Design and Implementation of FPGA-Based Real-Time Image Denoising Accelerator

Yunfei Wang

***Abstract*— In modern surveillance systems, the quality of real-time images is often affected by noise, especially under low-light conditions, which can significantly impact the system's accuracy and reliability. To overcome this challenge, infrared images have been widely regarded as an effective alternative. Infrared images are not affected by visible light interference and can provide clearer and more stable surveillance data in various complex environments.**

However, infrared images also have noise problems, particularly during long-term monitoring, where image quality may gradually deteriorate. To effectively reduce noise in infrared images, this paper proposes a noise reduction algorithm based on non-uniformity correction with parameter updates. Firstly, the algorithm dynamically adjusts correction parameters to achieve efficient real-time non-uniformity correction. Compared to traditional methods, the algorithm not only improves correction accuracy and image stability but also significantly reduces system response time, meeting the requirements of complex surveillance environments. Finally, a hardware design based on Field-Programmable Gate Array (FPGA) is used to implement this algorithm, which significantly enhances its performance.

Experimental results show that for a 640×512 resolution image, the frame rate can reach 66 frames per second, for a 1280×1024 resolution image, it can achieve 30 frames per second, and for a 640×640 resolution image, the frame rate can reach 50 frames per second, with a latency time of under 40ms.

***Index Terms*—Parameter update, Non-uniformity correction, FPGA,Denoising.**

I. Introduction

Image denoising plays a crucial role in computer vision and image processing, enhancing image quality and improving the performance of subsequent tasks such as object detection and recognition. Real-time image processing is particularly important in fields like video surveillance, autonomous driving, and medical imaging. Therefore, developing efficient real-time denoising algorithms is of practical significance. Infrared images are widely used in military, security, and night vision applications because they provide reliable imaging in low light and complex environments. Unlike visible light images, infrared images capture thermal radiation information.

The infrared focal plane array (IRFPA) [1] is the core component of infrared imaging systems but is often affected by fixed-pattern noise (FPN) [2]and pixel response nonuniformity, which degrade image quality. To improve image quality, an effective non-uniformity correction (NUC) [3] algorithm is needed and should be implemented on embedded hardware platforms. The FPGA[4]platform offers parallel computing and optimization, reducing memory latency and enhancing processing speed and real-time performance.

Manuscript received March 26, 2025

YunFei Wang, School of computer science and technology, Tiangong University, Tianjin, China.

By deploying the parameter-update-based NUC algorithm onto an FPGA platform, the goal is to achieve real-time parameter updates, ensuring the timeliness and effectiveness of the correction to meet real-time requirements, improve processing performance, reduce system costs and power consumption, and adapt to the needs of different sensors and application scenarios.

The main contributions of this paper are as follows:

(1) We propose a parameter-update-based NUC algorithm that fully utilizes fixed-point arithmetic during nonuniformity correction. The algorithm consists of video stream processing and adaptive parameter updates.

(2) The adaptive parameter update allows the algorithm to dynamically adjust parameters based on the number of image frames currently being processed. This real-time responsiveness enables the algorithm to quickly adapt to varying input conditions and environmental changes without requiring pre-fixed parameters.

(3) The DeNoise module is accelerated based on the runtime of each module, and Gaussian blur within the module is also optimized. Finally, other parameter update modules are accelerated using multi-core processing.

(4) The proposed parameter-update-based NUC algorithm is implemented on FPGA. With a 200 MHz clock, it achieves 66 FPS[5] for 640×512 resolution images, 30 FPS for 1280×1024 resolution images, and 50 FPS for 640×640 resolution images, with denoising performance equivalent to the original NUC algorithm.

II. Related Work

With the continuous development of hardware acceleration technology, FPGA has been widely used in image processing tasks, demonstrating its unique advantages in image denoising. In order to improve the effectiveness of image denoising, researchers have begun to explore the application of FPGA technology in image denoising tasks, using hardware acceleration to enhance processing speed and efficiency.

Rong Shenghui et al. [6] proposed an improved version of the neural network-based NUC algorithm and implemented it on FPGA hardware. This method combines guided image filtering and projection-based motion detection technology. The memory consumption is less than 45% of the chip’s capacity, and the usable block memory utilization rate is 72%. Additionally, by increasing the frame rate of the input infrared video, the processing speed can be boosted to 180 FPS (256×256 pixels). Rodolfo Redlich et al. [7] proposed an embedded hardware implementation of the Scribner algorithm for non-uniformity correction. On infrared video with a resolution of 320×240 pixels, the system achieved a processing speed of over 130 frames per second using the XC3S1200E, with a power consumption of 329mW.

