HDSuper: Algorithm-Hardware Co-design for Light-weight High-quality Super-Resolution Accelerator

Liang Chang, Xin Zhao, Dongqi Fan, Zhicheng Hu and Jun Zhou
University of Electronic Science and Technology of China, Chengdu, China, 611731.

***Abstract*—Super-resolution (SR) networks have been gradually applied to embedded devices with good-quality image reconstruction. However, the hardware performance and power efficiency are limited by a large number of algorithm parameters, computation complexity, and hardware resources, obstructing the development of a high-quality SR accelerator. This paper proposes an end-to-end platform with a lightweight super-resolution network (LSR) and an efficient, high-quality super-resolution architecture HDSuper, to perform algorithm-hardware co-design for the SR accelerator. For algorithm design, we employ depthwise separable convolution and pixelshuffle to reduce network size and computation complexity by considering the hardware constraints. For hardware design, we provide a unified computing core (UCC) combined with an efficient flattening-and-allocation (F-A) mapping strategy to support various operators with high computational utilization. We adopt the patch training method to reduce the external memory access of the hardware architecture. Based on the evaluation, the proposed algorithm achieves high-quality image reconstruction with $37.44dB$ PSNR. Finally, we implement the image reconstruction in FPGA demonstration, achieving high-quality image reconstruction with $2.08W$ power consumption under the lowest hardware resources compared to the state-of-the-art works.**

***Index Terms*—Super-Resolution, Co-design, Efficient Mapping, High-quality Image, FPGA**

I. Introduction

The intelligent embedded devices, including smartphones, autonomous robotics, and drones, are supposed to be small in size and low power consumption due to tightly limited battery life [1] [2]. Typically, the deployment of deep neural networks (DNNs) on hardware is constrained by massive parameters and computations. Consequently, boosting hardware efficiency is desirable in resource- and power-efficiency usage scenarios.

Super-resolution (SR) networks based on deep learning have obtained good reconstruction performance [3]. Various hardware accelerators have demonstrated a good image reconstruction quality and resource utilization for DNN-based SR [4]–[6]. [7] and [5] proposed SR accelerators based on the fast Winograd algorithm to reduce the number of multiplication. For the external memory access (EMA) problem, [8] and [9] designed layer fusion strategies to reduce the EMA by 95% and 94%, respectively. To improve the image reconstruction quality, the BSRA accelerator adopts the pixel attention mechanism, which makes peak signal-to-noise ratio (PSNR) reach $37.18dB$ [10]. However, DNN-based SR networks are large, and sufficient computational resources are required to accelerate the end-to-end network. The limitations of hardware efficiency and resource utilization motivate us to seek innovations on both algorithms and hardware architecture.

In this work, we propose an end-to-end solution to deploy the entire SR algorithm onto a hardware accelerator with both algorithm and hardware innovations. The lightweight hardware-oriented SR algorithm, called LSR, is provided to optimize the deep learning-based network by considering the constraints of embedded devices. We develop a high-quality SR accelerator, namely HDSuper, to improve hardware efficiency and utilization. The contributions of this work are listed as follows:

- We provide an end-to-end platform to perform lightweight SR networks and develop a hardware accelerator. In the lightweight SR network, the depthwise separable convolution and pixelshuffle are adopted to reduce the network size and computation greatly. In addition, we design a lightweight depthwise separable convolution and channel attention block (LDSCA) to achieve a great image reconstruction quality.
- We propose a HDSuper accelerator to meet the application requirement on embedded devices. A unified computing core (UCC) and an efficient flattening-and-allocation (F-A) mapping strategy is developed, supporting the computation of multiple operators with high computational utilization (100% for LSR).
- We demonstrate the end-to-end FPGA platform with optimized LSR algorithm. Based on our patch training mechanism, we can implement the patch inference in hardware to reduce 99.3% external memory accesses. Compared to several state-of-the-art works, HDSuper costs the lowest resources and obtains $37.44dB$ PSNR, which is a light-weight and high-quality SR accelerator.

