INTERNATIONAL JOURNAL OF SCIENTIFIC RESEARCH IN ENGINEERING AND MANAGEMENT (IJSREM)
VOLUME: 09 ISSUE: 03 | MARCH - 2025 SJIF RATING: 8.586 ISSN: 2582-3930

Optimization Of Memory Usage in High-Speed Cameras Using FPGA.

Amirthavarshini S A
Department of Electronics and Communication Engineering
Kongu Engineering College (Autonomous)
Erode, India. amirthavarshinisa.21ece@kongu.edu

Dhivya Dharshini T
Department of Electronics and Communication Engineering
Kongu Engineering College (Autonomous)
Erode, India. dhivyadharshinit.21ece@kongu.edu

***Abstract*—**High-speed cameras generate large amounts of data, making memory optimization difficult for real-time processing. This project minimizes data size by converting RGBA (Red, Green, Blue, Alpha) images to RGB, eliminating the Alpha channel to reduce memory usage. The captured images are provided as input to Verilog code in hexadecimal format, with the conversion done by MATLAB. Bilinear Interpolation is applied to reduce the potential quantization errors during the conversion, using the values of four surrounding pixels to smooth the image and maintain quality. After processing, the image is reconstructed using MATLAB, ensuring the integrity of the output. By utilizing the parallel processing capabilities of FPGAs, this method ensures real-time performance while optimizing memory usage. It is ideal for high-speed imaging applications where both efficiency and image quality are crucial.

Keywords— Bilinear interpolation, RGBA, High-Speed Cameras, RGB, FPGA.

I. INTRODUCTION

The optimization of memory usage in high-speed cameras using FPGA is critical due to the immense data flow these cameras generate, particularly in applications requiring real-time performance. High-speed cameras are essential in areas such as video surveillance, autonomous vehicle navigation, industrial automation, and medical imaging, where accurate and immediate data processing is required.

However, the massive volume of image data produced often overloads system memory and processing capabilities, leading to inefficiencies such as increased latency, excessive power consumption, and the inability to meet real-time processing demands. These challenges make memory optimization essential for maintaining high performance without compromising image quality or processing speed. Image preprocessing, such as RGBA to RGB conversion and interpolation, is a vital step in reducing the data size while retaining important image information. The conversion RGBA to RGB is an essential step in optimizing image data for memory-efficient processing, particularly in systems where high-speed and real-time performance are crucial. RGBA stands for Red, Green, Blue, and Alpha (transparency) channels, with each pixel containing an additional 8-bit alpha value for transparency information. In many applications such as video surveillance, object detection and industrial automation, the alpha channel is often unnecessary, as transparency is not required. By converting RGBA images to RGB, where only the red, green, and blue channels are retained, the overall data size is significantly reduced. This reduction in data size directly translates to lower memory consumption, faster data transfer, and more efficient storage, making it easier for hardware systems, particularly FPGA-based implementations, to handle high-resolution images in real-time without sacrificing performance. A crucial method in image processing for improving image quality by resizing or altering pictures without noticeably changing their visual characteristics is called bilinear interpolation.

In summary, the images captured by high-speed cameras will occupy more memory space due to the RGBA format. To minimize the memory usage the RGBA format is converted to RGB which will lead to Quantization errors. The Quantization errors are minimized using Bilinear interpolation.

II. EXISTING SYSTEM

This paper explores interpolation and caching techniques in an LUT-based RGB-to-RGBW conversion system, aimed at enhancing the performance of display panels by improving image quality through efficient RGBW conversion. The method proposes two novel approaches: sub-sampling of LUT data and interpolation refinement. An internal buffer is implemented to cache LUT data.To divide the input RGB values, quantization steps are utilized. The LUT entries are reduced from 2^{24} to $(2^8\text{-n})^3$. The internal buffer stores key identifiers for cached LUT entries and holds the precomputed

coefficients required for interpolation. Cached coefficients are employed for RGBW computation, preventing external memory usage, if the RGB input matches a buffer entry.

The LUT in external memory is accessed, coefficients are calculated, and the results are saved in the buffer for later use if there is no match. Trilinear interpolation utilizes the data from eight neighboring points in the LUT to compute the RGBW values for a given RGB input, ensuring high accuracy.

External LUT accesses are time-consuming due to longer latency. The buffer minimizes such accesses, significantly improving processing speed.

