Journal of Imaging Science and Technology® 62(6): 060404-1–060404-14, 2018.
© Society for Imaging Science and Technology 2018

Digital Circuit Methods to Correct and Filter Noise of Nonlinear CMOS Image Sensors

Maikon Nascimento, Jing Li, and Dileepan Joseph
Department of Electrical and Computer Engineering, University of Alberta, Edmonton, AB T6G 1H9, Canada
E-mail: dil.joseph@ualberta.ca

Abstract. *Nonlinear complementary metal-oxide semiconductor (CMOS) image sensors (CISs), such as logarithmic (log) and linear–logarithmic (linlog) sensors, achieve high/wide dynamic ranges in single exposures at video frame rates. As with linear CISs, fixed pattern noise (FPN) correction and salt-and-pepper noise (SPN) filtering are required to achieve high image quality. This paper presents a method to generate digital integrated circuits, suitable for any monotonic nonlinear CIS, to correct FPN in hard real time. It also presents a method to generate digital integrated circuits, suitable for any monochromatic nonlinear CIS, to filter SPN in hard real time. The methods are validated by implementing and testing generated circuits using field-programmable gate array (FPGA) tools from both Xilinx and Altera. Generated circuits are shown to be efficient, in terms of logic elements, memory bits, and power consumption. Scalability of the methods to full high-definition (FHD) video processing is also demonstrated. In particular, FPN correction and SPN filtering of over 140 megapixels per second are feasible, in hard real time, irrespective of the degree of nonlinearity.* © *2018 Society for Imaging Science and Technology.*
[DOI: 10.2352/J.ImagingSci.Technol.2018.62.6.060404]

1. INTRODUCTION

In a review paper, Kim [1] of Samsung has explained the importance of high dynamic range (HDR) imaging and examined several wide dynamic range (WDR) technologies, based on complementary metal-oxide semiconductor (CMOS) image sensors (CISs), to achieve it. While "dual-exposed or multiframe-capturing WDR sensors... will fill the role of real WDR sensors for a while," he concludes that "the ultimate goal of WDR sensor technology is to physically extend the dynamic range of a sensor, based on pixel technology," mainly to avoid "motion artifacts such as the ghost effect."

As for WDR "pixel technology," Kim prefers the linear–logarithmic (linlog) sensor, a nonlinear CIS with a response that transitions from linear, in dim lighting, to logarithmic (log), in bright lighting. Whereas fixed pattern noise (FPN) does degrade the raw image quality of linear sensors [2], the degradation is worse with log and linlog sensors due to their nonlinearity [3, 4]. Moreover, because of "the variation of a knee point" (Kim's words), the degree of nonlinearity is greater in linlog sensors, compared to log sensors.

An image sensor is a matrix of pixel sensors, so "sensor" has two context-sensitive meanings in this paper. Because perfect uniformity is impossible in CMOS fabrication, FPN is caused by time-independent sensor variations from pixel to pixel [5]. The response of a linear sensor is given by an offset and a gain. Offset variation is usually corrected by analog circuits, implementing correlated double sampling (CDS), integrated on the same chip [2], i.e., the linear CIS. Gain variation is usually corrected by digital circuits, using stored data obtained via calibration, integrated with other functions on a second chip [6], i.e., an image signal processor (ISP).

As for circuit-based nonlinear FPN correction, literature has addressed: analog circuits to correct offset variation only of both log and linlog sensors [7, 8]; mixed-signal circuits to correct both offset and gain variation of linlog sensors [9]; and digital circuits to correct offset, gain, and bias variation of log sensors [10]. Some authors are motivated to avoid calibration or use self-calibration [7–9]. Other authors, like us, are motivated to achieve the highest image quality possible and so, as with linear sensors, adopt calibration [10].

This work contributes, validates, and evaluates a method to generate digital circuits, suitable for ISP integration, to correct all FPN variation, in hard real time, of "arbitrary" sensors. Hard real time means that processing occurs strictly in sync with a clock signal, in this case the same clock that drives CIS readout. An "arbitrary" sensor is one where the response is defined by a monotonic (non)linear function, which need not be specified, having parameters that can vary from pixel to pixel. This includes linear, log, and linlog sensors.

This work also contributes, validates, and evaluates a method to generate digital circuits, suitable for ISP integration, to filter salt-and-pepper noise (SPN) of any monochromatic CIS. It is well known that stuck pixels, such as dead (always dark) or hot (always bright) pixels, require correction by the ISP with linear sensors [6]. In contrast, literature on integrated circuits for log and linlog sensors, including the above citations [7–10], does not address SPN filtering, which like FPN correction is affected by nonlinear responses.

The proposed digital circuit methods exploit recent software algorithms that our group previously published [5]. In Section 2, we summarize the background algorithms and present the novel methods under distinct sub-headings.

As described in Section 3, the proposed methods are validated and evaluated by generating and simulating very-high-speed integrated circuits (VHSICs), using field-programmable gate array (FPGA) tools, from Xilinx and

Received June 29, 2018; accepted for publication Nov. 5, 2018; published online Dec. 5, 2018. Associate Editor: Yeong-Ho Ha.
1062-3701/2018/62(6)/060404/14/$25.00

Altera, and VHSIC hardware description language (VHDL) designs. Section 3 also elaborates on the novelty and significance of this work, both of which have been introduced above.

Section 4 concludes the paper by summarizing its motivation, background, methods, results, and discussion.

2. BACKGROUND AND METHODS

In this section, we summarize relevant background, i.e., software algorithms and underlying concepts, that we have previously published. We also propose novel methods, i.e., digital circuit designs and a generic design flow, to implement FPN correction and SPN filtering, for one or more copies of an "arbitrary" image sensor, in hard real time.

2.1 *Generic Design Flow*

Our digital circuits are coded in VHDL, which is a popular hardware description language (HDL) that allows designs to be implemented in a wide variety of technologies, such as low-cost FPGAs, from Xilinx and Altera, or high-performance CMOS application-specific integrated circuits (ASICs), from TSMC, IBM, etc. However, we explain our circuits and methods using figures, tables, equations, and words.

We target FPGA implementations, due to the preliminary nature of our work, but occasionally we make design choices considering ASIC implementations, anticipating future work. Moreover, we go beyond proposing novel digital circuits for a specific image sensor by proposing methods that generate novel digital circuits for an "arbitrary" image sensor.

These digital circuit methods are implemented using the generic FPGA design flow shown in Figure 1. Unlike the standard design flow, in which Design Specification and Design Entry are both manual, we introduce three aspects that make the Design Entry automatic. The new aspects also add a scripting environment, in this case Matlab, to the standard design flow, which otherwise needs only FPGA design tools, such as ISE from Xilinx or Quartus from Altera.

Normally, digital circuits are realized in FPGAs as follows. First, a high-level description, called the Design Specification, is produced, e.g., using figures, tables, equations, and words. Design Entry means the high-level description is coded in a low-level HDL, which enables Functional Simulation. Using FPGA design tools, a Gate Level Model is obtained via Synthesis. This model, which enables Gate Level Simulation, has more importance with ASIC implementations.

To achieve a binary file, called firmware, suitable for FPGA Download, the design flow has aspects that target a specific FPGA device family, such as the Xilinx Spartan-6 or the Altera Cyclone III. Under Translation & Mapping, the design is flattened into a single "netlist," removing modular aspects, and functional resources, i.e., logic and memory units, of the FPGA family are allocated. Finally, Place & Route, which enables Timing Simulation, selects and configures resources physically available on a chosen FPGA device.

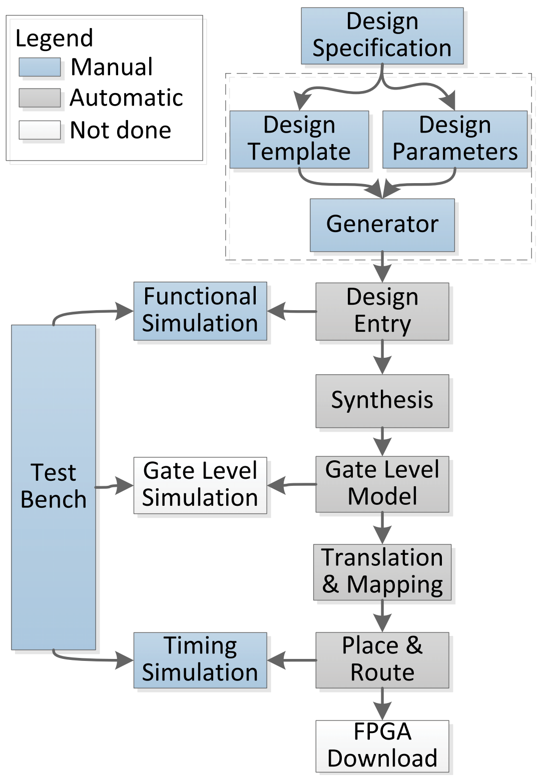

Figure 1. Generic FPGA design flow adopted here. The dashed box shows aspects added to a standard design flow. Functional Simulation suffices to demonstrate validity and estimate complexity. Timing Simulation suffices to evaluate max frequency, of valid operation, and power consumption.

