

Article

# HLS Based Approach to Develop an Implementable HDR Algorithm

Rappy Saha , Partha Pratim Banik and Ki-Doo Kim \*

School of Electronics Engineering, Kookmin University, Seoul 02707, Korea; rappysaha@kookmin.ac.kr (R.S.); ppbanik006@kookmin.ac.kr (P.P.B.)

\* Correspondence: kdk@kookmin.ac.kr; Tel.: +82-2-910-4707

Received: 20 September 2018; Accepted: 15 November 2018; Published: 19 November 2018



**Abstract:** Hardware suitability of an algorithm can only be verified when the algorithm is actually implemented in the hardware. By hardware, we indicate system on chip (SoC) where both processor and field-programmable gate array (FPGA) are available. Our goal is to develop a simple algorithm that can be implemented on hardware where high-level synthesis (HLS) will reduce the tiresome work of manual hardware description language (HDL) optimization. We propose an algorithm to achieve high dynamic range (HDR) image from a single low dynamic range (LDR) image. We use highlight removal technique for this purpose. Our target is to develop parameter free simple algorithm that can be easily implemented on hardware. For this purpose, we use statistical information of the image. While software development is verified with state of the art, the HLS approach confirms that the proposed algorithm is implementable to hardware. The performance of the algorithm is measured using four no-reference metrics. According to the measurement of the structural similarity (SSIM) index metric and peak signal-to-noise ratio (PSNR), hardware simulated output is at least 98.87 percent and 39.90 dB similar to the software simulated output. Our approach is novel and effective in the development of hardware implementable HDR algorithm from a single LDR image using the HLS tool.

**Keywords:** field-programmable gate array; high-dynamic range image; high-level synthesis; low-dynamic range image; system on chip

## 1. Introduction

Vision is one of the most important senses. The things observed by us are analyzed by the brain to help us in taking decision. When we talk about modern science, the camera is an instrument that is analogous to eyes but the capability of eyes is far better than the camera. A camera can capture a moment through an image or video and store it. However, camera by itself cannot take a decision and, for that purpose, camera needs a system similar to the human brain to analyze the data. A field-programmable gate array (FPGA) can be used in a camera as a real-time system to analyze the captured image. The language for FPGA is related to the way of resource implementation in the FPGA. In the case of software programming, we do not need to think about resource consumption. This fact limits the algorithmic computational complexity in FPGA. Hence, research has been conducted on different types of computational acceleration techniques [1,2]. Image classification [1], real-time anomaly detection of hyperspectral images [3], synthetic aperture imaging [4], feature detection for image analysis [5], bilateral filter design for real-time image denoising [6], and panorama real-time video system with high-speed image distortion correction [7] are FPGA-based implementations in the field of image processing.

HLS has become popular in recent years because of its design performance, low complexity, and reduced product development time [8]. HLS connects hardware description languages (HDL),

e.g., VHSIC hardware description language (VHDL) and Verilog, to high-level languages (HLL), e.g., C/C++. In simple terms, it converts HLL to HDL with optimization techniques. In [8], Nane et al. conducted a survey of HLS tools and compared their optimization techniques, e.g., operation chaining, bit-width analysis and optimization, memory space allocation, loop optimization, hardware resource library, speculation, and code notion, thereby exploiting spatial parallelism and if-conversion. As HLS brings relationship between HLL to HDL, we see a feasibility to develop an implementable algorithm using HLS. Therefore, we convert and optimize our proposed algorithm for the FPGA implementation using HLS.

Dynamic range of an image depends on the exposure quality and visual quality of the scene. However, highlight hides the perfect information about the surface of an image. It adds extra difficulty to any image processing algorithm. Highlights correspond to regions in an image where the light intensity is so high that we cannot see the object behind it. Active light sources, e.g., sun, light emitting diodes (LEDs), and tube light, are not included in this definition. According to Lee [9], the highlight parts are the combination of diffuse reflection and specular reflection where specular reflection dominates. Based on this point, Shafer introduced the dichromatic reflection model [10]. The total reflection,  $R$ , from an inhomogeneous object is the sum of the two independent parts: light reflected from the surface,  $R_s$ , and light reflected from the body,  $R_b$ , shown in Equation (1). Inhomogeneous objects include varnishes, paper, plastics, and ceramics, while homogeneous objects are polished objects, e.g., metals and diamonds.

$$R = R_s + R_b \quad (1)$$

Based on the dichromatic model, Ren et al. defined the illumination chromaticity [11]. They estimated illumination chromaticity from a novel idea of color line constraint to remove highlight from an image [11]. In Figure 1, we mark three highlight areas where the dynamic range of the object is considerably low due to severe illumination. Our primary objective is to remove the highlights and recover the missing information, which results in local improvement of the dynamic range and enhanced global visibility of the image.



**Figure 1.** Target highlight areas to improve dynamic range (red box).

In our previous papers [12,13], we proposed a new model of single LDR image to HDR image generation by highlight removal technique and described an HLS based implementation scenario, respectively. However, our previous technique performance completely depended on four parameters. This parameter dependency limited our technique in the case of hardware implementation. To remove such kind of limitation, in this paper, we describe a new technique based on the previous concept. Our new method is parameter free, which makes it more robust. Although our final target is complete SoC implementation, we describe a significant portion of our implementation here by using HLS tool. From this explanation, we can claim that our method is implementable to the hardware that increases the acceptability of our algorithm. Our proposed HDR algorithm removes the highlights from the image and recovers image information, e.g., color and texture. The main assumptions of the algorithm are: (a) the pixels are not fully saturated; and (b) the surface is inhomogeneous. At first, we detect the highlight area (HA) using statistical information of an image. We modify the HA depending on the information of non-highlight area (NHA). Finally, we improve the global brightness since highlight-free

(HF) image is dark type image. For hardware implementation, our target is system on chip (SoC) based implementation. Here, we elaborate on the programmable logic (PL) side development. Finally, we evaluate our method from various point of views using no-reference [14–17] and full-reference metrics [18,19].

The remainder of this paper is organized as follows: Section 2 discusses related work; Section 3 describes the proposed algorithm; Section 4 describes hardware development; Section 5 presents the results of our software and hardware evaluation; and Section 6 concludes this study.

## 2. Related Works

Tan et al. [20] proposed the idea of intensity logarithmic differentiation to remove highlight iteratively from the input image by comparing it between input image and its specular free image [20]. Yoon et al. [21] explained that diffuse reflection component of a non-saturated input image under the uniform illumination could be extracted by comparing the local ratios of input image and the specular-free two-band image [21]. Shen et al. [22] proposed an algorithm based on the error analysis of chromaticity to separate reflections [22]. Shen et al. [23] described another method by adding offset to modified specular free (MSF) image, whereas MSF chromaticity closes to the diffuse chromaticity [23]. Yang et al. [24] removed highlight from image with bilateral filter by propagating maximum diffuse chromaticity values from diffuse pixels to specular pixels [24].

Researchers introduced several algorithms to produce HDR image from a single low-dynamic range (LDR) image [25–28]. Reinhard et al. [25] described dodging and burning based tone-mapping method in high and low contrast region of LDR image. Dodging and burning is a printing approach to withhold or add light to a portion of an image [25]. Rempel et al. [26] boosted the dynamic range of images for viewing in HDR displays by using reverse tone-mapping algorithm [26]. Banterle et al. [27] introduced a new framework of inverse tone-mapping operator for boosting up the LDR image to HDR image by linear interpolation of original LDR image. Huo et al. [28] showed that linear expansion of HF LDR image can expand the dynamic range of LDR image. They developed highlight removal technique by the help of principal component analysis (PCA) and polynomial transformation. However, all of these methods target software analysis only, which does not guarantee real-time FPGA implementation. Our aim is to develop an algorithm as well as make sure that the algorithm is implementable to the hardware.

Vonikakis et al. [29] presented an image enhancement-based HDR imaging technique. They stretched the luminance value of every frame by building a pipelined structure and implemented their algorithm in Altera's Stratix II. Stretching the luminance value helps in adding more brightness to the resultant image, especially in the underexposed regions of the image, although it can also affect the overexposed regions. Multi-frame-based implementations were usually adopted to get HDR imaging [30,31]. Some researchers also implemented algorithm from dual-camera settings to increase the frame rate [32]. From the point of view of implemented work for HDR algorithm, the novelty of our work is that HDR images are generated from a single image. Besides, we describe the implementation scenario for single image, while others focus on multi-frame implementation.