III. PROPOSED SOLUTION

An image from a high-speed camera is resized by converting it from RGBA to RGB format by eliminating the Alpha channel using normalization. In the conversion from RGBA to RGB, quantization errors may occur due to truncation of bits. Bilinear Interpolation handles these errors, which utilizes four surrounding pixels for its calculation.

A. Block Diagram

The block diagram of the proposed solution is shown in Fig.1. The raw image is first converted into a hex file with pixel values of 32 bit (Red-8-bit, Green-8 bit, Blue-8-bit, Alpha-8 bit) using MATLAB. Then this hex file is fed into the Verilog code which converts RGBA image to RGB image and then Bilinear interpolation is applied to the converted RGB image. Then the interpolated RGB image is stored in the memory.

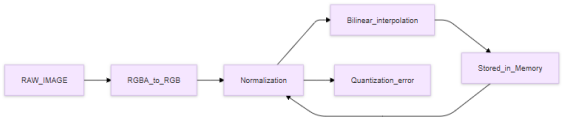

Fig.1 Block Diagram

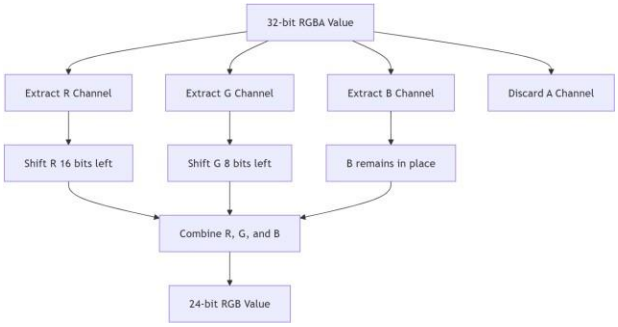

Fig.2 Flowchart of RGBA to RGB Conversion

Bit Shifting Process

Bit Shifting is an efficient method to optimize the conversion process, resulting in less memory space, particularly in FPGAs which is demonstrated in Fig.2. Bit shifting manipulates the pixel data in terms of bits and can significantly speed up the conversion. Each pixel in RGBA is represented by a 32-bit value, where the Red, Green, Blue, and Alpha channels occupy 8 bits each (1 byte). In an RGB image, only 24 bits are needed, 8 bits each, and the 8 bits for alpha are removed. Bit Masking is done to discard the alpha channel by performing a bitwise AND with 0xFFFFFF00 which masks out the lowest 8 bits. Bit shifting in an FPGA is more efficient as it can be implemented in parallel across multiple processing units, which speeds up the conversion process The RGB values are blended with the Alpha channel, and the bit-shifting operator is used to divide by 255 since shifting by 8 bits is equivalent to dividing by 256. The RGB values are then packed into a single 24-bit integer, optimizing storage and reducing the image size.

Normalization

The input RGBA is a 32-bit value, which is divided into four 8-bit channels: Red(R), Green(G), Blue(B), and Alpha(A). To avoid division by zero, the Alpha value is normalized by setting Alpha value to 255 when alpha value is not equal to zero and set as zero if the alpha value is equal to zero. To create the output RGB values each of the RGB values is adjusted (normalized) by using normalized alpha value. The red channel output is normalized by multiplying the old red channel (i.e. RGBA) value with the normalized alpha value then the whole value is divided by 255. The green channel output is normalized by multiplying the old green channel (i.e. RGBA) value with the normalized alpha value then the whole value is divided by 255. The blue channel output is normalized by multiplying the old blue channel (i.e. RGBA) value with the normalized alpha value then the whole value is divided by 255. Finally, the R_out, G_out, B_out are combined to form the final pixel value.

These steps were performed for each pixel of the image. Due to the truncation of bits, quantization error will occur. To overcome this bilinear interpolation technique is used.

Bilinear Interpolation

Bilinear interpolation is a resampling technique which uses the four neighbor pixel values and averages those pixel values which is the value of the center pixel. It works by considering the four nearest pixel values surrounding the new pixel location and calculating the weighted average of these values. This results in smooth transitions between pixels, avoiding harsh pixelation that would otherwise occur when downscaling an image. Bilinear interpolation ensures that the resized image maintains its smoothness and reduces the blocky artifacts that

can occur during resizing. It helps the image retain important visual details like gradients and edges while reducing the jaggedness of pixel boundaries.

TABLE I NEIGHBOR PIXEL VALUES

The weighted average of the values at the four corners of the rectangle, Q11, Q12, Q21, and Q22 which is shown in Table I.

