Reconfigurable Image Signal Processors for Nonlinear CMOS Image Sensors

by

Maikon Ribeiro do Nascimento

A thesis submitted in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

in

Integrated Circuits and Systems

Department of Electrical and Computer Engineering

University of Alberta

Abstract

Moore's Law is dying, but this is breathing new life into system-on-chip (SoC) architectures. Industry leaders like Intel and Xilinx are investing in heterogeneous computing devices like the reconfigurable SoC, which integrates a field-programmable gate array (FPGA), a multi-core microprocessor (μP), and a variety of peripheral interfaces on the same silicon chip. Market drivers include autonomous vision systems, which benefit from the hard real-time capabilities of the FPGA, the operating system capabilities of the μP, and the sensor/networking capabilities of the interfaces. To these ends, this thesis offers a case study on the image signal processor (ISPr) of a nonlinear complementary metal-oxide semiconductor (CMOS) imaging system. Unlike a linear CMOS image sensor (CIS), a nonlinear CIS is able to image high/wide dynamic range scenes in single exposures at video rates, ideal features for outdoor applications involving motion. The thesis leverages previously-published image signal processing (ISPg) algorithms, identified clearly as background work, to develop novel digital circuit methods for fixed pattern noise correction, salt-and-pepper noise filtering, and histogram-based tone mapping. Beyond digital circuits for a specific CIS, the proposed digital circuit methods generate circuits for an arbitrary monotonic CIS, such as linear, log, or linlog, based on supplied parameters. Generated circuits are validated for a variety of parameters, including megapixel scenarios at standard video rates. They are shown to be very efficient, in terms of circuit complexity, maximum frequency, and power consumption, and have significant advantages over literature approaches, including ones that use application-specific integrated circuits and graphics processing units. As a stepping stone to exploiting the ISPr, for a nonlinear CIS in an autonomous vision system, the thesis also proposes a novel design flow, for a reconfigurable SoC, to make the FPGA the master of the imaging system, via interrupt service routines, running on the µP, and direct memory access circuits, embedded in the FPGA. Two approaches are investigated to do this: one favours open source tools, and the other performance. The work demonstrates local processing of high definition video in hard real time with results communicated over a web page on demand.

Preface

Chapter 2 of this thesis has been published as Maikon Nascimento, Jing Li, and Dileepan Joseph, "Digital Circuit Methods to Correct and Filter Noise of Nonlinear CMOS Image Sensors," *Journal of Imaging Science and Technology*, 62(6), 60404 (2018). Nascimento was responsible for the generic design flow as well as the design, implementation, validation, and evaluation of all circuits. He contributed to the literature review and the explanation of novelty and significance. He wrote the initial version of the paper, which he defended at his Ph.D. candidacy exam, and edited the final version. Li was responsible for the background algorithms and contributed to the circuit designs. Joseph supervised the work and wrote the final paper, included as Chapter 2.

Chapter 3 of this thesis is intended for publication with the same authors, listed above, in the same order. Nascimento was responsible for the generic design flow as well as the design, implementation, validation, and evaluation of all circuits. He was also responsible for the literature review and the explanation of novelty and significance. He wrote the initial version of the work, which he defended at his internal Ph.D. final exam, and edited the final version, included here. Li was responsible for the initial background algorithms and contributed to the circuit designs. Nascimento developed the final background algorithms with Joseph. Joseph supervised the work and edited the final version, which included a rewriting of the two middle sections.

Chapter 4 of this thesis has been published as Maikon Nascimento and Dileepan Joseph, "System-on-Chip Design Flow for the Image Signal Processor of a Nonlinear CMOS Imaging System," *Proceedings of the IS&T International Symposium on Electronic Imaging*, 9, 363 (2019). Nascimento was responsible for the integrated design flow as well as the design, implementation, validation, and evaluation of the entire system, which comprised software, firmware, and hardware. He was also responsible for the literature review and the explanation of novelty and significance. Nascimento developed the interfacing method and wrote the final version of the work included here. Joseph supervised the work and contributed edits mainly after it was accepted for publication.

The introduction and conclusion chapters were written by Nascimento and edited by Joseph. Parts of the introduction were taken from Nascimento's Ph.D. candidacy report, also written by Nascimento and edited by Joseph. Nascimento wrote the initial version of the abstract, which Joseph rewrote. The preface was written by Joseph and edited by Nascimento.

As a journal article and conference proceeding, respectively, Chapters 2 and 4 were both peer reviewed. Moreover, the journal article of Chapter 2 won the 2019 Itek Award. According to the Society for Imaging Science and Technology, "This award recognizes an outstanding original student publication in the field of imaging science or engineering published in a Society journal (JIST or JEI) during the preceding calendar year. The paper must comprise a significant contribution in the field of imaging and must conform to high standards of technical and scientific writing."

Acknowledgements

Following a chronological order, my first thank you is to my supervisor, Dr. Dileepan Joseph. Since the moment zero, when he replied to my wondering emails, then following up with me when I was coming to Canada to start the Ph.D. program, and then finally working together, Dil has been a supervisor, a presentation coach, a friend, and someone who has helped me to elevate my engineering standards. The quality of my work has been pushed further and, at the same time, he was also flexible to respond to my interests.

Second, my studies were possible largely due to the financial support of CAPES, a Brazilian federal government agency under the Ministry of Education, and their program Science Without Borders. A special thanks, therefore, to Dilma Rousseff, former president of Brazil, as well as to those, in Brazil and Canada, who helped to administer the doctoral scholarship program.

At the end of the first year of my Ph.D., I started dating my current partner, Cole P. Moffat, who has helped me so much since then with all his support, stability, company, and love. He kept me grounded, adding a family to my life in Canada. With Cole and I, there are two dogs, Kuroka, who is my favourite, then Poncho, and a “mistake” cat, Luna.

In this direction, I must also thank my dear friends in Edmonton for so many nice adventures exploring the beauty of Alberta. Thank you especially to Dr. Jonas Valloton, my super close friend in Edmonton, and also to Dr. Fernando Saccon, whom I met more recently.

I want to thank the Faculty of Graduate Studies and Research (FGSR) and the Graduate Students’ Association (GSA), at the University of Alberta, for their financial support of my trip to San Francisco to present some of my work. I also want to thank the GSA for an Emergency Bursary, i.e., for extra financial support, at the end of my program.

Finally, thank you to my family and friends back in Brazil. Warm memories of you were helpful during the cold Canadian winters!

Table of Contents

List of Tables

List of Figures

List of Acronyms

4KUHD	4K ultra HD	37
ADC	analog-to-digital converter	27
AI	artificial intelligence	103
AMBA	Advanced Microcontroller Bus Architecture	91
API	application program interface	107
ASIC	application-specific integrated circuit	55
AXI	Advanced eXtensible Interface	90
BER	bit error rate	83
bpp	bits per pixel	26
BRAM	block RAM	96
CCD	charge-coupled device	2
CDF	cumulative distribution function	65
CDS	correlated double sampling	20
CIS	CMOS image sensor	100
CMOS	complementary metal-oxide-semiconductor	105
CPU	central processing unit	103
dB	decibel	74
DDR	double data rate	87
DMA	direct memory access	104
DR	dynamic range	101
DSPg	digital signal processing	96
DSPr	digital signal processor	8
DTB	device tree binary	91
FF	flip-flop	96
FHD	full HD	101
FIFO	first-in first-out	102
FPGA	field-programmable gate array	100
FPN	fixed pattern noise	100
fps	frames per second	85
GP	general purpose	90
GPU	graphics processing unit	103

GUI	graphical user interface	87
HD	high definition	101
HDF	hardware description file	91
HDL	hardware description language	101
HDR	high dynamic range	106
HLS	high-level synthesis	87
HP	high performance	90
HQVGA	half quarter VGA	74
HTML	hypertext markup language	94
IDE	integrated development environment	87
IO	input-output	8
IP	intellectual property	107
IPR	inverse polynomial regression	25
IRQ	interrupt request	91
ISPg	image signal processing	105
ISPr	image signal processor	100
ISR	interrupt service routine	84
JTAG	Joint Test Action Group	86
LDR	low dynamic range	85
LE	logic element	96
linlog	linear-logarithmic	105
log	logarithmic	105
LPF	low-pass filter	60
LSB	least significant bit	62
LUT	look-up table	86
MSB	most significant bit	30
NRE	non-recurring engineering	11
OGB	offset, gain, and bias	24
OS	operating system	106
PL	programmable logic	86
PMF	probability mass function	59
PS	processing system	86
PSNDR	peak SNDR	48
PSNR	peak SNR	74
RAM	random-access memory	106
RHS	right hand side	26
ROM	read-only memory	71
SD	Secure Digital	105
SDK	software development kit	87

SDRAM	synchronous dynamic RAM	87
SNDR	signal-to-noise-and-distortion ratio	48
SNR	signal-to-noise ratio	15
SoC	system-on-chip	100
SPN	salt-and-pepper noise	100
SRAM	static RAM	8
STA	static timing analysis	79
TMO	tone mapping operator	100
TTVGA	tenth tenth VGA	74
UART	universal asynchronous receiver-transmitter	86
µP	microprocessor	100
USB	Universal Serial Bus	86
VDMA	video DMA	99
VGA	video graphics array	74
VHDL	VHSIC hardware description language	21
VHSIC	very-high-speed integrated circuit	21
WDR	wide DR	106

Chapter 1

Introduction

This thesis advances the state-of-the-art in nonlinear complementary metal-oxide-semiconductor (CMOS) imaging systems by researching digital circuits and interfaces, including design flows thereof, which help to turn a nonlinear CMOS image sensor (CIS) into a complete imaging system that can operate efficiently in hard real time. For these purposes, the thesis investigates the use of low-cost reconfigurable system-on-chip (SoC) devices, focusing in particular on the field-programmable gate array (FPGA) part of these devices, but considering also its relationship to the microprocessor (µP) part.

This chapter begins with a statement of the thesis objectives, which determines the background of interest. The background concerns nonlinear CISs and the important distinction between image signal processing (ISPg) and image signal processors (ISPrs). It summarizes other research, in the industry and academia, relevant to this work. Thereafter, the chapter defines the scope of the thesis, elaborating on the methodology used, such as the chosen design tools and background algorithms, within the boundary of interest. Further details are given in the subsequent chapters.

1.1 Objectives and Background

This thesis explores digital circuit design and digital system integration to realize an ISPr for a nonlinear CIS. The platform of choice is an FPGA within a reconfigurable SoC, which is a heterogeneous computing architecture having also a µP on the same silicon chip. All circuits and systems investigated and developed, through this research, are conceptualized to be generic and applicable for a wide variety of nonlinear CISs. Selected ISPg operations, for the ISPr, comprise several functions that are essential to produce a useful high dynamic range (HDR) imaging system.

In terms of relevant background, a motivation for nonlinear CISs is first presented. This emphasizes their advantages for HDR imaging applications, as well as their disadvantages. The disadvantages may be mitigated by ISPg, a subject that is reviewed next, with an emphasis on fixed pattern noise (FPN) correction and tone mapping. Finally, this section reviews different architectures that may be used to realize an ISPr, emphasizing the FPGA architecture and, especially, the FPGA as part of a reconfigurable SoC.

1.1.1 Nonlinear CMOS Image Sensors

Cameras are a fundamental component in mobile devices and embedded systems, aggregating value to the product and adding the possibility of computer vision. At the beginning of the digital camera era with CMOS or charge-coupled device (CCD) sensors, improvements were made primarily by increasing the spatial resolution. Today's improvements are made in the image quality domain instead, as mentioned by Fontaine [18]. The dynamic range (DR) is a key feature that increases the information that an image or video can register.

The history of photography starts with analog cameras, which depended on film to register scenes. Fig. 1.1 presents the evolution of photography and market penetration, when there was a competition between CCD and CMOS at the beginning of the digital camera era. Largely due to the popularity of mobile phones, the CIS has surpassed the CCD image sensor in the industry today, having key advantages for embedded devices such as power consumption, integration, and architecture, as mentioned by Suzuki [51].

Increasing the DR is a direction that image sensors and imaging systems are taking to advance camera technology, from better manufacturing methods to the integration of extra circuitry. Besides capturing real scenes more accurately, HDR imaging targets better performance at lower light levels. Moreover, the approach can be applied to other ranges of the electromagnetic spec-

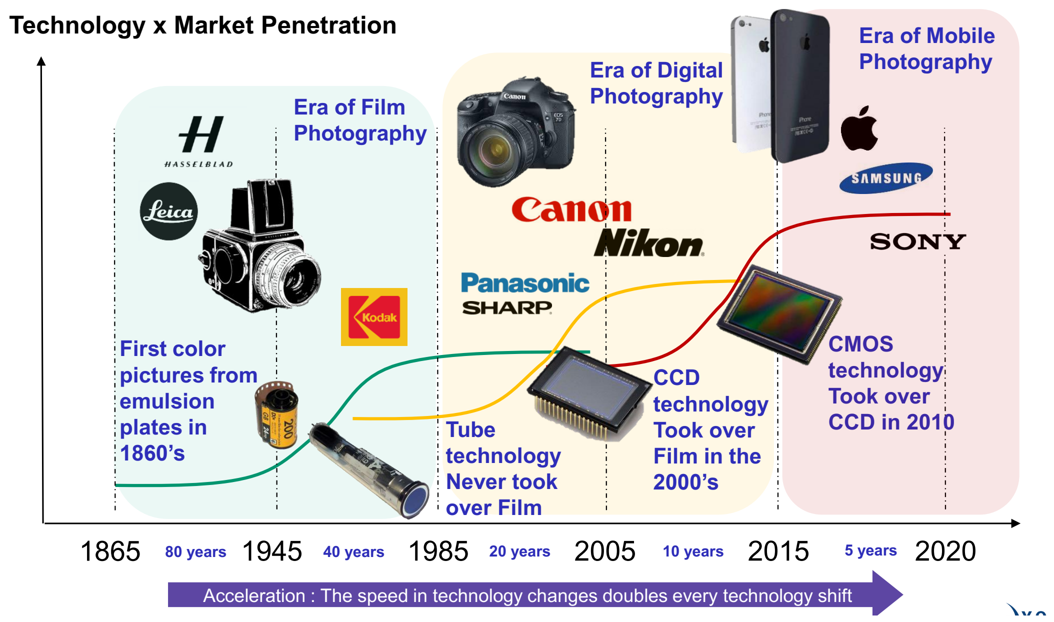


Figure 1.1: Evolution of photography. Continuing improvements to CIS technology is expected to lead to continued growth of CIS market share and applications. Taken from Cambou and Jaffard [10].

trum, allowing even more sensory information to be collected.

Ways to capture HDR images include: combining pictures from a standard linear CIS taken with multiple and different exposures; the use of a special linear CIS with interleaved sensors set to either overexpose or underexpose pixels; or another special CIS that has a nonlinear response covering a wide range of luminances simultaneously without saturation. Kim reviewed these different approaches recently [28]. Under nonlinear CISs, also called wide DR (WDR) sensors, he focused on the linear-logarithmic (linlog) sensor, a variation of the classic logarithmic (log) sensor, and described it as his preferred solution for HDR image capture, the technical challenges notwithstanding.

A nonlinear CIS captures a wide DR, as shown in Fig. 1.2. Similar to the human eye, it can register bright, dim, and in-between regions at the same time without saturation or motion-related artefacts. However, drawbacks of such WDR technologies include an increased need for ISPg solutions to handle complex nonlinear responses that vary from pixel to pixel. Compact low-cost and low-power solutions, which can operate in hard real time and scale to high

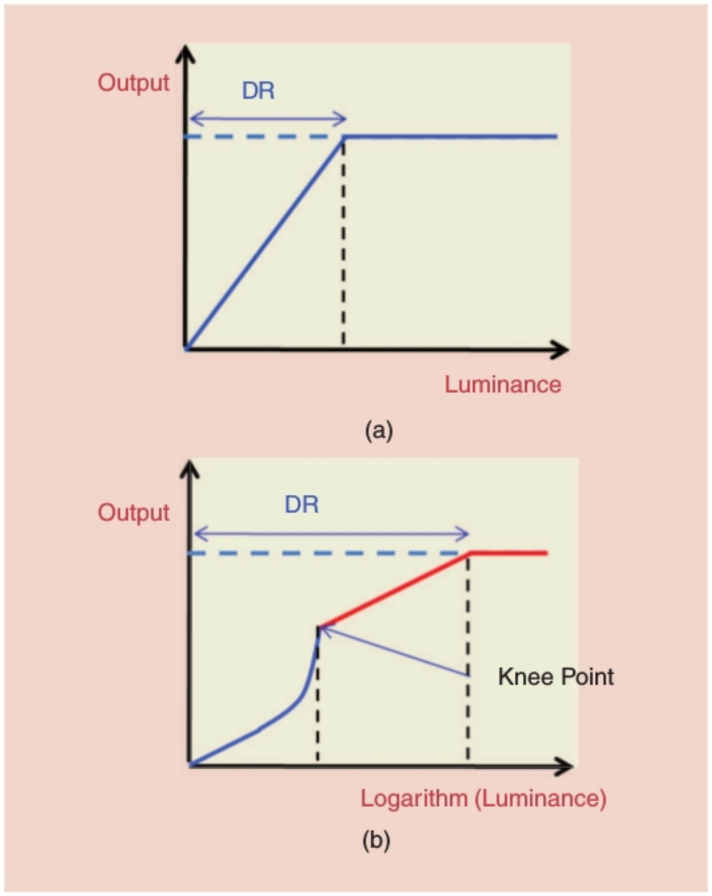
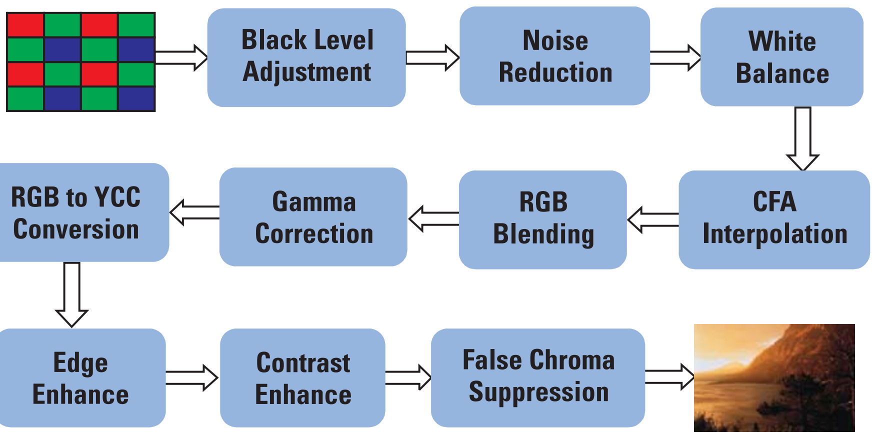


Figure 1.2: Comparison between (a) a linear and (b) a nonlinear sensor. The nonlinear response has a wider DR. In addition, for the linlog sensor shown, the unsaturated region has a complex response. Taken from Kim [28].

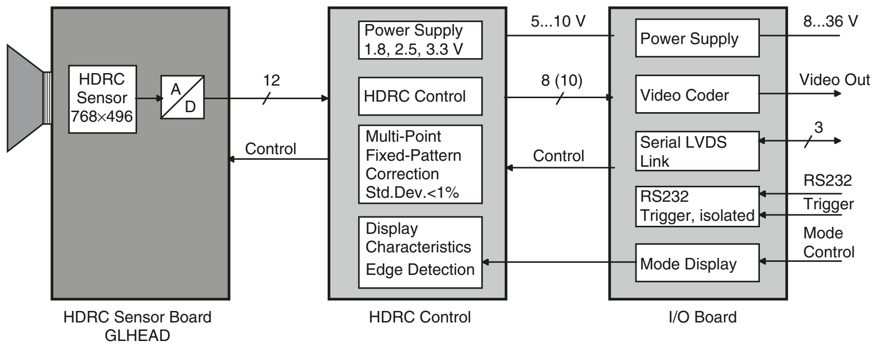
definition (HD) video formats, would help in this regard.

1.1.2 Image Signal Processing

There are two similar acronyms employed in this thesis: ISPg to describe the signal and image processing required by an imaging system; and ISPr to describe the integrated circuits and systems used to implement the ISPg. Figure 1.3(a) presents a block diagram, from a general-purpose ISPr manufactured by Texas Instruments, that identifies several ISPg operations. Choosing which ISPg operations to use may vary according to the complexity of the CIS and the end application. For example, in Fig. 1.3(b), a special-purpose ISPr implements ISPg operations specific to a log sensor.



(a)



(b)

Figure 1.3: Examples of ISPg developed in industry and academia. (a) Illustration, taken from Texas Instruments, of several ISPg operations present in a commercial ISPr. (b) Illustration, taken from Hoefflinger [20], of ISPg operations present in an ISPr for a nonlinear CIS.

To elaborate on the figure, Fig. 1.3(a) shows the multi-stage ISPg that Texas Instruments developed for colour linear CISs. This example illustrates what kinds of operations can be embedded in a conventional commercial ISPr. Each stage influences the final image quality. In this fashion, the image processing pipeline extracts the most data from the CIS. Figure 1.3(b) shows that, for a log sensor, the ISPg includes multi-point FPN correction. This image processing pipeline also includes edge detection, a higher-level operation associated with computer vision end applications.

For a simple linear CIS, FPN correction is accomplished using correlated double sampling (CDS), which is an analog circuit method. High-end cameras, including linear ones, may require more complex circuit methods, including

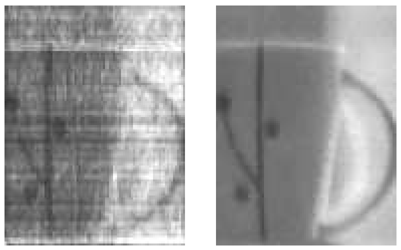

(a) (b)

Figure 1.4: Example of FPN correction with a nonlinear CIS. These images, taken from Skorka *et al.* [49], are from a log sensor: (a) before FPN correction; and (b) after FPN correction using Li *et al.*'s [34] algorithm.

digital circuit methods, to correct the pixel responses. Related to FPN correction, ISPg operations may include median filtering to handle dead pixels. In general, HDR imaging systems require tone mapping as a necessary operation since displays cannot typically reproduce all brightness levels.

When we look at an image sensor as a matrix of pixel sensors, there is a particular intrinsic response for every pixel sensor. The variation of the sensor response creates a similar effect to noise but in a fixed pattern. Figure 1.4 presents an image before and after FPN correction. The name FPN is used because it depends on the spatial positions of the pixels and does not change over time. Thus, FPN may also be called spatial noise, in contrast to temporal noise, which is also present. The FPN is worse for nonlinear CISs and its correction is more complicated in comparison to linear CISs [28].

The algorithm used to correct FPN, in the example of Fig. 1.4, was published recently by Li *et al.* [34]. It is introduced in Section 1.2.2. Another and more common solution, developed by Joseph and Collins [23], is to characterize the curve of the sensor response, which for log sensors is given by:

$$y \approx a + b\ln(c + x), \tag{1.1}$$

where y is the digital response of a pixel sensor to light stimulus x, a is the offset, b is the gain, and c is the bias. Joseph and Collins [23] explained that FPN correction may be performed after calibration of the parameters, by inverting (1.1). Hoefflinger's [20] ISPg, in Fig. 1.3(b), implements an approximation to this. In addition to its overly approximate nature, Hoefflinger's approach

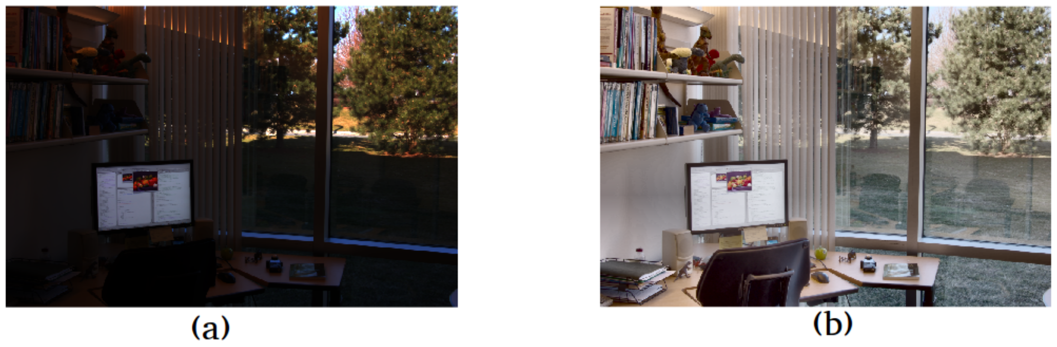


Figure 1.5: Examples of tone mapping applied to an HDR image. (a) An HDR image displayed with a naive TMO, resulting in loss of detail and contrast. (b) The same image displayed with a TMO that preserves more contrast and detail. Example taken from MATLAB's Image Processing Toolbox.

cannot be applied to other nonlinear sensors, such as linlog ones.

Tone mapping, performed with a tone mapping operator (TMO), is an essential operation in an HDR imaging system. Once the richer image has been registered, or video has been recorded, we need to display it on a low dynamic range (LDR) device, such as a computer or cellphone screen. The properties of the tone mapping are fundamental for avoiding artefacts, halos, or distortions. In addition, the richer details and higher contrast of the HDR image has to be preserved, as explained by Larson [31].

Figure 1.5 demonstrates the general concept of tone mapping. On the left is an HDR image that is naively displayed. The right side shows the image after using a TMO, where the room features can be seen even with the sunlit window. With tone mapping, the contrast is better, the picture registers both the interior and exterior of the room without saturation, and the observed reflection on the glass is even. Besides the static aspects, treated as shown in the figure, there are also temporal considerations for videos, which should play smoothly without visual artefacts caused by the TMO itself.

Typically, TMOs are classified into two mutually-exclusive categories: local and global. A global TMO consists of the same operation applied to each pixel in the image, a function that does not vary with pixel location although the function itself may depend on all pixel values. A local TMO uses a function that depends on the pixel position. Typically, the function depends on pixel values in the local neighbourhood, although all pixel values may contribute. Some authors have claimed that local TMOs perform better. However, in a review paper, Eilerstsen *et al.* [14] compared different TMOs and concluded that human observers preferred global ones.

1.1.3 Image Signal Processors

In general, ISPrs may be implemented using central processing units (CPUs) (also called µPs), graphics processing units (GPUs), FPGAs, or application-specific integrated circuits (ASICs). Unless a researcher is developing a new CPU or GPU architecture, these approaches tend to be algorithm-focused, although knowledge of the underlying hardware helps for optimization purposes. In contrast, FPGA and especially ASIC approaches are circuit-focused, although integrated development environments (IDEs) are employed, as in software development, with hardware description languages (HDLs).

An FPGA is a matrix of configurable digital electronic components that uses schematic capture or an HDL to specify a digital circuit or system. It is a reconfigurable highly-parallel architecture that is optimized for digital circuits built from a set of fundamental operations. The underlying technology of an FPGA is the static RAM (SRAM), which is volatile and meant to be stable and dense [52]. From the design specification, a digital circuit is created with the help of FPGA design tools. The circuit may use simple logic gates and routing to implement digital functions, as well as memory, arithmetic, digital signal processing (DSPg), input-output (IO), and other embedded components, depending on the family of the FPGA and project requirements.

Figure 1.6 compares the computation per unit power of FPGAs, GPUs, and digital signal processors (DSPrs), where the latter are specialized µPs. Because these results are from a leading FPGA vendor, we consider other sources below. Nevertheless, it is noteworthy that a leading FPGA vendor promotes its technology on the basis of its power efficiency.