The applicability and reliability of the HLS are discussed in [33–37]. Tambara et al. [33] analyzed the utilization and performance of HLS-based optimization techniques, e.g., pipelining, loop unrolling, array partitioning, and function inlining. These techniques are used in three different combinations on matrix multiplication (MM), advanced encryption standard (AES), and adaptive differential pulse code modulation (ADPCM). Choi et al. [34] measured the performance of different applications, e.g., Qsort, Log reg, Mat mul, and ConvNN using HLS-based coding. Li et al. [35] focused on multi-loop optimization technique in an algorithmic way for applications such as image segmentation, denoising, edge minimization, and matrix multiplication. A data acquisition system was built based on HLS using finite impulse response (FIR) filtering by CERN researchers [36]. Daud et al. [37] used an HLS-based approach to develop an intellectual property (IP) of glucose–insulin analysis. An IP is a package of HDL coding that can be used directly in system-level register transfer logic (RTL) design. Thus far,

HLS-related research has mainly focused on the performance estimation of pre-built algorithms [33–35]. In [36,37], the authors presented application based works, but none are related to the image processing application. The most novel aspect of our work is the development of a single image HDR algorithm by HLS-based implementation in hardware.

### 3. Proposed Method

The target of our algorithm is to make it simple while being able to produce a competitive result. The simplicity will help us for efficient implementation in the hardware. The algorithm is described by the block diagram in Figure 2.

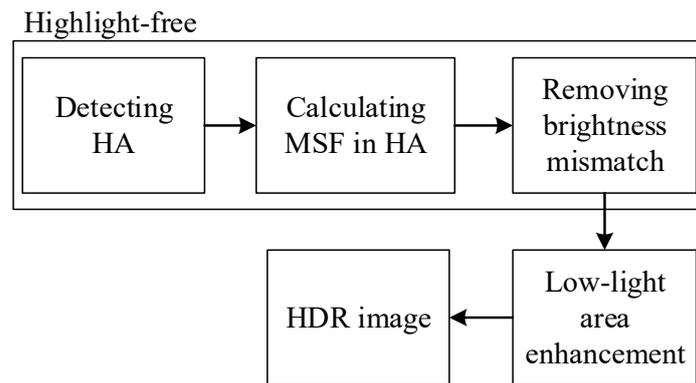


Figure 2. Proposed LDR to HDR generation algorithm.

#### 3.1. Highlight Detection and Modification

$P_i(x, y)$  is the input image.  $(x, y)$  indicates the input pixel location. According to previous researches [20,22–24,28], the minimum channel value  $P_{\min}(x, y)$  is used for highlight detection from the assumption that HA is not fully saturated.  $P_{\min}(x, y)$  can be expressed as follows:

$$P_{\min}(x, y) = \min(P_i(x, y)) \tag{2}$$

The highlight is simply detected by comparing in the following way,

$$P_{\min}(x, y) > 2 \times \overline{P_{\min}}, \text{ highlight else non-highlight} \tag{3}$$

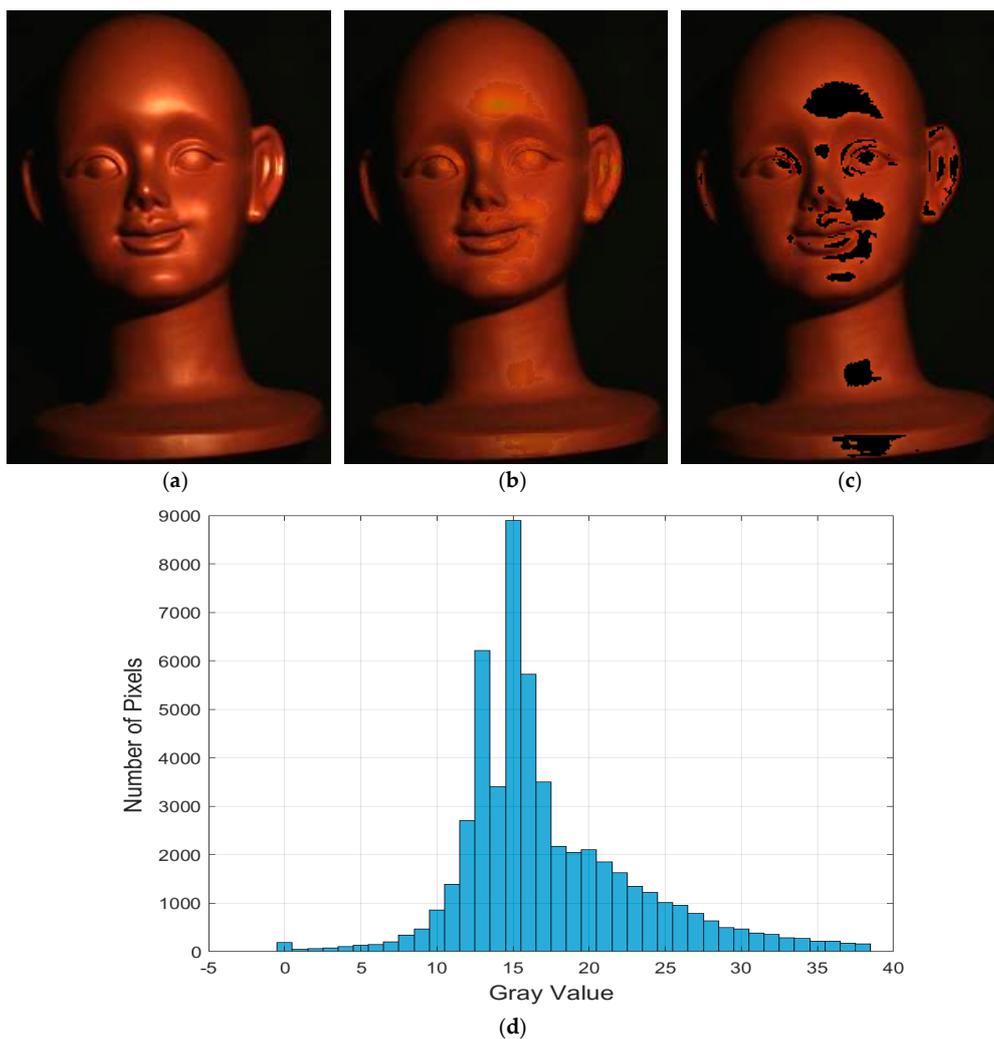
We experimentally set this condition. For all test images, it detects the HA properly. We detect the HA to work with only that region. We assume that other parts of the image contain properly diffused pixels. In the HA, we modify the highlight pixels by Equations (4) and (5). We call this image as MSF image.

$$P_{MSF,i,C}(x, y) = P_{i,C}(x, y) - P_{\min}(x, y) + C_{MSF}, i \in \text{RGB channel and } C \in \text{Pixels belong to HA} \tag{4}$$

$$C_{MSF} = \overline{P_{\min,NHA}} + P_{SD,NHA}^{\min} \tag{5}$$

First, we reduce the  $P_{\min}(x, y)$  from each HA pixel,  $P_{i,C}(x, y)$ . This is called specular free (SF) image [22,23,28]. SF image is usually dark and texture is not rich enough. Therefore, we add an offset  $C_{MSF}$  to  $P_{MSF,i,C}(x, y)$ . Since we reduce the  $P_{\min}(x, y)$  from HA and  $P_{\min}(x, y)$  in HA area is usually higher, it is more logical to calculate the reasonable portion of  $P_{\min}(x, y)$  for addition. For extracting the appropriate diffuse information of HA, we have to consider the  $P_{\min}$  of NHA because NHA is the diffuse area. However, we do not yet know the appropriate diffuse intensity of HA but we know about the area where we can find it. Thus, we can take the average of  $P_{\min}$  of NHA ( $\overline{P_{\min,NHA}}$ ) to use it for extracting the diffuse information of HA. However, NHA is comprised of object and background area. Due to the highlight, the background goes darker during capturing by LDR image sensor. It is

a general characteristic of LDR camera. Because of this characteristic of LDR image sensor, we can say that NHA is darker area and  $\overline{P_{\min,NHA}}$  will also become a low intensity than the appropriate diffuse value of HA. As  $\overline{P_{\min,NHA}}$  is the average of NHA, it brings the diffuse information of object and background surface of the image. From the results in Figure 3d, we can say that the pixel distribution is close to Gaussian distribution in NHA. If we add  $\frac{P_{SD,NHA}^{\min}}{P_{\min,NHA}}$  to  $\overline{P_{\min,NHA}}$ , it seems that we move to the direction of the diffuse pixel of HA because  $\overline{P_{\min,NHA}} - \frac{P_{SD,NHA}^{\min}}{P_{\min,NHA}}$  directs to the diffuse pixel of background of the NHA and  $\overline{P_{\min,NHA}} + \frac{P_{SD,NHA}^{\min}}{P_{\min,NHA}}$  directs to the diffuse pixel of object of NHA which is almost same surface of the HA. This is the approximate diffuse intensity of HA and the  $C_{MSF}$  is added to SF image to produce a better MSF image. Another reason for taking 1 SD (standard deviation) of  $P_{\min}(x,y)$  of NHA is that 2 SD may damage the diffuse pixel by directing to HA.



**Figure 3.** (a) Input image; (b) MSF replaced image of HA; (c) non-highlight pixels; and (d) distribution of  $P_{\min}(x,y)$  of NHA.

