the search process. Instead, we can decouple the search process and tree construction process. During the search process, only the attributes of ABDTrees and their construction details are recorded. Specifically, we use a tuple $(f_o, f_h, f_b, left, right)$ consisting of five elements to describe an ABDTree. The interpretations of f_o, f_h, and f_b are the same as that in Table 1, while *left* and *right* are pointers to the attributes of the left and right subtrees. Algorithm 1 illustrates the optimal DPE topology generation algorithm. We maintain a bucket to store the attributes of ABDTree rooted at the same number. Initially, we initialize $bucket_1$ and $bucket_2$. Then, we explore the properties of ABDTrees with larger and

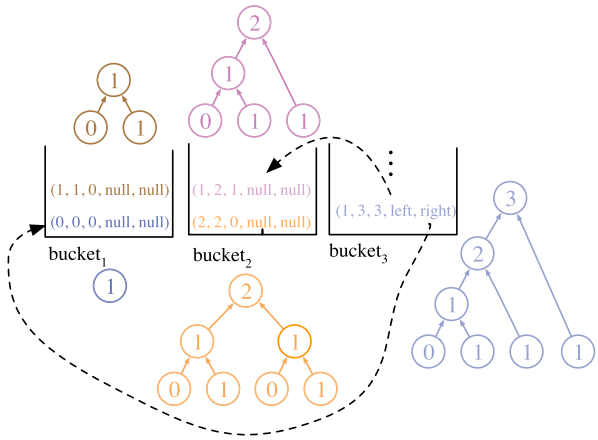

Fig. 10. Tree construction.

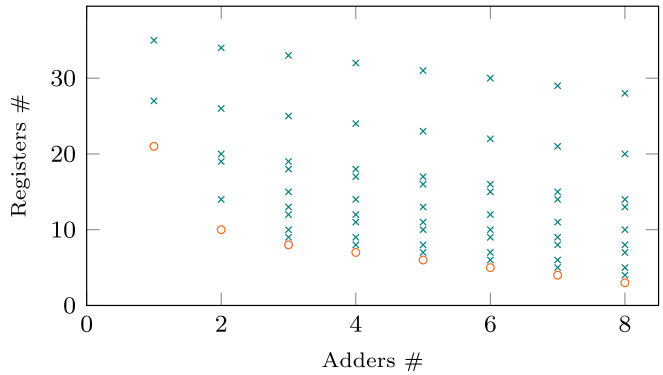

Fig. 11. Pareto frontier of DPE design.

larger size though *SearchABDTree* function. Lines 20-21 indicate that the number of DSP cascaded is limited. For example, if we use 8-bit precision, C_{max} is set to 7 [40]. After that, we prune these non-optimal ABDTrees by comparing the f_o, f_h and f_b. Finally, we can construct a Pareto optimal ABDTree via *ConstructABDTree* function. Figure 10 shows an example of building $bucket_3$ though $bucket_1$ and $bucket_2$. An ABDTree and its corresponding attributes are the same color.

We observe that the generation algorithm can produce many DPE topologies with the same resource utilization, which could increase the search space exponentially as DPE size increases. When search space compression is enabled, we only add DPE topologies with different resource consumption to the solution pool. Figure 11 demonstrates the Pareto frontier when $n = 9$ and $C_{max} = 3$. The red points denote Pareto optimal designs, and the green points represent non-Pareto optimal designs. These Pareto optimal designs achieve a tradeoff between LUT and FF consumptions so that we may choose corresponding DPE topology based on accessible sources for different FPGA boards.

4.2 DPE Architectural Design

This subsection introduces the architectural design of DPEs in the Convolution Engine and Einsum Engine to fully release the potential of DSPs.

ALGORITHM 1: Optimal DPE Topologies Generation Algorithm

Input: n: the number of DSPs in a DPE; C_{max}: the maximum number of cascaded DSPs allowed.
Output: Pareto optimal DSP organizations.

Init $bucket_1 \leftarrow \{(0, 0, 0, null, null)(1, 1, 0, null, null)\}$;
Init $bucket_2 \leftarrow \{(1, 2, 1, null, null)(2, 2, 0, null, null)\}$;
for $i \leftarrow 3$ **to** n **do**
    $bucket_i \leftarrow \emptyset$;
    for $j \leftarrow 1$ **to** $\lfloor \frac{i}{2} \rfloor$ **do**
        $C' \leftarrow$ `SearchABDTree`$(i, i-j)$;
        $bucket_i \leftarrow bucket_i \cup C'$;
    end for
end for
Prune none-optimal ABDTrees in $bucket_n$;
for each $T \in bucket_n$ **do**
    `BuildABDTree`(T);
end for
function `SearchABDTree` (i, j)
    $C' \leftarrow \emptyset$;
    for each $T_\eta \in bucket_i$ **do**
        for each $T_\gamma \in bucket_j$ **do**
            $curT.f_o \leftarrow T_\eta.f_o + T_\gamma.f_o$;
            $curT.f_h \leftarrow \max(T_\eta.f_h, T_\gamma.f_h) + 1$;
            $curT.f_b \leftarrow T_\eta.f_b + T_\gamma.f_b + |T_\eta.f_b - T_\gamma.f_b|$;
            $curT.left \leftarrow T_\eta$;
            $curT.right \leftarrow T_\gamma$;
            if $curT.f_h > C_{max}$ and $curT.f_h == i + j$ **then**
                continue;
            end if
            if Enable search space compression **then**
                if $\nexists T^* \in C' \Rightarrow (curT.f_o == T^*.f_o)$ and $(curT.f_h == T^*.f_h)$ and $(curT.f_b == T^*.f_b)$ **then**
                $C' = C' \cup curTree$;
                end if
            else
                $C' = C' \cup curTree$;
            end if
        end for
    end for
    return C';
end function
function `BuildABDTree`(T)
    if $T.left == empty$ and $T.right == empty$ **then**
        return Corresponding ABDTree;
    else
        Set `BuildABDTree`(T.left) as left subtree;
        Set `BuildABDTree`(T.right) as right subtree;
    end if
end function

