FPGA Codec System of Learned Image Compression With Algorithm-Architecture Co-Optimization

Heming Sun, *Member, IEEE*, Qingyang Yi, and Masahiro Fujita, *Life Member, IEEE*

***Abstract*— Learned Image Compression (LIC) has shown a coding ability competitive to traditional standards. To address the complexity issue of LIC, various hardware accelerators are required. As one category of accelerators, FPGA has been used because of its good reconfigurability and high power efficiency. However, the prior work developed the algorithm of LIC neural network at first, and then proposed an associated FPGA hardware. This separate manner of algorithm and architecture development can easily cause a layout problem such as routing congestion when the hardware utilization is high. To mitigate this problem, this paper gives an algorithm-architecture co-optimization of LIC. We first restrict the input and output channel parallelism with some constraints to ease the routing issue with more DSP usage. After that, we adjust the numbers of channels to increase the DSP efficiency. As a result, compared with one recent work with a fine-grained pipelined architecture, we can reach up to 1.5x faster throughput with almost the same coding performance on the Kodak dataset. Compared with another recent work accelerated by AMD/Xilinx DPU, we can reach faster throughput with better coding performance.**

***Index Terms*— Learned image compression, FPGA, co-optimization.**

I. Introduction

NEURAL network-based Learned Image Compression (LIC) has been developed significantly in the past few years. Starting from [1], the mainstream of LIC network is based on the hyperprior structure which is composed of main path and hyper path. The main path is used to transform the original image to latent nodes. The hyper path is used to generate the prior distribution of the latent nodes for the entropy coding. By using a more sophisticated network in the main path and enhancing the prediction accuracy of the entropy model in the hyper path, recent LIC [2], [3] can outperform VVC [4] performance in intra coding configuration by more than 10% in terms of BD-rate.

As one main track of LIC development, most paper [5], [6], [7], [8], [9], [10] aim at increasing the rate-distortion performance, and they are mainly implemented on machine learning library with floating-point arithmetic. However, the floating-point error in various hardware will cause the cross-platform coding problem as mentioned first in [11]. To solve this problem, there have been a few quantized framework [12], [13], [14], [15] with fixed-point arithmetic, and they can ensure the bit-wise hardware-independent computation consistency.

After building the network with fixed point arithmetic, a specific hardware accelerator is required for the acceleration. Among various hardware platforms, Field Programmable Gate Array (FPGA) has the advantage of high hardware utilization and power efficiency compared with CPU and GPU as reported in [16] and [17]. In addition, compared with ASIC, FPGA is more flexible and reconfigurable which can keep up with the fast developments of neural network models. As a result, FPGA-based accelerator has been explored in much literature.

As described in [18], FPGA accelerators can be mainly categorized into two types: generic [19], [20], [21] and pipeline [22], [23]. For the generic architecture, it consists of a general processing element array. Each layer is mapped to the general processor in a sequential order. For the pipeline architecture, we will allocate hardware resources (e.g. BRAM, DSP) to various layers, and all the layers are processed simultaneously. However, both categories have its own drawbacks. For the generic architecture, due to a general architecture with a fixed computation-to-communication (CTC) ability, it is not suitable for processing the layers with extremely different CTC ratios. For the pipeline architecture, because of a coarse granularity of resource allocation for different layers, the overall resource utilization is low. For LIC, [24] utilized the AMD/Xilinx Vitis AI compiler to create an xmodel and then deploy on AMD/Xilinx IP Deep Learning Processor Unit (DPU) for the processing. Considering that CTC ratio of various layers are quite different as shown in Section II-B, pipeline architecture is adopted in the previous work [25].

When using FPGA as the neural network accelerator, DSP [26] is the key component to deal with the multiply-accumulate operation. Though theoretically the proposed work [25] can be scalable to any DSP usages because of the fine-grained granularity, the actual DSP utilization

Manuscript received 5 January 2024; revised 7 March 2024; accepted 30 March 2024. Date of publication 8 April 2024; date of current version 27 June 2024. This work was supported in part by JSPS KAKENHI Grant Number JP23K16861, in part by the Telecommunications Advancement Foundation, and in part by Yazaki Memorial Foundation for Science and Technology. This article was recommended by Guest Editor Z. Chen. *(Corresponding author: Heming Sun.)*

Heming Sun is with the Faculty of Engineering, Yokohama National University, Kanagawa 240-8501, Japan (e-mail: sun-heming-vg@ynu.ac.jp).

Qingyang Yi was with the Department of Electrical Engineering and Information Systems, School of Engineering, The University of Tokyo, Tokyo 113-8654, Japan. He is now with Huawei Technologies Company Ltd., Shenzhen 518129, China (e-mail: yqymasteryi@gmail.com).

Masahiro Fujita is with the System Design Research Center, School of Engineering, The University of Tokyo, Tokyo 113-8654, Japan (e-mail: fujita@ee.t.u-tokyo.ac.jp).

Color versions of one or more figures in this article are available at https://doi.org/10.1109/JETCAS.2024.3386328.

Digital Object Identifier 10.1109/JETCAS.2024.3386328

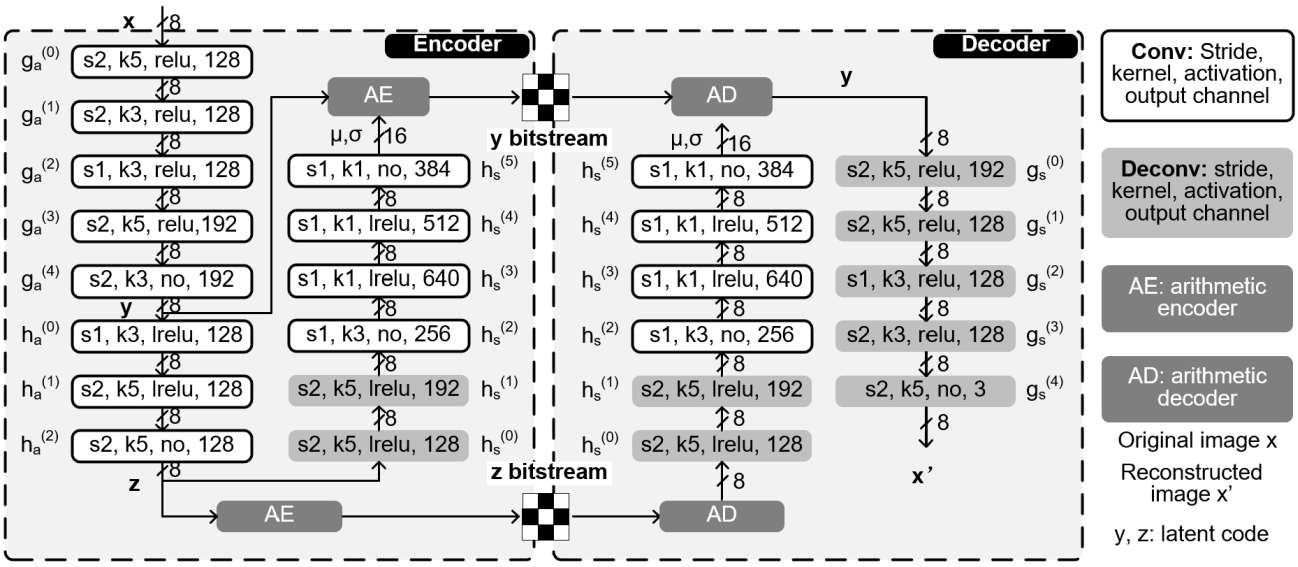


Fig. 1. Encoder and decoder diagram. The output channels of first four layers (M0, M1, M2, M3) of g_a and the first four layers (N0, N1, N2, N3) of g_s are decided by the network search. This network is marked as *hyper-small* in the experiments.

is sometimes low due to routing congestion. When using AMD/Xilinx Kintex UltraScale FPGA acceleration board KU115 for encoding 720P, only 2654 out of 5520 DSP can be utilized in [25]. To increase the actual DSP utilization (the number of DSP actually being used) and efficiency as well, we develop an FPGA codec of LIC based on algorithm-architecture co-optimization. The detail contributions are listed in the below.

- We improve the DSP utilization by adding the constraint to the channel parallelism. By reducing the Lookup Table (LUT) of the activation buffer which is one of the bottlenecks in the place and route phase, more DSPs can be used without routing error.
- We improve the DSP efficiency by neural network search. By adjusting the overall channel of each layer, allocated DSP can work with higher efficiency (i.e. fewer clock cycle bubbles).
- We evaluate the performance on three FPGA boards with different scales. Compared with the recent work [25], we can reach at most 1.5x faster coding throughput. We also evaluate the coding efficiency for the searched network and ensure that the network searching will not degrade the coding performance in our experiments.

II. Preliminary Algorithm and Architecture

A. Learned Image Compression Framework

In this work, we build the baseline network in [25] as shown in Fig. 1. There are five layers in the main analysis transform g_a and synthesis transform g_s respectively as described in the following equation

$$\boldsymbol{y} = g_a(\boldsymbol{x}) = g_a^{(4)} \circ g_a^{(3)} \circ g_a^{(2)} \circ g_a^{(1)} \circ g_a^{(0)}(\boldsymbol{x}) \quad (1)$$

$$\hat{\boldsymbol{x}} = g_s(\hat{\boldsymbol{y}}) = g_s^{(4)} \circ g_s^{(3)} \circ g_s^{(2)} \circ g_s^{(1)} \circ g_s^{(0)}(\hat{\boldsymbol{y}}) \quad (2)$$

where $\boldsymbol{x}$ is the original image, $\hat{\boldsymbol{x}}$ is the reconstructed image, $\boldsymbol{y}(\hat{\boldsymbol{y}})$ is the (quantized) latent. Operation $\circ$ represents function composition.