In the literature, researchers have compared the performance of multiple platforms for addressing the same or similar tasks. Asano *et al.* [4] compared FPGA, GPU, and CPU solutions for three ISPg problems. Their experimental

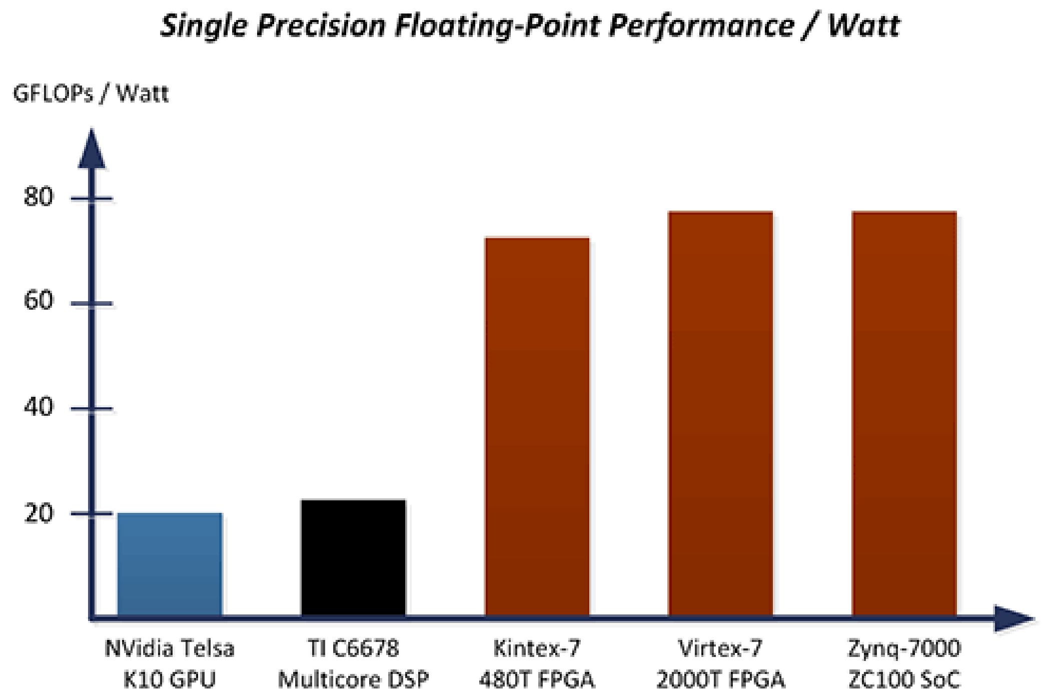


Figure 1.6: Comparison between FPGAs, GPUs, and DSPrs. This chart, taken from Xilinx [58], shows that FPGAs have a better performance per Watt for single-precision floating-point operations, for which GPUs are optimized.

results showed that, in terms of the frame rate, the FPGA solution surpassed the performance of the GPU and CPU ones for two of the three problems.

For computer graphics, the advantages of the GPU are well known. A GPU is a highly parallel architecture, with its own low-level programming language, to divide single-precision floating-point tasks, especially 3D vector operations, across many cores. However, a GPU can suffer in performance when the bottleneck between the cores and cache memory is not properly handled. Another drawback of GPUs is power consumption, which makes it difficult to apply them for embedded systems dependent on limited battery power.

In a review article, Benkrid *et al.* [7] compares FPGAs and GPUs, amongst other platforms, for the Smith-Waterman pairwise-sequence-alignment problem. The researchers compared speed, development time, overall cost, and energy consumption, and reported that FPGAs were more cost effective and energy efficient for that problem.

Berten [8] compares FPGAs to GPUs and concludes that FPGAs are better in terms of computation per Watt, latency, interfaces, and size (easier to cool). However, GPUs are better for floating-point operations, backward compatibility, flexibility, development, and processing per unit cost. Considering development, GPUs are superior because it is easier to find developers. The GPU field is adopting tools and personnel from the C/C++ community, which is larger than the HDL community. Moreover, there are open-source solutions for GPU development, making it more transparent.

Returning to CPUs, especially ones with operating systems (OSs), they are truly the only general-purpose platforms available and have the advantage of simplified programming. Thanks to widespread use, they enjoy programmer-friendly environments for prototyping and debugging. A drawback of the FPGA approach is a more complex design flow and difficult debugging.

Nonetheless, CPUs are often paired with hardware accelerators, such as GPUs and FPGAs. Texas Instruments developed ISPrs, like the one of Fig. 1.3(a), composed of a combination of CPUs and GPUs on the same circuit board. These ISPrs rely on parallel computing with the GPUs, to achieve high data processing rates, in order to handle ISPg of HD video efficiently.

For real world systems, a standard FPGA lacks out-of-the-box features such as Internet connectivity (WiFi), display drivers, memory card access, etc. A CPU running an OS would have support for all these peripherals, which are essential to develop a fully functional imaging system. To address this issue one could use an FPGA to execute the ISPg operations in real time, and a CPU with an OS for connectivity. If this is done on the same silicon device, it is called a reconfigurable SoC, where the FPGA implements a digital circuit part of a system, and the CPU implements an OS-based software algorithm part of the system.

Instead of an FPGA, an ASIC may be used as an accelerator. A comparison can be made between these two options. Projects involving FPGAs are more expensive in comparison to ASICs for high-volume production. However, from low to medium volume production, it makes more sense to employ FPGAs. Figure 1.7 presents the crossover point between these two platforms.

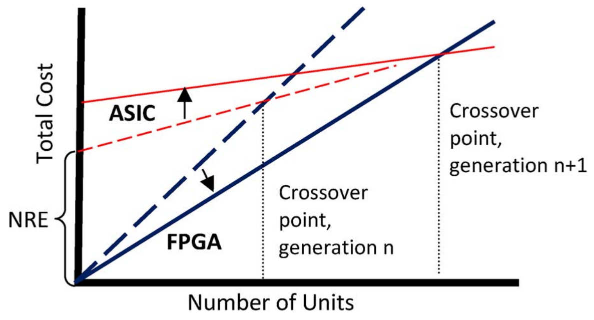


Figure 1.7: Crossover point between ASIC and FPGA solutions. The total cost of projects with FPGAs increases more with volume than it does with ASICs, but FPGA projects do not have the initial NRE cost. Taken from Trimberger [53].

While ASICs have a higher initial cost due to non-recurring engineering (NRE), FPGAs are more expensive when the number of units grows beyond the crossover point. However, technology trends are shifting the crossover point, making it more economical to use FPGAs for larger volumes than in the past. As a result, it is becoming harder and harder to justify an ASIC approach for an early stage design. Nevertheless, because of similar tools and techniques for digital circuit design, FPGAs remain suitable as a stepping stone toward an eventual ASIC solution. Many problems that would have to be solved for an ASIC design would be solved in the course of realizing an FPGA design. They may also be solved faster and at lower cost.

Improvements in manufacturing processes are adding components to low-cost FPGA devices, such as memory, transceivers, clock management, first-in first-outs (FIFOs), and encryption. Following recent trends for CMOS devices in general, FPGAs increasingly employ 3D integration, having their internal blocks stacked and using vertical connections to speed up data transfer, delivering more power efficiency and signal integrity. Furthermore, instead of emulating a CPU within the FPGA, modern SoC architectures, like the Zynq family from Xilinx, favour a dedicated multi-core µP integrated on the same silicon, along with a variety of interfaces for peripheral devices.

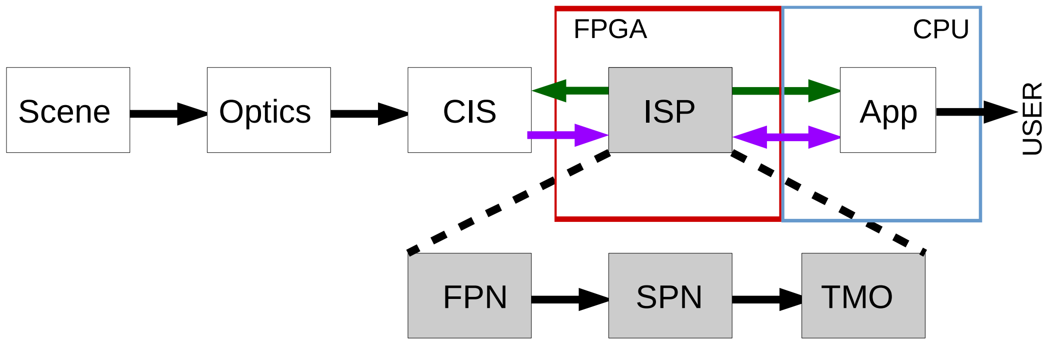


Figure 1.8: Proposed imaging system with a CIS and an ISPr. This thesis investigates digital circuits for FPN correction, SPN filtering, and tone mapping, essential ISPg operations for a nonlinear imaging system. It also investigates interfacing issues, when the circuits are realized in a reconfigurable SoC, where part of the system is implemented in an FPGA and another part is implemented in a CPU on the same silicon chip.

1.2 Scope and Methodology

Figure 1.8 illustrates the scope of this thesis, which is explained further in the figure's caption. The methodology of this work centers on the design of robust and scalable digital circuits, not to present one specific solution but a generic solution capable of solving a wide variety of problems. The platform chosen is the FPGA, when it is integrated within a reconfigurable SoC, which means a multi-core µP and peripheral interfaces are also available.

We provide a high-level overview of the chosen platform below, including its value as a case study for heterogeneous computing. Of particular interest is the interfacing, illustrated in Fig. 1.8, between the FPGA and multi-core µP, also called the CPU. Thereafter, we discuss the selected ISPg operations, namely FPN correction, salt-and-pepper noise (SPN) filtering, and tone mapping. Generic digital circuits will be investigated for each of these.

1.2.1 Reconfigurable System-on-Chip

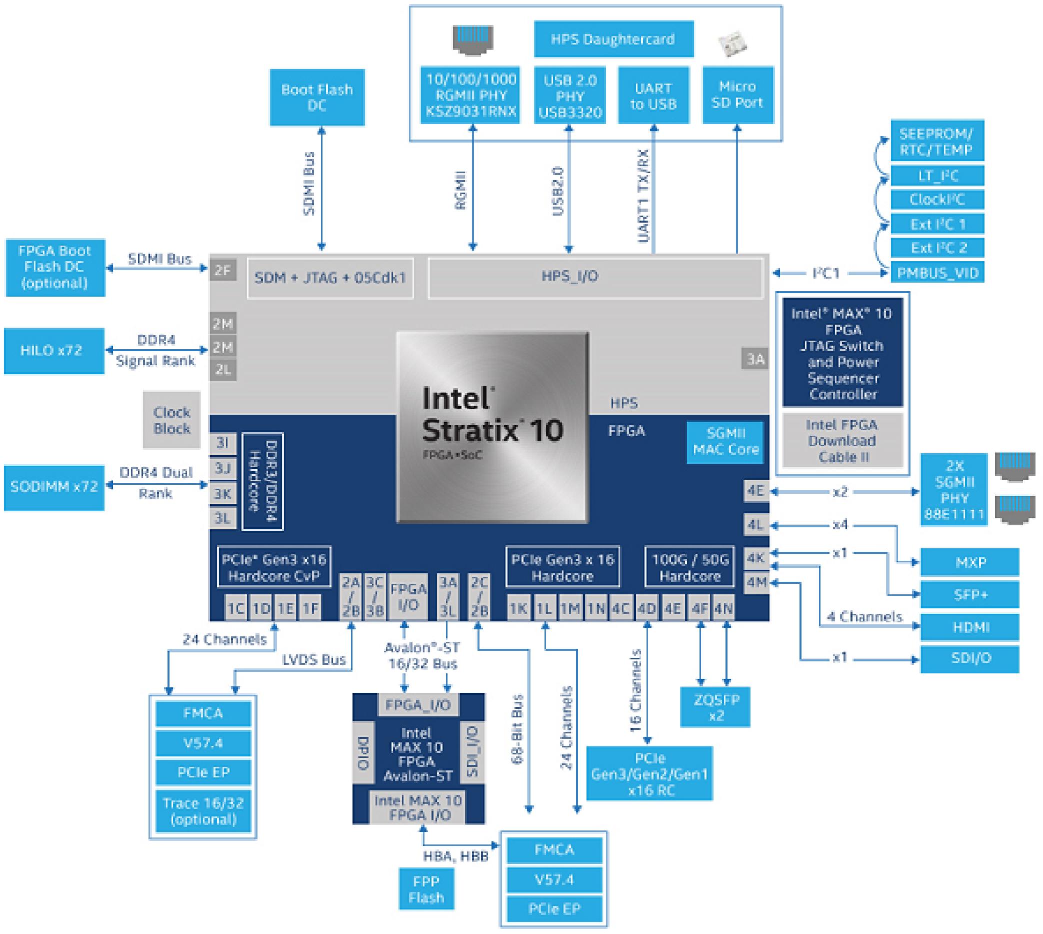
An ISPr could be implemented using a CPU, a GPU, an ASIC, or an FPGA, each with its own advantages and disadvantages. We chose an FPGA approach partly because we are more interested in novel integrated circuits and systems aspects, than in established signal and image processing aspects, of the chosen ISPg. Considering the high NRE costs of an ASIC approach, an FPGA approach is preferred for low volume production, which is the case for this research.

We also chose the FPGA approach because we hypothesize it would lead to an efficient solution, i.e., a solution where the ISPg of interest could be implemented in a low-cost device, with low power consumption, yet still scale to HD video formats. This hypothesis will be tested through design, validation, and evaluation. We assume that, as in Hoefflinger's work [20] on a nonlinear imaging system, an FPGA would be required anyway to control a nonlinear CIS. Thus, it makes sense to use the resources available in the FPGA, to implement the required ISPr, if our hypothesis proves true.

There are two main suppliers of FPGAs: Intel, who bought Altera in 2015, and Xilinx. Each has their own set of tools, which generate the binary file that configures a particular FPGA. Both sets of tools may be used to estimate the number of resources used, which is a good indication of the complexity of a design. To enhance the generality of our work, we use both sets of tools in all our work, except for interfacing issues between the FPGA and µP sides of a reconfigurable SoC, in which case we only use the Xilinx tools.

In particular, we synthesized our designs using Quartus 13.0, from Intel/Altera, and ISE 14.7, from Xilinx. To validate our novel digital circuits, we tested them against the output of the background algorithms implemented in Matlab. The validation aims for *bit true* design, i.e., zero bit error rate (BER), using functional simulation. We also run static timing analysis (STA) for timing closure. It is required to estimate max frequency of operation and power consumption. A sequence of small images were used first for manual validation. Afterward, HD images were tested automatically.

Both Intel/Altera and Xilinx offer reconfigurable SoCs for heterogeneous computing, as shown in Fig. 1.9. The technology is similar, which means that even though we use Xilinx tools alone, for part of our work, our conclusions may generalize. Overall, Intel is a larger company than Xilinx, but the FPGA division, formerly Altera, inside Intel is smaller than Xilinx, a company that has been focused on FPGAs for a longer time than Intel.



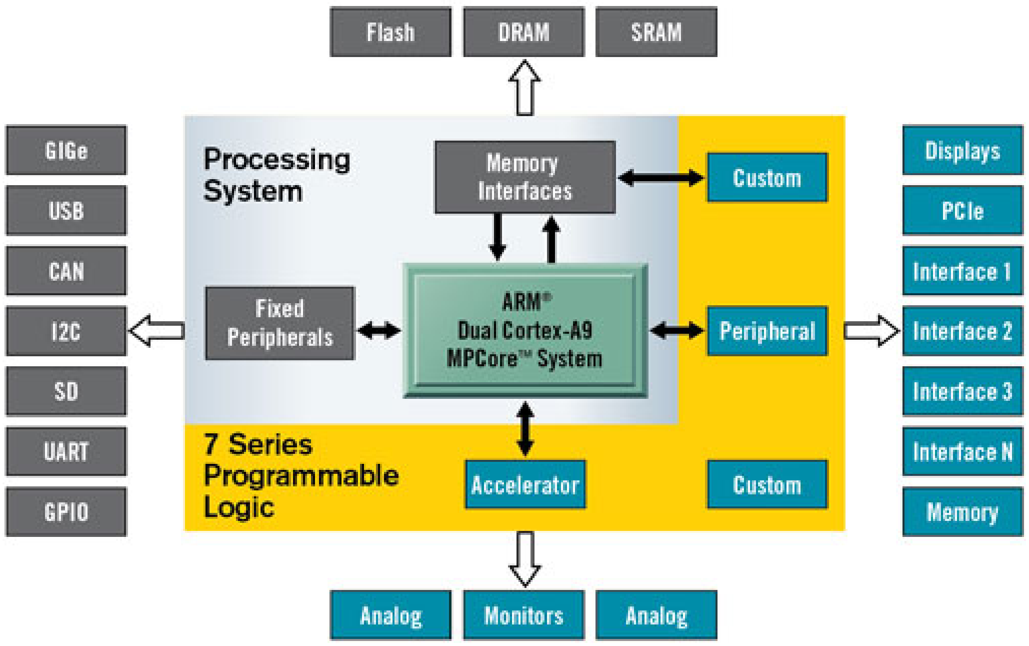


Figure 1.9: Reconfigurable SoCs from leading suppliers. Intel (top) and Xilinx (bottom) both offer heterogeneous platforms having an FPGA and a µP.

At the onset of this thesis, there were more reconfigurable SoC kits available from Xilinx, which was the major reason we focused on their Zynq family of devices. In recent years, both Xilinx and Intel have launched more advanced reconfigurable SoCs. The Zynq family remains a low-cost solution. Normally, a decision between the two vendors may consider the availability of custom components for a project. Specific circuits, already embedded in the silicon, will leave more generic resources available for other custom circuits, and will have better performance due to ASIC-level optimizations.

We envision a portable, fully functional, nonlinear imaging system for HDR computer vision applications, even though we focus on the ISPr required to get there. Challenges arise in ensuring hard real-time performance when a general purpose OS is running on the µP side of the SoC. Chapter 4, therefore, investigates design flows for heterogenous computing, mainly to make the FPGA the master of the system. Unlike other parts of our work, where simulation results from FPGA design tools suffice, we actually deployed our design in a real Zynq device to investigate these challenges experimentally, especially when scaled to HD resolutions at standard video rates.

1.2.2 FPN Correction and SPN Filtering

In terms of the chosen ISPg operations, FPN correction ensures that all pixels in an imaging system have the same response. Not only does it result in a uniform image for a uniform scene but also it increases the signal-to-noise-and-distortion ratio (SNDR), which depends on the signal-to-noise ratio (SNR) and residual FPN, of non-uniform images for non-uniform scenes [48]. By leveraging a background algorithm that uses low-degree polynomials, this thesis presents a generic digital circuit method for a nonlinear CIS.

The background algorithm was developed by Li *et al.* [34], where the average response of all pixels is considered to be the ideal response of each one. Actual responses are mapped toward ideal responses using a set of low-degree polynomials calibrated in the inverse direction. As with other FPN correction methods in the literature, a one-time calibration is required. The polynomial degree varies according to the complexity of the mapping, which is not the

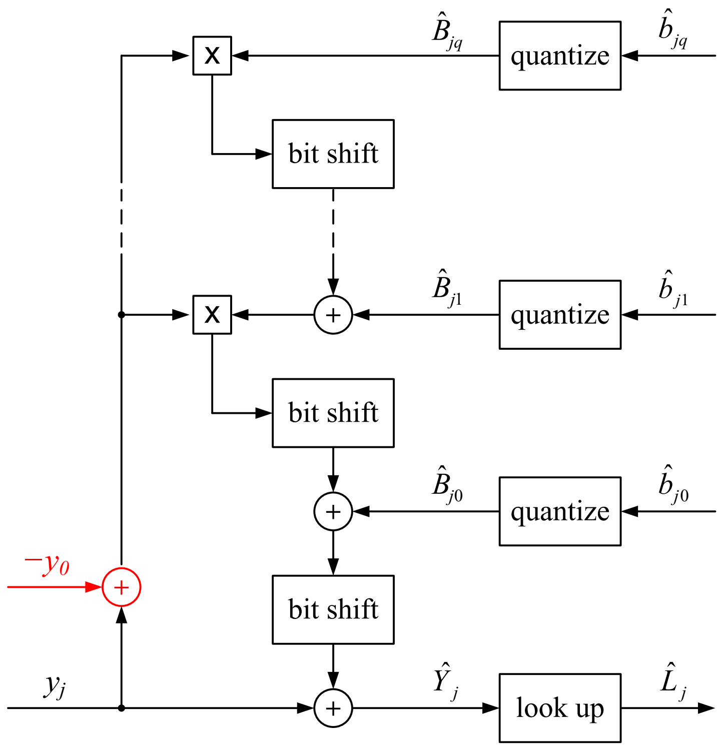


Figure 1.10: Background fixed-point algorithm for FPN correction. Here, y_j represents the actual pixel response, where j indexes pixels in sequence. Parameters $\hat{B}_{jk}$ represent FPN correction coefficients. We replace the "look up" operation with SPN filtering and tone mapping. Taken from Li *et at.* [34].

same as the complexity of the nonlinear response.

Figure 1.10 summarizes Li *et al.*'s [34] algorithm. Assuming there are n pixels, and that the polynomial degree is q, the number of operations, per frame, required to implement FPN correction are as follows, provided $q \geq 2$: $n(q+2)$ additions; nq multiplications; and $n(q+1)$ bit shifts.

A related problem to FPN correction is SPN filtering. The latter is required to deal with outliers, such as dead pixels, that cannot be properly handled using FPN correction alone. Once again, we leverage a background algorithm described very briefly in the same paper by Li *et al.* [34].

Chapter 2 presents our novel digital circuit methods to realize FPN correction and SPN filtering. In addition to proposing, validating, and evaluating digital circuits for these purposes, we also compare our results to analog, mixed-signal, and digital circuit methods from the literature.

1.2.3 Histogram-Based Tone Mapping

Tone mapping is an essential ISPg operation for an HDR imaging system, such as an imaging system based on log or linlog sensors, when humans are the end users. The problem is that conventional displays are LDR, offering only about two decades of DR. A TMO transforms an HDR image into one suitable for an LDR display [14], while retaining the information richness of the former, such as visible detail at all light levels between dark and bright limits. Another requirement of a TMO, especially desired in HDR video applications, is to avoid flickering and other temporal artefacts.

Ward Larson *et al.* [31] developed an algorithm to map scene luminance to display brightness and called it tone mapping. They employed histogram equalization with ceilings on the bin values, based on a model of human vision, to avoid exaggerating the contrast. Our TMO is based on Li *et al.*'s [35] work, which in turn is based on Ward Larson *et al.*'s work but with ceilings computed to limit the visibility of temporal noise and residual FPN, of a nonlinear CIS, after FPN correction. The required parameters, for Li *et al.*'s TMO, are already determined during the FPN calibration process.

Figure 1.11 illustrates, at a high level, the background algorithm of the tone mapping. It divides into two parts: one executes a mapping function; and the other updates the mapping function from the histogram of a frame. During real-time operation, the histogram of each frame is collected, and the single-frame *scene* histogram is computed. A first-order low-pass filter (LPF) is applied to the scene histogram, which defines a single-frame TMO, to get the *perceived* histogram, which defines a multi-frame TMO.

If each frame in a video is treated independently, image noise and abrupt scene changes may result in visual artefacts. Considering the temporal adaptation process of the human eye, Li *et al.* included the LPF to attenuate the impact of noise and abrupt changes in the histograms. After FPN calibration, noise-and-distortion ceilings for an image sensor are determined. These ceilings are applied to perceived histograms, creating what are called *modified* histograms. Thereafter, a mapping function for the video is built using

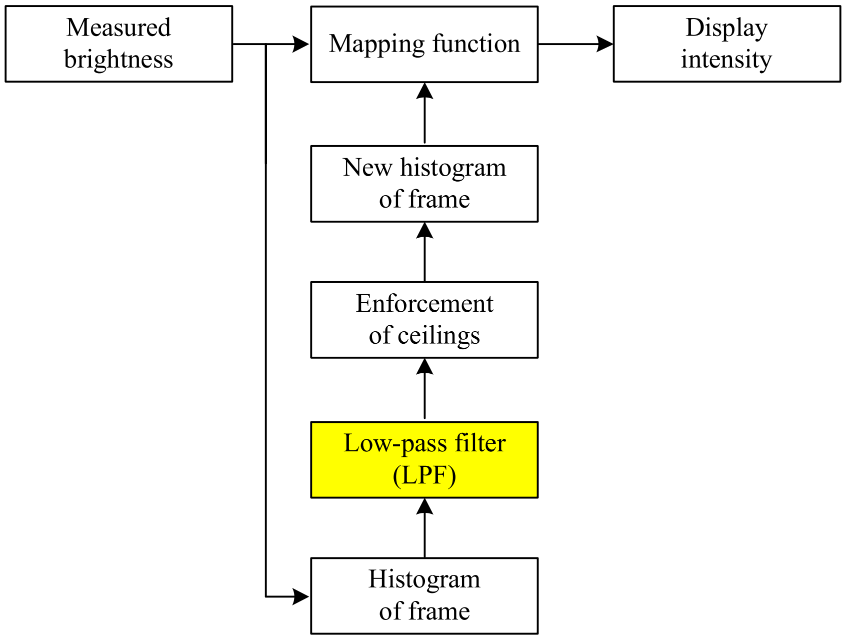


Figure 1.11: Signal flow of the background TMO algorithm. Noise-and-distortion ceilings are computed beforehand for a specific image sensor. Idealized responses are mapped to display intensities, and the mapping function is also updated, each frame. Taken from Li *et al.* [35].

a cumulative distribution function (CDF) computed from the modified histogram. Finally, idealized responses, i.e., sensor responses after FPN correction, are mapped to display intensities via the mapping function.

Li *et al.*'s work was developed and validated using a software algorithm, running on a CPU in soft real time. Chapter 3 presents the design, validation, and evaluation of efficient pipelined circuits that realize the TMO in hard real time. A circuit implementation poses new challenges. For example, we modify the background algorithm because otherwise it would not scale to HD resolutions at standard video rates, within the constraint of low-cost FPGA devices. The chapter also compares our results to relevant hardware solutions that have been described in the literature.

Chapter 2

FPN Correction and SPN Filtering

Nonlinear CMOS image sensors (CISs), such as logarithmic (log) and linear-logarithmic (linlog) sensors, achieve high/wide dynamic ranges (DRs) 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 chapter 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 (LEs), memory bits, and power consumption. The scalability of the methods to full high definition (HD) video processing is also demonstrated. In particular, FPN correction and SPN filtering of over 140 megapixels per second is feasible, in hard real time, irrespective of the degree of nonlinearity.

2.1 Introduction

In a review paper, Kim [28] of Samsung has explained the importance of high dynamic range (HDR) imaging and examined several wide DR (WDR) technologies, based on CISs, to achieve it. While "dual-exposed or multiframecapturing 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 linlog sensor, a nonlinear CIS with a response that transitions from linear, in dim lighting, to log, in bright lighting. Whereas FPN does degrade the raw image quality of linear sensors [15], the degradation is worse with log and linlog sensors due to their nonlinearity [11], [22]. 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 chapter. Because perfect uniformity is impossible in complementary metal-oxide-semiconductor (CMOS) fabrication, FPN is caused by time-independent sensor variations from pixel to pixel [34]. 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 [15], 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 [43], i.e., an image signal processor (ISPr).

As for circuit-based nonlinear FPN correction, the literature has addressed: analog circuits to correct offset variation only for both log and linlog sensors [26], [39]; mixed-signal circuits to correct both offset and gain variation of linlog sensors [50]; and digital circuits to correct offset, gain, and bias variation of log sensors [20]. Some authors are motivated to avoid calibration or use self-calibration [26], [39], [50]. Other authors, like us, are motivated to achieve the highest image quality possible and so, as with linear sensors, adopt calibration [20].

This work contributes, validates, and evaluates a method to generate digital circuits, suitable for ISPr 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 ISPr integration, to filter 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 ISPr with linear sensors [43]. In contrast, the literature on integrated circuits for log and linlog sensors, including the above citations [20], [26], [39], [50], 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 [34]. In Section 2.2, we summarize the background algorithms and present the novel methods under distinct subheadings.

As described in Section 2.3, the proposed methods are validated and evaluated by generating and simulating very-high-speed integrated circuits (VHSICs), using FPGA tools, from Xilinx and Altera, and VHSIC hardware description language (VHDL) designs. Section 2.3 also elaborates on the novelty and significance of this work, both of which have been introduced above.

Section 2.4 concludes the chapter by summarizing its motivation, background, methods, results, and discussion.

2.2 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.

2.2.1 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 Fig. 2.1. 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

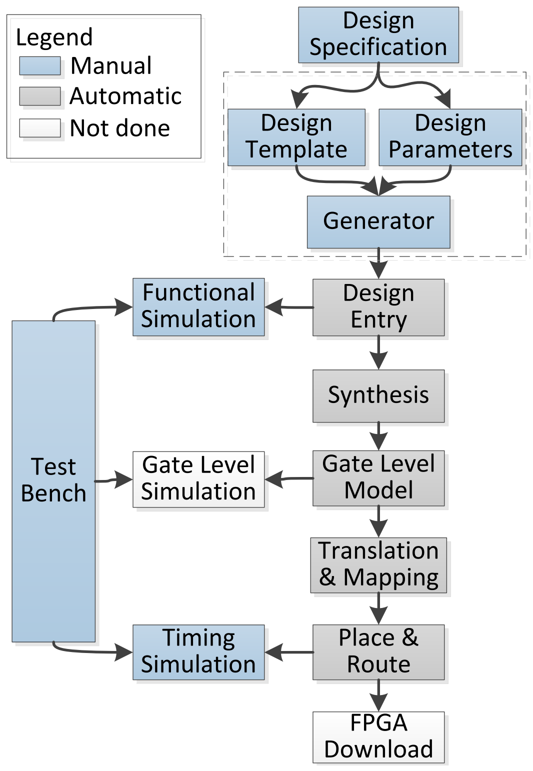


Figure 2.1: Generic FPGA design flow adopted here. The dashed box shows aspects added to a standard design flow. Functional Simulation suffices to demonstrate validity and estimate complexity. Timing Simulation suffices to evaluate max frequency, of valid operation, and power consumption.

allocated. Finally, Place & Route, which enables Timing Simulation, selects and configures resources physically available on a chosen FPGA device.

