IEICE TRANSACTIONS

on Fundamentals of Electronics, Communications and Computer Sciences

DOI:10.1587/transfun.2023VLP0017

Publicized:2023/08/31

A PUBLICATION OF THE ENGINEERING SCIENCES SOCIETY

The Institute of Electronics, Information and Communication Engineers

Kikai-Shinko-Kaikan Bldg., 5-8, Shibakoen 3 chome, Minato-ku, TOKYO, 105-0011 JAPAN

PAPER

Pipelined ADPCM Compression for HDR Synthesis on an FPGA

Masahiro NISHIMURA[†], **Taito MANABE**[†], ***Nonmembers, and*** **Yuichiro SHIBATA**[†], ***Member***

SUMMARY This paper presents an FPGA implementation of real-time high dynamic range (HDR) synthesis, which expresses a wide dynamic range by combining multiple images with different exposures using image pyramids. We have implemented a pipeline that performs streaming processing on images without using external memory. However, implementation for high-resolution images has been difficult due to large memory usage for line buffers. Therefore, we propose an image compression algorithm based on adaptive differential pulse code modulation (ADPCM). Compression modules based on the algorithm can be easily integrated into the pipeline. When the image resolution is 4K and the pyramid depth is 7, memory usage can be halved from 168.48 % to 84.32 % by introducing the compression modules, resulting in better quality.

key words: *FPGA, HDR synthesis, image compression*

1. Introduction

HDR (High Dynamic Range) synthesis is one of the methods to cope with a narrow dynamic range of typical cameras. Generally, HDR synthesis consists of two steps: image composition and tone mapping. The image composition is a process to aggregate multiple SDR (Standard Dynamic Range) images with different exposure times into a single HDR image by estimating real-world luminance of each pixel. Then, the output HDR image is converted into an SDR image without blown-out highlights and blocked-up shadows through the tone mapping process.

On the other hand, Mertens et al. proposed an HDR synthesis method that directly combines images [1]. Mertens' method does not require information on exposure time of each image and the camera-specific response function. Multiple images are directly combined into a single image using weighted average based on visual criteria, and no tone mapping process is required.

HDR synthesis has a wide range of applications, some of which require real-time and low-latency processing, such as autonomous driving. Hardware processing using an field-programmable gate array (FPGA) is promising for such applications. However, most of the existing FPGA implementations of HDR synthesis adopted the two-stage approach using image composition and tone mapping [2–7], which is computationally complex and requires parameter tuning for specific camera devices. Though FPGA-based HDR synthesis based on the simplified Mertens' method has been proposed in [8], the implementation depends on external memory (DRAM) for buffering image frames, which inevitably results in high latency and requires additional implementation area and costs.

[†]Nagasaki University

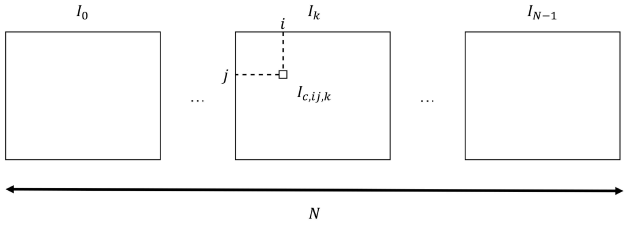


Fig. 1: Description of image list I.

On the other hand, our study group has implemented Mertens' method on FPGA with a fully pipelined architecture that can greatly reduce latency. In the previous implementation by Katayama et al. [9], lack of BRAM resources prevented implementation for high-resolution images. Since resources other than BRAM were still available, the imbalance in resource consumption needs to be alleviated. Therefore, we propose improved HDR synthesis hardware that incorporates an image compression technology in the pipeline and evaluate the impact of compression on resource usage and HDR synthesis quality. This paper is an extended version of [10] and describes the algorithm, architecture, and evaluation results in more detail.

2. HDR Synthesis Methods

2.1 Mertens' Method

In Mertens' HDR synthesis method, weights are calculated for pixel values of each of the N input images. The weights are then used to calculate a weighted average of the pixel values at the same coordinates across the different images. There are three types of weights:

- $C_{ij,k}$ (Contrast): Edge strength by Laplacian filter
- $S_{ij,k}$ (Saturation): Standard deviation of RGB values
- $E_{ij,k}$ (Well-exposedness): Product of distances between RGB values and 0.5 weighted by Gaussian curve

Let us define the list of N images as shown in Fig. 1 as I and the value of any RGB color channel at coordinates (i, j) of the k-th image I_k of I as $I_{c,ij,k}$ ($c \in \{r, g, b\}$, $0 \leq k < N$, $0 \leq i < HEIGHT$, $0 \leq j < WIDTH$). *HEIGHT* and *WIDTH* are the numbers of vertical and horizontal pixels in the input image, respectively. The pixel value of I is assumed to be normalized to the range $0 \leq I_{c,ij,k} < 1$.

The weight $C_{ij,k}$ is obtained by grayscaling the image I_k, applying an 8-nearest neighbor Laplacian filter L_{filter}, and taking the absolute value. This can be expressed as Eq. (1) and (2). Note that the operator $*$ denotes convolution, and application of the Laplacian filter does not change the image size because padding of width 1 is applied in advance.

$$L(I_k) = \mathrm{gray}(I_k) * L_{filter} \quad (padding = 1) \tag{1}$$

$$C_{ij,k} = |L(I_k)_{ij}| \tag{2}$$

Let us define $\mu_{I_{ij,k}}$ as the average of RGB values at the coordinates (i, j) of the image I_k. The weight $S_{ij,k}$ is the standard deviation of the RGB values, given by Eq. (3).

$$S_{ij,k} = \sqrt{\frac{1}{3}\sum_{c}^{r,g,b}(I_{c,ij,k} - \mu_{I_{ij,k}})^2} \tag{3}$$

In this paper, for simplicity of implementation, it is approximated by the mean absolute error as in Eq. (4).

