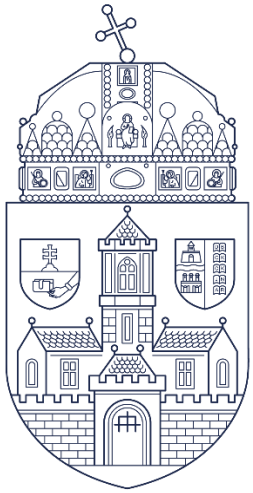
Óbuda University

PhD Thesis



High Performance Image Sensor Processing using Field Programmable Gate Arrays

Gábor Szedő Becker

Supervisor:

Róbert Lovas, PhD, habil.

Doctoral School of Applied Informatics and Applied Mathematics

Budapest, 2023.

Final Examination Committee

Chair:
Prof. Dr. József Tar, PhD, habil, DSc

Members:
Dr. Zoltán Vámossy, PhD
Dr. Antal Hiba, PhD (SZTAKI)

Date of Final Examination:
April 11, 2022

Full Committee of Public Defense

Opponents:
Dr. Zoltán Vámossy, PhD
Dr. Zoltán Nagy, PhD (PPKE)

Chair:
Prof. Dr. József Tar, PhD, habil, DSc

Secretary:
Dr. Eszter Kail, PhD

Members:
Prof. Dr. Sándor Szénási, PhD, habil
Dr. Ákos Odry, PhD
Dr. Antal Hiba, PhD (SZTAKI)

Date of Public Defense:
March 6, 2023

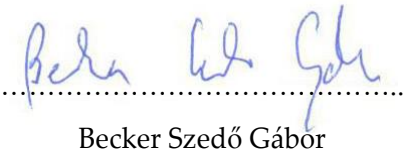
Declaration

I, the undersigned, **Gabor Szedo Becker**, hereby state and declare that this Ph.D. thesis represents my own work and is the result of my own original research. I only used the sources listed in the references. All parts taken from other works, either as word for word citation or rewritten keeping the original meaning, have been unambiguously marked, and reference to the source was included.

Gabor Szedo Becker

Nyilatkozat

Alulírott **Becker Szedő Gábor** kijelentem, hogy ez a doktori értekezés a saját munkámat mutatja be, és a saját eredeti kutatásom eredménye. Csak a hivatkozásokban fel- sorolt forrásokat használtam fel, minden más munkából származó rész, a szó szerinti, vagy az eredeti jelentést megtartó átiratok egyértelműen jelölve, és forrás hivatkozva lettek.



Becker Szedő Gábor

Contents

Acknowledgements

I would like to thank my doctoral supervisor *Róbert Lovas, PhD habil.* for his valuable support during my doctoral studies at the Óbuda University. His relentless encouragement, rigorous reviews and high scientific standards was a key factor to the successful completion of this work. I highly appreciate the support of *Gábor Kertész, PhD,* and *Ákos Hajnal, PhD*, who were supporting my research with guidance about the PhD program with practical advice, inspired me with their unyielding commitment to science, neural networks and AI.

I would also like to thank *Ted Babpty, PhD* for the opportunity and support during my research scholarship at Vanderbilt University. Ted supported my work in the research of FGPA implementation of FFTs both professionally and by securing research grants, which culminated in our paper accepted for an international conference on FPGAs.

I would like to express my gratitude to my late advisor, *Béla Fehér, PhD* , who introduced me to the world of programmable logic as an MSc student, and later allowed me to join the Department of Measurement and Instrumentation Systems at the Budapest University of Technology and Economics, as a graduate student. Béla introduced me to academia, provided guidance writing my first papers, and provided a professional background, inspiring environment, and opportunities to attend my first international conferences. Also at the department of Measurement and Information Systems, I would like to recognize the efforts of *Péter Szántó*. With Péter we ran labs, delivered courses, collaborated on projects, developed novel filter architectures, wrote papers and patents. Péter has been a stable source of professional advice, a great co-author and friend.

My commendation and gratitude are also due to Professor *Christopher Dick, PhD*, who offered me a position at Xilinx Inc, which provided access to cutting edge technology and collaboration with some of the best minds of Silicon Valley. While most for-profit corporations encourage capturing inventions and new scientific content in patents, Chris also found ways to publish our work at scientific conferences.

Special thanks to *Wilson Chung, PhD*, who encouraged me to lead the image sensor interface IP team at Xilinx, which was my first foray into the world of Image Signal Processing (ISP). Wilson is a pragmatic leader and talented architect with an excellent overview of contemporary silicon capabilities and detailed understanding of image and video processing algorithms.

Last, but not least, I owe special thanks to my family who put up with many weekends spent at home while I was working on the dissertation.

Köszönetnyilvánítás

Szeretnék köszönetet mondani doktori témavezetőmnek, *Dr. habil. Lovas Róbert*nek, az Óbudai Egyetemen végzett doktori tanulmányaim során nyújtott értékes segítségért. Róbert tudományos közreműködése, észrevételei, tanácsai, magas szintű szakmai vezetése nélkül ez a disszertáció nem készült volna el.

Nagyon nagyra becsülöm *Dr. Kertész Gábor* és *Dr. Hajnal Ákos* segítségét, a PhD programmal kapcsolatos gyakorlati tanácsaikat, közös kutatási, szakmai, és publikációs tevékenységüket, valamint lankadatlan elkötelezettségüket a számítástechnika, a neurális hálózatok, és a mesterséges intelligencia témakörében.

Meg szeretném köszönni *Dr. Ted Bapty* professzornak a lehetőséget, hogy kutatói munkatársként részt vehetettem a Vanderbilt Egyetem Szoftver Integrált Rendszerek tanszékének munkájában. Ez az egy év meghatározó volt az életemben. Ted pályázatainak és szakmai támogatásának köszönhetem kutatói munkám FFT struktúrák FPGA megvalósításával foglalkozó ágát, és az első közös cikkünket, amelyet befogadtak publikációra egy FPGA témájú tudományos konferenciára az Egyesült Államokban.

Hálával tartozom a tragikus körülmények között elhunyt *Dr. Fehér Béla* professzornak, aki még MSc hallgatóként bevezetett a programozható logikai áramkörök világába, és megadta a lehetőséget, hogy képviselhessem a Budapesti Műszaki Egyetemet Magyarország első FPGA programozási versenyén. Első helyezésünkért felvételt nyertem a Méréstechnika és Információs Rendszerek tanszék PhD programjára, és ezzel az akadémiai munkába. Béla vezetése alatt írtam meg első cikkeimet, neki köszönhetem az első konferencia előadásaimat, és az első nemzetközi konferencia publikációkat.

Szerzőtársaim közül szeretném kiemelni, és köszönetet mondani *Szántó Péternek*, akivel PhD hallgatóként együtt kutattunk, oktattunk, és vezettünk hallgatói laborokat. Péterrel sok közös projektünk, publikációnk és egy amerikai szabadalmunk is született.

Hasonlóan köszönöm és nagyra értékelem *Dr. Christopher Dick* professzor segítségét, aki a Melbourne-i Egyetemtől csatlakozott a Xilinx cég IP fejlesztő csoportjához, és tudományos munkám alapján pozíciót ajánlott a jelfeldolgozási csoportban. Chris-en keresztül bevezetést nyertem a legújabb FPGA technológiákba, és a Szilícium Völgy számos szakemberével ismerkedhettem meg közös kutatás-fejlesztési munkánk során.

Köszönettel tartozom *Dr. Wilson Chung*-nak, hogy bevont a Xilinx kép-, és video feldolgozó csoport alapításába, és rám bízta a szenzor interfész csoport vezetését. Ezzel nyertem bevezetést a CCD és CMOS szenzorok, valamint az ISP-k világába.

Végül, de nem utolsó sorban hálával tartozom családomnak, hogy kitartottak mellettem az évek során amíg ez a disszertáció megszülethetett.

Abstract

High Performance Image Sensor Processing using Field Programmable Gate Arrays

Image sensors are ubiquitous parts of everyday life, present in mobile phones, laptops, tablets, cars, Internet of Things (IoT) and security devices. Most camera products are prototyped using programmable logic. Low volume special cameras, such as Infrared (IR) imaging Focal Plane Arrays (FPAs) and high frame-rate industrial, scientific cameras use Field Programmable Gate Arrays (FPGAs) in the released version of the product as well.

Meanwhile FPGAs transformed from a uniform sea-of-gates architecture to a heterogenous multiprocessing System-on-a-Chip (SoC) platform, capable of implementing not only glue-logic but complex signal-, and image processing algorithms.

In Chapter 1, this dissertation first reviews recent progress in FPGA architecture and technology. Similarly, Chapter 2 introduces the function and architecture of Image Signal Processor (ISP) modules. The following chapters detail my contributions to new algorithms, architectures or methods on the field of ISP implementation on FPGAs, organized by topic matching thesis points.

Chapter 3 is devoted to uniformity correction of CMOS image sensors. I present my analysis of fixed-pattern noise and compare multiple practical approaches to reduce noise.

Chapter 4 describes a method to dynamically detect and interpolate pixel defects without access to a frame buffer.

Chapter 5 concentrates on image quality improvements, with focus on noise reduction. Median filters, a subclass of rank order filters, are widely used for noise reduction while preserving edges is signals and images. Optimizations of 1D/2D rank order filter structures for FPGA implementation are explored.

Chapter 6 describes my work on Automatic White Balance adjustment. Low-latency algorithms are provided for simple, robust control of sensor parameters based on image statistics collected on the fly.

Chapter 7 focuses on frequency domain filtering and efficient implementation of widely used image-, and signal processing functions: one, and Two-dimensional Fast Fourier Transforms. I propose optimizations in the storage and access sequencing of intermediate results, which reduces local BRAM allocation by 50%.

In Chapter 8 I propose a resonator based Direct Digital Synthesis (DDS) module, which allows further reduction in memory allocation.

Kivonat

Nagy teljesítményű szenzor képfeldolgozás FPGA áramkörökkel

A képfelvevő szenzorok mára szinte mindenütt jelen vannak. Mobiltelefonokban, laptopokban, gépjárművekben, biztonsági és IoT rendszerekben egyre több szenzor szolgáltat képeket, videókat. Ezen rendszerek többségét programozható logikai áramkörökkel fejlesztik. A kis-szériás, nagyfelbontású, ultra gyors kamerák, és speciális (pl. infravörös) kamerák kereskedelmi forgalomba került változatai is a Field Programmable Gate Array (FPGA) áramkörökre épülnek. Az elmúlt évtizedekben az FPGA áramkörök kezdeti homogén, sea-of-gates architektúráját kiegészítették lokális memóriákkal, jelfeldolgozó, és általános célú processzorokkal, számítástechnikai rendszereket (SoC) valósítva meg egyetlen chipen.

A disszertáció első fejezetében bemutatom az FPGA áramkörök alkotóelemeit, és a digitális jel-, és kép-feldolgozás szempontjából kritikus alapmodulok implementációját. A második fejezet a kamera előfeldolgozó (ISP) áramkörök moduljainak funkcióit, és ezek FPGA-n belüli megvalósítását mutatja be. A harmadik fejezet a CMOS szenzorok fix-mintázatú zaját, ennek hőmérséklet és paraméter függőségét elemzi, illetve vizsgálja a különböző kompenzációs stratégiák komplexitását és teljesítményét. A negyedik fejezetben kitérek a pixel hibák osztályozására, és megoldást mutatok a hibák dinamikus feltérképezésére és interpolációjára külső memória hozzáférés nélkül.