II. Preliminary and motivation

Super-resolution (SR) refers to reconstructing low-resolution images into high-resolution ones without losing image quality as much as possible. With the development of deep learning, the application of DNNs to SR has significantly improved. For example, typical networks, including SRCNN and FSRCNN, use the deconvolution and interpolation upsampling step to get high-resolution images [3], [11]. However, deconvolution brings massive computation with large memory accesses. To mitigate overhead of the

deconvolution layer, ESPCN [12] developed a sub-pixel convolution, i.e. pixelshuffle, to alternate the deconvolution, avoiding large computation. Recently, the attention mechanism was employed in the SR network. CSNL proposed a cross-scale attention mechanism and pushed SR to a new stage [13]. Although the image reconstruction quality is improving, the number of parameters and required computation of the SR network is also increasing exponentially. For optimizing computation, refer [14] proposed a method to transform a deconvolutional layer into a convolutional layer, which reduces the size of the convolution kernel by compressing invalid computation. Refer [5] proposed deconvolution implementation based on a fast Winograd algorithm to reduce the number of multiplication. External memory access is another severe challenge. Refer [5] cut the input images into multiple blocks and measured their total deviation value, reducing the required on-chip memory capacity. Refer [9] proposed a bandwidth-efficient architecture based on a layer fusion mechanism, which reduces the bandwidth by 98.4%.

Based on the above analyses, the SR accelerator can be further optimized from algorithm and hardware co-design. In this work, we develop an end-to-end SR solution, which includes a high-efficient accelerator with high computational utilization and a hardware-friendly lightweight network to break the computational bottleneck.

III. Hardware-Oriented SR-Algorithm Optimization

A. LSR Algorithm Structure and Flow

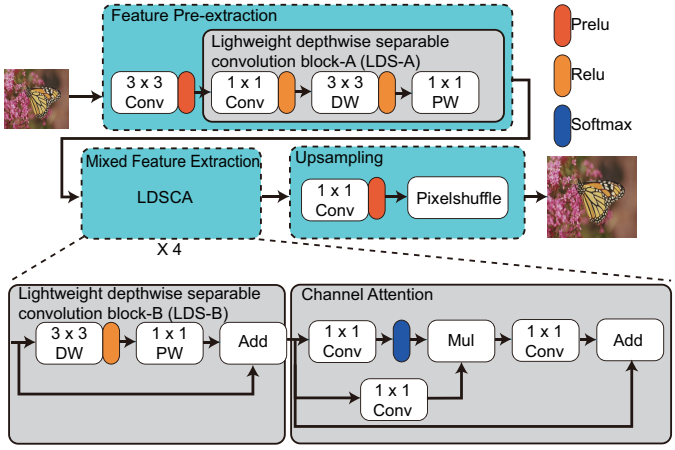

Fig. 1. LSR structure. It contains three parts, i.e., feature pre-extraction, mixed feature extraction, and upsampling.

To improve the reconstructed image quality with a lightweight SR network, we propose the LSR, as shown in Figure 1. We design a lightweight depthwise separable convolution block A (LDS-A) by removing the shortcut in original ones, hence obtain the ability to map high-dimensional feature maps to low-dimensional ones. A Lightweight depthwise separable convolution block B (LDS-B) is proposed by removing the original 1×1 convolution layer and fine-tuning the parameters. In addition, we modify the channel attention (CA) module by removing the 1×1 convolution layer and an element-wise multiplication in non-local blocks. The CA module can be expressed as:

$$z = g_z y + x = g_z(\mathrm{softmax}\left(g_v\left(x\right)\right) g_u\left(x\right)) + x \qquad (1)$$

Where g_z, g_v and g_u represent 1×1 convolution in CA, x represents input and z represents output. In LSR, the LDSCA contains LDS-B and CA. LDS-B extracts low-frequency and high-frequency information of the image, while CA assigns weight to each channel and captures long-distance dependencies between channels.

B. Optimization for Computation Reduction

Depthwise separable convolution (DS) and pixelshuffle are used in LSR to reduce the computation and parameters. DS reduces the computation and parameters by decomposing standard convolution into depthwise convolution (DW) and pointwise convolution (PW). The pixelshuffle is used to replace compute-intensive deconvolution for upsampling.

C. Patch Training for Hardware EMA Optimization

In the training stage, all images in the training set are first divided into multiple small patches, and then all patches are formed into new datasets for LSR training. In backpropagation, we compute the loss between the high-resolution output patch and ground truth patch, and continue to iterate until sound results are obtained. In the evaluation stage, the input low-resolution images are divided into small patches before being put into the network for forward propagation. Finally, we concatenate the output patches into complete images after completing the computation. During the computation, only small capacity memory are needed for one patch. The on-chip inference of hardware is feasible, significantly reducing the EMA and improving the hardware performance.