### 3.2. Removing Brightness Mismatch

The mismatch is noticeable clearly in Figure 3b. From the visual side, we can say that luminance of two portions does not match. To remove the brightness mismatch between HA and NHA, we argue that there is a lack of brightness offset (BO) in HA of MSF replaced image,  $P_{MSF,i,C}(x,y)$ , shown in Equation (4). We also argue that the value of BO will become very small because the information of HA of MSF image is quite visible and well-recovered but not bright enough. As the brightness is not completely dark in HA of MSF image,  $P_{MSF,i,C}(x,y)$ , we do not need to consider the average gray

value of NHA and, instead, we can consider the SD of gray value of NHA because, in most cases for Gaussian-like distribution, SD is much lower than average value. From this analysis, we measure the SD of gray value of HA ( $L_{SD,HA}$ ) and NHA ( $L_{SD,NHA}$ ). We take the small value between  $L_{SD,HA}$  and  $L_{SD,NHA}$  as a BO to remove the brightness mismatch. Experimentally, we decide that small value between  $L_{SD,HA}$  and  $L_{SD,NHA}$  is the appropriate BO to remove the mismatch. In Equation (6), we represent BO and, in Equation (7), we represent  $P_{HF,i,C}(x,y)$  as HF pixels on HA.

$$BO = \begin{cases} L_{SD,NHA}, & \text{if } L_{SD,NHA} < L_{SD,HA} \\ L_{SD,HA}, & \text{Otherwise} \end{cases} \quad (6)$$

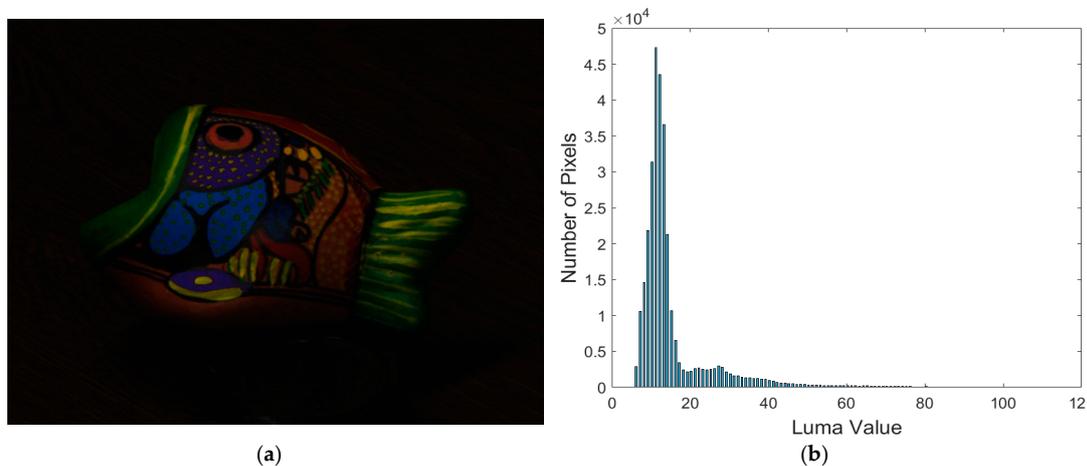
$$P_{HF,i,C}(x,y) = P_{MSF,i,C}(x,y) + BO \quad (7)$$

### 3.3. Low Light Area Enhancement

Most HF images have a histogram distribution similar to in Figure 4b. Thus, according to Hsia et al. [14], we can conclude that these images are low light images. After increasing the brightness of these low light images, we can claim these images as HDR images. Huo et al. [28] followed the same approach to achieve the HDR images, while they used the linear expansion method. Instead of linear expansion, we follow the algorithm of Li et al. [38]. They showed the following equation to brighten the under exposed region. We use Equation (8) for producing our final HDR image,  $P_{HDR,i}(x,y)$ . The specialty of Equation (8) is that it will boost up in the dark region only while keeping the higher luminance value intact.

$$P_{HDR,i}(x,y) = P_{HF,i}(x,y) \left( 1 + \exp \left( -14 \left( \frac{L_{HF}(x,y)}{255} \right)^\gamma \right) \right) V \quad (8)$$

The values of  $\gamma$  and  $V$  in Equation (8) are set experimentally to 2.2 and 1.25, respectively.



**Figure 4.** (a) HF image after removing mismatch; and (b) histogram of luminance of (a).

## 4. Hardware Development

The HLS development part is included in this paper as a part of our next development scenario. Our final goal is the SoC based development. We have chosen the Xilinx device Zynq for our development due to its popularity in the field of SoC. Our primary implementation scenario is described in Figure 5. Zynq architecture has two sides, processor (PS) and PL. The camera will feed the video to the PS. The individual image frame is stored temporally on an off-chip memory such as DDR3. We call it frame buffer. The required frame parameters will be estimated in the PS side. The image frame and parameters will be supplied to the algorithm block in the PL side. This algorithm block will be generated by HLS tool. The display driver will be used finally to see the output.

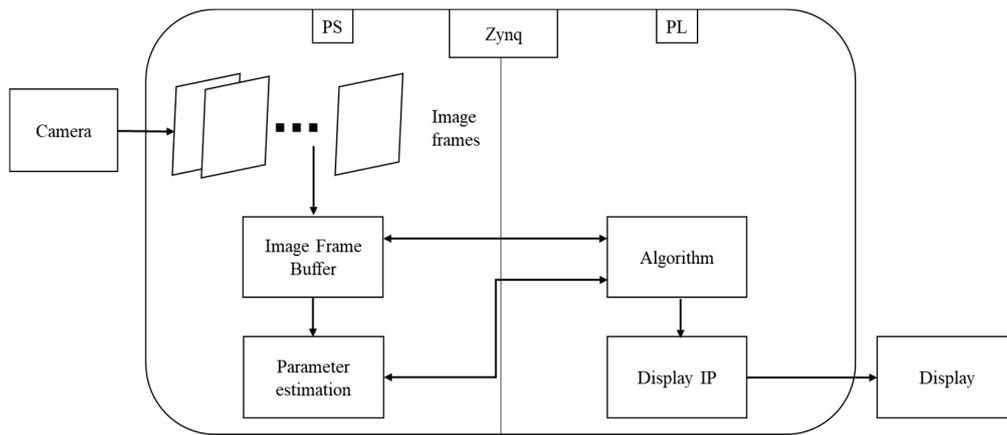


Figure 5. Idea of implementation flow in Zynq board.

4.1. HLS Development

The target of this paper is to represent that our algorithm can be implemented in hardware while most researchers [20,22] only focus on software part development and comparison. For this reason, we used the HLS tool to verify that our idea is implementable in hardware. Our main target is to simplify the PL part development using HLS tool while we also describe our optimized development method. Since Zynq is our final device, the vivado HLS tool is selected for the development. For our final design, we will need AXI bus to communicate from PS to PL and from PL to PS. The other academic HLS tools are not compatible with Xilinx Zynq devices. For example, Leg UP [39] and Intel HLS tool [40] are only compatible with Altera/Intel FPGA.

Each type of HLS tool may have different steps for achieving its goal (i.e., HLL-to-HDL translation). D. Bailey, in his survey of HLS tools [41], identified four steps: dataflow analysis, resource allocation, resource binding, and scheduling. The vivado HLS tool has four basic steps [42]: C-synthesis, C-simulation, RTL verification, and IP packaging. We discuss each step output in our description. Optimization techniques have been generalized by the survey paper [8]. The authors discussed eight types of HLS optimization. Among them, depending on our tool necessity, we use bit-width analysis and optimization, loop optimization, and hardware resource library.

In the beginning of our development, we have separated the part that is designated for the development in the PS while we have also developed a part in the PL. For the PL side development, we use the HLS tool. The operations separated for PS and PL, respectively, are shown in Figure 6.

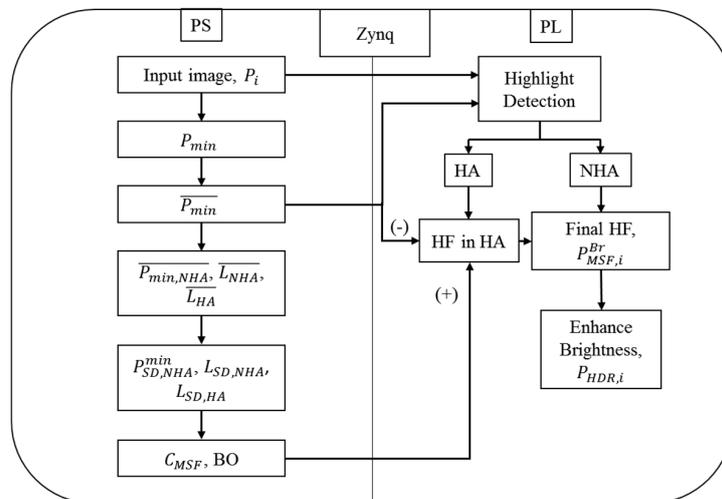


Figure 6. Our implementation steps/operations in PS and PL side.