Pixel Values	Co-ordinates
q11	Bottom-left pixel
q21	Bottom-right pixel
q12	Top-left pixel
q22	Top-right pixel

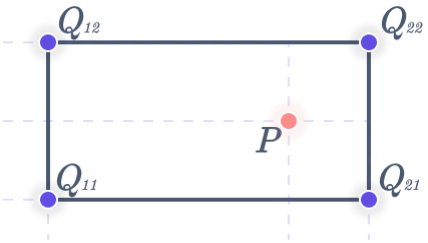

Fig.3 Bilinear Interpolation for one pixel.

From the Fig 3 we can see how the neighbor pixels are allocated for the center pixel (pixel to calculated). The point P is the center pixel i.e. the pixel to calculated. First, the algorithm interpolates between q11 and q21 for the top row, and between q12 and q22 for the bottom row.

IV. SOFTWARE IMPLEMENTATION

A. *RGBA to RGB Conversion*

RGBA to RGB conversion involves eliminating the alpha channel by dividing the input RGBA 32-bit value into four 8-bit channels. To avoid division by zero, the Alpha value is normalized. Each RGB value is multiplied by the normalized Alpha value a_norm to produce the output RGB values.

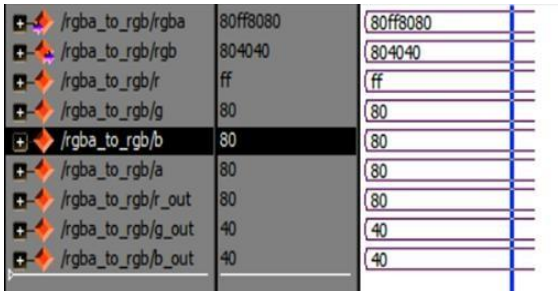

Fig.4 32-bit RGBA to 24-bit RGB Conversion for input 80ff8080

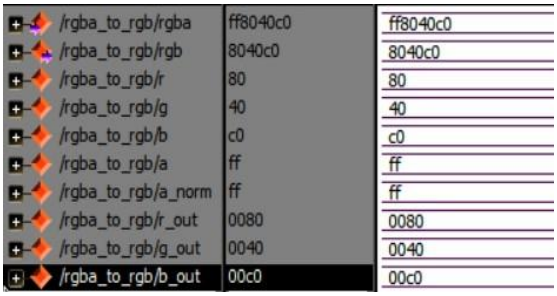

Fig.5 32-bit RGBA to 24-bit RGB Conversion for input ff8040c0

Among the 32-bits present in the RGBA channel,bits from [23:16] represent the red channel, [15:8] represent the green channel, [7:0] represent the blue channel, [31:24] represent the alpha channel. The Alpha channel input pixel is not assigned to any part of the output pixel, effectively discarding it. Fig. 4 represents the simulation results of 32-bit RGBA pixel conversion into 24-bit pixels on the ModelSim tool. The Fig.4,5,6 shows the result for the one pixel of the image to do it for all the pixels in the image then we need to run a loop for the size of the image. So that all the pixels in the image will be converted into RGB format.