IV. HDSuper Hardware Architecture

A. Overview of HDSuper Architecture and Computing Flow

Figure 2(a) indicates the HDSuper architecture, containing the unified computing core (UCC), data buffers, and global controller. The UCC consists of a computing unit (CU) array, addition-and-accumulation unit (AAU), and special-processing unit (SPU) arrays. The UCC supports multiple operators with high computational utilization, including convolution, element-wise operation, and activation functions. Data buffers include an input buffer, weight buffer, and on-chip buffer, which can realize the data storage of the feature map and weight. The global controller manages the hardware computing, such as inter-layer switching and data mapping.

The computing of the LSR can be divided into three cases according to data access. ❶ For the first layer of computing, it needs to transmit the patch from off-chip memory. ❷ When computing in middle layers, the output feature map is temporarily saved to the on-chip buffer, which avoids EMA. By controlling the (D)MUXs, the UCC can obtain input from the different buffers, which is flexible. ❸ The last layer is to realize upsampling using pixelshuffle, which does not involve computing. The implementation is integrated into the on-chip buffer by controlling the output fashion of the on-chip buffer.

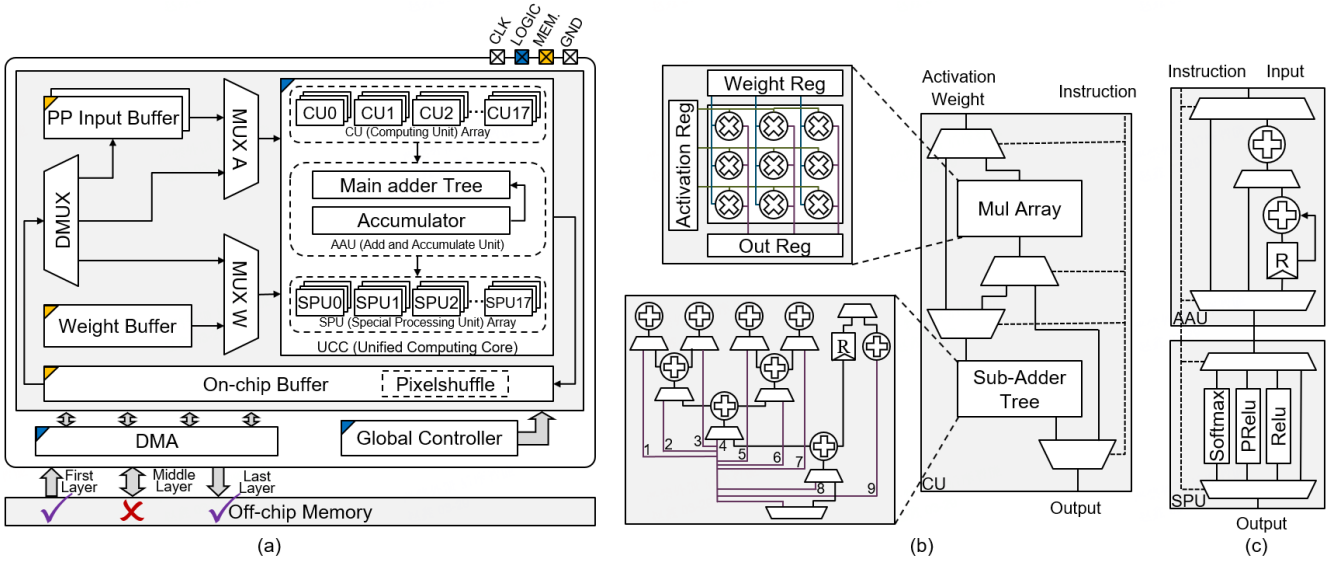

Fig. 2. HDSuper architecture design. (a) Overall HDSuper architecture. (b) CU (computing unit), including a multiplier (MUL) array, a sub-adder tree, and multiple MUXs. (c) AAU and SPU. AAU (addition and accumulation unit) includes an inter-channel adder tree, an accumulator, and multiple MUXs. SPU (special processing unit) includes the modules of softmax, PReLU, and ReLU.

B. Unified Computing Core Design

The LSR network contains different operators and multiple branches. The UCC can support various operators flexibly. As shown in Figure 2(a), it consists of three groups of CU blocks, each of which has 18 CUs. Therefore, in order to improve the computing efficiency, we can simultaneously compute three feature maps, that is, the batch size is 3. Figure 2(b) and (c) shows the components of the UCC, mainly including CU, AAU, and SPU. The main logic modules in CU are the multiplier (MUL) array and sub-adder tree. The MUL array can achieve multiplications with size 3×3. The sub-adder tree can support the accumulation of 9 data or the element-wise addition of 9 pairs. The specific working state of the sub-adder tree is selected according to the MUX condition. In addition, by controlling (D)MUXs in CU, different computing modes can be achieved, including 3×3 convolution, 3×3 element-wise multiplication, and 3×3 element-wise addition. Compared with the traditional convolution unit, CU only introduces a few MUXs with negligible overhead to efficiently support various operators.