For the hyper path, there are three layers in the analysis transform h_a and six layer in the synthesis transform h_s as below

$$\boldsymbol{z} = h_a(\hat{\boldsymbol{y}}) = h_a^{(2)} \circ h_a^{(1)} \circ h_a^{(0)}(\hat{\boldsymbol{y}}) \quad (3)$$

$$\boldsymbol{\mu_y}, \boldsymbol{\sigma_y} = h_s(\hat{\boldsymbol{z}}) = h_s^{(5)} \circ h_s^{(4)} \circ h_s^{(3)} \circ h_s^{(2)} \circ h_s^{(1)} \circ h_s^{(0)}(\hat{\boldsymbol{z}}) \quad (4)$$

where $\boldsymbol{z}(\hat{\boldsymbol{z}})$ is the (quantized) latent of the hyper path and $\boldsymbol{\mu_y}, \boldsymbol{\sigma_y}$ are the parameters to build prior of $\boldsymbol{y}$. In this paper, we assume $\boldsymbol{y}$ follows the single Gaussian distribution, thus the prior p_y is as below

$$p_{\hat{y}|\hat{z}}(\hat{y}|\hat{z}) = \prod_i \left(\mathcal{N}(\mu_{y_i}, \sigma_{y_i}^2) * \mathcal{U}(-\frac{1}{2}, \frac{1}{2})\right)(\hat{y}_i) \quad (5)$$

where i is the index of each sample of $\boldsymbol{y}$, $\mathcal{N}$ represents the Gaussian distribution, and a convolution with standard uniform density function $\mathcal{U}(-\frac{1}{2}, \frac{1}{2})$ is performed to make the prior more flexible to fit various shape of posterior.

For $\boldsymbol{z}$, similar as previous work [1], we model the prior p_z through a non-parametric factorized model $\boldsymbol{\psi}$ as below

$$p_{\hat{z}|\psi}(\hat{z}|\psi) = \prod_i \left(p_{z_i|\psi^{(i)}}(\psi^{(i)}) * \mathcal{U}(-\frac{1}{2}, \frac{1}{2})\right)(\hat{z}_i) \quad (6)$$

where i is the index of each sample of $\boldsymbol{z}$.

Then, we can build the loss function based on the traditional rate-distortion cost where rate is composed of $\boldsymbol{y}$ and $\boldsymbol{z}$.

$$\begin{aligned} J &= R + \lambda \cdot D \\ &= \mathbb{E}[-log_2(p_{\hat{\boldsymbol{y}}}(\hat{\boldsymbol{y}}))] + \mathbb{E}[-log_2(p_{\hat{\boldsymbol{z}}}(\hat{\boldsymbol{z}}))] + \lambda \cdot D(\boldsymbol{x}, \hat{\boldsymbol{x}}) \end{aligned} \quad (7)$$

where R is rate, D is distortion and λ is used to balance the weight of rate and distortion. J represents the cost metric of rate-distortion optimization [27].

For each layer, we quantize the weights and activations to 8-bit except $\boldsymbol{\mu_y}$ and $\boldsymbol{\sigma_y}$. The prior parameters are quantized to 16-bit for the sake of high precision. About the quantization

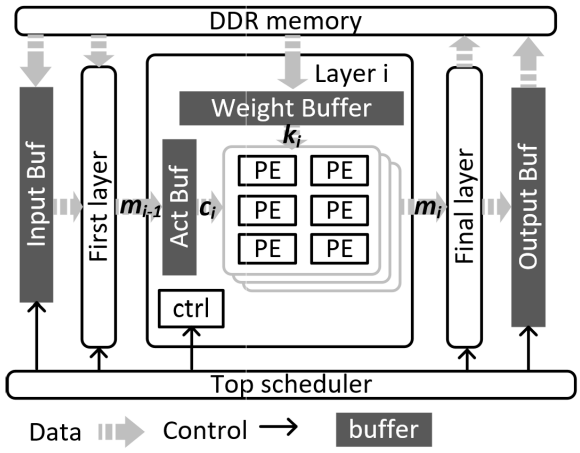


Fig. 2. Overall architecture.

group scheme, we follow the previous work [12] and adopt the channel-wise quantization.

For the activation function, ReLU and Leaky-ReLU rather than the generalized divisive normalization (GDN) [5] is used to ease the hardware implementation. The slope of Leaky-ReLU is set as 0.125 which can be operated as right shifting.

B. Pipelined FPGA Architecture

Fig. 2 shows the overall architecture. Input buffer is used to load the input images from DDR memory and output buffer is used to send the final results to the DDR memory. Each layer contains an activation buffer, a weight buffer and processing elements (PE) array. The activation buffer is in charge of buffering the calculation results of previous layer, weight buffer is used to load the required weights from DDR, and PE is in charge of multiplications and additions. We exploit hardware parallelism from three dimension: kernel width parallelism, input channel parallelism c and output channel parallelism m. After traversing all the input channels C, output channels M and kernel height, one output element is generated.

All the layers in Fig. 1 are processed in a pipeline manner. Each layer is processed row by row, and the processing row index is given in the cell of Fig. 3. We use g_a as an instance to explain the pipeline scheduling in the below. h_a, h_s and g_s in Fig. 1 are processed in a similar way.

First of all, to align the row processing of different layers, a maximum cycle budget is set for each layer. The budget is actually inversely proportional to the output height. For instance, the stride of $g_a^{(1)}$ is two, thus the height of output activation for $g_a^{(1)}$ is half of that for $g_a^{(0)}$, which means that we need to process two rows of $g_a^{(0)}$ while only one row of $g_a^{(1)}$. As a result, given that the budget is T for the first layer, it is $2 \times T$ for second layer. Similarly, the budget is $2 \times T$ for the third layer, $4 \times T$ for the fourth layer and $8 \times T$ for the final layer, as shown in Fig. 3.

Then we explain the startup latency of each layer. For the first layer $g_a^{(0)}$, the kernel size is five, thus two zero rows will be padded at the top in the case of zero padding. Therefore, the processing can start after loading three rows of input image from external memory. For the second layer $g_a^{(1)}$, the kernel size is three, thus one zero row will be padded at the top. Therefore, the processing can start after two rows of $g_a^{(0)}$ are generated. For the remaining layers, the mechanism is similar and the computation will start after required rows are produced from the previous layer. Finally, the output of $g_a^{(4)}$ (y in Fig. 1) will be sent to host CPU for the arithmetic encoding.

As shown in Fig. 1, there are 14 layers for the encoding and 11 layers for the decoding. We calculate CTC of each layer by the following equation

$$\text{CTC} = \frac{\text{MAC}}{\text{Bandwidth}} \tag{8}$$

where MAC is the number of required multiplications and Bandwidth is the required bandwidth for loading weights. The results of various layers are shown in Fig. 4. Main path owns larger CTC compared to hyper path. The fluctuation of CTC leads to the usage of pipeline category.

TABLE I
NOTATION

Notation	Meaning
H_i/W_i	Height/width of output activation for the i-th layer
C_i/M_i	Input/output channels for the i-th layer
c_i/m_i	Input/output channel parallelism for the i-th layer
k_i	Kernel size for the i-th layer
R_i	Kernel reuse ratio for the i-th layer
D_i	Number of multipliers operated in one DSP for the i-th layer

III. Algorithm-Architecture Co-Optimization

A. Channel Parallelism Selection

For the i-th layer, the number of provided multipliers can be calculated as

$$mul_i = k_i \times c_i \times m_i \tag{9}$$

where the meaning of c, m and k are shown in Table I. Thus, the number of clock cycles for generating one row of output activation is in the below

$$T_{ri} = W_i \times k_i \times \lceil \frac{C_i}{c_i} \rceil \times \lceil \frac{M_i}{m_i} \rceil \tag{10}$$

where the meaning of C, M and W are shown in Table I. The number of clock cycles for generating one frame of output activation can be obtained as below.

$$T_{fi} = H_i \times T_{ri} \tag{11}$$

Considering that the overall performance will be determined by the slowest layer, the maximum cycle budget to process one frame for each layer is set as below.

$$T_{f_{max}} = \max_{i \in N} \{T_{fi}\} \tag{12}$$

The frame per second (FPS) can be calculated as below

$$\begin{aligned} \text{FPS} &= \frac{\text{Frequency}}{T_{f_{max}}} \\ &= \frac{\text{Frequency}}{\max\limits_{i \in N} \{H_i \times W_i \times K_i \times \lceil \frac{C_i}{c_i} \rceil \times \lceil \frac{M_i}{m_i} \rceil\}} \end{aligned} \tag{13}$$

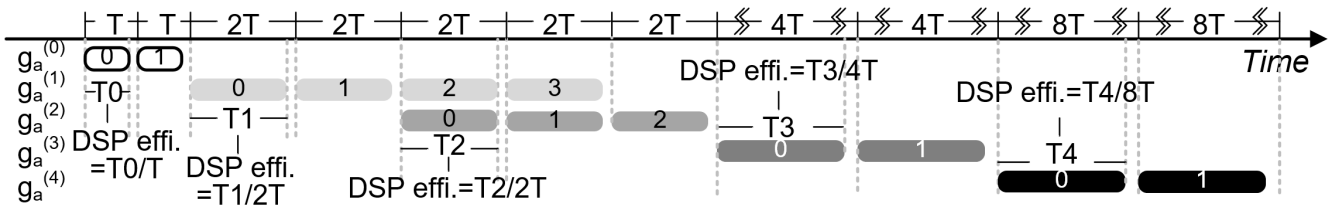


Fig. 3. Pipeline example for five layers of g_a in the encoder.

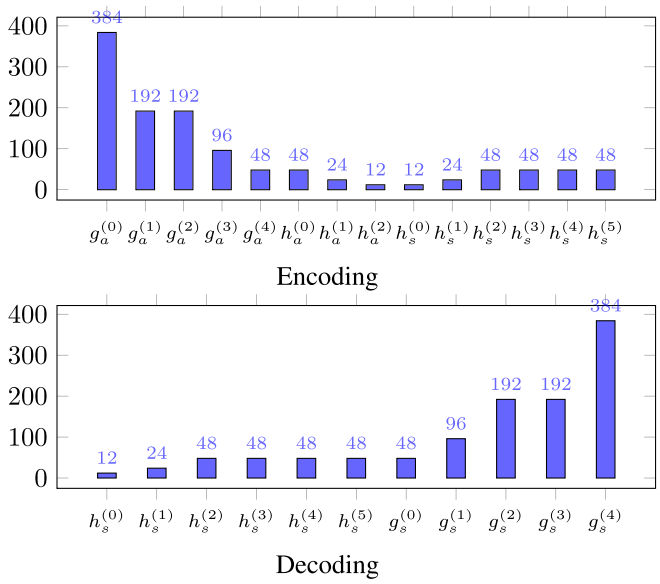


Fig. 4. CTC of various layers.

