

Focal Plane Array Sensor Readout Correction on a Reconfigurable Processor

Jagdish Sabde, David Buehler, Gregory Donohoe

Center for Advanced Microelectronic and Biomolecular Research, University of Idaho
sabd1073@uidaho.edu, bueh1172@uidaho.edu, donohoe@ece.uidaho.edu

Abstract

This paper describes Focal Plane Array (FPA) sensor readout correction using the Reconfigurable Data Path Processor (RDPP). The RDPP is a low power, radiation-tolerant processor for data intensive, streaming applications such as image and signal processing. The RDPP employs fixed-point arithmetic for compactness, in order to fit 16 arithmetic-logic units (ALUs) and 16 multipliers on a single chip. The output of a Focal Plane Array can be corrupted because of dark current through photo-diode sensors, non-compensatory circuits of conversion amplifiers, and process and temperature variations. Using the RDPP, we implemented a simple and efficient method to correct the corrupted data and produce a corrected image similar to the original image. The RDPP implementation gives high throughput at modest clock rates. We demonstrate the implementation on an RDPP simulator.

Index Terms— Focal Plane Array readout correction, Non-uniformity correction, Reconfigurable computing.

1. Introduction

THE Reconfigurable Data Path Processor (RDPP) is a low power, radiation-tolerant reconfigurable multiprocessing chip being developed for high-throughput, data-intensive processing aboard spacecraft. The RDPP consists of 16 reconfigurable on-board processing elements, programmable interconnect, programmable I/O ports, and a run-time execution module with integrated program memory. Each processing element (PE) has a hardware multiplier, an arithmetic/logic unit, data path formatting logic, and data path selection logic, operating on a 24-bit-wide data path. In addition, there are four 24-bit bi-directional I/O ports and one dedicated output port. The RDPP implements a synchronous data flow computational model. The RDPP operates in two phases: (1) configuration, and (2)

execution. In configuration, the processing elements are programmed for specific functions, programmable interconnects are configured to form a processing pipeline, and a run-time program is loaded. In the execution phase, the RDPP reads and processes data from an input stream and writes to an output stream.

The Focal Plane Array sensor readout application was chosen to illustrate two features of the RDPP: reconfigurability, and run-time data path switching.

The remainder of the paper is organized as follows. In section 2, we define the sensor readout correction problem and different measurement errors to be corrected. The RDPP implementation of the sensor readout correction problem is described in section 3. The simulation results and the conclusions are presented in sections 4 and 5, respectively.

2. Sensor readout correction problem definition

Figure 1 shows a simplified schematic diagram of a single pixel in an integrated focal plane array. The basic sensor is a photoconductive diode, which produces a photo-induced current, I_{ph} , proportional to the amount of light incident on the diode. All diodes have some leakage current. When no light is incident on a photo detector, some leakage current will flow through the diode: this is called dark current (I_{dc}). By Kirchoff's current law, these two currents add to form the total current out of the detector:

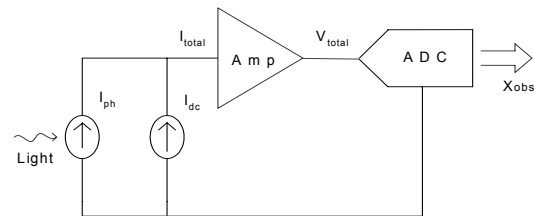


Figure 1: Simplified schematic model of pixel readout.

$$I_{total} = I_{ph} + I_{dc} \quad (1)$$

Co-located with the sensor diode is a simple current - to - voltage conversion amplifier, labeled Amp in the figure. The amplifier produces an analog voltage, V_{total} , which feeds an analog to digital converter, ADC, to produce the observed digital pixel value X_{obs} .

Each pixel has its own current-to-voltage conversion amplifier. In order to fit hundreds of these amplifiers on a single die, each amplifier is necessarily very simple, consisting of two or three transistors at most. Such simple amplifiers do not have sophisticated compensation circuitry, so they are particularly sensitive to process variations and temperature. The photo diodes also vary due to fabrication process limitations. Thus, each sensor has a unique dark current, and each conversion amplifier has a unique gain. This results in a gain error and an offset error at each pixel in the image. In order to reduce measurement errors, each digital pixel is corrected for its particular gain and offset error. This is called non-uniformity correction.