The separation depends on the convenience of the task. Implementation of the whole algorithm in the PS side will be easier. However, PL side has the advantage in terms of speed due to the capability of parallel processing. Therefore, we want to take this advantage by separating our task between PS and PL. It is generally easy to calculate the average and SD in the PS side, while implementation of an equation in PL will always give the advantage in terms of latency and speed. Obviously, there is trade-off among latency, resources and quality of the output. For calculation of average and SD in PL, Popovic et al. [43] assumed the no variation of light between two frames in a sequence. If we adopt this method in our case, we may need assumption that light is constant among four frames in a sequence because we need three frames in a sequence to calculate the third, fourth, and fifth blocks of PS side in Figure 6. In the fourth frame, finally, we can apply these values. High speed frame capturing can be one solution in this purpose, but this kind of feature will not generalize our algorithm for low speed (<60 fps) commercial camera. To avoid this kind of situation, we consider the PS based implementation in SoC. Hence, we assigned this part ( $C_{MSF}$  and BO) for PS development while other operations are possible conveniently in the PL side.

#### 4.1.1. Coding for IP Optimization

HLS coding starts with the function declaration. It is possible to have multiple functions declaration in an IP. The main function should be selected. During the synthesis, depending on the argument of the main function, the input/output interface is generated. We have one function in the IP with seven arguments: input, output, row size, column size,  $\overline{P_{\min}}$  (shown in Equation (2)), CMSF ( $C_{MSF}$ ), and BO. AXI bus is used to communicate between PS and PL. AXI stream interfaces are used as input/output interfaces for the image frame, named as data bus. The other arguments are received by AXI lite interfaces, named as controlled bus. For pipeline based designing, AXI stream interfaces are used. These interfaces are added by the *pragma* settings suggested by vivado HLS tool user guide [42]. We keep the same clock for both control bus and data bus. The HLS C code is different from general C code, as HLS does not support every feature of the C. For example, it is possible in C to allocate dynamic memory for an array, but the size of the array in HLS must be pre-defined. The array directly consumes space in the fixed block RAM (BRAM) of the FPGA chip and BRAM size is limited in FPGA. Usually, for stream-based input/output, all the operations are done in a single for-loop. This is also one of the main reasons that we have calculated all of the average and SD in the PS side. Therefore, we do not need to use multiple for loops in the PL side.

The optimized version of the code is shown in Tables 1–3. For optimization, first, we use pipeline directive for loop based optimization. We achieved the initiation interval (II) as one that indicates the high throughput [8]. Secondly, we focus on bit-width optimization. During this optimization, we need to select carefully the bit-width of every variable used in the code. Vivado HLS tool has an excellent feature of customized bit-width. Since writing in HLS is general C/C++ code, the important thing is combination of hardware implementation concept during the writing. We describe our code step by step.

In Table 1, Line 8, the main function is LDR2HDR. We consider AXI\_STREAM as a 24-bit stream data type. Hence, our input and output data bit are limited to 24-bits, which is reasonable since we are taking RGB image as an input and output. The rows (row size) and cols (column size) are at most 10-bit because all of our test images are within the limit of  $1023 \times 1023$ . However, our IP can also be applicable to higher resolution by increasing the bit-width of the rows and cols, respectively. The rest of the arguments ( $\overline{P_{\min}}$ , CMSF, BO) are 8-bit. Although they are float in nature, experimentally, we observed that rounding these values does not degrade the image quality. The variables are declared according to necessity of the operation. The constants and variables are declared according to the highest bit-width.

**Table 1.** Data types, main function, and variables.

Partial code for HLS environment
1. typedef ap_axiu <24,1,1,1> stream_24;
2. typedef hls: stream <stream_24> AXI_STREAM;
3. typedef ap_uint <24> uint_24;
4. typedef ap_uint <10> uint_10;
5. typedef ap_uint <8> uint_8;
6. typedef ap_uint <9> uint_9;
7. typedef ap_ufixed <14,9> floatC1;
8. void LDR2HDR (AXI_STREAM & image_in, AXI_STREAM & image_out, uint_10 rows, uint_10 cols, uint_8 minAvg, uint_8 CMSF, uint_8 BO) {
9. ...
10. uint_8 rp1, gp1, bp1, min_pix = 255;
11. uint_9 rp, gp, bp;
12. float phiz, Con4 = 1, Con5 = 14, Con6 = 2.2, Con7 = 255;
13. floatC1 LMSE, delta = 0.0039, Con1 = 0.2989, Con2 = 0.5870, Con3 = 0.1140, V = 1.25;
14. ...
15. }

**Table 2.** Start of for loop, highlight detection and modification.

Partial code for HLS environment
1. ....
2. loop: for (int idxPixel = 0; idxPixel < (rows * cols); idxPixel++)
3. {
4. # pragma HLS PIPELINE
5. # pragma HLS LOOP_TRIPCOUNT min = 100; max = 1,046,529 //(1023 × 1023)
6. stream_24 pixel_in;
7. pixel_in = image_in.read;
8. rp1 = pixel_in.data >> 16;
9. gp1 = pixel_in.data >> 8;
10. bp1 = pixel_in.data;
11. rp = rp1; gp = gp1; bp = bp1;
12. if (rp < min_pix) min_pix = rp;
13. if (bp < min_pix) min_pix = bp;
14. if (gp < min_pix) min_pix = gp;
15.
16. if (min_pix > (2 * minAvg))
17. {
18. rp = rp - min_pix + CMSF + BO;
19. gp = gp - min_pix + CMSF + BO;
20. bp = bp - min_pix + CMSF + BO;
21. }
22. ....
23. }
24. }

Beginning of the loop (Table 2) contains two loop *pragmas*. HLS LOOP\_TRIPCOUNT indicates the maximum and minimum number of pixels. Since our rows and cols are limited to 10-bit, the maximum number will be  $1023 \times 1023 = 1,046,529$ . At the time of this implementation, we have found that, during the use of comparator operator ( $>/<$ ) and arithmetic operator ( $+/-$ ), the data types should be same on both sides i.e., either they will be custom data type (e.g., *ap\_fixed* type) or regular data type (e.g., *float* (32-bit)). We want to keep everything in custom data type with minimum bit required. However, when we take the pixel line (Lines 8–10), the 8-bit data types are needed to take the data input. Eventually, we have cast in next line to keep everything in the 9-bit. Lines 12–14 indicate the minimum channel value selection while Lines 16–21 show HA detection and modification.

**Table 3.** Low light area enhancement and image out.

Partial code for HLS environment
1. ....
2. LMSF = Con1 * rp + Con2 * gp + Con3 * bp;
3. phiz = Con4 + expf(-Con5 * expf(Con6 * logf((float)LMSF/Con7)));
4. rp = rp * V * (floatC1)phiz;
5. gp = gp * V * (floatC1)phiz;
6. bp = bp * V * (floatC1)phiz;
7. min_pix = 255;
8. if (rp > min_pix) rp = min_pix;
9. if (bp > min_pix) bp = min_pix;
10. if (gp > min_pix) gp = min_pix;
11. ap_uint<24> pixelout = (uint_8)rp;
12. pixelout = pixelout << 16;
13. ap_uint <24> pixelout1 = (uint_8)gp;
14. pixelout1 = pixelout1 << 8;
15. ap_uint <24> pixelout2 = (uint_8)bp;
16. pixelout = pixelout   pixelout1   pixelout2;
17. ....
18. }
19. }

In Table 3, luminance is calculated in Line 2. The way of writing of  $X^y$  in vivado HLS is  $\text{expf}(y \times \log f(X))$ . This is the case where power is not an integer number. HLS math library (hls\_math.h) includes math functions that are synthesizable. The math functions are applicable only for single-precision float type or double-precision float type (64-bit) [42]. Therefore, we have written Equation (8) according to the tool’s way in Line 3. Lines 8–10 prevent the floating over flow. During the image out (Lines 11–16), we need to cast again to the 8-bit data type.

4.1.2. Resource and Latency Comparison

Table 4 shows the resource comparison between optimized and unoptimized implementation. In Table 4, we use terms such as latency, iteration latency (IL), initiation interval (II), trip count (TC), dataflow, pipeline, etc. Their definitions are provided in [34,42]. Optimized version is presented in Tables 1–3. The main difference between optimized and unoptimized is that we calculated every equation in float (32-bit) in the case of unoptimized version. We take arguments (minAvg, CMSE, and BO) in float as well. In our design, we do not need any BRAM. The design is optimized for 100 MHz clock. Only 76 clocks will be needed from input to output for one pixel while we achieved the II as one that indicates that, with every clock cycle, we can take a pixel input. In the case of unoptimized version, the IL is 105. Fewer resources are required for the optimized version. The DSP requisition is reduced by almost half in optimized version. Resource to quality comparison has been shown at the end of Section 5.

**Table 4.** Resource and latency optimizations of IP.

Design Considerations		Without Optimizations	With Optimizations
Design clock (MHz)		100 (10 ns)	100 (10 ns)
Total latency	Min	209 (0.00209 ms)	176 (0.00176 ms)
	Max	1,046,638 (~10.4 ms)	1,046,603 (~10.4 ms)
Loop II (No. of cycles)	IL	105 (0.00105 ms)	76 (0.00076 ms)
	II	1	1
	TC	100–1,046,529	100–1,046,529
BRAM		0	0
DSP		79	42
FF		8352	3766
LUT		15,791	6360

During RTL verification, for *Fish* image, it was completely successful. Figure 7 shows the RTL pass report for VHDL.