Algorithm 1 Selection Algorithm for $\boldsymbol{c}$ and $\boldsymbol{m}$, Temporary State Under Optimization Is Labeled With $\tilde{tilde}$

Initialize $\boldsymbol{c} = \vec{1}$ and $\boldsymbol{m} = \vec{D}$ for all the layers
Calculate $\boldsymbol{mul}$ and $\boldsymbol{T_f}$ for all the layers
while (1) **do**
  Find the layer (e.g. j-th layer) with the maximum T_{fj} among all the layers
  $\tilde{\boldsymbol{c}}, \tilde{\boldsymbol{m}} \leftarrow \boldsymbol{c}, \boldsymbol{m}$
  while $\tilde{T_{fj}} == T_{fj}$ **do**
    $\tilde{mul}_j \leftarrow \tilde{mul}_j + D \times k_j$
    $\tilde{c_j}, \tilde{m_j} \leftarrow \arg\min_{\tilde{c_j}, \tilde{m_j}} \tilde{T_{fj}}$
    if $\tilde{c_j}$ or $\tilde{m_j}$ does not meet the constraint **then**
      continue
    end if
  end while
  if $\sum_i \tilde{mul}_i < DSP \times D$ **then**
    $\boldsymbol{c}, \boldsymbol{m} \leftarrow \tilde{\boldsymbol{c}}, \tilde{\boldsymbol{m}}$
  else
    break //out of DSP usage
  end if
end while
Return $\boldsymbol{c}$ and $\boldsymbol{m}$

where Frequency is the operating frequency of FPGA chip. From the above equation, we know that larger c and m can simply increase FPS.

Besides, DSP efficiency can be calculated by the following equation and it is also related with c and m.

$$\begin{aligned} \mathrm{DSP}_{ei} &= \frac{T_{fi}}{T_{fmax}} \\ &= \frac{H_i \times W_i \times K_i \times \lceil \frac{C_i}{c_i} \rceil \times \lceil \frac{M_i}{m_i} \rceil}{\max\limits_{i \in N} \{H_i \times W_i \times K_i \times \lceil \frac{C_i}{c_i} \rceil \times \lceil \frac{M_i}{m_i} \rceil\}} \end{aligned} \tag{14}$$

The selection algorithm for $\boldsymbol{c}$ and $\boldsymbol{m}$ is shown in Algorithm 1. For each layer, we initialize c_i and m_i as 1 and D which are the smallest unit of channel parallelism. D represents the number of multipliers operated in one DSP. In the case of 8-bit by 8-bit multiplication by AMD/Xilinx DSP48, since one DSP48 can deal with two 8-bit×8-bit multiplication, D is equal to two. Based on Eq. 9, we can calculate the initialized number of multipliers for each layer. Based on Eq. 10 and Eq. 11, we can calculate the initialized number of clock cycles for generating one frame of each layer.

After the initialization, we start the selection algorithm. Considering that the throughput of overall pipeline is determined by the bottleneck layer with the maximum clock cycle, we first compare the number of required clock cycles among all the layers and find the bottleneck layer. For the bottleneck layer, we will iteratively allocate more multipliers to reduce the required clock cycles. In each iteration, we add $D \times k_j$ multipliers where D is the increment unit of the output channel parallelism and k_j is the increment unit of the kernel size parallelism. With more multipliers, we can find the combination of input and output channel parallelism which has the minimal number of required clock cycle. The iteration is looped until this layer is not the bottleneck anymore. Note that there is a constraint on input and output channel parallelism which will be introduced in the subsequent sections. Here, we simply skip the candidates not satisfying the constraints.

After finding the temporary input and output channel parallelism, if the overall number of multipliers exceeds the maximum value provided by the FPGA board, it represents that the DSP is out of usage thus the while loop will be terminated. Otherwise, we update the input and output channel parallelism and continue the while loop for the next-round selection.

Because of the fine granularity of DSP allocation, we can distribute the resource averagely to various layers. As a result, under the same DSP constraint, FPS and DSP efficiency will be high.

B. Channel Parallelism Constraint

Due to the routing congestion, the actual DSP usage in Algorithm 1 is fewer than the provided number in the board. When using KU115 for encoding 720P, only 2654 out

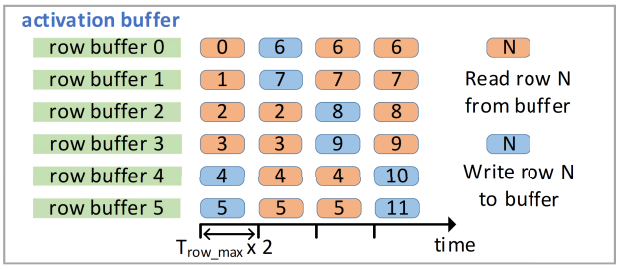


Fig. 5. Activation buffer scheduling.

of 5520 DSP can be utilized in the previous work. To mitigate this problem, we add the constraint to the channel parallelism.

Activation buffer is one of the bottlenecks in the routing phase. The activation buffer is composed of several row buffers, and each row buffer is to store the results of one complete row. The number of required row buffers is dependent on the kernel size, stride and the kernel reuse ratio R. In the case that kernel size is three, stride is one and R is two, we require six row buffers as shown in Fig. 5.

For each specific output channel, to generate a final result, we need to traverse the kernel of all the input channels. However, the number of input channels is so large that it is impossible to store all the kernels in on-chip BRAM. As a result, the kernel is stored in off-chip DDR at first, and then we periodically load kernel of partial input channels from off-chip DDR to on-chip BRAM. However, loading kernel from off-chip DDR to on-chip BRAM may exceed the maximum bandwidth. To solve this problem, kernel reuse is required.

Without any kernel reuse, each loaded kernel is only used for computing one row of one output channel. Considering that the kernel is actually the same for a specific input and output channel, each loaded kernel can then be used for computing multiple rows of one output channel. However, the downside of kernel reuse is more on-chip BRAM for the storage of intermediate output results.

In Fig. 5, both of the kernel reuse ratio for the previous layer and current layer are set as two. On one hand, the results of two rows (blue cell in Fig. 5) for the previous layer are generated and written in the activation buffer. On the other hand, since the kernel size is three, the results of four rows (orange cell in Fig. 5) of the previous layer are read to compute the results of two rows for the current layer.

The structure of row buffer is shown in Fig. 6, which is composed of an input crossbar, output crossbar and BRAM banks. Here, crossbar is a crossbar switch with multiple input and output that form a crossed pattern of interconnection between input and output. As shown in Fig. 5, each row buffer is iteratively used for reading and writing, thus the row buffer parallelism needs to be the maximum of the writing parallelism m_{i-1} and reading parallelism c_i.

$$r_i = max(m_{i-1}, c_i) \tag{15}$$

One BRAM (RAMB18K) can be configured as 8b×2048, 16b×1024 or 32b×512. The row result for one channel is stored in the depth direction of one BRAM bank.

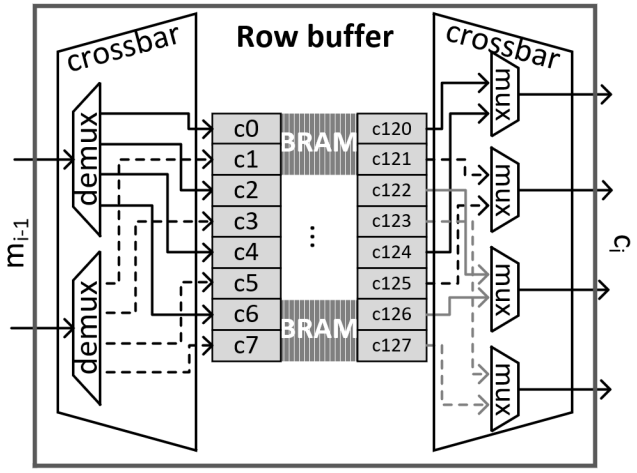


Fig. 6. Row buffer architecture.

After locating all the channels in the BRAM banks, there will usually be some margin in the depth direction of BRAM. To increase the BRAM efficiency in the depth-wise, BRAM is preferentially configured as a shallower shape (32b-width×512-depth).

When using 32b×512, given that the activation is 8b, four channels are stored in the same address of one BRAM bank. As a result, when the address of accessing m_{i-1} and c_i channels are at different address of one BRAM bank, the address conflict problem will occur. To solve this problem, a narrower configuration (i.e. 16b×1024, 8b×2048) will be selected. In the narrowest case of 8bx2048, the bitwidth of BRAM bank and activation are the same. Therefore, each activation has its own BRAM address, and there will be no address conflict within one BRAM bank.

Given that the width and depth of each BRAM slice is bw and bd, the bitwidth of activation is B, and the width of the output activation of the previous layer is W_{i-1}, the number of required BRAM banks in the width-wise and depth-wise can be calculated as below. Overall, $\text{BRAM}_{\text{width}} \times \text{BRAM}_{\text{depth}}$ banks are consumed for one row buffer.

$$\text{BRAM}_{\text{width}} = \lceil \frac{r_i \times B}{bw} \rceil \tag{16}$$

$$\text{BRAM}_{\text{depth}} = \lceil \frac{W_{i-1} \times \lceil \frac{C_i}{r_i} \rceil}{bd} \rceil \tag{17}$$

As shown in Eq. 14, given the neural network with C and M, c and m will decide the DSP efficiency. c and m with finer granularity can reach higher DSP efficiency, while they will lead to a larger hardware cost of LUT for the crossbar. Therefore, we need to set c and m as some value which could ease the hardware utilization.