Convolution predominates the computation overload of our SR accelerator. Besides, there exists a lot of data reuse in convolution operations. Here we introduce the DPE design in Convolution Engine to take advantage of modern DSP characteristics and data reuse possibilities. Figure 12(a) shows detailed information about the DSP48E2 slice on Xilinx UltraScale+ MPSoC. The DSP48E2

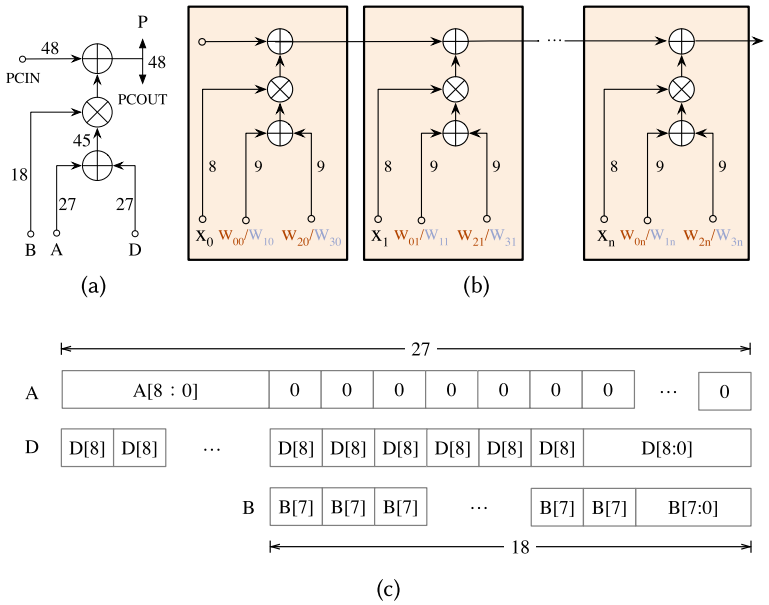

Fig. 12. (a) DSP48E2 on FPGA fabric; (b) DPE in Convolution Engine; (c) Data format of DPE inputs in Convolution Engine.

slice contains a 27-bit pre-adder, a 27 × 18 multiplier, and a 48-bit post-adder. Through proper configuration illustrated in [40], one DSP can accept three INT8 operands *A*, *B*, and *D* to compute *AB* and *AD* in parallel. Quantization inevitably induces quality degradation of output SR images. To keep quality loss as low as possible but preserve 2× speedup, the input activations are quantized to 8 bits, and the weights are quantized to 9 bits. Figure 12(c) shows the data format of DSP inputs, which is slightly different from that in [40]. Another way to improve throughput is to increase the working frequency of DPE. DSPs on FPGA are hard-core components, which means they can run at a very high frequency. However, most DNN accelerators on FPGA run below 200MHz, achieving less than half of the maximum DSP clock rate. To fully release the potential of DSPs, we propose a design with two clock domains, with one clock running 2× faster than another clock. In the DPE, the weights from two different convolution kernels are pre-cached on the chip, and they are fed into DPE alternately every 2× clock. Therefore, two convolution kernels reuse the input feature map in every lower clock. For instance, in Figure 12(b), the weights with the same color are fed into DPE at the same time, while weights with different colors are switched every 2× clock. The input vector $\boldsymbol{x}$ is renewed every 1× clock. In this way, the computation works 2× faster than the data communication between off-chip and on-chip memory.

There is no data reuse in einsum operation, so we simply function a DSP as a MAC unit. Therefore, the input operands are fed into DPE every 1× clock. Also, they are updated every 1× clock.

5 HARDWARE DESIGN PARAMETER CONFIGURATION

DNN-based SR models are hungry for both computation and memory resources. Unsuitable resource management can result in already strained resources, leading to suboptimal system performance. Brute-force search for resource allocation can become untractable when dealing with a vast design space. To address this, various approaches have been proposed to explore optimal hardware design parameter configurations. [41] prioritizes computation resources to achieve maximum

throughput, followed by adjustments to memory and bandwidth resources. Another approach, discussed in [42], develops analytical models for resource utilization and latency modeling. Also, the authors design a tool to determine the optimal configuration. For our accelerator, the allocation of resources is about deciding the parallel factors of computation engines. As mentioned in Section 3.2, our memory hierarchy makes sure that computation engines are not stalled by data communication. This important property makes the modeling of latency and resource consumption very straightforward. Firstly, the latency of our accelerator can be inferred by summing up the required cycles to process feature maps for computation engines. Furthermore, the parallel factors directly determine the consumption of DSPs and BRAMs. Finally, the required bandwidth can be deduced given the defined dataflow. Here, we complete the resource allocation by solving a constraint programming problem. In the optimization problem, our objective is to minimize latency with variables representing the parallel factors and constraints representing utilization of FPGA resources should not exceed the resources the FPGA board equipped. The problem can be solved easily by heuristics without a time-consuming trial-and-error search.

If we define the input channel parallelism factor and output channel parallelism factor of 3×3 convolution as p_1 and q_1 respectively, and use C_i^{in} and C_i^{out} to denote the number of input channels and output channels of the i_{th} 3×3 convolution layer, the latency of all 3×3 convolutional layers L_1 can be formulated by

$$L_1 = \sum_{i \in \mathcal{X}} \left\lceil \frac{C_i^{in}}{p_1} \right\rceil \times \left\lceil \frac{C_i^{out}}{4q_1} \right\rceil \times H_i^{out} \times W_i^{out} \times \frac{1}{Fre}, \tag{9}$$

where H_i^{out} and W_i^{out} represent the height and width of the output feature map for the i_{th} 3×3 convolution layer. $\mathcal{X}$ denotes the set of 3×3 convolutional layers and Fre represents the working frequency of the accelerator. The "4" before q_1 refers to the 4× speedup discussed in Section 4.2. Given that the time spent on data transfer can be overshadowed by computation time, L_1 is close to the actual runtime of the filter generation phase. Similarly, the latency of 5×5 convolutional layer L_2 is given by

$$L_2 = \left\lceil \frac{C_*^{in}}{p_2} \right\rceil \times \left\lceil \frac{C_*^{out}}{4q_2} \right\rceil \times H_*^{out} \times W_*^{out} \times \frac{1}{Fre}, \tag{10}$$

where H_*^{out} and W_*^{out} represent the height and the width of the output feature map after the 5×5 convolution. C_*^{in}, C_*^{out} denote the input channels and output channels of 5×5 convolution. Since the 5×5 Convolution Engine and the Einsum Engine form a balanced pipeline, the latency of the two computation engines is nearly equal to the latency of the 5×5 convolution engine. Thus, the latency of our accelerator can be approximated by $L_1 + L_2$.