Javier Contreras et al. [8] designed a digital hardware filter for non-uniformity noise estimation and real-time correction of infrared focal plane arrays. The NUC board, implemented on the XC3S500E, operates at a frequency of 75 MHz with a dynamic power consumption of 17.3mW, utilizing only 10% of the FPGA's logic resources. After calibration with a reference black body at two points, the system achieves a peak signal-to-noise ratio (PSNR) [13] of 35dB within 50 frames, and the error in its double-precision Matlab implementation is less than 0.35dB, allowing the system to process 1,254 frames per second for 320×240 pixel video.

Liang Zou et al. [9] designed a pipelined circuit structure based on Very Large Scale Integration (VLSI) [14]FPGA that efficiently executes real-time non-uniformity correction algorithms for infrared focal plane arrays. This architecture, implemented on the EP2C35F672C8, uses 12,548 logic units, 11,320 combinational logic units, and 6,207 dedicated logic registers, supporting a global clock frequency of 96 MHz. For 320×256-sized corrected images, data acquisition uses a 16 MHz local clock, and the time for correcting and transmitting each image is approximately 5,122 microseconds, with a system throughput of 100 frames per second, i.e., a processing time of about 10 milliseconds per frame.

In 2023, Andrejs Cvetkovs et al. [10] designed an FPGA-based digital circuit for infrared image acquisition and preprocessing, which includes non-uniformity correction functionality, implemented on the XCZU9EG SoC platform. The system can process 320×240 infrared images and 1024×720 RGB images, with a maximum throughput of 30 frames per second, using 445 hardware multipliers, 57,474 logic units, 79,894 registers, and 171 BRAM blocks. In 2021, Huawei Wang et al. [11] designed an FPGA-based short-wave infrared camera logic architecture, implemented on the XC4VS55. This design supports two operating modes and can achieve 25 FPS at a resolution of 640×512 and 4000 FPS at a resolution of 64×48. Sheng Yicheng et al. [12] proposed a hardware system for IRFPA real-time two-point calibration (TPC) [15]algorithm, using the EP3C120F780 as the controller. The system completes operations such as module design, ping-pong memory, high and low-temperature image processing, correction coefficient calculation, and non-uniformity correction within the FPGA.

III. NUC ALGORITHM BASED ON PARAMETER UPDATES

A. The design idea of the algorithm

Non-uniformity noise (NU) in infrared images is caused by sensor non-uniformity, environmental changes, and other factors, affecting image quality and accuracy. NUC algorithms are typically divided into two types: reference-based and scene-based. Scene-based algorithms are flexible but complex, making them unsuitable for hardware acceleration platforms. In contrast, the Two-Point Correction method is simple and computationally light, making it suitable for embedded hardware, but it has limitations such as reliance on reference sources and poor adaptability.

To address these issues, this paper proposes a hardware-friendly parameter update-based NUC algorithm. This algorithm adjusts calibration parameters in real time, avoiding the dependency on fixed reference sources and improving calibration performance in dynamic environments. It can automatically learn and adjust parameters, reducing manual calibration and enhancing system intelligence. The adaptive update strategy based on image frame counting, combined with parallel processing techniques, optimizes denoising performance and algorithm efficiency. The Figure 1 illustrates the architecture of the NUC algorithm based on parameter updates.

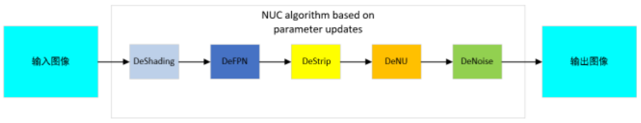

Fig.1 The architecture of the parameter update-based NUC algorithm

B. The overall process of the algorithm.

When the algorithm performs non-uniformity noise removal, it uses fixed-point arithmetic exclusively. The basic rules for fixed-point implementation of the algorithm are as follows:

(1) The data is first scaled up and then scaled down to improve the precision of intermediate calculations.

(2) 32-bit variables are preferred, and 64-bit variables are used sparingly.

(3) Data scaling is achieved using shift operations to enhance efficiency.