As shown in Fig. 6, the crossbar is a connection between the input/output parallelism and the row buffer parallelism. For the i-th layer, the row buffer parallelism r_i is the maximum of the output parallelism of the previous layer m_{i-1} and the input parallelism of the current layer c_i. Therefore, we know that r_i will not be smaller than c_i and m_{i-1}. For each output sample of the previous layer, demultiplexer is used to select

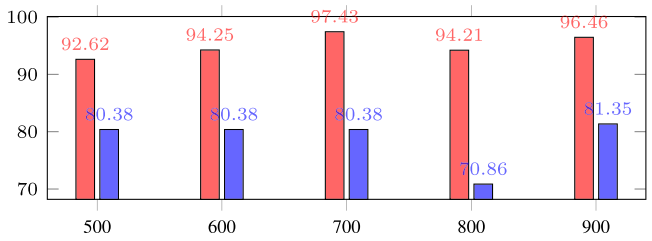


Fig. 7. DSP efficiency evaluation with different DSP usages. Red is the result without the channel parallelism constraint and blue is the result with the channel parallelism constraint *power-of-two*.

the writing BRAM channel. For each demultiplexer, the output number is counted as

$$A = \underset{\alpha}{\operatorname{argmin}}(r_i \mid m_{i-1} \times \alpha) \tag{18}$$

where $a \mid b$ means that b can be divided by a without remainder. On the other hand, we need to read the data from BRAM as the input for the current layer. Multiplexer is used to select the reading BRAM channel. The input number of multiplexer is given as below.

$$B = \underset{\beta}{\operatorname{argmin}}(r_i \mid c_i \times \beta) \tag{19}$$

To reduce the hardware cost of demultiplexer and multiplexer, A and B should be as small as possible. Without any limitation for c and m, c and m can be any value thus A and B are likely to be large. For example, when c_i is 7 and m_{i-1} is 16, then r_i is 16. Since the greatest common divisor of 7 and 16 is 1, thus B will be as large as 16, which means that each input channel is connected with all the row buffer channels. To reduce A and B, we restrict c and m with some constraints so that r can be multiple times more than c and m, which is corresponding to line 9 to 11 in *Algorithm* 1.

We provide two types of constraints. One is the hard constraint which limits c and m be the power of two, the other is the soft constraint which limits c and m be the multiple of two/four/eight. The optimal one is selected based on the throughput as shown in the experimental results.

C. Neural Network Search

Adding some constraint to the channel parallelism can reduce the hardware cost. However, given a neural network model, DSP allocation with a coarser granularity (i.e. channel parallelism) will decrease the DSP efficiency. Five cases with various DSP usages are shown in Fig. 7. We can see that the blue bar with the constraint suffers lower DSP efficiency.

To increase the efficiency, we alter the overall channel $\boldsymbol{C}$ and $\boldsymbol{M}$. An example for the encoder is described in *Algorithm* 2. Based on the default channel number, we explore different output channel for the first four g_a layers. $\boldsymbol{L}$ is the search space. For each searching iteration, we select the channel parallelism $\boldsymbol{c}$ and $\boldsymbol{m}$ according to *Algorithm* 1 and then calculate the DSP efficiency. The network with the highest DSP efficiency is finally selected. In the case that there are several searched results with the same DSP efficiency, we will pick up the one with smallest BRAM usages. Note that the output channel of the final g_a layer is not altered so as not to influence the decoder part.

Algorithm 2 Neural Network Search Algorithm, Temporary State Under Optimization Is Labeled With $\tilde{tilde}$

1: $\boldsymbol{C}$ and $\boldsymbol{M}$ are original input and output channel numbers, MAC_0 is original operation
2: $\boldsymbol{L}$ are the search space
3: DSP_effi_opt = 0
4: **for** $\tilde{\boldsymbol{M}}_{g_a^{(0)}} \in \boldsymbol{M}_{g_a^{(0)}} + \boldsymbol{L}$ **do**
5: **for** $\tilde{\boldsymbol{M}}_{g_a^{(1)}} \in \boldsymbol{M}_{g_a^{(1)}} + \boldsymbol{L}$ **do**
6: **for** $\tilde{\boldsymbol{M}}_{g_a^{(2)}} \in \boldsymbol{M}_{g_a^{(2)}} + \boldsymbol{L}$ **do**
7: **for** $\tilde{\boldsymbol{M}}_{g_a^{(3)}} \in \boldsymbol{M}_{g_a^{(3)}} + \boldsymbol{L}$ **do**
8: Select $\boldsymbol{c}$ and $\boldsymbol{m}$ according to *Algorithm* 1
9: Calculate DSP efficiency according to Eq. 14
10: Calculate MAC
11: **if** DSP efficiency > DSP_effi_opt **then**
12: **if** $\left|\frac{\text{MAC}-\text{MAC}_0}{\text{MAC}_0}\right| < 5\%$ **then**
13: $\hat{\boldsymbol{C}} \leftarrow \tilde{\boldsymbol{C}}, \hat{\boldsymbol{M}} \leftarrow \tilde{\boldsymbol{M}}$
14: Update DSP_effi_opt
15: **end if**
16: **end if**
17: **end for**
18: **end for**
19: **end for**
20: **end for**
21: Return $\hat{\boldsymbol{C}}$ and $\hat{\boldsymbol{M}}$

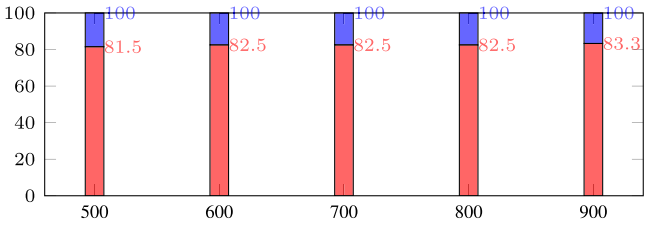


Fig. 8. DSP usage distribution of the main path (red) and hyper path (blue) without the channel parallelism constraint.

To avoid a significant coding loss due to the neural network search, we calculate multiply-accumulate (MAC) and compare with the original one. Once the MAC difference is larger than 5%, we supposed that the explored network will have a different coding ability with the origin one, thus we skip the selection.

The decoder is optimized in a similar way. We alter the output channel of the first four layers of g_s and select the combination with the highest DSP efficiency. The output channel of the final g_s layer is fixed as three for the representation of RGB channels. As a result, encoder and decoder can be searched separately at first, and then combined as a codec system.

In this paper, we do not take the hyper path into account in order to reduce the search complexity. Besides, the hyper path consume much fewer computation compared with main path as shown in Fig. 8. More than 80% DSP are cost for the main path.

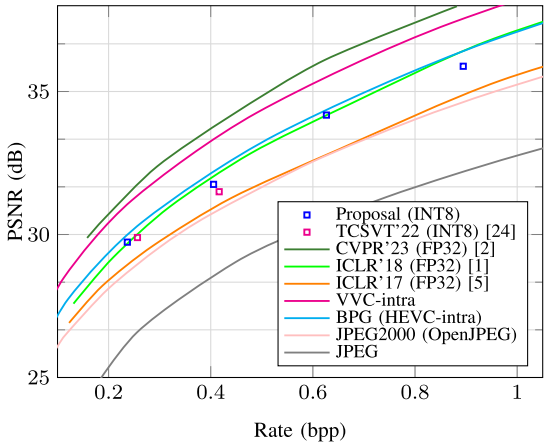


Fig. 9. Coding gain comparison with other LIC and traditional methods.

IV. Experimental Results

A. Implementation Detail

1) Algorithm: We first train a floating-point model from scratch by using Tensorflow. We use the rate-distortion function $J = R + \lambda \cdot D$ as the loss function. λ are set as 0.005 and 0.01 to align with the previous work [24], [25] for a fair comparison and D is set as MSE in our experiment. ImageNet and CLIC dataset are used as the training dataset. We train 2×10^6 iterations.

After that, we quantize the pre-trained model and fine tune the quantized model by following the method in [12]. Overall 4×10^4 iterations are fine tuned.

For the testing, we use Kodak dataset [28] for the evaluation. Note that the coding performance results in the next sections are all for the integer quantized model rather than the floating-point model.

2) Hardware: We use Verilog to ensure the clock-cycle-wise hardware behavior manipulation, and use Vivado to do the synthesis and implementation, and finally write the bitstream.

About the verification, we make a compiler to translate the pre-trained quantized Tensorflow model to a Python model which has exactly the same computational process with the hardware behavior. After that, we compare the software results and hardware results for the functional verification.

B. Algorithmic Rate-Distortion Results

We compare with other LIC methods and traditional methods in Fig. 9. First, we compare with [24] which is also an integer network on FPGA. Reference [24] provided the results of two λ (0.005 and 0.01). For both of the λ, our compression performance is better. Second, we compare with three LIC methods, among which [2] is one of recent works, [5] is classical factorized prior model and [1] is classical hyperprior. Note that [2] provided three models, and we use the medium one for the comparison. The results show that ours is close to [1] which is also based on CNN without any attention or Transformer network. Third, we compare with traditional methods. Ours is close to BPG.

TABLE II
MAC and Parameter Comparison With Other LIC

Method	Parameter (/M)	MAC (/G)
CVPR'23 [2]	58.72	415.20
ICLR'18 [1]	5.08	74.05
ICLR'17 [5]	3.0	72.41
Proposal	6.89	59.54

TABLE III
Soft Constraint for Various Boards

Board	**Soft constraint**
ZC706	Multiple-of-two
KU115	Multiple-of-four
VCU118	Multiple-of-eight

TABLE IV
Search Space for Various Constraints

Constraint	**Search space**
Multiple-of-two	$[0, \pm 2, \pm 4]$
Multiple-of-four	$[0, \pm 4, \pm 8]$
Multiple-of-eight	$[0, \pm 8, \pm 16]$
Power-of-two	$[0, \pm 4, \pm 8]$ for ZC706 $[0, \pm 8, \pm 16]$ for KU115 $[0, \pm 12, \pm 24]$ for VCU118

We also provide the number of parameters and MACs in Table II. The MAC is evaluated based on a Kodak image. We have the same degree of parameters and MACs compared to [1] and [5]. Reference [2] utilized about 8.52x higher parameters and 6.97x higher MACs than us. Using a more sophisticated network and deploying more DSPs will further enhance the coding performance, which could be our future work.