Az ötödik fejezet a zajelnyomást, ezen belül a nem-lineáris szűrők FPGA alapú optimális megvalósítását taglalja. A medián-, és rank-szűrők eredményesen használhatók kiugró minták szűrésére, élességromlás nélkül. A gyakorlatban jól alkalmazható, és FPGA megvalósításra optimalizált algoritmust adok egy-, és kétdimenziós rank szűrők megvalósítására.

A hatodik fejezet a fehéregyensúly valós idejű automatikus beállítására mutat algoritmust. Az ISP áramkör statisztikai moduljának jeleit egy kompakt, FPGA áramkörökön belül megvalósítható neurális hálózattal értékelve, gyors és a jelenlegi beágyazott megoldásoknál pontosabb megoldást mutatok be.

A hetedik fejezet a frekvencia tartományban végzett jel-, és képfeldolgozó algoritmusok alapját képező diszkrét gyors Fourier Transzformáció (FFT) FPGA optimalizációját vizsgálja. A javasolt megoldásokkal a transzformáció elvégzéséhez szükséges lokális memória felére csökkenthető, illetve a szükséges konstans faktorok valós idejű számításával a megvalósításhoz szükséges szilícium terület jelentősen csökkenthető.

A nyolcadik fejezet az FFT számításhoz szükséges komplex egységvektorok számítására mutat optimális megoldásokat FPGA áramkörökön belül.

Abbreviations & notations

In this study the following notations and abbreviations were used:

AE	Automatic Exposure adjustment
ASIC	Application Specific Integrated Circuit
ASSP	Application Specific Standard Parts
AWB	Automatic White Balance adjustment
BRAM	18-, or 36 kbit dual ported Block RAM primitive in FPGA devices
CCM	Color Correction Matrix
CFA	Color Filter Array
CMOS	Complementary Metal Oxide Semiconductor
CNN	Convolutional Neural Network
DCT	Discrete Cosine Transform
DDS	Direct Digital Synthesis
DIF	Decimation In Frequency
DIT	Decimation In Time
DPC	Defective Pixel Correction
DSNU	Dark Signal Non-Uniformity
DSP	Digital Signal Processing
FFC	Flat-Field Correction
FFT	Fast Fourier Transform
FPGA	Field Programmable Gate Array
FPR	Fixed-Point Representation
GPU	Graphical Processing Unit
HDL	Hardware Description Language
ISP	Image Signal Processor
IoT	Internet of Things
LUT	Look Up Table
MANR	Motion Adaptive Noise Reduction
MIMO	Multiple Input, Multiple Output
PCC	Pearson Correlation Coefficient
PRNU	Photon Response Non-Uniformity
QDDFS	Quadrature Direct Digital Frequency Synthesizer
SFDR	Spurious Free Dynamic Range
SISD	Single Instruction Single Data
SIMD	Single Instruction Multiple Data
SoC	System on a Chip
SNR	Signal to Noise Ratio

List of Figures

List of Tables

1 Introduction

Logic will get you from A to B.

Imagination will take you anywhere.

— Albert Einstein

1.1 Field Programmable Gate Arrays

A Field Programmable Gate Array (FPGA) is an array of configurable logic blocks (CLB) connected by programmable routing on a single silicon device [1]. The logic blocks contain a combinatorial unit, usually implemented as a small Static Random Access Memory (SRAM) component, a set of registers, logic gates and multiplexers to support typical computation tasks such as carry logic.

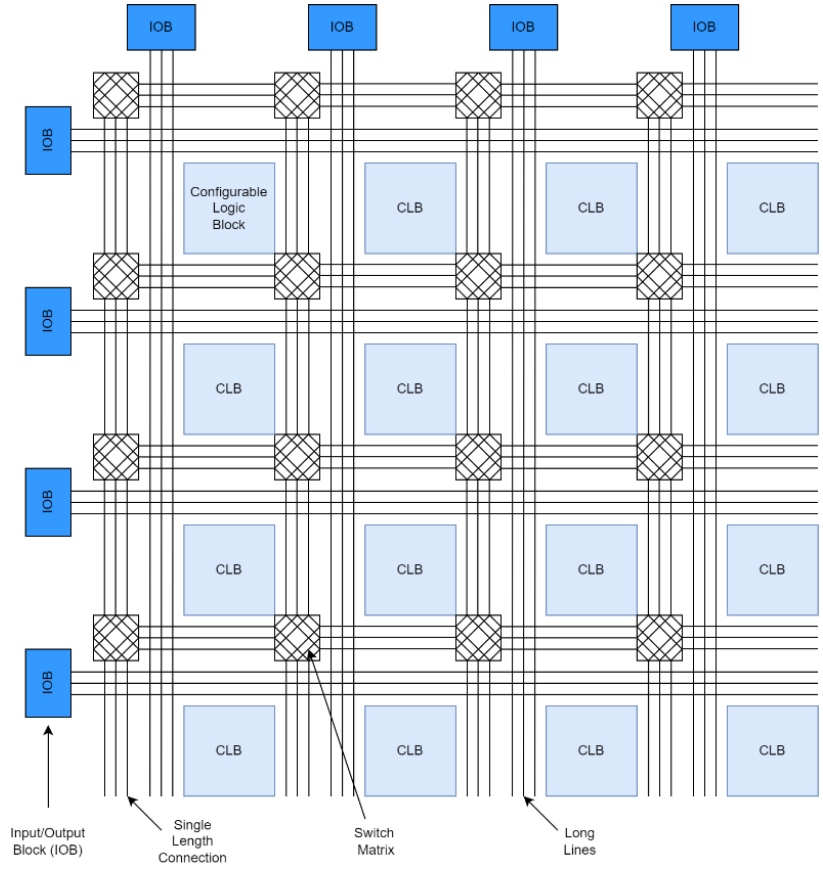


Figure 1: Simple Sea of Gates FPGA architecture.

Configuration of CLBs allow implementation of basic logic gates as well as more complex logic functions such as arithmetic operators, multiplexers, decoders, or state machines. Programmable, hierarchical routing via switch matrixes allows cells to communicate with each other and the Input/Output blocks which connect the programmable network to external components, memories, processors, etc. All operations take place simultaneously across the whole array, synchronized by dedicated, low-skew, high fanout clock routing to all cells. The architecture of a simple, Sea-of-Gates FPGA is illustrated on Figure 1.

1.1.1 Evolution and Structure

In the early 2000s, CLBs were combined to slices, and devices were segmented into an array of tiles. More complex blocks, such as 18kbit RAM blocks (BRAM), and 18x18 multipliers were embedded between the tiles. Next 18x18 block multipliers were augmented with wide accumulators to reduce the implementation footprint of typical DSP operations. In the past decade, processor cores, external memory interfaces, and hardened peripherals (I2C, SPI, UART) were embedded into the configurable array leading to the latest heterogenous multiprocessing System-on-a-Chip (SoC) platform [2],[3].

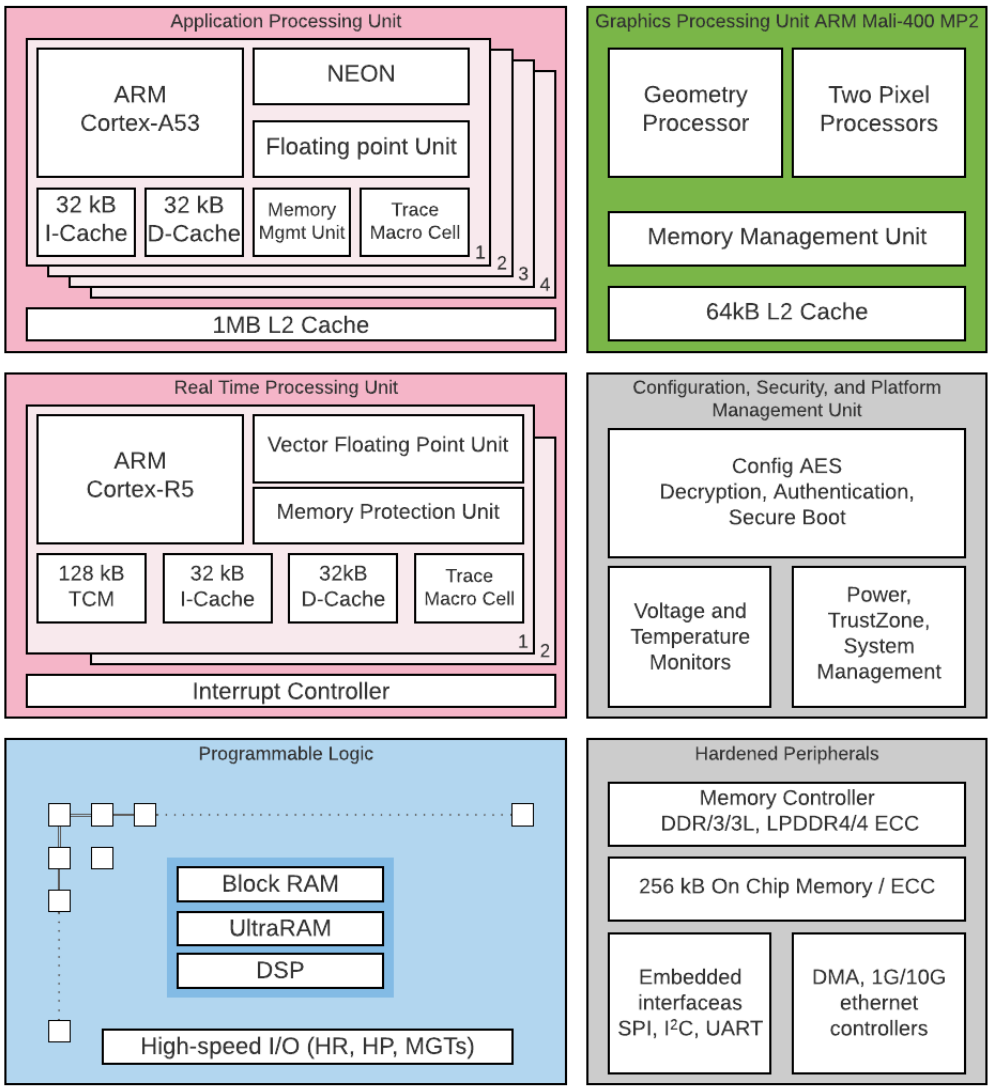


Figure 2: Example block diagram of a modern FPGA device[1].

The latest generation of FPGA devices, such as the Xilinx Zynq Ultrascale+ MPSoC (Figure 2), include multicore, heterogenous processing system (PS), GPUs, and various hardened functions, such as Memory and Network controllers, peripherals, such as I2C, SPI controllers, DMA and watchdog timers. Thousands of DSP engines with single precision floating point units with integrated hierarchical cache memories can implement complex, high performance video processing solutions. Beyond implementing SIMD or MIMD architectures, fine grain

[1] Source: adapted from https://www.xilinx.com/products/silicon-devices/soc/zynq-ultrascale-mpsoc.html

configurability enables FPGAs to implement custom DSP and image processing blocks, allowing multiple trade-offs between area, performance, and numerical precision.