The constraint programming then can be formulated as in Formula (11).

$$\min_{p_1, q_1, p_2, q_2, v} \quad L_1 + L_2 \tag{11a}$$

$$\text{s.t.} \quad 9p_1q_1 + 25p_2q_2 + 8v \le R_{DSP}; \tag{11b}$$

$$3\left\lceil \frac{p1}{4} \right\rceil + 5\left\lceil \frac{p2}{4} \right\rceil \le R_{BRAM}; \tag{11c}$$

$$4q_2 = 8v; \tag{11d}$$

$$b_a(p_1 + 4q_1 + 4q_1) \le \frac{BW}{Fre}; \tag{11e}$$

$$b_a(p_2 + 8v + v) \le \frac{BW}{Fre}; \tag{11f}$$

$$p_1, q_1, p_2, q_2 \text{ are power of two.} \tag{11g}$$

Table 2. Performance Comparison between Our Network and Two Lightweight SR Networks with Similar Computation Cost

Method	Params	MAC	Set5 PSNR/SSIM	Set14 PSNR/SSIM	BSDS100 PSNR/SSIM	Urban100 PSNR/SSIM
FSRCNN [18]	12K	6G	37.00/0.9558	32.63/0.9088	31.53/0.8920	29.88/0.9020
MoreMNAS-C [25]	25K	6G	37.06/0.9561	32.75/0.9094	31.50/0.8904	29.92/0.9023
Ours	103K	6G	**37.24/0.9724**	**32.79/0.9106**	**31.59/0.8920**	**30.01/0.9043**

Here b_a and BW denote the bit precision of activations and DRAM throughput, respectively. In our design, the bit width of activation is 8 and the theoretical throughput of DDR for Zynq UltraScale+ MPSoC is 19200MB/s. v is the parallel factor of the Einsum Engine defined in Figure 6. Constraint (11b) requires the sum of DSP slices consumed by different computational engines should be no more than the overall DSP resources on the FPGA board. According to the analysis in Section 3.2, we can get the BRAM consumption of 3×3 and 5×5 convolution. The input channel of stride 2 convolution is always 1, which is much less than that of stride 1 convolution. Besides, BRAMs for stride 1 and stride 2 convolution can be reused. Therefore, we only consider the BRAM consumption of stride 1 convolution. Constraint (11c) requires BRAMs to cache convolution engine should be less than total BRAMs.

As mentioned in Section 3, the 5×5 Convolution Engine and Einsum Engine form a pipeline. To ensure the perfect pipeline, the 5×5 Convolution Engine's output data should be half the size of the Einsum Engine's input, which is guaranteed by Constraint (11d). Besides resource restrictions, bandwidth limitations are also considered. For 3×3 convolution, we consider the bottleneck case where it processes not just input feature maps, but also partial sums for accumulation. Every cycle, the engine takes in p_1 input feature map data points and $4q_1$ partial sum data points, while giving out $4q_1$ data points. Constraint (11e) shows that the data exchange rate between DRAM and on-chip buffers should be less than the bandwidth of DRAM. Constraint (11f) can be obtained by considering the memory traffic of the filtering stage. When 5×5 convolution reads p_2 data points, The Einsum Engine reads $8v$ data points and gives out v data points. (11g) indicates the parallel factors should be the power of two for efficient hardware implementation. Since the optimization problem's solution space is small, enumerating all possible values of decision variables can yield the solution.

6 EXPERIMENTS

6.1 Effectiveness of Proposed SR Network

As described in Section 2.3, our lightweight SR network is a simplified version derived from LAPAR. The training of our network was conducted on a single NVIDIA GeForce RTX 3090 GPU, utilizing a mini-batch size of 32, and spanning 800K iterations on the DIV2K [43] and Flickr2K image datasets. During the testing phase, we evaluated the performance of our SR network using multiple standard benchmarks, including Set5 [23], Set14 [44], BSDS100 [45], and Urban100 [46]. The evaluation was based on two widely used metrics: **peak signal-to-noise ratio (PSNR)** and the **structural similarity index (SSIM)**. In Table 2, we present a performance comparison between our network and two other lightweight SR networks, namely FSRCNN [18] and MoreMNAS-C [25]. Our SR network achieves superior performance with a similar computational cost. It is important to note that although our network has more parameters, after quantization, they can all be stored in on-chip buffers without increasing the data communication overhead between on-chip and off-chip memory.

Table 3. Optimal Solution Found by Constraint Programming

Parallel factor	p_1	q_1	p_2	q_2	v
Optimal solution	32	4	1	2	1

Table 4. Implementation Details of Our Accelerator on Two Embedded FPGA Boards

Board	SR	Frequency	DSP Frequency	DSP	BRAM	FF	LUT
KV260	540p → 1080p	200MHz	400MHz	1210 (97%)	116 (81%)	143252 (61%)	65212 (56%)
ZCU104	720p → 1440p	250MHz	500MHz	1211 (70%)	134 (43%)	146782 (32%)	69164 (30%)

6.2 Experimental Settings

To achieve fine-grained DSP control and high-frequency design, we implement our accelerator using SystemVerilog, and bitstream generation is performed by Vivado 2021.1. Our accelerator will only process the Y channel, while other operations like color space conversion, interpolation of the other two channels, and so on, are finished by the ARM CPU on the FPGA board. The whole application runs on a customized Linux system based on Xilinx Petalinux, and we rely on memory-mapped kernels to control the execution of the accelerator.