We then give the results of neural network search. The results are for three FPGA boards ZC706, KU115 and VCU118 with different scales, which are used for 360P, 720P and 1080P resolution processing respectively.

As described in Section III-B, we provide hard constraint and soft constraint. Hard constraint is always set as power-of-two, and soft constraint is selected according to the of FPGA boards. Considering the scale of c and m, the soft constraint is set as multiple-of-two, multiple-of-four and multiple-of-eight for ZC706, KU115 and VCU118 respectively as shown in Table III. The search space $\boldsymbol{L}$ in Algorithm 2 is shown in Table IV. As depicted in Algorithm 2, the first four layers in the main path are adjusted, thus we traverse 5^4 cases to pick up the one with the highest DSP efficiency.

The search networks are shown in Table V. Originally, the channel number of the first four layers are {128,128,128,192} and {192,128,128,128} for g_a and g_s respectively. For ZC706, the number of optimized channel are {128,120,128,190} and {198,124,120,124} for g_a and g_s respectively. For KU115, the results are {128,136,128,184} and {192,112,144,128}. For VCU118, the results are {120,120,144,192} and {192,128,128,128}. Note that for g_s of VCU118, the FPS of the searched network is same as origin, therefore, we do not adjust the network.

We then evaluate the coding performance of integer network models in Table VI. For Kodak dataset, when comparing

TABLE V
CHANNEL NUMBER BY NEURAL NETWORK SEARCH

Neural network	g_a/g_s	Optimal channel number
[25]	g_a	128, 128, 128, 192
	g_s	192, 128, 128, 128
Proposed model for ZC706	g_a	128, 120, 128, 190
	g_s	198, 124, 120, 124
Proposed model for KU115	g_a	128, 136, 128, 184
	g_s	192, 112, 144, 128
Proposed model for VCU118	g_a	120, 120, 144, 192
	g_s	192, 128, 128, 128

TABLE VI
CODING PERFORMANCE EVALUATION OF INTEGER MODEL

Neural network	PSNR (dB) ↑	bpp ↓	R-D cost ↓
[24]	29.89	0.2565	-
[25]	29.94	0.2542	0.6373
Proposal for ZC706	29.86	0.2434	0.6344
Proposal for KU115	30.06	0.2531	0.6227
Proposal for VCU118	29.73	0.2367	0.6424

the proposed network for ZC706 with [25], PSNR is worse with smaller bpp. When comparing the proposed network for KU115 with [25], PSNR is better and bpp is also smaller. The situation of VCU118 is similar with ZC706. We also provide R-D cost for an explicit comparison. We can see that the searched networks for ZC706 and KU115 are better than [25] in terms of R-D cost. The searched network for VCU118 is slightly worse than [25] in terms of R-D cost.

C. Architectural Hardware Results

1) Comparison With Previous Work: We first give the comparison with previous work. The hardware performance comparison with [25] is given in Table VII. We evaluate encoding and decoding of three boards. For ZC706, compared with [25], encoding and decoding can be 1.32x and 1.23x faster respectively. For KU115, compared with [25], encoding and decoding can be 1.5x and 1.22x faster respectively. For VCU118, compared with [25], encoding and decoding can be 1.29x and 1.25x faster respectively.

For each combination of board and coding, there are three configurations. First one is without channel parallelism constraint and network search, which is exactly [25]. Second is with channel parallelism constraint but without network search. Third is with both channel parallelism constraint and network search. When using ZC706 for the encoding, only 680 DSP can be used in the first configuration. By adding some constraints on the channel parallelism, the hardware cost of activation buffer can be reduced so that the place and route is loosened. As a result, DSP usage can be increased from 680 to 850. However, DSP efficiency is reduced from 94.01% to 89.09%. Finally, to increase the DSP efficiency, we search the optimal neural network. DSP efficiency can be increased to 96.47%. Note that DSP usage generally become higher after the network search. Here we propose a heuristic method, when performing the network search, we set the DSP usage in the second configuration as the targeting DSP number. If the design can be successfully placed and routed, we then gradually add the number of targeting DSP until the routing error.

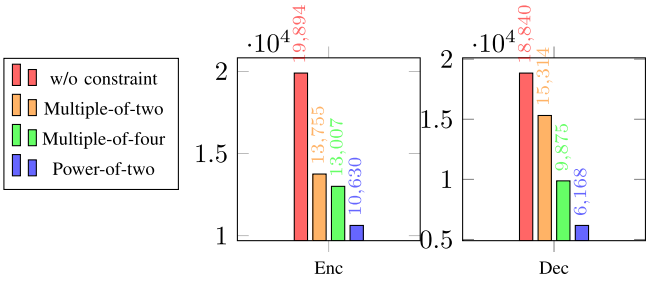


Fig. 10. LUT usage of activation buffer.

For the other combinations of board and coding, the tendency of results are similar. Among the three configurations, performing channel parallelism constraint can improve DSP usage while reduce DSP efficiency. However, network search can improve both DSP utilization and DSP efficiency.

2) Hardware Performance Evaluation: As described in Section III-B, the target of adding the constraint to the channel parallelism is to reduce the LUT cost. To convince this assumption, we evaluate four different constraints: no contraint, multiple-of-two, multiple-of-four, power-of-two. For ZC706, the results are shown in Fig. 10. To make a fair comparison, we set the encoding throughput of all the four cases close to 25FPS, and set the decoding throughput of all the four cases close to 28FPS. For the encoding, 19894 LUT are cost without any constraint, and it can be reduced to 13755, 13007 and 10630 with the constraints of multiple-of-two, multiple-of-four and power-of-two respectively. For the decoding, 18840 LUT are cost without any constraint, and it can be reduced to 15314, 9875 and 6168 for multiple-of-two, multiple-of-four and power-of-two respectively. We can conclude that no-constraint case consumes the most LUT. Among various constraints, harder constraint leads to smaller LUT cost. On the other hand, a harder constraint can lead to more DSP usage. The layer-wise DSP utilization is shown in Fig. 11. More DSP will be used with the constraint of power-of-two.

As described in Section III-C, the target of network search is to improve the DSP efficiency which has been reflected in Table VII. In addition, we evaluate the layer-wise DSP efficiency of the searched layers in Fig. 12. Since we only adjust the layers in the main path, we evaluate all the five layers in analysis transform g_a and synthesis transform g_s. According to the results, DSP efficiency of most layers are increased. Since the constraint of power-of-two is selected for KU115 decoding, it cannot realize as high DSP efficiency as other scenarios after the network search. This implies that harder constraint will cause relatively low DSP efficiency.

As described above, when using the hard constraint, we may use more DSP because of lower LUT cost. However, the DSP efficiency will be lower due to the coarse granularity. As a result, this is a trade-off between DSP usage and DSP efficiency, and the optimal one is selected according to their product. The comparison of hard constraint (power-of-two) and soft constraint (multiple-of-four) for KU115 are shown in

TABLE VII
HARDWARE PERFORMANCE COMPARISON WITH [25]

Board	Enc/dec	Method	c, m constraint	Network search	DSP usage	DSP efficiency	Throughput
ZC706	Enc	[25]	no	no	680	94.01	360P@25.68FPS
		Proposal	yes	no	850	89.09	360P@30.42FPS
			yes	yes	847	96.47	360P@33.91FPS
	Dec	[25]	no	no	738	93.06	360P@28.55FPS
		Proposal	yes	no	813	89.79	360P@30.35FPS
			yes	yes	847	95.17	360P@35.00FPS
KU115	Enc	[25]	no	no	2654	93.94	720P@25.04FPS
		Proposal	yes	no	3704	91.14	720P@33.91FPS
			yes	yes	4160	91.67	720P@37.74FPS
	Dec	[25]	no	no	2517	94.26	720P@24.66FPS
		Proposal	yes	no	3314	78.75	720P@27.13FPS
			yes	yes	3354	86.04	720P@30.14FPS
VCU118	Enc	[25]	no	no	4500	93.23	1080P@19.83FPS
		Proposal	yes	no	5376	83.73	1080P@21.28FPS
			yes	yes	5880	92.86	1080P@25.53FPS
	Dec	[25]	no	no	4454	93.75	1080P@20.42FPS
		Proposal	yes	no	6020	86.71	1080P@25.53FPS
			yes	yes	6020	86.71	1080P@25.53FPS

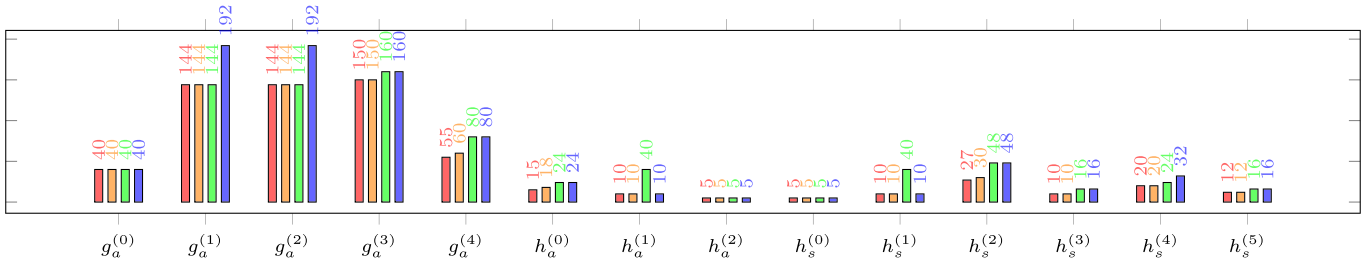


Fig. 11. Layer-wise DSP utilization.