Early use of FPGAs as a derivative of Complex Programmable Logic Devices (CPLD) was glue-logic between system components. With relentless miniaturization and specialization, modern devices have the capacity and features to implement complex machine vision algorithms [4],[5]. FPGAs are increasingly used in high performance computing applications either as the sole central computing element or as a coprocessor. As a dedicated peripheral, or custom, reconfigurable accelerator, FPGAs can execute digital signal processing (DSP) operations such as convolutions, pixel level image processing such as edge detection, or perform Convolutional Neural Network (CNN) inference. The latest generation of FPGAs can implement thousands of single precision floating point units, which can compute a result every single clock cycle. This flexibility allows trading numeric precision for an increased number of parallel arithmetic units.

Segments of a physical part can work at different clock rates matched to source synchronous data rates. The inherent parallelism of logic resources on an FPGA enables the implementation of high throughput custom hardware accelerators, high-bandwidth custom DSP or video processing equipment.

1.1.2 Design Entry and Configuration

CLBs and routing are configured by programmable switches. Several technologies are used to implement these programmable switches, which is one of the technological factors determining the density and versatility of the resulting network. Fuse or anti-fuse based programming technology allocates minimal footprint for switches, but the resulting structure is one-time programmable. SRAM based configuration switches allocate more silicon area but are reprogrammable. Flash based devices are reprogrammable and retain configuration content during power-down.

Programmable routing, slice, and hard macro configuration, as well as memory initialization for the PS, is carried out via configuration files. Configuration files are generated by vendor specific back-end tools. On the front-end of the toolchain the user specifies the design in the form of schematics, Hardware Description Language (HDL) description, or using higher level programming languages such as of C++ or MATLAB code. Synthesis translates the design to a netlist of logical operators, which subsequently got mapped to the logic resources offered by the target FPGA. Finally, the mapped design needs to be placed and routed in such that the resulting network of interconnected components meet timing specifications.

1.2 Signal and Video Processing

One application area of FPGAs is rapid prototyping of integrated circuits. With the FPGA manufacturer taking the costs, risks, and complexities of silicon design and fabrication, users can take advantage of novel process nodes and transistor densities while paying a higher price for each device. FPGAs are the final implementation solution for high-cost, low-volume products requiring custom high speed signal processing. Prime examples are airborne and maritime radar, high security IP networks and early deployments of cell phone base stations. In these application areas FPGAs perform Digital Signal Processing (DSP) operations such as

- baseband signal processing: filtering, enhancement, reconstruction,
- compression, decompression, encryption, decryption, error detection and correction,
- modulation, demodulation, up-, and down conversion,
- Multiple Input Multiple Output (MIMO) beamforming,
- channel equalization

Another area of widespread FPGA adoption is custom video equipment. Professional video capturing-, recording-, and processing equipment, airborne and military camera systems, high resolution, and high frame scientific cameras can carry the cost and power requirements of FPGA based computing.

Most high level signal and video processing algorithms use a few basic building blocks, such as:

- 1D and 2D correlations and convolutions,
- array thresholding, minimum-, and maximum calculation, normalization,
- Finite Impulse Response (FIR) and Infinite Impulse Response (IIR) filters,
- Fast Fourier, Discrete Cosine and Wavelet transforms,
- Data interleaving, indexing and reorganization.

An FPGA SoC may be the sole processing module in a system, but often FPGAs are used in conjunction with higher density, specialized processing modules. Based on the nature and complexity of the signal processing problem, architects have several options to balance cost, power, and complexity. Problems with low to moderate computational complexity can be addressed by single processor systems. Computationally intensive applications may need multi-core systems, or a General Purpose CPU extended with GPUs, Neural Network inference engines, or specialized hardware accelerators. One particularly attractive, low cost way to increase performance is the use of FPGA based hardware accelerators. This paradigm is in the focus of major processor manufacturers, underscored by a string of acquisitions by large processor manufacturers, such as Altera (now part of Intel)[2], Xilinx (now part of AMD)[3] and Atmel (now part of Microchip)[4].

Appendix A.1 reviews concepts and limitations of GP/GPU systems collocated in modern FPGA based SoCs, as well as the basic DSP operations and their optimizations in FPGAs.

[2] https://www.intel.com/content/dam/www/central-libraries/us/en/documents/agilex-fpgas-game-changing-white-paper.pdf

[3] https://www.xilinx.com/support/documentation/data_sheets/ds891-zynq-ultrascale-plus-overview.pdf

[4] https://www.microchip.com/en-us/products/fpgas-and-plds/system-on-chip-fpgas/polarfire-soc-fpgas

2 Image Signal Processing

FPGAs are ideal for implementing demanding image and signal processing algorithms due to the inherent parallelism of logic resources. Recognition, registration, and reconstruction all need consistent, high-quality images [6]. The purpose of Image Signal Processing (ISP) is to condition images for higher level processing functions. As opposed to other machine vision platforms, such as GPGPU SoCs, like the Nvidia Xavier AGX[5], specialized ASICs, like the Intel Movidius Myriad[6], or mobile platforms, like the Qualcomm Snapdragon 865 [7], FPGAs do not currently have hardened ISP units. While embedded ASSPs have dedicated, proprietary image sensor processing pipelines, FPGAs offer PL resources, tightly integrated with a system processor (PS).

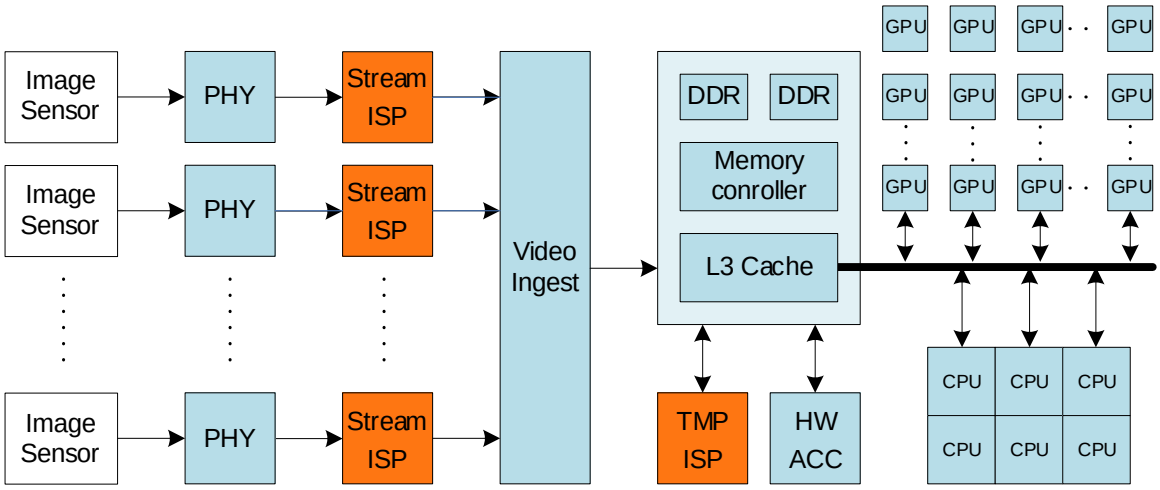


Figure 3: Example implementation of a machine vision platform

Figure 3 presents a typical machine vision platform, with one-, or multiple image sensors providing input to processing algorithms executed on an array of GPU and CPU cores. Image sensors typically connect to the platform via MIPI CSI2.0 DPHY or CPHY, sLVDS, SLVS-EC, or parallel CMOS interfaces. FPGAs provide high-speed I/O features to receive, de-serialize, and deinterleave streams, performed by the PHY blocks on Figure 3.

Spatial ISP functions can be performed directly on the image streams, without having access to previous frames. A Streaming ISP pipeline block can perform:

- Flat-field and pixel defect correction,
- Lens shading correction,
- Bayer Color Filter Array (CFA) interpolation,
- Spatial edge enhancement and noise removal,
- Color correction and Tone mapping,
- Video format conversion (chroma resampling, video scaling)

---

5 https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-agx-xavier

6 https://www.intel.com/content/www/us/en/products/sku/204770

7 https://www.qualcomm.com/media/documents/files/qualcomm-snapdragon-865-5g-mobile-platform-product-brief.pdf

Received frames are subsequently stored in video frame buffers, and are made available to a second set of ISP functions, performing functions using the frame buffer such as:

- Motion adaptive temporal noise reduction,
- Image composition,
- Geometry correction, de-warping, rotation and/or translation
- Frame rate changes or High-dynamic-range (HDR) processing,

which are dependent on accessing multiple video frames at a time.

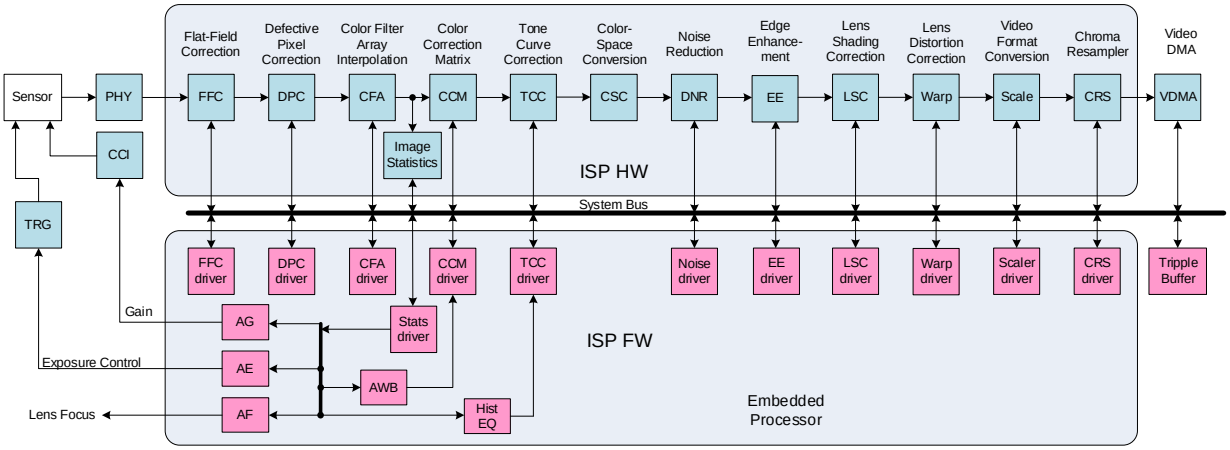


Figure 4: Streaming ISP pipeline HW and FW components

Figure 4 illustrates a typical streaming ISP pipeline, with blue functional units implemented in FPGA PL as custom accelerators, with corresponding drivers and low-level ISP functions (pink blocks) implemented as Firmware (FW), executed by the SoC PS. The camera-side interface of the image sensor processor receives the sensor pixel stream via Physical (PHY) interface module, which de-serializes high speed transmission formats to parallel video data.

2.1 Flat-Field Correction

Flat-field Correction (FFC) aims to remove fixed-pattern image artifacts introduced by the non-uniformities of the sensor and/or the optical system. FFC allows correction of:

- Dark Signal Non-Uniformity (DSNU), pixel-to-pixel differences in dark current,
- Photo Response Non-Uniformity (PRNU), pixel to pixel sensitivity differences,
- vignetting or lens shading,
- other optical defects, such as scratches or contamination of the optical system.

For assessing DSNU, a set of reference images are collected with no exposure – often with the sensor covered [7]. For PRNU, a large set of reference images with uniform illumination are captured and combined into a single frame referred as the flat frame [8].