AAU is mainly composed of an inter-channel adder tree and accumulator. Among them, the inter-channel adder tree supports the addition of feature maps among input channels. The accumulator accumulates the output from the inter-channel adder tree and temporarily stores it in the accumulation buffer until all input channels are accumulated. Several layers without inter-layer addition/accumulation can be skipped via (D)MUXs. In LSR, some convolution layers are followed by nonlinear function. The SPU can support the nonlinear functions such as softmax, PReLU, and ReLU, selected by MUX. In addition, since the SR network is an upsampling process, there is no support for pooling operations such as max pooling because they belong to the downsampling process and generally do not appear in the SR network.

C. Efficient F-A Mapping

To support more network structures and operator types with high computational utilization, we propose an efficient mapping strategy including two steps: flattening and allocation (F-A mapping).

1) Flattening: Flattening refers to transforming a multi-branch network structure into a single-branch ones for subsequent allocation. The basic principle is to take the network layer containing convolution or element-wise operation as a separate layer. The following nonlinear function belongs to the current convolution layer. In addition, the intermediate output of the multi-branch structure is temporarily stored in the on-chip buffer. Take the LDSCA block as an example, Figure 3 contains five convolution layers and three element-wise multiplication or addition layers with three branches, which are flattened into a single-branch 8-layer network.

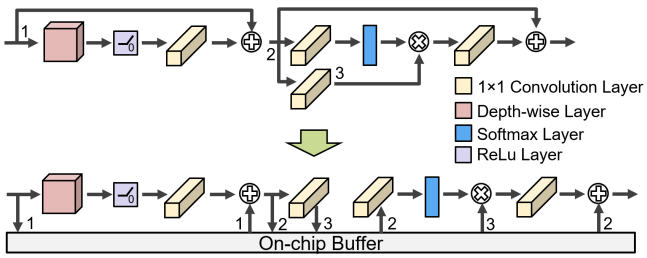

Fig. 3. Flattening for LDSCA block. The number represents the intermediate result of the multi-branch that is temporarily stored in the on-chip buffer.

2) Allocation: For data allocation, it refers to allocating the flattened single-branch network to the UCC for computing. According to the different parameter configurations of each network layer, there are different allocation methods. Algorithm 1 shows the specific allocation strategy. The goal is to find the mapping method with the maximum computational utilization for each layer. We set batch size as three. Three patches are calculated in the UCC. The inputs are operator type

(OP), kernel size (KS), the number of input channels (CN_{in}), and the number of output channels (CN_{out}), respectively. The outputs are data flow (DF), input channel parallelism (P_{in}), input reload times (R_{in}^{times}), output channel parallelism (P_{out}), and output reload times (R_{out}^{times}), respectively. Reload times mean loading the input data several times to complete all computations. The allocation can be classified into three cases according to the operators.

Algorithm 1 Data allocation. The goal is to obtain the allocation with maximum computational utilization.

Input: (OP, KS), CN_{in}, CN_{out}
Output: DF, (P_{in}, R_{in}^{times}), (P_{out}, R_{out}^{times})

if (OP == convolution)
  if (KS == 1) ▷ case 1
    DF = HWC
    if (CN_{in} >9)
      P_{in} = 2, R_{in}^{times} = $\lceil CN_{in} / (9\times2) \rceil$
    else
      P_{in} = 1, R_{in}^{times} = 1
    P_{out} = 18 / P_{in}, R_{out}^{times} = $\lceil CN_{out} / P_{out} \rceil$
  else ▷ case 2
    DF = CHW
    P_{num} = $\lfloor 18 / (\lceil KS\times KS / 9 \rceil) \rfloor$
    P_{in}, P_{out} = find_optim_parall(P_{num})
    R_{in}^{times} = $\lceil CN_{in} / P_{in} \rceil$, R_{out}^{times} = $\lceil CN_{out} / P_{out} \rceil$
else if (OP == element-wise multiplication or add)
  DF = HWC ▷ case 3
  P_{in} = 1, R_{in}^{times} = 1
  P_{out} = 18, R_{out}^{times} = $\lceil CN_{out} / 18 \rceil$
return DF, (P_{in}, R_{in}^{times}), (P_{out}, R_{out}^{times})