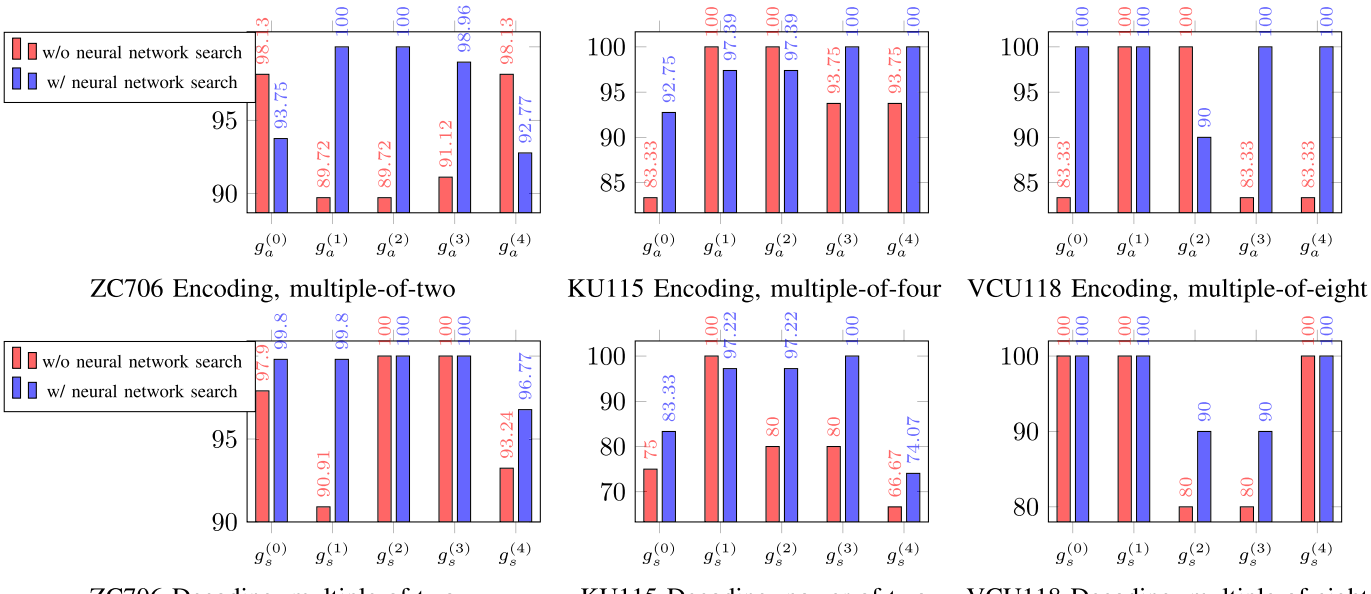


Fig. 12. Layer-wise DSP efficiency.

Table VIII. For both encoding and decoding, more DSP can be used in the case of hard constraint, while DSP efficiency is high in the case of soft constraint. As a result, the product of DSP usage and efficiency will decide the optimal constraint. For ZC706 and VCU118, we do the similar comparison and select the soft constraint.

3) Implementation Results: The hardware utilization ratio of six cases are shown in Fig. 13. For LUT, thanks to adding the constraint to the input and output channel parallelism, there is no obvious increment after using the proposal. For FF, PE contains some sequential circuits whose scale is proportional to the input and output parallelism. Therefore,

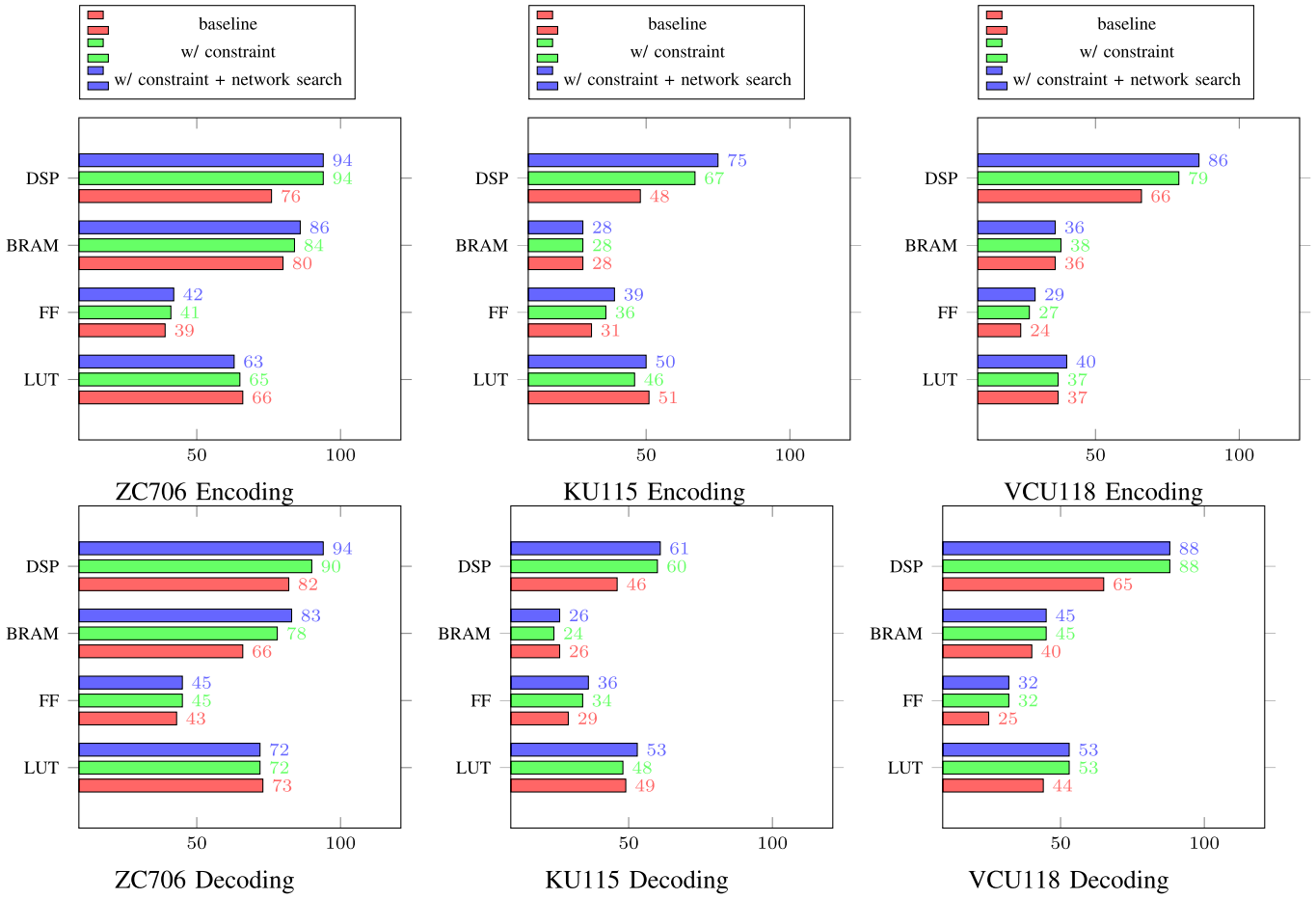


Fig. 13. Resource utilization result after implementation.

TABLE VIII
HARD AND SOFT CONSTRAINT COMPARISON

	Constraint	DSP usage	DSP efficiency	Throughput
Enc	Hard	4204	84.64	36.17
	Soft	4160	91.67	37.74
Dec	Hard	3354	86.04	30.14
	Soft	2998	91.88	29.59

using more DSPs will lead to larger FF usage. For BRAM, there is no big difference in most cases. The reason is that the searched network with the smallest BRAM has the higher priority to be selected. For DSP, we can utilize more than the baseline which is one of the target of this paper.

D. Discussion

1) Generality of the Proposal: To illustrate the generality of the proposed method, we have added the results from three aspects. First, we evaluate a factorized prior model without the hyper path. Second, we apply the method to a hyperprior model with larger channels in the main path. Third, we have added the results with various λ to support various rate-distortion trade-off.

First, we apply the method to the factorized model without hyper path. The network architecture is shown in Fig. 14.

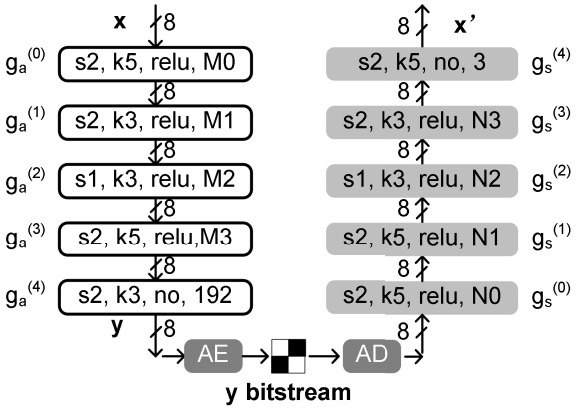


Fig. 14. Factorized prior encoder and decoder diagram. The output channels of first four layers (M0, M1, M2, M3) of g_a and the first four layers (N0, N1, N2, N3) of g_s are decided by the network search.

Now that there are only the main path, we apply the proposal to the first four layers in g_a and g_s. The final layer of g_a is not modified so that g_a and g_s can be adjusted individually. We evaluate the encoding and decoding of two resolutions on KU115. For each scenario, we evaluate the baseline and proposal, and the results are shown in Table IX.