Due to the characteristic, row/column separable nature of DSNU and PRNU, for high-resolution cameras high-frequency pixel to pixel correction is often performed by row and column projections of calibration images [9]. In this case, lens-shading correction is performed by a separate ISP module based on a parametric vignetting model of the optical system. For systems where access to a compressed, or uncompressed calibration frame is feasible, pixel to pixel compensation can be applied. The flat-field corrected image can be given formally as:

$$I_{FFC} = \frac{I - I_D}{I_F - I_D} m - c, \tag{1}$$

where I is the input image to be corrected, I_D is the dark calibration image, I_F is the flat calibration image, and m and c are constants to adjust the output brightness and black level, respectively. My work on the field of Flat-Field correction is detailed in Chapter 3.

2.2 Defective Pixel Correction

An image sensor may have defects by the linear correction capabilities of the FFC module, affecting individual pixels, or clusters. Affected pixels may have significantly altered dark currents (hot pixels), sensitivities (dead pixels), or excessive noise load (flickers)[8]. Sensors failing image quality checks are discarded by manufacturers. Most defects affecting machine vision systems arise during use by high energy particles impacting photodiodes. The purpose of defective pixel correction (DPC) is the identification and interpolation of these defective pixels.

In static solutions, locations of defects outside the range correctable by FFC are identified during FFC calibration and are stored in a Look-Up-Table. During regular operation sensor images are not analyzed for defective pixels, but defective pixel locations are interpolated on the fly.

Dynamic solutions analyze the sensor data stream for outlier pixels, which are apparently stuck or flicker independent of local motion and neighboring pixel values. While dynamic solutions are significantly more complex and often require access to frame buffers, they can keep up with the degradation of the sensor due to the detection of accumulating pixel defects. My work on the field of DPC is detailed in Chapter 4.

2.3 Color Filter Array Interpolation

Color image sensors use a color filter matrix, laid over the photodiode array, so adjacent, uniform photodiodes can have different spectral sensitivities. The Bayer Color Filter Array (CFA) is the most commonly used arrangement of color mosaic filters, using Red, Green, and Blue (RGB) or Cyan, Magenta and Yellow (CMY) dyes (Figure 5). The mosaic pattern is laid out in repeating lines.

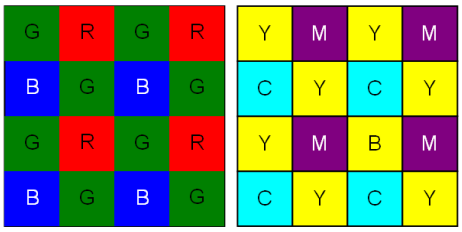


Figure 5. Examples of the Bayer RGB Color Filter Array pattern (left: RGB, right CMY)

From one measured color component per pixel, the Bayer CFA interpolation module restores 3 color components per pixel, using information from the neighboring pixels. The pigments used in the color filters absorb some spectral components of incident light, reducing pixel sensitivity and quantum efficiency. With the green filter most translucent and most similar to human perception of luminance, the array contains twice as many green pixels than red or blue pixels. The CFA sub-samples the image digitized by the sensor, which may introduce color aliasing effects, if the region interpolated contains spectral components between the spatial sampling frequency of the pixel array $f_M = 1/D$, the sampling frequency of the color channels $f_{R,B} = 2/D$, where D is the pixel size.

[8] The International Standard on Ergonomic Image Quality Requirements, pp 2-3: https://standards.iteh.ai/catalog/standards/cen/346aee7a-22c1-466a-9acf-622f79ee1937/en-iso-13406-2-2001

Simple bilinear interpolation with constant weights may introduce severe color artifacts, as illustrated by zoneplate image (Figure 6, left), which represents a 2D spatial frequency sweep from the corners (low frequencies) to the center (high frequencies).

The right side image of Figure 6 shows an excerpt of the light house test image, demonstrating suppressed color artifacts on the natural horizontal frequency sweep of the picket fence.

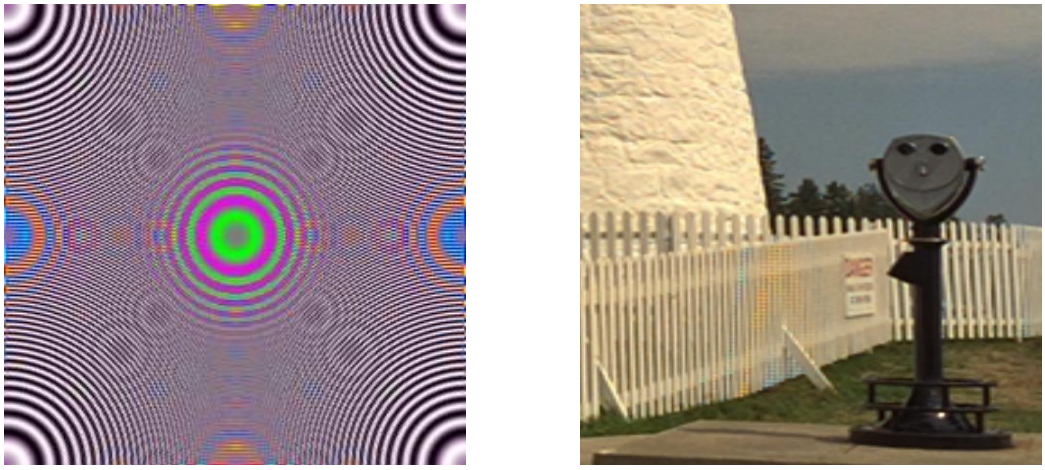


Figure 6. Examples of CFA interpolation color artifacts (left: Zoneplate, right: Lighthouse[9])

Details on my work on the field of FPGA optimized CFA interpolation and post-processing have been presented in section A.2 of the Appendix and were published in [S1].

2.4 Image Statistics

The Image Statistics module implements the computation intensive metering functionality common in digital cameras and imaging devices. Collecting data on a per pixel basis is in many cases prohibitively expensive using ASSPs. Statistical data is consumed by Automatic Exposure (AE)-, Gain (AG), Focus (AF), and White Balance (AWB) control as well as tone curve correction algorithms. The module collects global statistical information over the entire image frame, such as a global histogram, which can be used for tone map correction. Automatic Exposure Control (AE), Automatic White Balancing (AWB), and Automatic Focus (AF) adjustment, often referred to as "3A"algorithms, often rely on local, zone based information, often referred to as zone based metering. High-, and low-frequency, or edge content in considered for adjusting focus, local mean, minimum, maximum, and histograms are used to control exposure and gain. The statistics module analyzes the pixel stream passing through the module, and deposits data either as in ancillary video lines, or metadata corresponding to the frame.

2.5 Color Correction Matrix

The color correction matrix (CCM) is a 3x3 programmable coefficient matrix multiplier with offset compensation that can be used to adjust color component gains, mix-, or decorrelate color channels, adjust zero offset, or convert between color spaces. The 9 coefficients for the 3x3 multiplier, and the 3 coefficients for the output channel offsets can be re-programmed on a frame-by-frame basis. The CCM is the actuator for AWB, controlled by the SW component of the AWB algorithm.

[9] Source: Kodak Lossless True Color Image Suite, image#19 by Alan Fink, Eastman Kodak Company

The physical implementation of this module uses 9 DPS48 tiles, which can perform the matrix multiplication, offset compensation, and unbiased rounding using 24 bit coefficients on 8-18 bit color channels.

2.6 Tone Curve Correction

The tone mapping module can implement arbitrary one-dimensional functions specific for a color channel. Typical use cases include gamma correction, logarithmic encoding, or mapping from 10,12 or higher input color-depth to 8-bit color representation, typical to consumer displays. Tone curves can be re-programmed on a frame-by-frame basis. The tone-curve module is the actuator for histogram equalization, and for the adaptive histogram equalization / CLAHE algorithm [10].

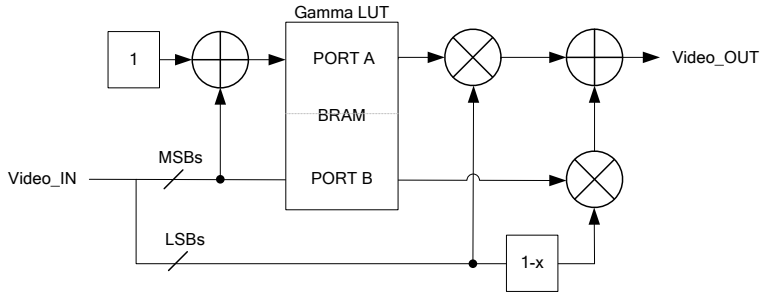


Figure 7: 12 bit gamma correction module with interpolation

BRAM primitives can store 1024x18 bits, and map 10 bit inputs to up to 18 bit outputs. For input color channels with more than 10 bit resolution, the interpolator illustrated on Figure 7 can be used. The desired mapping function $f(s)$ can be interpolated by $\hat{f}(s)$ by splitting the bits of the input sample as $s = (y \ll k) + x$, where x represents the k LSBs:

$$f(s) \approx \hat{f}(s) = f(y)(1 - x) + f(y + 1)x \tag{2}$$

The 10 most significant bits of the input (y) are used to address the dual-ported BRAM and fetch corresponding values $f(y)$ and $f(y + 1)$. The lower k LSBs are used to interpolate between the two nearest mapping function values $f(y)$ and $f(y + 1)$.

2.7 Color-Space Conversion

A color space is a numerical representation of colors. The most popular color models are:

- RGB, and gamma corrected R'G'B', used in computer graphics,
- YIQ, YUV, YCrCb used in video systems ,
- CMYK used in color printing,
- CIELAB, CIEXYZ, designed to represent colors the average human can see,
- HSV (Hue, Saturation, Value) and HSL (Hue, Saturation, Luminance).

HSV and HSL are related to the intuitive notions of hue, saturation, and brightness. Representation in one color space can be transformed to another using appropriate mapping functions. Native input to the conversion is typically the RGB information supplied by devices such as cameras and scanners. Conversion from RGB to YUV/YCrCb (Figure 8) is a typical operation transforming sensor input pixels to industry standard video transport formats, such as YUV 422 and YUV420. The color-space conversion module simplifies the 3x3 linear transformation between input and output color channels and uses only 4 multipliers and 5 adders to carry out the computations.

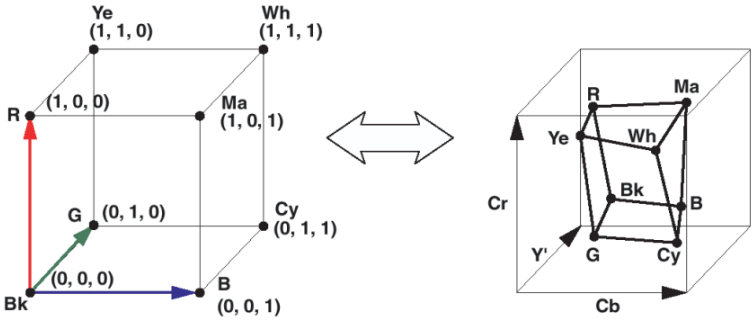


Figure 8: Conceptual conversion between the RGB and YCrCb color-spaces[10]

2.8 Noise Reduction

Images in video-, or image sensor streams can be corrupted by noise. The three most common sources of noise are shot noise, thermal noise (analog electronic interference with Gaussian distribution can also be modeled as thermal noise), and salt-and-pepper noise, which is due to occasional bit-flips during serial transmission. According to the central limit theorem, the aggregate of statistically independent shot noise instances captured by pixels converge to a Gaussian distribution over the entire image. Salt-and-Pepper noise can be effectively combated by 2D non-linear filtering. My work on the field of efficient FPGA implementation of 2D rank-, and median filters is expanded in Chapter 6. A brief summary of common methods to suppress Gaussian noise is provided in Appendix section A.2.4.