The parameter update-based NUC algorithm consists of two parts: video stream processing and adaptive parameter updates. The algorithm relies on adaptive parameters to perform non-uniformity correction tasks, while the adaptive parameter update and correction processes are executed in parallel. The flow of the algorithm is shown in Figure 2.

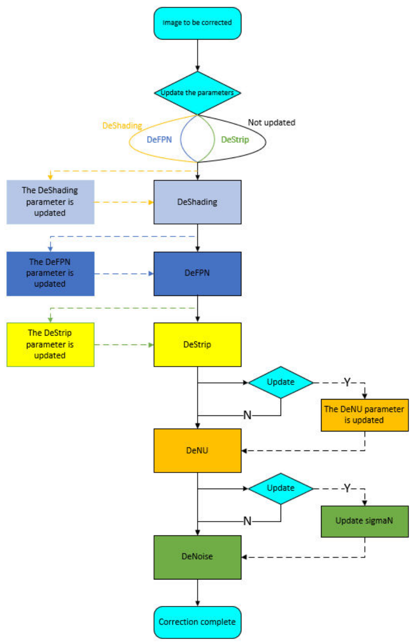

Fig.2 The flow of the parameter update-based NUC algorithm

First, based on the input image frame count, the algorithm makes a judgment to determine one of the four choices for adaptive parameter updates: updating the DeShading parameter, updating the DeFPN parameter, updating the DeStrip parameter, or not updating any parameters. Then, the Shading, FPN, and Strip noise removal processes are completed. If the previous judgment indicates that an adaptive parameter needs to be updated, a new sub-thread is created to update the adaptive parameters in parallel during the noise removal process.

The updates for DeShading, DeFPN, and DeStrip parameters correspond to the orange, blue, and green processes in the diagram, with dashed lines indicating parallel processing. Next, the NU noise is removed, which also depends on the image frame count to determine whether the corresponding adaptive parameters need to be updated. If needed, a sub-thread concurrently updates them without affecting the main correction process. Finally, the DeNoise module, which involves updating the parameter sigmaN, also updates in a sub-thread concurrently. After all the modules have processed the image, the non-uniformity noise correction of the infrared image is complete.

C. *Overall performance and analysis of the algorithm*

All modules are activated to process the infrared image. The results of the algorithm after fixed-point processing are compared with those of double precision processing, and the maximum error for each frame of the image is recorded. As shown in Figure 3, tests were conducted in an outdoor scene. From the figure, it can be seen that the maximum error is around 16, and it is only brief. For most of the time, the error stays within 10. Since this is a test in an outdoor scene, and considering the high dynamic range of outdoor environments, this error is within an acceptable range. The test results for the indoor scene are shown in Figure 4. It can be seen that the error is smaller indoors, with the maximum error value not exceeding 4, so the impact on the results is minimal. The reason why the error is much smaller than in the outdoor scene is that the dynamic range of the indoor scene is smaller than that of the outdoor scene.

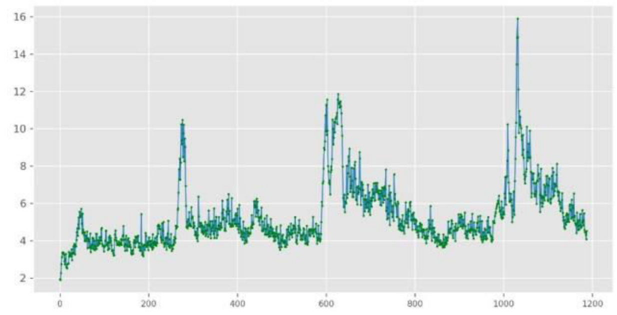

Fig.3 Maximum error of the algorithm (outdoor scene)

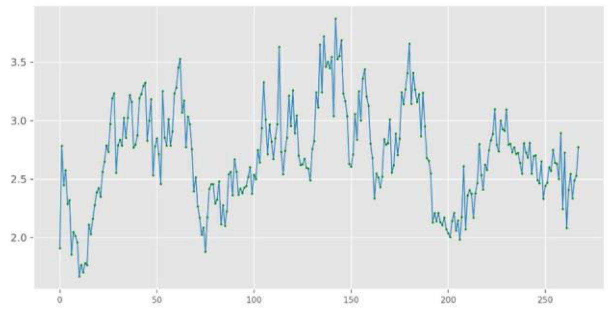

Fig.4 Maximum error of the algorithm (indoor scene)

Next, the actual imaging results are shown in Figures 5 and 6. From an overall perspective, whether in indoor or outdoor scenes, there is no noticeable difference between the floating-point results and the fixed-point results to the naked eye. The noise reduction effects have also achieved the desired results. The center point marked by the red box in the image is the point with the maximum error in that frame. Even at the location with the maximum error, the difference is virtually invisible to the naked eye.