$$S_{ij,k} = \frac{1}{3}\sum_{c}^{r,g,b}|I_{c,ij,k} - \mu_{I_{ij,k}}| \tag{4}$$

The weight $E_{ij,k}$ is the product of distances between RGB values and 0.5 weighted by a Gaussian curve. Therefore, the weight $E_{ij,k}$ is expressed as Eq. (5).

$$E_{ij,k} = \prod_{c}^{r,g,b}\exp\left(-\frac{(I_{c,ij,k} - 0.5)^2}{2\sigma^2}\right) \quad (\sigma = 0.2) \tag{5}$$

The three weights $C_{ij,k}$, $S_{ij,k}$, and $E_{ij,k}$ are multiplied together to obtain the weight $W_{ij,k}$, whose expression is given as Eq. (6). ωC, ωS and ωE are parameters that determine which weight to emphasize. All the parameters are set to 1 in our implementation to treat all the weights equally.

$$W_{ij,k} = (C_{ij,k})^{\omega C} \times (S_{ij,k})^{\omega S} \times (E_{ij,k})^{\omega E} \quad (\omega C = \omega S = \omega E = 1) \tag{6}$$

Since the image merging process takes a weighted average across images, the sum of the weights $W_{ij,k}$ at the same coordinates must equal 1. The normalized weight is defined as $\hat{W}_{ij,k}$. The formula is shown in Eq. (7).

$$\hat{W}_{ij,k} = \left[\sum_{k'=0}^{N-1} W_{ij,k'}\right]^{-1} W_{ij,k} \tag{7}$$

Using the normalized weight, the result synthesized image R is given as Eq. (8).

$$R_{ij} = \sum_{k=0}^{N-1} \hat{W}_{ij,k} I_{ij,k} \tag{8}$$

However, this naive algorithm is influenced easily by noise and does not produce a synthesized image of sufficient quality. Therefore, Mertens et al. also proposed the technique to solve this problem in [1], based on the image pyramid mechanism described in the next section.

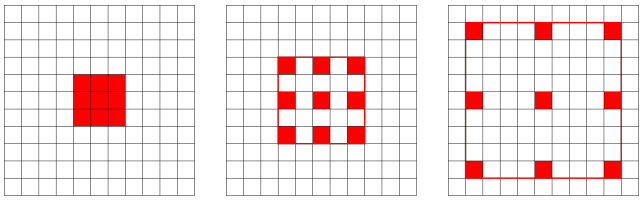


Fig. 2: Dilated Convolution.

2.2 Image Pyramid

An image pyramid consists of a set of images with multiple resolutions from high to low. It is called an image pyramid because it looks like a pyramid when stacked from the bottom to the top, starting with the highest resolution image. Image pyramids are often used to process the same image at various resolutions.

There are two types of image pyramids: Gaussian pyramids and Laplacian pyramids. In this paper, for simplicity of explanation, the first level of the pyramid hierarchy is represented as L1, the second level as L2, and so on, starting from the highest-resolution image. When generating a Gaussian pyramid, the L2 image is generated by applying a Gaussian filter to the L1 image and downsampling the horizontal and vertical resolutions by half, respectively. The L3 and coarser images can be generated in the same way. The Laplacian pyramid can also be generated by taking the difference between the images of the Gaussian pyramid and the upsampled images of their coarser neighbors. The last layer of the Laplacian pyramid is the same image as the last layer of the Gaussian pyramid. This results in the two pyramids having the same number of layers.

The Laplacian pyramid has the property that the original image can be reconstructed by repeatedly upsampling the layer and adding it to the finer next layer, starting from the last level. Mertens' algorithm using image pyramids can achieve high-quality HDR synthesis by generating a Laplacian pyramid of the input image and a Gaussian pyramid of the weights, calculating the weighted average using the two pyramids, and reconstructing the result.

2.3 Dilated Convolution

The image pyramid described in Sec. 2.2 is complex to control in a pipelined implementation because the resolution varies with the hierarchy. Therefore, we introduce dilated convolution [11] to achieve the same effect as the image pyramid described in Sec. 2.2 without changing the resolution of each layer. Dilated convolution refers to the process of performing convolution with spaces inserted between kernel elements, as shown in Fig. 2. By increasing the dilation rate (distance between the closest two elements) instead of downsampling, dilated convolution can take account of global features without reducing the size of the image. It

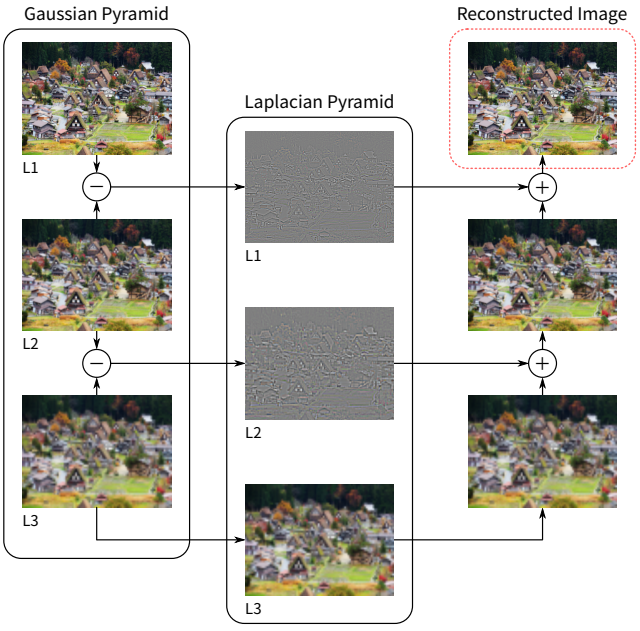


Fig. 3: Image reconstruction from the image pyramids with constant resolution using dilated convolution.

also has the advantage of preventing the loss of information that occurs when reducing an image.

An example of reconstructing the original image from the Laplacian pyramid generated using dilated convolution is shown in Fig. 3. The Gaussian pyramid can be generated using the same 3×3 Gaussian kernel and increasing the dilation rate by a factor of 2 as the layer level increases. Since all layers have the same resolution as L1, no upsampling is required, and the original image can be recovered by simply adding the layers of the Laplacian pyramid in order from the last layer.