2.9 Edge Enhancement

Enhancement of features, such as edges and contours, is a typical image pre-processing operation, improving not only the visual appearance of captured images but the accuracy and performance of downstream recognition and classification tasks [11]. Simple approaches like Prewitt and Sobel edge detectors produce predictable results, but are sensitive to noise, distort texture and unlikely to connect broken edges. Robust algorithms often rely on context from image pyramids or persistence of features across multiple image frames. However, these pre-processing features are seldom available for streaming ISPs. I proposed an edge enhancement and spatial noise suppression solution suitable for FPGA / ASIC implementation [S1], [S2], based on approximation of local edge orientation and strength based on edge detection and morphological operations. The proposed architecture used steerable edge enhancement and blurring kernels. Blurring kernels were used on smooth image areas with low edge strength to reduce noise. Edge enhancement kernels were oriented perpendicular to detected edges. Edge enhancement halo artifacts were controlled by specific detection and suppression kernels. The enhancement of features in the presence of noisy input continues to remain an area of active research.

2.10 Lens Shading Correction

Lens shading, or vignetting, is the progressive darkening of image content from the center towards the corners of the image due to optical-, physical-, and pixel vignetting. Optical vignetting is caused by the lens entrance aperture, which limits the amount of light reaching the sensor. Optical

[10] Source: RGB to YCrCb Color-Space Converter v7.1, LogiCORE IP Product Guide, Xilinx Vivado Design Suite, PG013. https://docs.xilinx.com/v/u/en-US/pg013_v_rgb2ycrcb

vignetting is proportional to $\cos^4\theta$, where θ is the incident angle relative to the optical axis of the lens system. Physical vignetting occurs when other mechanical components, such as the lens barrel, further clips peripheral light before or after the entrance pupil.

Pixel vignetting is due to a mismatch between the Chief Ray Angle (CRA) of the lens system and the micro-lenses on top of the image sensor pixels. Micro-lenses over the sensor pixels at nadir are more efficient at focusing incident light to the buried photodiode than those closer to the edges of the sensor.

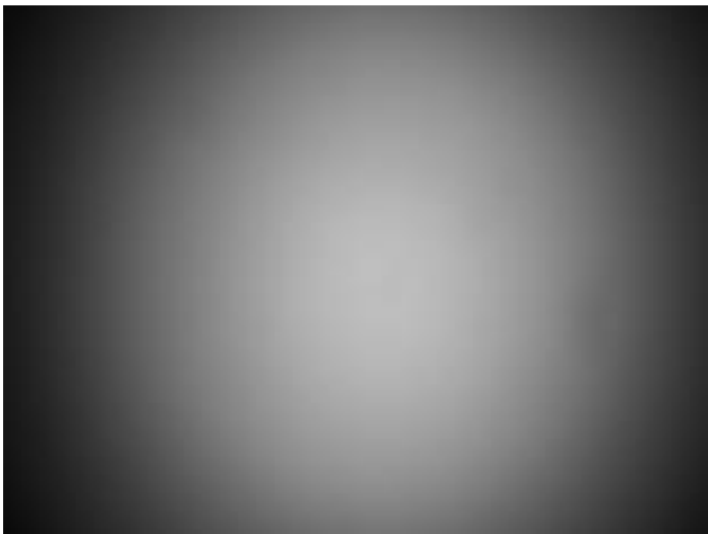


Figure 9: Flat-field image demonstrating strong lens-shading

The simple, low-cost solution to correct lens shading is to record a flat-field image (Figure 9) with the lens assembly attached, and record horizontal and vertical intensity profiles across the center of the image. The intensity profiles can be stored in BRAMs, and dedicated horizontal-, and vertical multipliers can correct for intensity drop, preferably on a single channel before CFA interpolation. For a color image, the two sets of multipliers would require a total of 6 DSP48 primitives. For a more optimal solution, the horizontal and vertical correction factors can be pre-combined, then the combined correction coefficient can be applied to all 3 color channels (4 DSP48 primitives).

However, the 2D intensity drop function is rarely separable. A higher-quality, more precise approach is to store a down-sampled (32 x 32) version of the intensity map using a single BRAM primitive. Either bilinear-, or biquadratic interpolation can be used to compute correction factors for streaming pixels. As pixels are accessed in a scanline-continuous order (left to right, top to bottom), the quadratic differential-integrator structure introduced in section 8.4.1 can be used, to create a highly accurate low-cost interpolator.

For cameras with variable lens-, and shutter assemblies the 2D intensity drop-off function is dependent on zoom and aperture settings, which need to be recorded during manufacturing. The optical de-center for each image in the resulting image stack is the same, which enables storing only a few representative samples along the axes (aperture and zoom) to interpolate. Since aperture and zoom is constant for a frame, and is varying relatively slowly due to mechanical actuators, in practical systems the aperture – zoom image stack is down-sampled (e.g., 32x32), and stored in external memory next to the system processor. Camera FW interpolates between sub-sampled (32x32) images using the current aperture and zoom settings and updates the correction coefficients in the ISP lens-shading correction module between frames.

2.11 Image de-warping

Wide-angle, fish-eye lenses bend rays more near the edges of the lens than the rays near the center of the lens. Due to radial distortion straight lines in real world appear to be curved on the image. Image de-warping, or camera lens distortion correction compensates for deviation from rectilinear projection.

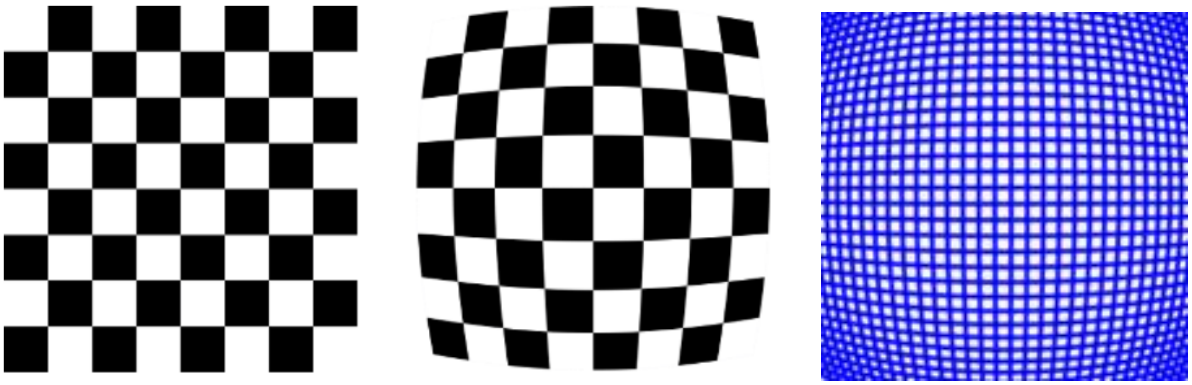


Figure 10: positive radial distortion (a: calibration pattern, b:barrel distortion, c: model)

The first step of distortion correction is characterizing the camera in a series of calibration steps. In a controlled laboratory-, or manufacturing environment the camera is exposed to checkerboard patterns (Figure 10.a) in different orientations. From the recorded images coordinates of checkerboard square corners are extracted. Using the method introduced in [12] extrinsic and intrinsic camera parameters are established. Using linear correction operations translation, clocking, skewing errors can be compensated, fitting the checkerboard pattern with residual non-linear distortions into to output coordinate system (Figure 10.b).

At this point FPGA-, and ASSP/GPU-based ISP algorithms bifurcate. ASSP/GPUs typically prefer a parametric definition of radial/tangential distortion parameters, which are established using the Brown–Conrady even-order polynomial model, or Fitzgibbon's method [13]. For an FPGA implementation, storing a re-sampled distortion map (Figure 10.c) provides a simple way to avoid divisions and polynomial evaluation.

In order to generate a corrected, rectilinear output frame, the exact (sub-pixel) location of each output pixel (x,y) needs to be located in the input frame (x',y'), an operation referred to as backward mapping (Figure 11). Considering that output pixels are generated in a scanline continuous order, the x', and y' coordinates of the distortion grid can be continuously interpolated using the quadratic differential-integrator structure introduced in section 8.4.1, as in the case of lens-shading correction, assuming resampled grid coordinates are stored in respective BRAMs.

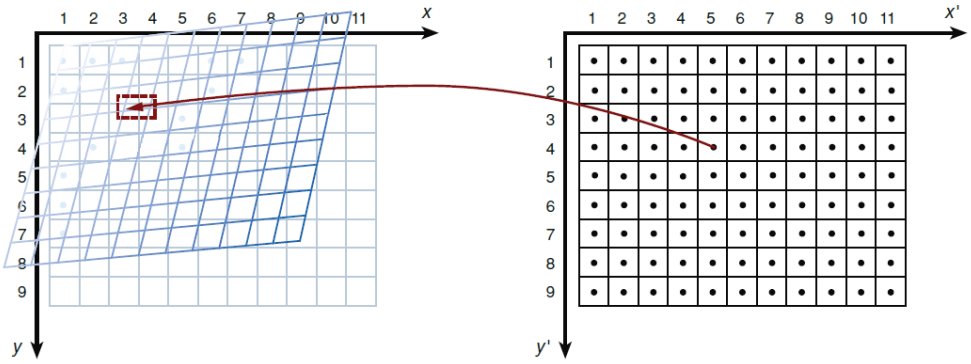


Figure 11: Backward mapping of output pixel positions into the input sampling grid

The backward-mapped coordinates typically do not align with the input sampling grid, therefore output pixels have to be interpolated. Considering that no significant downscaling is performed during distortion correction, bilinear interpolation (Figure 12) delivers adequate results with no perceivable aliasing effects.

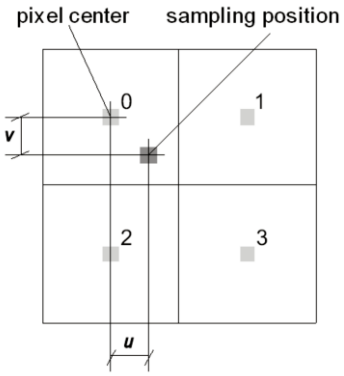


Figure 12: Bilinear interpolation using 4 nearest neighbors

To perform bilinear interpolation, the backward mapped pixel coordinates (x', y') are quantized to $\lfloor x' \rfloor, \lfloor y' \rfloor$, where $\lfloor\ \rfloor$ denotes the integer part of the coordinates, and the four pixels $\underline{p} = \{p(\lfloor x' \rfloor, \lfloor y' \rfloor),\ p(\lfloor x' \rfloor + 1, \lfloor y' \rfloor),\ p(\lfloor x' \rfloor, \lfloor y' \rfloor + 1),\ p(\lfloor x' \rfloor + 1, \lfloor y' \rfloor + 1)\}$ are retrieved from memory.

Note that for each output pixel four input pixels have to be retrieved, which demands considerable bandwidth from the frame buffer, typically in external memory (DDR). Additionally, DDR, shared with other processes, is accessed in arbitrated bursts. Reading a set of horizontally adjacent pixels in a read burst is orders of magnitude faster than randomly accessing single pixels in the frame buffer. Therefore, the local neighborhood around $\lfloor x' \rfloor, \lfloor y' \rfloor$, needs to be cached using BRAMs. Efficient four-way cache design, using dual ported BRAMs, predictively populating the cache before pixels in $\underline{p}$ are accessed, is the key to high performance de-warping.