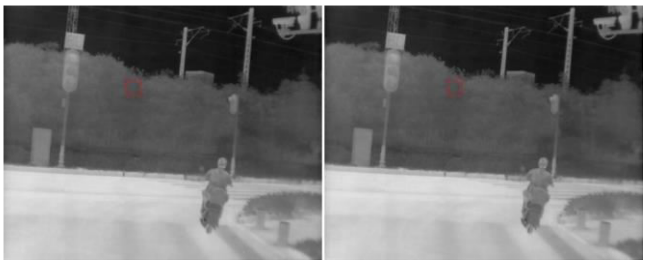
Fig.5 Outdoor scene effect (left - floating-point, right - fixed-point, maximum error 12)

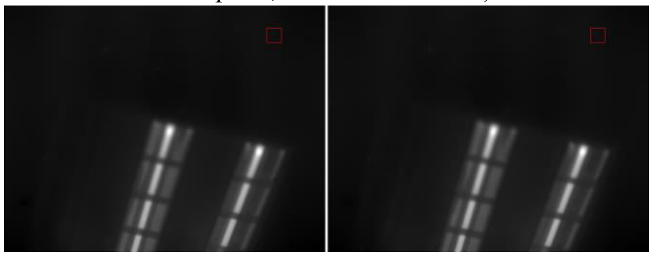
Fig.6 Indoor scene effect (left - floating-point, right - fixed-point, maximum error 4)

IV. FPGA-BASED HARDWARE DESIGN

A. *Runtime analysis*

First, the runtime of each module of the algorithm was measured, as shown in Table 1. This time is without the O2 optimization enabled. The original algorithm's frame rate parameter is 25, which means that the 0th and 12th frames will be updated at half the frame rate, and at one-fifth of the frame rate, the 0th, 5th, 10th, 15th, 20th, and 25th frames will be updated. The main purpose is to observe the computationally intensive parts of each module, analyze acceleration methods, and determine how to accelerate them. Based on these test data, the following explanations and analyses can be made:

DeNoise module: From the data, it can be seen that the runtime of the DeNoise module is 0.511408 seconds/frame, making it the most time-consuming module in single-frame processing. This is likely because the module requires complex denoising algorithms, involving image processing, filtering, and other operations, which results in a longer runtime.

Internal time-consuming parts of the module: The data shows that the Gaussian blur part of the DeNoise module takes up a large portion of its total runtime (about 63.21%). This indicates that the Gaussian blur operation in the DeNoise module is a time-consuming step that requires special attention and optimization.

Parameter update module: It can also be observed that the runtime of the parameter update modules (DeShading, DeFPN s32, DeStrip2, DeNU s32) is relatively short, but still occupies a certain proportion of the total time. These modules often require parameter updates based on different input conditions and environments, which may necessitate more frequent calculations and adjustments

Table 1 O2 optimized runtime is not enabled on ZYNQ

Module Name	Runtime	Image Update Method
Denoising Time without Parameter Update	0.691538	per frame
DeNoise	0.511408	per frame
Gaussian blur in DeNoise	0.323404	per frame
DeShading (parameter update)	0.073489	Half the frame rate
DeFPN s32 (parameter update)	0.068231	Half the frame rate
DeStrip2 (parameter update)	9.077074	Half the frame rate
DeNU s32 (parameter update)	1.260932	Frame rate one-fifth

B. The DeNoise module accelerates analysis and implementation

The DeNoise module has the longest runtime, occupying the majority of the total runtime (about 74.02%), indicating that the DeNoise module takes up a significant portion of the system's performance consumption. Implementing the DeNoise module can increase the system's frame rate by three to four times.

It is necessary to analyze the data dependencies and memory access patterns in the algorithm. The algorithm involves multiple matrix operations and data copying. In FPGA design, data copying can directly move the data. The data flow is quite complex, involving multiple matrix calculations. The data required later is temporarily stored in FIFO, and the subsequent data is processed using computation to ensure efficient data flow and storage. The entire module can be pipelined for hardware acceleration.

