An FPGA-based Real-Time Nonuniformity Correction System for Infrared Focal Plane Arrays

Rodolfo Redlich, Gonzalo Carvajal, and Miguel Figueroa
Department of Electrical Engineering
University of Concepción
Concepción, Chile
e-mail: {rredlich, gcarvaja, mfiguer}@udec.cl

Abstract—Spatial and temporal nonuniformity in Infrared Focal Plane Arrays (IRFPA) severely degrades the quality of images obtained from modern infrared cameras. An efficient implementation of a nonuniformity correction algorithm is therefore necessary in real-time thermal-image visualization systems. This paper presents an FPGA-based implementation of the scene-based Constant Range algorithm for adaptive nonuniformity correction. The system processes an NTSC infrared video signal at 30fps in real time and consumes only 157 mW of power. The performance of our system is currently limited by the input video frame rate and the external memory bandwidth, but can be readily scaled to a frame rate of more than 250fps.

Keywords—FPGA, nonuniformity correction, scene-based calibration

I. INTRODUCTION

Infrared images are widely used in a variety of fields such as defense, surveillance, medical imaging, health care, condition monitoring, among others. Currently, a large portion of infrared image sensors are based on the InfraRed Focal Plane Array (IRFPA) technology [1]. An IRFPA is a group of independent infrared detectors forming a regular array on the focal plane. Each detector in the array converts the received infrared radiation into an electrical response representing a pixel of the image.

Despite rapid advances in their underlying technology, IRFPAs still suffer from nonuniformity (NU) limitations associated to variations in the response of different detectors in the array faced with the same stimulus. This undesired behavior is typically represented by a linear model that adds a gain and an offset to the response of each detector to the actual input irradiance. These parameters vary across different detectors in the array, and they also vary slowly in time according to the operational conditions (temperature, sensor degradation, etc.). Spatial and temporal nonuniformities result in a Fixed Pattern Noise (FPN) superimposed on the true image.

FPN is an intrinsic characteristic of IRFPAs, and consequently the acquisition system must implement nonuniformity correction (NUC) mechanisms to improve the quality of the images obtained from the sensor. There are two types of NUC techniques: reference-based and scene-based correction. The former uses reference images to estimate the nonuniformity parameters of each sensor in the array. For instance, the two-point calibration technique [2] records the pixel values of the images of two reference blackbodies, obtaining a linear system with two equations and two unknowns variables at each pixel to obtain the exact value of the offsets and gains. Because these parameters vary in time, it is necessary to periodically stop the normal operation of the device to recalibrate for the drift in the computed parameters. On the other hand, scene-based techniques use images from the incoming video sequence to continually estimate the parameters on-line, without interrupting normal system operation. Some classical implementations of these techniques are based on the Least Mean Square (LMS) algorithm [3], Kalman Filters [4], Constant Statistics [5, 6], and Constant Range [7, 8].

Integrating the nonuniformity correction with the image sensor requires a compact, low-power implementation of the algorithm, which must be also capable of keeping up with the frame rate of the corrected video output. In this paper, we present a dedicated hardware architecture that implements the Constant Range correction algorithm. Our implementation bases on a low-cost Spartan 3E Xilinx FPGA, and is able to process a 720x480-pixel NTSC video signal from an infrared camera at 30 frames per second (fps), consuming only 157mW of power. The architecture can be readily scaled to process video at more than 250fps and interface directly with the image sensor.

The rest of the paper is organized as follows: Section II describes the constant-range algorithm. Section III describes the architecture and design choices for our nonuniformity correction system. Section IV describe the actual hardware implementation of the system. Section V presents experimental results and discusses the scalability of the architecture. Finally, Section VI presents our conclusions and outlines future work.

II. CONSTANT RANGE ALGORITHM

Let us consider the following linear model representing the response $y^{(i,j)}$ of an individual detector in the IRFPA at
\[ y_{k}^{(i,j)} = a_{k}^{(i,j)} x_{k}^{(i,j)} + b_{k}^{(i,j)} \]  
where \( x_{k}^{(i,j)} \) is the input irradiance, and \( a_{k}^{(i,j)} \) and \( b_{k}^{(i,j)} \) are the gain and offset associated to the detector \((i,j)\) in the array, respectively. In general, temporal drift in the gain and offset is much slower than the changes in the infrared irradiance coming from an input video sequence, and therefore we can consider them as constants \( a_{(i,j)}^{(i,j)} \) and \( b_{(i,j)}^{(i,j)} \) when processing a few minutes of a video frame. The response of each individual detector is independent of the other sensors in the array, and then we simplify the notation by omitting the index \((i,j)\) in the following analysis.