3. Real-Time Image Compression

Stream processing using a deep pipeline is an effective way of real-time video processing on FPGA. In the stream processing, stencil computations including convolutions can be implemented using shift registers and line buffers (FIFOs). When applying dilated convolutions, however, many block memory resources (Block RAM, or BRAM) are used as the line buffer. For example, memory size required to apply a 3×3 kernel with dilation = 8 (equivalent to a 17×17 kernel) to 4K (3840 pixels in width) 24-bit RGB images amounts to about 180 KiB. This is a burden to FPGAs, especially for small ones equipped with less than 1 MiB of BRAM.

Introducing some image compression technique would help reduce memory usage of line buffers. However, complicated video codecs such as VP9 [12] and H.265/HEVC [13] cannot be used here because of their high computational cost and latency. Even the VESA Display Stream Compression (DSC) [14], the low-latency image compression standard for video interfaces, consumes tens of thousands of LUTs [15, 16] and thus is not suitable for this purpose.

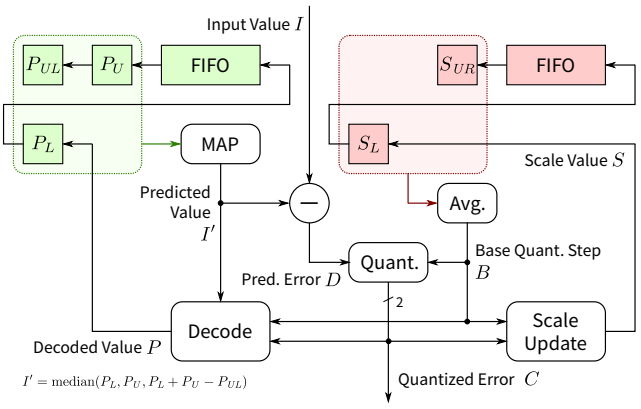


Fig. 4: Overview of the encoder module.

Therefore, we have developed a simple image compression algorithm which can be easily integrated into a pipeline. Latency of the implemented encoder and decoder is 2 clock cycles for each, LUT usage in total for a single color channel is less than 500 when the quantization bit width is 2, and no DSP block is required. Though the modules themselves consume some block memory (equivalent to 2 horizontal lines for each), a benefit of compression exceeds the overhead when a dilation factor is large.

The algorithm is based on adaptive differential pulse-code modulation (ADPCM), combining prediction-based delta encoding and adaptive quantization. First, a value of each input pixel is predicted from preceding pixels (intra-frame prediction), then a difference (delta) between the predicted and actual values is calculated. The difference value is then quantized using a quantization step size which varies adaptively. Combining the two techniques brings both high followability in a rapidly-changing area and high precision in a slowly-changing area with low computational cost. Since human perception is logarithmic, high visual quality can be maintained even with a few quantization bits.

An overview of the encoder module is shown in Fig. 4, and the detailed encoding algorithm is explained in the following sections. The decoding algorithm is almost the same except that the quantized error C is input from outside. Note that the input image is assumed to be represented in an 8-bit unsigned format in this section. For a signed input, we can first convert it to unsigned by adding a bias before encoding and then subtract the bias after decoding.

3.1 Prediction of a Pixel Value

Let the coordinates currently focused on be (v, h), where v and h are vertical and horizontal coordinates with the origin placed at the top-left of an image, respectively. First, the 8-bit pixel value $I(v, h)$ is predicted from 3 adjacent pixels. Since the actual pixel values are not available for the decoder, the decoded values $P_L = P(v, h-1)$, $P_U = P(v-1, h)$, and $P_{UL} = P(v-1, h-1)$ are used in both the encoder and the decoder. The predicted value $I'(v, h)$ is then given using the median adaptive prediction (MAP) [17, 18], as follows:

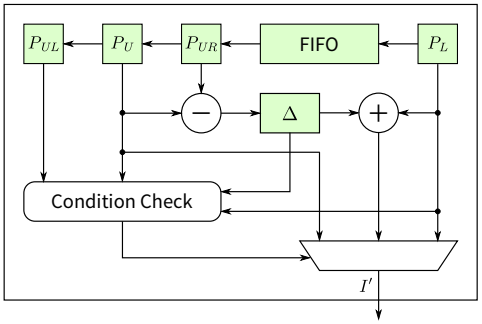


Fig. 5: Prediction circuit of a pixel value.

$$I'(v,h) = \text{median}(P_L, P_U, P_L + P_U - P_{UL}) \tag{9}$$

Values outside the image are regarded as $2^{8-1} = 128$.

The implemented pixel prediction circuit can be illustrated as Fig. 5. The decoded values can be obtained using shift registers and a line buffer. To improve timing, the difference $\Delta = P_U - P_{UL}$ in Eq. (9) is calculated in the previous clock cycle and held in a register.

3.2 Prediction of a Base Quantization Step

In the proposed algorithm, the scale value $S(v,h)$ is assigned to each of the pixels which have already been encoded, as explained later. From the 2 adjacent scale values $S_L = S(v, h-1)$ and $S_{UR} = S(v-1, h+1)$, the base quantization step size $B(v,h)$ is predicted as follows:

$$B(v,h) = \left\lfloor \frac{S_L + S_{UR} + 1}{2} \right\rfloor \tag{10}$$

If S_L and/or S_{UR} do not exist, the following values are used instead. Note that W is the width of the image.

$$S_L = \begin{cases} 16 & (v = h = 0) \\ S_{UR} & (v \neq 0, h = 0) \end{cases} \tag{11}$$

$$S_{UR} = \begin{cases} S_L & (v = 0) \\ S_U & (v \neq 0, h = W-1) \end{cases} \tag{12}$$

This prediction algorithm can be easily implemented using shift registers and a line buffer, as shown in Fig. 4.