To verify the effectiveness of the proposed super-resolution framework, we implement our target SR network on two embedded FPGAs. Firstly, we implement a 540p → 1080p SR accelerator on Kria KV260, a low-end FPGA board. To tackle the resource shortage, our proposed resource allocation scheme is applied to minimize latency, and the optimal solution we find is listed in Table 3. To see the potential of DPE to run at a higher frequency, we also realize a 720p → 1440p SR accelerator on the ZCU104 with the same parallel factors. It is worth noting that our design can easily be adapted to support higher resolutions, such as 4K. However, handling larger feature maps would require larger parallel factors, consuming more hardware resources. In such cases, a higher-end FPGA card like Alveo U200 would be necessary to achieve real-time inference. Additionally, if we aim to achieve better SR quality, we may need to incorporate more layers into our SR network, which would also require increased hardware resources. Nevertheless, the proposed optimization techniques, such as fine-grained DSP control, design space exploration of DPE, and resource allocation techniques, remain valuable in developing a high-performance accelerator. Table 4 lists the details of the two implementations. We can see that the DSP consumption is almost the same (ZCU104 spends an extra DSP on calculating the AXI address). On the KV260, the system runs at 200MHz and the DSPs run at only 400MHz due to high resource utilization. As the ZCU104 possesses adequate resources, timing closure is more accessible, so the accelerator can run at a higher frequency. Figure 13(a) shows the 540p→1080p demo on KV260. Figure 13(b) and Figure 13(c) show the layout after placement and route on the two FPGA boards respectively. We observe that the resource utilization is very high on KV260 but is much lower on ZCU104.

6.3 Comparison with FPGA Automation Tools

Firstly, we compare our accelerator with two FPGA automation tools. One is DNNBuilder [41], an open-source automated tool to generate FPGA accelerators. Another is Xilinx DPU [50], a commercial DNN accelerator IP. As DNNBuilder only supports the convolutional operations of our SR network, we test on a model with five convolution layers that can mimic the memory footprint and required computations of the filter generation phase in our SR network. We replace LeakyReLU with ReLU, and substitute the concat operation, together with the two convolutions before it by a 3×3 convolution that can produce a feature map with the same dimension (We find that we can also use a 1×1 convolution, which does not affect the results). The resource budget is adjusted

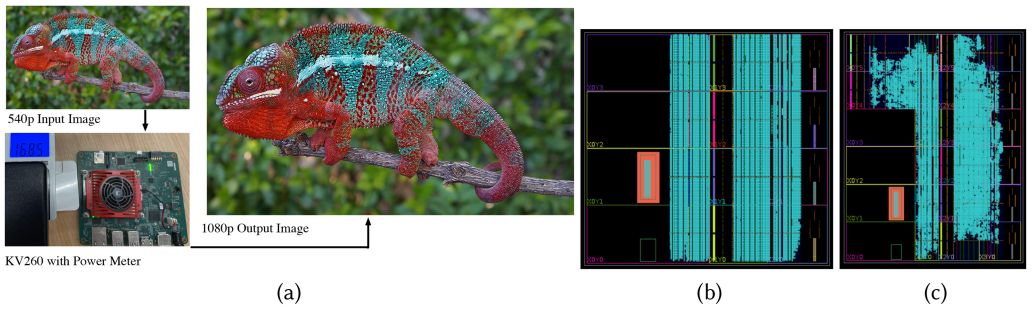

Fig. 13. (a) 540p → 1080p SR running on KV260 with a power meter monitoring the power consumption; (b) Physical implementation on KV260; (c) Physical implementation on ZCU104.

Table 5. Comparison with SOTA FPGA Solutions and Commercial GPU and ASIC

Accelerator	540p→1080p				720p→1440p			
	Latency (ms)	FPS	Power (W)	Power Efficiency (FPS/W)	Latency (ms)	FPS	Power (W)	Power Efficiency (FPS/W)
NX GPU [47]	41.82	23.91	17.35	1.38	73.37	13.63	17.67	0.77
Ascend 310 [48]	52.04	19.22	10.14	1.90	94.32	10.60	10.46	1.01
Xilinx DPU [49]	>37.31	<26.80	N/A	N/A	>56.14	<17.81	N/A	N/A
DNNBuilder [41]	>186.57	<5.36	N/A	N/A	>82.92	<12.06	N/A	N/A
Ours	**27.94**	**35.79**	16.85	**2.12**	**39.74**	**25.16**	23.81	**1.06**

540p→1080p SR is based on KV260 and 720p→1440p is based on ZCU104 for Vitis-AI, DNNBuilder and our accelerator.

based on attainable resources on KV260 and ZCU104, and the reported throughput is estimated by the profiler embedded in the tool. As shown in Table 5, DNNBuilder can only achieve less than 5.36 FPS for 540p→1080p SR on KV260, and less than 12.06 FPS for 720p→1440p SR on ZCU104. We use "<" here because only a sub-network is tested. If we implement unsupported operators on the CPU, the end-to-end runtime would be even worse. DNNBuilder uses a pipeline structure, with every layer possessing an individual compute unit. This architecture works well for networks with small feature maps in each layer but not for SR networks with much larger feature maps. For Xilinx DPU, we rely on the Vitis AI development stack to test the performance of DPU. We only count the time for convolution, concat, and LeakyReLU because Vitis AI does not support other operators. Xilinx DPU can only attain less than 26.80 FPS for 540p→1080p SR on KV260 and less than 17.81 FPS for 720p→1440p SR on ZCU104. Although DPUs are designed to accelerate general convolution neural networks, they can only support very limited operators, thus unable to enable real-time SR.

6.4 Comparison with Commercial GPU & AI Chips

We also compare our accelerator with two commercial tools: NVIDIA Jetson Xavier NX [47] and Atlas 200 DK [48]. NVIDIA Jetson Xavier NX is an embedded computing board that can deliver up to 21 TOPS for AI workloads. Atlas 200 DK is a higher-performance AI application development board equipped with one of the most powerful edge AI chips, Ascend 310. The two commercial accelerators both take advantage of a compiler to translate a DNN model to machine codes. As our accelerator uses INT8 data type to boost the performance, the SR model also inferences at INT8 mode on the two devices if any operators support INT8 quantization for a fair comparison. For the software stack, we adopt CANN 5.0.2.alpha005 on Atlas 200 and TensorRT 8.0.1 on NVIDIA NX, respectively. TensorRT does not support einsum operation directly. It will automatically convert it

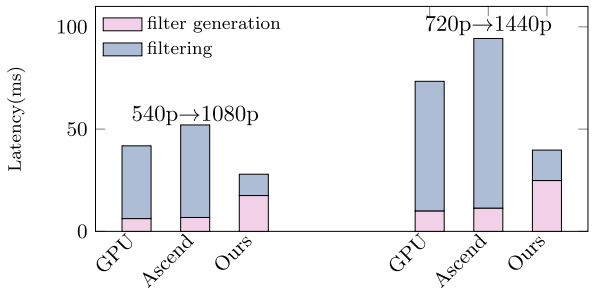

Fig. 14. Runtime breakdown of accelerators.