The estimated data throughput design is expected to reach 200 MHz, with one pixel of data (1 ppc) processed per cycle. The image size is 640×512, requiring a total of 327,040 cycles to complete the computation, with a calculation delay of several thousand clock cycles. Considering the data transmission delay, the total computation time will not exceed 2ms, achieving a good acceleration ratio.

Using the PS, the XAxiDma_SimpleTransfer function configures the source address, destination address, and transfer length registers to transfer data from DDR to the PL side for the DeNoise module to compute. DMA configuration commands and DeNoise configuration commands are passed through the AXI interconnect module to the AXI_DMA module and the DeNoise module, respectively.

Once the DeNoise module receives the control data, it begins waiting for data to be transferred from the PS-side DDR via the DMA's M_AXIS_MM2S port. The results obtained by DeNoise are transferred to the S_AXIS_S2MM module, and the DMA transfers the data back to the PS-side DDR, allowing the CPU to perform other computations.

C. Gaussian fuzzy acceleration analysis and implementation

The time test shows that the Gaussian blur occupies 46.76% of the total runtime without parameter updates. After FPGA acceleration, the frame rate can be roughly improved by less than half. Therefore, the first step is to accelerate the Gaussian blur module. The Gaussian blur consists of three parts:

(1) Gaussian function computation: The input parameters are fixed, so the results are always the same. Therefore, preprocessing can be performed to obtain the data, avoiding the need for real-time computation during the acceleration process.

(2) Row-column separated convolution for acceleration: To speed up the Gaussian blur computation, a row-column separation convolution method is used to reduce the computational load. In the row convolution, padding is applied first, and in FPGA, this can be achieved by changing the data path.

(3) Subsequent multiply-accumulate operations: These operations can be parallelized and pipelined in FPGA for acceleration. The row-column convolution can be reused in the hardware-accelerated code.

The algorithm contains multiple Gaussian blur modules, with differences in the image sizes they accept, the configurable kernel size (ksize), and the corresponding Gaussian weight matrix used for calculation. The rest of the data flow remains mostly the same. Therefore, using SpinalHDL to implement the Gaussian blur requires support for different ksize configurations. Different ksize values lead to different padding, mirrored padding data flows, different weights, and a varying number of multiply-accumulate operations. While describing this in Verilog may be difficult, SpinalHDL allows for recursive functions, describing addition trees, separating the code from the logic, and configuring different data flows. The data flow architecture for the hardware-accelerated Gaussian blur implementation is shown in Figure 7.

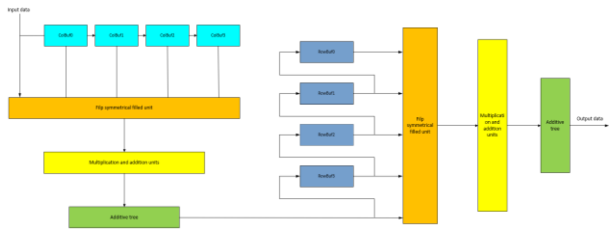
Input data
ColBuf0
ColBuf1
ColBuf2
ColBuf3
Flip symmetrical filled unit
Multiplication and addition units
Additive tree
RowBuf0
RowBuf1
RowBuf2
RowBuf3
Flip symmetrical filled unit
Multiplication and addition units
Additive tree
Output data

Fig.7 Gaussian fuzzy hardware accelerates the implementation of data flow architecture

D. Multi-core computing and algorithm optimization

To accelerate the parameter update time, the approach is to offload all non-parameter updates to be computed in the FPGA. Two CPU cores are reserved for parameter updates.

Non-parameter update denoising acceleration: Data is passed through the interface and then shifted left on the PL side to calculate the average value. The data is then stored in DDR, and an interrupt is triggered to the main core. The main core configures the DMA register to move the data and compute the DeNoise module. A weight loading module needs to be implemented, while the remaining tasks such as averaging, left shifting, and subtraction involve very little workload.