3.3 Quantization

In the encoder, the prediction error $D(v,h) = I(v,h) - I'(v,h)$ is calculated and quantized based on the base quantization step $B(v,h)$. Given that the quantization bit width is n (≥ 2), the quantized error $C(v,h)$ ($0 \leq C \leq 2^n - 1$) is calculated as follows:

$$C(v,h) = \text{clip}\left(\left\lfloor \frac{2^{n-2} D(v,h)}{B(v,h)} \right\rfloor + 2^{n-1}, 0, 2^n - 1 \right) \tag{13}$$

Here, $\text{clip}(x,a,b) = \min(\max(x,a),b)$ is the function to clip the value range of x into $[a,b]$. This equation can be visualized as Fig. 6. As can be seen from the figure, the actual size of quantization step is given as $B(v,h)/2^{n-2}$.

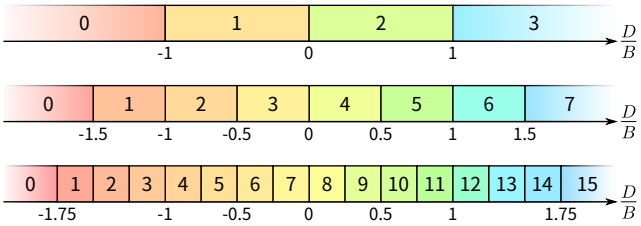


Fig. 6: Relationship among the prediction error D, the base quantization step B, and the quantized error C. The bars correspond to $n = 2, 3, 4$ from top to bottom, respectively.

The division in Eq. (13) is difficult to implement directly. Considering that only 2^n values are possible for C, however, we can calculate all the threshold values of D from B in parallel. Then, C can be calculated without a divider by comparing D with each of the thresholds.

3.4 Decoding

The decoded value $P(v,h)$ is calculated based on the predicted value $I'(v,h)$, the base quantization step $B(v,h)$, and the quantized error $C(v,h)$, as follows:

$$P(v,h) = \text{clip}\left(I'(v,h) + \lfloor B(v,h) \times \mu \rfloor, 0, 255\right) \tag{14}$$

$$\mu = \frac{2C(v,h) - 2^n + 1}{2^{n-1}} \tag{15}$$

The product of B and the scaling factor μ, which corresponds to the center of the distribution range in Fig. 6, is added to the predicted value I' as a correction value, yielding the decoded value $P(v,h)$.

Concurrently with the decoding, the scale value $S(v,h)$ is determined as follows:

$$S(v,h) = \text{clip}\left(\lfloor B(v,h) \times \mu' + 0.5 \rfloor, S_{min}, S_{max}\right) \tag{16}$$

$$\mu' = \begin{cases} 1 + \alpha(|\mu| - 1) & (|\mu| > 1) \\ 1 + \beta(|\mu| - 1)/2^n & (|\mu| \leq 1) \end{cases} \tag{17}$$

Eq. (17) means that the scale value is increased when the absolute scaling factor $|\mu| > 1$, and decreased when $|\mu| \leq 1$. As can be seen in Fig. 6, a larger quantization bit width n makes it possible to represent smaller errors even if the base quantization step size is the same. Therefore, the scale factor is decreased slowly when n is large. $\alpha = 1.5$ and $\beta = 2$ are the pre-defined hyperparameters based on preliminary evaluation results. Using these values, Eq. (17) can be rewritten as follows:

$$\mu' = \begin{cases} (3|\mu| - 1)/2 & (|\mu| > 1) \\ 1 + (|\mu| - 1)/2^{n-1} & (|\mu| \leq 1) \end{cases} \tag{18}$$

To maintain the followability to an input and avoid an overflow, the range of the scale value $[S_{min}, S_{max}]$ in Eq. (16) is defined as follows:

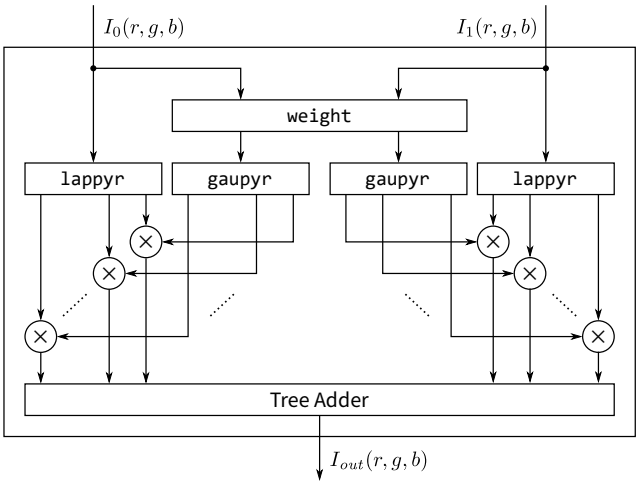


Fig. 7: `hdr_top` module.

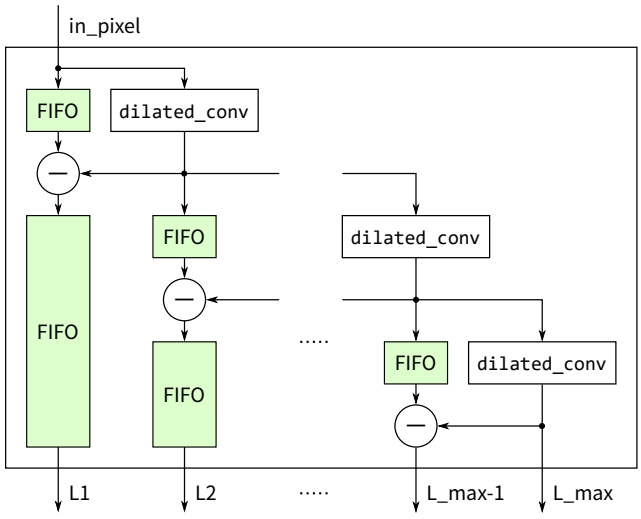


Fig. 8: `lappyr` module.

$$S_{min} = 2^{n-1}, \quad S_{max} = \left\lfloor \frac{255 \times 2^{n-1}}{2^n - 1} \right\rfloor$$

These values are set so that the product of B and $|\mu|$ is restricted within the range of $[1, 255]$.

In the hardware implementation, Eq. (14) and (16) are calculated for all the possible values of μ in parallel to improve timing. Then, one of the candidate results is chosen using multiplexers after C is prepared.