As shown in Fig. 2.1, 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.

2.2.2 FPN Correction

In this chapter, 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.

Background

To create an effective and efficient algorithm, which Li *et al.* previously published [34], 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 [37], Fig. 2.2 has been newly prepared to illustrate this concept.

Calling scene luminance x and pixel response y, in Fig. 2.2, 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 are not shown, these two observations remain true. Because the 'knee point', called the bias [22], 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 [22]:

$$y_j = a_j + b_j \ln(c_j + x_j) + \epsilon_j, \tag{2.1}$$

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

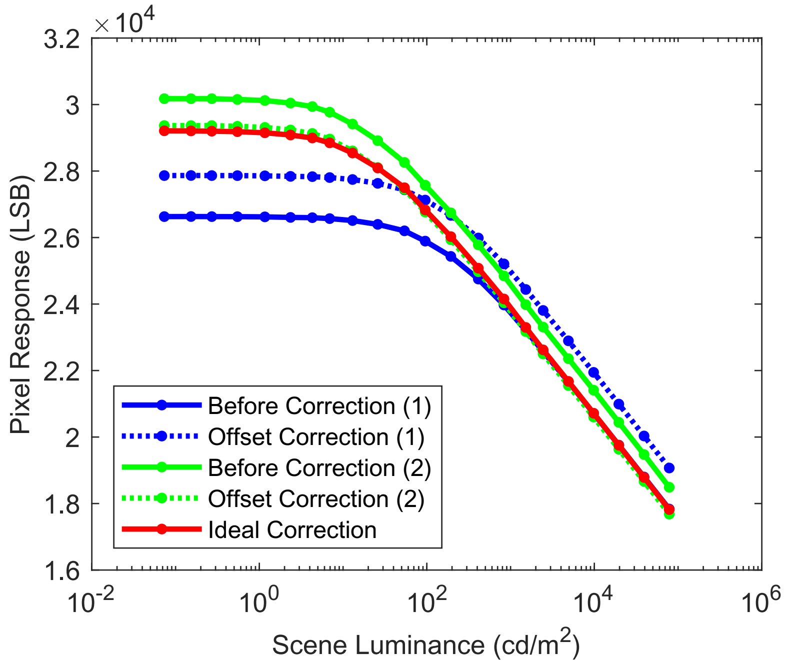


Figure 2.2: FPN correction need not invert nonlinear responses. Two pixels are shown from a log CIS having 48 × 64 pixels. Offset correction, which is inadequate, simply adds a pixel-dependent number to each response. Ideal correction is well approximated, in this case, using cubic polynomials.

intensity, FPN correction is achieved as follows:

$$\hat{x}_j = \exp((y_j - \hat{a}_j)/\hat{b}_j) - \hat{c}_j, \tag{2.2}$$

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. Thirdly, modeling a linlog sensor in a similar way to (2.1) proves complicated [11].

An alternative, i.e., our inverse polynomial regression (IPR) approach [34], uses the following generic model:

$$y_j = f_j(x_j) + \epsilon_j, \tag{2.3}$$

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:

$$\hat{y}_j = y_j + \hat{b}_{j0} + y_j(\hat{b}_{j1} + y_j(\hat{b}_{j2} \cdots + y_j(\hat{b}_{jq}))), \tag{2.4}$$

where $\hat{b}_{jk}$ are the coefficients of degree q polynomials, with $0 \leq k \leq q$, 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. 2.2, over all $3,072$ pixels [34]. The ideal response, which remains highly nonlinear over the WDR, is shown in Fig. 2.2.

We also developed a fixed-point version of FPN correction [34]. 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:

$$B_{jk} = \text{round}(2^{-s_k}\hat{b}_{jk}), \tag{2.5}$$

$$|B_{jk}| < 2^{t_k-1}. \tag{2.6}$$

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

$$t = t_0 + t_1 + t_2 \cdots + t_q. \tag{2.7}$$

The binary-point shifting of (2.5), to scale coefficients before rounding, means that the FPN correction of (2.4) must be amended, to undo the binary-point shifting, as follows:

$$\begin{aligned} Y_j = y_j + 2^{s_0}(B_{j0} + 2^{s_1-s_0}y_j(B_{j1} + 2^{s_2-s_1}y_j(B_{j2} \\ \cdots + 2^{s_q-s_{q-1}}y_j(B_{jq})))), \end{aligned} \tag{2.8}$$

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 [34].

To complete the explanation, two additional details are needed. First, instead of y_j in the right hand side (RHS) of (2.8), except the leftmost y_j, we use the following:

$$y_j' = y_j - y_0, \tag{2.9}$$

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 (2.9) 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 [34].

The final detail concerns binary-point, or bit, shifts in the RHS of (2.8). Because s_0 is expected to be non-negative in an optimal configuration, it entails a left shift. The left shift of an integer stays an integer. On the other hand, $s_k - s_{k-1}$ for $1 \leq k \leq q$ 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 (2.8), except the leftmost one. This turns the fixed-point correction into a more efficient integer correction [34].

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 [34], 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 just for one nonlinear image sensor but for a wide variety of them, we use our generic design flow, shown in Fig. 2.1, 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 $0 \leq k \leq q$; the number of pixels, n; the polynomial coefficients, B_{jk}, where $0 \leq j \leq n-1$ assuming 0-based indexing, what VHDL uses, instead of 1-based indexing, i.e., $1 \leq j \leq n$, 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 B_j, where t in (2.7) is constant. As this is done once, it is done in MATLAB after calibration. The resulting tn-bit data is not

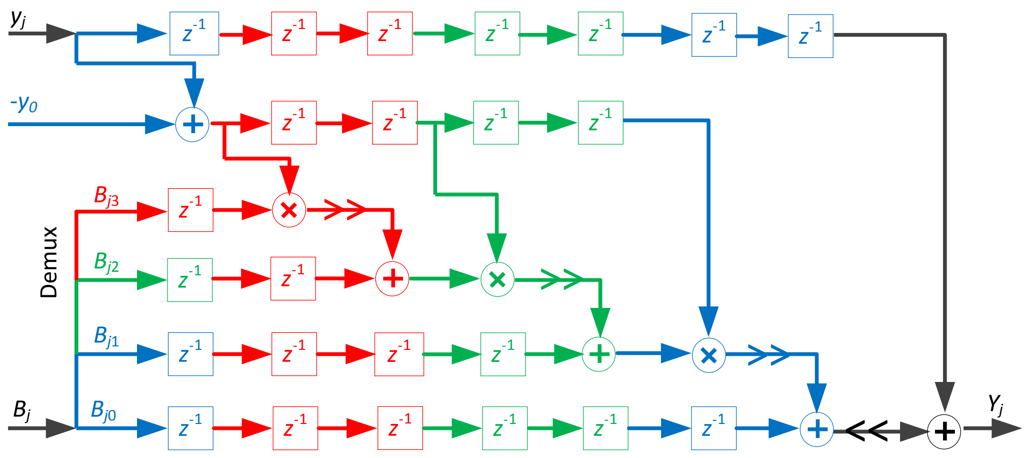


Figure 2.3: FPN correction using a recursive pipeline circuit. This schematic shows: offset correction in black; offset and gain, i.e., linear, correction in blue and black; quadratic correction in green, blue, and black; and cubic correction in red, green, blue, and black. Bus operations $\gg$ and $\ll$ represent bit shifts.

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 [20] also externalized some of the coefficients.

Fig. 2.3 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 design, of a clock signal.

In Fig. 2.3, addition ($+$), subtraction ($-$), multiplication ($\times$), 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. 2.3, 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 at each clock cycle, IPR(q) correction has a latency of $2(q+1)$ clock cycles, for $q > 0$. Because offset correction does not require (2.9), 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. 2.3, 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 $0 \leq k \leq q$. 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. 2.3, 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:

$$Y \gg |\Delta s| = \lfloor 2^{\Delta s} Y \rfloor, \tag{2.10}$$

where Y is the digital signal on the incoming bus.

Although convenient for a circuit implementation, using (2.10) 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:

$$Y \gg |\Delta s| = \lfloor 2^{\Delta s} Y \rfloor + c_{\text{out}}, \tag{2.11}$$

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:

$$c_{\text{out}} = Y_{v-1} \wedge (\bar{Y}_{u-1} \vee d_{\text{rem}}), \tag{2.12}$$

$$d_{\text{rem}} = Y_{v-2} \vee Y_{v-3} \cdots \vee Y_0, \tag{2.13}$$

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 $^-$ are logical AND, OR, and NOT operators, respectively.

Addition of the carry out in (2.11), to correct each right shift, may actually be integrated into a following adder. In Fig. 2.3, 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. 2.3 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 de-multiplexing, signals

B_{jk} are t_k bits wide, where $0 \leq k \leq q$. Delays do not change the 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.

2.2.3 SPN Filtering

Whereas Li *et al.* have previously disclosed the SPN filtering approach [34], 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.

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. 2.2, 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 [37], Fig. 2.4 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

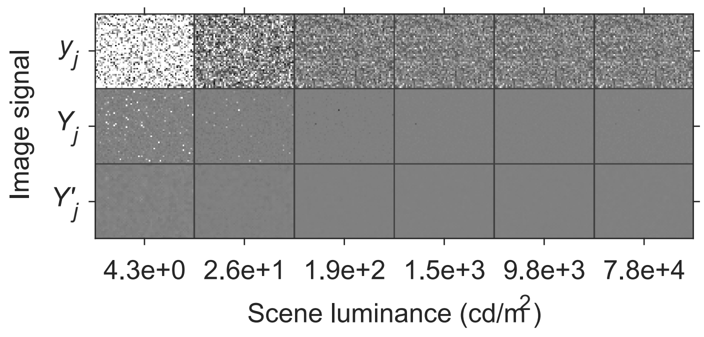


Figure 2.4: FPN correction and SPN filtering are complementary. To deal with noise in raw images (top row), from a log sensor over a WDR (left to right), both correction (middle row) and filtering (bottom row) are needed.

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 colour filter arrays for now.

Fig. 2.5 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 neighbours are used. For boundary pixels, at the borders and corners, the pixel and two neighbours are used. When we developed our software algorithm [34], we were thinking ahead to a circuit method. With odd-size windows, only sorting is needed to compute medians; averaging is not needed.

Method

Figure 2.6 presents a schematic of our SPN filter. We use our generic design flow, of Fig. 2.1, 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

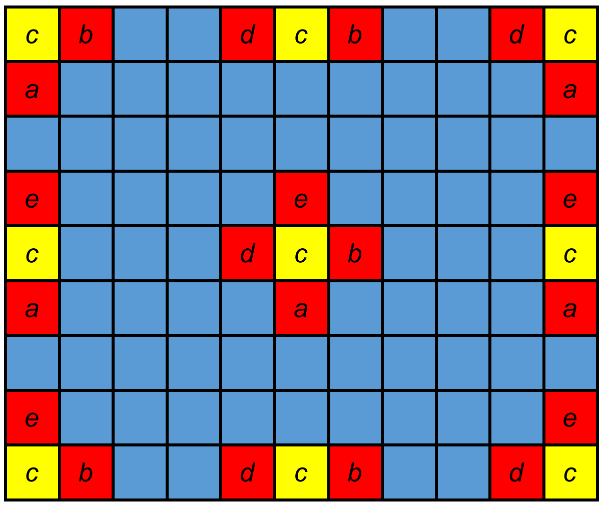


Figure 2.5: SPN filtering employs windows that vary with pixel. In each window, pixels are coloured red except for the center pixel, which is coloured yellow. The center pixel is replaced with the median value of its window.

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 a to e in Fig. 2.6, 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. 2.5. 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 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

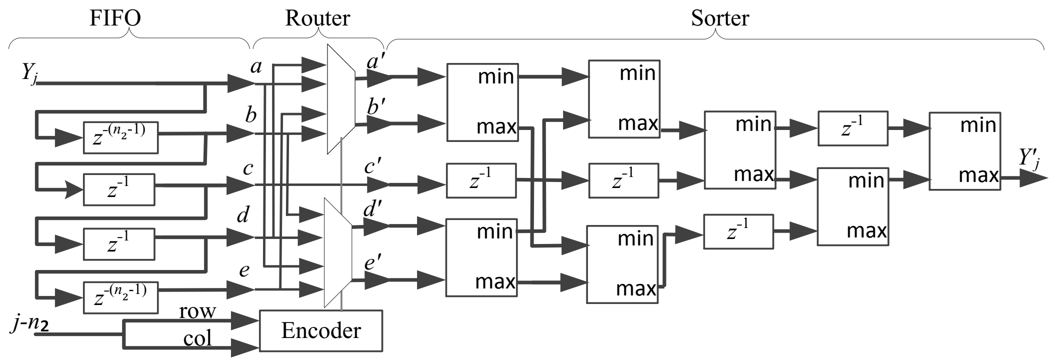


Figure 2.6: SPN filtering using a three-stage pipeline circuit. The FIFO buffers two rows of pixel values. The sorter computes the median of five pixel values. The 'no-delay' router is needed to compute medians for three-pixel windows at the image corners and borders. Table 2.1 elaborates on the router logic.

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. 2.6, is a router. The router enables median filtering of three-pixel windows at the boundary, as shown in Fig. 2.5, 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 2.1 elaborates on the router. The position of the center pixel, denoted c in Figs. 2.5 and 2.6, is given by its address j. An encoder converts the address, which is $t_{\text{row}} + t_{\text{col}}$ 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

Table 2.1: 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. 2.5 and 2.6 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

equals Y_j in Fig. 2.6, 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 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. 2.3, 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 $j - n_2$ in Fig. 2.6, 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. 2.6 and Table 2.1, 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.

2.3 Results and Discussion

Section 2.2 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.

2.3.1 Test Benches

Using the design flow shown in Fig. 2.1, 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 families still in production, at the time of this work, are the Xilinx Spartan-6 [57] and the Altera Cyclone III [3]. 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 a degree of 3 suffices for a log sensor [34], 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 the number of pixels, division into rows and columns, and frame rate. Power consumption depends on clock frequency, which equals the number of pixels times the frame rate, in frames

Table 2.2: 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

per second (fps).

Table 2.2 lists three popular video formats, two of which are HD formats, where the 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 [37]. 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. 2.1, Functional Simulation after Design Entry suffices for validation but Place & Route is needed to 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 2.2, is below the max frequency. Power consumption is evaluated only for supported video formats.

2.3.2 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.

Validation

Illustrated in Fig. 2.7, the initial validation of the generated FPN correction circuit was done manually for a 4×4 pixel subset of the TTVGA format,

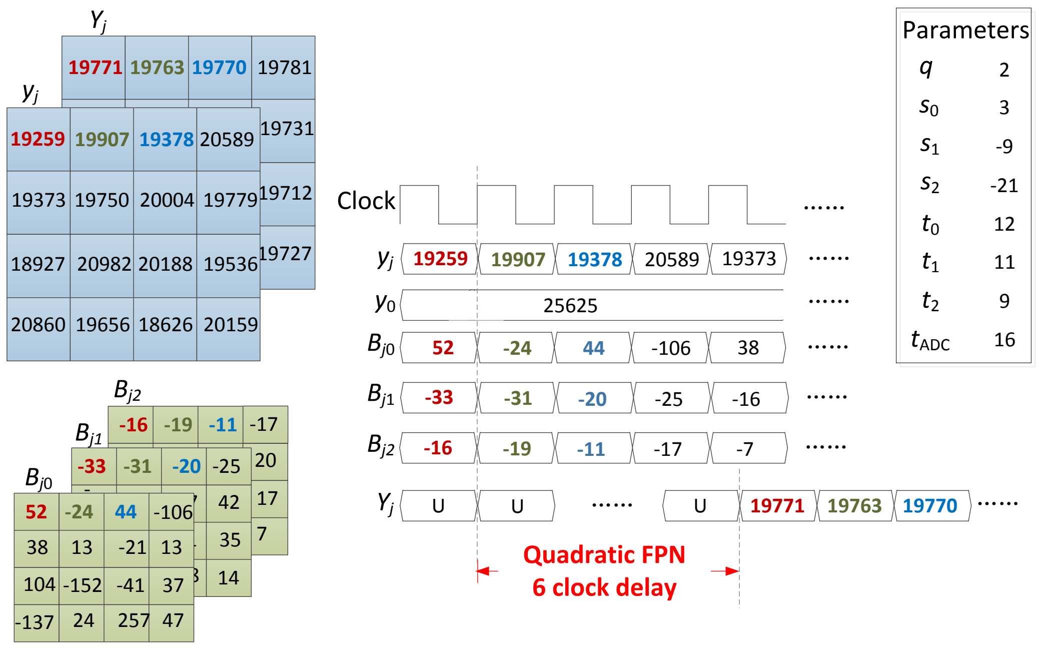


Figure 2.7: Initial validation of a generated FPN correction circuit. Input, output, and intermediate signals are shown for a small-format test case (Fig. 2.3 elaborates on the signals). Larger-format test cases were validated by automatic comparison of circuit and software outputs, given the same inputs.

using experimental data from a log sensor [37]. 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. 2.7, 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:

$$y_1' = 19259 - 25625 = -6366 \tag{2.14}$$

$$\begin{aligned} Y_1 &= 19259 + 2^3(52 + [2^{-9-3}(-6366)(-33 \\ &\quad \cdots + [2^{-21+9}(-6366)(-16)])]) \\ &= 19771, \end{aligned} \tag{2.15}$$

where square brackets indicate rounding.

As shown in Fig. 2.7, the correct output appears with a latency of 6, i.e.,

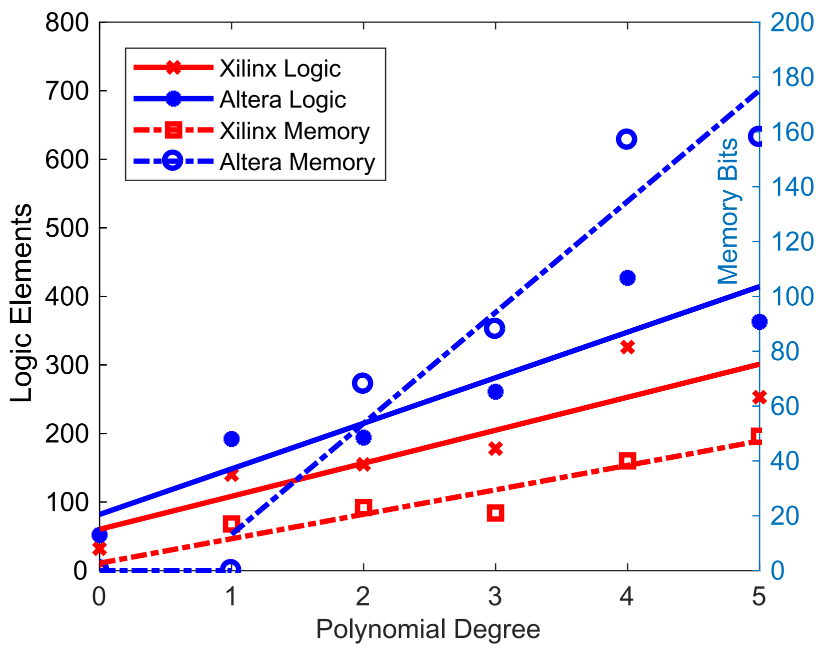


Figure 2.8: Complexity of FPN correction vs. polynomial degree. Required LEs and bits depend linearly on degree, Altera memory for offset correction aside. Even so, these requirements use a tiny fraction of available resources.

$2(q+1)$, 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.

Complexity

Given functional correctness, we then analyzed the complexity of generated circuits, illustrated in Fig. 2.8, 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 [34].

In Fig. 2.8, actual data is shown using symbols, for each FPGA device, and trends are shown using best-fit lines. Complexity is measured in LEs and memory bits, on the left and right y-axes, respectively. The LEs required grows roughly linearly with degree (R^2, the coefficient of determination, equals 80% and 86% with Xilinx and Altera, respectively). Outliers aside, i.e., degree zero with Altera, the bits required also grows 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 ISPr 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.

Frequency

Next, we determined the maximum clock frequency at which functional correctness is maintained. These results are shown in Fig. 2.9 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. 2.3, 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. 2.9.

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. 2.9, indicate the frequencies, listed in Table 2.2, required to support the FHD and 4KUHD formats.

Power

Our final results, shown in Fig. 2.10, concern power consumption. Because this depends on clock frequency, we use the corresponding frequencies, listed in Table 2.2, for the supported video formats. We also vary the polynomial

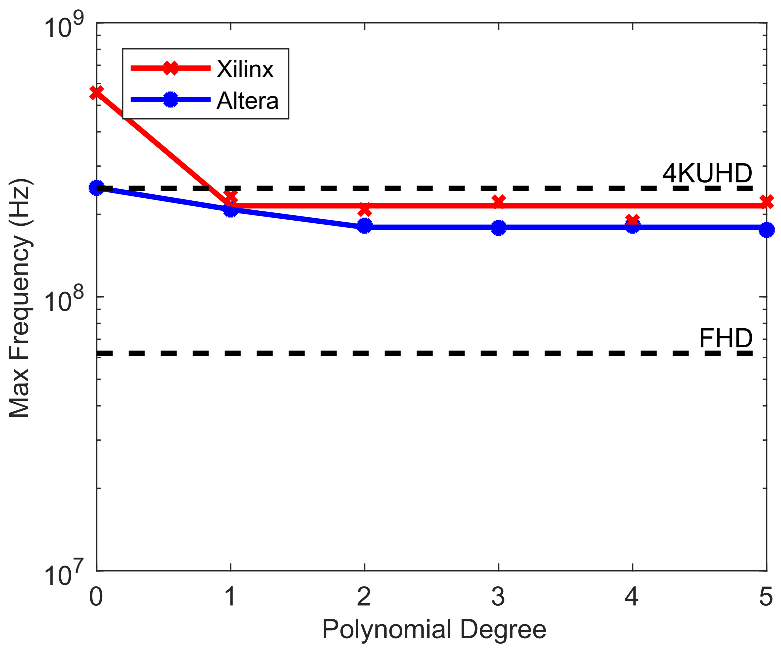


Figure 2.9: Max frequency of FPN correction vs. polynomial degree. Except at the lowest degrees, max frequencies are essentially independent of polynomial degrees. FHD and simpler video formats are readily supported.

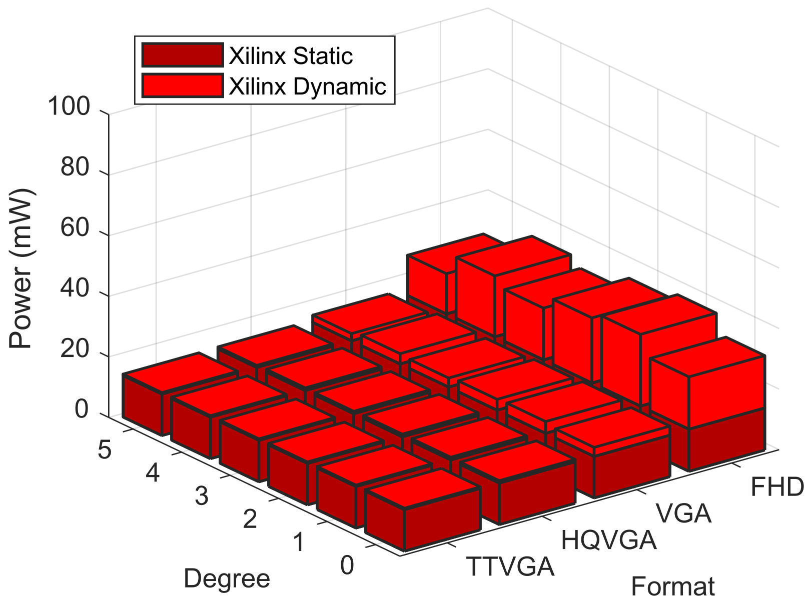
degree, as before. Other parameters were unchanged.

In Fig. 2.10, 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. 2.10, 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.

2.3.3 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 brief.



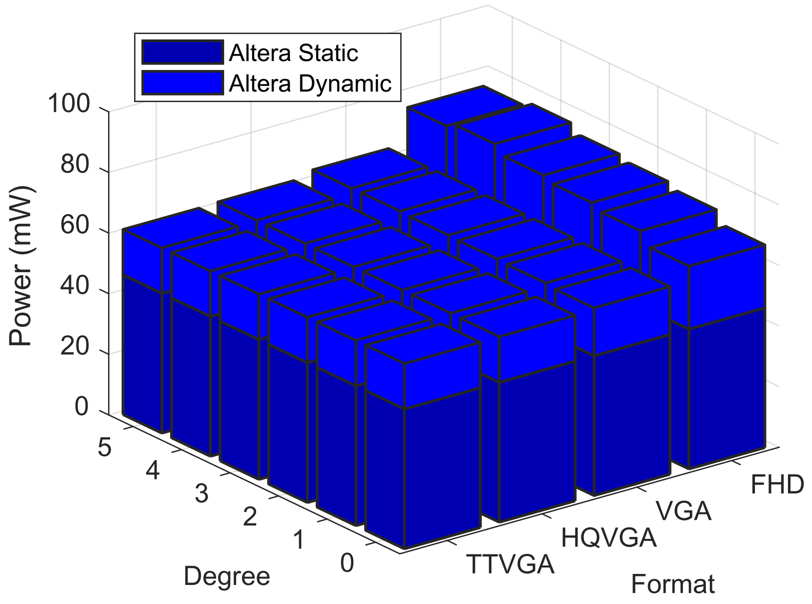


Figure 2.10: Power consumption of FPN correction vs. parameters. Except for the FHD video format, where it increases a little, dynamic power is nearly constant. Compared to static power, dynamic power is generally low.

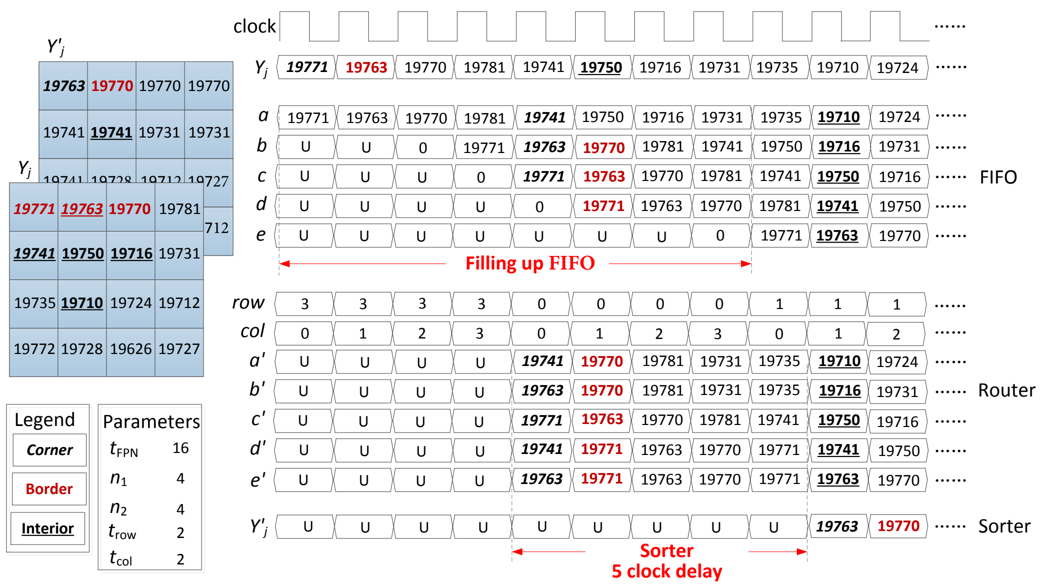


Figure 2.11: Initial validation of a generated SPN filtering circuit. Input, output, and intermediate signals are shown for a small-format test case (Fig. 2.6 elaborates on the signals). Larger-format test cases were validated by automatic comparison of circuit and software outputs, given the same inputs.