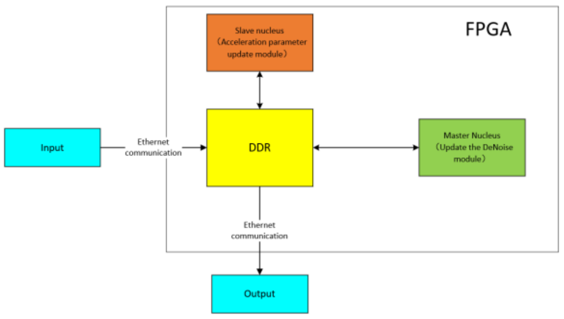
FPGA
Slave nucleus (Acceleration parameter update module)
Input
Ethernet communication
DDR
Master Nucleus (Update the DeNoise module)
Ethernet communication
Output

Fig.8 Multi-core computational flowchart

Figure 8 is the multi-core computing flowchart. The input image is transmitted through the Ethernet interface and stored in the FPGA's DDR memory. The main core and slave core inside the FPGA exchange data via an internal bus. The main core is mainly responsible for accelerating the DeNoise (denoising) module, while the slave core is responsible for accelerating the parameter update module. These two cores work in parallel without interference. The processed image data is then stored back in the DDR memory and sent through the Ethernet interface to the display interface for presentation.

CPU0 core: handles DeShading, DeFPN, updateSigmaN, and DeNU s32. Among these, DeNU s32 has a higher update frequency and is a computational bottleneck, requiring acceleration on the FPGA. The Gaussian blur for the DeNU module has a ksize of 17, and using Int64 bits is significantly faster than previous accelerations. The Gaussian blur can be modified, and operations such as sub and abs have already been implemented and can be directly invoked and pipelined.

CPU1 core: handles DeStrip2 acceleration. The loop of 9 iterations occupies 98% of the computation, so the focus is mainly on optimizing the loop. For loop unrolling optimization, after the calculation, the matGetMedCol and matGetMedRow are optimized to avoid expanding the matrix size, resulting in 640 and 512 numbers, respectively. By using these 1000 numbers to optimize the Avg computation time, the computational load can be reduced. The add and sub operations total 1000 numbers, which are sent to the FPGA to restore the matrix size. The Add and Sub operations are then integrated into the Strip process for pipelined computation in the FPGA.

V. Experimental Results and Analysis

A. *FPGA resource usage analysis*

In FPGA hardware design, effectively utilizing resources is one of the most crucial factors. Table 2 summarizes the resource usage in the FPGA.

Table 2 FPGA resource usage

Resource	Utilization	Available	Utiliaztion%
LUT	61894	87840	70.46
LUTRAM	19988	57600	34.70
FF	67823	175680	38.61
BRAM	126	128	98.44
URAM	3	48	6.25
DSP	124	728	17.03
IO	10	252	3.97
BUFG	6	352	1.70
MMCM	2	4	50.00

The most used resource is BRAM, which is used to store temporary data, parameters, and intermediate results during the image processing process. It can be used to store calibration coefficients, weight matrices, historical frame data, etc., in the calibration algorithm. These data typically need to be frequently read and written between image frames or during algorithm iterations. The usage rate reaches 98.44%. The next most used resource is LUT, which is primarily used to store and process the mapping relationship between pixel values and calibration coefficients. In the NUC algorithm, each pixel may have different calibration requirements, which are typically represented by calibration coefficients. LUT maps the original pixel values (such as grayscale values) to the corresponding calibration coefficients for pixel-level non-uniformity correction. The usage rate reaches 70.46%.

B. *Calculate time analysis*

For real-time images, computation time is an important performance metric. For a 640×512 size image, the original NUC algorithm achieves 2-3 frames per second, and with hardware running at 200 MHz, after FPGA acceleration, it can reach 66 fps. For a 1280×1024 size image, it can reach 30 fps after acceleration; for a 640×640 size image, it can reach 50 fps. Moreover, after FPGA acceleration, the denoising effect of the NUC algorithm remains the same.

The running time of the DeNoise module is about 0.000211s after acceleration. The running time of the DeNU module is about 0.039269s after acceleration. The running time of the DeStrip module, where the histogram and median calculation are run on the CPU, is about 40ms + 3 FPGA offload pipeline calculation times, approximately 43ms after FPGA acceleration. The histogram calculation can be completed in about 20ms + 3 FPGA offload pipeline calculation times (the pipeline calculation takes approximately 640×512 clock cycles).

Compared to typical hardware platforms like ARM, DSP, and ASIC, using FPGA for hardware acceleration offers excellent customizability, flexibility, and a short development cycle. ARM processors have a fixed architecture and instruction set, making it difficult to reconfigure hardware functions as needed, limiting flexibility in optimizing specific algorithms (such as the NUC algorithm). DSP designs are mainly used for digital signal processing, and their hardware structure is relatively fixed, making it hard to adapt flexibly to changes and optimization requirements of different algorithms. The design and manufacturing cycle of ASIC is long, making it unsuitable for applications in the rapid iteration and experimental development stages. In the early optimization phase of the NUC algorithm, such a long cycle would limit the rapid iteration and verification of the algorithm.