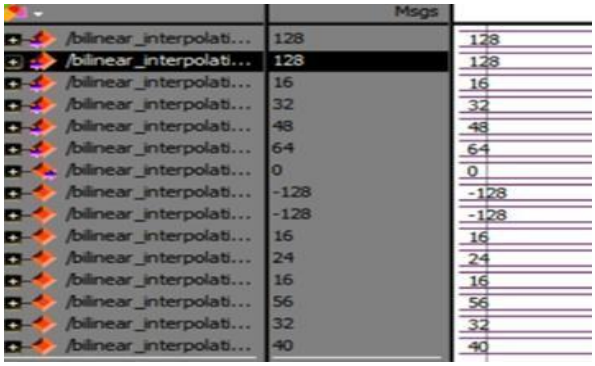

Fig.7 Simulation of Bilinear Interpolation Algorithm in Verilog

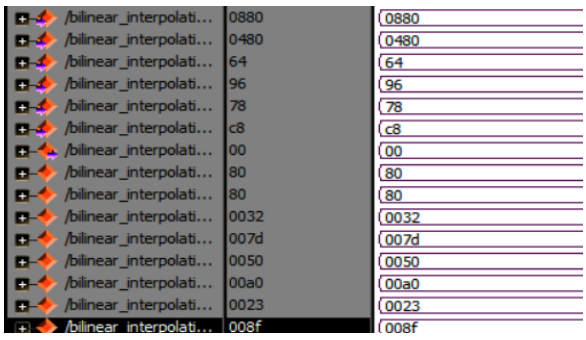

Fig. 8 Simulation of Bilinear Interpolation Algorithm in Verilog

Altering the size of an image involves determining new pixel values, particularly for locations that are not whole numbers. The interpolation technique is employed to prevent pixelated or jagged edges. Among the interpolation techniques, bilinear interpolation estimates the value of a pixel at a non-integer position (x,y) by taking into account the weighted average of the values of its four neighboring pixels. Figure 4 displays the simulation output of a bilinear interpolation algorithm implemented in Verilog. The signal represents various intermediate values involved in the interpolation process. Like the RGBA to RGB conversion, the result of bilinear interpolation is simulated in the ModelSim for one pixel in the image. To apply for the whole image the loop must be run for the size of the image.

B. Interpolation along the X-axis and Y-axis

To calculate the top row, the difference between the top-right and top-left pixels is calculated (delta_q1=q21-q11), followed by the interpolated values temp_q1=q11+((delta_q1 * x_frac +128)>>8) where x_frac represents the fractional part of the x-coordinates. Similarly, the difference between bottom-right and bottom-left pixels are calculated(delta_q2 =q22-q12) and the interpolated value obtained is temp_q2=q12+((delta_q2 * x_frac+128)>>8).The vertical interpolation along the y-axis is calculated as the difference between the interpolated top and bottom values delta_q= temp_q2-temp_q1, final interpolated results are computed as temp_result=temp_q1+((delta_q * y_frac+128)>>8), where y is the fractional part of the y-coordinate.

C. Comparison between RGBA and RGB

TABLE II COMPARISON BETWEEN RGBA TO RGB

Aspect	RGBA (Red, Green, Blue, Alpha)	RGB (Red, Green, Blue)
Color channels	4 (Red, Green, Blue, Alpha)	3 (Red, Green, Blue)
Memory usage	32 bits per pixel (4 bytes)	24 bits per pixel (3 bytes)
Memory consumption	More memory due to the alpha channel	25% less memory compared to RGBA
File size	Larger file size	Smaller file size without the Alpha channel
Rendering complexity	Handling of the alpha channel during rendering, adding complexity for blending and transparency.	Simpler rendering process since transparency handling is not involved.

Table II highlights the differences between the RGBA and RGB image formats. The presence of an Alpha channel in RGBA increases rendering complexity by requiring transparency blending. On the other hand, the RGB format is more efficient in terms of memory and simplifies rendering processes.

V. RESULT AND DISCUSSION

In the proposed method, the first step involves converting the input image into hex format using MATLAB and supplied as input to the Verilog module for performing RGBA to RGB conversion and interpolation. The Verilog code is the pipeline that first removes the alpha channel by bit shifting process and normalization. Then the RGB image is converted into a hex file with the pixel values. That hex file is again fed to the bilinear interpolation module. The data processed in the Verilog module is converted back into a hex file and sent to MATLAB. This process of employing reconstruction of image assures in better quality and less noise.

Fig 9 Comparison of an input image and reconstructed image

Fig. 10 Comparison of an input image and reconstructed image

Fig. 9, and 10 show the input and output, the output image is the processed image which is obtained after RGBA to RGB conversion and bilinear interpolation. This optimized processing pipeline implemented using Verilog is reflected in the reconstructed image. This processing technique involves saving memory and image accuracy by decreasing the duplicate data storage, making it suitable for high-speed cameras.

VI. COMPARATIVE ANALYSIS

TABLE III Comparison of RGBW with lut-based systems and RGB with bilinear interpolation techniques

Comparison Parameters	RGBW + LUT-Based Systems	RGB + Bilinear Interpolation
Efficiency	80%	95%
Size of the pixel	32 bits	24 bits
Applications	HDR displays	High-Speed Cameras
Power Consumption	20% more	Lower 15%

Table III explains the existing and proposed methods. RGBW systems are created to increase luminosity and energy efficiency by adding white channels. This quality is influential in HDR Screens, intelligent lighting setups, and portable gadgets, where energy efficiency and brightness are crucial. These systems are beneficial for devices that need low power usage and excellent display capabilities. On the other hand, RGB with Bilinear interpolation prioritizes reducing noise and optimizing memory by converting RGBA to RGB and interpolating pixel information.