As shown in Fig. 1, instead of manual Design Entry, we generate VHDL code automatically from a Design Template, i.e., VHDL pseudo-code that is image sensor independent, and Design Parameters, i.e., data that is image sensor dependent. Using a Matlab program, these files are processed to generate the VHDL code of a digital circuit for a specific image sensor. Although VHDL has some capability, i.e., generics, to support templating, we required the sophistication of Matlab to realize a recursive digital circuit method.

Because digital circuits are predictable and FPGA testing tools are sophisticated, reliable results are possible without performing FPGA Download. We use Functional Simulation to validate operation, debugging included. Although we may use it also to estimate complexity, i.e., logic and memory needed, we evaluate complexity after Place & Route for 100% accuracy. We do not use Gate Level Simulation but we do use Timing Simulation, including static timing analysis (STA), to evaluate max frequency and power consumption.

2.2 *FPN Correction*

In this paper, as in relevant literature, the word "sensor" may mean either an image sensor or one pixel sensor thereof. Sometimes, the meaning is specified. Sometimes, the meaning is evident. Sometimes, either meaning works.

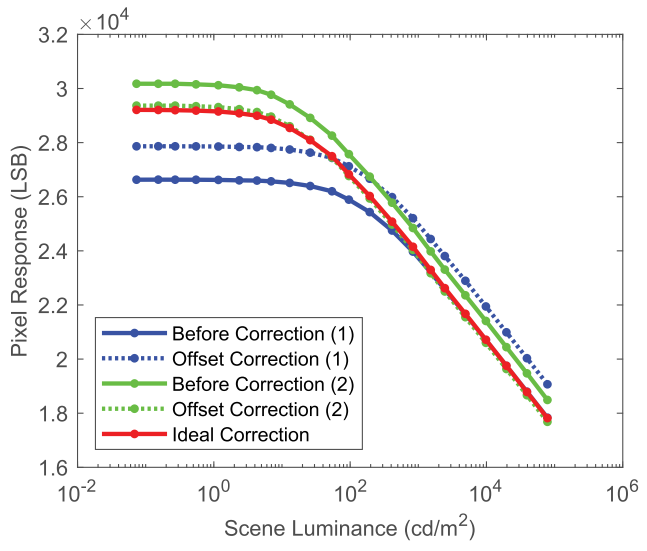

Figure 2. FPN correction need not invert nonlinear responses. Two pixels are shown from a log CIS having 48 × 64 pixels. Offset correction, which is inadequate, simply adds a pixel-dependent number to each response. Ideal correction is well approximated, in this case, using cubic polynomials.

2.2.1 *Background*

To create an effective and efficient algorithm, which we previously published [5], for the FPN correction of an "arbitrary" image sensor, a key concept is that FPN correction need not invert monotonic (non)linear responses of the pixel sensors. Using experimental data from an available log sensor, which we previously documented [11], Figure 2 has been newly prepared to illustrate this concept.

Calling scene luminance x and pixel response y, in Fig. 2, we see first that offset correction does not require computing x. Second, the result of offset correction is still highly nonlinear over the WDR. Although offset and gain correction is not shown, these two observations remain true. Because the "knee point," called the bias [3], varies in this example, even offset and gain correction cannot result in overlapping responses over the WDR, the ideal result of FPN correction.

To improve FPN correction of log sensors, the offset, gain, and bias (OGB) approach uses a specific model [3]:

$$y_j = a_j + b_j \ln(c_j + x_j) + \epsilon_j, \tag{1}$$

where a_j, b_j, and c_j are called the offset, gain, and bias of pixel j, with $1 \le j \le n$, respectively. Temporal and quantization noise, plus residual FPN, are represented by ϵ_j above. After calibration, using uniform luminance of varying intensity, FPN correction is achieved as follows:

$$\hat{x}_j = \exp((y_j - \hat{a}_j)/\hat{b}_j) - \hat{c}_j, \tag{2}$$

where $\hat{a}_j$, $\hat{b}_j$, and $\hat{c}_j$ are parameters estimated by the one-time calibration, and $\hat{x}_j$ is the OGB correction.

The above approach is unsuitable for FPN correction of an "arbitrary" sensor. As it requires inversion of the nonlinear response, it may not even be the best approach for FPN correction of a log sensor. Third, modeling a linlog sensor in a similar way to Eq. (1) proves complicated [4].

An alternative, i.e., our inverse polynomial regression (IPR) approach [5], uses the following generic model:

$$y_j = f_j(x_j) + \epsilon_j, \tag{3}$$

where f_j is a monotonic (non)linear function with parameters that vary with pixel j. We showed that FPN correction is possible, using low-degree polynomials, as follows:

$$\hat{y}_j = y_j + \hat{b}_{j0} + y_j(\hat{b}_{j1} + y_j(\hat{b}_{j2} \cdots + y_j(\hat{b}_{jq}))), \tag{4}$$

where $\hat{b}_{jk}$ are the coefficients of degree q polynomials, with $0 \le k \le q$, and $\hat{y}_j$ is the IPR(q) correction.

Irrespective of q, IPR(q) correction requires arithmetic only. Moreover, IPR(0) correction is simply offset correction, IPR(1) correction equates to offset and gain correction, and IPR(3) correction is ideal for the log sensor example of Fig. 2, over all 3,072 pixels [5]. The ideal response, which remains highly nonlinear over the WDR, is shown in Fig. 2.

We also developed a fixed-point version of FPN correction [5]. Denoting binary-point positions and word lengths as s_k and t_k, respectively, double-precision coefficients $\hat{b}_{jk}$ convert to signed-integer coefficients B_{jk} as follows:

$$B_{jk} = round(2^{-s_k}\hat{b}_{jk}), \tag{5}$$
$$|B_{jk}| < 2^{t_k - 1}. \tag{6}$$

We showed how to calculate optimal s_k and t_k values, given a total word length t, in bits per pixel (bpp):

$$t = t_0 + t_1 + t_2 \cdots + t_q. \tag{7}$$

The binary-point shifting of Eq. (5), to scale coefficients before rounding, means the FPN correction of Eq. (4) must be amended, to undo the binary-point shifting, as follows:

$$\begin{aligned} Y_j = y_j + 2^{s_0}\,(B_{j0} + 2^{s_1 - s_0} y_j\,(B_{j1} + 2^{s_2 - s_1} y_j \\ \times\ (B_{j2} \cdots + 2^{s_q - s_{q-1}} y_j(B_{jq})))), \end{aligned} \tag{8}$$

where Y_j denotes the result of fixed-point IPR(q) correction. When t is sufficiently large, the results of floating-point and fixed-point correction are indistinguishable [5].

To complete the explanation, two additional details are needed. First, instead of y_j in the right hand side (RHS) of Eq. (8), except the leftmost y_j, we use the following:

$$y'_j = y_j - y_0, \tag{9}$$

where y_0 is an unsigned-integer constant. Because y_j, the response of pixel j after an analog-to-digital converter (ADC), is an unsigned integer, we use Eq. (9) to produce signed integers where the worst-case magnitude is significantly lower. This change allows us to significantly lower the total word length t required by the fixed-point correction [5].

The final detail concerns binary-point, or bit, shifts in the RHS of Eq. (8). Because s_0 is expected to be non-negative in an optimal configuration, it entails a left shift. The left

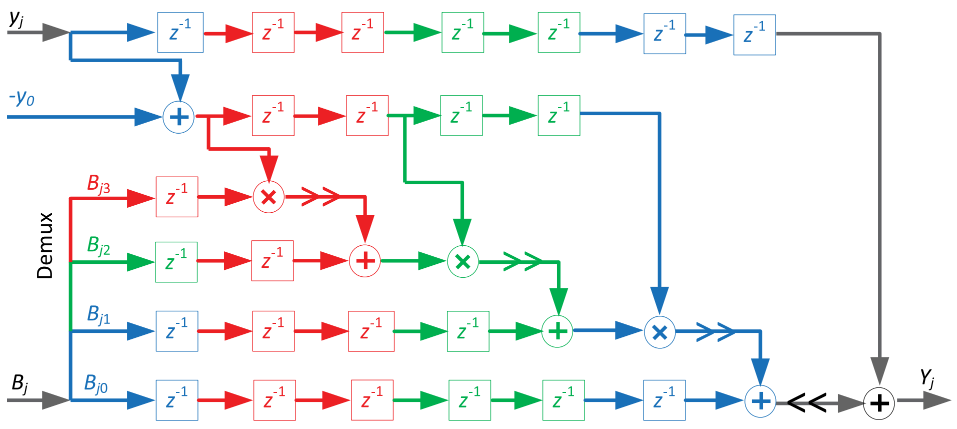

Figure 3. FPN correction using a recursive pipeline circuit. This schematic shows: offset correction in black; offset and gain, i.e., linear, correction in blue and black; quadratic correction in green, blue, and black; and cubic correction in red, green, blue, and black. Bus operations $\gg$ and $\ll$ represent bit shifts.

shift of an integer stays an integer. On the other hand, $s_k - s_{k-1}$ for $1 \le k \le q$ may be negative, entailing potential right shifts. The right shift of an integer may have a fractional part. To avoid fractional parts and reduce word lengths of intermediate values, a round operation is performed after each shift in Eq. (8), except the leftmost one. This turns the fixed-point correction into a more efficient integer correction [5].