VI. Conclusion

This paper presents a parameter-update-based NUC algorithm that can dynamically adjust parameters based on the sensor's performance under different operating conditions. This capability allows the system to maintain high-quality imaging despite changes in time and environment. When performing non-uniformity noise removal, the algorithm uses fixed-point arithmetic exclusively. To ensure calculation precision, some variables undergo a process of amplification followed by reduction, and shift operations are employed for data scaling to guarantee efficiency. Additionally, a hardware design method for this algorithm based on FPGA implementation is proposed, realized on the Xilinx XCZU4EV-1SFVC784I. Experimental results show that it outperforms the original NUC algorithm, achieving 66 fps for a 640×512 image at a 200 MHz clock frequency, with a latency (including data input and output time) of under 40 ms.

REFERENCES

[1] Jiang H, Zhang X, Xu C, et al. A response function expansion method for cooled IRFPA with multiple neutral density filters under a variable integration time[J]. Infrared Physics & Technology, 2023, 133: 10486 0.

[2] Gong Y, Yu X, Ding Y, et al. Effective fusion factor in FPN for tiny object detection[C]//Proceedings of the IEEE/CVF winter conference on applications of computer vision. 2021: 1160-1168.

[3] Wang X, Song P, Zhang W, et al. A systematic non-uniformity correct ion method for correlation-based ToF imaging[J]. Optics Express, 202 2, 30(2): 1907-1924.

[4] Cong J, Lau J, Liu G, et al. FPGA HLS today: successes, challenges, a nd opportunities[J]. ACM Transactions on Reconfigurable Technology and Systems (TRETS), 2022, 15(4): 1-42.

[5] Chang Y, Zhou C, Hong Y, et al. 1000 fps hdr video with a spike-rgb hybrid camera[C]//Proceedings of the IEEE/CVF Conference on Comp uter Vision and Pattern Recognition. 2023: 22180-22190.

[6] Shenghui R , Huixin Z , Zhigang W ,et al.An improved non-uniformit y correction algorithm and its hardware implementation on FPGA[J].I nfrared Physics & Technology, 2017:S135044951630696X.DOI:10.10 16/j.infrared.2017.07.007.

[7] Celedon N, Redlich R, Figueroa M. FPGA-based neural network for nonuniformity correction on infrared focal plane arrays[C]//2012 15th Euromicro Conference on Digital System Design. IEEE, 2012: 193-200.

[8] Contreras J, Redlich R, Figueroa M, et al. A hardware Kalman-based offset estimator for nonuniformity correction on IRFPA[C]//Electro-O ptical and Infrared Systems: Technology and Applications IX. SPIE, 2 012, 8541: 247-256.

[9] Zou L, Fu Z, Zhao Y Z, et al. A pipelined architecture for real time correction of non-uniformity in infrared focal plane arrays imaging system using multiprocessors[J]. Infrared physics & technology, 2010, 53(4): 254-266.

[10] Lielāmurs E, Cvetkovs A, Novickis R, et al. Infrared Image Pre-Proce ssing and IR/RGB Registration with FPGA Implementation[J]. Electro nics, 2023, 12(4): 882.

[11] Liu Q, Wang H, Bian H, et al. Dual Mode High Speed Uncooled Short Wave Infrared Camera Based on FPGA[C]//Journal of Physics: Confe rence Series. IOP Publishing, 2021, 1952(3): 032042.

[12] Sheng Y, Yun L, Shi J, et al. Design of the cooled IRFPA real-time no n-uniformity correction system based on FPGA[C]//2011 International Conference on Optical Instruments and Technology: Optical Systems and Modern Optoelectronic Instruments. SPIE, 2011, 8197: 109-116.

[13] Setiadi D R I M. PSNR vs SSIM: imperceptibility quality assessment for image steganography[J]. Multimedia Tools and Applications, 2021, 80(6): 8423-8444.

[14] Amuru D, Zahra A, Vudumula H V, et al. AI/ML algorithms and applications in VLSI design and technology[J]. Integration, 2023, 93: 102048.

[15] Abud A A, Abi B, Acciarri R, et al. Design, construction and operatio n of the ProtoDUNE-SP Liquid Argon TPC[J]. Journal of Instrumentat ion, 2022, 17(01): P01005.