The remainders, $u = x' - \lfloor x' \rfloor, v = y' - \lfloor y' \rfloor$ are used as coefficients for bilinear interpolation. Coefficient vector $\underline{c} = \{(1-u)(1-v),\ u(1-v), (1-u)v, uv\}$ are used by the bilinear interpolator module, which performs

$$o(x, y) = \underline{c}^T \underline{p} = \sum_{i=1}^{4} c_i p_i. \tag{3}$$

For color image processing with deep colors (10-, or more bits per channel) each color channel may require a bilinear interpolator, allocating 12 DSP48 primitives. For 8 bit color channels, the number of DSP48s can be reduced if the R,G,B components are combined into a single 24 bit logic vector with guard bits between the components {R[7:0], 8'b0, G[7:0]}, and using a single 24x18 multiplier in the DSP48E*x* primitive to perform two eight-bit multiplications simultaneously, reducing the number of DSP48 primitives used to 8. Another option, for high-performance FPGA devices is to run the interpolation kernel at 3x the pixel clock rate, processing color channels sequentially.

Figure 13 presents the top-level block diagram of lens distortion correction module example for color video streams using dual-ported BRAMs and DSP48 primitives.

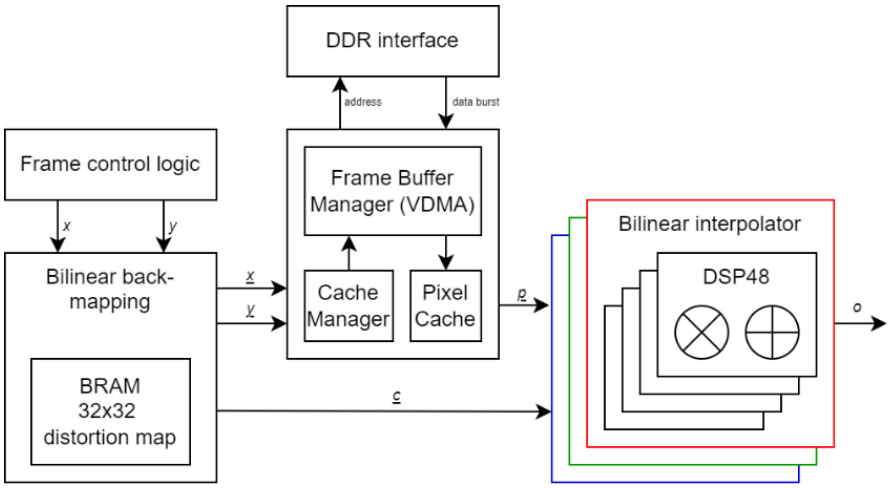


Figure 13: Block diagram of a lens-distortion correction module for Xilinx FPGAs

2.12 Video Scaling

Image sensor output formats may differ from typical streaming video formats, such as 720p, 1080p, or 2160p. Modern image sensors also support binning and windowing / Region of Interest (ROI) operations. In binning mode, the sensor aggregates photons captured by adjacent pixels and outputs a lower resolution image with higher SNR. In ROI mode only a rectangular region of the pixel array is read out, enabling faster frame rates. ROI mode is particularly attractive for wide-angle security camera applications implementing digital Pan-Tilt-Zoom (PTZ) features. In order to support a continuously changing input resolution due to ROI zooming, a configurable video-scaling unit is necessary to convert the input to the selected standard output format.

Video scaling is the process by which an input color image W_{in} pixels wide with H_{in} scanlines is converted to an output color image of dimensions W_{out}, H_{out}. Similar to de-warping described above, output pixels are backward mapped. To avoid aliasing, instead of a simple bilinear kernel, a 2D filter kernel of size $W_{taps} \times H_{taps}$, $W_{taps} = 2w + 1$, $H_{taps} = 2h + 1$ $(h, w \in \mathbb{Z})$ is often used:

$$o(x,y) = \sum_{i=-h}^{h} \sum_{j=-w}^{w} c_{i+h,j+w} p(x+j, y+i), \tag{4}$$

where x and y are discrete locations on the output sampling grid, and $c_{i,j}$ are the 2D filter coefficients. The difference between the bilinear, bicubic, and Lanczos interpolation filters are differences in kernel size and coefficients only. For up-scaling applications, a bilinear, or bicubic kernel is usually adequate. However, for down-scaling properly designed low-pass kernels are necessary to avoid aliasing (Moire) artifacts. The size of the filter kernel is proportional to the downscaling ratio, e.g., to downscale $W_{out} = 0.25W_{in}$ the horizontal size of the kernel needs to span across at least 4 pixels both directions, for $w \geq W_{in}/W_{out}$.

For most practical up-sampling/down-sampling kernels a separable low-pass filter is designed simplifying equation (4) to

$$o(x,y) = \sum_{i=-h}^{h} \mu_{i+h} \sum_{j=-w}^{w} v_{j+w} p(x+j, y+i), \tag{5}$$

where μ_i are coefficients for the vertical-, and v_j are coefficients for the horizontal filters.

Polyphase implementation of one-dimensional FIR filters greatly reduces the number of taps necessary for high-quality resampling. Conceptually, the space between 2 consecutive input pixels is partitioned into a number of phases. Since the resampling ratio is a rational fraction, the location of any arbitrary output pixel between two input pixels in the sampling grid maps onto a discrete grid, exemplified on Figure 14.

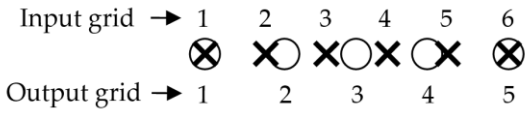


Figure 14: One dimensional resampling from 6 input to 5 output pixels

The 5 output pixels shown from left to right can be interpolated with a two-tap polyphase FIR filter using phases 0..4. In phase 0, the output pixel is co-located with an input pixel, therefore the coefficients for phase 0, $\nu[0] = \{1.0, 0.0\}$. Coefficients for phase 1 are $\nu[1] = \{0.2, 0.8\}$.

To create a configurable 2D scaler implementing a separable low-pass filter, the vertical (Figure 15) and horizontal (Figure 16) resampling stages need to be cascaded. The vertical filtering stage needs to have access to $2h$+1 lines, for which $2h$ image lines need to be buffered. The horizontal stage consumes the output of the vertical stage via an intermediate buffer, providing parallel access to all pixels in the buffer.

The line buffer macro, implemented using BRAMs, is typically driving the footprint of the video scaler module. Even though the module uses several DSP48 primitives, for high resolution sensors, buffering each video line may consume multiple BRAM primitives.

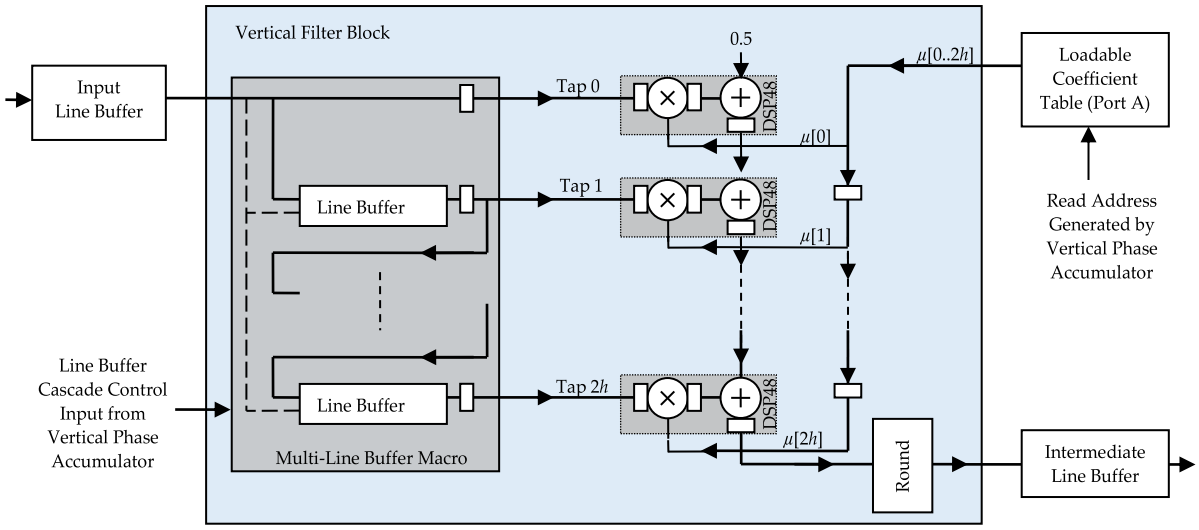


Figure 15: Example of a vertical scaling FIR filter stage, implemented using DSP48 primitives

For up-scaling (zoom-in) operation, it is beneficial to perform vertical scaling first, then horizontal scaling, while for down-scaling the reverse order is optimal to minimize the line-buffer BRAM allocations.

To complete the scaler design, control logic initialized with the scaling ratio tracks x,y position in the frame, and generates horizontal-, and vertical phase accumulator and shift enable signals. Control logic and filter coefficient initialization typically happens in FW, with coefficients pertinent to actual scaling ratios either precomputed and stored in a filter-bank or computed on the fly.

The polyphase concept, polyphase re-sampling low-pass filter design, and coefficient generation is described in detail in [4] pp. 799-806.

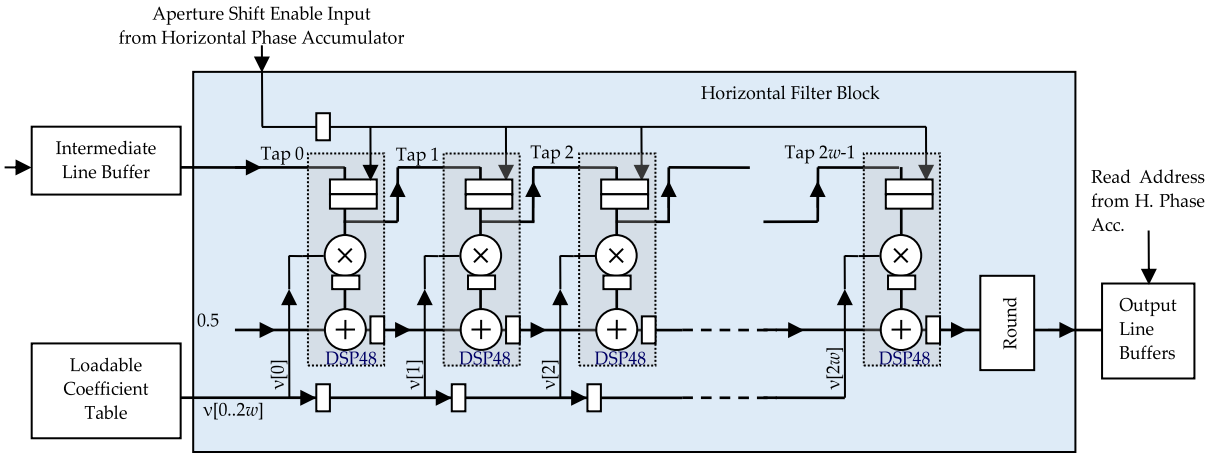


Figure 16: Example of a horizontal scaling FIR filter stage, implemented using DSP48 primitives

2.13 Chroma resampling

For transmission and processing often time camera output is converted to the YUV or YCbCr color-spaces, where Y is Luminance information, and $U(Cb)$ and $V(Cr)$ are derived color difference signals. Acuity of human vision for chrominance is less than half of that for luminance, which allows down sampling the chrominance components without perceivable loss of fidelity. Besides reducing transmission and storage loads, storing two pixels with 8 bit color components in a single 32 bit word ($Y_0U_0Y_1V_0$) greatly simplifies addressing and aligning pixels with DDR bursts and 32/64 bit memory operations. For video transmission this format is referred to as 422, implying that for 4 adjacent pixels in a 2x2 square over two scanlines 2 chroma components are transmitted in each scanline. On Figure 17, white squares represent luminance (Y) samples, blue circles represent the blue chroma components, and red circles represent the red chroma components.