2.2.2 *Method*

Because FPN calibration is a one-time process with no real-time constraints, there is no need to design a circuit to implement it. The software algorithm we detailed previously [5], implemented in Matlab, suffices for this purpose. However, we require a digital circuit to implement FPN correction efficiently in hard real time. Moreover, because we want a solution not for one nonlinear image sensor but for a wide variety of them, we use our generic design flow, shown in Fig. 1, to realize a digital circuit method.

Parameters of the FPN correction include: the polynomial degree, q; the binary-point positions, s_k, and word lengths, t_k, where $0 \le k \le q$; the number of pixels, n; the polynomial coefficients, B_{jk}, where $0 \le j \le n-1$ assuming 0-based indexing, what VHDL uses, instead of 1-based indexing, i.e., $1 \le j \le n$, what Matlab uses; and the word length, t_{ADC}, of pixel responses, y_j. Because FPN correction is agnostic to the division of n pixels into n_1 rows and n_2 columns, where n equals $n_1 n_2$, the latter are not parameters.

For each pixel j, we pack the coefficients B_{jk}, where word lengths t_k may vary, into t-bit words denoted by B_j, where t in Eq. (7) is constant. As this is done once, it is done in Matlab after calibration. The resulting tn-bit data is not a part of the proposed FPN correction circuit but is external data, e.g., stored in flash memory, that is repeatedly read into the circuit synchronously with y_j, the response of pixel j.

Given an image sensor design, B_j may be the only set of parameters that needs to vary with each instance, or fabricated copy, of the design. The remaining parameters may be fixed, which therefore fixes the FPN correction circuit. Whereas FPGA implementations allow circuit reconfiguration, ASIC implementations do not. In his FPN correction work, Hoefflinger [10] also externalized coefficients.

Figure 3 presents a schematic, or rather multiple schematics, of the proposed FPN correction circuit. An important feature of the circuit is its recursive nature. The IPR(3) circuit has the IPR(2) circuit as a sub-circuit. In turn, the IPR(2) circuit has the IPR(1) circuit as a sub-circuit. For $q > 3$, the IPR(q) circuit follows from the pattern. Even though the IPR(1) circuit has the IPR(0) circuit as a sub-circuit, they are both special cases as neither follows the higher-degree pattern.

Digital circuit elements may be classified as sequential logic, operating synchronously with a clock signal, or combinational logic, operating asynchronously. Unlike software algorithm steps running on central processing units (CPUs), where parallel processing is absent or limited to a few CPU cores, digital circuit elements always operate in parallel, including state changes of memory bits in sequential logic. However, the state changes happen either on rising or falling edges, depending on the design, of a clock signal.

In Fig. 3, addition (+), subtraction (−), multiplication (×), and delay (z^{-1}) elements are synchronous, each with a latency of one clock cycle. Other elements are asynchronous. Digital circuits require synchronous elements, for reliable operation at very high speed, because of race conditions that arise in a purely asynchronous design. A sequence of synchronous elements with intervening asynchronous elements, as shown in the figure, results in a pipeline circuit, which exploits parallel processing in the fashion of an assembly line.

Elements are arranged, in Fig. 3, to illustrate the pipeline processing. Each column of synchronous elements performs one arithmetic operation while equally delaying other signals required for a subsequent arithmetic operation, final one aside. Although a pixel is corrected each clock cycle, IPR(q) correction has a latency of $2(q+1)$ clock cycles, for $q > 0$.

Because offset correction does not require Eq. (9), i.e., a subtraction, IPR(0) correction takes exactly one clock cycle.

For high-speed operation, i.e., to increase the max frequency of the clock signal, elements are kept as simple as possible. The combinational logic, in Fig. 3, consists primarily of bus operations, where a bus is a group of wires, each carrying one bit of a digital signal. The Demux element, for "demultiplexer," partitions a t-bit bus, carrying B_j, into multiple t_k-bit buses, carrying B_{jk}, where $0 \le k \le q$. The left-shift ($\ll$) element, before the final addition, simply pads the incoming bus with s_0 zero-valued least significant bits (LSBs).

The right-shift ($\gg$) elements, in Fig. 3, require elaboration. First, the value Δs_k of each shift, from left to right, is $s_k - s_{k-1}$, where k goes from q to 1, respectively. Except for positive Δs_k, in which cases a left shift is implemented as described above, the $|\Delta s_k|$ LSBs of each incoming bus are ideally discarded. However, this implements a right shift with rounding down, which may be expressed as follows:

$$Y \gg |\Delta s| = \lfloor 2^{\Delta s} Y \rfloor, \tag{10}$$

where Y is the digital signal on the incoming bus.

Although convenient for a circuit implementation, using Eq. (10) leads to bit errors because the background algorithm uses right shifts with rounding off. Because the difference between rounding off and rounding down is either 0 or 1, a right shift with rounding off may be implemented as follows:

$$Y \gg |\Delta s| = \lfloor 2^{\Delta s} Y \rfloor + c_{\text{out}}, \tag{11}$$

where c_{out}, for carry out, is a one-bit correction. This exploits a bus operation but ensures a bit-true implementation.

For a u-bit signed-integer signal Y, the carry out, c_{out}, of $Y \gg v$ may be calculated as follows, assuming a standard two's complement representation for negative values:

$$c_{\text{out}} = Y_{v-1} \wedge (\bar{Y}_{u-1} \vee d_{\text{rem}}), \tag{12}$$
$$d_{\text{rem}} = Y_{v-2} \vee Y_{v-3} \cdots \vee Y_0, \tag{13}$$

where Y_{u-1} is the most significant bit (MSB) of Y (its sign bit), Y_{v-1} is the MSB of the v discarded bits, and Y_{v-2} to Y_0 are the remaining discarded bits. Symbols $\wedge$, $\vee$, and $\bar{}$ are logical AND, OR, and NOT operators, respectively.

Addition of carry out in Eq. (11), to correct each right shift, may actually be integrated into a following adder. In Fig. 3, every right-shift element is followed by an adder element. Standard two-input adders always have a third one-bit input, called the carry in, that is added to the sum of the two inputs. Therefore, the carry out of the right shift may be directed to the carry in of the adder. This efficiency may be readily exploited with an ASIC implementation. With an FPGA implementation, even though adder elements support carry in, the exact mapping of operations to circuitry is, however, fully automatic.

To complete the digital circuit, all bus sizes in Fig. 3 must be specified. Input signal y_j and constant signal $-y_0$, the two's complement of y_0, are both t_{ADC} bits wide. Input signal B_j is t bits wide. After demultiplexing, signals B_{jk} are t_k bits wide, where $0 \le k \le q$. Delays do not change bus sizes. The output bus size of each adder is $\max(u, v)$, for inputs with bus sizes u and v. While, in theory, such adders could overflow by one bit, it is unlikely because each addition represents a perturbation to correct, in stages, deviation from an ideal response that is never close to the saturation limits. In the unlikely event of overflow, the outlier would be removed by the SPN filter of the next section. Finally, the output bus size of each multiplier is $u + v$, for inputs with bus sizes u and v.

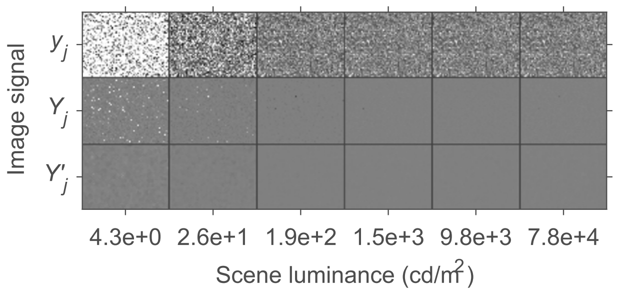

Figure 4. FPN correction and SPN filtering are complementary. To deal with noise in raw images (top row), from a log sensor over a WDR (left to right), both correction (middle row) and filtering (bottom row) are needed.

2.3 *SPN Filtering*

Whereas we have previously disclosed our SPN filtering approach [5], which was implemented as a software algorithm, it was only briefly justified. Before introducing a novel digital circuit method, we briefly review the background approach, while offering additional justification for it.

2.3.1 *Background*

Stuck pixels are one source of SPN, also called impulse noise. Because stuck pixels may be identified during calibration, instead of filtering they may be corrected using a static procedure, similar to FPN correction.

However, with nonlinear pixels, such as the log pixels shown in Fig. 2, pixels may appear stuck at some luminances, behaving as outliers after FPN correction, but may contribute useful information at other luminances. In addition, there may be a few nonlinear pixels that are truly stuck. An SPN filtering approach can deal with both cases dynamically.

Using experimental data from the previously documented log sensor [11], Figure 4 has been newly prepared to illustrate how FPN correction and SPN filtering are complementary. Images are shown of six uniform scenes at three stages of signal processing. Whereas FPN is mostly corrected by the FPN correction, it yields SPN especially at lower light levels. The SPN, which includes bright and dark pixels that vary with luminance, is filtered by the SPN filtering.

Median filtering is a well known approach to dynamically remove SPN. A median filter replaces each pixel's response with the median response from a local window. For simplicity, we consider only monochromatic image sensors, avoiding the complexities of color filter arrays for now.