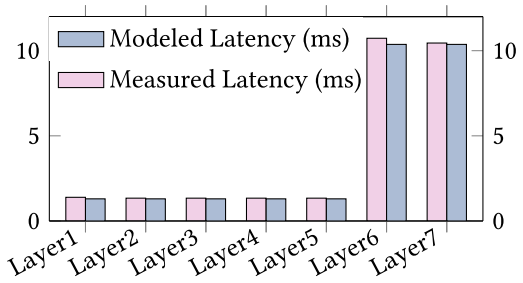

Fig. 15. Comparison between measured latency and modeled latency of different layers.

into an `element-wise multiplication` operation followed by a `reducesum` operation, which the GPU can provide native hardware support. This equivalent transformation can ensure the consistency of the output image. As CANN cannot finish the operation mapping, we perform the same conversion on the original SR network before feeding the network into CANN for compilation. Then, we can run our SR network end to end, providing a fair comparison.

Figure 14 shows the runtime breakdown of the two phases of our SR model. We can see that the latency of filtering on NX and Ascend310 is much lower than that on our accelerator because the two commercial tools possess hardware components highly optimized for convolution. However, they do not perform well on operators like `pixelshuffle`, `multiplication`, and `reduceSum`, which are common in SOTA DNN-based SR networks. Generally, the architectures of NX and Ascend310 are optimized for widely adopted DNN models. There is a gap between these commercial development boards and SR models, both in operator and data flow levels.

6.5 Comparison with SR Accelerators on FPGAs

Table 6 presents a comprehensive comparison of performance and resource consumption among various super-resolution (SR) accelerators implemented on FPGAs. We use AdaRound [53] quantization algorithm to quantize our SR network into 8-bit. It is worth noting that previous accelerators prioritized high frame rates for high-resolution images but relied on simplified learning techniques, resulting in compromised super-resolution quality. In contrast, our accelerator adopts an end-to-end approach based on **deep neural networks (DNN)**, ensuring the preservation of superior SR quality associated with DNN-based algorithms. In edge scenarios, where hardware resources, power supply, and transmission bandwidth are often constrained, our design strategically focuses on achieving an optimal balance between FPS, resolution, and super-resolution quality. Our accelerator prioritizes efficient and practical performance over merely pursuing high FPS and

Table 6. Comparisons of Different SR Accelerators on FPGAs

Method	[51]	[7]	[8]	[52]	Ours-1	Ours-2
FPGA device	XCKU040	XCKU040	XC7K410T	XCKU15P	KV26	XCZU7EV
Freqency	150MHZ	150MHz	130MHz	160MHz	200MHz	250MHz
DSP	108	1920	1512	1820	1210	1211
LUT	3K	151K	167K	98K	65K	69K
FF	2K	121K	158K	57K	143K	147K
Memory	92K	194KB	945KB	4842KB	4176KB	4824KB
Output resolution	3840×2160	3840×2160	2880×1280	3840×2160	1920×1080	2560×1440
Bit-width	Fxp	14-bit Fxp	13-bit Fxp	16-bit Fxp	8-bit Fxp	8-bit Fxp
Frame rate	60fps	60fps	62.7fps	76fps	35.8fps	25.2fps
SR dataset	PSNR/SSIM	PSNR/SSIM	PSNR/SSIM	PSNR/SSIM	PSNR/SSIM	
Set5	34.78/0.9460	36.64/0.9543	36.40/0.9527	36.76/0.9553	37.14/0.9566	
Set14	31.63/0.9083	32.47/0.9770	32.21/0.9047	32.51/0.9076	32.15/0.8968	
BSDS100	30.48/0.8776	31.31/0.8877	31.15/0.8858	31.31/0.8887	31.52/0.8905	
Urban100	-	29.32/0.8939	-	29.30/0.8952	29.94/0.9026	

Table 7. Comparison of DSP Efficiency with CNN Accelerators

Accelerators	[54]	[55]	[56]	[42]	Ours-1	Ours-2
FPGA chip	Zynq XC7Z045	Zynq XC7Z045	Altera Arria-10 GX 1150	PYNQ Z1	KV26	Zynq ZU7EV
DSP Frequency	150MHz	100MHz	150MHz	100MHz	**400MHz**	**500MHz**
Precision	Fix16	Fix16	Fix16	Fix12	Fix8	Fix8
DSPs (used/total)	780/900	824/900	1046/1518	220/220	1210/1248	1211/1728
Performance (GOPS)	137	230	316	83	987	1234
DSP Efficiency (GOPS/DSP)	0.18	0.28	0.30	0.38	**0.82**	**1.02**

resolution. There is a lower performance on the Set14 benchmark due to its sensitivity to low-bit quantization. However, this benchmark, along with Set5, comprises a limited number of images (5 and 14, respectively) and may not fully represent broader performance. Our analysis across more comprehensive benchmarks like BSDS100 and Urban100, which contain 100 images each, shows a more substantial improvement. This underscores the effectiveness of our approach in delivering improved super-resolution quality across a wider range of scenarios. Furthermore, our accelerators are capable of achieving real-time performance (>25 fps), allowing for seamless processing of video streams without any visual degradation.