Validation

Illustrated in Fig. 2.11, 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. 2.6. It is straightforward to show that all waveforms in Fig. 2.11 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. 2.5. There were zero bit errors.

Complexity

Given functional correctness, we then analyzed the complexity of the generated circuit, illustrated in Fig. 2.12, versus the number of pixels, n. Each number,

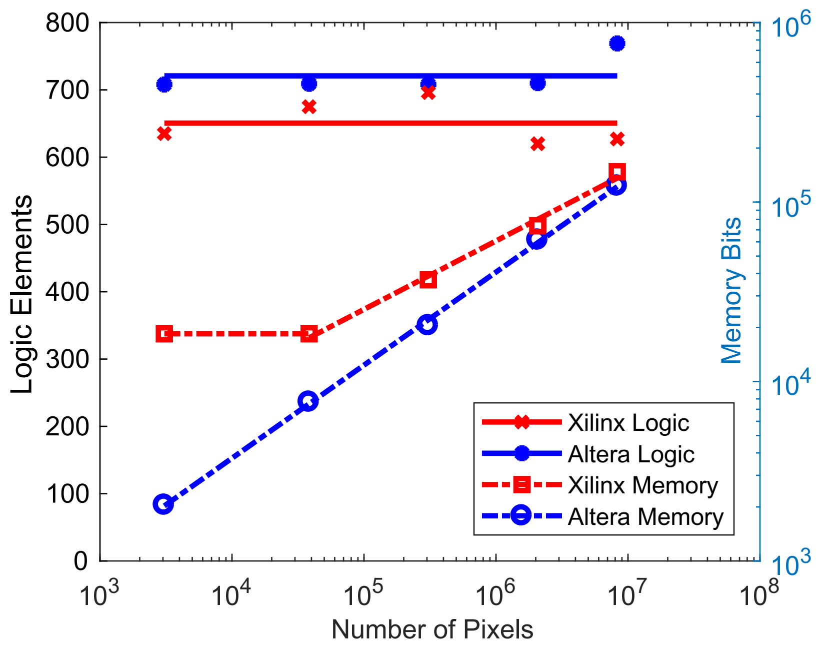


Figure 2.12: Complexity of SPN filtering vs. number of pixels. Required LEs are approximately constant and use a fraction of available resources. Required bits grow with the number of pixels but remain well below capacities.

n, and its breakdown into rows, n_1, and columns, n_2, is taken from Table 2.2. 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. 2.12. However, there is a linear relationship, on a log-log scale, between the bits required and the number of pixels (R^2 equals 99% and 100% with Xilinx and Altera, respectively), excluding one outlier. The memory capacity required with Altera exactly equals the minimum bits, i.e., $2n_2t_{\text{FPN}}$, needed to implement the FIFO stage shown in Fig. 2.6.

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.

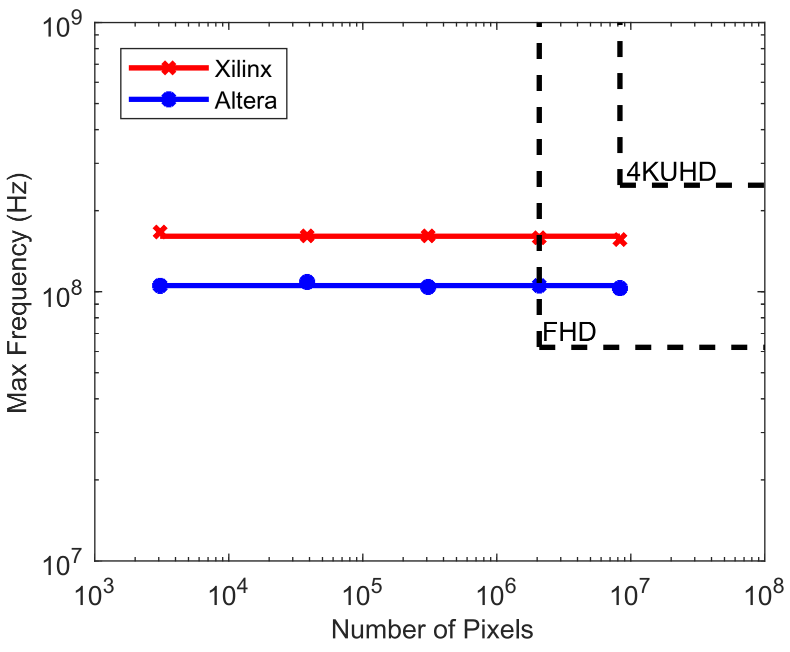


Figure 2.13: Max frequency of SPN filtering vs. number of pixels. The max frequency is essentially independent of the number of pixels. FHD and simpler video formats, listed in Table 2.2, are readily supported.

Frequency

Next, we determined the maximum clock frequency at which functional correctness is maintained. These results are shown in Fig. 2.13 versus number of pixels, as before. Other parameters are the same as with Fig. 2.12.

In Fig. 2.13, the max frequency is nearly constant in both FPGAs. The fact that the LEs required are roughly constant, in Fig. 2.12, 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. 2.6.

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. 2.13, indicate the numbers of pixels and clock frequencies, listed in Table 2.2, required to support the FHD and 4KUHD formats.

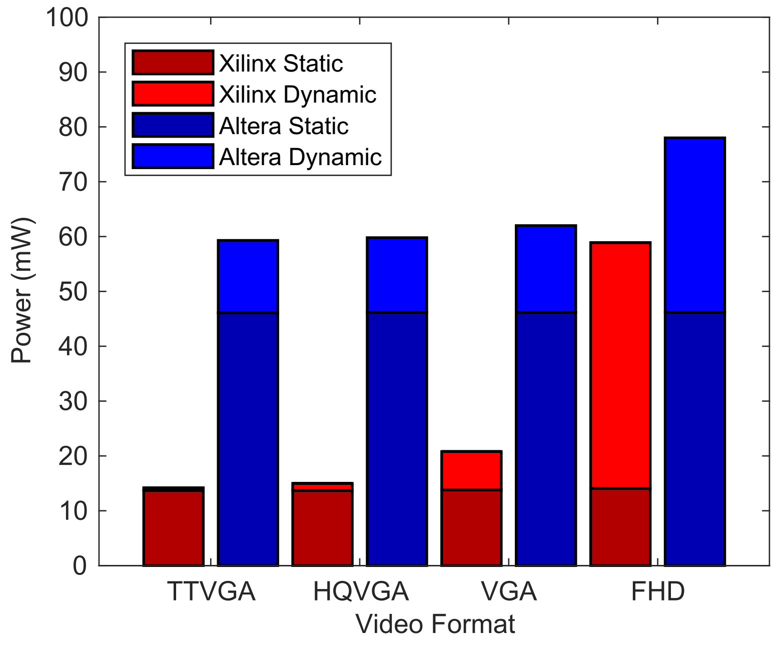


Figure 2.14: Power consumption of SPN filtering vs. video format. Static power is a constant and significant part of total power. Except for a jump at the FHD video format, dynamic power is approximately constant.

Power

Our final results, shown in Fig. 2.14, concern the power consumption for the supported video formats. We use the numbers of pixels and clock frequencies listed in Table 2.2. Other parameters are the same as with Fig. 2.12.

Except for the FHD case, as shown in Fig. 2.14, 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.

2.3.4 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.

Table 2.3: Specifications of the 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 (LEs)	Memory (bits)	Max Freq.	Static P.	Dynam. P.
FPN	Xilinx XC6SLX4	178 (2.06%)	21 (0.01%)	222.2 MHz	13.9 mW	17.1 mW
Correction	Altera EP3C5	261 (2.53%)	88 (0.02%)	178.6 MHz	46.1 mW	27.9 mW
SPN	Xilinx XC6SLX4	620 (7.17%)	73,737 (24.7%)	158.7 MHz	14.1 mW	44.8 mW
Filtering	Altera EP3C5	710 (6.88%)	61,440 (14.5%)	105.5 MHz	46.2 mW	31.8 mW
Complete	Xilinx XC6SLX4	817 (9.45%)	73,767 (24.8%)	140.8 MHz	14.2 mW	48.4 mW
ISPr	Altera EP3C5	972 (9.42%)	61,528 (14.5%)	108.7 MHz	46.2 mW	42.6 mW

Specifications

Table 2.3 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 2.3.2 and 2.3.3. These circuits were also combined into one ISPr 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 ISPr 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 ISPr 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. 2.6, 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.

Analog Competitor

De Moraes Cruz *et al.* [39] 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 [34], [37], we demonstrated a peak SNDR (PSNDR) of 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 2.3, 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.

Mixed-Signal Competitor

To correct offset and gain variation in linlog sensors, Storm *et al.* [50] 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 [34], [37] that image quality is limited with log sensors unless higher-order FPN correction is employed. We calculate the PSNDR [48] of Storm *et al.*'s imaging system, using data that they provided, 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.

Digital Competitors

Hoefflinger [20] proposed a digital circuit to correct OGB variation in log sensors. After FPN calibration, by approximate curve fitting of the model given in (2.1), 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 [34], which we also developed, that is not specific either to log sensors or (2.1). 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 [11] have developed a model, similar to but more complex than (2.1), 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 [43] addresses "stuck pixels" alongside "pixel non-uniformity," i.e., FPN, advising to "replace with filtered values." We show, in Fig. 2.4, that they are complementary, address both, and evaluate joint complexity, max frequency, and power consumption.

Latha and Sasikumar [32] 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 2.3 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.

2.4 Conclusion

Kim [28] 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.* [34], i.e., our group's 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 chapter. Both works use experimental data from a log sensor [37] 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 group's 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 validated and evaluated 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 shown to be 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 chapter 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.

Chapter 3

Histogram-Based Tone Mapping

Tone mapping is extensively researched to address the issue of displaying high dynamic range (HDR) scenes on low dynamic range (LDR) displays. Even though several tone mapping operators (TMOs) exist, not all are designed for hard real time. The operator has to be capable of scaling up the spatial resolution without compromising the frame rate. The implementation of a TMO should also be simple enough to embed in low-cost platforms for imaging systems. A computationally efficient, and well accepted, class of TMOs are global ones based on histograms. This work presents a method to implement TMOs that use histograms. Our method is suitable for low-cost field-programmable gate arrays (FPGAs), using simple components such as adders, multipliers, and random-access memorys (RAMs), and is particularly suited for a non-linear CMOS image sensor (CIS) operating continuously in hard real time. We develop a fixed-point design, validated in bit true fashion using Xilinx and Altera tools with a background algorithm implemented using MATLAB. Our generic design uses pipelined circuits and operates with low latency. The use of a hardware description language (HDL) to model our circuits guarantees portability and modularity. Moreover, the complete TMO is generated from design parameters and a design template. The architecture is robust and scales well from kilopixel to megapixel formats. We achieve 30 frames per second (fps) at high definition (HD) resolutions, while occupying only a small fraction of the available logic elements (LEs) in low-cost FPGA devices.

3.1 Introduction

Advanced CISs, particularly nonlinear CISs, are able to capture images with high/wide dynamic ranges (DRs). High/wide DR displays have emerged in the market, but they will not reach the same range of the real world, and will not displace standard displays due to their price and power consumption. Kim [28] also declares the relevance of high/wide DR to continue the development and improvement of embedded cameras. All of these sensors need a TMO to adapt the high/wide DR image to a low DR display.

Though TMOs have been extensively researched, they are often not designed for video applications in real time. This chapter, on the other hand, is concerned primarily with hard real time operation. Because of the lack of high/wide DR video sources, research into TMOs is often validated only with still images, which may not be valid for videos due to temporal artefacts, as pointed out by Eilertsen *et al.* [14].

Our focus is a real-time TMO for videos to be embedded in a mobile device. There are two main platforms for this: graphics processing units (GPUs) [1], [2], [6]; and FPGAs [19], [41]. Urena *et al.* [54] deployed their TMO in both platforms and concluded that their FPGA implementation achieves higher frame rate and lower power consumption, while their GPU implementation is more precise. Lo *et al.* [36] also deployed their algorithm on both platforms, achieving lower power consumption with the FPGA solution. Because we are proposing a methodology of digital circuits, coded using a HDL, FPGAs are a better platform also to implement and validate our method.

There are two kinds of TMOs, which are mutually exclusive: local and global. Reinhard *et al.* [44] defined global TMOs as the same operation for all pixels, treating each pixel independently of its location. Local TMOs consider the pixel location, and the operator may vary from pixel to pixel. One reason that we use global is because it is considered more computationally efficient.

Ofili *et al.* [41] employed a TMO based on Glozman *et al.*'s operator, which is based on an inverse exponential function. The operator was modified by introducing an adaptation factor that reshapes the curve according to the image mean and the max luminance value. The final operator was translated to a hardware-friendly FPGA design, and deployed on Stratix II and Cyclone III devices from Altera. This digital design, using 68046 bits of memory, achieves a high frame rate (up to 126 fps) and a spatial resolution of 1024×768 pixels, but no video tests were realized and halo artefacts were reported.

Urena *et al.* [54] presented a real-time TMO with GPU and FPGA implementations. It is a local operator, according to the mutually-exclusive definitions provided here. However, a global enhancement is realized via histogram equalization over V from the HSV colour space, derived from the original RGB colour space. The implementation of histogram equalization is based on the work of Reza [45] and has some similarities with the work presented here, also based on histograms. Reza delivers a real-time design, to be implemented in FPGAs or an application-specific integrated circuit (ASIC), for adaptive histogram equalization of images. Our platform includes temporal adaptation, is validated for multiple FPGAs devices, and offers a more generic approach to histogram-based tone mapping. Urena *et al.* also do not address temporal adaptation, nor how their design handles videos. The results also had only 64 bins for the histogram.

Even if global is a better choice for embedded systems, many researchers choose local TMOs claiming it delivers better results [1], [19], [41], [54]. Eilertsen *et al.* compared TMOs for high/wide DR videos and their survey found that global TMOs are the most preferred. Local TMOs present more temporal artefacts than global, making them less favoured when compared side-by-side.

Popovic *et al.* [42] developed a hardware version of the Drago operator [17], which uses a logarithmic (log) mapping function with a variable base. They benchmarked the chosen operator to find a design bottleneck at the log function that used external memory, which led them to implement a "hardware-only system" and approximation of the log calculation. Their architecture is pipelined and uses Taylor and Chebyshev polynomials directly implemented in a high-end FPGA. The design is capable of processing full HD (FHD) at 60 fps, but they do not address the suitability of their approach for high/wide DR video, which means it is at risk of temporal adaptation artefacts.

Amongst the possible global TMOs, there are the ones based on histograms. In fact, these are the operators that showed better results in Eilertsen *et al.*'s research. Of the top three operators, two are based on histograms. Indeed, there are multiple tone mapping algorithms based on histograms [13], [21], [31], which could leverage the results of this research.

This work contributes, validates, and evaluates digital circuits to implement a global TMO based on histograms, suitable for image signal processor (ISPr) integration in hard real time. We propose a hard real-time pipelined architecture to be implemented in low-cost FPGAs and capable of processing HD videos from a nonlinear CIS. Our novel circuits are based on Li *et al.*'s [35] background algorithm. For efficient continuous operation in hard real time, we employ components with low complexity, such as adders, multipliers, bit shifts, and registers. We also employ block RAMs (BRAMs).

The FPGA design flow for our proposed digital circuits is based on our previously published work [40], which introduced a methodology of circuit generation from design parameters and a design template.

The efficient pipelined circuits we are proposing here can be applied to any tone mapping algorithm that requires histogram computation [1], [24], [27], [31], [35], [55], especially ones based on histogram equalization. The proposed circuits facilitate operation in hard real time. We have chosen low-cost FPGAs as the target platform but the circuits could be readily implemented in an ASIC.

3.2 Background and Methods

The proposed circuits leverage an algorithm, by Li *et al.* [35], for histogram-based tone mapping of nonlinear complementary metal-oxide-semiconductor (CMOS) image sensors. Though we made changes to the background algorithm, while developing the circuit methods, we distinguish the former from the latter, in this section, to emphasize the novelty of this work.

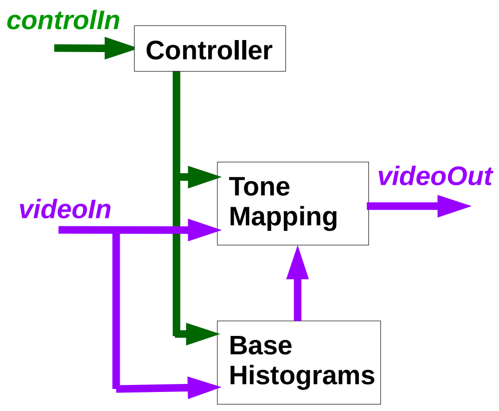


Figure 3.1: Top-level circuit design of histogram-based tone mapping. The design is composed of two modules and a controller. Data and control buses are shown in purple and green, respectively. Thick lines and bold fonts identify multi-bit buses. There are two parallel data paths. One updates the base histograms while the other uses them to map tones.

3.2.1 Design Overview

To achieve our goals, we used a hierarchical and modular circuit design approach. The topmost level is illustrated in Fig. 3.1. Encapsulated details of the main modules are summarized in the ensuing sections.

Table 3.1 lists the dual names of key signals, such as the video input and output shown in Fig. 3.1. Video signals are streamed because we adopted a pipeline approach, following our previous work on FPN correction and salt-and-pepper noise (SPN) filtering. Pixel values flow one by one, into and out of the top-level circuit, left to right and top to bottom with respect to corresponding images.

Although it is simple, relative to other circuits in Fig. 3.1, the controller serves a critical purpose. Synchronization is important in a pipelined design, especially one with multiple paths operating in parallel. The controller receives two control signals, via the bus called *controlIn*. They are a *reset* signal, which pulses for one clock cycle when the FPGA is ready to process data, and a *startOfFrame* signal, which pulses for one clock cycle in sync with the first pixel of each frame. The controller generates additional control signals required by the other modules.

Table 3.1: Important signals and their corresponding symbols. Whereas the background algorithm is summarized using symbols and equations, the circuit methods are proposed using signals and schematics.

Signal	Symbol	Signal	Symbol
alpha	α'	*histValP*	h'_{p}
beta	β'	*histValS*	h_{s}
cumSum	c_Y	*ratio*	R
gain	A	*toneMap*	T
gainMax	$A_{\max}$	*videoIn*	y_j
gainMin	$A_{\min}$	*videoBin*	y'_j
histBin	y'	*videoMax*	$w_{\max}$
histMax	$h_{\max}$	*videoOut*	w_j
histValM	h_{m}	*videoRef*	w_{ref}

Pipelined circuits operate synchronously according to one or more clock signals. Because the proposed TMO may be implemented, for a wide variety of image sensors, using one clock signal only, the clock is not treated as a control signal here, nor is it illustrated in the circuit schematics. Anyway, in an FPGA, clocks are ideally routed differently from control signals.

Finally, following our previous work [40], we use a generic design flow so that our proposed circuits may be applied to a wide variety of image sensors. This means that we use a general-purpose scripting language, i.e., MATLAB, to generate the VHDL code of a specific circuit design from a set of design parameters, i.e., a text file, and a generic design template, i.e., VHDL code. Thereafter, we perform synthesis, translation-and-mapping, and place-and-route to create firmware suitable for FPGA download.

3.2.2 Base Histograms

As shown in Fig. 3.1, the proposed design has a module to compute base histograms. We first review the background algorithm of these histograms. Next, we introduce the method by which we realized it as a circuit.

Background

The histogram-based algorithm developed by Li *et al.* [35] is a foundation of this work. It computes a sequence of *scene* and *perceived* histograms, which we call the *base* histograms, from a sequence of images, i.e., the video input. We review these base histograms below, including some changes to how they are computed.

Instead of treating each input image as a set of n pixel *stimuli*, i.e., x_j with x in real world units (like cd/m^2) and $1 \leq j \leq n$, we treat each image as a set of n pixel *responses*, i.e., y_j where y is an unsigned integer, from a monotonic image sensor. In another paper, Li *et al.* [34] justified the following model for such a sensor, after FPN correction via a polynomial-based algorithm:

$$y_j = F(x_j) + \epsilon_j, \tag{3.1}$$

where F is a monotonic function and ϵ_j represents temporal noise and residual FPN. Depending on F, (3.1) may represent a linear, log, or linlog sensor.

In a recent publication [40], we proposed circuit methods to implement Li *et al.*'s [34] polynomial-based algorithm for FPN correction. Generated circuits, including a median filter to handle outliers after FPN correction, proved effective and efficient even for FHD video. We treat the video output of that work, called Y'_j there, as the video input of this work, called y_j here.

Returning to Li *et al.*'s [35] histogram-based algorithm for tone mapping, we model each pixel response, y, as the observation of a *discrete* random variable, Y, governed by a probability mass function (PMF), p_Y. In other words, $p_Y(y)$ is the probability that Y equals y.

Unlike Li *et al.*, we assume discreteness because our circuit methods, for FPN correction and SPN filtering, always output unsigned integers. We can estimate the PMF as follows:

$$p_Y(y) \approx \frac{h_{\mathrm{s}}(y)}{n}, \tag{3.2}$$

where $h_{\mathrm{s}}(y)$ is the number of pixels equal to y. If we use bins of uniform width Δy, where $h_{\mathrm{s}}(y)$ is the number of pixels in the bin that contains y, then we get instead:

$$p_Y(y) \approx \frac{h_{\mathrm{s}}(y)}{n\Delta y}. \tag{3.3}$$

As with Li *et al.*, we call h_s the *scene* histogram even though, technically, it is an *image* histogram. Also, Li *et al.* actually define h_s as the histogram of $\ln x$, called the *brightness*; x is called the *luminance*. We note that, with a log or linlog sensor, the response, y, is proportional to the brightness, $\ln x$, over a high/wide DR.

To avoid flickering artefacts in tone-mapped videos, Li *et al.* compute a sequence of *perceived* histograms, h_p, from a sequence of *scene* histograms, h_s. We perform the same computation, substituting y for $\ln x$:

$$h_\mathrm{p}(y)[k] = \alpha\, h_\mathrm{p}(y)[k-1] + \beta\, h_\mathrm{s}(y)[k], \tag{3.4}$$

$$\alpha = e^{\frac{-T}{\tau}}, \tag{3.5}$$

$$\beta = 1 - \alpha, \tag{3.6}$$

where k represents the frame number.

Equation (3.4) represents a simple low-pass filter (LPF) operating on all bins of the histogram. With reference to the works of other authors, Li *et al.* offer some justification for this model, including the 'perceived' adjective. They also recommend a value, 0.4 s, for the time constant, τ. As for T, it simply equals the frame period.

Finally, Li *et al.* observe that whereas the scene histogram is always a set of integers, the perceived histogram is not, according to (3.4)–(3.6). We therefore adopt the integer version of their computation, as follows:

$$h'_\mathrm{p}(y)[k] = \lfloor 2^{-s}(\alpha' h'_\mathrm{p}(y)[k-1] + \beta' h_\mathrm{s}(y)[k]) \rfloor, \tag{3.7}$$

$$\alpha' = \mathrm{round}(2^s \alpha), \tag{3.8}$$

$$\beta' = \mathrm{round}(2^s \beta), \tag{3.9}$$

where s, a positive integer, gives the magnitude of a right bit shift, in (3.7), and left bit shifts, in (3.8) and (3.9). According to Li *et al.*, a suitable value for s is eight.

Unlike Li *et al.*, who use a rounding operation in (3.7), we use a floor operation to ensure the perceived histogram goes to zero, for a bin, if the scene histogram goes to zero. This change was validated after careful testing.

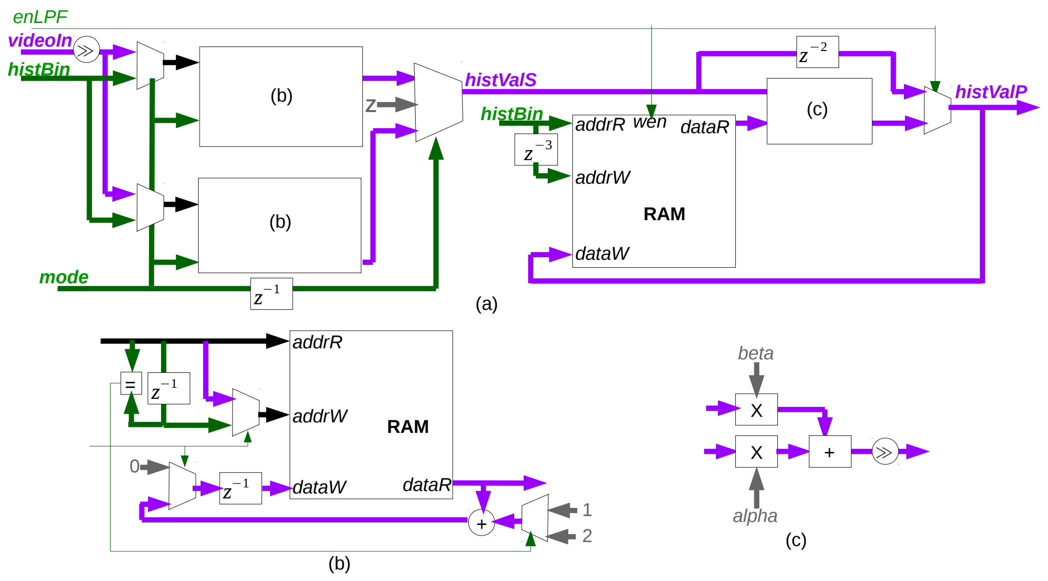


Figure 3.2: Circuit schematics of the base histograms module. Three RAMs are used, two to store the scene histogram and one the perceived histogram. One RAM outputs the scene histogram of the previous frame, which is used to compute the perceived histogram, while the other RAM is used to compute the scene histogram of the current frame. In a pipelined design, like this, careful attention must be paid to synchronization.

Method

Figure 3.2 illustrates our proposed circuit to compute the base histograms efficiently in pipelined fashion. Whereas some design choices may be rationalized, others represent mini-hypotheses that one accepts only after validation and evaluation. We elaborate on the design method below.

In a pipelined circuit design, one choice we may have for an array of data, which changes over time, is whether or not to store it in a RAM. An alternative may be to compute what is needed just when it is needed, for the next computation, in an assembly line fashion. When the design is mapped to resources available in an FPGA device, this choice will primarily impact the circuit complexity and will secondarily impact maximum frequency and power consumption.

Sometimes we have no choice but to use a RAM. We require a RAM to store the perceived histogram, h'_{p}, because it is used in a feedback loop. Recalling (3.7), we need the bin values of frame $k-1$ to compute those of frame k. We also require a RAM to store the scene histogram, h_s, because we need to process an entire frame of the video input, $y_j[k]$, where j indexes the n pixels in sequence, before we have the correct histogram values.

We choose to use an additional RAM to implement the scene histogram. We implement a ping-pong buffer, where one RAM is used for writing and the other for reading the 'same' array, during one frame, with the roles reversed each frame. As such, we distribute the perceived histogram computation over the frame period, instead of squeezing it into a short interval between frames.

A consequence of the ping-pong buffering is that, at the end of a frame, the perceived histogram is computed from the scene histogram of the previous frame, not the current one. Therefore, compared to the background algorithm, we have introduced a one frame latency. However, for a video rate of 30 fps and a LPF time constant of 0.4 s, in (3.5), the impact of this small latency is negligible.

The memory required by our proposed design is at least the memory required by the three RAMs. Although linear in the number of data bits, the size of each RAM is exponential in the number of address bits. When we use a RAM to compute the scene histogram of a t_in-bit sequence of integers, y_j, binning is therefore desirable. A simple way to achieve binning is to discard s_bin least significant bits (LSBs), which is a zero-cost circuit operation, to produce a t'_in-bit sequence of integers, y'_j, where:

$$y'_j = \lfloor 2^{-s_\mathrm{bin}} y_j \rfloor, \tag{3.10}$$

$$t'_\mathrm{in} = t_\mathrm{in} - s_\mathrm{bin}. \tag{3.11}$$

In Fig. 3.2(a), binning is represented as a right-shift operation ($\gg$) on the *videoIn* signal. The memory, B, required by the three RAMs, in bits, is as follows:

$$B = 3 \cdot 2^{t'_\mathrm{in}} t_h, \tag{3.12}$$

$$t_h = \lceil \log_2 n \rceil, \tag{3.13}$$

where the number of data bits, t_h, represents the worst case where all n pixels fall into a single histogram bin. The number of data bits is also called the wordlength of the RAM.