Output of the FPA can also be corrupted by “hot” or “dead” pixels. The hot pixels are caused by radiation: ions lodging in the P-N junction of the photodiode induce a steady current, which produces a constant bright pixel. Dead pixels, which can also be caused by radiation, are due to failure of the sensor readout circuits, resulting in a constant pixel value. Hot and dead pixels can be replaced by a spatial average of neighbors.

2.1. Non-uniformity correction

To compensate for gain and offset errors, we must compute correction parameters. First, assume that there is no gain error. Then the observed digital pixel value x_0 is a function of the voltage V_{total} : $x_0 = f_{ADC}(V_{total})$. This voltage, in turn, is a function of I_{total} : $V_{total} = f_{Amp}(I_{total})$. Thus, the no-gain-error value is $x_0 = f_{ADC}(f_{Amp}(I_{total})) = f(I_{total})$, where $f(.)$ is the combined gain function. From (1):

$$x_0 = f(I_{ph} + I_{dc}) \quad (2)$$

We model the gain error of the system as a unique gain factor g , so that the observed value for any input is:

$$x_{obs} = g \cdot x_0 = g \cdot f(I_{total}),$$

or

$$x_{obs} = g \cdot f(I_{ph} + I_{dc}) \quad (3)$$

In an ideal or standard pixel, $g = 1$, and (3) reverts to (2).

To find the correction parameters, we perform two experiments:

1. Block all light to the sensor and measure the output due to dark current, x_{dc} .
2. Illuminate the sensor with a standard light source, and measure the observed output x_{obs} .

Since $I_{ph} = 0$, the dark current measurement is:

$$x_{dc} = g \cdot f(I_{dc}) \quad (4)$$

Now stimulate the sensor with a standard light source to produce a photo current $I_{ph} = I_{ref}$, and observe the output:

$$x_{obs} = g \cdot f(I_{dc} + I_{ref}) \quad (5)$$

If the both the amplifier and the analog-to-digital converter are linear, then the overall function $f(.)$ is linear. Invoking the superposition property of linear systems on (5), We have:

$$x_{obs} = g \cdot f(I_{dc}) + g \cdot f(I_{ref}) \quad (6)$$

We define the expected output from the standard source

$$x_{ref} = f(I_{ref}) \quad (7)$$

Substituting (3) and (7) into (6)

$$x_{obs} = x_{dc} + g \cdot x_{ref} \quad (8)$$

Solving for g , we have

$$g = (x_{obs} - x_{dc}) / x_{ref} \quad (9)$$

We assume that the desired pixel value, x_{ref} , is known. It can be obtained from a separate calibration procedure or from system knowledge.

In operation, we convert each new observed value x_{obs} to a corrected value x_{corr} . We substitute x_{corr} for x_{ref} in (8) and solve:

$$x_{corr} = (x_{obs} - x_{dc}) / g \quad (10)$$

We define the gain correction factor as $g = 1/a$ and the correction equation becomes

$$x_{corr} = a \cdot (x_{obs} - x_{dc})$$

2.2. Bad pixel correction

In order to replace the bad pixels, we need to mark the “hot” or “dead” pixels and replace these with the average of previously corrected pixel values. Hot and dead pixels are easy to detect, as their value does not change with

variations in scene brightness. Offset correction values are never negative, so we can use negative value as a special code for bad pixels. We set the offset value of bad pixel to a hexadecimal value of *FFFFFF*, or all ones. At run time, the RDPP, detecting this special code, will output the neighborhood average instead of the non-uniformity corrected pixel value.

3. RDPP Implementation

All large memories are external to the RDPP; only small amounts of local memory are contained in registers within the RDPP. This application requires two image memories. At execution time, one memory holds the gain correction factor, a_k and other memory holds offset correction factor, b_k . In addition, the First-In First-Out (FIFO) memory is used as a pixel delay element, sufficient to buffer one line of pixels. Figure 2 shows the board level configuration for sensor readout correction.