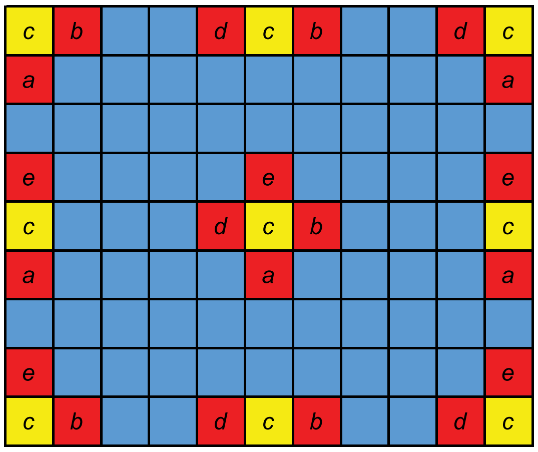

Figure 5. SPN filtering employs windows that vary with pixel. In each window, pixels are colored red except for the center pixel, which is colored yellow. The center pixel is replaced with the median value of its window.

Figure 5 illustrates the different windows used by our SPN filter. Image dimensions are preserved because a median is computed at every pixel. Small symmetric windows are chosen to minimize distortion. For interior pixels, the pixel and four neighbors are used. For boundary pixels, at the borders and corners, the pixel and two neighbors are used. At the time of our software algorithm [5], we were thinking ahead to a circuit method. With odd-size windows, only sorting is needed to compute medians; averaging is not needed.

2.3.2 *Method*

Figure 6 presents a schematic of our SPN filter. We use our generic design flow, of Fig. 1, to realize a digital circuit method, suitable for a variety of monochromatic image sensors, as opposed to a digital circuit, suitable for just one set of parameters. The parameters in question are: n_1 and n_2, which are the number of rows and columns, respectively, in the n-pixel image, where n equals $n_1 n_2$; and t_{FPN}, which is the word size of the input signal, Y_j. Additional parameters, namely t_{row} and t_{col}, are explained below.

Because FPN correction precedes SPN filtering, we exploit pipeline processing in the latter also. Whereas it does not matter for FPN correction whether pixels are processed in row-major or column-major order, we assume they are processed in row-major order, for clarity, in explaining the SPN filtering. The first row of n_2 pixels is processed, one by one from left to right, followed by the second row, and so on.

The first stage of the SPN filter is a first-in first-out (FIFO) buffer. Its five outputs, denoted by a to e in Fig. 6, are delayed versions of the input signal, Y_j. The delays are 0, $n_2 - 1, n_2, n_2 + 1$, and $2n_2$ clock cycles, respectively. They are chosen so that, when c corresponds to an interior pixel, a to e correspond to its five-pixel cross-shaped window, as shown in Fig. 5. Bus sizes of the input and output signals equal t_{FPN}.

Bypassing the second stage momentarily, the third stage of the SPN filter is a simplified pipeline sorter of five digital

Table I. Combinational logic performed by the router. The encoder outputs a four-bit code, which controls the multiplexers, based on the center pixel address. See Figs. 5 and 6 also.

Pixel address (j)	Encoder	a'	b'	c'	d'	e'
Corner, top left	1010	a	b	c	a	b
Border, top	1000	b	b	c	d	d
Corner, top right	1001	a	a	c	d	d
Border, left	0010	a	a	c	e	e
Interior	0000	a	b	c	d	e
Border, right	0001	a	a	c	e	e
Corner, bottom left	0110	b	b	c	e	e
Border, bottom	0100	b	b	c	d	d
Corner, bottom right	0101	d	e	c	d	e

signals, e.g., FIFO outputs a to e. A five-input pipeline sorter may be realized using multiple two-input pipeline sorters. Although all five signals may be fully sorted with a latency of five clock cycles, the circuit may be simplified because only the third output, i.e., the median signal, is required. Each two-input sorter outputs the same two signals in min–max order with a latency of one clock cycle. Only one of the two outputs is required in some cases. All bus sizes equal t_{FPN}.

On their own, a combination of the above FIFO and sorter stages would compute invalid outputs at boundary pixels, where a five-pixel cross-shaped window cannot be formed. One solution is to add a one-bit output signal, of the SPN filter, to indicate validity of the main output signal, Y_j'. This solution would require some combinational logic to distinguish interior from boundary pixels. At a cost of some more combinational logic, valid outputs may be computed at the boundary pixels and the additional one-bit signal may be avoided.

The second stage of the SPN filter, between the FIFO and the sorter in Fig. 6, is a router. The router enables median filtering of three-pixel windows at the boundary, as shown in Fig. 5, using the same FIFO and sorter. Mathematically, the median of three numbers equals the median of five numbers where two of the original three numbers are copied.

Table I elaborates on the router. The position of the center pixel, denoted by c in Figs. 5 and 6, is given by its address j. An encoder converts the address, which is $t_{\mathrm{row}} + t_{\mathrm{col}}$ bits wide, into a four-bit code. This code controls multiplexers that, at boundary pixels, replace two of the five inputs a to e with two selected copies. The five outputs a' to e' of the router, where c' always equals c, become inputs of the sorter.

For the above reasons, SPN filtering requires a pixel-address input signal, unlike FPN correction. With pipeline processing, careful attention must be given to synchronization when there are multiple input signals. Because a equals Y_j in Fig. 6, the address of pixel c does not equal j. One solution is to use an address signal j delayed by n_2 clock cycles, the delay between c and a. Because delay elements

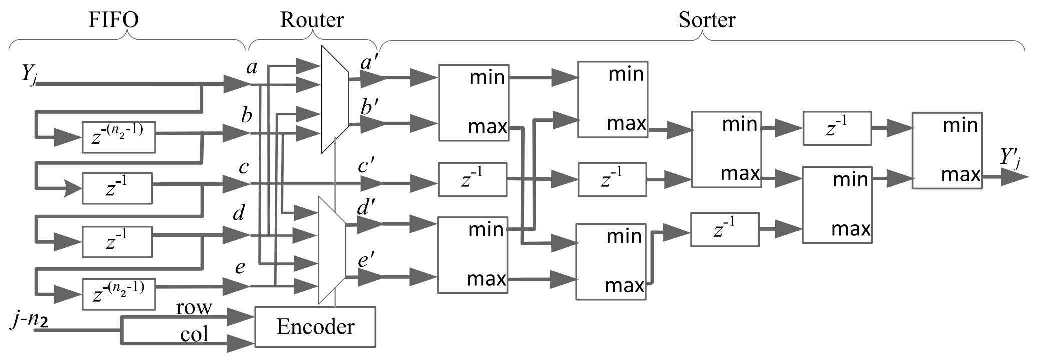

Figure 6. SPN filtering using a three-stage pipeline circuit. The FIFO buffers two rows of pixel values. The sorter computes the median of five pixel values. The "no-delay" router is needed to compute medians for three-pixel windows at the image corners and borders. Table I elaborates on the router logic.

map to memory resources, this would increase memory use by about 50%.

The image sensor, whose output signal, y_j, becomes the input signal in Fig. 3, itself requires an address signal, j. Addresses would be supplied in sequence by a controller circuit, typically using counters, wholly separate from the FPN correction and SPN filtering. We assume that, with minor changes, e.g., extra counters, the same controller circuit could also provide a "delayed" address signal, denoted by $j - n_2$ in Fig. 6, suitable for SPN filtering. The exact "delay," implemented using counters not delay elements, must also account for the latency of FPN correction, which is $2(q+1)$ clock cycles, for $q > 0$, or 1 clock cycle, for $q = 0$.

Assuming the address signal may be demultiplexed into row and column parts that are t_{row} and t_{col} bits wide, respectively, the logic of the encoder, in Fig. 6 and Table I, is simple. The first two bits of the code are computed from the row address, and the last two bits from the column address. The first bit is one at the first row only, the second bit is one at the last row only, the third bit is one at the first column only, and the fourth bit is one at the last column only.

Because memory is relatively scarce in a low-cost FPGA, our SPN filtering circuit avoids buffering a whole image frame, i.e., all n pixels, before computing medians. Only two rows, i.e., $2n_2$ pixels, are buffered, the fewest values needed to form cross-shaped windows for interior pixels. Not only does this reduce memory requirements from $O(n)$ to $O(\sqrt{n})$ bits, because n_2 is usually proportional to $\sqrt{n}$, but also it reduces latency by the same order of magnitude.

3. RESULTS AND DISCUSSION

Section 2 presented methods to generate digital circuits to correct FPN and filter SPN in hard real time. These methods are validated and evaluated by generating and simulating specific circuits using FPGA tools from Xilinx and Altera. Results are compared to the state of the art.

3.1 *Test Benches*

Using the design flow shown in Fig. 1, digital circuits are generated for specific FPGA devices, namely the Xilinx XC6SLX4 and the Altera EP3C5. Both Xilinx and Altera, now part of Intel, have multiple device families. The lowest-cost

Table II. Video formats used to evaluate proposed methods. Frames are composed of n_1 scan lines and n_2 pixels per line. The clock frequency is the number of pixels times the frame rate.