The binned *videoIn* signal addresses one RAM, based on *mode* signals generated by the controller, for scene histogram counting. The controller also generates a *histBin* signal, which addresses another RAM, for scene histogram readout. This signal, with symbol y', starts at the lowest (or highest) possible address and increments (or decrements), with each clock cycle, until it reaches the highest (or lowest) possible address. Routing of the two possible address signals to the two possible RAMs is done by two dual-input multiplexers, shown in Fig. 3.2(a).

For simplicity, our single-clock design requires that it is possible to read out the entire scene histogram within one frame period. Assuming the video input streams continuously from a nearby CIS, within a hard real-time imaging system, it means we have n clock cycles to address $2^{t'_{\text{in}}}$ words. Fortunately, we have control over the size of the latter, via the parameter s_{bin} in (3.11). We must also allow for latencies, within our circuits, of a few clock cycles to make it all work without glitches.

The *mode* signal, shown in Fig. 3.2(a), is a three-bit control bus. One bit configures one scene histogram RAM into the counting or readout mode. Another bit configures the other scene histogram RAM likewise. For correct synchronization, these signals are not exactly complements of each other, which is why two are needed. The third control bit indicates completion of readout. Counting and readout occur in parallel. However, readout may take fewer than n clock cycles, unlike counting.

Implementing the counting or readout of a scene histogram in firmware, i.e., with digital circuits, is not as straightforward as doing so in software. Figure 3.2(b) illustrates the additional details. In the counting mode, the *addrR* signal, of the RAM, is the binned *videoIn* signal, y'_j. The *dataR* signal is incremented to compute the new histogram bin value. The result feeds back, with a one clock cycle latency, to the *dataW* port, overwriting the original value. In the readout mode, the *addrR* signal is the external *histBin* signal, y'. The *dataR* signal is the external *histValS* signal. The *dataW* signal is zero to initialize the RAM for subsequent counting.

Regardless of mode, each scene histogram RAM performs both a read and write operation each clock cycle. Because the addresses are different, a dual-port RAM is used. To avoid errors, the "read during write" specification of the RAM is configured as "old data". This introduces a latency of one clock cycle between when data is written to an address and when it can be read back. In the event that consecutive *addrW* values are the same, which can only happen during counting, a +2 incrementation is required, as shown in Fig. 3.2(b), for correctness. Which increment to use is decided simply by comparing the current and previous *addrW* values for equality.

Every frame period, one scene histogram RAM is read out. Its output signal, *histValS*, is shown in Fig. 3.2(a). To select the correct *dataR* signal, from the two possible RAMs, we use a multiplexer controlled by the *mode* signal, which is delayed by one clock cycle to account for the latency of the read operation. The reason the multiplexer has a third input, labelled Z for high-impedance, is to signify the end of read out, during the frame period, as determined from the third bit of the *mode* signal. A high-impedance state, for a signal, is supported by the FPGA development tools. Using it helps to debug and avoid errors.

The right half of Fig. 3.2(a) shows the circuitry used to compute the perceived histogram, also stored in a dual-port RAM. This circuitry includes a LPF subcircuit, shown in Fig. 3.2(c). The LPF uses two constants, two multipliers, one adder, and one right shift to implement (3.7) with a latency of two clock cycles. Each multiplier and the adder uses a register, causing the small latency. In a pipelined design, it is desirable to use registers periodically as it enables high-speed operation.

The read-port address of the dual-port RAM matches the address of the scene histogram output. This ensures the same bin of the current scene histogram and previous perceived histogram arrive at the LPF simultaneously. The address signal of the read port is delayed by three cycles for the write port because data presentation and LPF computation take three cycles. This ensures the updated value of the perceived histogram is stored in the right bin. After all bins are computed, the perceived histogram has been updated.

To enable *bit true* validation of the module, a multiplexer is added to switch between the computed LPF output, *histValP*, and the scene histogram itself, *histValS*. To avoid the impact on a feedback circuit of unknown initial states, the first two frames have the perceived histogram RAM follow the bin values exactly from the scene histogram RAMs. Only after the third frame does the perceived histogram RAM depend on the LPF.

3.2.3 Tone Mapping

According to Fig. 3.1, the tone mapping module computes the video output from the input using the base histograms. After reviewing the background algorithm, we introduce the circuit method developed for this purpose.

Background

Li *et al.*'s [35] tone mapping, like Ward Larson *et al.*'s [31] classic algorithm, is a modification of histogram equalization. As a result, we summarize equalization while reviewing Li *et al.*'s algorithm below. We include minor changes to the algorithm, for simplicity, in this section.

Histogram equalization may be used to map tones from a t_{in}-bit video input, denoted y_j, to a t_{out}-bit video output, denoted w_j, where $t_{\text{in}} > t_{\text{out}}$, as follows:

$$w_j = T(y_j), \tag{3.14}$$

$$T(y) = \lceil w_{\text{ref}} P_Y(y) \rceil - 1, \tag{3.15}$$

$$w_{\text{ref}} = 2^{t_{\text{out}}}, \tag{3.16}$$

where P_Y, a cumulative distribution function (CDF), is computed from the PMF, p_Y. Equalization is considered a global operator, T, because the tone of each pixel, indexed by j, is mapped using the same function.

For a discrete random variable, Y, the CDF approximately equals the cumulative sum, c_Y, of the histogram, h, with normalization. Usually, it represents the probability that $Y \leq y$:

$$P_Y(y) = \frac{c_Y(y)}{c_{\max}}, \tag{3.17}$$

$$c_Y(y) = \sum_{k=0}^{y} h(k), \tag{3.18}$$

$$c_{\max} = c_Y(2^{t_{\text{in}}} - 1), \tag{3.19}$$

where y is a t_{in}-bit unsigned integer. If we use the scene histogram, h_{s}, to compute the CDF, then $c_{\max}$ equals the number of pixels, n. However, if instead we use the perceived histogram, h'_{p}, it may be different from n.

Recalling (3.1), the monotonic function F, which defines the sensor response, need not be increasing. For example, the log sensor considered by our previous work [40] has a response that decreases as the stimulus increases. Before any images are displayed, this decreasing response must be inverted. If we define the CDF as the probability that $Y \geq y$ then the required inversion is baked into the tone mapping. Instead of (3.18) and (3.19), we employ:

$$c_Y(y) = \sum_{k=y}^{2^{t_{\text{in}}}-1} h(k), \tag{3.20}$$

$$c_{\max} = c_Y(0). \tag{3.21}$$

After equalization, the histogram of the video output will be uniform, notwithstanding the constraints imposed by the discreteness of the video input. Although $T(y)$ is -1, in (3.15), when $P_Y(y)$ is exactly zero, w_j is never less than zero, in (3.14), because Y never takes on values y where the probability of doing so is zero. The form of (3.15), which is a little different from Li *et al.*'s corresponding equation, ensures that if the video input is perfectly equalized then the video output will be likewise equalized.

Li *et al.* modify the histogram equalization to constrain the visibility, in the video output, of the sensor noise, ϵ_j, given in (3.1). Following the same approach:

$$w_j = T(F(x_j) + \epsilon_j), \tag{3.22}$$

$$\approx T(F(x_j)) + T'(y_j)\,\epsilon_j, \tag{3.23}$$

$$T'(y) \approx w_{\mathrm{ref}}\, p_Y(y), \tag{3.24}$$

$$\approx \frac{w_{\mathrm{ref}}\, h'_{\mathrm{p}}(y)}{n\,\Delta y}, \tag{3.25}$$

where the derivative of the CDF, P_Y, is the PMF, p_Y. The latter is computed, according to (3.3), using the perceived histogram, h'_{p}, instead of the scene histogram, h_{s}. The sum of the perceived histogram approximates n, the exact sum of the scene histogram, and so we continue to use it, in the denominator of p_Y, for simplicity.

From (3.23) and (3.25), the standard deviation of the sensor noise in the video output, σ_w, is as follows:

$$\sigma_w(y) \approx \frac{w_{\mathrm{ref}}\, h'_{\mathrm{p}}(y)}{n\,\Delta y}\sigma_\epsilon, \tag{3.26}$$

where σ_ϵ is the standard devation of the sensor noise, ϵ, in the video input. We assume the latter is independent of stimulus, x, and response, y. Considering Li *et al.*'s [34] work on polynomial-based FPN correction, this is a reasonable assumption for a nonlinear sensor.

Returning to Li *et al.*'s [35] work on histogram-based tone mapping, we hide the sensor noise in the quantization error of the video output, w_j. For an integer output signal, the quantization error is uniformly distributed over a ± 0.5 LSB interval, yielding a standard deviation of $1/\sqrt{12}$ LSBs. To keep σ_w, in (3.26), less than or equal to this threshold, we require:

$$h'_{\mathrm{p}}(y) \le h_{\max}, \tag{3.27}$$

$$h_{\max} = \left\lceil \frac{n\,\Delta y}{w_{\mathrm{ref}}\sqrt{12}\,\sigma_\epsilon} \right\rceil, \tag{3.28}$$

where rounding up is chosen to ensure that $h_{\max}$, a uniform ceiling on histogram bins, is never zero.

As with Ward Larson *et al.*'s [31] classic algorithm, Li *et al.*'s algorithm enforces a ceiling, computed differently, before equalization. To produce the

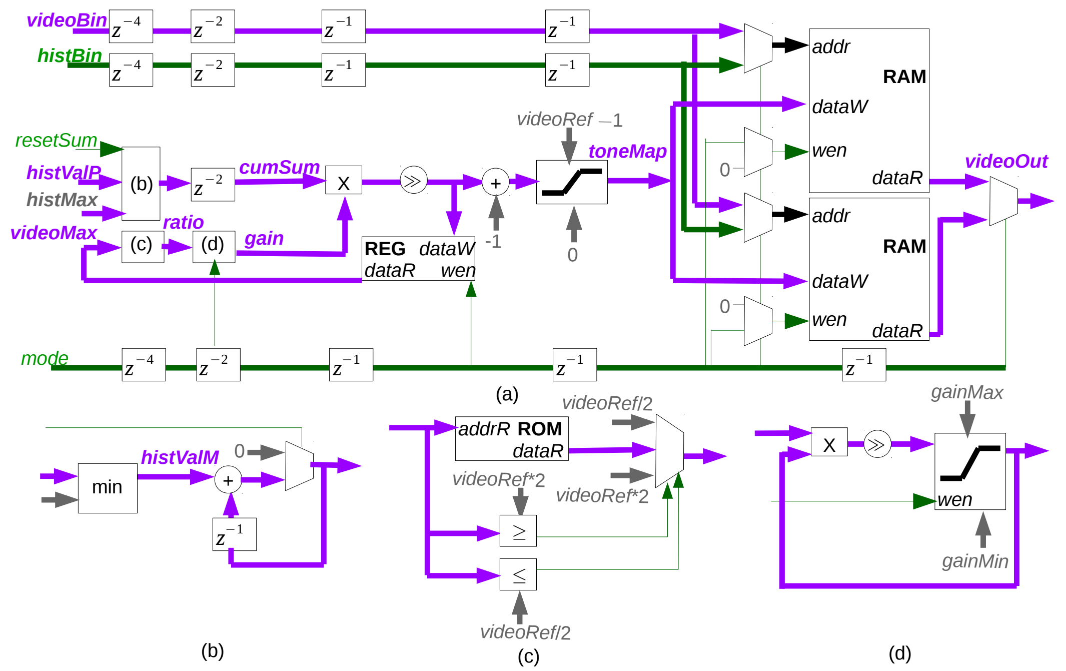


Figure 3.3: Circuit schematics of the tone mapping module. Two RAMs are used, one to map the video input to the output, based on the previous perceived histogram, and the other to compute a new map, based on the current perceived histogram. To avoid division, which is relatively difficult to implement, a combination of feedback, multiplication, and bit shifting is used, with the help of registers and a ROM.

operator T, in (3.15), we use the following for h, in (3.18) or (3.20):

$$h_{\mathrm{m}}(y) = \min(h'_{\mathrm{p}}(y), h_{\max}), \tag{3.29}$$

where the perceived histogram, h'_{p}, is replaced with a *modified* histogram, h_{m}, and the normalization value, in the denominator of (3.17), is replaced by the sum of the modified histogram, denoted $c_{\max}$ as before, which in general may be much less than n.

Method

Figure 3.3 presents our proposed circuit to compute the tone map efficiently in pipelined fashion. We elaborate on the design method below. While other choices are possible in places, the ones that we make are ultimately validated and evaluated in the next section.

Fundamentally, we need subcircuits to employ and compute the mapping function, T, in (3.14) and (3.15), respectively. We design these to operate in parallel. Because the video input must be tone mapped continuously in hard real time, circuit requirements are relaxed if we distribute the tone map computation over the frame period. In turn, this enables higher operating frequencies.

Additionally, we represent the mapping function, T, as a look-up table (LUT) using a RAM. To implement the parallel processing described above, in ping-pong fashion, two single-port RAMs are needed, as shown in Fig. 3.3(a). Based on a multi-bit control signal, *mode*, one RAM operates, in read mode only, on the binned video input, y'_j, and the other operates, in write mode only, on the histogram bins, y'. Both are t'_{in}-bit signals, which means the memory required depends on the design parameter, s_{bin}, in (3.11). The RAMs switch roles each frame.

No additional RAM is required for the modified histogram, h_{m}, of the background algorithm. Instead, we compute and employ it on the fly, in pipeline fashion, as shown in Fig. 3.3(b). This is because the base histograms module, in Fig. 3.2, outputs the bin values of the perceived histogram, h'_{p}, in sequence. We enforce a ceiling, as per (3.29), in one clock cycle and then compute a cumulative sum with a simple accumulator. A small extra part resets the accumulator to zero at the start of each frame, requiring a one-bit control signal.

The cumulative sum is computed according to (3.18) or (3.20), depending on whether F, in (3.1), is monotonically increasing or decreasing, respectively. We configure the controller to output the histogram bins, y', from low to high or high to low, respectively. Either way, in the last clock cycle of the process, the sum of the modified histogram, c_{max}, is computed. This value is required to compute the CDF, P_Y, in (3.17), which is the main part of the tone map, T, in (3.15).

There is one minor issue and one major problem computing the CDF, P_Y, in (3.17). The minor issue is that we require c_{max} before it is computed. Because the modified histogram is a function of the perceived histogram, which changes slowly, we expect the sum of the modified histogram, $c_{\max}$, to change slowly. Therefore, with a simple register, we could store and employ the previous $c_{\max}$ to compute the current P_Y approximately.

The major problem is that division is relatively difficult to implement in an FPGA. A simple implementation we initially adopted prevented our circuit from operating at high frequencies. As a result, we developed an alternative, composed primarily of the subcircuits shown in Figs. 3.3(c) and (d). Usage of these subcircuits, as well as an additional register, is shown in Fig. 3.3(a).

A first step in the no-division method is to combine (3.15) and (3.17) and replace the ratio $w_{\text{ref}}/c_{\max}$ with an integer multiplication and right shift, as follows:

$$T(y') = \lceil 2^{-t_{\text{out}}} A\, c_Y(y') \rceil - 1, \tag{3.30}$$

where t_{out} is the wordlength of the video output. These operations are shown in Fig. 3.3(a). Our previous work [40] elaborates on how to implement a right shift followed by a round down or round off operation. A similar approach is used to implement a right shift followed by a round up operation.

The gain, A, in (3.30) should be as follows:

$$A = \text{round}(2^{t_{\text{out}}} w_{\text{ref}}/c_{\max}). \tag{3.31}$$

When it is the correct value, we get the following:

$$\begin{aligned} w_{\max} &= \lceil 2^{-t_{\text{out}}} A\, c_{\max} \rceil, & (3.32)\\ &\approx w_{\text{ref}}, & (3.33) \end{aligned}$$

where $w_{\max}$ is an intermediate value of the tone map computation, in (3.30), at the end of the process, i.e., when $c_Y(y')$ equals $c_{\max}$. As shown in Fig. 3.3(a), we store $w_{\max}$ in a feedback register. The register is updated once each frame, at the appropriate clock cycle, using a control signal generated by the controller.

At the beginning of each frame period, we compare the previous intermediate value, $w_{\max}$, to the expected value, w_{ref}, a constant. If it is too low or high then we know that our gain, A, is too low or high, in which case we adjust the latter by an appropriate amount. In particular, if it is too low or high by a factor of two or more, we simply double or halve A. Because the modified histogram changes slowly, A will very quickly be within a factor of two of the correct value.

We know that A is near to its correct value when w_{max} is greater than $w_{\text{ref}}/2$ and less than $2w_{\text{ref}}$. In this situation, we obtain the adjustment ratio using a constant LUT, which is a relatively small read-only memory (ROM). The LUT implements the following function:

$$R(w_{\text{max}}) = \text{round}(2^{t_{\text{out}}} w_{\text{ref}}/w_{\text{max}}). \tag{3.34}$$

Thus, we precompute the integer values in (3.34), which include divisions, for a small range of possible w_{max} and store them in a ROM. The number of words is proportional to w_{ref} and the wordlength to t_{out}. Usually, the latter is 8 bits, in which case the former is 256.

Figures 3.3(c) and (d) present the subcircuits used to compute the adjustment ratio and the gain, respectively. Because of the left shift in (3.34), a corresponding right shift and round off is implemented in Fig. 3.3(d). This realizes the following equation:

$$A[k] = \text{round}(2^{-t_{\text{out}}} R[k]A[k-1]), \tag{3.35}$$

where k is the frame number. For consistency, the cases where A must be halved or doubled are also treated by computing a left-shifted adjustment ratio, as shown in Fig. 3.3(c). Note that w_{ref} equals $2^{t_{\text{out}}}$, as per (3.16). Because the ROM takes one clock cycle for read out, the parallel comparisons are equally clocked.

The minimum, A_{min}, and maximum, A_{max}, of the gain may be calculated, using (3.31), from the maximum and minimum of the modified histogram sum, c_{max}, respectively. The maximum of the latter is either the total number of pixels, n, or the histogram ceiling, h_{max}, times the total number of bins, $2^{t'_{\text{in}}}$, whichever is smaller. The minimum is simply h_{max}, which corresponds to the case where all pixels fall in one bin.

The limits on A, which are precomputed constants, are enforced by the limiter in Fig. 3.3(d). It includes a register to store the limited value of A.

Updating of the register occurs once per frame period, at the appropriate clock cycle based on a control signal generated by the controller. The initial value of A is set to A_{min} and, to ensure validity, the control signal is only provided from the third frame on, i.e., after the base histograms LPF is enabled. The upper limit, A_{max}, determines the wordlength of the register, as well as the sizes of related buses.

Finally, as shown in Fig. 3.3, delays are introduced in various places, which is essential for a pipelined design to function correctly. These delays are based on the latencies of parallel circuit paths that must remain synchronous. As with the base histograms module, operations inside circles are not clocked, i.e., they use combinational logic. Operations inside rectangles are clocked.

3.3 Results and Discussion

The previous section proposed a pipelined circuit method to realize histogram-based tone mapping of a monotonic nonlinear CIS in hard real time. The proposed method is validated by comparing its internal signals and video outputs, for a variety of video inputs, to the same data computed by a high-level implementation of the background algorithm in MATLAB. In addition, realized circuits are evaluated using FPGA tools, such as ISE from Xilinx and Quartus from Altera, now owned by Intel.

3.3.1 Validation

Digital circuits were designed and synthesized for the Xilinx XC6SLX45T and Altera EP3C40 devices. These two FPGAs were chosen because they were the simplest ones, from the Xilinx Spartan-6 and Altera Cyclone III families, that fit our design. Moreover, these device families were the lowest cost ones in production, from the two leading suppliers, at the time this research began.

Video Input

For test purposes, we create a variety of video inputs as follows. To begin with, we downloaded several high/wide DR image sequences provided by Kronander

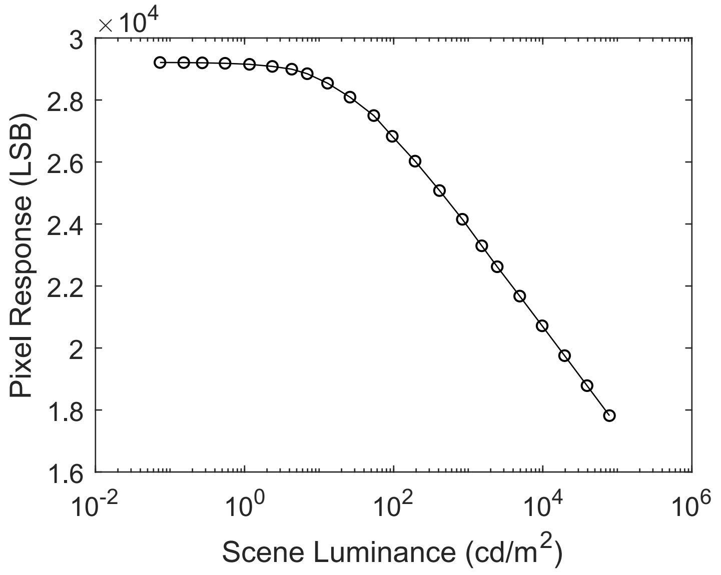


Figure 3.4: Response of a simulated log sensor. Experimental data from a TTVGA pixel array, reported in the literature, was monotonically interpolated to simulate images of high/wide DR scenes for a wide variety of video formats.

et al. [29]. These binary files contain pixel responses that have been calibrated and converted to scene luminances. The data readily supports the HD video format at 30 fps. Using standard downsampling and upsampling techniques, applied offline to each image separately, we could also test smaller and larger video formats.

We convert the floating-point scene luminances to fixed-point pixel responses from a monotonic nonlinear CIS as follows. Using experimental data, collected by Li *et al.* [34], from a small-format log sensor, we constructed a cubic Hermite spline to represent the monotonic function, F, in (3.1). This is shown in Fig. 3.4. We then apply the function to each scene luminance, add zero-mean Gaussian noise of the appropriate magnitude, and round off to the nearest integer. In this manner, we create a 16-bit video input, i.e., the *videoIn* signal.

As explained in Section 3.2, we shift out a few bits of the *videoIn* signal before further processing. An advantage of this is that it reduces the size of the RAMs required by the proposed design. However, shifting out too many bits may alter the tone map significantly. Considering the log sensor noise

Table 3.2: Video formats used to evaluate the proposed circuits. Frames are composed of n_1 scan lines and n_2 pixels per line. The clock frequency is the number of pixels times a frame rate of 30 Hz.

Format	$n_1 \times n_2$	Pixels	Clock
TTVGA	48×64	3.07 k	92.2 kHz
HQVGA	160×240	38.4 k	1.15 MHz
VGA	480×640	0.31 M	9.22 MHz
HD	720×1280	0.92 M	27.6 MHz
FHD	1080×1920	2.07 M	62.2 MHz

levels after FPN correction, as measured by Li *et al.*, we shifted out two bits, resulting in a 14-bit binned video input, i.e., the *videoBin* signal.

Table 3.2 defines the video formats used in our analysis. The well known video graphics array (VGA) format provides a basis for the standard half quarter VGA (HQVGA) format and the non-standard tenth tenth VGA (TTVGA) format. Small formats, such as the latter, may be used by academics who wish to experiment with alternative CIS architectures. Furthermore, by careful testing with smaller format videos, it was easier to debug errors in the proposed circuit method before scaling up to the HD and FHD formats of primary interest.

Internal Signals

As with our previous award-winning work [40], on circuit methods for FPN correction and SPN filtering, we aim for a *bit true* design in this work. This means that we require a perfect match, bit-for-bit, between the proposed circuit's output and the background algorithm's output. An alternative approach would be to aim for a high peak SNR (PSNR), where the PSNR represents the error, in decibel (dB), between the two outputs. A bit-true design corresponds to a PSNR of infinite dB.

One advantage of a bit true design is that it decouples the algorithm design, in which all approximations must be incorporated, from the circuit design, which has its own set of challenges that require attention and a different skill set. Another advantage is that, in the process of developing a bit true design, a thorough circuit validation is required, which increases the robustness of the end design.

Unlike our previous published work, it was essential in this work to tap internal signals of the proposed circuit, due to its relative complexity. In the course of developing a bit-true design for the video output, we had to develop a bit true design for critical circuit states, as deduced from internal signals that could be tapped. Using the FPGA design tools, we saved these internal signals to files that were then analyzed.

The proposed circuit has five RAMs, which define a large number of circuit states. Circuit outputs are a function of inputs and states, where the states themselves are functions of inputs. Because two pairs of RAMs are configured as ping-pong buffers, we could take them in groups. By tapping the *histValS* and *histValP* signals, shown in Fig. 3.2, we validated the scene and perceived histogram states, which use three RAMs. Similarly, we tap the *toneMap* signal, shown in Fig. 3.3, to validate the tone map states, which use two RAMs.

Figure 3.5 illustrates the validation described above. The x-axis gives nonzero values of the *videoBin* signal. The left y-axis gives frequency counts of the validated scene and perceived histograms, computed at the 6 s mark for a video input based on the "students" data from Kronander *et al.* [29]. The right y-axis gives the validated "Li (w/o div)" tone map, as well as additional tone maps not computed by the circuit but provided simply for reference.

As shown in the figure, the perceived histogram is close to the scene histogram. Differences arise because of low-pass filtering. When these internal signals are observed over time, the perceived histogram changes smoothly, which reduces video artefacts after tone mapping, as previously demonstrated by Li *et al.* [35]. The circuit computes a modified histogram on the fly by limiting the perceived histogram to a precomputed ceiling. Without this step, the tone map would implement a histogram equalization. As shown, the equalization map can exhibit higher contrast in places, which has the disadvantage of amplifying sensor noise.

The proposed tone map is computed by accumulation and normalization of the modified histogram. Accumulation is done from high to low bin values, i.e., from right to left in Fig. 3.5, because the log sensor has an inverting response,

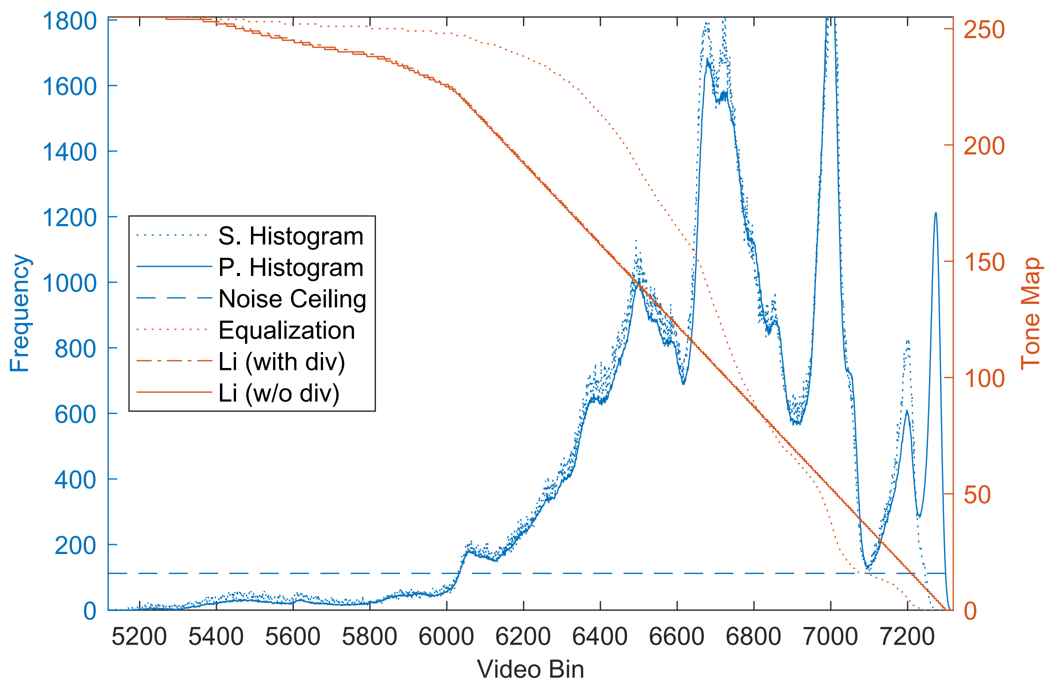


Figure 3.5: Internal states of the proposed TMO circuit. An example is shown for the HD video format at the end of one frame period. The scene (S) and perceived (P) histograms are computed by the base histograms module. The tone map, which applies a ceiling to the perceived histogram before accumulation, is computed by the tone mapping module, which implements Li *et al.*'s algorithm without division. Also shown are tone maps for histogram equalization and for Li *et al.*'s algorithm with division.

as shown in Fig. 3.4. Therefore, the resulting tone map is also inverting, which corrects the original inversion. Although normalization requires division, it is realized without division, as explained in Section 3.2. Nevertheless, circuit results without division are practically identical to algorithm results with division, as shown in Fig. 3.5.