4. Hardware Design

This section describes details of the FPGA implementation of the HDR synthesis method shown in Sec. 2 combined with the image compression technique introduced in Sec. 3.

4.1 HDR Synthesis Module

An overview of the HDR synthesis module (`hdr_top`) is shown in Fig. 7. The number of input images can be changed using a parameter, and it is set to 2 in this paper. The module takes as input the pixel values $I_0(r,g,b)$ and $I_1(r,g,b)$ of two differently exposed images, calculates 10-bit weights for both in the `weight` module, and generates image pyramids in the `gaupyr` and `lappyr` modules. Then, the synthesized image $I_{out}(r,g,b)$ is reconstructed from the weighted average of the Laplacian pyramids using the tree adder.

4.2 Image Pyramid Generation Module

An overview of the Laplacian pyramid (`lappyr`) module is shown in Fig. 8. The `lappyr` module applies the Gaussian filter with dilated convolution in the `dilated_conv` module, and then subtracts the output from the previous layer to generate the Laplacian pyramid with the same resolution in all the layers. In the last layer, the result of `dilated_conv` is directly output, as explained in Sec. 2.2. Also, the `gaupyr` module is the same as the `lappyr` module in Fig. 8 except

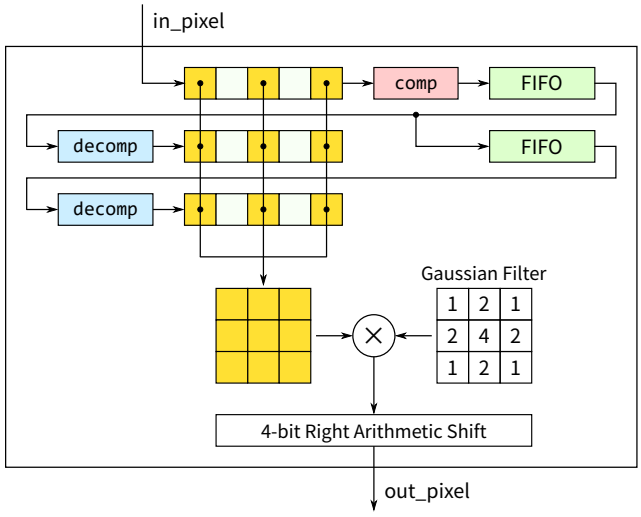


Fig. 9: `dilated_conv` module.

for the calculation of the difference, and generates the Gaussian pyramid with the same latency. In both modules, an appropriately-sized FIFO is placed just before the output of each layer to match the output timings of all the layers.

4.3 Dilated Convolution Module

An overview of the `dilated_conv` module is shown in Fig. 9 with the dilation rate of 2 to simplify the explanation. As described in Sec. 2.3, the dilation rate doubles as the layer level increases. This causes an exponential increase in BRAM usage and restricts the maximum number of layers. In this paper, we add the image compression and decompression (`comp`/`decomp`) modules to FIFOs to reduce the amount of BRAM usage. However, the `comp`/`decomp` module itself also requires a FIFO for pixel value prediction. Considering this overhead, the compression is used only after the fifth layer, where the dilation rate is 8 or higher. Similarly, the

FIFO of the image pyramid generation module described in Sec. 4.2 is also compressed only when the size exceeds the threshold.

5. Evaluation and Discussion

In this section, we evaluate and discuss the HDR synthesis system with the image compression technique shown in Sec. 3 and Sec. 4. We first evaluate the effect of the image compression on resource utilization, processing performance, and image quality respectively in Sec. 5.1, 5.2, and 5.3. Since the image compression allows the system to have more layers, we also evaluate how the number of layers affects the image quality in Sec. 5.4.

The target FPGA is a Zynq UltraScale+ MPSoC XCZU9EG-FFVB1156-2-E, and the evaluation board is the ZCU102 Evaluation Kit. The entire system is described in RTL using SystemVerilog and is implemented using Vivado 2019.2. For the evaluation, we use simulation results from Cadence Xcelium.

5.1 Resource Utilization

We evaluate how the compression affects resource utilization of the hardware implementation of HDR synthesis. Table 1 and 2 show the resource utilization without and with the compression, respectively. The amount of DSP utilization is omitted because it was 6 (0.24 %) in all the configurations. For configurations which cannot be implemented because of overutilization of BRAM, we show estimated utilization after the logic synthesis. The results show that in all the configurations, the application of compression reduces the amount of BRAM utilization, and the reduction rate increases as the number of layers increases. For example, at 7 layers of 4K (3840×2160), the amount of BRAM utilization could be reduced from 168.48 % to 84.32 %. Utilization ratios of LUT and FF increased from 11.16 % to 20.03 % and from 4.84 % to 7.48 %, respectively, due to the addition of the compression and decompression modules. However, none of the resource utilization rates exceeded that of BRAM, indicating that compression improved the balance of resource utilization, which had been a problem. As a result, the maximum number of layers is improved from 6 to 7 at 4K and from 7 to 8 at Full HD (1920×1080). We will evaluate how this contributes to the HDR quality later in Sec. 5.4.

5.2 Processing Performance

We evaluate how the compression affects processing performance of the hardware implementation of HDR synthesis. Values of Fmax and latency for each configuration are shown in Tables 1 and 2. We can find a tendency that Fmax decreases as the resolution and the number of layers increase. However, almost all the configurations can achieve over 200 MHz of Fmax. In addition, the compression does not cause clear degradation in Fmax. Specifically, Fmax is about 209.2 MHz even with 7 layers of 4K with compression, which is equivalent to a frame rate of about 25 fps. At Full HD, a frame rate reaches around 100 fps. On the other hand, latency is almost proportional to the image width and almost doubles as the number of layers increases. However, it is not affected by the use of compression, either. For example, the latency of the 7-layer 4K configuration is 245894 clock cycles, equivalent to about 1.2 μs when operating at the Fmax.