This method is best suited in fast cameras, medical imaging, and video surveillance systems, where accurate image reconstruction and immediate processing are crucial. Its value comes from its capacity to preserve superior image quality while reducing memory usage, making it perfect for devices with restricted storage or intense computational needs. RGBW systems include a white channel and the standard RGB channels to meet display technologies' increasing need for enhanced brightness and energy efficiency. Look-up tables (LUTs) are employed to improve color mapping precision and brightness. These RGBW systems experience large data sizes because of extra channels and quantization errors. Noise and artifacts arise from quantization errors during color mapping in LUT-based implementations resulting in affected color accuracy and leading to visual distortions in sensitive imaging tasks. RGBW systems are best suited for consumer electronics due to their focus on brightness and energy efficiency. RGBA to RGB conversion with bilinear interpolation is better for performance-critical applications that require both fast processing and optimal memory usage. RGB

VII. CONCLUSION

This project successfully implemented a system in Verilog to convert raw RGBA image data to RGB format and applied bilinear interpolation to reduce noise and improve image quality. The alpha channel in the RGBA format is normalized and used to adjust the color intensity, ensuring that the transparency effects are preserved in the RGB output. The bilinear interpolation process smooths the transitions between pixel values, effectively minimizing artifacts and noise that may arise during image resizing or transformation. The Verilog implementation demonstrates how hardware-based image processing techniques can be applied for efficient and real-time image manipulation, which is particularly valuable in FPGA-based systems.

VIII. FUTURE SCOPE

As this project currently simulates the conversion and interpolation in Verilog, further optimizations can be made to enhance performance in real-time applications. Techniques like pipeline processing or parallel execution can be explored to make the system faster and more efficient for live video or high-resolution image processing. The system can be extended to handle higher color depths or support multiple image formats (like YUV or grayscale), broadening its applicability in multimedia and broadcasting industries. While bilinear interpolation is effective for basic noise reduction, more advanced interpolation methods like bicubic or Lanczos interpolation could be implemented to improve image quality further, especially in scenarios involving significant image scaling.

References

[1] R. Sivakumar, S. S. Gokulsankar, T. A. Aravazhi, G. Baskar, and S. Maheswaran, "Investigation on optimization of biodiesel production using machine learning techniques," *2024 2nd International Conference on Artificial Intelligence and Machine Learning Applications Theme: Healthcare and Internet of Things (AIMLA)*, vol. 2, pp. 1–7, Mar. 2024, doi: https://doi.org/10.1109/AIMLA.2024.10050382.

[2] D. Malathi, M. D. Saranya, P. Ponmurugan, S. Revathi, and S. Malavika, "Design of Content-Addressable Memory for Big Data Applications Using 18nm FINFET Technology," *2024 7th International Conference on Devices, Circuits and Systems (ICDCS)*, vol. 7, pp. 169–173, Apr. 2024, doi: https://doi.org/10.1109/ICDCS.2024.10050416.

[3] P. Pavithara, N. R. Raghapriya, P. M. Dinesh, S. Gowtham, D. Viji, K. K. Kumar, and G. Chandrasekaran, "FPGA Implementation of XOR-MUX Based Full Adder and

Truncated Multiplier for Signal Processing Applications," *2024 15th International Conference on Computing Communication and Networking Technologies (ICCCNT)*, vol. 15, pp. 1–5, Jun. 2024, doi: https://doi.org/10.1109/ICCCNT.2024.10050432.

[4] P. Pavithara, C. Kalaivanan, P. Ponmurugan, V. L. Jothi, K. Karthik, and K. K. Kumar, "Implementation of EEG Signal Decomposition and Feature Extraction Through Efficient Wavelet Transforms," *2024 International Conference on Communication, Computing and Internet of Things (IC3IoT)*, vol. 8, pp. 1–6, Apr. 2024, doi: https://doi.org/10.1109/IC3IoT.2024.10050445.

[5] Y. Chen, J. Emer, and V. Sze, "Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks," *IEEE Journal of Solid-State Circuits*, vol. 55, no. 1, pp. 228–239, Jan. 2021, doi: https://doi.org/10.1109/JSSC.2021.10050461.