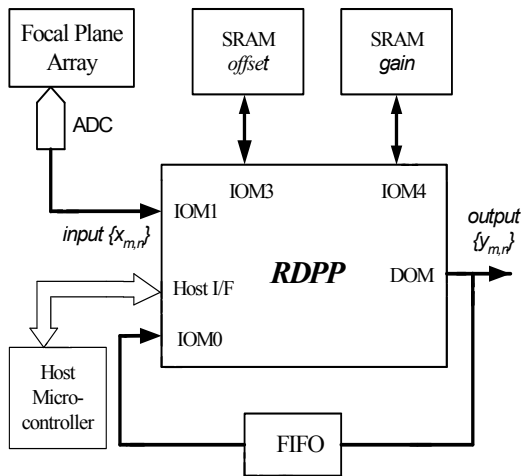


Figure 2. Hardware components for sensor readout correction.

The RDPP has four configurable 24-bit I/O modules, IOM0 through IOM4, and one dedicated output module, DOM. Each I/O module can be set up as either an input or an output at configuration time. RDPP IOM0 (input) receives “raw” pixel data, while IOM1 receives the data delayed by 256 time samples. Inputs IOM2 and IOM3 receive the calibration memories. External circuitry, controlled by the host processor, will enable the appropriate destination for the data during different configurations.

Figure 3 shows the components of the processing element. The RDPP processing element has three 24-bit inputs and two 24-bit outputs, plus a 1-bit control output. The processing modules consist of a 24-bit multiplier, an

arithmetic-logic unit (ALU), data-path formatting modules, and switching elements. The RDPP is built on 24-bit-wide, signed data paths. Some internal data paths are 48 bits wide. The multiplier accepts two 24-bit inputs and produces a product of 48-bits. In order to receive this product, the ALU accepts 48-bit inputs and produces a 48-bit output. Data paths are expanded from 24 to 48 bits with PAD modules. Data paths are narrowed from 48-bits to 24-bits using SHIFT, ROUND and CLIP operations embedded in the Format block.

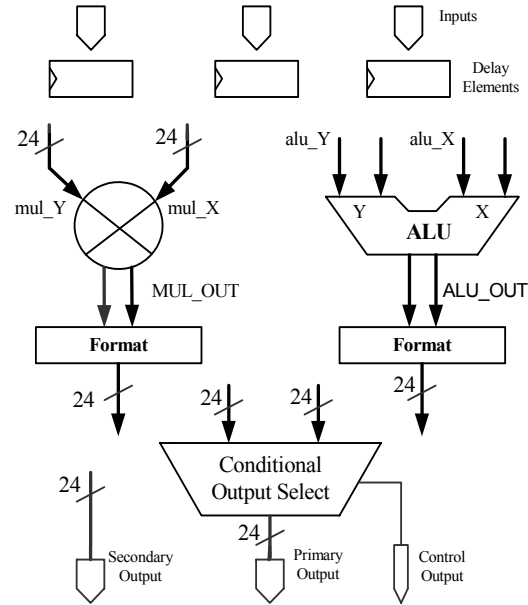


Figure 3. Processing element components.

Sensor readout correction on the RDPP requires two operational phases:

1. Calibration Phase:
 - a) Acquire calibration data and store in two memories, one for gain, and the other for offset.
 - b) Encode “bad pixel” state in calibration memory for all pixels that must be replaced at execution time.
2. Correction Phase:
 - a) Read pixel from sensor, and corresponding calibration memories, and adjust offset and gain.
 - b) Compute neighborhood spatial average, and replace bad pixels with this average.

3.1. Calibration Phase Overview

Calibration proceeds in three steps, requiring three separate RDPP pipelines.

1. Find the dark current pixel value b_k . Close the aperture, read an image, and test for bad pixels. If $b_k < T$ for predetermined threshold T , then store b_k in memory. Otherwise, $b_k = FFFFFFFF$, the special “bad pixel” code.
2. Flat-field and fixed-pattern calibration. Present a known light source. Capture an image and calculate a reference sensor output, x_{ref} . Store this in an internal RDPP data register.
3. Compute flat-field gain factor for a known light source. For each pixel, grab pixel x_k . Gain factor for sample k : $a_k = x_{ref}/(x_k - b_k)$.

3.2. Calibration Phase Implementation

3.2.1 First Configuration

Function: Find the dark current pixel value b_k . We close the aperture, read an image, and test for bad pixels.

If the difference between hot pixel value (HPV) and b_k is greater than zero, then store b_k in memory. Otherwise, set $b_k = FFFFFFFF$, the special “bad pixel” code. Figure 4 shows flow diagram for computing offset error and marking bad pixels.