5.3 Image Quality with Compression

To evaluate how the image compression affects the image quality, we calculated SSIM (Structural Similarity) [19] and HDR-VDP 3.07 (PPD: 26.36) [20] of two synthesized images with or without the compression using 2 test images (*house*, *dress*). SSIM is a metric of similarity between two images. Let $\boldsymbol{x}$ and $\boldsymbol{y}$ be a vector whose elements are pixel values, then SSIM is represented by Eq. (19). $K_1 = 0.01$, $K_2 = 0.03$, $L = 255$, $C_1 = (K_1 L)^2$, $C_2 = (K_2 L)^2$ are the pre-defined parameters. $\mu_{\boldsymbol{x}}$ and $\mu_{\boldsymbol{y}}$ are the mean values of $\boldsymbol{x}$ and $\boldsymbol{y}$, $\sigma_{\boldsymbol{x}}$ and $\sigma_{\boldsymbol{y}}$ are the standard deviations of $\boldsymbol{x}$ and $\boldsymbol{y}$. $\sigma_{\boldsymbol{x},\boldsymbol{y}}$ is the covariance of $\boldsymbol{x}$ and $\boldsymbol{y}$, and the value range of SSIM is $(0, 1]$. The more similar the input images are, the larger the SSIM becomes. If the SSIM is 0.98 or higher, it is indistinguishable from the original image, and if it is 0.90 or lower, it is clearly distinguishable from the original image.

$$SSIM\,(\boldsymbol{x}, \boldsymbol{y}) = \frac{\left(2\mu_{\boldsymbol{x}}\mu_{\boldsymbol{y}} + C_1\right)\left(2\sigma_{\boldsymbol{xy}} + C_2\right)}{\left(\mu_{\boldsymbol{x}}^2 + \mu_{\boldsymbol{y}}^2 + C_1\right)\left(\sigma_{\boldsymbol{x}}^2 + \sigma_{\boldsymbol{y}}^2 + C_2\right)} \tag{19}$$

HDR-VDP is a similar metric, but is based on a more comprehensive visual model for all luminance conditions. As a result, HDR-VDP is compatible with not only SDR but also HDR images, and shows good correlation to human perception. The maximum quality value of HDR-VDP is 10, and the value can be negative if the difference between two images are very large.

For each number of layers from 5 to 8, we performed HDR synthesis with and without the compression, and calculated SSIM between the two results. For reference, we also tested the result of the software implementation which uses floating-point arithmetic and uses the non-approximated saturation weight (Eq. (3)). The values of SSIM are summarized in Table 3, and the result images as well as their histograms of luminance are shown in Fig. 10 and 11.

Comparing the result values between with and without the compression at each number of layers, we can find that the values decrease as the number of layers increases. This is thought to be due to the fact that compression is not performed for small FIFOs, and as the number of layers increases, the proportion of the compressed FIFOs increases. However, the SSIM values exceed 0.95 even at 8 layers, indicating that the image compression has almost negligible impact on the image quality. Also, comparing the SSIM values vs. the software implementation, it can be seen that fixed-point arithmetic and the approximation of the saturation weight has little impact on the image quality. Comparing

Table 1: Resource utilization in HDR synthesis without compression.

	VGA (640×480)			HD (1280×720)			FHD (1920×1080)			4K (3840×2160)			
Layers	6	7	8	6	7	8	6	7	8	6	7	8	Available
LUT	17622 (6.43%)	21618 (7.89%)	26899 (9.81%)	18647 (6.80%)	23521 (8.58%)	29050 (10.60%)	20147 (7.35%)	26183 (9.55%)	34989 (12.77%)	22464 (8.20%)	30599 (11.16%)	71185 (25.97%)	274080
LUT-RAM	1329 (0.92%)	1766 (1.23%)	2568 (1.78%)	1323 (0.92%)	1789 (1.24%)	2696 (1.87%)	1593 (1.11%)	2105 (1.46%)	3240 (2.25%)	1597 (1.11%)	2309 (1.60%)	3323 (2.31%)	144000
FF	19443 (3.55%)	22869 (4.17%)	26566 (4.85%)	19685 (3.59%)	23206 (4.23%)	27977 (5.10%)	20345 (3.71%)	24012 (4.38%)	30253 (5.52%)	20672 (3.77%)	26554 (4.84%)	61630 (11.24%)	548160
BRAM	150 (16.45%)	292 (32.02%)	588 (64.47%)	265.5 (29.11%)	541.5 (59.38%)	1107 (121.38%)	376.5 (41.28%)	788.5 (86.46%)	1664 (182.46%)	721 (79.06%)	1536.5 (168.48%)	3282.5 (359.92%)	912
Fmax	243.8	232.3	225.6	271.5	196.3	-	241.7	216.4	-	212.4	-	-	[MHz]
Latency	20596	41094	82124	41056	82054	164044	61536	123014	245964	122976	245894	491724	[clock cycles]

Table 2: Resource utilization in HDR synthesis with compression.

	VGA (640×480)			HD (1280×720)			FHD (1920×1080)			4K (3840×2160)			
Layers	6	7	8	6	7	8	6	7	8	6	7	8	Available
LUT	39137 (14.28%)	50675 (18.49%)	63367 (23.12%)	39288 (14.33%)	52338 (19.10%)	65211 (23.79%)	40280 (14.70%)	53477 (19.51%)	66949 (24.43%)	42058 (15.35%)	54903 (20.03%)	69648 (25.41%)	274080
LUT-RAM	1710 (1.19%)	2005 (1.39%)	2990 (2.08%)	1440 (1.00%)	2112 (1.47%)	3141 (2.18%)	1611 (1.12%)	2328 (1.62%)	3369 (2.34%)	1755 (1.22%)	2410 (1.67%)	3553 (2.47%)	144000
FF	30189 (5.51%)	37928 (6.92%)	46115 (8.41%)	29548 (5.39%)	39250 (7.16%)	47882 (8.74%)	30468 (5.56%)	39934 (7.29%)	48333 (8.82%)	32306 (5.89%)	40998 (7.48%)	49840 (9.09%)	548160
BRAM	131 (14.36%)	198 (21.71%)	378 (41.45%)	225.5 (24.73%)	399.5 (43.80%)	585.5 (64.20%)	277.5 (30.43%)	434.5 (47.64%)	674.5 (73.96%)	510 (55.92%)	769 (84.32%)	1296.5 (142.16%)	912
Fmax	230.0	222.0	230.3	235.2	232.5	196.6	219.4	227.3	200.7	228.5	209.2	-	[MHz]
Latency	20596	41094	82124	41056	82054	164044	61536	123014	245964	122976	245894	491724	[clock cycles]

Table 3: Quality evaluation results.