6.6 Effectiveness of DPE Optimization

In Section 4, we introduce a DPE topology generation algorithm aimed at minimizing resource consumption and an architectural design methodology that enables high throughput. These two techniques significantly enhance the effective utilization of DSPs. Since convolution contributes to the main computation overhead of our SR accelerator, we compare our design with previous **Convolutional Neural Network (CNN)** accelerators as well. As shown in Table 7, our design achieves a DSP working frequency of 400MHz to 500MHz, which is notably higher than that of previous CNN accelerators. This efficient utilization of DSPs allows our design to deliver the highest DSP efficiency compared to previous works. On the ZCU104 platform, our SR accelerator surpasses those in [54], [55], [56], and [42] by 5.6×, 3.6×, 3.4×, and 2.7×, respectively, in terms of DSP efficiency.

6.7 Effectiveness of Cache Strategy

In order to validate the efficiency of our cache strategy, we study the relationship between measured latency and modeled latency of different layers on KV260 when the input image is 540p. The theoretical latency of a layer is obtained by counting the number of cycles needed to finish the computation, as listed in Equation (9) and Equation (10). Since 5×5 convolution and einsum

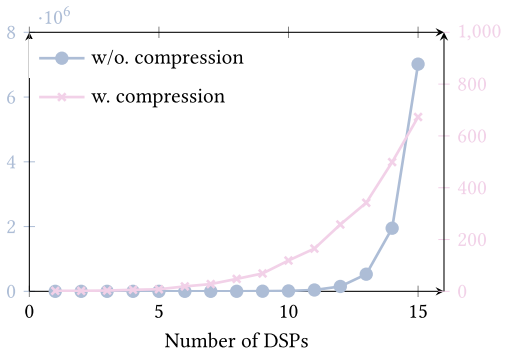

Fig. 16. Effectiveness of search space compression technique in DPE topology generation procedure.

are fused, we refer to it as a single layer named layer 7. From Figure 15, we notice that the ratio between measured latency and modeled latency is very close to 1. The data fetch stage, compute stage, and data writeback stage pipeline are not stalled by data communication, which means the performance is bounded by computation, and DSPs are fully utilized. Also, the ratio validates the rationality of our latency modeling in Section 5.

6.8 Effectiveness of Search Space Compression

The effectiveness of our search space compression technique, discussed in Section 4.1, is demonstrated in Figure 16. Without compression, the search space undergoes exponential growth, becoming unsolvable when the number of DSPs is large. By employing search space compression, we eliminate the need to store multiple DPE designs with identical resource consumption. While this reduction may limit the number of attainable Pareto optimal DPE designs, it enables the handling of larger-scale DSP cascades. For instance, with 15 DSPs, the potential DPE topologies prior to compression can exceed 6×10^6, whereas with compression enabled, it is reduced to less than 800.

7 CONCLUSION AND FUTURE WORK

This paper introduces a lightweight SR network paired with an accelerator designed specifically for real-time super-resolution on edge FPGAs. Addressing the challenge of reconciling compute-intensive SR tasks with limited FPGA resources, we optimize the DSPs in two ways: by increasing the number of operations executed per cycle and enhancing the operating frequency. To streamline the timing closure of the accelerator, we offer a design space exploration framework that identifies the most resource-efficient DSP cascade configurations in terms of LUTs and FFs. Moreover, we put forward a flexible and scalable cache strategy that prevents computation stalling. Our customized architecture and resource allocation approach are crafted to maximize the use of the limited resources available on FPGAs. Experimental results demonstrate that our accelerator not only excels beyond previous FPGA solutions but also outpaces commercial GPUs and AI chips.

Multi-modal computing with multiple DNNs has emerged as a critical workload at the edge in recent years [57]. This domain presents similar challenges, including resource scarcity and high computational demands. In future work, we plan to extend our research to design efficient multi-modal computing accelerators on FPGAs.

REFERENCES

[1] Chao Dong, Chen Change Loy, Kaiming He, and Xiaoou Tang. 2014. Learning a deep convolutional network for image super-resolution. In *European Conference on Computer Vision*. Springer, 184–199.

[2] Yulun Zhang, Kunpeng Li, Kai Li, Lichen Wang, Bineng Zhong, and Yun Fu. 2018. Image super-resolution using very deep residual channel attention networks. In *Proceedings of the European Conference on Computer Vision (ECCV)*. 286–301.

[3] Wei-Sheng Lai, Jia-Bin Huang, Narendra Ahuja, and Ming-Hsuan Yang. 2018. Fast and accurate image super-resolution with deep Laplacian pyramid networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 41, 11 (2018), 2599–2613.

[4] Kai Zhang, Wangmeng Zuo, and Lei Zhang. 2018. Learning a single convolutional super-resolution network for multiple degradations. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 3262–3271.

[5] Tao Dai, Jianrui Cai, Yongbing Zhang, Shu-Tao Xia, and Lei Zhang. 2019. Second-order attention network for single image super-resolution. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 11065–11074.

[6] Zhuolun He, Hanxian Huang, Ming Jiang, Yuanchao Bai, and Guojie Luo. 2018. FPGA-based real-time super-resolution system for ultra high definition videos. In *2018 IEEE 26th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. IEEE, 181–188.

[7] Yongwoo Kim, Jae-Seok Choi, and Munchurl Kim. 2018. A real-time convolutional neural network for super-resolution on FPGA with applications to 4K UHD 60 fps video services. *IEEE Transactions on Circuits and Systems for Video Technology* 29, 8 (2018), 2521–2534.

[8] Jung-Woo Chang, Keon-Woo Kang, and Suk-Ju Kang. 2018. An energy-efficient FPGA-based deconvolutional neural networks accelerator for single image super-resolution. *IEEE Transactions on Circuits and Systems for Video Technology* 30, 1 (2018), 281–295.

[9] Ngoc Long Nguyen, Jérémy Anger, Axel Davy, Pablo Arias, and Gabriele Facciolo. 2021. Self-supervised multi-image super-resolution for push-frame satellite images. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 1121–1131.

[10] Liangpei Zhang, Hongyan Zhang, Huanfeng Shen, and Pingxiang Li. 2010. A super-resolution reconstruction algorithm for surveillance images. *Signal Processing* 90, 3 (2010), 848–859.

[11] Pejman Rasti, Tonis Uiboupin, Sergio Escalera, and Gholamreza Anbarjafari. 2016. Convolutional neural network super resolution for face recognition in surveillance monitoring. In *International Conference on Articulated Motion and Deformable Objects*. Springer, 175–184.