The goal of any NUC technique is to find the values of the gain and offset using the information obtained from \( y_{k} \), and then apply an inverse model to obtain the actual infrared irradiance \( x_{k} \). In order to achieve this, the Constant Range (CR) method assumes that all the detectors in the array receive random values of input irradiance \( x_{k} \) with uniform distribution in the range \([x_{\text{min}}, x_{\text{max}}]\) within each sequence of frames. The values of \( x_{\text{min}} \) and \( x_{\text{max}} \) are known parameters depending on the particular sensor.

Considering the previous assumption, we can obtain the mean value and the standard deviation of (1) as:

\[ \bar{y} = a\bar{x} + b \]  
(2)
\[ \sigma_{y} = a\sigma_{x} \]  
(3)
and then obtain the values for the gain and offset as:

\[ a = \frac{\sigma_{y}}{\sigma_{x}} \]  
(4)
\[ b = \bar{y} - \frac{\sigma_{y}}{\sigma_{x}} \bar{x} \]  
(5)
where \( \bar{x} \) and \( \sigma_{x} \) are the mean value and standard deviation of the input irradiance \( x_{k} \), respectively. To simplify the analysis, we assume that the offset and gain are stationary parameters. Considering an uniform distribution for \( x_{k} \), we have:

\[ \bar{x} = \frac{x_{\text{max}} + x_{\text{min}}}{2} \]  
(6)
\[ \sigma_{x} = \frac{x_{\text{max}} - x_{\text{min}}}{\sqrt{12}} \]  
(7)

In order to follow the temporal drift in the parameters, the CR methods use a recursive estimation algorithm for \( \bar{y} \) and \( \sigma_{y} \) at time \( k \) using an exponential window:

\[ \bar{y}_{k} = (1 - \alpha) y_{k} + \alpha \bar{y}_{k-1} \]  
(8)
\[ \sigma_{y,k} = (1 - \alpha) |y_{k} - \bar{y}| + \alpha \sigma_{y,k-1} \]  
(9)
where \( \alpha \), with \( 0 < \alpha < 1 \), is the time constant controlling the size of the window. Small values of \( \alpha \) emphasize recent information, so that the correction mechanism can adapt quickly to variations in the operation point. However, they may also reduce the quality of the compensation. Large values of \( \alpha \) achieve higher-quality correction over a longer period of time, but can cause undesired effects such as ghosting artifacts [6].

We can use expressions (6) to (9) to estimate the gain and offset of the detector from (4) and (5), respectively; and then obtain the actual irradiance from the inverse model of (1) as:

\[ x_{k} = \frac{(y_{k} - \bar{y}) \sigma_{x}}{\sigma_{y,k}} + \bar{x} \]  
(10)

The previous analysis is valid for each individual pixel in the image. As we can see, the expression for each pixel is independent of its neighbors, and thus we can speed up the correction of an input frame by processing multiple pixels simultaneously. In the next section, we describe a hardware architecture that acquires a video stream from the camera’s analog video output, and uses multiple pixel processors to correct the video signal in real-time.

### III. Architecture

The pseudocode in Figure 1 shows the CR algorithm described in the previous section for the correction of each pixel \((i,j)\) on a frame at time \( k \). The algorithm first receives the current response \( y_{k} \) from the detector, and then computes the mean value and standard deviation of \( y \) estimated at time \( k - 1 \). In the next step, the algorithm computes the current mean value and standard deviation of \( y \) using expressions (8) and (9), respectively.

At this point, we are in condition to compute the actual irradiance \( x_{k} \) using the inverse model (10). However, we can see that this expression has a potential arithmetic overflow when \( \sigma \to 0 \). In practice, this means that the captured image has been static for a while and there is no significant variation between consecutive frames. To avoid this problem, we compare the current value of \( \sigma \) to a preset threshold (determined empirically), so that if \( \sigma \) exceeds the threshold we compute the correction using the current system parameters. If the value of \( \sigma \) is lower than the threshold, then the system computes the value of \( x_{k} \) using the parameters determined in the previous iteration.