Format	Pixels	$(n_1 \times n_2)$	Rate	Clock
TTVGA	3,072	(48×64)	30 fps	92.16 kHz
HQVGA	38,400	(160×240)	30 fps	1.152 MHz
VGA	307,200	(480×640)	30 fps	9.216 MHz
FHD	2,073,600	(1080×1920)	30 fps	62.21 MHz
4KUHD	8,294,400	(2160×3840)	30 fps	248.8 MHz

families still in production, at the time of this work, are the Xilinx Spartan-6 [12] and the Altera Cyclone III [13]. The chosen devices, i.e., the XC6SLX4 and the EP3C5, are the simplest ones in these lowest-cost families.

We use ISE 14.7, from Xilinx, and Quartus 13.0, from Altera, for synthesis, translation-and-mapping, place-and-route, etc. Validation involves manual signal analysis and automatic comparison against background software algorithms. Evaluation assesses circuit complexity, max frequency, and power consumption versus parameters of interest.

For FPN correction, the main parameter is the polynomial degree. As 3 suffices for a log sensor [5], we considered degrees from 0 to 5. For SPN filtering, the main parameter is the number of pixels, or rather columns. We considered five video formats, which specify number of pixels, division into rows and columns, and frame rate. Power consumption depends on clock frequency, which equals number of pixels times frame rate, in frames per second (fps).

Table II lists three popular video formats, two of which are high definition (HD) formats, where pixel numbers are roughly equidistant on a log scale. They are the video graphics array (VGA), full HD (FHD), and 4K ultra HD (4KUHD) formats. While tenth tenth VGA (TTVGA) is a non-standard format, it matches our log sensor prototype [11]. The half quarter VGA (HQVGA) format, a rare standard format, fills a gap between the TTVGA and VGA formats on a log scale.

Recalling Fig. 1, Functional Simulation after Design Entry suffices for validation but Place & Route is needed to

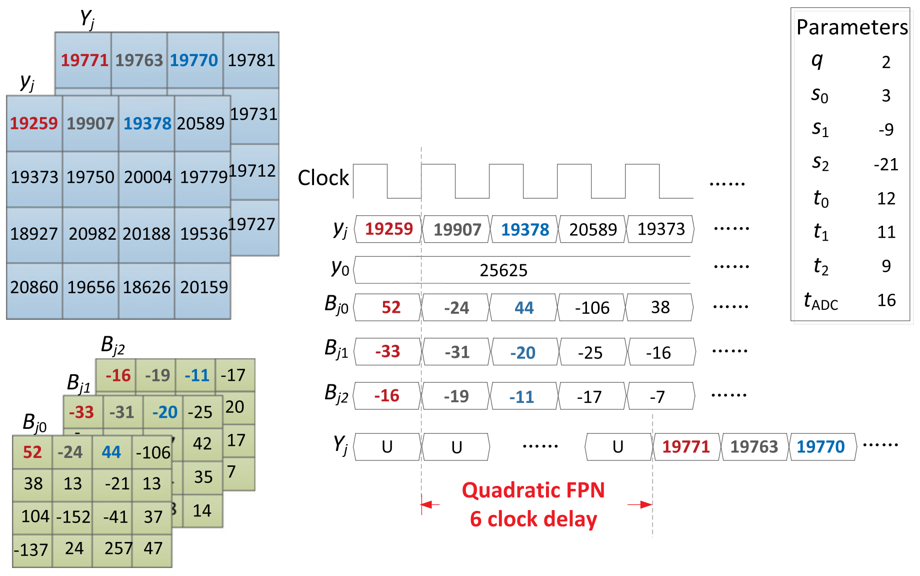

Figure 7. Initial validation of a generated FPN correction circuit. Input, output, and intermediate signals are shown for a small-format test case. Fig. 3 elaborates on the signals. Larger-format test cases were validated by automatic comparison of circuit and software outputs, given the same inputs.

evaluate complexity accurately. Timing Simulation is used to evaluate max frequency and power consumption. The max frequency is the highest clock frequency at which the circuit operates correctly. It is determined via STA, which identifies critical circuit paths. A video format is supported if its clock frequency, in Table II, is below the max frequency. Power consumption is evaluated only for supported video formats.

3.2 *FPN Correction*

Given that the simplest FPGA devices were chosen, in the lowest-cost device families from two leading vendors, the following results show that the generated FPN correction circuit is not only effective but also efficient.

3.2.1 *Validation*

Illustrated in Figure 7, the initial validation of the generated FPN correction circuit was done manually for a 4×4 pixel subset of the TTVGA format, using experimental data from a log sensor [11]. The figure shows, at left, the input image, y_j, the output image, Y_j, and the FPN correction coefficients, B_{jk}. Circuit parameters are given, at right.

Validation was done for the chosen Xilinx and Altera devices. FPGA tools are used to analyze input, intermediate, and output signals, depicted in Fig. 7, in simulated hard real time, i.e., against a clock signal with fixed period. For a small-format test case, the expected intermediate and output signals, including latencies, may be calculated. For example, the first output, Y_1, may be manually calculated as follows:

$$y'_1 = 19259 - 25625 = -6366 \tag{14}$$

$$\begin{aligned} Y_1 &= 19259 + 2^3\,(52 + [\,2^{-9-3}(-6366)\,(-33 \\ &\quad + [2^{-21+9}(-6366)(-16)]\,]\,]) \\ &= 19771, \end{aligned} \tag{15}$$

where square brackets indicate rounding.

As shown in Fig. 7, the correct output appears with a latency of 6, i.e., $2(q+1)$, clock cycles, as expected. Unknown signal values, based on initial conditions of memory elements, are indicated with a “U,” as with the FPGA tools.

Manual validation on small-format test cases was key to debugging all issues. For large-format test cases, the same input data was processed by the generated circuit and a MATLAB implementation of the background algorithm. The two output data sets were compared bit-for-bit in MATLAB to ensure a bit-true design, i.e., zero bit error.

3.2.2 *Complexity*

Given functional correctness, we then analyzed the complexity of generated circuits, illustrated in Figure 8, versus polynomial degree, q. The word lengths of the pixel response, t_{ADC}, and of the packed correction coefficients, t, were kept constant, at 16 and 32 bits, respectively. Parameters s_k and t_k were automatically determined [5].

In Fig. 8, actual data is shown using symbols, for each FPGA device, and trends are shown using best-fit lines. Complexity is measured in logic elements (LEs) and memory bits, on the left and right y-axes, respectively. The LEs required grow roughly linearly with degree (R^2, the coefficient of determination, equals 80% and 86% with Xilinx and Altera, respectively). Outlier aside, i.e., degree zero with Altera, the bits required also grow linearly with degree (R^2 equals 92% and 93% with Xilinx and Altera, respectively).

More significant than linearity perhaps, the generated FPN correction circuits are of very low complexity relative to the available resources, leaving plenty of LEs and bits for other ISP operations on the same FPGA. The available resources in the chosen devices, i.e., the simplest ones in the lowest-cost families, are 8,648 LEs and 297,984 bits with Xilinx, and 10,318 LEs and 423,936 bits with Altera.

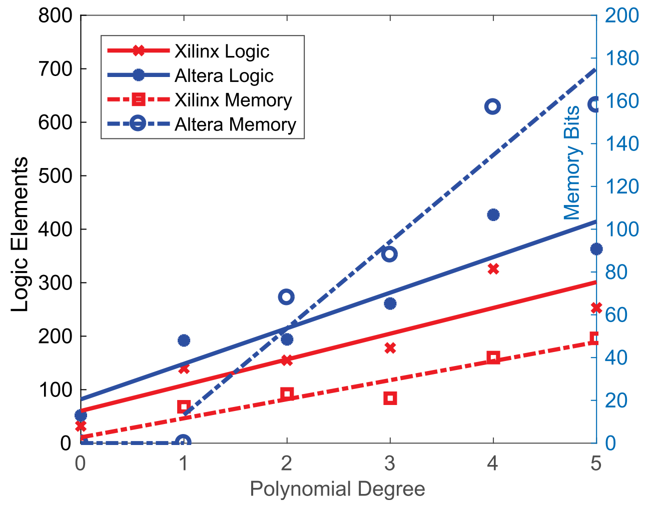

Figure 8. Complexity of FPN correction versus polynomial degree. Required LEs and bits depend linearly on degree, Altera memory for offset correction aside. Even so, these requirements use a tiny fraction of available resources.

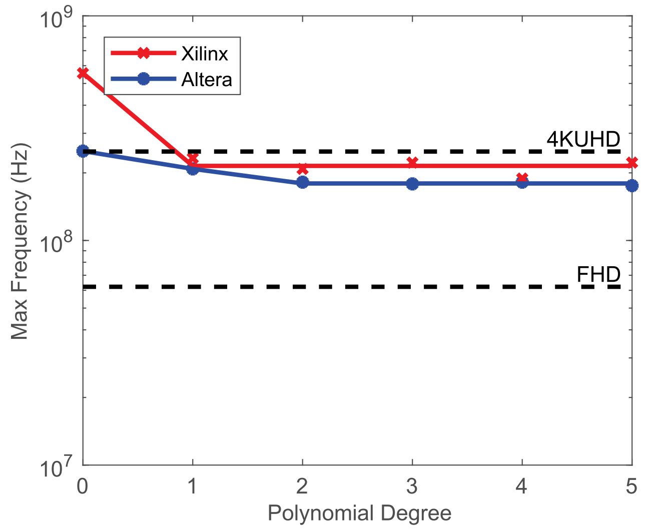

Figure 9. Max frequency of FPN correction versus polynomial degree. Except at the lowest degrees, max frequencies are essentially independent of polynomial degrees. FHD and simpler video formats are readily supported.