[12] Hayit Greenspan. 2009. Super-resolution in medical imaging. *The Computer Journal* 52, 1 (2009), 43–63.

[13] Yawen Huang, Ling Shao, and Alejandro F. Frangi. 2017. Simultaneous super-resolution and cross-modality synthesis of 3D medical images using weakly-supervised joint convolutional sparse coding. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 6070–6079.

[14] Jithin Saji Isaac and Ramesh Kulkarni. 2015. Super resolution techniques for medical image processing. In *2015 International Conference on Technologies for Sustainable Development (ICTSD)*. IEEE, 1–6.

[15] Hyunho Yeo, Youngmok Jung, Jaehong Kim, Jinwoo Shin, and Dongsu Han. 2018. Neural adaptive content-aware internet video delivery. In *13th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 18)*. 645–661.

[16] Yinjie Zhang, Yuanxing Zhang, Yi Wu, Yu Tao, Kaigui Bian, Pan Zhou, Lingyang Song, and Hu Tuo. 2020. Improving quality of experience by adaptive video streaming with super-resolution. In *IEEE INFOCOM 2020-IEEE Conference on Computer Communications*. IEEE, 1957–1966.

[17] Chao Dong, Chen Change Loy, Kaiming He, and Xiaoou Tang. 2015. Image super-resolution using deep convolutional networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 38, 2 (2015), 295–307.

[18] Chao Dong, Chen Change Loy, and Xiaoou Tang. 2016. Accelerating the super-resolution convolutional neural network. In *Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11-14, 2016, Proceedings, Part II 14*. Springer, 391–407.

[19] Jiwon Kim, Jung Kwon Lee, and Kyoung Mu Lee. 2016. Accurate image super-resolution using very deep convolutional networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 1646–1654.

[20] C. Ledig, L. Theis, F. Huszar, J. Caballero, A. Cunningham, A. Acosta, A. Aitken, A. Tejani, J. Totz, Z. Wang, and W. Shi. 2017. Photo-realistic single image super-resolution using a generative adversarial network. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 4681–4690.

[21] Younghyun Jo, Seoung Wug Oh, Jaeyeon Kang, and Seon Joo Kim. 2018. Deep video super-resolution network using dynamic upsampling filters without explicit motion compensation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 3224–3232.

[22] Ben Mildenhall, Jonathan T. Barron, Jiawen Chen, Dillon Sharlet, Ren Ng, and Robert Carroll. 2018. Burst denoising with kernel prediction networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2502–2510.

[23] Marco Bevilacqua, Aline Roumy, Christine Guillemot, and Marie-Line Alberi-Morel. 2012. Low-complexity single-image super-resolution based on nonnegative neighbor embedding. In *British Machine Vision Conference, BMVC 2012, Surrey, UK, September 3-7, 2012*, BMVA Press, 1–10.

[24] Wenbo Li, Kun Zhou, Lu Qi, Nianjuan Jiang, Jiangbo Lu, and Jiaya Jia. 2020. LAPAR: Linearly-assembled pixel-adaptive regression network for single image super-resolution and beyond. *Advances in Neural Information Processing Systems* 33 (2020), 20343–20355.

[25] Xiangxiang Chu, Bo Zhang, and Ruijun Xu. 2020. Multi-objective reinforced evolution in mobile neural architecture search. In *European Conference on Computer Vision*. Springer, 99–113.

[26] Naveen Suda, Vikas Chandra, Ganesh Dasika, Abinash Mohanty, Yufei Ma, Sarma Vrudhula, Jae-sun Seo, and Yu Cao. 2016. Throughput-optimized OpenCL-based FPGA accelerator for large-scale convolutional neural networks. In *Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. 16–25.

[27] Yifan Yang, Qijing Huang, Bichen Wu, Tianjun Zhang, Liang Ma, Giulio Gambardella, Michaela Blott, Luciano Lavagno, Kees Vissers, John Wawrzynek, and Kurt Keutzer. 2019. Synetgy: Algorithm-hardware co-design for convnet accelerators on embedded fpgas. In *Proceedings of the 2019 ACM/SIGDA International Symposium on Field Programmable Gate Arrays*. 23–32.

[28] Yao Chen, Jiong He, Xiaofan Zhang, Cong Hao, and Deming Chen. 2019. Cloud-DNN: An open framework for mapping DNN models to cloud FPGAs. In *Proceedings of the 2019 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. 73–82.

[29] Yichi Zhang, Junhao Pan, Xinheng Liu, Hongzheng Chen, Deming Chen, and Zhiru Zhang. 2021. FracBNN: Accurate and FPGA-efficient binary neural networks with fractional activations. In *The 2021 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. 171–182.

[30] Duy Thanh Nguyen, Tuan Nghia Nguyen, Hyun Kim, and Hyuk-Jae Lee. 2019. A high-throughput and power-efficient FPGA implementation of YOLO CNN for object detection. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 27, 8 (2019), 1861–1873.

[31] Hongxiang Fan, Shuanglong Liu, Martin Ferianc, Ho-Cheung Ng, Zhiqiang Que, Shen Liu, Xinyu Niu, and Wayne Luk. 2018. A real-time object detection accelerator with compressed SSDLite on FPGA. In *2018 International Conference on Field-Programmable Technology (FPT)*. IEEE, 14–21.

[32] Anupreetham Anupreetham, Mohamed Ibrahim, Mathew Hall, Andrew Boutros, Ajay Kuzhively, Abinash Mohanty, Eriko Nurvitadhi, Vaughn Betz, Yu Cao, and Jae-sun Seo. 2021. End-to-end FPGA-based object detection using pipelined CNN and non-maximum suppression. In *2021 31st International Conference on Field-Programmable Logic and Applications (FPL)*. IEEE, 76–82.

[33] Suchang Kim, Seungho Na, Byeong Yong Kong, Jaewoong Choi, and In-Cheol Park. 2021. Real-time SSDLite object detection on FPGA. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 29, 6 (2021), 1192–1205.