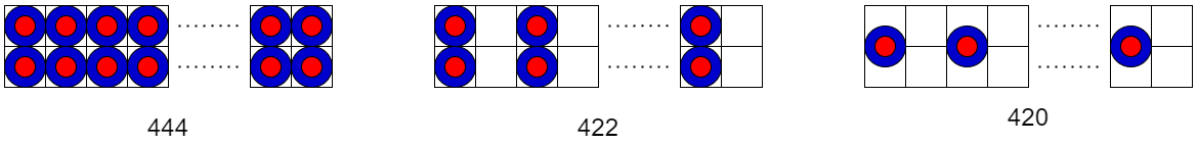


Figure 17: YUV 444, 422, and 420 transmission and storage formats

For the 444 and 422 formats, chrominance samples are co-sited with luminance samples. However, even for 422, chrominance needs to be horizontally low-pass filtered before down-sampling to avoid chrominance aliasing. The 420 format down-samples by two in the vertical direction as well, transmitting / storing one red, and one blue chrominance components for 2x2 adjacent luminance pixels. For MPEG-2, MPEG-4, and H.264 the 420 standard specifies chrominance sampled between the scanlines [14]. In order to re-sample chrominance, and avoid vertical color aliasing, the chrominance components need to be low-pass filtered with a two-, or four tap vertical anti-aliasing filter. Usually, the chroma samples are generated for every output pixel row, but they are valid only for the even pixel rows. Besides streaming video synchronization signals, a typical chroma down-sampling module provides an additional signal marking valid chroma samples.

Higher level algorithms in machine vision platforms may expect RGB, or Luma and Chroma components stored in different planes / arrays, such as for the NV12 format (Figure 18). In machine vision platforms the output of the ISP is often fed to an image compositor layer, which sorts the streaming components into planes. On an FPGA this may be performed by the last stage of the chroma re-sampler, optionally collecting luminance and chrominance samples in different line-buffers. Line buffer outputs in turn can feed separate video DMA write engines, set up such that write transfers are deposited into adjacent memory arrays.

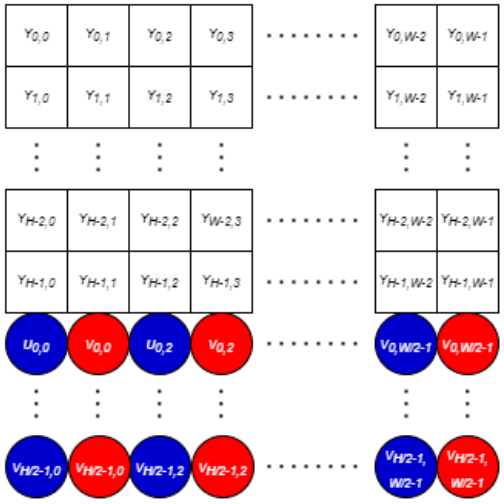


Figure 18: NV12 component planes

2.14 ISP Sensor Control

Besides conditioning image streams, ISPs initialize image sensors and adjust processing parameters, such as exposure time and analog gain, to maintain optimal imaging conditions. [15]The ISP needs to perform 3A functions with minimal latency during video acquisition [15]. Namely, after each exposure, while the frame is being read out from the sensor, image statistics need to be gathered and evaluated. Based on the statistical data, the 3A processing engine needs to set new exposure parameters and control the voice coils of the lens assembly to maintain focus. Similar to the effects of latency introduced to a fed-back control loop, if frames were first committed to a frame buffer, then read back out and analyzed for statistics, the latency introduced would create a system with slow response to changing photographic conditions. Chapter 5, White Balance dives deep into real time algorithmic solutions for optimizing color correction settings based on statistical data.

2.15 Chapter Summary

Sections in this chapter defined the ISP sensor interface and introduced major ISP functional components. Some of these modules, like color-space conversion and gamma correction are simple and well established, others, like CFA interpolation and noise reduction are still actively researched. The goal of FPGA optimization is to maximize performance while minimizing resource footprint and external memory bandwidth. The ISP modules presented, with the exception of Flat-Field correction, Lens Distortion Correction and Motion Adaptive Noise Reduction (MANR) can be implemented without access to an external frame buffer, enabling a basic ISP with reasonable image quality without using external memory.

3 Image Sensor Uniformity Correction

As introduced in section 2.1, digital images captured by image sensors are contaminated with noise, which deteriorate performance and reduce sensitivity. Image noise can be characterized as temporal, or Fixed Pattern Noise (FPN). Temporal noise changes from frame to frame, while FPN is mostly constant, but may depend on temperature or sensor configuration.

3.1.1 FPN Noise Reduction In CMOS Sensors

Figure 19 shows the typical structure of a CMOS image sensor. A matrix of pixels can collect electrons, generated by the photoelectric effect, on the cathodes of photodiodes. A row of pinned photodiodes can be reset, exposed, and read out by corresponding row reset, row transfer, and select drivers. Multiple pixels of a row can be read out simultaneously via a set of programmable gain amplifiers (PGAs) and analog to digital converters (ADCs).

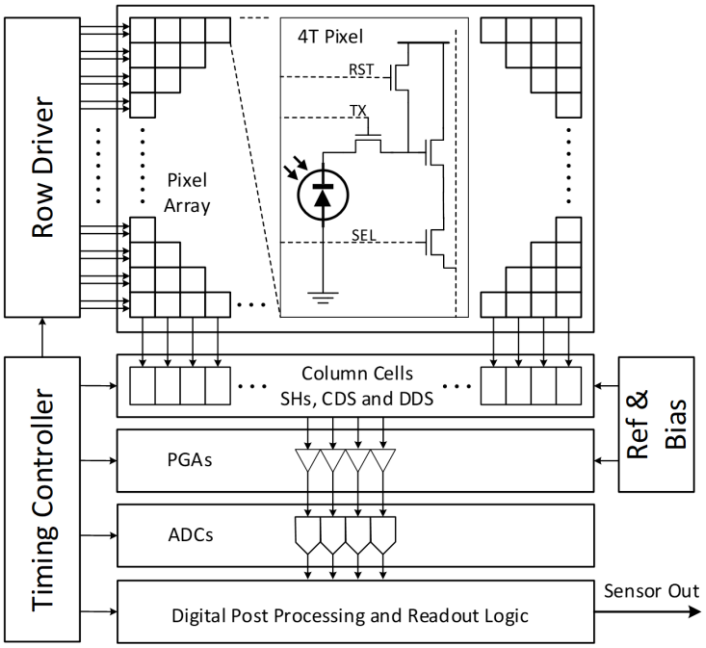


Figure 19. Typical CMOS image sensor block diagram

Modern CMOS sensors use Correlated Double Sampling (CDS) [16] or Correlated Multiple Sampling (CMS) [17]. Analog or digital hardware solutions eliminate dark charge (Q_R), by sampling and holding the output of a pixel after reset, then sampling the same output during readout. Both samples are stored temporarily in column specific capacitors. The column amplifier outputs the difference between the two samples, which effectively removes any common mode bias, such as reset noise.

Differential Delta Sampling (DDS) was introduced to remove fixed pattern noise introduced by small differences between the Sample and Hold (SH) capacitors, and the differential biases of the programmable gain amplifiers by introducing a crowbar operation [18]. The column-wise readout of pixels via PGAs and ADCs, and row wise addressing, before the introduction on CDS and DDS, DSNU and PRNU had a characteristic striated, row-column oriented structure, shown on the left-side image of Figure 20. The right-side image shows a magnified, contrast enhanced sample from the Sony IMX265LLR-C image sensor at 60°C, using 12 dB of analog gain, with the striated structure of the image barely noticeable.

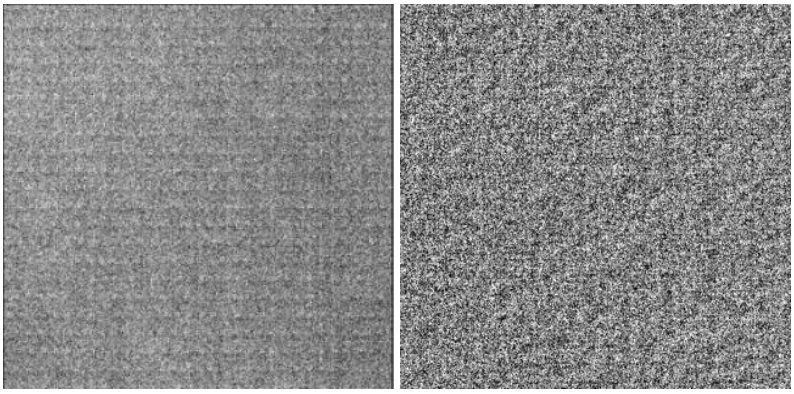


Figure 20. FPN enhanced for visibility

Another often used digital technique to remove DSNU is to surround the active pixel area with rows and columns of optically masked pixels. These pixels are affected by the same conditions (temperature, electronic noise, analog gain) as the active pixels, and their values could be digitally subtracted from the corresponding rows and columns, further reducing systemic, row-column structured noise.

3.2 A Linear model of Spatial Non-Uniformity

The mathematical framework for analysis was introduced by Mooney et al. [19] and later simplified by Perry [20] for the linear model of FPN for Infrared Focal Plane Arrays. Though the Non-Uniformity Correction (NUC) was expanded by multi-point analysis to account for non-linearities of IR FPNs (Takayanagi [16] pp. 147-161), linearity of CMOS image sensor photo-response allows generalization of Mooney's framework for CMOS imagers. The luminous flux received by a small surface element with A, exposed to irradiance L, with incident angle Θ is

$$P = LA\Theta. \tag{6}$$

Without considering the effect of temporal noise sources, such as electronic, thermal, and shot noise, the number of electrons present on the cathode of a reverse biased photodiode illuminated by a narrow-band light source can be modelled by:

$$N = T_{int}(\eta P + D) + Q_R, \tag{7}$$

where T_{int} is the integration time, η is the quantum efficiency, assumed constant for the narrow spectrum of the illuminator, D is the dark current, a Q_R is the residual charge present on the cathode after reset. Contemporary CMOS image sensors use correlated double sampling (CDS) which effectively cancels out the Q_R term [21]. For a pixel with area A, at position x, y in the pixel array, illuminated via a lens by a wide-band illuminator, the number of electrons collected by the pixel is

$$N_{x,y} = T_{int}\left[\tau_{x,y}\int_{\lambda_1}^{\lambda_2} L_{x,y}(\lambda)\eta_{x,y}(\lambda)d\lambda\, A\Omega_{x,y} + D_{x,y}\right], \tag{8}$$

Where $\tau_{x,y}$, $L_{x,y}$, $\eta_{x,y}$, and $D_{x,y}$ respectively are the optical efficacy, spectral radiance density in the λ_1 to λ_2 band, the quantum efficiency, and the integrated dark current, specific for pixel x,y, while

$$\Omega_{x,y} = \left[\frac{\pi cos^4\Theta_{x,y}}{4F^2+1}\right], \tag{9}$$

is the projected solid angle subtended by the exit pupil of the optical system, as viewed from the sensor pixel x,y, where $\Theta_{x,y}$ is the off-axis angle of pixel x,y, and F is the F-number of the lens.