Case 1: For 1×1 convolution, the data flow is the height-width-channel (HWC), supporting nine input channels data to fill a CU with a computational utilization of 100%. Both the P_{in} and R_{in}^{times} are set to 1. Correspondingly, the P_{out} is 18, and the R_{out}^{times} is $\lceil CN_{out} / 18 \rceil$. On the other hand, if the number of input channels CN_{in} is larger than 9, we set the P_{in} to two limited by the on-chip buffer bandwidth (18 data per cycle). The data from 18 input channels are read from the on-chip buffer simultaneously.

Case 2: When the kernel size is 3×3, 5×5, 7×7 or 9×9, the data flow is channel-height-width (CHW). Firstly, we compute the maximum parallel computing (P_{num}) that can be supported. The 3×3 convolution can fill a CU, so the P_{num} is 18. The 5×5 convolution requires 3 CU units to satisfy the computation, and the P_{num} is 6. In addition, the 7×7 and 9×9 convolution require 6 and 9 CU units to satisfy the computation, respectively, and the P_{num} is 3 or 2. Then, find out the optimal input and output channel parallelism (find_optim_parall(P_{num})). The basic idea is to factor the number of input channels and output channels to find two factors, whose product is exactly equal to the P_{num}. On the contrary, we analyze the corresponding computational utilization under different factor combinations to select the P_{in} and P_{out} factors with largest utilization.

Case 3: For element-wise addition or multiplication, since the number of input channels is one, the P_{in} is one. Correspondingly, P_{out} and R_{out}^{times} are 18 and $\lceil CN_{out} / 18 \rceil$, respectively.

3) Case study for LSR: Figure 4 indicates the data allocation in LSR, the computing of convolution can be summarized into three cases. ❶ Case for 3×3 convolution: P_{in} = 1, P_{out}=18. ❷ Case for 1×1 convolution and PW: P_{in} = 2, P_{out} = 9. ❸ Case for 3×3 DW: P_{in} = 18, P_{out} = 1. In above each case, the computational utilization is up to 100%.

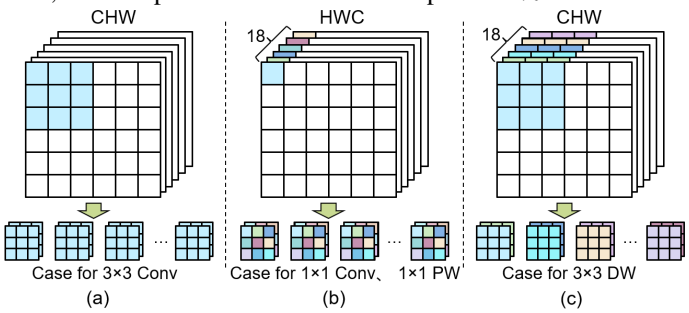

Fig. 4. LSR data allocation. (a) For 3×3 convolution, the data flow is CHW. (b) For 1×1 convolution and PW, the data flow is HWC. (c) For 3×3 DW, the data flow is CHW.

D. EMA and Bandwidth Optimization

To reduce the data access with off-chip memory, we adopt the patch inference based on the patch training strategy. Only one patch of the whole image is calculated each time. For each patch, the size of the output feature map generated by the middle layer is tiny, which enables on-chip storage. In this way, only the first and last layers must interact with the off-chip memory, significantly reducing the EMA.

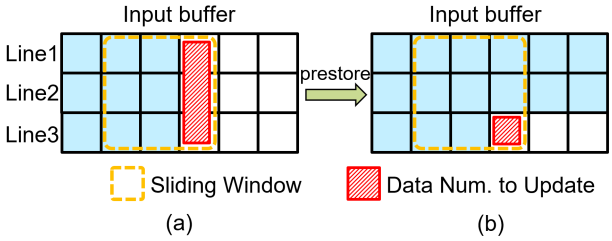

Fig. 5. Optimal design for input buffer. (a) Three data need to be updated each cycle. (b) Only one data need to be updated each cycle.

In addition, we can optimize the input buffer to save bandwidth, and take 3×3 convolution as an example. The 3×3 convolution requires 9 input feature map data with bandwidth 9 data per cycle. Two optimization steps are adopted. Firstly, we develop the sliding window strategy to update only three data each cycle, as shown in 5(a). The required bandwidth can be reduced 66.7%. Secondly, the pre-store mechanism is developed to store two rows of input feature map in advance, as shown in Figure 5(b). The required bandwidth can be further reduced by 88.9%.