TABLE IX
HARDWARE PERFORMANCE COMPARISON FOR THE FACTORIZED PRIOR MODEL

Board	Enc/dec	Method	c, m constraint	Network search	DSP usage	DSP efficiency	Throughput
KU115	Enc	[25]	no	no	3116	91.97	720P@34.27FPS
		Proposal	yes	no	3752	90.70	720P@40.69FPS
			yes	yes	3752	93.69	720P@43.20FPS
	Dec	[25]	no	no	2956	94.46	720P@33.39FPS
		Proposal	yes	no	3440	87.94	720P@36.17FPS
			yes	yes	3560	94.13	720P@38.75FPS
	Enc	[25]	no	no	2675	94.24	1080P@14.18FPS
		Proposal	yes	no	3560	91.04	1080P@18.24FPS
			yes	yes	3792	92.95	1080P@20.42FPS
	Dec	[25]	no	no	2792	92.87	1080P@14.59FPS
		Proposal	yes	no	3560	91.04	1080P@18.24FPS
			yes	yes	3776	93.81	1080P@20.42FPS

TABLE X
CHANNEL NUMBER BY NEURAL NETWORK SEARCH FOR THE FACTORIZED PRIOR MODEL

Neural network	g_a/g_s	Optimal channel number
Baseline	g_a	128, 128, 128, 192
	g_s	192, 128, 128, 128
Proposal for 720P	g_a	136, 120, 120, 200
	g_s	192, 128, 136, 128
Proposal for 1080P	g_a	120, 120, 136, 188
	g_s	192, 136, 120, 120

TABLE XI
CODING PERFORMANCE EVALUATION OF INTEGER FACTORIZED PRIOR MODEL

Neural network	PSNR (dB) ↑	bpp ↓	R-D cost ↓
Baseline	30.42	0.3552	0.6847
Proposal for 720P	30.42	0.3529	0.6833
Proposal for 1080P	30.45	0.3568	0.6851

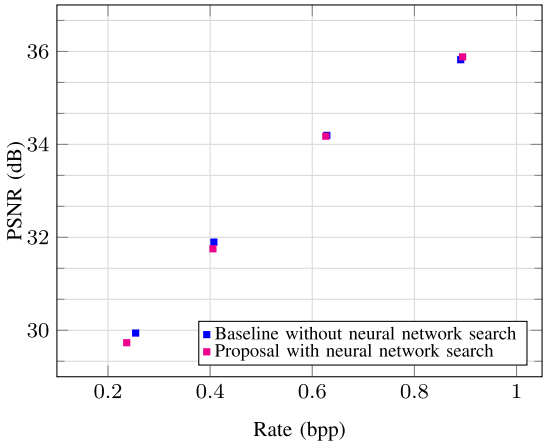


Fig. 15. Coding gain comparison between the baseline and proposal. The results illustrate that the neural network search will not influence the coding performance.

The interpretation of the experimental results are similar as the hyperprior model. When adding the constraint for input and output channel parallelism, more DSP can be utilized with lower DSP efficiency. After that, by changing the channel number through the neural network search, DSP efficiency can be increased. As a result, for the encoding and decoding of 720P, the throughput can become 1.26x and 1.16x faster. For the encoding and decoding of 1080P, the throughput can become 1.44x and 1.40x faster.

The searched results are shown in Table X, and the coding performance evaluation is shown in Table XI. We can see that the searched network has almost the same coding performance as the origin.

In addition to the factorized model, we apply the method to the hyperprior model with larger channels in the main path, which is marked as *hyper-large* model. For *hyper-large*, the channel number of the first four layers are {192,192,192,256} and {256,192,192,192} for g_a and g_s respectively. The channel number of the bottleneck layer is 256. The hardware results for 720P processing on VCU118 are shown in Table XII. For the encoding and decoding, the throughput can be increased by 1.20 and 1.23 times respectively. Note that similar with the *hyper-small* model, for the decoding, since the network searching cannot improve the throughput anymore thus the final two rows are the same in Table XII.

Finally, we evaluate the results with multiple bitrates. For the *hyper-small*, we evaluate two λ 0.005 and 0.01. The comparison with [24] on two λ is shown in Table XIV. For the *hyper-large*, we provide the results of two large λ (0.02 and 0.04).

For each model, we compare the coding performance of the baseline and proposal in Fig. 15. From the results, we can see that our network searching will not influence the coding performance for a wide range of bitrates.

2) MAC Threshold: The target of the neural network search is to find a neural network with faster throughput while not influencing the coding performance. However, it is not practical to train and test each neural network candidate during the neural network search stage, which is too time consuming. As a result, we add one threshold for the MAC difference.

From the algorithmic point of view, the coding performance of LIC is generally related to the neural network size (i.e. MAC). We train several neural networks with various MAC. The basic network structure is shown in Fig. 1 with $\boldsymbol{M} = \{128, 128, 128, 192\}$ and $\boldsymbol{N} = \{192, 128, 128, 128\}$. For the basic network, the R-D cost with λ being 0.01 is shown in the fourth row of Table XV. After that, we increase and decrease the number of $\boldsymbol{M}$ and $\boldsymbol{N}$ to set the MAC difference threshold as $\pm 3\%$, $\pm 5\%$ and $\pm 8\%$. For example, in the case of -8%, $\boldsymbol{M} = \{121, 121, 121, 182\}$, $\boldsymbol{N} = \{182, 121, 121, 121\}$. In the case of $+8\%$, $\boldsymbol{M} = \{134, 134, 134, 202\}$,

TABLE XII
HARDWARE PERFORMANCE COMPARISON FOR THE *Hyper-Large* MODEL

Board	Enc/dec	Method	c, m constraint	Network search	DSP usage	DSP efficiency	Throughput
VCU118	Enc	[25]	no	no	4625	96.00	720P@23.25FPS
		Proposal	yes	no	5496	85.68	720P@24.66FPS
			yes	yes	5864	93.69	720P@27.82FPS
	Dec	[25]	no	no	4704	95.79	720P@24.11FPS
		Proposal	yes	no	5850	94.54	720P@29.59FPS
			yes	yes	5850	94.54	720P@29.59FPS

TABLE XIII
CHANNEL NUMBER BY NEURAL NETWORK SEARCH FOR THE *Hyper-Large* MODEL

Neural network	g_a/g_s	Optimal channel number
Baseline	g_a	192, 192, 192, 256
	g_s	256, 192, 192, 192
Proposal	g_a	192, 208, 192, 248
	g_s	256, 192, 192, 192

TABLE XIV
COMPARISON WITH [24] FOR 1080P PROCESSING

Method	Proposal	[24]
Platform	VCU118	ZCU104
Technology	16nm	16nm
Encoding FPS ↑	25.53	12.87
Decoding FPS ↑	25.53	3.00
PSNR (λ=0.005) ↑	29.73	29.89
bpp (λ=0.005) ↓	0.2367	0.2565
PSNR (λ=0.01) ↑	31.75	31.50
bpp (λ=0.01) ↓	0.4055	0.4165

TABLE XV
R-D COST WITH DIFFERENT MAC

MAC diff	R-D cost ↓
-8%	0.8829
-5%	0.8736
-3%	0.8650
0	0.8692
+3%	0.8672
+5%	0.8733
+8%	0.8643

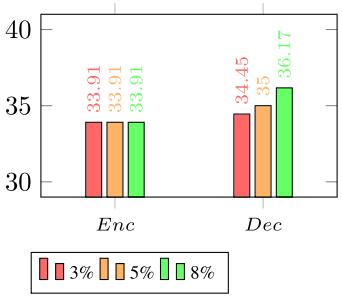


Fig. 16. Searched result (throughput) with various MAC difference threshold.

TABLE XVI
THROUGHPUT WITH VARIOUS CONSTRAINTS FOR ZC706 ENCODING

Constraint	x1	x2	x4	x8	x16
Throughput	25.68	33.91	27.13	20.35	0.85

TABLE XVII
THROUGHPUT WITH VARIOUS CONSTRAINTS FOR VCU118 ENCODING

Constraint	x1	x2	x4	x8	x16
Throughput	19.83	20.42	21.28	21.28	21.28

$\boldsymbol{N} = \{202, 134, 134, 134\}$. From the results in Table XV, we know that the R-D cost is likely to become smaller (i.e. better coding performance) with larger MAC.

From the architectural point of view, the throughput by the neural network search may become higher under the constraint of a larger MAC difference. For ZC706, the searched results with different thresholds are shown in Fig. 16. For the encoding, the same throughput can be reached with all the three thresholds. For the decoding, the fastest throughput can be reached in the case of 8% threshold.

Therefore, the MAC threshold actually leads to a trade-off between the algorithmic coding performance and architectural throughput performance. A larger MAC threshold may lead to a faster throughput, while it takes the risk of worse coding performance. A smaller MAC threshold can be more likely to ensure the coding performance, while it loses some opportunities of achieving high throughput.

3) Rationale of Constraints: The soft constraint for three boards are shown in Table III. With a harder constraint, the choice of input and output channel parallelism is limited thus the DSP efficiency will become lower. On the other hand, a harder constraint help relieving the routing congestion thus the DSP utilization will become higher. As a result, it is a trade-off between DSP utilization and DSP efficiency.

We compare the results with various constraints. The results for ZC706 encoding are shown in Fig. 17 and Table XVI. The results for VCU118 encoding are shown in Fig. 18 and Table XVII. In the results, *multiple-of-one* represents *no-constraint*. The results of DSP utilization and DSP efficiency are as expected. The multiplication of DSP utilization and DSP efficiency is proportional to the throughput in Table XVI. According to the result, *multiple-of-two* is selected for ZC706. The results for VCU118 is shown in Fig. 18. We know that with a harder constraint, the DSP utilization become larger while DSP efficiency becomes lower. As a result, the throughput for *multiple-of-four/eight/sixteen* are the same.

Theoretically, we can traverse all the possible constraints and select the one with the best throughput. However, going through all the cases is very time consuming. Considering that the input and output channel parallelism is related with the DSP resource of the FPGA, the constraint for ZC706, KU115 and VCU118 are set as *multiple-of-two/four/eight* respectively in this paper.

4) Extension to ASIC: The methodology of algorithm-architecture co-optimization is also applicable to ASIC design.

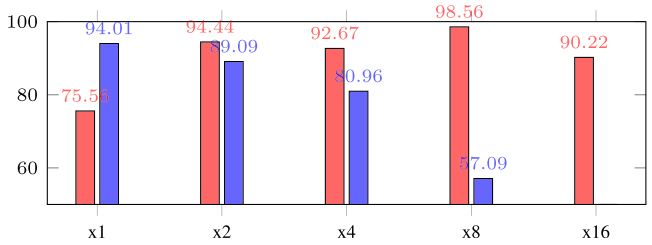


Fig. 17. DSP utilization and DSP efficiency with various constraints for ZC706 encoding.