Video Output

Whereas validation of the internal signals was critical for debugging the proposed circuit methods, the primary validation of interest concerns the video output, *videoOut*, an 8-bit unsigned integer signal that may be viewed on a LDR display. Display values range from 0 to 255, thanks to tone mapping of the high/wide DR video input, *videoIn*, a 16-bit unsigned integer signal.

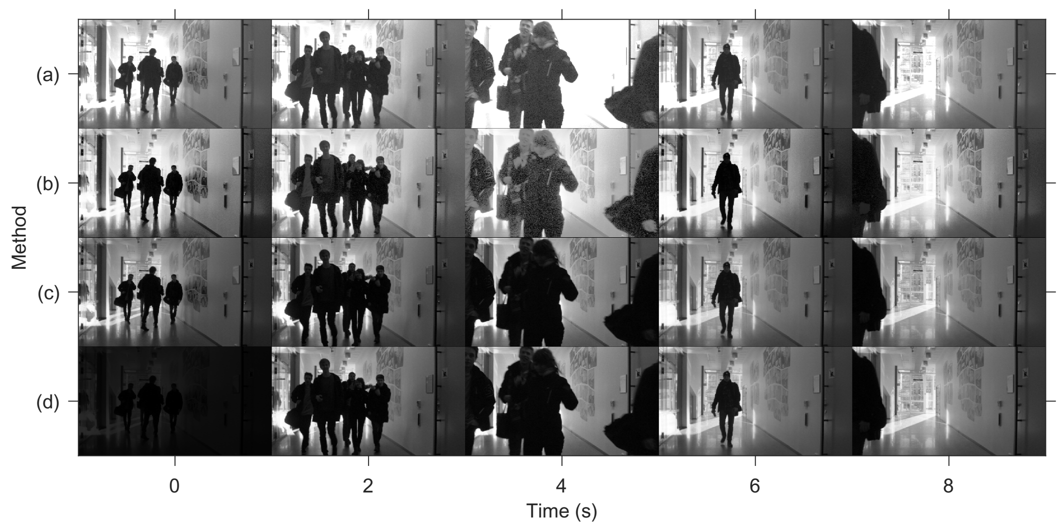


Figure 3.6: Video output of four TMOs for the HD video format. Three TMOs were implemented only in software: (a) a simple photometric algorithm; (b) histogram equalization; and (c) Li *et al.*'s algorithm with division. The fourth one, (d) Li *et al.*'s algorithm without division, was implemented also in firmware using the proposed circuit methods. First 0.1 s aside, (c) and (d) were practically identical. Video input is based on high/wide DR data from the literature.

Figure 3.6 illustrates this final validation, using the "students" video input previously described. In this example, which is just over 8 s long, a group of students are walking down a dark hallway in the foreground, while daylight enters through windows in the background. As before, a bit true validation was performed, meaning the circuit was debugged until there were no bit errors.

A need for superior tone mapping is evident in Fig. 3.6(a), which depicts the results of a simple photometric algorithm [35]. This algorithm implements white-point saturation, gamma correction, and scaling, according to the sRGB specification. Although histogram equalization, shown in Fig. 3.6(b), avoids the saturation issues, it can exaggerate contrast and thereby amplify sensor noise. Moreover, when implemented independently from frame to frame, at 30 fps, flickering artefacts may arise due to dramatic changes in the scene histogram occuring too quickly and/or frequently.

Advantages of Li *et al.*'s [35] algorithm, shown in Fig. 3.6(c), have been previously demonstrated. What this work proposes, however, is a novel pipelined circuit to realize those advantages, in hard real time, using an FPGA. To eliminate a problematic division operation, a significant change was also made to the background algorithm. The results of the proposed approach are shown in Fig. 3.6(d). It takes a few frames for a feedback circuit, used to replace division by multiplication, to settle. Thereafter, results are practially identical with or without division.

One issue that remains, evident in Figs. 3.6(c) and (d), is that the students look too dark. This is partly due to the relatively high dark limit of the log sensor, indicated by the knee point in Fig. 3.4. Below the dark limit, noise dominates over the signal. Because Li *et al.*'s method limits the visibility of sensor noise after tone mapping, contrast is reduced at low scene luminances. The best way to address this issue may be to use a linlog sensor instead, which enjoys a lower dark limit.

3.3.2 Evaluation

After the proposed circuits were validated, for all video formats in Table 3.2, we proceeded to evaluate the following: the circuit complexity, in terms of logic and memory required; the max frequency of valid operation; and the power consumption, broken down into static and dynamic parts.

Complexity

Because of differences in the underlying FPGA architectures, Xilinx and Altera design tools report logic and memory in different ways. Quartus from Altera simply reports the number of LEs and memory bits required. To compute the LEs with Xilinx, we sum the different types of "slices" reported by ISE, including digital signal processing (DSPg) ones. Similarly, we perform a weighted sum to compute the number of bits from the numbers and types of BRAMs reported. In both cases, some logic resources may be used to implement memory.

Figure 3.7 gives the logic and memory, versus number of pixels, required by the complete circuit for both devices. The number of pixels vary with video format, as per Table 3.2. Other parameters were kept constant.

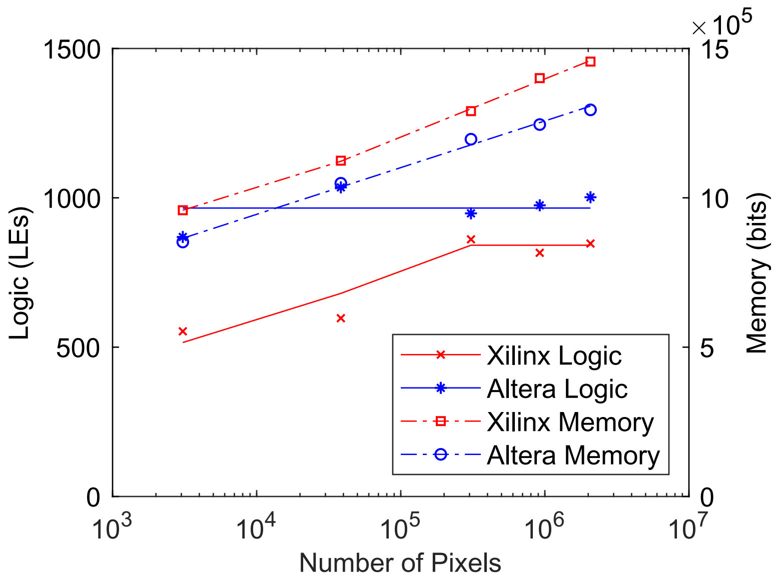


Figure 3.7: Complexity of the complete TMO circuit. With the Altera device, the required logic is roughly constant and the required memory is proportional to the log of the number of pixels. The Xilinx device behaves similarly, especially for the larger video formats.

Relative to the available resources in the chosen low-cost devices, the required logic is low. However, the required memory is significant because of the five RAMs involved. Nevertheless, the required memory capacity grows slowly, i.e., linearly with the log of the number of pixels. With the Altera device, the required memory is almost exactly the value computed by summing the size of the five RAMs.

Frequency

Figure 3.8 presents the max frequency, versus the number of pixels, of the complete circuit for both devices. We determine this frequency, independent of video input, using static timing analysis (STA). We reduce the clock period systematically until the FPGA design tool reports timing violations, e.g., failure to meet setup or hold constraints somewhere in the circuit.

A video format is supported if the max frequency, determined by STA, exceeds the required frequency, given in Table 3.2. As shown in Fig. 3.8, all video formats are supported even though the max frequency decreases as the

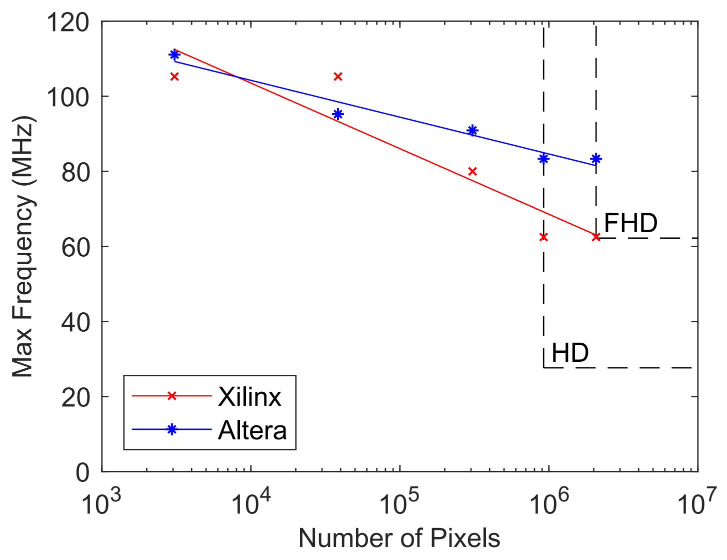


Figure 3.8: Max frequency of the complete TMO circuit. With both devices, the max frequency decreases roughly linearly with the log of the number of pixels. All video formats that were considered are supported, although the margin is small with one device for the FHD case.

number of pixels increases. Considering a similar though opposite trend for required memory, as shown in Fig. 3.7, RAM operations may be the limiting factor. Larger RAMs require more time due to propagation delays.

Power

Figure 3.9 presents the power consumption, versus video format, of the complete circuit for both devices. The FPGA tools report total power and static power, from which dynamic power is easily computed. Also, power consumption depends on the clock frequency. The chosen frequencies are the ones reported in Table 3.2, not the max frequencies.

The static power is the power consumed by the device irrespective of functionality. As shown in Fig. 3.9, this power dominates for the smaller formats. For the larger formats, the dynamic power increases but remains a small multiple of the static power. Because the RAM sizes are strongly correlated with the number of pixels, they are likely responsible for the observed trend.

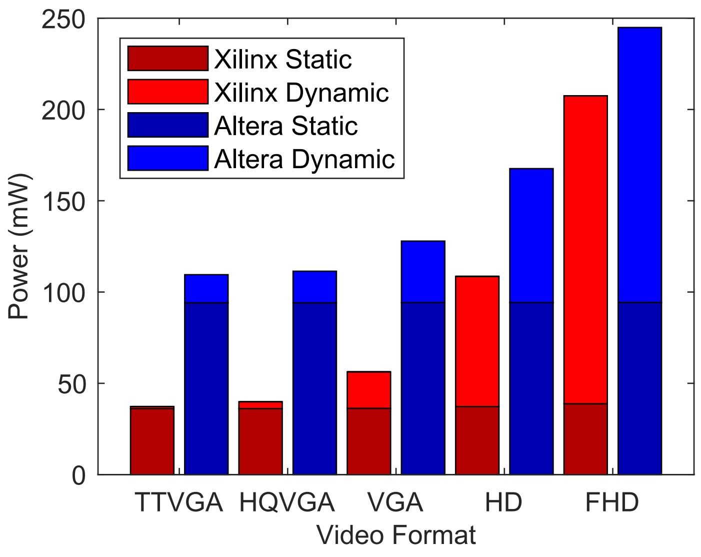


Figure 3.9: Power consumption of the complete TMO circuit. With both devices, the dynamic power grows with the video format. Nevertheless, it remains on the order of the static power.

3.3.3 Significance

Table 3.3 summarizes the specifications of the designed circuits, including a breakdown by the main modules, for the largest video format that was supported. For reference, there are 81, 922 LEs and 2, 138, 112 bits available in the chosen Xilinx device. There are 56, 168 LEs and 2, 396, 160 bits available in the chosen Altera device.

We revisit our main competitors, namely other TMOs implemented using FPGAs, to compare our results against their designs. In comparison to Ofili *et al.* [41], our design does not present visual artefacts, although we consume more memory, and we are able to process FHD videos. Their design processes HD videos.

Against Popovic *et al.* [42], our work also achieves FHD but with a lower frame rate. They choose 60 fps and our design runs at 30 fps. Our design can use a low-cost FPGA in comparison to the high end component used by them. Their design uses more DSPg slices, i.e., 30, whereas our Xilinx design uses 13. But Popovic *et al.*'s design does not consume any memory, which is a good advantage depending on the nature of the project. However, Popovic's TMO

Table 3.3: Circuit specifications for the FHD video format. In addition to the complete TMO circuit, which includes the controller, the base histograms and tone mapping modules were evaluated as separate circuits. Required LEs and bits are given, in parentheses, as a fraction of available resources. The chosen Xilinx (XC6SLX45T) and Altera (EP3C40) devices were the simplest ones in the Spartan-6 and Cyclone III families, respectively, that fit the design.

Circuit	Technology	Logic (LEs)	Memory (bits)	Max Freq.	Static P.	Dynam. P.
Base	XC6SLX45T	421 (0.51%)	$1,161,216$ (54.3%)	83.3 MHz	37.3 mW	80.9 mW
Histograms	EP3C40	652 (1.16%)	$1,031,282$ (43.0%)	90.9 MHz	94.2 mW	133.5 mW
Tone	XC6SLX45T	477 (0.58%)	$294,912$ (13.8%)	62.5 MHz	38.2 mW	138.6 mW
Mapping	EP3C40	737 (1.31%)	$262,318$ (10.9%)	111.1 MHz	94.1 mW	49.0 mW
Complete	XC6SLX45T	847 (1.03%)	$1,456,128$ (68.1%)	62.5 MHz	38.8 mW	168.7 mW
TMO	EP3C40	$1,002$ (1.78%)	$1,294,540$ (54.0%)	83.3 MHz	94.3 mW	150.6 mW

does not consider temporal adaptation and was not tested against videos, only still images. Our design has temporal adaptation, which can handle rapidly changing scenes without visual artefacts. Whereas their design approach is based on achieving a high PSNR, we used a bit true approach. As a result, our design was subject to a thorough debugging.

Urena *et al.* [54] also use a low cost FPGA to deploy their TMO. They reported a VGA format only, at 60 fps. Though we set 30 fps for the VGA format, we would be able to achieve 260 fps using the max frequency achieved for this configuration with Xilinx. Their design uses 30,086 LUTs and 36 BRAMs. For Xilinx and the VGA format, our design uses 471 LUTs and 70 BRAMs. Urena *et al.*'s design had 64 levels in the histogram, which limited the performance. Our design has 16,384 levels. Both designs use fixed-point computation.

3.4 Conclusion

Eilertsen *et al.* [14] presents a survey of TMOs for high/wide DR video divided in categories where the top two performers are TMO designs based on histograms, and both global operators. Most of the FPGA implementations of TMOs are simple with fixed global operators using LUTs, e.g., Mann *et al.*'s [38] work, or are implemented in high-end FPGAs [42].

Li *et al.* [35] proposed a tone-mapping algorithm for nonlinear CISs based on histograms. It is a global operator that is updated every frame, considering the noise model of the CIS to improve the histogram quality, and has no temporal effects due to the use of a LPF. We developed a circuit method for this algorithm, a hard real-time design for a truly parallel architecture.

We applied the same generic framework of our previous work [40]. Validation is done using a background algorithm as reference, and zero bit error rate (BER) was achieved. Our novel circuit designs for low-cost FPGAs exhibit low latency and operate continuously in hard real time. The circuits can scale up to FHD video and still fit in low-cost FPGAs. The evaluation was made for the two major FPGA suppliers: Xilinx and Intel, who now own Altera.

Although implented using a HDL, the proposed circuits are explained using schematics, equations, and words. Not only was the output signal validated, but internal signals were tapped to validate internal states in bit-true fashion. All circuits are pipelined and synchronous. No glitches were observed during testing. Besides the zero BER, we visualize the correct operation with the use of sample HD high/wide DR video, as modified using experimental data from a nonlinear CIS.

In conclusion, this work developed, validated, and evaluated a global TMO that can fit in a low-cost FPGA. It does not present visual artefacts, and can be used as a stepping stone to implement other histogram-based TMOs in an FPGA.

Chapter 4

Reconfigurable System-on-Chip

A system-on-chip (SoC) platform having a dual-core microprocessor (µP) and a field-programmable gate array (FPGA), as well as interfaces for sensors and networking, is a promising architecture for edge computing applications in computer vision. In this chapter, we consider a case study involving the low-cost Zynq-7000 SoC, which is used to implement a three-stage image signal processor (ISPr), for a nonlinear CMOS image sensor (CIS), and to interface the imaging system to a network. Although the high definition (HD) imaging system operates efficiently in hard real time, by exploiting an FPGA implementation, it sends information over the network on demand only, by exploiting a Linux-based µP implementation. In the case study, the Zynq-7000 SoC is configured in a novel way. In particular, to guarantee hard real-time performance, the FPGA is always the master, communicating with the µP through interrupt service routines (ISRs) and direct memory access (DMA) channels. Results include a validation of the overall system, using a simulated CIS, and an analysis of the system complexity. On this low-cost SoC, resources are available for significant additional complexity, to integrate a computer vision application, in future, with the nonlinear complementary metal-oxide-semiconductor (CMOS) imaging system.

4.1 Introduction

Nowadays, autonomous devices require high processing data capabilities and portability, to be able to connect to any operating system (OS), but have restrictions in power, weight, and cost. Low latency, between inputs and outputs, is also crucial for high speed applications, which means the use of cloud computing is not ideal. These are technical challenges of edge computing systems, as described by Lee *et al.* [33], which feature large scales, distributed networks, cyber-physical interfaces, dynamic and adaptive environments, and heterogeneous platforms.

A case study of a multi-stage ISPr, implemented with a SoC platform, demonstrates a capacity to process a large amount of data in hard real-time while providing web content. A FPGA embeds the image signal processing (ISPg) algorithms, while realizing true parallel processing. A µP handles hardware interfaces for storage and networking, including the high-level protocols to serve web pages.

Our system corresponds well to key aspects of edge computing, where the device is capable of processing a large amount of data, namely HD high dynamic range (HDR) video, reporting processed information in a high-level format, not overloading the network, and pushing the processing to the edge. Our method also addresses issues faced by current cloud solutions, reviewed by Shi and Dustdar in "The Promise of Edge Computing" [46], by having low latency and private data (all processing is done locally), while processing a large amount of data.

This work uses the FPGA as the main platform to embed all circuits that compose a three-stage pipelined ISPr required for a nonlinear CIS. The research delivers a design flow for hard real-time to be achieved with an SoC. Underlying software algorithms were previously published by our group. They include fixed pattern noise (FPN) correction and salt-and-pepper noise (SPN) filtering [34], as well as digital circuit designs thereof [40], and a tone mapping operator (TMO) [35].

Lapray *et al.* [30] developed an HDR video camera using a Virtex 6 FPGA from Xilinx. They achieved 60 frames per second (fps) with a spatial resolution of 1280×1024 pixels in a homogeneous implementation, using only an FPGA. A lot of FPGA resources were spent in generating the HDR image by combining low dynamic range (LDR) images with different exposure times.

However, the Virtex 6 is a high-end FPGA and their TMO might not be able to handle abrupt changes in illumination.

Another full HDR video camera applied for welding was developed by Mann *et al.* [38]. The system is real-time and low cost, deployed using a Spartan-6 LX45 FPGA from Xilinx. Their design employs look-up tables (LUTs), which guarantees real-time, but a lot of pre-processing has to be made to generate the LUTs. It is a good implementation for homogeneous computing, using only an FPGA that lacks connectivity. As well, the use of LUTs may be too simple to handle fast changes in luminance.

The novelty of this research is the use of a heterogeneous SoC to realize an embedded real-time ISPr for an HDR imaging system that will incorporate a nonlinear CIS. Acting as the master of the system, the FPGA post-processes and interfaces the nonlinear CIS to the µP. The µP enables the system connectivity and extra processing could be realized in the µP as well.

4.2 Apparatus and Application

After describing the platform of our ISPr, this section explains its functionality. As the explanation is high level, using input-output images, please consult the references for more details about the background algorithms and circuits.

4.2.1 System-on-Chip Platform

Figure 4.1 shows the SoC development kit we used. The SoC itself is a XC7Z020 from the Zynq-7000 family of the Xilinx All Programmable SoC architecture. It is composed of a dual-core ARM Cortex-A9, called the processing system (PS), and a 7-series FPGA from Xilinx, called the programmable logic (PL). In this chapter, we simply write µP for the PS and FPGA for the PL to favour generic terminology.

From the available peripherals, we use: the Joint Test Action Group (JTAG) for debugging; the Universal Serial Bus (USB) universal asynchronous receiver-transmitter (UART) for serial communication; the Ethernet controller for networking; and the memory controller to access 1 GB of random-access

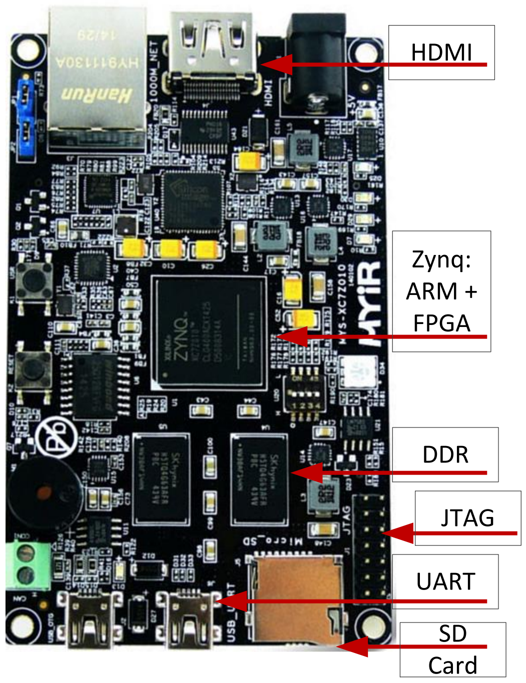


Figure 4.1: Zturn development kit with Zynq-7000 SoC. This board is the platform chosen to deploy the proposed ISPr, and thereby to realize a case study on edge computing. The image is adapted from the literature [56].

memory (RAM), specifically double data rate (DDR) synchronous dynamic RAM (SDRAM). Our ISPr requires memory to store the FPN coefficients, which fit easily into the DDR memory after loading from the Secure Digital (SD) card. The OS is booted from the SD card as well.

Documentation and examples are provided by the vendor, such as C code to embed in Linux and access the board's peripherals, kernel files to configure and recompile the Linux kernel, and device tree source files, to add new hardware components.

Xilinx has a suite of tools, called Vivado, for system design with the Zynq-7000. Vivado can work with hardware description language (HDL) coding and drag-and-drop graphical user interfaces (GUIs). For bare-metal development, i.e., to program the µP with no OS, a software development kit (SDK) is included for C and C++. The Vivado high-level synthesis (HLS) is a third integrated development environment (IDE) that allows designers to develop their FPGA projects using C, C++, and/or System C only, without the need

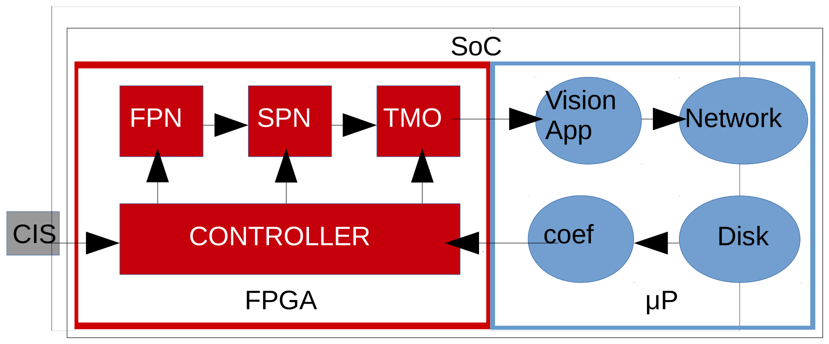


Figure 4.2: Interconnection of primary SoC components. Some are in the FPGA, while others are in the µP. A key part of this work is the interfacing.

for HDL coding.

For standard components, such as DMA and video drivers, we use intellectual property (IP) cores from Xilinx. We also designed custom circuits to make our ISPr. Drivers needed by the SoC are available, according to the FPGA design. For this project, we do not use Vivado HLS.

4.2.2 Image Signal Processor

The SoC allows the design of a heterogeneous system, where work is divided between an FPGA and µP, as shown in Fig. 4.2. The proposed system takes advantage of the highly parallel architecture provided by the FPGA and the rich set of peripheral devices and software libraries available on the µP.

On the FPGA side, a custom digital architecture can execute pixel-level algorithms and multi-channel tasks efficiently, taking advantage of the high throughput and low power of FPGAs, especially for fixed-point computation. The FPGA can also extend the peripherals of the µP, e.g., to accommodate a nonlinear CIS that requires a custom and sophisticated ISPr to be functional.

The µP is composed of: a dual-core ARM processor; standard peripherals for connectivity; and GB-scale memory access with memory controllers. It is capable of running Linux, a multi-threaded OS, and performing additional computation.

As it is "low-power yet high-performance," to quote Kalb *et al.* [25], this heterogeneous system is ideal for realizing the ISPr of a nonlinear CMOS

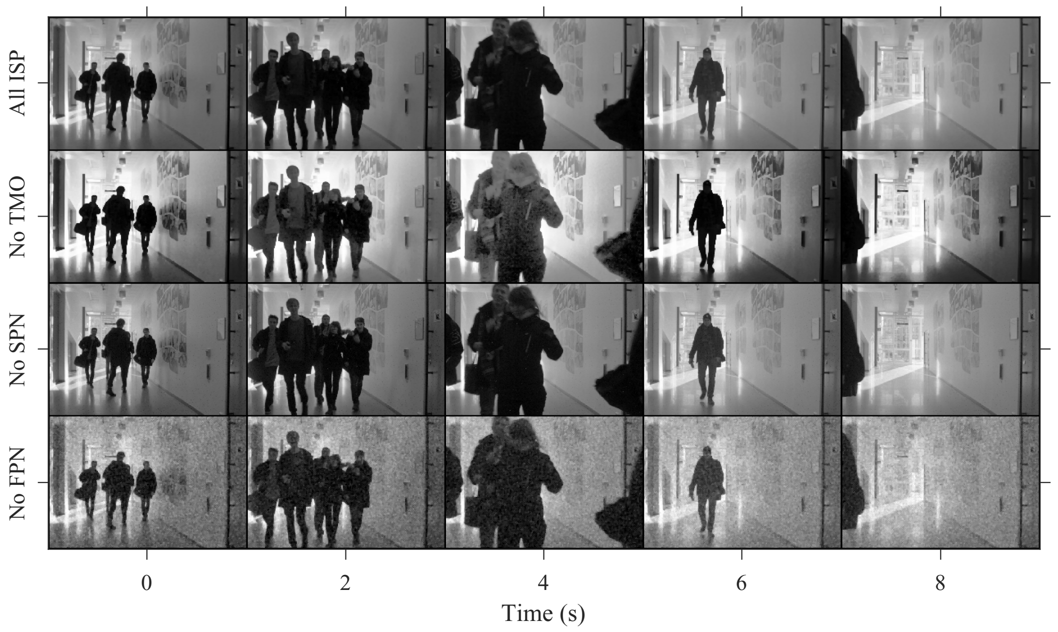


Figure 4.3: Illustration of the ISPr functions. From bottom to top, the image quality improves by adding circuits for FPN correction, SPN filtering, and tone mapping.

imaging system. However, the design flow is challenging due to multiple development environments and the complexities of interfacing the FPGA and µP. Kalb *et al.* worked on a more unified tool chain, a customized real-time kernel for parallel computing, and a hardware system.

A nonlinear CIS, such as the logarithmic one by Mahmoodi *et al.* [37], is ideal for HDR imaging at video rates because it does not need to create the HDR image from multiple LDR images [30] [38]. While nonlinear CISs have been investigated before, the literature generally focuses on small-format cases.

In contrast, using HD HDR videos from Kronander *et al.* [29], this work simulates a large-format nonlinear CIS. Videos were pre-processed to simulate the responses of Mahmoodi *et al.*'s image sensor [37], except with the larger HD format. This resulted in compressed logarithmic responses, having 16 bits of pixel depth, with FPN and SPN incorporated.

Figure 4.3 demonstrates the importance of the three main circuits in the proposed ISPr. The "All ISP" row shows the result with all circuits. The "No

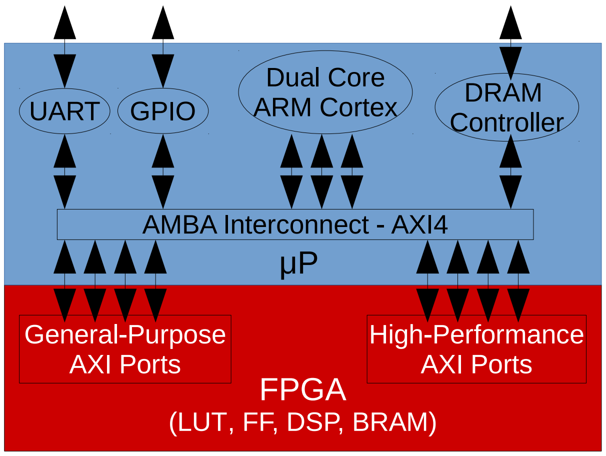


Figure 4.4: Zynq-7000 SoC main blocks and interfaces. The AMBA interconnect protocol is utilized to bridge the FPGA and µP sides of the system.