Figure 2 shows a block diagram of the proposed pipelined architecture for pixel NUC processor using the CR algo-
algorithm. We consider the following optimizations to reduce the computation time within a single pixel:

- Simultaneous computation of the factors \((1 - \alpha)y_k\) and \(\alpha\bar{y}_{k-1}\) in (8)
- Simultaneous computation of the factors \((1 - \alpha)|\bar{y}_k - y_k|\) and \(\alpha\sigma_{y,k-1}\) in (9)
- All the multiplications involve one constant coefficient. We implement these operations using optimized Constant Coefficient Multiplier (KCM) [9] units, which use precomputed multiplication tables to reduce the logic and time required by traditional full multipliers.
- The value of \(y_k - \bar{y}_k\) is used in both (9) and (10). We avoid recomputing this value by propagating the value computed in the estimation of \(\sigma_{y,k}\) and reusing it in the correction stage.

The architecture works as follows. The processor starts sampling the current value of the input \(y_k\) and reads the previously stored recursion parameters \(\bar{y}_{k-1}\) and \(\sigma_{y,k-1}\) from memory. These values are then propagated to the stages where they are actually needed using shift registers. We represent this behavior using the shift blocks in Figure 2, where the number in the block represents the propagation stages in clock cycles. The first pipeline stage computes the current mean \(\bar{y}_k\) by adding the output of two parallel KCM blocks. In the second stage, block S1 computes \(y_k - \bar{y}_k\) using a sign-magnitude representation, so that the absolute value used to compute \(\sigma_{y,k}\) is obtained by simply omitting the sign bit. Simultaneously, the block S2 computes the alternative term \(y_k - \bar{y}_{k-1}\) using a similar representation. The third stage compares the current value of the standard deviation to the threshold, and decide whether to use the current values of the mean and standard deviation or the previous ones to compute the correction. The correction stage includes a division block based on classic digit recurrence techniques[10], which produce a single quotient-digit at each clock cycle. In order to reduce the complexity of this block, we simply omit the sign of the operands at this stage, and use it later to select the correct sign of the corrected pixel.

In our current implementation, the architecture pipeline has a total latency of \(3n_M + n_S + n_D + 6\) clock cycles, where \(n_M\), \(n_S\) and \(n_D\) are the individual latencies associated to the blocks performing multiplication, subtraction, and division, respectively. All these latencies are related to the bit-length of the involved operands. After this initial latency, the system delivers one corrected pixel per each clock cycle.

IV. IMPLEMENTATION

We use the architecture proposed in the previous section to implement an on-line NUC mechanism for NTSC infrared cameras using a Digilent Nexys 2 development board based on a Xilinx Spartan 3E XC3S500 FPGA. Figure 3 shows a diagram of the system. The video acquisition block outputs a digital video signal. The NUC system on the FPGA receives this signal and processes the individual pixels according to the CR algorithm, using the recursion parameters stored in an external SDRAM memory. Finally, the FPGA sends the corrected signal to a VGA display for visualization. This section describes the individuals components of the proposed system.
A. System Description

The video acquisition system consists of an analog IR camera attached to a Digilent VDEC1 peripheral board with an ADV7183B chip, which outputs a digital video stream to the FPGA board. From the analog NTSC signal, the video decoder generates an interlaced video stream at 30fps, with a frame size of 858x525 pixels and 24 bits per pixel. Each pixel has three components of eight bits each: one for luminance (Y) and two for chrominance (Cb and Cr). For gray scale video from the infrared camera, only the luminance Y is relevant. Thus, each pixel in our system has an eight-bit representation in the range $[x_{min} = 0, x_{max} = 255]$.