		SSIM			HDR-VDP-3			MEF-SSIMd		
Image	Layers	software w/o comp	software w/ comp	w/o comp w/ comp	software w/o comp	software w/ comp	w/o comp w/ comp	software	w/o comp	w/ comp
house	5	0.998	0.981	0.979	9.965	9.494	9.523	0.957	0.958	0.939
	6	0.998	0.975	0.974	9.935	9.362	9.398	0.965	0.965	0.945
	7	0.998	0.969	0.968	9.913	9.257	9.301	0.973	0.972	0.951
	8	0.997	0.962	0.961	9.812	9.124	9.212	0.969	0.970	0.946
dress	5	0.999	0.978	0.978	9.959	9.303	9.320	0.975	0.975	0.963
	6	0.999	0.973	0.973	9.921	9.145	9.177	0.986	0.986	0.972
	7	0.997	0.967	0.969	9.702	8.930	9.072	0.992	0.991	0.975
	8	0.994	0.961	0.964	8.904	8.426	9.008	0.994	0.992	0.975

the respective histograms in Fig. 11, there is no significant change caused by the compression. The above results show that the application of image compression can reduce resource usage with little degradation in image quality.

5.4 Number of Layers and Image Quality

Fig. 10 and 11 show the evaluation results of the change in HDR synthesis quality when the number of layers is varied from 6 to 8. Fig. 11 shows that the 6-layer configuration produces unnatural gradations (halo) at the boundary between the building and the sky, but the 8-layer configuration improves the gradations and produces more natural results. For quantitative evaluation, we also show the values of MEF-SSIMd [21,22], an image quality assessment metric for multi-exposure fusion images based on the structural similarity approach, in Table 3. According to the result, deeper configurations give better quality in terms of MEF-SSIMd, except for the slight decrease of the image *house* found in the 8-layer configurations.

From the above, it is confirmed that the increase in the maximum number of layers brought by the introduction of image compression in this study contributes to the improvement in quality. On the other hand, as the number of layers increases, a thin band of noise is observed at the periphery of the image. This is because the padding algorithm for the Gaussian filtering in the hardware implementation is different from that of the software to reduce hardware complexity. As shown in Fig. 11, the band-shaped noise is more prominent in the image *dress*, which explains the lower values of HDR-VDP-3 when compared to the software implementation at 8 layers.

6. Conclusion

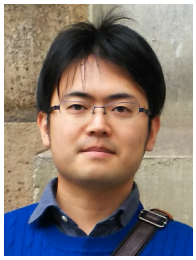
We proposed a method to reduce BRAM usage by incorporating image compression technology into the pipeline of FPGA-based real-time HDR synthesis hardware, and evaluated the impact of this method on resource utilization and HDR synthesis quality. Since the image compression based on ADPCM can be calculated using only the values of adjacent pixels and does not require complex operations, the

image compression module can be compactly integrated into the FIFO of the dilated convolution used for HDR synthesis processing easily, thereby improving the balance of resource consumption. Specifically, in the case of 4K (3840×2160) and 7 layers, the resource usage of LUT and FF increased from 11.16 % to 20.03 % and from 4.84 % to 7.48 %, while the BRAM was reduced from 168.48 % to 84.32 %. As a result, the number of layers that can be implemented increased from 6 to 7, yielding better visual quality. To achieve a higher number of layers, it is essential to apply further resource reduction methods. Regarding the processing performance, the frame rate reached about 25 fps at 4K and about 100 fps at FHD. Based on these results, the system can be applied to a wide range of applications.