**Cosimulation Report for 'LDR2HDR'**

Result							
RTL	Status	Latency			Interval		
		min	avg	max	min	avg	max
VHDL	Pass	307280	307280	307280	0	0	0
Verilog	NA	NA	NA	NA	NA	NA	NA

Figure 7. RTL verification report.

Finally, at the IP packaging stage, final resource count for optimized version is shown in Table 5. Three DSP have been reduced in the IP packaging stage while flip-flop (FF) and look up table (LUT) count also reduced in number. At the same time, 193 shift register lookups (SRLs) have been consumed in the IP packaging stage. Besides, our IP achieved the post implementation clock pulse of 9.546 ns, which is less than our desired clock pulse (10 ns). Therefore, timing requirement was met successfully.

Table 5. Resource comparison between C-synthesis and IP packaging stage.

IP Name	Resources	C-Synthesis	IP Package
LDR2HDR	BRAM	0	0
	DSP	42	39
	FF	3766	2705
	LUT	6360	4145
	SRL	NA	193

### 5. Results and Discussion

We used ten test images during our experiment: *Doll, Stone, Hen, Idol, Red Ball, Face, Fish, Bear, Green Pear, and Cups*. For software evaluation, we used MATLAB and, as an HLS tool, Vivado HLS v2016.4 was used. In Figure 8, we show each step output. We also compared our final HDR output with the HDR image generated from Shen’s HF image [22]. Shen’s target was only to generate HF image. We generated Shen’s HF image from the code that is provided in [22]. During generation, the parameter chromaticity threshold was set to 0.05, as in [22]. We applied the same low light area enhancement algorithm (Equation (8)) to Shen’s HF [22] image to compare with our final HDR image. In Figure 8(aiii–ciiii),(aiv–civ), our HDR output is better than the HDR output from Shen’s HF [22] image. In the case of *Doll* image, Figure 8(biii), a color mismatch is noticeable at the area below the guitar. In Figure 8(biv), we cannot see this kind of mismatch. We also compared our MATLAB output with optimized HLS C simulation output, as shown in Figure 8(aiv–civ),(av–cv). Although, in HLS, we reduced the bit-width, we cannot see any visible difference between MATLAB and HLS C simulation output, indicating our IP can generate outputs with software precision level.

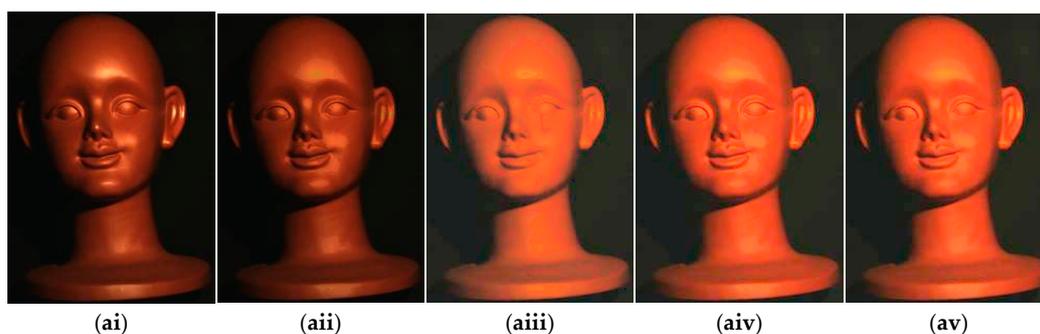
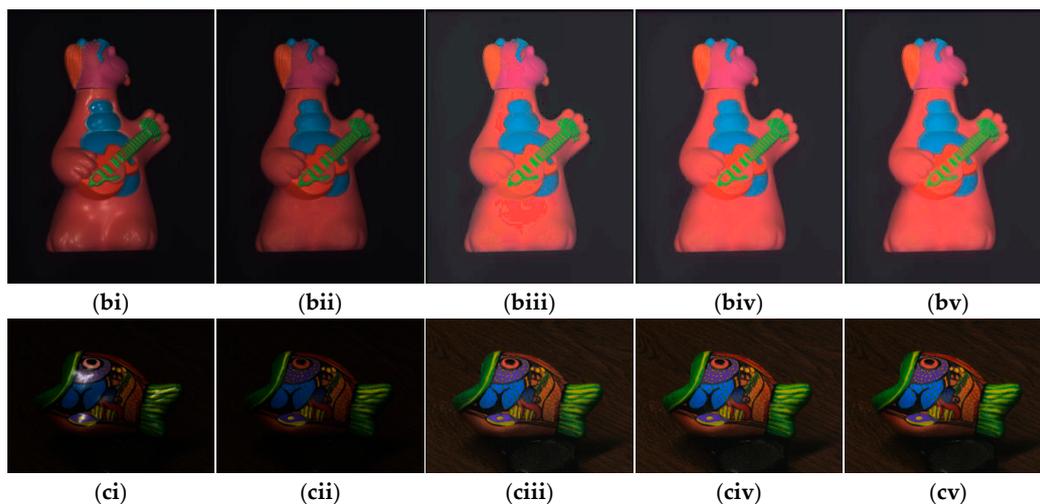


Figure 8. Cont.



**Figure 8.** (ai–ci) Input LDR images (*Idol*, *Doll*, and *Fish*); (aii–cii) our HF images; (aiii–ciiii) HDR images using Shen’s HF [22] images; (aiv–civ) our HDR images; and (av–cv) our HDR images by HLS C Simulation.

In this stage, we compared our output numerically. No-reference metrics are selected on the basis of quality evaluation related to the HDR image. Each metric indicates the quality improvement of an image in a specific area. Uniform distribution of light level, good color, contrast, and overall better visual quality ensure HDR quality image. Lower value of histogram balance (HB) indicates the image is visually better in HA and low light area [14]. Entropy (E) ensures the good contrast of an image, whereas larger value of E represents better contrast of image. Naturalness image quality evaluator (NIQE) and colorfulness-based patch-based contrast quality index (CPCQI) guarantee the overall quality of the image [15–17]. Lower NIQE and larger CPCQI value represent better quality of image. Thus, satisfying the conditions of these metrics ensures the validity of the proposed algorithm. The software output, validated by the no-reference metrics, was used as a reference for checking the accuracy of the hardware stage simulated output using SSIM and PSNR. SSIM indicates the similarity between two images in terms of luminance, contrast and image structure [18]. Paris et al. [19] assumed that PSNR value greater than 40 dB indicates almost invisible difference between two images. Table 6 shows the detailed numerical comparison among input, HDR from Shen’s HF, and our HDR images. The superior values are indicated in bold font. On average, our method performs numerically better than HDR from Shen’s HF.

**Table 6.** Numerical Comparison among Input, HDR using Shen’s method, and Proposed HDR Images.

Image Name	Input Image				Shen’s Method [22]				Our Method			
	HB	E	NIQE	CPCQI	HB	E	NIQE	CPCQI	HB	E	NIQE	CPCQI
<i>Doll</i>	<b>111,110</b>	3.82	12.08	0.3061	114,538	4.167	8.632	0.404	113,648	<b>4.201</b>	<b>7.948</b>	<b>0.414</b>
<i>Stone</i>	55,478	4.39	6.74	0.4845	<b>49,383</b>	4.495	<b>5.552</b>	0.492	49,508	<b>4.511</b>	6.148	<b>0.499</b>
<i>Hen</i>	734,038	4.41	5.87	<b>0.3418</b>	713,070	4.720	4.793	0.316	<b>711,976</b>	<b>4.714</b>	<b>4.575</b>	0.323
<i>Idol</i>	71,551	5.17	20.34	0.4068	66,451	5.551	16.343	0.540	<b>62,474</b>	<b>5.609</b>	<b>16.002</b>	<b>0.557</b>
<i>Red Ball</i>	316,007	5.62	3.33	0.3863	<b>256,030</b>	<b>6.116</b>	<b>3.403</b>	0.487	257,432	6.088	3.471	<b>0.507</b>
<i>Face</i>	48,895	6.04	6.31	0.4819	47,570	6.044	<b>5.566</b>	0.548	<b>46,089</b>	<b>6.110</b>	5.697	<b>0.575</b>
<i>Fish</i>	49,4260	4.08	5.50	0.1565	<b>433,726</b>	5.073	5.659	0.196	434,470	<b>5.074</b>	<b>5.254</b>	<b>0.197</b>
<i>Bear</i>	234,030	5.45	7.16	0.3715	190,986	5.798	5.113	0.486	<b>177,910</b>	<b>5.908</b>	<b>5.060</b>	<b>0.507</b>
<i>Green Pear</i>	<b>88,910</b>	<b>5.23</b>	9.60	<b>0.5500</b>	122,001	4.955	<b>8.077</b>	0.497	120,263	4.992	8.555	0.514
<i>Cups</i>	489,820	4.24	5.19	0.2435	<b>429,220</b>	<b>5.176</b>	5.523	<b>0.326</b>	431,192	5.169	<b>4.892</b>	0.320
Average	26,4409.9	4.845	8.212	0.3728	242,297.5	5.209	6.866	0.429	<b>240,496.2</b>	<b>5.237</b>	<b>6.760</b>	<b>0.441</b>