TMO" row shows the result with a simple TMO in place of Li *et al.*'s one [34]. The "No SPN" row shows what happens when SPN is not filtered. Finally, the "No FPN" row shows what happens when FPN is not corrected.

To make the ISPr operational, a controller circuit is added to the design to provide all control and status signals. The controller implements the external communication protocol, called Advanced eXtensible Interface (AXI), between the ISPr and the DMA, an AXI4-Stream. At the ISPr input, the controller extracts CIS pixel values and FPN correction coefficients from data packages received via DMA. Pixel addresses are generated by the controller and provided to the SPN filter. Finally, frame synchronization signals are generated for the TMO.

4.3 Interfacing Method

As shown in Fig. 4.4, the interfaces between the FPGA and µP include: high performance (HP) ports, for high speed data transfer; and general purpose (GP) ports, for configuration and control. We also set up interrupts to make the FPGA the master of the system and to avoid wasteful polling by the µP.

Our approach differs from the hardware acceleration methodology, shown in Fig. 4.5(1), of the literature. Because we currently simulate the nonlinear CIS, our approach is as shown in Fig. 4.5(2). Ideally, the architecture would

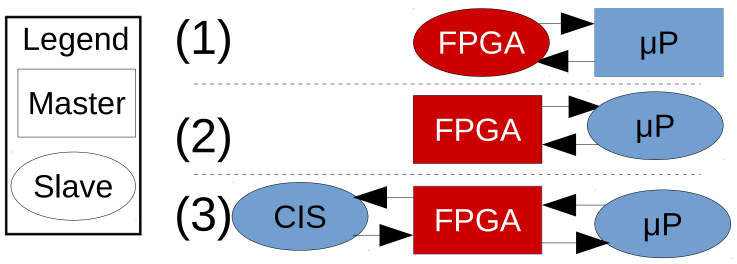


Figure 4.5: Potential master-slave configurations of the system. Squares indicate the master, and ovals the slave(s). We currently use approach (2).

be as shown in Fig. 4.5(3), where an actual nonlinear CIS is controlled by the FPGA.

The protocol we adopted is AXI from Advanced Microcontroller Bus Architectures (AMBAs). It has three variations: AXI4-Full, for multiple devices on HP buses; AXI4-Little, for control; and AXI4-Stream, for point-to-point communication. We use AXI4-Stream, which is the simplest and more efficient version. This protocol, which has a finite state machine with four states, is implemented in the controller circuit.

The design flow for the SoC can be classified either as horizontal or vertical. In the horizontal case, as in Fig. 4.6(1), the FPGA design in Vivado yields the bit file that configures the FPGA. For the µP, repositories and makefiles, configured for cross compilation, generate the boot file, kernel, device tree binary (DTB), and root file system. In the vertical case, as in Fig. 4.6(2), Vivado exports a hardware description file (HDF), which is a container with the FPGA bit file and other design configuration files. PetaLinux [16] then uses the HDF to generate the configuration files for the µP. Modules, packages, and applications can be added in PetaLinux before building the whole project.

We use DMA to transfer data efficiently between the FPGA and µP. The DMA is realized by a standard IP block, which we add to the FPGA design, that is configured and controlled by software running on the µP. To make the FPGA the master, we trigger the µP to run the software, implemented as a Linux ISR, in response to interrupt requests (IRQs) generated by the FPGA and detected by the µP.

The ISR may be in the kernel space or user space of the OS. We imple-

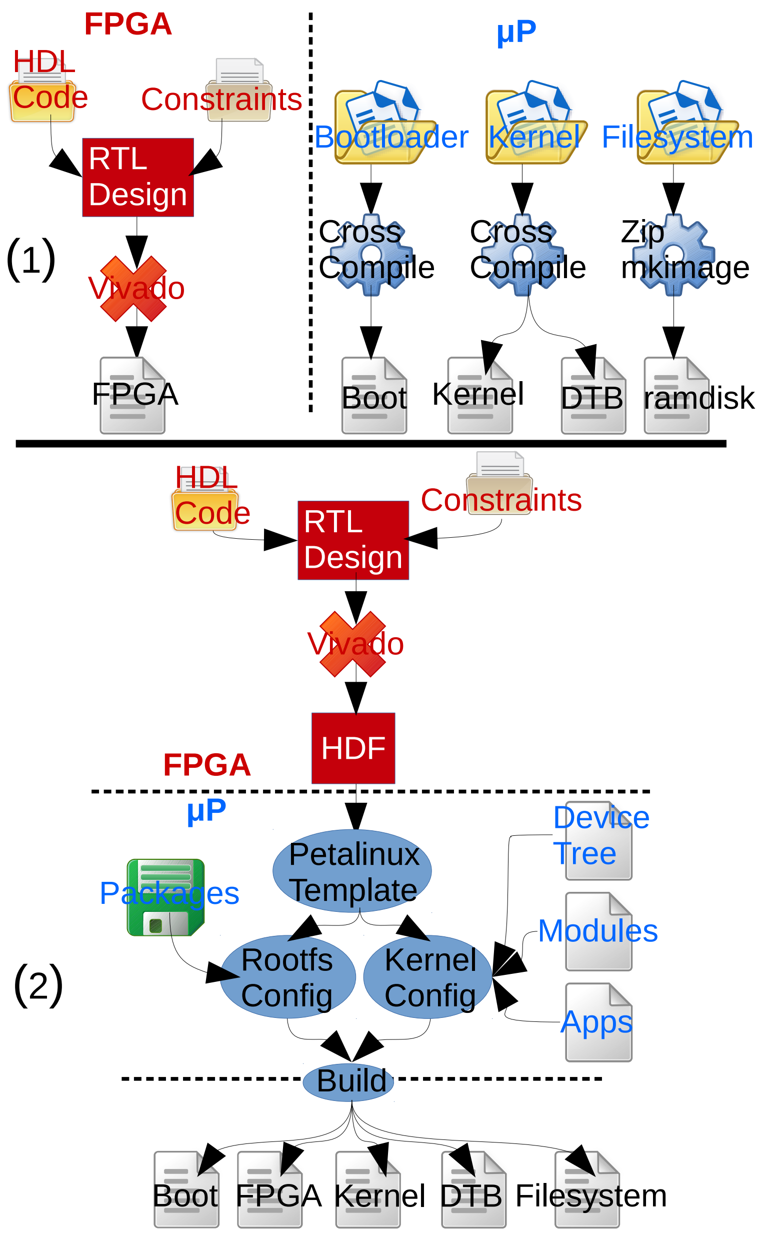


Figure 4.6: Potential design flows for the SoC system. (1) The horizontal one has a more independent methodology, where compilation of the FPGA and μP projects are done separately. (2) The vertical one emphasizes integration, making it easier to load extra packages and configurations.

mented it in the user space, which does not require a custom device driver. We make use of control and status registers, which are memory mapped, of the DMA circuit [5].

A kernel space implementation is more robust, but requires more time and expertise. To design a custom device driver, there are three components: memory allocation, DMA device control, and cache control. Also, the kernel has to reserve a contiguous area of memory in which the driver operates. As in our user space implementation, we favour the simple operation mode of the DMA, as opposed to the scatter-gather mode, because it simplifies both the ISR and AXI4-Stream implementations.

4.4 Results and Discussion

Having presented the apparatus, application, and method, we now turn our attention to system validation and evaluation.

4.4.1 System Validation

On the FPGA side, after HDL code entry and synthesis, the design is tested with functional validation, where most bugs are caught. An initial validation is realized manually with a few small images. Thereafter, for many large images, automatic validation is performed against the same data processed in Matlab. Binary files are used to load input data and save output data. After debugging, zero bit error is consistently achieved.

On the µP side, an initial test is to verify that the Linux kernel detects the new device, i.e., the interrupt-based DMA transfer. This can be done by entering "`dmesg`" at a command prompt. Other useful commands are "`cat /proc/interrupts`," to verify the interrupt number and name, and "`cat /proc/iomem`," to verify the DMA address. When using a PetaLinux driver, self tests of the DMA can be loaded, in the kernel configuration phase, and basic DMA operation can be verified during boot.

To validate the system, a web server was implemented on the µP. It serves a simple web page, accessible from any browser on the same network, that

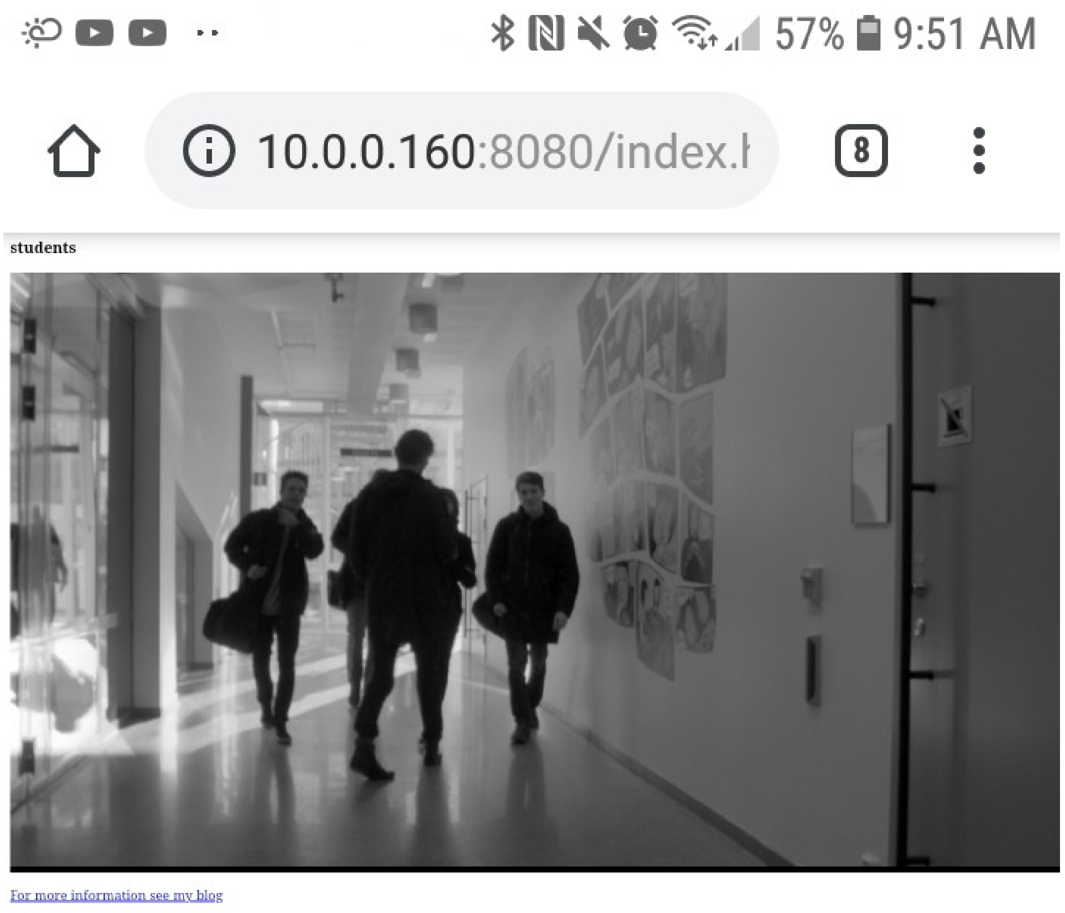


Figure 4.7: Browser screenshot of a web page served by the SoC system. The µP is running a web server to serve the FPGA output over a network.

contains a bitmap image. The ISR we wrote for Linux, coded in C, converts binary data, received from the FPGA via DMA, to a bitmap image file, which is embedded in a static hypertext markup language (HTML) script. Figure 4.7 shows a screenshot of the web page, using a cellphone browser.

Using an oscilloscope, we verified that IRQs were generated at exactly 30 Hz. A trace is shown in Fig. 4.8. For this experiment, we made the µP busy from time to time, also shown in Fig. 4.8, while verifying that the FPGA still generated the IRQs at a constant 30 Hz, which enables hard real-time performance.

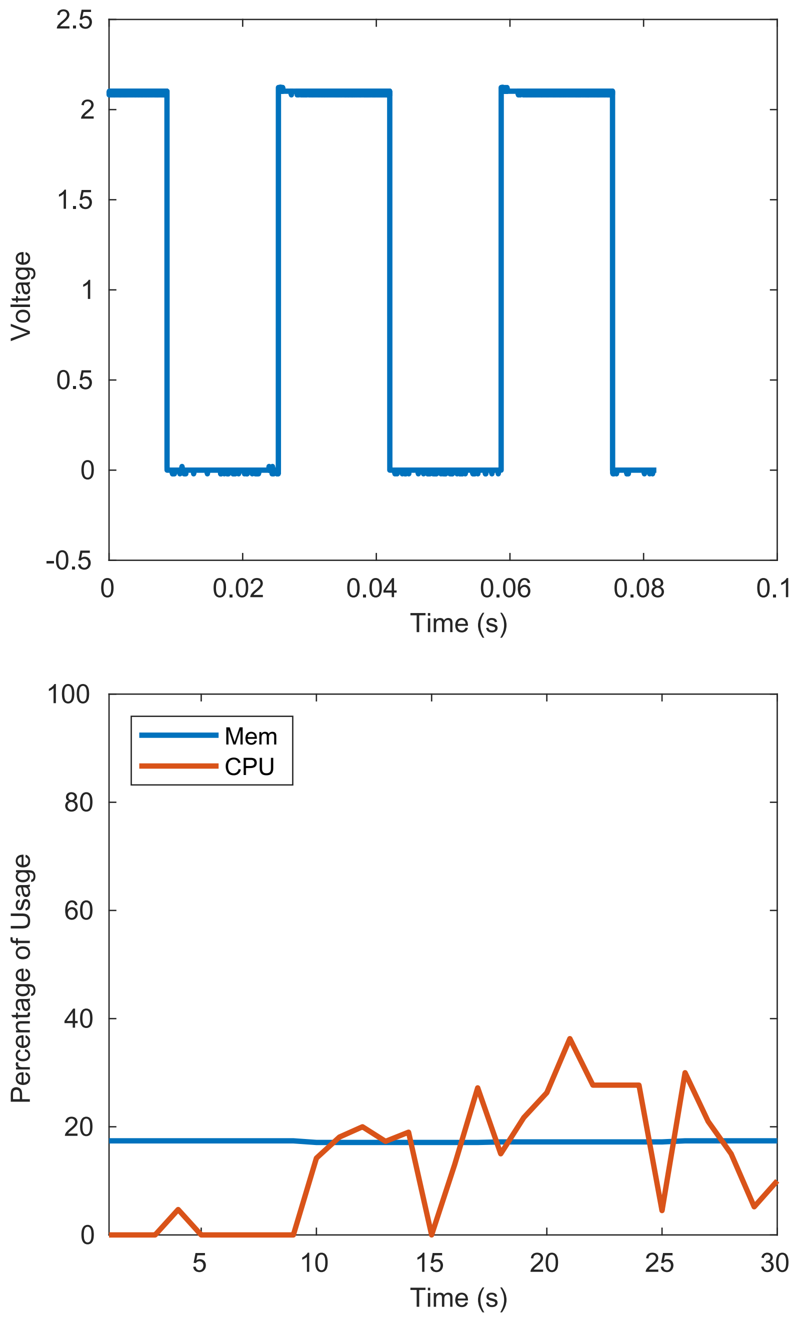


Figure 4.8: IRQ voltage of the FPGA and usage of the µP. The IRQ rate, measured by an oscilloscope, is never disturbed by the µP load, measured by a shell script. Usage is varied by interleaving calculations and sleeping.

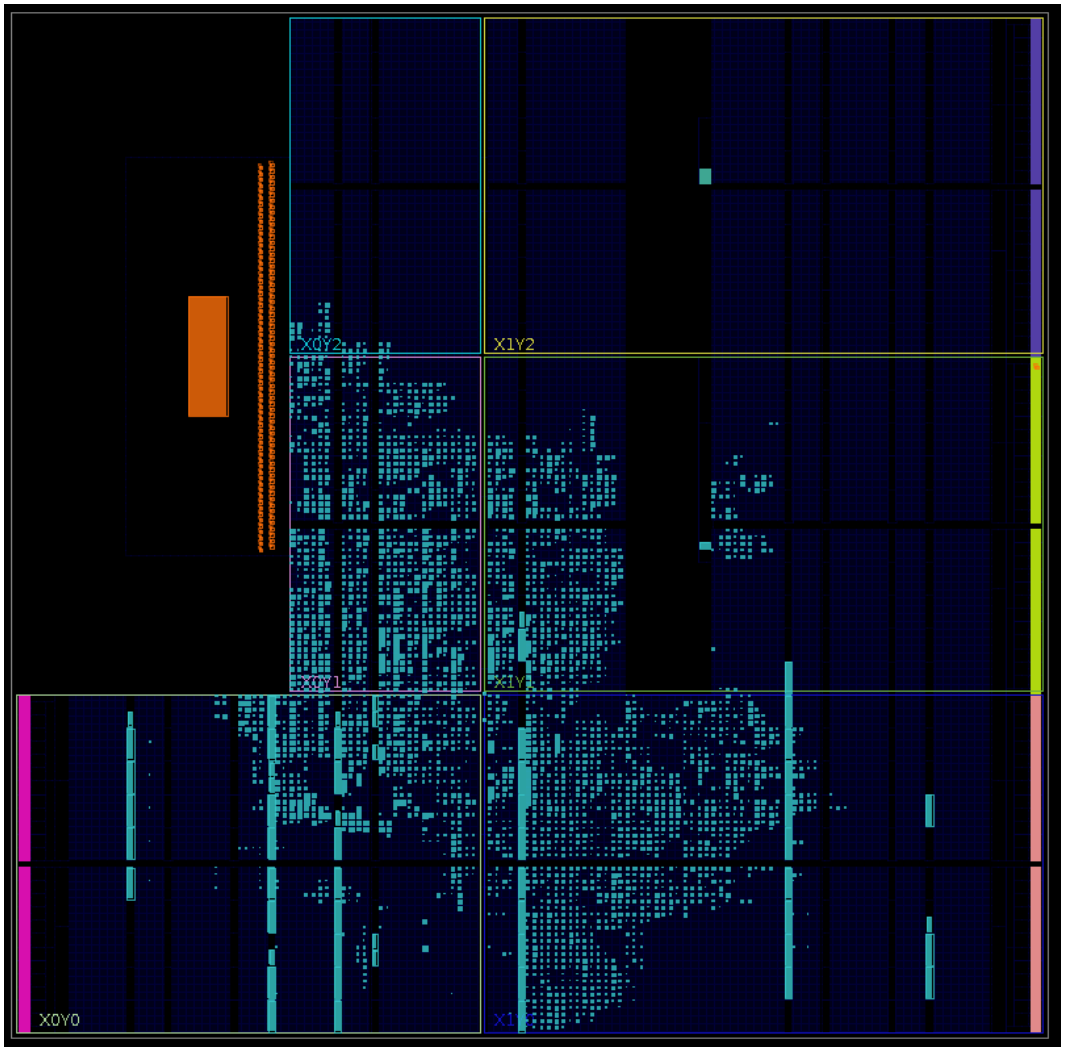


Figure 4.9: Zoomed-out floorplan image of the ISPr design. Using the Vivado IDE, designers can explore the complexity of their FPGA circuits. For this case study, plenty of resources are available for additional complexity.

4.4.2 System Evaluation

To evaluate the complexity of an FPGA design, we have previously reported aggregate numbers of logic elements (LEs) and memory bits [40]. In this work, because we focus on a Xilinx device only and use the Vivado IDE, we report four parameters: logic and memory in LUTs; additional memory in block RAMs (BRAMs); additional memory in flip-flops (FFs); and additional logic in digital signal processing (DSPg) slices.

An overview of the occupancy is given in Fig. 4.9. This floorplan shows the

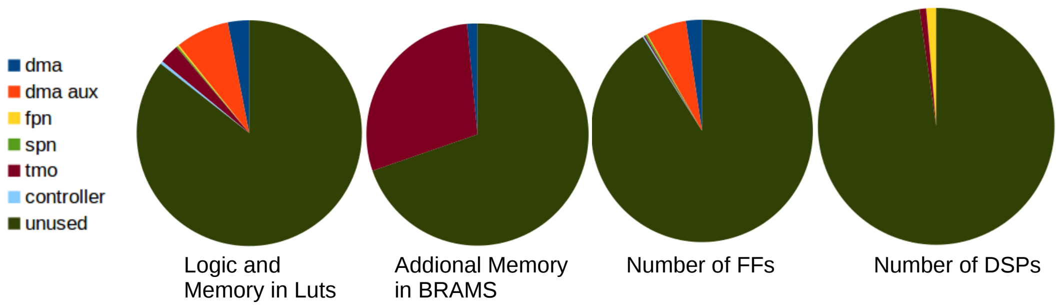


Figure 4.10: Utilization of FPGA resources, at HD resolution, by the ISPr. Relative to the DMA and TMO, few resources are used by the FPN, SPN, and controller circuits. Even considering BRAM memory, mainly used by buffers in the TMO, plenty of unused resources are available for additional processing.

layout of the ISPr for HD resolution. Vertical narrow rectangles, for example, represent BRAMs instantiated for the TMO circuit. Placement of components may be affected by introducing additional constraints on the design. A breakdown of resource usage, as reported by Vivado, is given in Fig. 4.10.

In terms of the circuits being evaluated, we refer to all the auxiliary circuitry needed to support the DMA as "DMA aux," which includes: the smart connectors to bridge HP AXI channels, between the FPGA and the µP; the inter-connectors for bridging GP AXI channels; and the reset circuits to reset and synchronize data transfer, between the FPGA and the µP. These circuits were all inserted automatically by Vivado.

The controller circuit refers to all circuitry responsible for integrating the FPN correction, SPN filtering, and TMO circuits, as well as for interfacing the DMA using AXI4-Stream.

In Fig. 4.10, the number of LUTs primarily represents logic used, but some of it could represent memory. The total usage, primarily for the DMA and DMA aux circuits, is less than a quarter of the amount available. Especially considering the HD resolution of the ISPr, all of the remaining circuits are highly efficient in terms of logic used. There are a total of 53,200 LUTs available.

The BRAMs represent most of the memory in the design. Xilinx embeds in their FPGAs configurable 36 Kb blocks of true dual-port RAM. Figure 4.10 shows that our TMO consumes a significant number of BRAMs, but it is not much more than a quarter of the total available. The TMO relies on BRAMs to store histograms and mapping functions in ping-pong buffers. There are 140 BRAM totalling 4.9 Mb available.

As with the LUTs, Fig. 4.10 shows that the DMA and DMA aux circuits are the ones that use the most FFs. Notwithstanding a router inside the SPN circuit, which is purely combinational logic, the FPN, SPN, and TMO circuits use pipelined sequential logic, which requires FFs. However, as a fraction of the 106,400 FFs available, the amount used by these circuits is negligible.

The last resource depicted, in Fig. 4.10, is the number of DSPg slices. We can see here the influence of the FPN circuit, which is basically composed of adders and multipliers. The TMO also has arithmetic to calculate a mapping function. Only a small fraction of the 220 DSPg slices that are available is used.

In addition to unused resources on the FPGA side, there are unused resources on the µP side. Whereas IRQs are always generated at 30 Hz, if the ISR takes too long to execute, e.g., when saving an image unnecessarily to a disk, then some IRQs will not be served. However, given that the µP has two high-performance cores, there is plenty of scope for computation on its end.

4.5 Conclusion

This work achieved a SoC design flow for hard real-time ISPg of a nonlinear HDR imaging system. Although algorithms and some digital circuits in the ISPr are from previous work, this work integrates all of them into a single real-time platform. The design flow to implement such a system is presented and the system design is validated and evaluated.

Our ISPr used 31.4% of BRAMs, 14.5% of LUTs, 9.0% of FFs, and some DSPg slices, available in a Xilinx Zynq-7000 SoC, to process HD video. Because the FPGA in the SoC is a low-cost device, the realized ISPr design is therefore very efficient.

To interface the FPGA and the µP, IRQs and DMA were used. With a horizontal design flow, having the ISR software in user space, we achieved

25 fps at HD resolution. With a vertical design flow, using PetaLinux, we compiled and inserted a new driver in kernel space, updated missing packages, and implemented a DMA self test. We leave completion of the PetaLinux approach, which would support higher bandwidths, to future work.

One novelty of our design flow is that the FPGA is the master of the SoC platform, which includes a µP running Linux and will, in future, include a nonlinear CIS. Using this approach, our design can achieve hard real-time operation. At present, we simulated the nonlinear CIS using binary data loaded from a file, which consumed some of the bandwidth between the FPGA and µP. By freeing up this bandwidth, in future, we would be able to achieve the full 30 fps required for HD video.

We implemented a new circuit to integrate the ISPr to the µP via DMA. The main interface uses an AXI4-Stream bus and not video DMA (VDMA), because our goal was not a video processing pipeline with the end point being a display. Instead, our goal was to send the data to the µP for communication over a network, i.e., an edge computing case study.

Even taking simple modes of operation for the DMA, and AXI4-Stream for the protocol, a heterogeneous system requires mastery of multiple disciplines of knowledge. Working with embedded Linux provided the advantages of its networking, open source tools, and standard OS functions. However, any new interface between the FPGA and µP has to be recognized by the Linux kernel. This means editing the Linux device tree, configuring the Linux kernel to recompile, and possibly, for better performance, creating a device driver in the kernel space. All of these tasks were investigated here in order to fully support HD video.

The proposed design flow is especially suited for future edge computing applications involving HDR imaging and computer vision. In addition to substantial unused logic and memory available on the FPGA, the µP was used here, beyond the ISR software, only to connect to a network and serve simple web content. Its dual-core ARM processor may be used for additional processing, by leveraging open-source frameworks for Linux.

Chapter 5

Conclusion

In this chapter, we summarize the novelties and achievements of this research and envision future work. All contributions are grouped into each of the major thesis parts: on fixed pattern noise (FPN) correction and salt-and-pepper noise (SPN) filtering; on a histogram-based tone mapping operator (TMO); and on reconfigurable system-on-chip (SoC) interfacing. We foresee our future work involving research with an actual nonlinear CMOS image sensor (CIS). This includes leveraging the capabilities of our reconfigurable SoC to explore the suitability of the nonlinear imaging system for computer vision applications.

5.1 Summary and Contributions

This research has proposed generic circuit methods for FPN correction, SPN filtering, and tone mapping, as well as an image signal processor (ISPr) for a nonlinear CIS. Our FPN correction gives the option of generating different circuits based on the polynomial degree. Our TMO is a modular framework that can be customized for other histogram-based approaches. Providing reconfigurable circuits maximizes the use of field-programmable gate arrays (FPGAs), allowing the ISPr to be easily targeted to specific applications. These digital circuits were characterized in terms of the utilization of FPGA resources, power consumption, and maximum frequency. However, we go further than FPGAs alone and present methods of working with reconfigurable SoCs where an FPGA and microprocessor (µP) are integrated on the same silicon, expanding the computing power via a heterogeneous platform.

A novelty of this work is the creation of efficient generic circuits, which were based on previously designed algorithms. Ours were efficient solutions because we were able to deploy them in low-cost FPGAs without sacrificing performance. We were able to process high definition (HD) video formats. Other researchers in this field have also taken the approach of investigating novel digital circuits for FPGAs based on previously published algorithms.

5.1.1 FPN Correction and SPN Filtering

Using a nonlinear CIS to expand the dynamic range (DR) comes with a more complicated response curve to correct. This work has presented a new class of digital circuits capable of correcting the FPN. The recursive digital circuit based on polynomials is flexible and robust, but it does not compromise the performance since it used only 2.06% of the available logic and 0.01% of the available memory, in a low-cost FPGA, for full HD (FHD) video.

The use of a hardware description language (HDL) allowed the insertion of an extra level of abstraction, in the normal standard design flow of FPGAs, where a high-level scripting language, capable of processing text files, generates the circuit based on a previously coded design template and user parameters. This new methodology was extended for an automatic *bit true* validation process, where the generated circuit is compared to the algorithm implemented in software and every single pixel of both outputs matches 100%.