3.2.3 *Frequency*

Next, we determined the maximum clock frequency at which functional correctness is maintained. These results are shown in Figure 9 versus polynomial degree, as before. Other parameters were unchanged.

Notwithstanding the lowest degrees, at which the generated circuit can run faster, the max frequency is approximately constant in both FPGAs. Reflecting on Fig. 3, each increase in degree introduces a synchronous stage in the recursive pipeline circuit. However, each stage is composed of parallel circuit paths where the worst-case circuit path is of constant complexity. This explains the trends in Fig. 9.

What is also significant is that the max frequency is high enough, in both FPGAs, to support FPN correction of FHD video in hard real time. Horizontal dashed lines, shown in Fig. 9, indicate the frequencies, listed in Table II, required to support the FHD and 4KUHD formats.

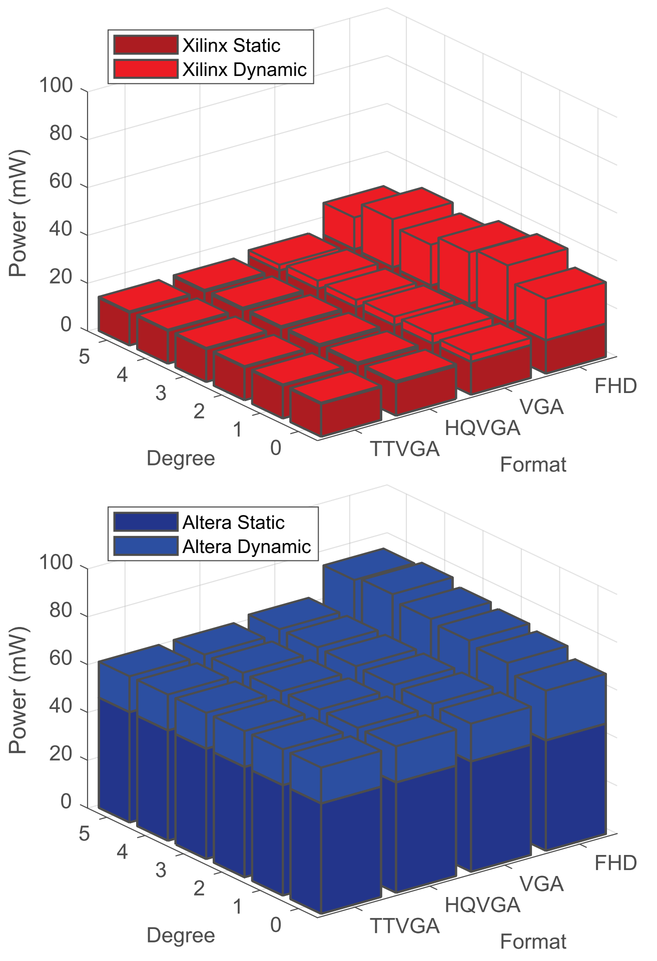

Figure 10. Power consumption of FPN correction versus parameters. Except for the FHD video format, where it increases a little, dynamic power is nearly constant. Compared to static power, dynamic power is generally low.

3.2.4 *Power*

Our final results, shown in Figure 10, concern power consumption. Because this depends on clock frequency, we use the corresponding frequencies, listed in Table II, for the supported video formats. We also vary the polynomial degree, as before. Other parameters were unchanged.

In Fig. 10, total power is decomposed, using a stacked bar graph, into static and dynamic components, and this is done for each device. The FPGA tools enable this decomposition, where the static consumption represents the background power consumed by the device, an approximate constant that is independent of the circuit and its operation.

Not only is the total power on the order of 50 mW, in Fig. 10, but also the dynamic power is, in general, low relative to the static power. Except for the FHD video format, where the power increases a little and depends a little on degree, the dynamic power is otherwise nearly constant.

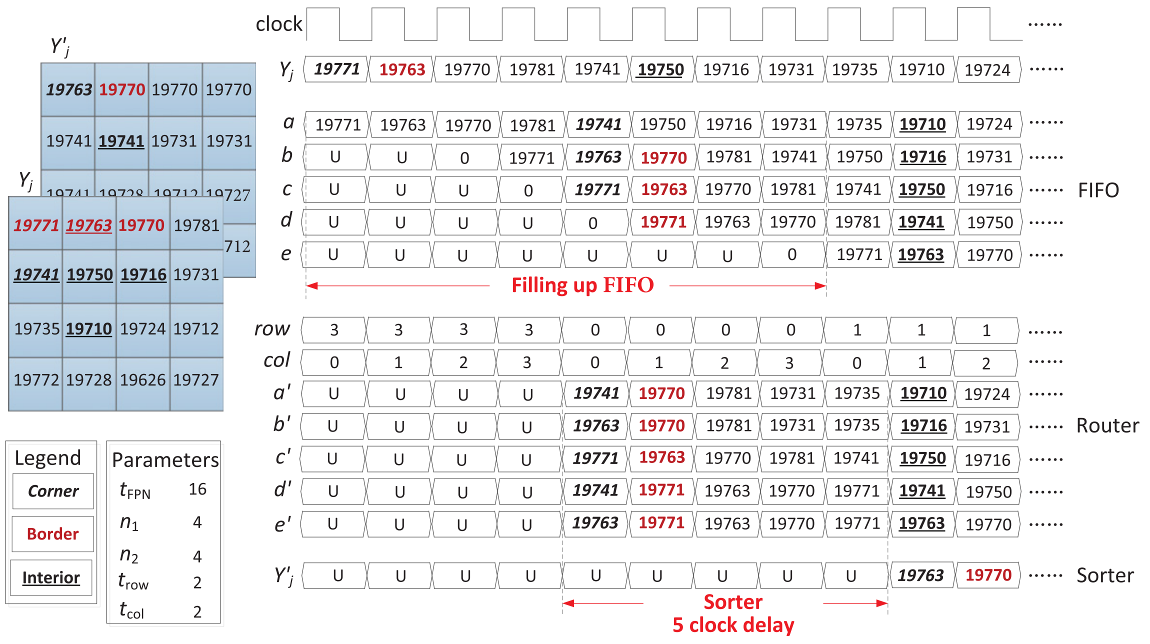

Figure 11. Initial validation of a generated SPN filtering circuit. Input, output, and intermediate signals are shown for a small-format test case. Fig. 6 elaborates on the signals. Larger-format test cases were validated by automatic comparison of circuit and software outputs, given the same inputs.

3.3 *SPN Filtering*

Evaluation of the generated SPN filtering circuit proceeds similarly to the preceding evaluation of the generated FPN correction circuit. Therefore, we will be briefer.

3.3.1 *Validation*

Illustrated in Figure 11, initial validation was done manually using small-format test cases. Input (Y_j) and output (Y'_j) images are shown at left, as are circuit parameters. Example corner, border, and interior pixels are indicated (see legend). Waveforms are shown, at right, and they are grouped as per Fig. 6. It is straightforward to show that all waveforms in Fig. 11 are correct, including the latencies.

Manual validation on small-format test cases was followed by automatic validation on large-format test cases. In the latter situation, output from the generated circuit was compared bit-for-bit to output from a MATLAB program, implemented using high-level matrix-vector operations to perform median filtering, as per Fig. 5. There was zero bit error.

3.3.2 *Complexity*

Given functional correctness, we then analyzed the complexity of the generated circuit, illustrated in Figure 12, versus number of pixels, n. Each number, n, and its breakdown into rows, n_1, and columns, n_2, is taken from Table II. The word size of the input signal, t_{FPN}, was kept constant at 16 bits. Address bus sizes, t_{row} and t_{col}, were set to the minimum values, i.e., $\lceil \log_2 n_1 \rceil$ and $\lceil \log_2 n_2 \rceil$, respectively.

The LEs required are roughly independent of image size, as shown in Fig. 12. However, there is a linear relationship, on a log–log scale, between the bits required and the number

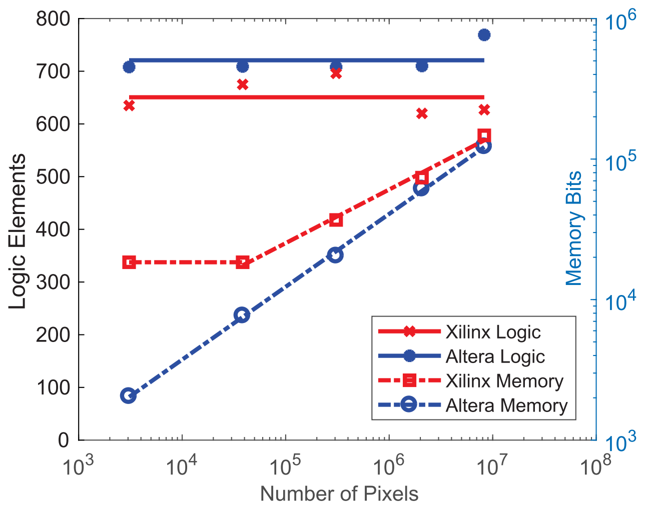

Figure 12. Complexity of SPN filtering versus number of pixels. Required LEs are approximately constant and use a fraction of available resources. Required bits grow with the number of pixels but remain well below capacities.

of pixels (R^2 equals 99% and 100% with Xilinx and Altera, respectively), excluding one outlier. Memory required with Altera exactly equals the minimum bits, i.e., $2n_2t_{\mathrm{FPN}}$, needed to implement the FIFO stage shown in Fig. 6.