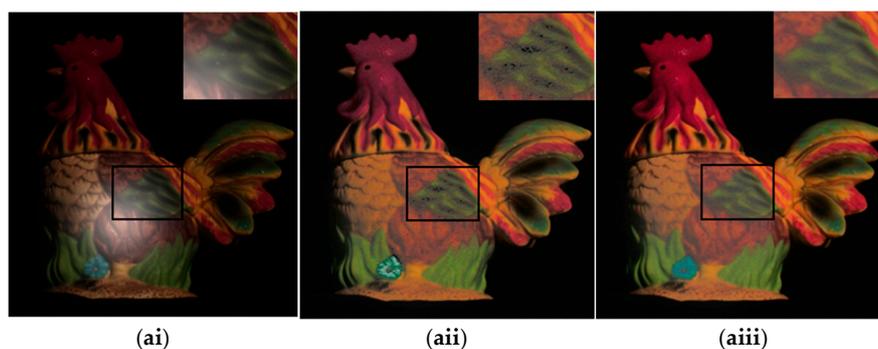
Shen [22] removed highlight by solving the least squares problem of the dichromatic reflection model based on the error analysis of chromaticity and appropriate selection of body color in iterative way. The whole process was done in three steps. In first step, Shen [22] classified diffuse and highlight pixels. In the second and third steps, highlight was removed in an iterative way. On the other hand,

our proposed method uses only two steps. First, we also classify diffuse and highlight pixels as in [22]. However, after that, we only process on highlight pixel using the idea of diffuse and highlight pixel distribution that directs the highlight pixel to diffuse pixel. On the other hand, Shen [22] iterated on whole image in every step to remove highlight, although they classified the diffuse and highlight components in the first step. At this point, our proposed method has achieved advantages in terms of processing speed compared to Shen [22], as shown in Table 7. At the same time, quality of our output images is also better, as proved in Figures 9 and 10. We compared the processing speed between Shen’s method [22] and our method by using the same PC configurations. The configuration of the PC is Windows 7 64-bit operating system, Intel Core i7-3770 K CPU and 12 GB RAM. In Table 7, we can verify that our method is around 76 times faster than Shen’s method [22].

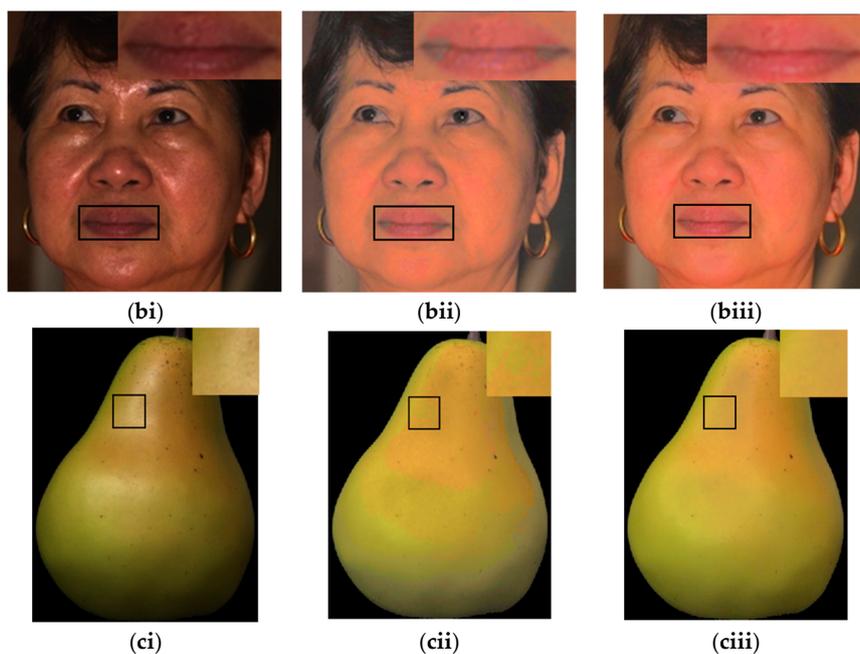
**Table 7.** MATLAB Processing Time (in Second).

Image Name	Shen’s Method [22]	Our Method
	LDR2HDR	LDR2HDR
<i>Doll</i> (240 × 320)	0.573134	0.008595
<i>Stone</i> (202 × 239)	0.060746	0.00531
<i>Hen</i> (735 × 779)	4.357683	0.068634
<i>Idol</i> (199 × 281)	0.235915	0.007351
<i>Red Ball</i> (628 × 355)	1.554282	0.031045
<i>Face</i> (200 × 208)	0.061709	0.005579
<i>Fish</i> (640 × 480)	5.534559	0.033494
<i>Bear</i> (369 × 461)	1.208264	0.022383
<i>Green Pear</i> (276 × 360)	0.148036	0.009028
<i>Cups</i> (640 × 480)	3.806368	0.039217
Average	1.7540696	<b>0.0230636</b>

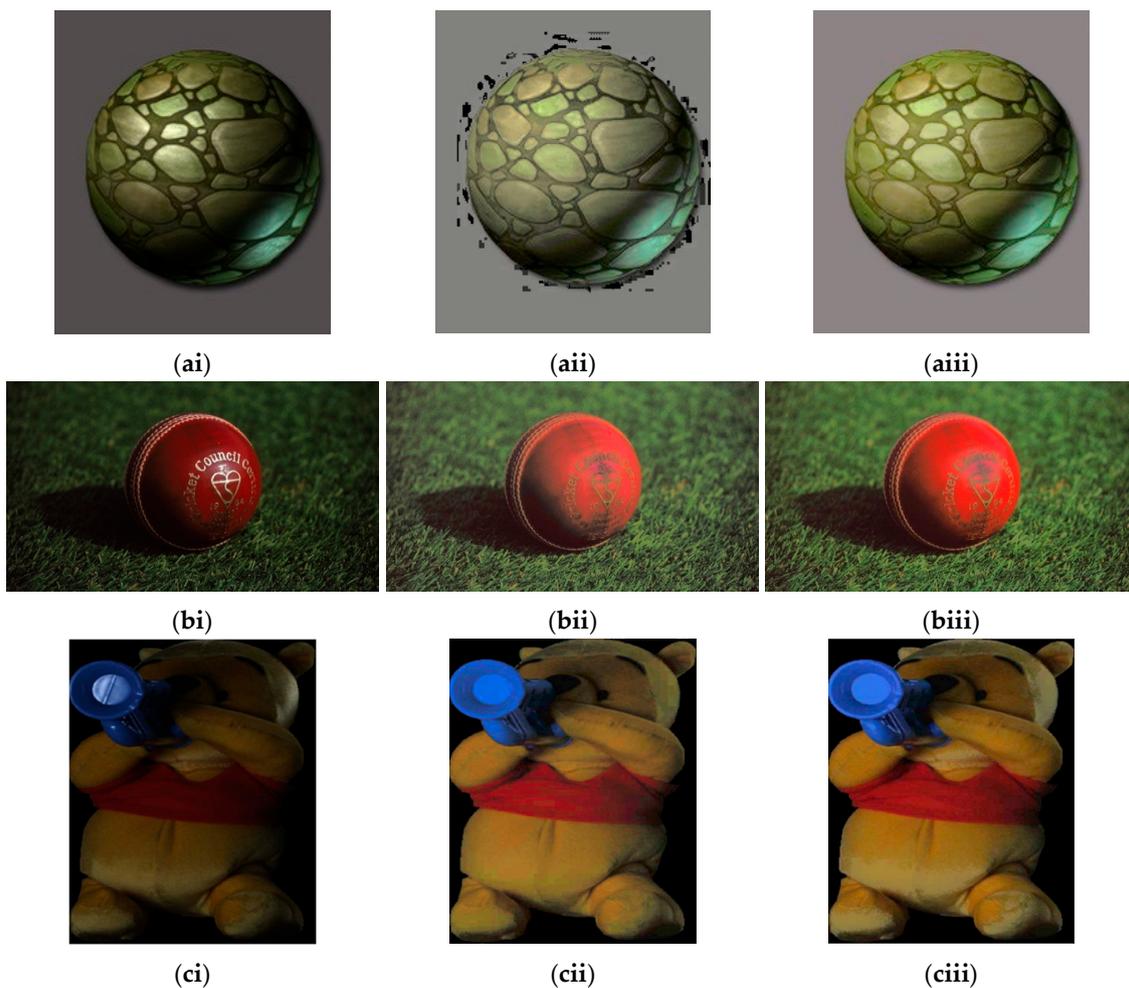
The MATLAB output images are used as a reference to measure the quality of the HLS C-simulation output. For HLS C-simulation, we include the `hls_math.h` library in our testbench as well as the original source file. The HLS math library includes floating point precision factors of synthesizable math functions that are applicable to the hardware [44]. Although the HLS tool calls the GCC compiler (C compiler) for C-simulation, it follows the HLS math library to generate the output of the used math functions instead of standard C output [44]. Therefore, our C-simulation output verifies the hardware stage output. Since the precision level of a math function (e.g., `exp`) for hardware is different from standard C-math libraries [44], we verify our C-simulation output with the software stage output. Table 8 shows the average SSIM and PSNR for both optimized and unoptimized versions. It is obvious that unoptimized version is closer to the MATLAB output, since data (e.g., `minAVG`, `CMSF`, `BO`, etc.) bit-width is close to the MATLAB. In Table 8, average PSNR for without optimization is higher than that of optimized version, which verifies our expectation numerically. However, in both cases, average PSNR is above 40 dB. According to Paris et al. [19], the difference between MATLAB and HLS C simulation will not be visible. Besides, average SSIM is almost same for both cases, which indicates that visual differences are indistinguishable.