The digital video signal arrives to the FPGA in horizontal lines of 858 pixels each. The first 720 pixels of each line correspond to active video containing the image information. The remaining 138 pixels correspond to a horizontal blanking stage used for synchronism of analog video signals and thus contain no image information. The individual pixels arrive at a frequency of 13.5 Mhz, and therefore the periods of active video ($T_{av}$) and blanking ($T_{b}$) are:

$$T_{av} = 720 \cdot \frac{1}{13.5 \text{ MHz}} = 53.3 \mu s$$  \hspace{1cm} (11)$$

$$T_{b} = 138 \cdot \frac{1}{13.5 \text{ MHz}} = 10.2 \mu s$$  \hspace{1cm} (12)$$

Because the algorithm corrects each pixel independently of the other pixels in the frame, each individual pixel has its own recursion parameters $\hat{y}_{k-1}$ and $\sigma_{k-1}$. With an active frame size of 720x480 pixels (there are 45 lines of vertical blanking), and assuming a fixed-point representation of 16 bits (eight integer and eight decimal) for each parameter, we need over 5Mb of memory. This requirement exceeds the available memory in the embedded BlockRAMs (BRAMs) of the Spartan 3 chip, therefore we need to store these parameters in external RAM.

In order to perform the real-time NUC correction, the system stores the pixels of a complete line during the active video stage, processes them during the blanking stage, and sends the corrected pixels to a VGA controller while receiving the following video line. We illustrate this process in Figure 4. During the active video stage, the system stores the pixels of a line as they arrive in an input buffer, reads the corresponding recursion parameters for each pixel from the external memory, and stores the updated parameters for the pixels computed on the correction stage of the previous line. During the blanking stage the system processes the pixels of the current line and stores the corrected values in the output buffer, which is then sent to the VGA controller for visualization during the processing stage of the following line.

B. Implementation of the algorithm on the FPGA

The FPGA chip contains the NUC unit and the interfaces to the external devices. The Line-Field-Decoder block receives the external video signal and selects the luminance component Y of each pixel as they arrive. Additionally, this block sends information to the Memory Management Unit (MMU) to extract the corresponding recursion parameters for each pixel from the current line. All this information is stored in the input buffer during the active video stage.

The NUC unit starts operating during the blanking stage of the current line. As described in Section III, it reads the current value and recursion parameters associated to each pixel, and performs the correction in a pipelined fashion, storing the result in the output buffer. The implemented NUC unit illustrates a key aspect of the proposed architecture: because the correction process for each pixel is independent of the others, it is possible to process multiple pixels simultaneously to accelerate the execution of the algorithm. We achieve this by instantiating as many processing units as needed, restricted only by the amount of available logical resources in the FPGA.

The minimum frequency to process a complete video line during the corresponding horizontal blanking stage, considering an equitable distribution of pixels between $N$ independent processing units, is given by the following expression:

$$f_{min} = \left( \frac{n_{lat} + n_{pix}}{N} - 1 \right) \frac{13.5}{138} \text{ [MHz]}$$  \hspace{1cm} (13)$$
where \( n_{\text{lat}} \) is the total latency of the pipeline, and \( n_{\text{pix}} \) is the number of pixels per line.

Considering a 16-bit numeric representation, the KCM and subtraction blocks deliver the result in 3 clock cycles, while the divider use 16 clock cycles. Thus, the total latency for our pipeline is 34 cycles (see Section III). With 720 pixels per line, we need a minimum clock frequency of 73.66 MHz with a single processing unit (\( N = 1 \)), and 38.44 MHz with two processing units (\( N = 2 \)).

In the current implementation, we opted for instantiating two pixel processors driven by a 60 MHz clock, which is the clock frequency required for the memory controller. This design choice allows us run the entire system from a single clock (except for the video acquisition block, which uses an independent clock), thus simplifying the design and saving a Digital Clock Manager (DCM) unit, which are responsible for a significant portion of the power budget. As a result, we can correct a complete video line in 6.55\( \mu \)sec, corresponding to a 64.1% of the length of the horizontal blanking stage.

V. EXPERIMENTAL RESULTS

We synthesized a gate-level implementation of our system from a Verilog description using the Synopsys Synplify design tool, and mapped it onto the Spartan 3E XC3S500 FPGA. Tables I and II show the utilization of logical resources in the FPGA for a single pixel processor, and for the entire implemented system including two pixel processors and the interfaces for external devices, respectively. Each pixel processor utilizes less than 15% of the available resources on the Spartan chip.

Table III summarizes the power consumption of the implemented system. The chip consumes a total of 157mW of power. The video processing hardware, including both pixel processors, the line field decoder, memory controller, VGA controller and I/O buffers, consume 43mW. The remaining 114mW are dissipated in the clock distribution network, the two DCMs, and the 104 IOBs used to interface with the video acquisition board, external RAM, and VGA output.