References

[1] T. Mertens, J. Kautz, and F.V. Reeth, "Exposure Fusion," Proc. Pacific Conference on Computer Graphics and Applications, PG '07, USA, p.382–390, 2007.
[2] F. Hassan and J.E. Carletta, "An FPGA-Based Architecture for a Local Tone-Mapping Operator," Journal of Real-Time Image Processing, vol.2, pp.293–308, 2007.
[3] V. Popovic, E. Pignat, and Y. Leblebici, "Performance Optimization and FPGA Implementation of Real-Time Tone Mapping," IEEE Transactions on Circuits and Systems II: Express Briefs, vol.61, no.10, pp.803–807, 2014.
[4] P.J. Lapray, B. Heyrman, and D. Ginhac, "HDR-ARtiSt: An Adaptive Real-time Smart Camera for High Dynamic Range Imaging," Journal of Real-Time Image Processing, vol.12, no.4, pp.747–762, 2016.
[5] P. Ambalathankandy, A. Horé, and O. Yadid-Pecht, "An FPGA Implementation of a Tone Mapping Algorithm with a Halo-Reducing Filter," Journal of Real-Time Image Processing, vol.16, pp.1317–1333, 2019.
[6] S. Nosko, M. Musil, P. Zemcik, and R. Juranek, "Color HDR Video Processing Architecture for Smart Camera," Journal of Real-Time Image Processing, vol.17, pp.555–566, 2020.
[7] M. Ikebe, P. Ambalathankandy, and Y. Ou, "HDR Tone Mapping: System Implementations and Benchmarking," ITE Transactions on Media Technology and Applications, vol.10, no.2, pp.27–51, 2022.
[8] M.A. Sangiovanni, F. Spagnolo, and P. Corsonello, "Hardware-Oriented Multi-Exposure Fusion Approach for Real-Time Video Processing on FPGA," 2022 17th Conference on Ph.D Research in Microelectronics and Electronics (PRIME), pp.129–132, 2022.
[9] T. Katayama, Y. Imamura, T. Manabe, and Y. Shibata, "Application of Extended Convolution to High Dynamic Range Synthesis Processing in FPGA," IEICE Technical Report, vol.121, no.59, pp.50–55, 2021.
[10] M. Nishimura, Y. Imamura, T. Manabe, and Y. Shibata, "FPGA Implementation of HDR Synthesis Processing with Image Compression Techniques," Proc. 2022 International Conference on Field-Programmable Technology (ICFPT), pp.1–2, 2022.
[11] F. Yu and V. Koltun, "Multi-Scale Context Aggregation by Dilated Convolutions," Proc. International Conference on Learning Representations (ICLR), 2016.
[12] The WebM Project, "VP9 Video Codec Summary." https://www.webmproject.org/vp9/.
[13] International Telecommunication Union, "ITU-T Recommendation H.265: High efficiency video coding." https://www.itu.int/rec/T-REC-H.265.
[14] Video Electronics Standards Association (VESA), "VESA Display Compression Codecs." https://vesa.org/vesa-display-compression-codecs/.
[15] Hardent Inc., "Display Stream Compression (VESA DSC) 1.2a Encoder IP Core For Xilinx FPGAs." https://www.xilinx.com/products/intellectual-property/1-14a3qhz.html.
[16] Hardent Inc., "Display Stream Compression (VESA DSC) 1.2a Decoder IP Core For Xilinx FPGAs." https://www.xilinx.com/products/intellectual-property/1-133tc21.html.
[17] S.A. Martucci, "Reversible Compression of HDTV Images Using Median Adaptive Prediction and Arithmetic Coding," Proc. IEEE International Symposium on Circuits and Systems (ISCAS), pp.1310–1313, 1990.
[18] M.J. Weinberger, G. Seroussi, and G. Sapiro, "LOCO-I: A Low Complexity, Context-Based, Lossless Image Compression Algorithm," Proc. Data Compression Conference (DCC), pp.140–149, 1996.
[19] Z. Wang, A. Bovik, H. Sheikh, and E. Simoncelli, "Image Quality Assessment: From Error Visibility to Structural Similarity," IEEE Transactions on Image Processing, vol.13, no.4, pp.600–612, 2004.
[20] R.K. Mantiuk, D. Hammou, and P. Hanji, "HDR-VDP-3: A Multi-Metric for Predicting Image Differences, Quality and Contrast Distortions in High Dynamic Range and Regular Content." https://arxiv.org/abs/2304.13625, 2023.
[21] K. Ma, K. Zeng, and Z. Wang, "Perceptual Quality Assessment for Multi-Exposure Image Fusion," IEEE Transactions on Image Processing, vol.24, no.11, pp.3345–3356, 2015.
[22] Y. Fang, H. Zhu, K. Ma, Z. Wang, and S. Li, "Perceptual Evaluation for Multi-Exposure Image Fusion of Dynamic Scenes," IEEE Transactions on Image Processing, vol.29, pp.1127–1138, 2020.

Masahiro Nishimura Masahiro Nishimura graduated from Nagasaki University, Japan, in 2022. Now he is a master student at the Graduate School of Engineering, Nagasaki University. His research interests include image processing with an FPGA.



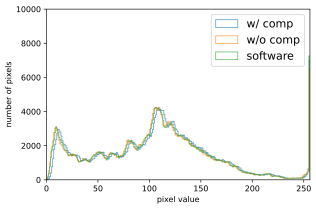
Taito Manabe Taito Manabe received the B.E, M.E, and Ph.D. degrees from Nagasaki University, Japan, in 2016 and 2018, and 2021, respectively. Now he is an assistant professor at School of Information and Data Sciences, Nagasaki University. His research interests include real-time processing with an FPGA.

Yuichiro SHIBATA Yuichiro Shibata received the B.E. degree in electrical engineering, the M.E. and Ph.D. degrees in computer science from Keio University, Japan, in 1996, 1998 and 2001, respectively. Currently, he is a professor at School of Information and Data Sciences, Nagasaki University. He was a Visiting Scholar at University of South Carolina in 2006. His research interests include reconfigurable systems and parallel processing.

(a) 6 layers with compression

(b) 6 layers without compression

(c) 6 layers software

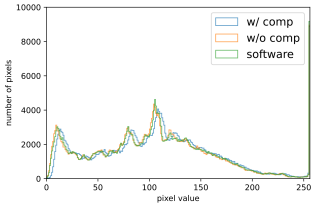


(d) Histogram at 6 layers

(e) 7 layers with compression

(f) 7 layers without compression

(g) 7 layers software

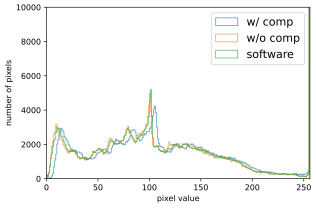


(h) Histogram at 7 layers

(i) 8 layers with compression

(j) 8 layers without compression

(k) 8 layers software



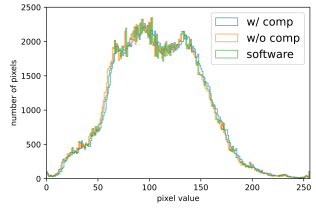
(l) Histogram at 8 layers

Fig. 10: HDR synthesis results (house)

(a) 6 layers with compression

(b) 6 layers without compression

(c) 6 layers software

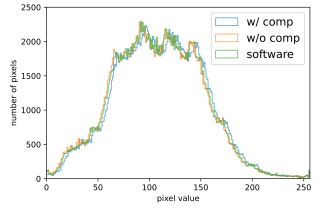


(d) Histogram at 6 layers

(e) 7 layers with compression

(f) 7 layers without compression

(g) 7 layers software

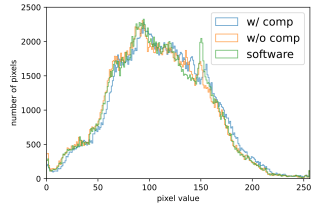


(h) Histogram at 7 layers

(i) 8 layers with compression

(j) 8 layers without compression

(k) 8 layers software



(l) Histogram at 8 layers

Fig. 11: HDR synthesis results (dress)