What is more significant is that, relative to the available resources in the Xilinx and Altera devices, the LEs required are very low, e.g., 7.17% and 6.88%, respectively, for the FHD video format. The bits required are also low, e.g., 24.7% and 14.5%, respectively, for the same video format.

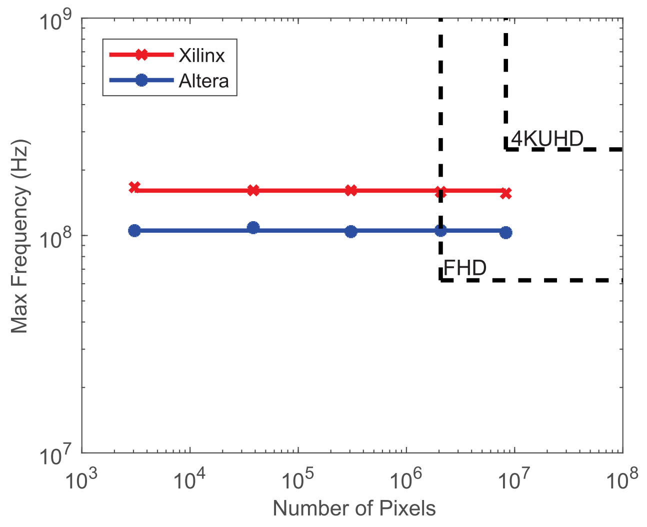

Figure 13. Max frequency of SPN filtering versus number of pixels. The max frequency is essentially independent of the number of pixels. FHD and simpler video formats, listed in Table II, are readily supported.

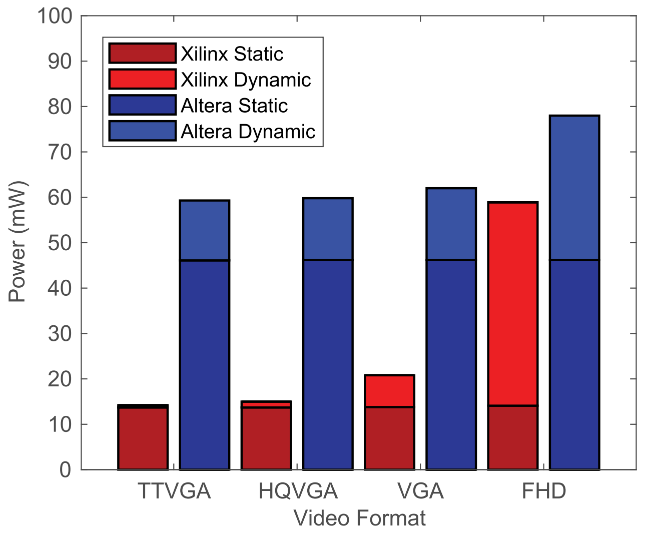

Figure 14. Power consumption of SPN filtering versus video format. Static power is a constant and significant part of total power. Except for a jump at the FHD video format, dynamic power is approximately constant.

3.3.3 *Frequency*

Next, we determined the maximum clock frequency at which functional correctness is maintained. These results are shown in Figure 13 versus number of pixels, as before. Other parameters are the same as with Fig. 12.

In Fig. 13, the max frequency is nearly constant in both FPGAs. The fact that the LEs required are roughly constant, in Fig. 12, largely explains this result. Max frequency is expected to depend on circuit paths, i.e., logic not memory. Changes in video format, such as the number of pixels, primarily affect the memory used by the FIFO stage in Fig. 6.

What is also significant is that the max frequency is high enough, in both FPGAs, to support SPN filtering of FHD video in hard real time. Dashed lines, in Fig. 13, indicate the numbers of pixels and clock frequencies, listed in Table II, required to support the FHD and 4KUHD formats.

3.3.4 *Power*

Our final results, shown in Figure 14, concern power consumption for the supported video formats. We use the numbers of pixels and clock frequencies listed in Table II. Other parameters are the same as with Fig. 12.

Except for the FHD case, as shown in Fig. 14, dynamic power is essentially independent of video format, with both FPGA devices, and is lower than static power. For the FHD video format, averaging over both FPGA devices, dynamic power increases to a level comparable to static power, but the total power remains on the order of 50 mW.

3.4 ***Significance***

After summarizing selected results, we compare our digital circuit for FPN correction to an analog competitor, a mixed-signal competitor, which uses both analog and digital circuitry, and a digital competitor. We also compare our digital circuit for SPN filtering to a digital competitor.

3.4.1 *Specifications*

Table III summarizes the specifications of the designed FPN correction and SPN filtering circuits for a specific scenario, namely cubic polynomials and the FHD video format. Other parameters are as described in Sections 3.2 and 3.3. These circuits were also combined into one ISP circuit, i.e., FPN correction followed by SPN filtering. Specifications of the combined circuit, obtained in the same way using FPGA tools, are also reported.

Percentages shown are with respect to available resources of the chosen devices. Even for the combined circuit, LEs required are very low relative to available logic. This leaves plenty of room for logic needed by other ISP operations, e.g., tone mapping. Even for the combined circuit, bits required are low relative to available memory. This leaves some room for memory needed by other ISP operations. If additional memory or logic is needed, a different device may be selected from the same family, or from a different family.

When comparing the combined circuit to the separate circuits, LEs and bits required do not exactly sum due to optimizations. The same may be said for dynamic power. Also, max frequency is not exactly the worst max frequency. Due to the FIFO stage in Fig. 6, SPN filtering requires more memory and power than FPN correction. Finally, as static power is significant in the separate circuits, the combined circuit achieves notable savings in total power.

3.4.2 *Analog Competitor*

De Moraes Cruz et al. [8] proposed an analog circuit to correct offset variation only in linlog sensors. While the circuit is simple, the signal-to-noise-and-distortion ratio (SNDR) in the log region, which depends on temporal noise and residual FPN, was limited to 29 dB. In our previous work [5, 11], we demonstrated a peak SNDR (PSNDR) of

Table III. Specifications of designed circuits. FPN correction and SPN filtering, using cubic polynomials and for the FHD video format, were evaluated as separate and combined circuits. LEs and bits are given, in parentheses, as a fraction of available resources. The chosen Xilinx and Altera devices were the simplest ones in the Spartan-6 and Cyclone III families, respectively.

Circuit	Technology	Logic Elements	Memory Bits	Max Frequency	Static Power	Dynamic Power
FPN Correction	Xilinx XC6SLX4	178 (2.06%)	21 (0.01%)	222.2 MHz	13.9 mW	17.1 mW
	Altera EP3C5	261 (2.53%)	88 (0.02%)	178.6 MHz	46.1 mW	27.9 mW
SPN Filtering	Xilinx XC6SLX4	620 (7.17%)	73,737 (24.7%)	158.7 MHz	14.1 mW	44.8 mW
	Altera EP3C5	710 (6.88%)	61,440 (14.5%)	105.5 MHz	46.2 mW	31.8 mW
Combined ISP	Xilinx XC6SLX4	817 (9.45%)	73,767 (24.8%)	140.8 MHz	14.2 mW	48.4 mW
	Altera EP3C5	972 (9.42%)	61,528 (14.5%)	108.7 MHz	46.2 mW	42.6 mW

45 dB, the highest ever reported for either a log sensor, what we used, or a linlog sensor in the log region. Higher-order FPN correction was critical to our result.

Whereas De Moraes Cruz et al.'s self-calibration method is intended for hard real time, they do not report any clock frequencies of their 8 × 8 pixel prototype. They write "the proposed calibration can be executed at least at the same rate of a regular CDS operation," but add that "the frame rate of the array will not be evaluated in this work." As shown in Table III, our digital circuit for higher-order FPN correction can process up to 222 megapixels per second, or 7.4 megapixels at 30 fps, with the simplest Spartan-6 FPGA.

De Moraes Cruz et al. also do not report any measures of power consumption. With the simplest Spartan-6 FPGA, our digital circuit consumes 31 mW of power at the 62 megapixels per second required for FHD video.

3.4.3 *Mixed-Signal Competitor*

To correct offset and gain variation in linlog sensors, Storm et al. [9] proposed a mixed-signal circuit. The analog circuitry is simple and well documented, comprising several extra transistors per pixel and per column. Digital parts, some at chip level, adjacent to the sensor array, and some in an external FPGA, are documented so briefly that it is impossible to assess their complexity. The digital circuitry provides control signals for a self-calibration process and participates also in FPN correction.

Despite the appeal of a self-calibration process, we have shown [5, 11] that image quality is limited with log sensors unless higher-order FPN correction is employed. We calculate the PSNDR [14] of Storm et al.'s imaging system, using data they provide, to be 26 dB in the log region, which is significantly lower than the 45 dB we achieved.