As the focus of this research has always been to develop a generic solution, a comparison between the two major FPGA manufacturers was made by reporting the results for both FPGA platforms. It also demonstrates that the solution is portable since it is independent of the architectural details of one particular FPGA. Only a generic digital circuit can be applied to a variety of problems, thus making it a class of circuits and not a single solution.

After the FPN is corrected, some pixels are stuck at a low or high logic level, in a manner that varies dynamically with light level, producing characteristic SPN that can be filtered with a median filter. An SPN filter is required, therefore, and this research presented a pipelined circuit to buffer the minimal amount of pixels to implement the median filter. The circuit has three stages where circuit diagrams and accompanying text explain how to implement them. The proposed design can also be deployed in a low-cost FPGA.

The three stages of the SPN filter are: a first-in first-out (FIFO) memory to buffer only two lines of the image and still be able to assemble a complete filter kernel; a router to modify the FIFO output and not waste any pixels by changing the shape of the kernel at the corners and borders; and a median filter that always processes five pixels.

Our SPN filter consumes 7.17% of the available logic and 24.7% of the available memory, in a low-cost FPGA, for FHD video. It was also efficient compared to an FPGA implementation of a median filter described in the literature.

5.1.2 Histogram-Based Tone Mapping

In a review paper, Eilertsen *et al.* [14] reported that test subjects, i.e., human participants in a study, preferred global TMOs with no visual artefacts. Of the top three most-preferred TMOs, two of them are based on histograms. With this motivation, this work has presented a digital circuit framework of how to implement histogram-based TMOs in hard real time.

Because the scope of this thesis is not software solutions, but digital circuits, a background algorithm developed by Li *et al.* [35] was leveraged. Computations performed by the algorithm were restructured to support hard real-time operation, where the novel circuits operate synchronous to a clock and in parallel fashion. While one circuit module maps the incoming pixels with the TMO stored in memory, another circuit module updates the TMO. The operator is global, but adapts itself according to the received video frames.

The generality of our design is illustrated by how it was factored into two major parts, i.e., a module for the base histograms and a module for the tone mapping. Such an approach facilitates the replacement of the tone mapping module to realize a different histogram-based approach. The base histograms module encapsulates the computation of the histogram with low-pass filtering, which is desirable to avoid flickering artefacts. Sub-parts of the tone mapping module may also be reused in another solution. One sub-part computes the mapping function and another applies the map to the video.

In order to achieve video rates at HD resolutions, the algorithm was modified to avoid division and to use multiplication and bit shifts instead. This strategy saved logic resources and increased overall performance, still allowing this complex circuit to be implemented in a low-cost FPGA.

Even though TMOs are a well-researched topic, not all of the examples found in the literature are designed for hard real time, or are able to achieve video rates at FHD resolution. Also, previous TMOs designed for real-time scenarios have presented visual artefacts, when using a local operator, or limited contrast, when using a global operator, due to the approach taken. Either that or they require a high-end FPGA, due to high circuit complexity.

By presenting results for circuits scaled from thousands to millions of pixels, we show how robust and generic the proposed method is, with the right adaptation of internal buses and memory sizes to accommodate such scaling.

5.1.3 Reconfigurable System-on-Chip

Moore's law is dying but digital system requirements continue to scale. They are growing with the growth in artificial intelligence (AI), machine learning, and video processing applications.

Doug Burger, a Distinguished Engineer from Microsoft, said in a conference: "... to support live AI services with very low response times at large scale, with great efficiency – better than central processing units (CPUs), we've made a major investment in FPGAs" [9]. CPUs have reached the clock speed limit due to heat dissipation issues, and they cannot provide much more parallelism. Julien Simon, Principal Evangelist at Amazon AWS, also justified the use of FPGAs as a better alternative to graphics processing units (GPUs) regarding power consumption and efficiency [47].

Acknowledging that they stand to benefit from what they say and do, it is nonetheless worthwhile to consider the words and actions of leading FPGA vendors. For example, Xilinx President and CEO Victor Peng said, at the XDF 2018 Keynote, because of "... the explosion of data, ... the emergence of

AI across all platforms, and the disruption of the end of Moore's Law, people have to deliver products to market faster than ever and to scale at world level you need systems that are adaptable." He was saying this to promote FPGAs and their reconfigurability. Also noteworthy is Intel's acquisition of Altera, which was aimed at agility and customization.

New approaches to increase computing performance have included a shift toward heterogeneous computing, where an SoC with an FPGA and a µP on the same silicon is one of the architectures available, an architecture that is explored in this work. A case study of a hard real-time ISPr based on a novel methodology encompassing the entire cycle of SoC design is presented. The design and interfacing of the ISPr, intended for a nonlinear CIS, was investigated, including FPN correction, SPN filtering, and a TMO.

In contrast to typical hardware acceleration, our approach to the use of the reconfigurable SoC is innovative. By making the FPGA the master of the system, we describe how to achieve high processing speeds in hard real time via interrupts, from the FPGA to the µP, and direct memory access (DMA). An experiment where the dual-core µP was made periodically busy was conducted to show the system working with no latency caused by the busy µP.

Due to the complexity of heterogeneous systems, this work investigated two design flows to work with a reconfigurable SoC. The advantages and disadvantages were presented to support other researchers who want to use the same technology. For example, one of the difficulties of working with the reconfigurable SoC is the interface between the µP and FPGA. This work explains how to handle such a technical difficulty and how to debug possible issues. As a result, this work will help the community to understand the design challenges and opportunities when exploiting heterogeneous computing architectures.

5.2 Future Work

Here, we address some ways in which this research may be continued. First, we focus on the hardware, i.e., how to integrate a nonlinear CIS with the ISPr to investigate imaging capabilities of the resulting system. The second section is dedicated to an application that could take advantage of such a nonlinear complementary metal-oxide-semiconductor (CMOS) imaging system. It discusses an end application envisioned for this design, how to commercialize components of the design, and finally how the design can be improved.

5.2.1 Nonlinear CMOS Imaging System

For this thesis research, we did not use an actual nonlinear CIS. Instead, we used experimental data, from a small format logarithmic (log) sensor, to simulate a nonlinear CIS from small to large formats. Because our focus was on the ISPr, it was a convenient yet appropriate strategy to validate the proposed circuits and interfaces, while demonstrating scalability.

With an actual CIS, additional investigations would be possible. For example, in a publication entitled "Toward a digital camera to rival the human eye," an SPIE Top Download for two months, Skorka and Joseph [48] demonstrated the difficulty of doing exactly that. One avenue they considered was the nonlinear CIS and they made a case, using different words, that low-power image signal processing (ISPg) in hard real time was a limiting factor.

To rival human vision, or even to significantly surpass it, one avenue may be to investigate a nonlinear CMOS imaging system comprised of a nonlinear CIS, such as a log or linear-logarithmic (linlog) sensor, and a corresponding ISPr. Outcomes of this thesis, therefore, facilitate such future work.

When the system is updated with the actual CIS additional variables may be investigated, including ones, such as temperature, that drift according to environmental conditions. The polynomial-based FPN correction would have to be extended to take into account such difficulties.

Some development work will be needed to facilitate such research work. For example, the FPGA-based system controller, discussed in Chapter 4, would have to set the row and column addresses of the CIS, and provide other control signals, to read pixel values sequentially. In addition, the initial configuration of the actual CIS would also have to be done by the controller.

Currently, when transferring data from a Secure Digital (SD) card to the

FPGA, a process controlled by the FPGA, the simulated pixel values and the FPN correction coefficients were packed together, resulting in 64-bit words, i.e., 16 bits for each pixel value and 48 bits for the corresponding coefficients. When the pixel values come from an actual CIS, the FPGA will require only the coefficients from a file stored on the SD card. Although this change requires less bandwidth for the data transfer from the SD card to the µP, data that would actually be cached in random-access memory (RAM) by the operating system (OS), a surmountable challenge will be to stream a 48-bit signal over a 64-bit DMA channel.

The multiple clock signals that may be required to solve the above challenge would require synchronization circuits to handle data crossing from one clock domain to another. One solution may be to use a FIFO with different clock signals for read and write, as explained by Cummings [12]. In addition, the FPGA-based controller will have to synchronize the pixel values, coming from the CIS, with the FPN correction coefficients, coming via a DMA channel from cached data on the µP side of the SoC.

Ultimately the reconfigurable SoC may be upgraded as well. A more recent component, with better technology, would increase the overall performance of the system, and leave additional resources to implement other ISPg operations using digital circuit methods. Such upgrading could significantly increase the maximum frame rate, or allow higher image resolutions to be processed at the same frame rate.

5.2.2 Computer Vision Application

A currently growing market that requires better quality cameras is the automotive industry. With the advent of self-driving cars, digital cameras increasingly require wide DR (WDR) capabilities to deal with the high dynamic range (HDR) luminances, from dark tunnels to bright sunshine, present in the real world. They must also do so with low latency, for rapid responses to rapidly changing stimuli. All these challenging requirements make it worthwhile to investigate the use of a nonlinear CMOS imaging system, built with a reconfigurable SoC, in such a computer vision application.

One of the advantages of working with the reconfigurable SoC platform, especially with Linux running on the µP side, is the possibility of incorporating open-source computer vision frameworks, such as OpenCV or Caffe. Use of these application program interfaces (APIs) would make it easy to add back-end sophistication to the imaging system. For example, Caffe is a deep learning framework. Although we did not test any of these APIs, using Petalinux with the horizontal methodology, as described in Chapter 4, seems to be the better design flow option for system management and control.

The proposed application would take advantage of the FPGA for low-level low-latency pixel-level ISPg, leaving the dual-core µP, running Linux with pre-configured open-source software, to continue the high-level processing. For example, the FPGA would perform the correction, filtering, and mapping operations addressed in this thesis, while the µP would perform object detection and classification. Whereas the FPGA would implement fixed-point calculations efficiently with pipelined circuits, the µP would implement floating-point calculations with software developed efficiently.

Future work could include the development of intellectual property (IP) cores to help commercialize, and further disseminate, thesis innovations. Components may be delivered via the Amazon AWS platform, which is already set up to share such IP. One aspect of this work is the use of generic methods. Circuits that were designed for FPN correction, SPN filtering, and histogram-based tone mapping may be adapted for other uses, which our IP cores could facilitate. For example, with the right parameters, the FPN correction and SPN filtering may be readily applied to a linlog sensor. Also, if we share the TMO as two IP cores, a third party could integrate, for example, our base histograms module with their own tone mapping module.

References

[1] M. Akil, T. Grandpierre, and L. Perroton, “Real-time dynamic tone-mapping operator on gpu,” *Journal of Real-Time Image Processing*, vol. 7, no. 3, pp. 165–172, Sep. 2012, ISSN: 1861-8219. DOI: `10.1007/s11554-011-0196-7`. [Online]. Available: `https://doi.org/10.1007/s11554-011-0196-7`.

[2] A. O. Akyüz, “High dynamic range imaging pipeline on the gpu,” *Journal of Real-Time Image Processing*, vol. 10, no. 2, pp. 273–287, Jun. 2015, ISSN: 1861-8219. DOI: `10.1007/s11554-012-0270-9`. [Online]. Available: `https://doi.org/10.1007/s11554-012-0270-9`.

[3] Altera, *Intel FPGAs*, `https://www.altera.com/products/fpga/overview.html`, Oct. 2017. [Online]. Available: `https://www.altera.com/products/fpga/overview.html`.

[4] S. Asano, T. Maruyama, and Y. Yamaguchi, “Performance comparison of fpga, gpu and cpu in image processing,” in *2009 International Conference on Field Programmable Logic and Applications*, Aug. 2009, pp. 126–131. DOI: `10.1109/FPL.2009.5272532`.

[5] *Axi dma v7.1*, PG021, Vivado Design Suite, Xilinx, Apr. 2018.

[6] F. Banterle, A. Artusi, E. Sikudova, P. Ledda, T. Bashford-Rogers, A. Chalmers, and M. Bloj, “Mixing tone mapping operators on the gpu by differential zone mapping based on psychophysical experiments,” *Signal Processing: Image Communication*, vol. 48, pp. 50–62, 2016, ISSN: 0923-5965. DOI: `https://doi.org/10.1016/j.image.2016.09.004`. [Online]. Available: `http://www.sciencedirect.com/science/article/pii/S0923596516301308`.

[7] K. Benkrid, A. Akoglu, C. Ling, Y. Song, Y. Liu, and X. Tian, “High performance biological pairwise sequence alignment: Fpga versus gpu versus cell be versus gpp,” *Int. J. Reconfig. Comput.*, vol. 2012, 7:7–7:7, Jan. 2012, ISSN: 1687-7195. DOI: `10.1155/2012/752910`. [Online]. Available: `http://dx.doi.org/10.1155/2012/752910`.

[8] Berten, “GPU vs FPGA Performance Comparison,” *White Paper*, vol. BWP001, no. v1.0, 2016.

[9] D. Burger, *Transitioning from the Era of Multicore to the Era of Specialization*, `https://youtu.be/vo1qHEqLK4c`, Oct. 2014. [Online]. Available: `https://youtu.be/vo1qHEqLK4c`.

[10] P. Cambou and J.-L. Jaffard, "Status of the CMOS Image Sensors Industry," YOLE Developpment, Tech. Rep., 2015.

[11] B. Choubey and S. Collins, "Models for Pixels With Wide-Dynamic-Range Combined Linear and Logarithmic Response," *IEEE Sensors Journal*, vol. 7, no. 7, pp. 1066–72, Jul. 2007. [Online]. Available: `https://doi.org/10.1109/JSEN.2007.895959`.

[12] C. Cummings, "Synthesis and Scripting Techniques for Designing Multi-Asynchronous Clock Designs," *SNUG*, vol. 1.1, no. 3, Mar. 2001.

[13] J. Duan, M. Bressan, C. Dance, and G. Qiu, "Tone-mapping High Dynamic Range Images by Novel Histogram Adjustment," *Pattern Recogn.*, vol. 43, no. 5, pp. 1847–1862, May 2010, ISSN: 0031-3203. DOI: `10.1016/j.patcog.2009.12.006`. [Online]. Available: `http://dx.doi.org/10.1016/j.patcog.2009.12.006`.

[14] G. Eilertsen, R. Wanat, R. K. Mantiuk, and J. Unger, "Evaluation of Tone Mapping Operators for HDR-Video.," *Comput. Graph. Forum*, vol. 32, no. 7, pp. 275–284, 2013. [Online]. Available: `http://dblp.uni-trier.de/db/journals/cgf/cgf32.html#EilertsenWMU13`.

[15] A. El Gamal and H. Eltoukhy, "CMOS image sensors," *IEEE Circuits and Devices Magazine*, vol. 21, no. 3, pp. 6–20, May 2005, ISSN: 8755-3996. [Online]. Available: `http://dx.doi.org/10.1109/MCD.2005.1438751`.

[16] *Embedded Design Hub - PetaLinux Tools*, `https://www.xilinx.com/support/documentation-navigation/design-hubs/dh0016-petalinux-tools-hub.html`, Accessed: 2019-02-28.

[17] D. F., M. K., A. T., and C. N., "Adaptive logarithmic mapping for displaying high contrast scenes," *Computer Graphics Forum*, vol. 22, no. 3, pp. 419–426, DOI: `10.1111/1467-8659.00689`. eprint: `https://onlinelibrary.wiley.com/doi/pdf/10.1111/1467-8659.00689`. [Online]. Available: `https://onlinelibrary.wiley.com/doi/abs/10.1111/1467-8659.00689`.

[18] R. Fontaine, "The State-of-the-Art of Mainstream CMOS Image Sensors," in *Proceedings of the International Image Sensors Workshop*, Jun. 2015, pp. 1–4.

[19] F. Hassan and J. Carletta, "An FPGA-based architecture for a local tone-mapping operator," *Real-Time Image Processing*, vol. 2, no. 2, pp. 293–308, 2007.

[20] B. Hoefflinger, *High-Dynamic-Range (HDR) Vision*, ser. Advanced Microelectronics. Springer, 2007, vol. 26, ISBN: 978-3-540-44433-6. [Online]. Available: `https://link.springer.com/book/10.1007/978-3-540-44433-6`.

[21] P. Irawan, J. A. Ferwerda, and S. R. Marschner, "Perceptually based tone mapping of high dynamic range image streams," in *Proceedings of the Sixteenth Eurographics Conference on Rendering Techniques*, ser. EGSR '05, Konstanz, Germany: Eurographics Association, 2005, pp. 231–242, ISBN: 3-905673-23-1. DOI: `10.2312/EGWR/EGSR05/231-242`. [Online]. Available: `http://dx.doi.org/10.2312/EGWR/EGSR05/231-242`.

[22] D. Joseph and S. Collins, "Modeling, calibration, and correction of nonlinear illumination-dependent fixed pattern noise in logarithmic CMOS image sensors," *IEEE Transactions on Instrumentation and Measurement*, vol. 51, no. 5, pp. 996–1001, Oct. 2002, ISSN: 0018-9456. [Online]. Available: `http://dx.doi.org/10.1109/TIM.2002.807803`.

[23] D. Joseph and S. Collins, "Modeling, Calibration, and Correction of Nonlinear Illumination-Dependent Fixed Pattern Noise In Logarithmic CMOS Image Sensors," *IEEE Transactions on Instrumentation and Measurement*, vol. 51, no. 5, pp. 996–1001, 2002.

[24] C. Jung and T. Sun, "Optimized perceptual tone mapping for contrast enhancement of images," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 27, no. 6, pp. 1161–1170, Jun. 2017, ISSN: 1051-8215. DOI: `10.1109/TCSVT.2016.2527339`.

[25] T. Kalb, L. Kalms, D. Göhringer, C. Pons, F. Marty, A. Muddukrishna, M. Jahre, P. G. Kjeldsberg, B. Ruf, T. Schuchert, I. Tchouchenkov, C. Ehrenstrahle, F. Christensen, A. Paolillo, C. Lemer, G. Bernard, F. Duhem, and P. Millet, "Tulipp: Towards ubiquitous low-power image processing platforms," in *2016 International Conference on Embedded Computer Systems: Architectures, Modeling and Simulation (SAMOS)*, Jul. 2016, pp. 306–311. DOI: `10.1109/SAMOS.2016.7818363`.

[26] S. Kavadias, B. Dierickx, D. Scheffer, A. Alaerts, D. Uwaerts, and J. Bogaerts, "A logarithmic response CMOS image sensor with on-chip calibration," *IEEE Journal of Solid-State Circuits*, vol. 35, no. 8, pp. 1146–52, Aug. 2000, ISSN: 0018-9200. [Online]. Available: `http://dx.doi.org/10.1109/4.859503`.

[27] I. R. Khan, S. Rahardja, M. M. Khan, M. M. Movania, and F. Abed, "A tone-mapping technique based on histogram using a sensitivity model of the human visual system," *IEEE Transactions on Industrial Electronics*, vol. 65, no. 4, pp. 3469–3479, Apr. 2018, ISSN: 0278-0046. DOI: `10.1109/TIE.2017.2760247`.

[28] T.-C. Kim, "Wide Dynamic Range Technologies: For mobile imaging sensor systems," *IEEE Consumer Electronics Magazine*, vol. 3, no. 2, pp. 30–35, Apr. 2014, ISSN: 2162-2248. [Online]. Available: `http://dx.doi.org/10.1109/MCE.2014.2298072`.

[29] J. Kronander, S. Gustavson, G. Bonnet, and J. Unger, "Unified hdr reconstruction from raw cfa data," in *IEEE International Conference on Computational Photography (ICCP)*, Apr. 2013, pp. 1–9. DOI: `10.1109/ICCPhot.2013.6528315`.

[30] P. J. Lapray, B. Heyrman, M. RossÃ©, and D. Ginhac, "A 1.3 megapixel fpga-based smart camera for high dynamic range real time video," in *Distributed Smart Cameras (ICDSC), 2013 Seventh International Conference on*, Oct. 2013, pp. 1–6. DOI: `10.1109/ICDSC.2013.6778230`.

[31] G. W. Larson, H. Rushmeier, and C. Piatko, "A Visibility Matching Tone Reproduction Operator for High Dynamic Range Scenes," *IEEE Transactions on Visualization and Computer Graphics*, vol. 3, no. 4, pp. 291–306, Oct. 1997.

[32] T. Latha and M. Sasikumar, "A Novel Non-linear Transform Based Image Restoration for Removing Three Kinds of Noises in Images," *Journal of the Institution of Engineers (India): Series B*, vol. 96, no. 1, pp. 17–26, Mar. 2015, ISSN: 2250-2106. [Online]. Available: `http://dx.doi.org/10.1007/s40031-014-0123-y`.

[33] E. A. Lee, B. Hartmann, J. Kubiatowicz, T. S. Rosing, J. Wawrzynek, D. Wessel, J. Rabaey, K. Pister, A. Sangiovanni-Vincentelli, S. A. Seshia, D. Blaauw, P. Dutta, K. Fu, C. Guestrin, B. Taskar, R. Jafari, D. Jones, V. Kumar, R. Mangharam, G. J. Pappas, R. M. Murray, and A. Rowe, "The swarm at the edge of the cloud," *IEEE Design Test*, vol. 31, no. 3, pp. 8–20, Jun. 2014, ISSN: 2168-2356. DOI: `10.1109/MDAT.2014.2314600`.

[34] J. Li, A. Mahmoodi, and D. Joseph, "Using Polynomials to Simplify Fixed Pattern Noise and Photometric Correction of Logarithmic CMOS Image Sensors," *Sensors*, vol. 15, no. 10, pp. 26 331–52, Oct. 2015, ISSN: 1424-8220. [Online]. Available: `http://dx.doi.org/10.3390/s151026331`.

[35] J. Li, O. Skorka, K. Ranaweera, and D. Joseph, "Novel Real-Time Tone Mapping Operator for Noisy Logarithmic CMOS Image Sensors," *Journal of Imaging Science and Technology*, vol. 60, no. 2, pp. 020404-1–13, Mar. 2016.

[36] R. C. H. Lo, S. Mann, J. Huang, V. Rampersad, and T. Ai, "High dynamic range (hdr) video image processing for digital glass," in *Proceedings of the 20th ACM International Conference on Multimedia*, ser. MM '12, Nara, Japan: ACM, 2012, pp. 1477–1480, ISBN: 978-1-4503-1089-5. DOI: `10.1145/2393347.2396525`. [Online]. Available: `http://doi.acm.org/10.1145/2393347.2396525`.

[37] A. Mahmoodi, J. Li, and D. Joseph, "Digital Pixel Sensor Array with Logarithmic Delta-Sigma Architecture," *Sensors*, vol. 13, no. 8, pp. 10 765–82, Aug. 2013, ISSN: 1424-8220. [Online]. Available: `http://dx.doi.org/10.3390/s130810765`.

[38] S. Mann, R. C. H. Lo, K. Ovtcharov, S. Gu, D. Dai, C. Ngan, and T. Ai, "Realtime hdr (high dynamic range) video for eyetap wearable computers, fpga-based seeing aids, and glasseyes (eyetaps)," in *2012 25th IEEE Canadian Conference on Electrical and Computer Engineering (CCECE)*, Apr. 2012, pp. 1–6. DOI: `10.1109/CCECE.2012.6335012`.

[39] 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 Transactions on Circuits and Systems I*, vol. 61, no. 10, pp. 2825–33, Oct. 2014, ISSN: 1549-8328. [Online]. Available: `http://dx.doi.org/10.1109/TCSI.2014.2327284`.

[40] M. Nascimento, J. Li, and D. Joseph, "Digital Circuit Methods to Correct and Filter Noise of Nonlinear CMOS Image Sensors," *Journal of Imaging Science and Technology*, vol. 62, no. 6, 2018, ISSN: 1062-3701. DOI: `doi:10.2352/J.ImagingSci.Technol.2018.62.6.060404`. [Online]. Available: `https://www.ingentaconnect.com/content/ist/jist/2018/00000062/00000006/art00005`.

[41] C. Ofili, S. Glozman, and O. Yadid-Pecht, "Hardware implementation of an automatic rendering tone mapping algorithm for a wide dynamic range display," *Journal of Low Power Electronics and Applications*, vol. 3, no. 4, pp. 337–367, 2013, ISSN: 2079-9268. DOI: `10.3390/jlpea3040337`. [Online]. Available: `http://www.mdpi.com/2079-9268/3/4/337`.

[42] V. Popovic, E. Pignat, and Y. Leblebici, "Performance optimization and fpga implementation of real-time tone mapping," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 61, no. 10, pp. 803–807, Oct. 2014, ISSN: 1549-7747. DOI: `10.1109/TCSII.2014.2345306`.

[43] K. Pulli, *Camera Processing Pipeline*, `https://web.stanford.edu/class/cs231m/lectures/lecture-11-camera-isp.pdf`, May 2015. [Online]. Available: `https://web.stanford.edu/class/cs231m/lectures/lecture-11-camera-isp.pdf`.

[44] E. Reinhard, G. Ward, S. Pattanaik, and P. Debevec, *High Dynamic Range Imaging: Acquisition, Display, and Image-Based Lighting (The Morgan Kaufmann Series in Computer Graphics)*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2005, ISBN: 0125852630.

[45] A. M. Reza, "Realization of the contrast limited adaptive histogram equalization (clahe) for real-time image enhancement," *J. VLSI Signal Process. Syst.*, vol. 38, no. 1, pp. 35–44, Aug. 2004, ISSN: 0922-5773.

DOI: 10.1023/B:VLSI.0000028532.53893.82. [Online]. Available: http://dx.doi.org/10.1023/B:VLSI.0000028532.53893.82.

[46] W. Shi and S. Dustdar, "The promise of edge computing," *Computer*, vol. 49, no. 5, pp. 78–81, May 2016, ISSN: 0018-9162. DOI: 10.1109/MC.2016.145.

[47] J. Simon, *FPGAs in the cloud ?* https://youtu.be/SFyW2HVimiU, Nov. 2017. [Online]. Available: https://youtu.be/SFyW2HVimiU.

[48] O. Skorka and D. Joseph, "Toward a digital camera to rival the human eye," *Journal of Electronic Imaging*, vol. 20, no. 3, pp. 033009 1–18, Aug. 2011. [Online]. Available: https://doi.org/10.1117/1.3611015.

[49] O. Skorka, J. Li, and D. Joseph, "Nonlinear Digital Pixels: Idea to Innovation (Phase I)," University of Alberta, Tech. Rep., 2013.

[50] 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 Journal of Solid-State Circuits*, vol. 41, no. 9, pp. 2095–106, Sep. 2006, ISSN: 0018-9200. [Online]. Available: http://dx.doi.org/10.1109/JSSC.2006.880613.

[51] T. Suzuki, "Challenges of Image-Sensor DEvelopment," *IEEE International Solid-State Circuits Conference*, vol. ISSCC 2010, no. Session 1, pp. 27–30, 2010.

[52] S. Trimberger, "A Reprogrammable Gate Array and Applications," *Proceedings of the IEEE*, vol. 81, no. 0018-9219, 2013.

[53] S. Trimberger, "Three Ages of FPGAs: A Retrospective on the First Thirty Years of FPGA Technology," *Proceedings of the IEE*, vol. 103, no. 3, Mar. 2015.

[54] R. Ureña, P. Martínez-Cañada, J. M. Gómez-López, C. Morillas, and F. Pelayo, "Real-time tone mapping on gpu and fpga," *EURASIP Journal on Image and Video Processing*, vol. 2012, no. 1, p. 1, Feb. 2012, ISSN: 1687-5281. DOI: 10.1186/1687-5281-2012-1. [Online]. Available: https://doi.org/10.1186/1687-5281-2012-1.

[55] X. Wu, "A linear programming approach for optimal contrast-tone mapping," *Trans. Img. Proc.*, vol. 20, no. 5, pp. 1262–1272, May 2011, ISSN: 1057-7149. DOI: 10.1109/TIP.2010.2092438. [Online]. Available: http://dx.doi.org/10.1109/TIP.2010.2092438.

[56] *www.myirtech.com*, http://www.myirtech.com/list.asp?id=502, Accessed: 2019-02-04.

[57] Xilinx, *All Programmable FPGAs and 3D ICs*, https://www.xilinx.com/products/silicon-devices/fpga.html, Oct. 2017. [Online]. Available: https://www.xilinx.com/products/silicon-devices/fpga.html.

[58] ——, *High Performance Computing and Data Storage*, `https://www.xilinx.com/content/xilinx/en/applications/high-performance-computing.html`, [Online; accessed 27-May-2017], 2017.