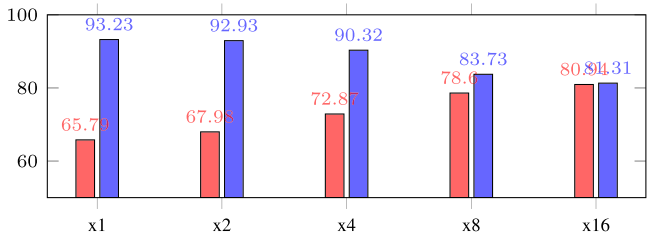


Fig. 18. DSP utilization and DSP efficiency with various constraints for VCU118 encoding.

However, the hardware logic is different from FPGA and ASIC. For the combinational circuit, the basic unit is LUT for FPGA, while it is NAND gate for ASIC. Besides, the memory configuration for ASIC is very flexible, while BRAM can only be configured to several shapes in FPGA. Also, depending on the used ASIC technology, the availability of memory configurations and well-designed arithmetic circuits may be different.

Therefore, when applying the algorithm-architecture co-optimization in ASIC design, the algorithm-level modification can become more flexible while the collaborative interactions between the algorithm and architecture also become more important.

V. Conclusion

This paper gives an algorithm-architecture co-design for LIC. By adding a limitation of the channel parallelism and a consequent neural network search, the processing throughput can be improved by up to 50% compared with the latest work. In fact, the methodology of this algorithm-architecture co-optimization can also be applied to other applications.

About future work, first, in the current design, all the multiplications are performed by the DSP. As a result, the hardware utilization of various resource is not so balanced. In some cases, the DSP utilization is more than 80%, while the BRAM/FF/LUT utilizations are still lower than 50%. To fully utilize all the resources, we can use LUT or BRAM to construct the multiplier as well. Second, though DSP utilization can be increased by our proposal, it still cannot reach 100%. To reach higher DSP utilization, we need to manually place the DSP and some other resources as well. Correspondingly, we can have a floorplan-aware algorithm optimization.

References

[1] J. Ballé, D. Minnen, S. Singh, S. J. Hwang, and N. Johnston, "Variational image compression with a scale hyperprior," in *Proc. Int. Conf. Learn. Represent.*, 2018, pp. 1–47.

[2] J. Liu, H. Sun, and J. Katto, "Learned image compression with mixed transformer-CNN architectures," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2023, pp. 14388–14397.

[3] W. Duan et al., "Learned image compression using cross-component attention mechanism," *IEEE Trans. Image Process.*, vol. 32, pp. 5478–5493, 2023.

[4] B. Bross et al., "Overview of the versatile video coding (VVC) standard and its applications," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 31, no. 10, pp. 3736–3764, Oct. 2021.

[5] J. Ballé, V. Laparra, and E. P. Simoncelli, "End-to-end optimized image compression," 2016, *arXiv:1611.01704*.

[6] D. Minnen, J. Ballé, and G. D. Toderici, "Joint autoregressive and hierarchical priors for learned image compression," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 31, Dec. 2018, pp. 10771–10780.

[7] Y. Zhu, Y. Yang, and T. Cohen, "Transformer-based transform coding," in *Proc. Int. Conf. Learn. Represent.*, 2021, pp. 1–12.

[8] Y. Qian, X. Sun, M. Lin, Z. Tan, and R. Jin, "Entroformer: A transformer-based entropy model for learned image compression," in *Proc. Int. Conf. Learn. Represent.*, 2021, pp. 1–15.

[9] R. Zou, C. Song, and Z. Zhang, "The devil is in the details: Window-based attention for image compression," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2022, pp. 17492–17501.

[10] A. B. Koyuncu, H. Gao, A. Boev, G. Gaikov, E. Alshina, and E. Steinbach, "Contextformer: A transformer with spatio-channel attention for context modeling in learned image compression," in *Proc. Eur. Conf. Comput. Vis.* Tel Aviv, Israel: Springer, Oct. 2022, pp. 447–463.

[11] J. Ballé, N. Johnston, and D. Minnen, "Integer networks for data compression with latent-variable models," in *Proc. Int. Conf. Learn. Represent.*, 2019, pp. 1–10.

[12] H. Sun, L. Yu, and J. Katto, "Learned image compression with fixed-point arithmetic," in *Proc. Picture Coding Symp. (PCS)*, Jun. 2021, pp. 1–5.

[13] D. He, Z. Yang, Y. Chen, Q. Zhang, H. Qin, and Y. Wang, "Post-training quantization for cross-platform learned image compression," 2022, *arXiv:2202.07513*.

[14] E. Koyuncu, T. Solovyev, E. Alshina, and A. Kaup, "Device interoperability for learned image compression with weights and activations quantization," in *Proc. Picture Coding Symp. (PCS)*, Dec. 2022, pp. 151–155.

[15] E. Koyuncu, T. Solovyev, J. Sauer, E. Alshina, and A. Kaup, "Quantized decoder in learned image compression for deterministic reconstruction," 2023, *arXiv:2312.11209*.

[16] E. Nurvitadhi, J. Sim, D. Sheffield, A. Mishra, S. Krishnan, and D. Marr, "Accelerating recurrent neural networks in analytics servers: Comparison of FPGA, CPU, GPU, and ASIC," in *Proc. 26th Int. Conf. Field Program. Log. Appl. (FPL)*, Aug. 2016, pp. 1–4.

[17] E. Nurvitadhi, D. Sheffield, J. Sim, A. Mishra, G. Venkatesh, and D. Marr, "Accelerating binarized neural networks: Comparison of FPGA, CPU, GPU, and ASIC," in *Proc. Int. Conf. Field-Programmable Technol. (FPT)*, Dec. 2016, pp. 77–84.

[18] X. Zhang et al., "DNNExplorer: A framework for modeling and exploring a novel paradigm of FPGA-based DNN accelerator," in *Proc. IEEE/ACM Int. Conf. Comput. Aided Design (ICCAD)*, Nov. 2020, pp. 1–9.

[19] H. Ye, X. Zhang, Z. Huang, G. Chen, and D. Chen, "HybridDNN: A framework for high-performance hybrid DNN accelerator design and implementation," in *Proc. 57th ACM/IEEE Design Autom. Conf. (DAC)*, Jul. 2020, pp. 1–6.

[20] J. Zhang and J. Li, "Improving the performance of OpenCL-based FPGA accelerator for convolutional neural network," in *Proc. ACM/SIGDA Int. Symp. Field-Program. Gate Arrays*, Feb. 2017, pp. 25–34.

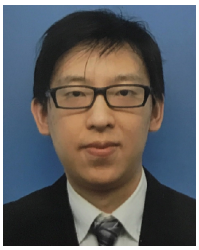
[21] C. Hao et al., "FPGA/DNN co-design: An efficient design methodology for IoT intelligence on the edge," in *Proc. 56th Annu. Design Autom. Conf.*, 2019, pp. 1–6.

[22] X. Zhang et al., "DNNbuilder: An automated tool for building high-performance DNN hardware accelerators for FPGAs," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD)*, 2018, pp. 1–8.

[23] X. Wei, Y. Liang, X. Li, C. H. Yu, P. Zhang, and J. Cong, "TGPA: Tile-grained pipeline architecture for low latency CNN inference," in *Proc. IEEE/ACM Int. Conf. Computer-Aided Design (ICCAD)*, Nov. 2018, pp. 1–8.

[24] C. Jia, X. Hang, S. Wang, Y. Wu, S. Ma, and W. Gao, "FPX-NIC: An FPGA-accelerated 4K ultra-high-definition neural video coding system," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 32, no. 9, pp. 6385–6399, Sep. 2022.

[25] H. Sun, Q. Yi, F. Lin, L. Yu, J. Katto, and M. Fujita, "F-LIC: FPGA-based learned image compression with a fine-grained pipeline," in *Proc. IEEE Asian Solid-State Circuits Conf. (A-SSCC)*, Nov. 2022, pp. 1–3.
[26] Xilinx. *Dsp48 Macro V3.0*. Accessed: Mar. 4, 2024. [Online]. Available: https://docs.xilinx.com/v/u/en-U.S./pg148-dsp48-macro
[27] G. J. Sullivan and T. Wiegand, "Rate-distortion optimization for video compression," *IEEE Signal Process. Mag.*, vol. 15, no. 6, pp. 74–90, Nov. 1998.
[28] R. Franzen. (1999). *Kodak Lossless True Color Image Suite*. [Online]. Available: http://r0k.us/graphics/kodak



Heming Sun (Member, IEEE) received the B.E. degree in electronic engineering from Shanghai Jiao Tong University, Shanghai, China, in 2011, the M.E. degree from Waseda University in 2012, the M.E. degree from Shanghai Jiao Tong University in 2014, and the Ph.D. degree from Waseda University in 2017. He was a Researcher with NEC Central Research Laboratories from 2017 to 2018. He was an Assistant Professor with Waseda University from 2018 to 2023. He is currently an Associate Professor with Yokohama National University and a Guest Researcher with Waseda University. His research interests include algorithms and VLSI architectures for image/video processing and neural networks.

Qingyang Yi received the master's degree in information engineering from The University of Tokyo, Tokyo, Japan, in 2022. Since 2020, he has been researching on hardware implementation of neural network-based information processing systems with the VLSI Design and Education Center, The University of Tokyo. He has been an Engineer with Huawei Technologies Company Ltd., Shenzhen, China, since 2022.

Masahiro Fujita (Life Member, IEEE) received the Ph.D. degree in information engineering from The University of Tokyo, Tokyo, Japan, in 1985.

He joined Fujitsu, Tokyo, in 1985. From 1993 to 2000, he was with Fujitsu Laboratories of America, Santa Clara, CA, USA, where he was the Director of the VLSI CAD Research Group. Since 2000, he has been a Professor with the VLSI Design and Education Center, The University of Tokyo, until he retired in 2022. He is currently a Researcher with The University of Tokyo. His research interests include hardware/software codesigns targeting embedded systems and formal analysis, verification, and synthesis of cyber-physical systems. Recently, he has also working on hardware implementation of neural network-based information processing systems, such as learned image compression. He has been a program and steering committee member of many prestigious international conferences and journals.