V. Experiment Results and Discussion

A. Experiment Setup

We train the designed LSR network under patched DIV2k dataset where each image is segmented. Then, the trained

network is quantized for FPGA deployment. In the hardware implementation, we use Verilog HDL to implement the HDSuper based on the Vivado 2021.2 development environment. In addition, we design the corresponding PC test system to facilitate the demonstration, as shown in Figure 8.

B. Patch Training Analyses

To evaluate the impact of patch size on image reconstruction quality, we use Set5 and Set14 datasets for experiments. As shown in Figure 6, patch sizes with 84×84, 42×42, 36×36, and 28×28 are adopted, respectively. From the results, there is little influence on PSNR with different patch sizes, which proves that patch training method adopted by LSR is effective.

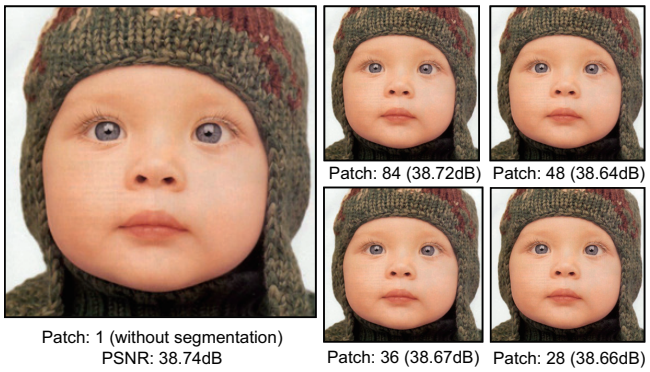

Fig. 6. Patch training comparison. We segment the image into multiple patch sizes to compare the PSNR loss.

C. Algorithm Model Comparison

Table I demonstrates the algorithm comparison results, and all models are tested on the y channel with the same input image size. The proposed LSR has better PSNR and SSIM than SRCNN and FSRCNN, slightly lower than VDSR. Nevertheless, LSR consumes much lower MACs operations and has less network parameters.

TABLE I
COMPARISON OF THE ALGORITHM.

Method	Scale	Parameter	MAC (GOPS)	Set5		Urban100	
				PSNR	SSIM	PSNR	SSIM
SRCNN [11]	2	8.1k	66.62	36.66	0.9542	29.50	0.8946
	3		149.89	32.75	0.9090	26.24	0.7989
	4		266.48	30.48	0.8628	24.52	0.7221
FSRCNN [3]	2	12.5k	25.84	37.05	0.9560	29.88	0.9020
	3			33.18	0.9140	26.43	0.8080
	4			30.72	0.8660	24.62	0.7280
VDSR [15]	2	665k	5513	37.53	0.9587	30.76	0.9140
	3		12405	33.66	0.9213	27.14	0.8279
	4		22053	31.35	0.8838	25.18	0.7524
LSR (Ours)	2	8.9k	18.90	37.44	0.9592	30.29	0.9088
	3	9.0k	19.09	33.43	0.9210	27.78	0.8425
	4	9.1k	19.35	31.09	0.8798	24.89	0.7443

* Input image: 1080p. Blue: best result. Green: second best result.

D. Utilization and EMA Analysis

Through LSR case study in Section IV-C3, the computational utilization is up 100%. In addition, HDSuper can be adapted to other super-resolution networks. For example, by flattening and allocating the widely used FSRCNN network, the actual average computational utilization can reach 95.8%. Figure 7 shows the comparison on EMA. The EMA of HDSuper is reduced by 99.3%, 4.2% higher than the state-of-the-art work, thanks to the patch inference mechanism.

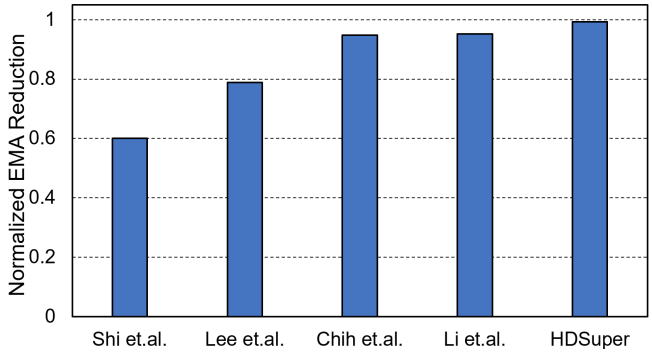

Fig. 7. Comparison of EMA. The above four works, Shi et.al., Lee et.al., Chih et.al. and Li et.al. are from references [5], [17], [9] and [8], respectively.