[34] Song Han, Junlong Kang, Huizi Mao, Yiming Hu, Xin Li, Yubin Li, Dongliang Xie, Hong Luo, Song Yao, Yu Wang, Huazhong Yang, and William (Bill) J. Dally. 2017. Ese: Efficient speech recognition engine with sparse lstm on fpga. In *Proceedings of the 2017 ACM/SIGDA International Symposium on Field- Programmable Gate Arrays*. 75–84.

[35] Chen-Lu Li, Yu-Jie Huang, Yu-Jie Cai, Jun Han, and Xiao-Yang Zeng. 2018. FPGA implementation of LSTM based on automatic speech recognition. In *2018 14th IEEE International Conference on Solid-State and Integrated Circuit Technology (ICSICT)*. IEEE, 1–3.

[36] Huaixiang Hu, Jiatong Li, Chunchun Wu, Xueyang Li, and Yuping Chen. 2022. Design and implementation of intelligent speech recognition system based on FPGA. In *Journal of Physics: Conference Series*, Vol. 2171. IOP Publishing, 012010.

[37] Samuel Williams, Andrew Waterman, and David Patterson. 2009. Roofline: An insightful visual performance model for multicore architectures. *Commun. ACM* 52, 4 (2009), 65–76.

[38] Chen Zhang, Peng Li, Guangyu Sun, Yijin Guan, Bingjun Xiao, and Jason Cong. 2015. Optimizing FPGA-based accelerator design for deep convolutional neural networks. In *Proceedings of the 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. 161–170.

[39] Jiaxi Zhang, Wentai Zhang, Guojie Luo, Xuechao Wei, Yun Liang, and Jason Cong. 2019. Frequency improvement of systolic array-based CNNs on FPGAs. In *2019 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 1–4.

[40] Yao Fu, Ephrem Wu, and Ashish Sirasao. 2017. 8-bit dot-product acceleration. *Xilinx Inc.: San Jose, CA, USA* (2017).

[41] Xiaofan Zhang, Junsong Wang, Chao Zhu, Yonghua Lin, Jinjun Xiong, Wen-mei Hwu, and Deming Chen. 2018. DNNBuilder: An automated tool for building high-performance DNN hardware accelerators for FPGAs. In *2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 1–8.

[42] Hanchen Ye, Xiaofan Zhang, Zhize Huang, Gengsheng Chen, and Deming Chen. 2020. HybridDNN: A framework for high-performance hybrid DNN accelerator design and implementation. In *2020 57th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 1–6.

[43] Eirikur Agustsson and Radu Timofte. 2017. NTIRE 2017 challenge on single image super-resolution: Dataset and study. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*. 126–135.
[44] Roman Zeyde, Michael Elad, and Matan Protter. 2012. On single image scale-up using sparse-representations. In *Curves and Surfaces: 7th International Conference, Avignon, France, June 24-30, 2010, Revised Selected Papers 7*. Springer, 711–730.
[45] David Martin, Charless Fowlkes, Doron Tal, and Jitendra Malik. 2001. A database of human segmented natural images and its application to evaluating segmentation algorithms and measuring ecological statistics. In *Proceedings Eighth IEEE International Conference on Computer Vision. ICCV 2001*, Vol. 2. IEEE, 416–423.
[46] Jia-Bin Huang, Abhishek Singh, and Narendra Ahuja. 2015. Single image super-resolution from transformed self-exemplars. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 5197–5206.
[47] ([n. d.]). NVIDIA-NX. https://developer.nvidia.com/embedded/jetson-xavier-nx
[48] ([n. d.]). Atlas 200 DK. https://e.huawei.com/hk/products/cloud-computing-dc/atlas/ascend-310
[49] Vinod Kathail. 2020. Xilinx Vitis unified software platform. In *Proceedings of the 2020 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. 173–174.
[50] ([n. d.]). Xilinx DPU. https://www.xilinx.com/products/intellectual-property/dpu.html
[51] Yongwoo Kim, Jae-Seok Choi, and Munchurl Kim. 2018. 2X super-resolution hardware using edge-orientation-based linear mapping for real-time 4K UHD 60 fps video applications. *IEEE Transactions on Circuits and Systems II: Express Briefs* 65, 9 (2018), 1274–1278.
[52] Kaicong Sun, Maurice Koch, Zhe Wang, Slavisa Jovanovic, Hassan Rabah, and Sven Simon. 2021. An FPGA-based residual recurrent neural network for real-time video super-resolution. *IEEE Transactions on Circuits and Systems for Video Technology* 32, 4 (2021), 1739–1750.
[53] Markus Nagel, Rana Ali Amjad, Mart Van Baalen, Christos Louizos, and Tijmen Blankevoort. 2020. Up or down? Adaptive rounding for post-training quantization. In *International Conference on Machine Learning*. PMLR, 7197–7206.
[54] Jiantao Qiu, Jie Wang, Song Yao, Kaiyuan Guo, Boxun Li, Erjin Zhou, Jincheng Yu, Tianqi Tang, Ningyi Xu, Sen Song, Yu Wang, and Huazhong Yang. 2016. Going deeper with embedded FPGA platform for convolutional neural network. In *Proceedings of the 2016 ACM/SIGDA International Symposium on Field Programmable Gate Arrays*. 26–35.
[55] Qingcheng Xiao, Yun Liang, Liqiang Lu, Shengen Yan, and Yu-Wing Tai. 2017. Exploring heterogeneous algorithms for accelerating deep convolutional neural networks on FPGAs. In *Proceedings of the 54th Annual Design Automation Conference 2017*. 1–6.
[56] Yufei Ma, Minkyu Kim, Yu Cao, Sarma Vrudhula, and Jae-sun Seo. 2017. End-to-end scalable FPGA accelerator for deep residual networks. In *2017 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 1–4.
[57] Xiaofeng Hou, Cheng Xu, Jiacheng Liu, Xuehan Tang, Lingyu Sun, Chao Li, and Kwang-Ting Cheng. 2022. Characterizing and understanding end-to-end multi-modal neural networks on GPUs. *IEEE Computer Architecture Letters* 21, 2 (2022), 125–128.

Received 22 April 2023; revised 17 January 2024; accepted 6 March 2024