Storm et al.'s prototype, comprising a 288 × 352 CIS and an FPGA, operates in hard real time at 26 fps. This corresponds to 2.6 megapixels per second. Because of timing issues with the self-calibration process, it is unclear how the work scales. The authors note "a maximum frame rate of 26 fps for an array of 288 rows." From data Storm et al. provide, it is impossible to separate out power consumption of the FPN correction. Their imaging system used 5.3 mW of digital power, "not incl. FPGA," and 61–84 mW of analog power.

Our all-digital circuit is competitive on frame rate and seems competitive on power too, while performing higher-order FPN correction on a much larger number of pixels.

3.4.4 *Digital Competitors*

Hoefflinger [10] proposed a digital circuit to correct OGB variation in log sensors. After FPN calibration, by approximate curve fitting of the model given in Eq. (1), FPN correction is implemented, using an FPGA, by transforming the fitted model approximately into a set of piecewise linear functions. While it is briefly explained and its complexity not reported, the digital circuit is likely of similar complexity to our FPN correction circuit.

Hoefflinger's imaging systems, which consumed up to 5 W of power, operated in hard real time. One system supported the VGA format, i.e., 480 × 640 pixels at 30 fps, or 9.2 megapixels per second. Another supported 496 × 768 pixels at 38 fps, or 14 megapixels per second. It is likely that Hoefflinger's FPN correction, on its own, would scale to larger formats. While a breakdown was not given, it is likely that power consumption of his FPN correction alone, in an equivalent FPGA, would be comparable to our reported figures.

An important difference between our digital circuit method and Hoefflinger's digital circuit is that our method leverages a recently published algorithm [5], which we also developed, that is not specific either to log sensors or Eq. (1). Hence, our method may be applied to realize a digital circuit for FPN correction of any monotonic nonlinear sensor, including linlog sensors. While Choubey and Collins [4] have developed a model, similar to but more complex than Eq. (1), for linlog sensors, no corresponding circuit has been proposed.

Stuck pixels exist in log and linlog sensors, as in linear sensors. However, neither De Moraes Cruz et al., nor Storm et al., nor Hoefflinger address them. In his Stanford lecture on the "Camera Processing Pipeline," Pulli [6] addresses "stuck pixels" alongside "pixel non-uniformity," i.e., FPN, advising to “replace with filtered values.” We show, in Fig. 4, that they are complementary, address both, and evaluate joint complexity, max frequency, and power consumption.

Latha and Sasikumar [15] implemented a two-stage median filter to process 256 × 256 pixels, i.e., 66 kilopixels, with 8 bpp. They showed that their circuit filtered salt-and-pepper, speckle, and Gaussian noise effectively. Although not reported in LEs, their circuit uses a similar amount of logic to what we report in Table III for 1080 × 1920 pixels, i.e., 2.0 megapixels, with 16 bpp. While briefly explained, their circuit needs more memory than ours, at least 100% the capacity, about 129 Kb, of their Xilinx Spartan-II device. It is unclear, from their paper, if their circuit also needed external memory.

While Latha and Sasikumar’s median filter operates in hard real time, it is unclear if they determined the max frequency of their circuit itself. The 200 MHz figure they report is simply the rated max frequency of the Spartan-II device. Although it is unclear at what frame rate, they report a power consumption of 202 mW. In contrast, we use STA to reliably determine a max frequency of 159 MHz, with our Xilinx Spartan-6 device, for processing 31 times as many pixels in pipeline fashion. Our circuit consumes 59 mW of power to process these pixels at 30 fps, i.e., what is required for FHD video.

4. CONCLUSION

Kim [1] writes, in a review paper, “WDR imaging is currently a hot issue in the mobile CIS market. Many commercial sensor providers are proposing various types of WDR sensors, such as the [linlog] type,” an approach that he champions. Kim also recognizes that FPN, especially in the log region, is a serious problem with nonlinear sensors.

Li et al. [5], i.e., our recent work, propose an algorithm for FPN correction of monotonic (non)linear sensors, which include linear, log, and linlog sensors, using low-degree polynomials. This background work is taken in a significant new direction in the current paper. Both works use experimental data from a log sensor [11] for validation.

The new direction includes the development, validation, and evaluation of a digital circuit method to automatically implement the background algorithm, for a wide variety of parameters, effectively and efficiently in hard real time. We also elaborate here on SPN filtering, mentioned briefly in our previous work. A digital circuit method for SPN filtering is similarly developed, validated, and evaluated.

To support a wide variety of parameters, such as polynomial degree and number of pixels, a design template in VHDL and a data file of parameters are processed by a Matlab script to generate a specific VHDL design. The design includes a recursive pipeline circuit for FPN correction that could not be implemented via VHDL generics. Using an FPGA design flow, the design is turned into digital circuits.

For readability, design templates are explained here using figures, tables, equations, and words. They include circuit schematics comprising synchronous and asynchronous elements, i.e., sequential and combinational logic, where all elements operate 100% in parallel. Image signals are processed in pipeline fashion strictly in sync with a clock signal. This is what guarantees hard real-time performance.

We validate and evaluate our novel methods by generating specific digital circuits, using the proposed design flow, for a variety of parameters. We target the simplest devices in the Xilinx Spartan-6 and Altera Cyclone III families, the lowest-cost families in the market at the time of this work. Evaluation assessed the complexity, max frequency, and power consumption versus parameters of interest.

Resulting circuits were effective, with either FPGA device, in processing FHD video, using cubic polynomials for FPN correction, at a rate of 62 megapixels per second. Moreover, with the Xilinx device, the FPN correction circuit functioned correctly up to 222 megapixels per second, and the SPN filtering circuit up to 159 megapixels per second.

The circuits were also efficient, especially the FPN correction. With the Xilinx device, the combined circuit to process FHD video used 9.45% of the available logic, 24.8% of the available memory, and 63 mW of power. SPN filtering aside, the FPN correction used 2.06% of the available logic, 0.01% of the available memory, and 31 mW of power.

In conclusion, this paper developed, validated, and evaluated novel digital circuit methods to correct and filter noise of nonlinear CMOS image sensors. Presented results provide excellent benchmarks against which future analog, mixed-signal, and digital circuits may be measured.

ACKNOWLEDGMENTS

The authors gratefully acknowledge the support of the Natural Sciences and Engineering Research Council, Canada, and Science Without Borders, Brazil.

REFERENCES

1 T.-C. Kim, “Wide dynamic range technologies: for mobile imaging sensor systems,” IEEE Consum. Electron. Mag. **3**, 30–35 (2014).

2 A. E. Gamal and H. Eltoukhy, “CMOS image sensors,” IEEE Circuits Devices Mag. **21**, 6–20 (2005).

3 D. Joseph and S. Collins, “Modeling, calibration, and correction of nonlinear illumination-dependent fixed pattern noise in logarithmic CMOS image sensors,” IEEE Trans. Instrum. Meas. **51**, 996–1001 (2002).

4 B. Choubey and S. Collins, “Models for pixels with wide-dynamic-range combined linear and logarithmic response,” IEEE Sensors J. **7**, 1066–1072 (2007).

5 J. Li, A. Mahmoodi, and D. Joseph, “Using polynomials to simplify fixed pattern noise and photometric correction of logarithmic cmos image sensors,” Sensors **15**, 26331–26352 (2015).

6 K. Pulli, “Camera Processing Pipeline.” https://web.stanford.edu/class/cs231m/lectures/lecture-11-camera-isp.pdf, May 2015.

7 S. Kavadias, B. Dierickx, D. Scheffer, A. Alaerts, D. Uwaerts, and J. Bogaerts, “A logarithmic response CMOS image sensor with on-chip calibration,” IEEE J. Solid-State Circuits **35**, 1146–1152 (2000).

8 C. A. de Moraes Cruz, D. W. de Lima Monteiro, E. A. Cotta, V. Ferreira de Lucena, and A. K. Pinto Souza, “FPN Attenuation by Reset-Drain Actuation in the Linear-Logarithmic Active Pixel Sensor,” IEEE Trans. Circuits Syst. I **61**, 2825–2833 (2014).

9 G. Storm, R. Henderson, J. E. D. Hurwitz, D. Renshaw, K. Findlater, and M. Purcell, “Extended dynamic range from a combined linear-logarithmic CMOS image sensor,” IEEE J. Solid-State Circuits **41**, 2095–2106 (2006).

[10] B. Hoefflinger, “High-Dynamic-Range (HDR) Vision,” *Springer Series in Advanced Microelectronics* (Springer, Berlin, Germany, 2007), Vol. 26.

[11] A. Mahmoodi, J. Li, and D. Joseph, “Digital pixel sensor array with logarithmic delta-sigma architecture,” Sensors **13**, 10765–10782 (2013).

[12] Xilinx, “All Programmable FPGAs and 3D ICs” https://www.xilinx.com/products/silicon-devices/fpga.html, Oct. 2017.

[13] Altera, “Intel FPGAs.” https://www.altera.com/products/fpga/overview.html, Oct. 2017.

[14] O. Skorka and D. Joseph, “Toward a digital camera to rival the human eye,” J. Electron. Imaging **20**, 033009-1–18 (2011).

[15] T. Latha and M. Sasikumar, “A novel non-linear transform based image restoration for removing three kinds of noises in images,” J. Inst. of Eng. (India): Series B **96**, 17–26 (2015).