E. Architecture Comparison

The HDSuper is compared to other SR hardware architectures, as shown in Table II. The HDSuper uses a lightweight network LSR and has a higher PSNR in hardware implementation. Specifically, HDSuper can achieve $37.44dB$ PSNR under a scaling factor of 2 in Set5 datasets, which is much higher than other architectures. In addition, HDSuper supports multiple scaling factors, including 2, 3, and 4. HDSuper has lower power consumption and less resource overhead, with only $2.08W$ power consumption and less than half resources compared to [6]. The memory resources are also less than in other works, thanks to the patch inference mechanism of this work. To meet the real-time requirement of the embedded system, the frame rate of HDSuper inference can reach $81fps$ by considering the limited computing resources and strict power budget. In the HDSuper demonstration, we only use a small amount of DSP resources, which is only 38% of [6]. Figure 8 shows the implementation of our SR system.

F. Co-design Discussion

The hardware can achieve higher efficiency through the algorithm and hardware co-design. For calculation optimization, the algorithm replaces the standard convolution with the depthwise separable convolution can reduce the computation with only slightly precision loss. On the other hand, the patch training mechanism is employed to significantly reduce EMA, making the hardware support patch inference.

VI. CONCLUSION

Hardware resources and power budget constrain embedded devices. The SR algorithm with an large network and complex computation is challenging to implement the efficient hardware accelerator. In this work, we co-design the hardware architecture and SR algorithm, which is a hardware-friendly lightweight network named LSR with smaller network size and higher image reconstruction quality. In addition, we develop

TABLE II
COMPARISON OF DIFFERENT HARDWARE ARCHITECTURE

Work	Kim et.al. [4]	Chang et.al. [16]	Shi et.al. [5]	Sun et.al. [6]	HDSuper (Proposed)
SR Method	CNN-based	FSRCNN-based	FSRCNN-s-based	RNN-based	LSR
Supported Scales	2	2, 3, 4	2	2	2, 3, 4
Technology	Xilinx XCKU040	Xilinx XC7K410T	Xilinx ZCU102	Xilinx XCKU15P	Xilinx XC7VX485T
Frequency	150 MHz	130 MHz	200 MHz	160 MHz	200 MHz
Power	5.69 W	5.40 W	-	5.47 W	2.08 W
Hardware Resources	LUT:151 K, REG:121 K, DSP:1,920	LUT:167 K, REG:158 K, DSP:1,512	LUT:173 K, REG: - K, DSP:746	LUT:98 K, REG:57 K, DSP:1,820	LUT:73 K, REG:53 K, DSP:704
Memory Size	194 KB	945 KB	1,396 KB	4,842 KB	496 KB
PSNR*	36.51 dB	36.40 dB	-	36.76 dB	37.44 dB
Frame Rate	60 fps	62.7 fps	120.4 fps	76 fps	81 fps

* All PSNR is measured in dataset Set5 with the scaling factor 2.

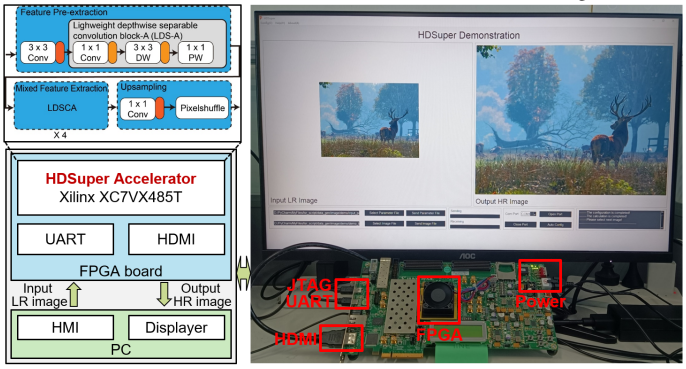

Fig. 8. Demonstration of super resolution system.

an efficient hardware architecture HDSuper to support various operators with high computational utilization. On the FPGA platform, the HDSuper achieves a real-time image reconstruction with nearly 100% computational utilization and only $2.08W$ power consumption. In future work, we can further improve the universality of the HDSuper architecture.