**Figure 9.** Cont.



**Figure 9.** Comparison of Shen [22] and our method locally: (ai–ci) input LDR images (*Hen*, *Face*, and *Green Pear*); (aii–cii) HDR images using Shen’s [22] HF images; and (aiii–ciii) HDR images by our proposed method.



**Figure 10.** *Cont.*



**Figure 10.** Visual comparison of Shen [22] and our method globally: (ai–di) input LDR images (*Stone, Ball, Bear, and Cups*); (aii–dii) HDR images using Shen’s [22] HF images; and (aiii–diii) HDR images by our proposed method.

**Table 8.** Evaluation of hardware stage output.

Image Name	Our Method (without Optimization)		Our Method (with Optimization)	
	SSIM (%)	PSNR (dB)	SSIM (%)	PSNR (dB)
<i>Doll</i>	98.898351	45.797372	98.941133	43.509281
<i>Stone</i>	99.291045	44.870771	99.338060	40.386618
<i>Hen</i>	99.228186	45.912994	99.218398	44.955246
<i>Idol</i>	99.364302	46.234078	99.418754	44.070200
<i>Red Ball</i>	98.977379	42.460855	98.968631	42.071957
<i>Face</i>	98.928368	43.099939	98.871433	39.907252
<i>Fish</i>	99.839522	45.480920	99.840969	44.913258
<i>Bear</i>	99.966664	47.223252	99.955782	46.110997
<i>Green Pear</i>	99.991725	47.732003	99.970616	43.052628
<i>Cups</i>	99.828980	45.323002	99.823142	44.667605
Average	99.431452	<b>45.413518</b>	99.434469	43.3645

## 6. Conclusions

The main focus of this paper is the development of parameter free single LDR image to HDR image generation technique using highlight removal algorithm that removes the parameter dependency of our previous paper [12]. We compared our method with the state of the art [22] and showed that our method performs better in terms of quality and processing speed. At the same time, by describing a SoC based implementation scenario, we tried to verify that our method was hardware friendly where PL side development was described depending on the HLS tool. This kind of hardware development approach was completely new for single image based LDR to HDR algorithm. By comparing SSIM and PSNR values, we can claim that, even though we limited the bit width for operation in HLS, the C simulated output was similar to MATLAB output. The main achievement was resolution independency by obtaining throughput one clock cycle and by avoiding the use of BRAM. Finally, although it is true that our algorithm was developed on the assumption that highlight part should not be fully saturated, our method worked very well for all of the test images. In the future, we will develop manual HDL IP with more optimized resources as well as a complete SoC implementation.

**Author Contributions:** The manuscript was written by R.S.; his main contribution is HLS part. P.P.B. contributed the algorithm part and also helped to write the paper. As the corresponding author, K.-D.K. proposed the idea as well as supervised the research.

**Funding:** This research was supported by Human resources Exchange program in Scientific technology through the National Research Foundation of Korea (NRF) funded by the Ministry of Science and ICT (NRF-2018H1D2A2075823) and was also supported by the National Research Foundation of Korea Grant funded by the Ministry of Science, ICT, Future Planning (2015R1A5A7037615).

**Conflicts of Interest:** The authors declare no conflicts of interest.

## Abbreviations

The following abbreviations are used in this manuscript:

ADPCM	Adaptive Differential Pulse Code Modulation
AES	Advanced Encryption Standard
BO	Brightness Offset
BRAM	Block RAM
CPCQI	Colorfulness-based Patch-based Contrast Quality Index
DSP	Digital Signal Processor
E	Entropy
FF	Flip-Flop
FIR	Finite Impulse Response
FPGA	Field-Programmable Gate Array
HA	Highlight Area
HB	Histogram Balance
HDL	Hardware Description Language
HDR	High Dynamic Range
HF	Highlight-Free
HLL	High-Level Languages
HLS	High-Level Synthesis
II	Initiation Interval
IL	Iteration Latency
IP	Intellectual Property
LDR	Low Dynamic Range
LEDs	Light Emitting Diodes
LUT	Look Up Table
MM	Matrix Multiplication
MSF	Modified Specular-Free
NHA	Non-Highlight Area
NIQE	Naturalness Image Quality Evaluator
PCA	Principal Component Analysis
PL	Programmable Logic
PS	Processor
PSNR	Peak Signal-to-Noise Ratio
RTL	Register Transfer Logic
SD	Standard Deviation
SF	Specular-Free
SoC	System on Chip
SRL	Shift Register Lookup
SSIM	Structural Similarity
TC	Trip Count
VHDL	VHSIC Hardware Description Language

## References

1. Qasaimeh, M.; Sagahyroon, A.; Shanableh, T. FPGA-based parallel hardware architecture for real-time image classification. *IEEE Trans. Comput. Imaging* **2015**, *1*, 56–70. [[CrossRef](#)]
2. Wang, P.; McAllister, J. Streaming elements for FPGA signal and image processing accelerators. *IEEE Trans. Very Large Scale Integr. Syst.* **2016**, *24*, 2262–2274. [[CrossRef](#)]
3. Yang, B.; Yang, M.; Plaza, A.; Gao, L.; Zhang, B. Dual-mode FPGA implementation of target and anomaly detection algorithms for real-time hyperspectral imaging. *IEEE J. Sel. Top. Appl. Earth Obs.* **2015**, *8*, 2950–2961. [[CrossRef](#)]
4. Amaro, J.; Yiu, B.Y.S.; Falcao, G.; Gomes, M.A.C.; Yu, A.C.H. Software-based high-level synthesis design of FPGA beamformers for synthetic aperture imaging. *IEEE Trans. Ultrason. Ferroelectr. Freq. Control* **2015**, *62*, 862–870. [[CrossRef](#)] [[PubMed](#)]