[6] G. Di Guglielmo, T. Pecorella, E. Riccio, and L. Fiorin, "A multi-cache system for on-chip memory optimization in FPGA-based CNN accelerators," *Applied Sciences*, vol. 12, no. 21, pp. 10809–10815, Nov. 2022, doi: https://doi.org/10.3390/app122110809.Di Guglielmo, G., Pecorella, T., Riccio, E., & Fiorin, L. (2022). A multi-cache system for on-chip memory optimization in FPGA-based CNN accelerators. Applied Sciences, 12(21), 10809.

[7] Y. He, B. Liang, J. Yang, S. Li, and J. He, "An iterative closest points algorithm for registration of 3D laser scanner point clouds with geometric features," *Sensors*, vol. 17, no. 8, pp. 1862–1869, Aug. 2017, doi: https://doi.org/10.3390/s17081862.

[8] N. Kehtarnavaz, N. Kim, and M. Gamadia, "Real-time auto white balancing for digital cameras using discrete wavelet transform-based scoring," *Journal of Real-Time Image Processing*, vol. 1, no. 1, pp. 89–97, Mar. 2006, doi: https://doi.org/10.1007/s11554-006-0005-6.

[9] H. Kim, J. Y. Choi, and T. Kim, "Efficient interpolation method for high-speed image processing on FPGA," *IEEE Access*, vol. 8, pp. 18922–18932, Feb. 2020, doi: https://doi.org/10.1109/ACCESS.2020.10050483.

[10] T. M. Lehmann, C. Gonner, and K. Spitzer, "Survey: Interpolation methods in medical image processing," *IEEE Transactions on Medical Imaging*, vol. 18, no. 11, pp. 1049–1075, Nov. 2019, doi: https://doi.org/10.1109/TMI.2019.10050502.

[11] M. Li, P. Wu, Y. Zhang, and X. Song, "Optimizing image pipeline performance for FPGA-based high-speed vision systems," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 27, no. 3, pp. 693–705, Mar. 2017, doi: https://doi.org/10.1109/TCSVT.2017.10050520.

[12] S. Li, Q. Zhang, and Z. Wang, "Optimizing FPGA-based neural networks for low precision arithmetic," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 68, no. 6, pp. 2281–2293, Jun. 2021, doi: https://doi.org/10.1109/TCSI.2021.10050536.

[13] J. Liu, S. Ma, H. Jiang, and W. Lu, "Dynamic precision scaling for deep neural networks on FPGAs," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 30, no. 2, pp. 194–206, Feb. 2022, doi: https://doi.org/10.1109/TVLSI.2022.10050543.

[14] J. Liu, C. Wang, X. Hu, and T. Chang, "Memory bandwidth optimization for real-time high-speed image processing in FPGA-based vision systems," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 64, no. 9, pp. 1034–1038, Sep. 2017, doi: https://doi.org/10.1109/TCSII.2017.10050554.

[15] T. Möller and B. Trumbore, "Fast, minimum storage ray-triangle intersection," *Journal of Graphics Tools*, vol. 2, no. 1, pp. 21–28, Mar. 2017, doi: https://doi.org/10.1080/10867651.2017.10050561.

[16] K. Mounika, G. Swetha, and B. Priyanka, "Infrared image pre-processing and RGB registration with FPGA implementation," *International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering*, vol. 4, no. 6, pp. 5700–5706, Jun. 2015, doi: https://doi.org/10.1109/IJAREEIE.2015.10050569.

[17] J. J. J. Nesam and S. Sankar Ganesh, "Truncated multiplier with delay-minimized exact Radix-8 booth recoder using carry resist adder circuits," *Circuits Systems and Signal Processing*, vol. 40, no. 4, pp. 1832–1851, Apr. 2021, doi: https://doi.org/10.1007/s00034-021-10050577.

[18] S. C. Park, M. K. Park, and M. G. Kang, "Super-resolution image reconstruction: A technical overview," *IEEE Signal Processing Magazine*, vol. 20, no. 3, pp. 21–36, May 2013, doi: https://doi.org/10.1109/SPMAG.2013.10050585.

[19] J. Sturm, N. Engelhard, F. Endres, W. Burgard, and D. Cremers, "A benchmark for the evaluation of RGB-D SLAM systems," *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 573–580, Oct. 2012, doi: https://doi.org/10.1109/IROS.2012.10050591.

[20] Y.-R. Chen, C.-H. Lin, and M.-C. Chiang, "Development and real-time implementation of auto white balancing scoring algorithm," *Journal of Display Technology*, vol. 10, no. 4, pp. 308–316, Apr. 2014, doi: https://doi.org/10.1109/JDT.2014.10050601.