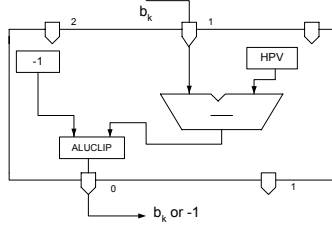


Figure 4. Flow diagram of Calibration Phase to compute offset error and mark bad pixels.

3.2.2. Second Configuration

Function: Flat-field and fixed-pattern calibration for a known light source.

Capture an image and calculate a reference sensor output, x_{ref} . Store this in an internal RDPP data register for use in third calibration configuration.

The reference image value x_{ref} is the sample mean of the all of the pixels in the image.

$$\text{The expression is: } x_{ref} = \frac{1}{k} \cdot \sum_{k=0}^{k-1} x_k$$

For a 256X256 pixel image, $k = 2^8 \times 2^8 = 2^{16} = 65536$. The maximum pixel value from the 12-bit sensor is $2^{12} - 1 = 4095$. In the worst case, the sum could produce a value with an upper bound of $2^{12} \times 2^{16} = 2^{32}$. Since the RDPP can only store numbers up to 2^{32} without overflow, pre-scaling is required. Our solution is to

compute the sample mean of each row, then compute the sample mean of these means.

3.2.3. Third Calibration Configuration

Function: Compute flat-field gain factor for a known light source.

The gain factor is computed as $a_k = x_{ref}/(x_{obs} - b_k)$, where x_{ref} is the reference pixel value, stored in data register in the processing element. Subtracting the offset error from observed pixel value results in an offset corrected value $d = x_{obs} - b_k$. We have chosen division by repeated multiplication [B. Parhami, 6] to divide x_{ref} by d . To compute the ratio $a_k = x_{ref}/d$, one can repeatedly multiply x_{ref} and d by a sequence of m multipliers

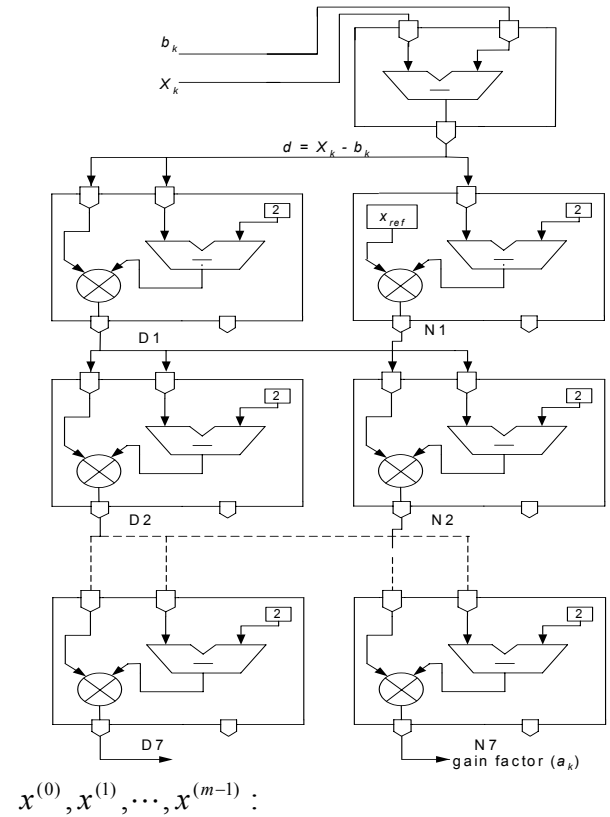


Figure 5. Computation of inverse gain correction factor.

$$a_k = \frac{x_{ref}}{d} = \frac{x_{ref} \cdot x^{(0)} \cdot x^{(1)} \dots x^{(m-1)}}{d \cdot x^{(0)} \cdot x^{(1)} \dots x^{(m-1)}}$$

If this is done in such a way that the denominator converges to 1, the numerator will converge to a_k . Assume a bit-normalized fractional divisor d in $[1/2, 1]$. If this condition is not satisfied initially, it can be made to hold by appropriately shifting both x_{ref} and d . The

corresponding convergence computation is formulated as follows:

$$d^{(i+1)} = d^{(i)} \cdot x^{(i)} \quad (11)$$

Set $d^{(0)} = d$; make $d^{(m)}$ converge to 1.

$$x_{ref}^{(i+1)} = x_{ref}^{(i)} \cdot x^{(i)} \quad (12)$$

Set $x_{ref}^{(0)} = x_{ref}$; obtain $\frac{x_{ref}}{d} = a_k \approx x_{ref}^{(m)}$.