Timing analysis of the synthesized circuit indicates that we can run the pixel processors at up to 118.5MHz. In the current implementation, the performance of the system is limited by the memory bandwidth and the input video rate, so there is no advantage on running the processors above 38.44 MHz. However, our simulation and synthesis results show that, omitting the bottleneck in the external memory bandwidth, we could scale the system up to six processors on the same XC3S500 chip running at 100 MHz.

Such a system could process a video stream of over 257 fps at the same 720x480-pixel resolution, which is around 2.7 times faster than an equivalent software implementation in C, running over a 2.27GHz Intel Core i3 processor. A larger FPGA would accommodate even more processors and scale the system performance further. We remark that the reported speed up is only related to the implementation of the algorithm, without considering the peripherals and interfaces for the acquisition and visualization system.

We tested our implementation processing 400 frames of video from two different infrared cameras: An Amber AE-4128 featuring 128x128-pixel resolution and a Flir JADE-UC33 with a resolution of 320x240 pixels. The VDEC1 video acquisition board scales both images to 720x480 pixels to be processed by the FPGA. Figure 5 shows the result of using our system to apply nonuniformity correction to the output of two infrared cameras, where we have have scaled the output images to their original resolution to provide a better comparison between the original and corrected frames. The example frames on the left side of Fig.5 show the uncorrected camera output. The frames on the right side show the output of our hardware implementation of the Constant Range algorithm. It is easy to appreciate the improvement in the quality of the image.

In order to evaluate the numeric performance of our implementation, we compared our results to a C implementation which uses double-precision floating point arithmetic. We use two different quality indexes: normalized Root Mean Square Error (RMSE) and roughness \( \rho \). RMSE is the RMS value of the error between the corrected output and an external reference given by a two-point off-line calibration using a blackbody. The error is normalized to the pixel range (0-255) and averaged over all pixels in each video frame:

\[
\text{RMSE}(I, I_{pc}) = \frac{1}{255} \sqrt{\frac{1}{N \cdot M} \sum_{i=1}^{N \cdot M} (I(i) - I_{pc}(i))^2} \tag{14}
\]

Roughness \( \rho \) measures the amount of high spatial-frequency content in the image, and is obtained by comput-
Camera Output
Corrected frame
(a) Camera: Amber AE-4128

Camera Output
Corrected frame
(b) Camera: Flir JADE-UC33

Figure 5: Example frames from two IR cameras.

ing a convolution between each frame and an edge-detection mask $h$:

$$\rho(I) = \frac{||h * I||_1 + ||h^T * I||_1}{||I||_1}$$

Table IV compares the results obtained by the software implementation of the CR algorithm to those obtained by our system. We show only the results for the Amber camera, as the results for the Flir device were similar. We computed the RMSE and roughness indexes for each of the 400 frames of video and report both their mean and standard deviation. As the table shows, the quality of our results are very close to those obtained from the double-precision software implementation, to the point that the image outputs are visually indistinguishable. Our resulting hardware system is more than eight times faster than an equivalent software implementation in C and, because of the small size and low-power requirements, is also well-suited for integration with the image sensor in the camera. We are currently considering to replace the FPGA with a custom VLSI implementation to reduce the size and power requirements even further.

VI. Conclusions

We presented a FPGA-based implementation of the scene-based Constant Range NUC algorithm for real-time processing of infrared video sequences. The proposed architecture exploits hardware parallelism and features a pipelined approach to accelerate the sequential correction process of the pixels in a video frame. We also discussed the scalability of the architecture for further acceleration by processing multiple pixels simultaneously.

The implemented hardware system uses a low-cost Spartan 3E Xilinx FPGA to perform real-time correction of grayscale video sequences in the NTSC standard, at 30fps with a frame size of 720x480 pixels, and dissipating 157mW of power. The quality of the correction is equivalent to a software implementation of the algorithm using double-point arithmetic. We can scale our system to six pixel processors in the same FPGA chip, processing more than 250fps.

We are currently working on optimizing the architecture, and integrating a custom-VLSI implementation with the
image sensor in an infrared camera. We are also exploring the hardware implementation of other on-line correction algorithms.

ACKNOWLEDGMENTS

This work was funded by CONICYT Grant PFB-0824. Simulation and synthesis were done with EDA software donated by Synopsys.

REFERENCES