5. Chang, H.-Y.; Jiang, I.H.-R.; Hofstee, H.P.; Nam, G.-J. Feature detection for image analytics via FPGA acceleration. *IBM J. Res. Dev.* **2015**, *59*, 8:1–8:10. [[CrossRef](#)]
6. Rose, A.G.; Kube, M.; Weigel, R.; Rose, R. An FPGA-based fully synchronized design of a bilateral filter for real-time image denoising. *IEEE Trans. Ind. Electron.* **2014**, *61*, 4093–4104. [[CrossRef](#)]
7. Xu, Y.; Zhou, Q.; Gong, L.; Zhu, M.; Ding, X.; Teng, R.K.F. High-speed simultaneous image distortion correction transformations for a multicamera cylindrical panorama real-time video system using FPGA. *IEEE Trans. Circuits Syst. Video Technol.* **2014**, *24*, 1061–1069. [[CrossRef](#)]
8. Nane, R.; Sima, V.; Pilato, C.; Choi, J.; Fort, B.; Canis, A.; Chen, Y.; Hsiao, H.; Brown, S.; Ferrandi, F.; et al. A survey and evaluation of FPGA high-level synthesis tools. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.* **2016**, *35*, 1591–1604. [[CrossRef](#)]
9. Lee, H. Method for computing the scene-illuminant chromaticity from specular highlights. *J. Opt. Soc. Am. A* **1986**, *3*, 1694–1699. [[CrossRef](#)] [[PubMed](#)]
10. Shafer, S. Using color to separate reflection components. *Color Res. Appl.* **1985**, *10*, 210–218. [[CrossRef](#)]
11. Ren, W.; Tian, J.; Tang, Y. Specular reflection separation with color-lines constraint. *IEEE Trans. Image Process.* **2017**, *26*, 2327–2337. [[CrossRef](#)] [[PubMed](#)]
12. Banik, P.P.; Saha, R.; Kim, K.-D. HDR Image from Single LDR Image after Removing Highlight. In Proceedings of the IEEE International Conference on Consumer Electronics (ICCE), Las Vegas, NV, USA, 12–14 January 2018; pp. 1–4.
13. Saha, R.; Banik, P.P.; Kim, K.-D. Conversion of LDR Image to HDR-Like Image through High-Level Synthesis Tool for FPGA Implementation. In Proceedings of the IEEE International Conference on Consumer Electronics (ICCE), Las Vegas, NV, USA, 12–14 January 2018; pp. 1–2.
14. Hsia, S.; Kuo, T. High-performance high dynamic range image generation by inverted local patterns. *IET Image Process.* **2015**, *9*, 1083–1091. [[CrossRef](#)]
15. Kamandar, M. Automatic color image contrast enhancement using gaussian mixture modeling, piecewise linear transformation, and monotone piecewise cubic interpolant. *Signal Image Video Process.* **2017**, *12*, 625–632. [[CrossRef](#)]
16. Mittal, A.; Soundararajan, R.; Bovik, A. Making a “Completely Blind” image quality analyzer. *IEEE Signal Process. Lett.* **2013**, *20*, 209–212. [[CrossRef](#)]
17. Gu, K.; Tao, D.; Qiao, J.; Lin, W. Learning a no-reference quality assessment model of enhanced images with big data. *IEEE Trans. Neural Netw. Learn. Syst.* **2018**, *29*, 1301–1313. [[CrossRef](#)] [[PubMed](#)]
18. Wang, Z.; Bovik, A.; Sheikh, H.; Simoncelli, E. Image quality assessment: From error visibility to structural similarity. *IEEE Trans. Image Process.* **2004**, *13*, 600–612. [[CrossRef](#)] [[PubMed](#)]
19. Paris, S.; Durand, F. A fast approximation of the bilateral filter using a signal processing approach. *Int. J. Comput. Vis.* **2009**, *81*, 24–52. [[CrossRef](#)]
20. Tan, R.; Ikeuchi, K. Separating reflection components of textured surfaces using a single image. *IEEE Trans. Pattern Anal. Mach. Intell.* **2005**, *27*, 178–193. [[CrossRef](#)] [[PubMed](#)]
21. Yoon, K.-J.; Choi, Y.; Kweon, I.S. Fast Separation of Reflection Components Using a Specularity-Invariant Image Representation. In Proceedings of the 13th IEEE International Conference on Image Processing, Atlanta, GA, USA, 8–11 October 2006; pp. 973–976.
22. Shen, H.; Zhang, H.; Shao, S.; Xin, J. Chromaticity-based separation of reflection components in a single image. *Pattern Recogn.* **2008**, *41*, 2461–2469. [[CrossRef](#)]
23. Shen, H.; Cai, Q. Simple and efficient method for specular removal in an image. *Appl. Opt.* **2009**, *48*, 2711–2719. [[CrossRef](#)] [[PubMed](#)]
24. Yang, Q.; Wang, S.; Ahuja, N. Real-Time Specular Highlight Removal Using Bilateral Filtering. In Lecture Notes in Computer Science, Proceedings of the in European Conference on Computer Vision (ECCV), Heraklion, Greece, 5–11 September 2010; Part IV. Daniilidis, K., Mragos, P., Praglios, N., Eds.; Springer: Berlin/Heidelberg, Germany, 2010; Volume 6314, pp. 87–100.
25. Reinhard, E.; Stark, M.; Shirley, P.; Ferwerda, J. Photographic tone reproduction for digital images. *ACM Trans. Graph.* **2002**, *21*, 267–276. [[CrossRef](#)]
26. Rempel, A.G.; Trentacoste, M.; Seetzen, H.; Young, H.D. Ldr2Hdr: On-the-fly reverse tone mapping of legacy video and photographs. *ACM Trans. Graph.* **2007**, *26*, 39:1–39:6. [[CrossRef](#)]
27. Banterle, F.; Ledda, P.; Debattista, K.; Chalmers, A.; Bloj, M. A framework for inverse tone mapping. *Visual. Comput.* **2007**, *23*, 467–478. [[CrossRef](#)]

28. Huo, Y.; Yang, F. High-dynamic range image generation from single low-dynamic range image. *IET Image Process.* **2016**, *10*, 198–205. [[CrossRef](#)]
29. Vonikakis, V.; Iakovidou, C.; Andreadis, I. Real-Time Biologically-Inspired Image Exposure Correction. In *IFIPAICT, VLSI-SoC: Design Methodologies for SoC and SiP*; Pigué, C., Reis, R., Soudris, D., Eds.; Springer: Berlin/Heidelberg, Germany, 2010; Volume 313, pp. 133–153.
30. Lapray, P.; Heyrman, B.; Ginhac, D. HDR-ARTiSt: An adaptive real-time smart camera for high dynamic range imaging. *J. Real-Time Image Process.* **2014**, *12*, 747–762. [[CrossRef](#)]
31. Jacquot, B.; Johnson-Williams, N. Real-Time Algorithm Enabling High Dynamic Range Imaging and High Frame Rate Exploitation for Custom CMOS Image Sensor System Implemented by FPGA with Co-Processor. In Proceedings of the SPIE 9400, Real-Time Image and Video Processing, San Francisco, CA, USA, 10 February 2015; pp. 940003-1–940003-13.
32. Popovic, V.; Seyid, K.; Pignat, E.; Çogal, Ö.; Leblebici, Y. Multi-camera platform for panoramic real-time HDR video construction and rendering. *J. Real-Time Image Proc.* **2016**, *12*, 697–708. [[CrossRef](#)]
33. Tambara, L.A.; Tofat, J.; Santos, A.; Kastensmidt, F.L.; Medina, N.H.; Added, N.; Aguiar, V.A.P.; Aguirre, F.; Silveira, M.A.G. Analyzing reliability and performance trade-offs of HLS-based designs in SRAM-based FPGAs under soft errors. *IEEE Trans. Nucl. Sci.* **2017**, *64*, 874–881. [[CrossRef](#)]
34. Choi, Y.-K.; Zhang, P.; Li, P.; Cong, J. HLScope+: Fast and accurate performance estimation for FPGA HLS. In Proceedings of the 2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD), Irvine, CA, USA, 13–16 November 2017; pp. 691–698.
35. Li, P.; Zhang, P.; Pouchet, L.-N.; Cong, J. Resource-Aware throughput Optimization for High-Level Synthesis. In Proceedings of the 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, Monterey, CA, USA, 22–24 February 2015; pp. 200–209.
36. Husejko, M.; Evans, J.; Silva, J.C.R.D. Investigation of high-level synthesis tools' applicability to data acquisition systems design based on the CMS ECAL Data Concentrator Card example. *J. Phys. Conf. Ser.* **2015**, *664*, 082019. [[CrossRef](#)]
37. Daud, N. A Hardware Acceleration based on High-Level Synthesis Approach for Glucose-Insulin Analysis. In Proceedings of the AIP Conference ICESNANO, Solo, Indonesia, 3–5 August 2016; pp. 030087-1–030087-7.
38. Li, Z.; Wei, Z.; Wen, C.; Zheng, J. Detail-enhanced multi-scale exposure fusion. *IEEE Trans. Image Process.* **2017**, *26*, 1243–1252. [[CrossRef](#)] [[PubMed](#)]
39. Canis, A.; Choi, J.; Aldham, M.; Zhang, V.; Kammoona, A.; Anderson, J.H.; Brown, S.; Czajkowski, T. LegUp: High-Level Synthesis for FPGA-Based Processor/Accelerator Systems. In Proceedings of the 19th ACM/SIGDA International Symposium on Field Programmable Gate Arrays, Monterey, CA, USA, 27 February–1 March 2011; pp. 33–36.
40. Intel HLS Tool. Available online: <https://www.intel.com/content/www/us/en/software/programmable/quartus-prime/hls-compiler.html> (accessed on 18 November 2018).
41. Bailey, D.G. The Advantages and Limitations of High Level Synthesis for FPGA Based Image Processing. In Proceedings of the 9th International Conference on Distributed Smart Cameras, Seville, Spain, 8–11 September 2015; pp. 134–139.
42. Vivado HLS Tool User Guide, UG902. 30 November 2016. Available online: [https://www.xilinx.com/support/documentation/sw\\_manuals/xilinx2016\\_4/ug902-vivado-high-level-synthesis.pdf](https://www.xilinx.com/support/documentation/sw_manuals/xilinx2016_4/ug902-vivado-high-level-synthesis.pdf) (accessed on 21 March 2018).
43. Popovic, V.; Pignat, E.; Leblebici, Y. Performance optimization and FPGA implementation of real-time tone mapping. *IEEE Trans. Circuits Syst. II Exp. Briefs* **2014**, *61*, 803–807. [[CrossRef](#)]
44. Hrica, J. Floating-Point Design with Vivado HLS, Application Note: Vivado Design Suite, XAPP599 v1.10. 20 September 2012. Available online: [https://www.xilinx.com/support/documentation/application\\_notes/xapp599-floating-point-vivado-hls.pdf](https://www.xilinx.com/support/documentation/application_notes/xapp599-floating-point-vivado-hls.pdf) (accessed on 21 March 2018).