ACKNOWLEDGMENT

This work was supported by the NSAF under Grant NO. U2030204, the National Natural Science Foundation of China under Grant No. 62104025 and State Key Laboratory of Computer Architecture (ICT, CAS) under Grant No. CARCHB202117. Jun Zhou is the corresponding author.

REFERENCES

[1] Y. Yang, Q. Huang, B. Wu, T. Zhang, L. Ma, G. Gambardella, M. Blott, L. Lavagno, K. Vissers, J. Wawrzynek *et al.*, "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*, 2019, pp. 23–32.

[2] M. Mettler, M. Rapp, H. Khdr, D. Mueller-Gritschneder, J. Henkel, and U. Schlichtmann, "An fpga-based approach to evaluate thermal and resource management strategies of many-core processors," *ACM Transactions on Architecture and Code Optimization (TACO)*, vol. 19, no. 3, pp. 1–24, 2022.

[3] C. Dong, C. C. Loy, and X. Tang, "Accelerating the super-resolution convolutional neural network," in *European conference on computer vision*. Springer, 2016, pp. 391–407.

[4] Y. Kim, J.-S. Choi, and M. Kim, "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*, vol. 29, no. 8, pp. 2521–2534, 2018.

[5] B. Shi, Z. Tang, G. Luo, and M. Jiang, "Winograd-based real-time super-resolution system on fpga," in *2019 International Conference on Field-Programmable Technology (ICFPT)*. Tianjin: IEEE, 2019, pp. 423–426.

[6] K. Sun, M. Koch, Z. Wang, S. Jovanovic, H. Rabah, and S. Simon, "An fpga-based residual recurrent neural network for real-time video super-resolution," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 32, no. 4, pp. 1739–1750, 2022.

[7] P.-W. Yen, Y.-S. Lin, C.-Y. Chang, and S.-Y. Chien, "Real-time super resolution cnn accelerator with constant kernel size winograd convolution," in *2020 2nd IEEE International Conference on Artificial Intelligence Circuits and Systems (AICAS)*. IEEE, 2020, pp. 193–197.

[8] Z. Li, S. Kim, D. Im, D. Han, and H.-J. Yoo, "An 0.92 mj/frame high-quality fhd super-resolution mobile accelerator soc with hybrid-precision and energy-efficient cache," in *2022 IEEE Custom Integrated Circuits Conference (CICC)*. IEEE, 2022, pp. 1–2.

[9] C.-Y. Chih, S.-S. Wu, J. P. Klopp, and L.-G. Chen, "Accurate and bandwidth efficient architecture for cnn-based full-hd super-resolution," in *2018 IEEE International Symposium on Circuits and Systems (ISCAS)*. Florence: IEEE, 2018, pp. 1–5.

[10] D.-H. Yang and T.-S. Chang, "Bsra: Block-based super resolution accelerator with hardware efficient pixel attention," in *IEEE International Symposium on Circuits and Systems (ISCAS)*, 2022.

[11] C. Dong, C. C. Loy, K. He, and X. Tang, "Learning a deep convolutional network for image super-resolution," in *European conference on computer vision*. Springer, 2014, pp. 184–199.

[12] W. Shi, J. Caballero, F. Huszár, J. Totz, A. P. Aitken, R. Bishop, D. Rueckert, and Z. Wang, "Real-time single image and video super-resolution using an efficient sub-pixel convolutional neural network," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 1874–1883.

[13] Y. Mei, Y. Fan, Y. Zhou, L. Huang, T. S. Huang, and H. Shi, "Image super-resolution with cross-scale non-local attention and exhaustive self-exemplars mining," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2020, pp. 5690–5699.

[14] J.-W. Chang and S.-J. Kang, "Optimizing fpga-based convolutional neural networks accelerator for image super-resolution," in *2018 23rd Asia and South Pacific Design Automation Conference (ASP-DAC)*. Jeju: IEEE, 2018, pp. 343–348.

[15] J. Kim, J. K. Lee, and K. M. Lee, "Accurate image super-resolution using very deep convolutional networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 1646–1654.

[16] J.-W. Chang, K.-W. Kang, and S.-J. Kang, "An energy-efficient fpga-based deconvolutional neural networks accelerator for single image super-resolution," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 30, no. 1, pp. 281–295, 2018.

[17] J. Lee, J. Lee, and H.-J. Yoo, "Srnpu: An energy-efficient cnn-based super-resolution processor with tile-based selective super-resolution in mobile devices," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 10, no. 3, pp. 320–334, 2020.