By selecting,

$$x^{(i)} = 2 - d^{(i)}$$

This transforms the recurrence equations into:

$$d^{(i+1)} = d^{(i)} \cdot (2 - d^{(i)}) \quad (13)$$

Set $d^{(0)} = d$; iterate until $d^{(m)} \approx 1$.

$$x_{ref}^{(i+1)} = x_{ref} \cdot (2 - d^{(i)}) \quad (14)$$

Set $x_{ref}^{(0)} = x_{ref}$; obtain $\frac{x_{ref}}{d} = a_k \approx x_{ref}^{(m)}$.

Figure 5 shows the implementation of above equations to compute the inverse gain correction factor, a_k .

As previously noted, computations on values from the mathematical field of reals are implemented using fixed-point values in the RDPP. Fixed-point implementations of computations are historically challenging to create, because proper alignment of operands for mathematical operations have been determined and implemented by hand.

As part of the RDPP software project, a fixed-point data path design tool is being created. The tool is currently at a preliminary stage of development, and was used in conjunction with hand-design of the computations to determine operand alignment operations and verify correctness of the fixed-point computations performed by the division algorithm.

The fixed-point data path design tool is the focus of ongoing research [1]. It is expected that it will be developed into a semi-automatic fixed-point data path design system, with features including heuristic-based automated fixed-point optimization and computational error tracking.

3.3. Correction Phase Overview

The correction phase implements two parallel processes chains:

1. Compute the non-uniformity correction
2. Calculate a spatial mean of four neighboring pixels. Replace “bad” pixels with the spatial mean of neighbors.

The processing chains join when the RDPP checks the offset correction factor for the special “bad pixel” code. At run time, the RDPP, detecting this special code, will output the four neighboring pixels average instead of the flat-field corrected value. The expression for flat-field correction is: $f_k = a_k(x_k - b_k)$, where x_k is the input pixel, f_k is the output pixel, a_k is the gain factor, and b_k is the offset correction factor, or dark current pixel value. The expression for the spatial mean is:

$$s_k = s_{m,n} = \frac{1}{4}(x_{m-1,n-1} + x_{m,n-1} + x_{m+1,n-1} + x_{m-1,n})$$

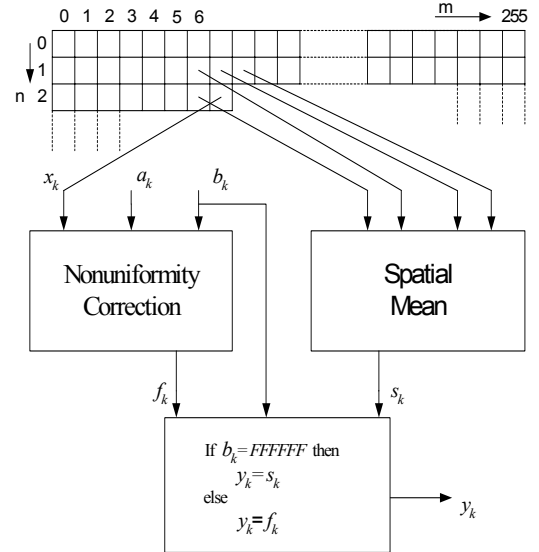
where m and n are horizontal and vertical indices, respectively as shown in figure 6.

The set of pixels chosen for the average are pixels already received from the sensor, i.e. this is a causal algorithm. This simplifies memory management, and reduces latency, over more general schemes.

The final output, y_k , is determined by the coding of the “bad pixel” bit in b_k . If $b_k = FFFFFFFF$, then $y_k = s_k$. Otherwise, $y_k = f_k$.

3.4. Correction Phase Implementation

Figure 6. Block diagram of correction phase.



Unlike the calibration phase, The correction phase has only one configuration. The non-uniformity correction computation is straightforward, and only requires a single processing element with three inputs as shown in figure 7. One input receives the gain correction factor a_k , one the offset correction b_k , and one, the current input sample x_k . It implements the expression: $f_k = a_k(x_k - b_k)$.

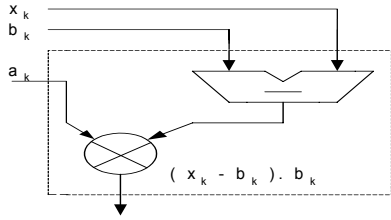


Figure 7. Non-uniformity correction computation.

The spatial mean computation requires two inputs: x_k and x_{k-255} , the latter being the output of the external FIFO. All of the other pixel delays are generated inside the RDPP. Figure 6 shows a block diagram of the three logical components of the correction phase. One logical block computes the non-uniformity correction $f_{m,n}$, or f_k , one computes the spatial mean s_k , and one decides which to output, depending on the “bad pixel” code. Figure 8 shows how the pixel delays are computed for the spatial mean. The external FIFO provides a delay of 255 samples. From the external delayed sample and raw input, processing elements compute the other required delays.

The entire configuration uses six processing elements for the spatial mean calculation, one for non-uniformity computation and one for output selection, for a total of eight processing elements.

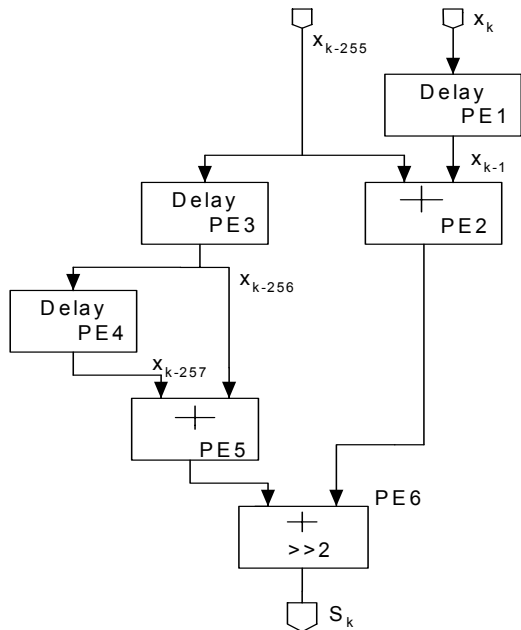


Figure 8. Flow diagram of spatial mean calculation.

4. Results

Figure 9 shows the original image.



Figure 9. Original image.

Figure 10 displays the 256x256 corrupted image frame, which is the output of FPA. The image is corrupted by gain error, offset error, and bad pixel values.



Figure 10. Corrupted (input) image.

Figure 11 shows the final corrected output obtained from the RDPP simulator. The corrected image is similar to the original image.



Figure 11. Corrected (output) image.

5. Conclusions

We have demonstrated an implementation of focal plane array readout correction on the Reconfigurable Data Path Processor¹.

This application demonstrates two key features of the RDPP architecture: reconfigurability, and conditional data path selection. The implementation requires three configurations to acquire the calibration data, and one to acquire and correct focal plane array images.

The implementation simultaneously computes the gain and offset correction, and bad pixel replacement value, for each pixel. Conditional data path selection passes one result or the other to the output. This implementation produces one corrected pixel for each instruction cycle. With an instruction cycle rate of 60 MHz, this implementation corrects 60 million pixels per second.

6. References

- [1] G.W. Donohoe, D. Buehler, and S. Bruder, "A Design Strategy for Fixed Word Length Data Paths", Proc. NASA Symposium on VLSI Design, Albuquerque, NM, Nov. 7-8, 2000.
- [2] G.W. Donohoe "Focal Plane Array Sensor Non-Uniformity Correction", A White Paper, August 20, 2001.
- [3] G.W. Donohoe, "Focal Plane Array Readout Correction Implementation", A White Paper, August 20, 2001.
- [4] G. W. Donohoe and P.-S. Yeh, "Reconfigurable Data Path Processor", Proc. NASA Earth Sciences Technology Conference, University of Maryland, August 29, 2001.
- [5] G. W. Donohoe and P.-S. Yeh, "A Low Power Reconfigurable Processor", Proc. IEEE Aerospace Conference, Big Sky, MT, March 9-16, 2002.
- [6] B. Parhami, Computer Arithmetic: Algorithms and Hardware Designs, Oxford University Press, New York, 2000.

¹ The authors do not necessarily recommend this algorithm as the best way to perform FPA sensor readout correction. It was chosen to demonstrate certain features of the RDPP. Each application will have different readout correction requirements.