The transmittance of the optical system is assumed to be homogenic in Mooney's model, however, for many CMOS cameras optical efficiency tends to drop towards the corner of the image due to Chief Ray Angle (CRA) mismatch between the last lens element and the micro-lens array focusing light onto the photodiodes. Hence the optical efficacy, $\tau_{x,y}$, is dependent on pixel position and is an important source of fixed pattern non-uniformity.

With the introduction of a response coefficient,

$$R_{x,y} = A\tau_{x,y}\left[\frac{\pi cos^4\Theta_{x,y}}{4F^2+1}\right], \tag{10}$$

Equation (8) can be simplified to

$$N_{x,y} = T_{int}\left[R_{x,y}\int_{\lambda_1}^{\lambda_2} L_{x,y}(\lambda)\eta_{x,y}(\lambda)d\lambda + D_{x,y}\right]. \tag{11}$$

To analyze the impact of pixel-to-pixel variation of parameters, parameters can be expressed as:

$$D_{x,y} = \langle D\rangle + d_{x,y}, \tag{12}$$

$$R_{x,y} = \langle R\rangle + r_{x,y}, \text{ and} \tag{13}$$

$$\eta_{x,y}(\lambda) = \langle\eta(\lambda)\rangle + \kappa_{x,y}(\lambda), \tag{14}$$

where the bracketed variables/functions denote the mean value of the corresponding parameter across the entire image, and the additive quantities capture pixel-to-pixel variations. Namely, $d_{x,y}$, $r_{x,y}$, and $\kappa_{x,y}(\lambda)$ are the pixel-to-pixel variation in dark-current, response coefficient, and quantum efficiency, respectively. When capturing an image with zero illumination, referred as the dark image, $L_{x,y}(\lambda) = 0$, and $N_{x,y} = \mathrm{D}_{x,y} = \langle D\rangle + d_{x,y}$.

The time invariant pixel-to-pixel non-uniformity $T_{int}d_{x,y}$, is referred as the Dark Signal Non-Uniformity (DSNU), with variance $T^2{}_{int}\sigma_d$, where σ_d is the dark current variance. When looking at a uniform gray field, often referred to as Flat-Field (FF), with $L_{x,y}(\lambda) = L$, equation (8) yields:

$$N_{x,y} = T_{int}\left[\mathrm{R}_{x,y}L\int_{\lambda_1}^{\lambda_2}\eta_{x,y}(\lambda)d\lambda\right] + \mathrm{D}_{x,y}. \tag{15}$$

By substituting equations (12) – (14) into (15):

$$N_{x,y} = T_{int}\left[\left(\langle R\rangle + \mathrm{r}_{x,y}\right)L\int_{\lambda_1}^{\lambda_2}\langle\eta(\lambda)\rangle + \kappa_{x,y}(\lambda)\,d\lambda\right] + \langle D\rangle + \mathrm{d}_{x,y}, \tag{16}$$

which in turn can be separated to a spatially uniform part, the perfectly reproduced, constant, FF:

$$N(L) = T_{int}\left[\langle R\rangle L\int_{\lambda_1}^{\lambda_2}\langle\eta(\lambda)\rangle\,d\lambda\right] + \langle D\rangle, \tag{17}$$

and another term constituting the fixed-pattern noise:

$$n_{x,y}(L) = T_{int}\left[Lr_{x,y}\int_{\lambda_1}^{\lambda_2}\langle\eta(\lambda)\rangle + \kappa_{x,y}(\lambda)\,d\lambda\right] + d_{x,y}, \tag{18}$$

the first term of which is referred to as the Photo Response Non-Uniformity (PRNU). The variance of PRNU, following the derivation in [22], assuming $\kappa_{x,y}(\lambda)$, $d_{x,y}$ and $r_{x,y}$ are statistically independent, can be expressed as:

$$\sigma_n^2 = \sigma_D^2 + \frac{\sigma_R^2}{\langle R\rangle^2}(N(L) - \langle D\rangle) + \langle R\rangle L\int_{\lambda_1}^{\lambda_2}\kappa_{x,y}(\lambda)\,d\lambda \tag{19}$$

3.2.1 Temperature Dependence

The temperature dependence of dark current in silicon photodiodes (Nakamura [16], p. 71) can be expressed as:

$$\mathrm{I}_D(T) = \mathrm{A}_{dgen}T^{1.5}e^{-\frac{\mathrm{E}_g}{2kT}} + \mathrm{B}_{ddiff}T^3e^{-\frac{\mathrm{E}_g}{kT}}, \tag{20}$$

where E_g is the activation energy, k is the Boltzmann constant, and A_{dgen} and B_{dgen} are technology dependent coefficients.

The shape of the aggregate temperature dependence function has two exponential regions, one dominated by the diffusion current and one by the spontaneous generation current. The magnitudes of A_{dgen} and B_{dgen} control blending of the current sources.

3.3 Motivation

While my analysis focused on large pixel, high quality, low noise, global shutter sensors, many CMOS sensors are small, low-cost modules in cell phones, laptops, tablets, and web cameras. Smaller pixels often lead to reduced full well capacity, which in turn translates to reduced dynamic range. Video recorded from sensors contains temporal-, and fixed pattern noise superimposed on the signal. The human visual system is excellent at disregarding the temporal, Gaussian noise, but notices patterned FPN deeply buried in temporal noise. FPN is particularly disturbing when it is superimposed on human faces during video conferencing. In this scenario viewers track facial features. Small movements of the face relative to the image sensor cause an apparent shift of FPN over the subject, which most viewers find noticeable and objectionable.

On the other end of the sensor price/quality spectrum, large, stabilized focal plane arrays are used in staring cameras. High end staring cameras typically track their targets and use extended exposure intervals or collect many exposures then register the images stacks to form output images. Image stacking may amplify DSNU if motion between constituent frames is negligible relative to the spatial frequencies of the DSNU. Scheimpflug Lidars using CMOS sensors [23]also depend on FFC to improve SNR.

In machine vision camera applications, the consumer of video streams are algorithms, which may be less effective at canceling noise than the human visual system. Especially for high frame rate imaging tasks with short integration time, the relatively low signal to noise ratio calls for digital post-processing of images to reduce FPN. A prime example of this use case is disparity mapping for visual odometry. In this case two image sensors, with different FPN profiles are looking at the scene, and the processing algorithm looks at the difference, or disparity between the images to infer

depth, or z-axis distance from the sensor pair. Disparity mapping attempts to find correspondence between an image pair, and often the algorithms amplify sub-pixel differences in x-y plane disparities to z-axis depth. Analysis of stereo image pairs collected using a low-cost sensor (EV76C661 [24]) showed that matching density improved from 10.2% to 13.1% with FFC enabled, a 28% improvement. Matching performance can also be simulated without the actual image sensor in hand. If detailed FPN information is available, like the EMVA1288 [7] analysis of a sensor, synthetic image pairs with ground truth depth information like the Middlebury benchmark dataset [25], can be analyzed with and without FPN (Figure 21).

Figure 21. Cone dataset original (left) and with FPN (right)

Similar to lab results with actual image sensors, matching densities without lens shading, DSNU, and PRNU were 28% better for both the cone and teddy datasets (Figure 22).

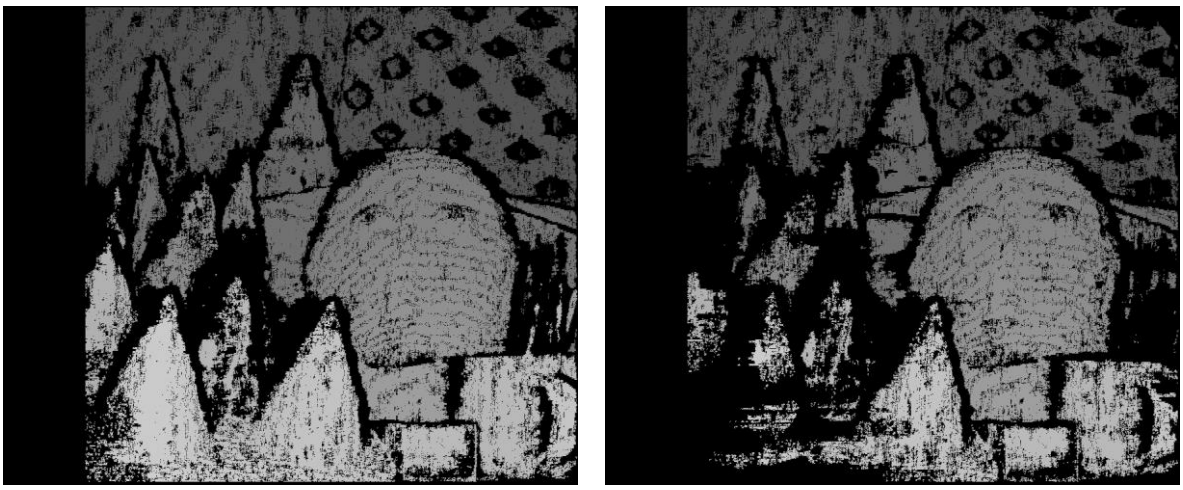


Figure 22. Matching Density with (right) and without (left) FPN

To extract DSNU and PRNU from live video or image frames using non-uniform illumination different algorithmic solutions had been proposed, based on regularization [26], or CNNs [27]. DSNU and PRNU can be recovered using large frame sets with large, flat image areas, such as images of the sky.

Similar techniques can be used to identify the source of a video, by matching FPN as a watermark, embedded in the sources. One application of FFC is to promote data privacy by reducing FPN to a level below the capabilities of forensic algorithms [28].

3.4 Uniformity Calibration Method

At least two exposures, one with no illumination, and one with flat uniform illumination are necessary to capture the sensor specific correction images. For low noise, CMOS, visible light sensors with improved image sensor circuitry (7 transistor pixel, CDS, DDS), thermal, electrical, and shot noise can be several orders of magnitude larger than FPN. In order to cancel temporal noise and to measure DSNU and PRNU, thousands of images need to be captured and averaged.

To analyze the temperature and analog gain level dependency of DSNU and PRNU, capture sequences were repeated in a temperature controlled environment, with different gain settings. Specifically, images were captured

- for two 2nd generation, global shutter, monochrome, Sony Pregius machine vision sensor candidates, the IMX265LLR-C, and the IMX273LLR-C.
- across the entire analog gain range supported by the two sensor candidates, at 0.0, 6.0, 12.0, 18.0 and 24.0 dB.
- for the above datasets, for both sensors, for 5 gain settings, via temperature range supported by the sensors, at 0.0, 15.0, 30.0, 45.0 and 60 Celsius degrees.

The above dataset was collected for the Sony IMX265LLR-C, with the lens, and lens housing removed. A smaller dataset, with two temperatures (10°C and 50°C) and two analog gain settings (2.0 and 24.0 dB), was collected for both the IMX265 and IMX273, with two different lens assemblies attached to the sensor PCBA. The motivation for this is detailed in section 3.9, Lens Shading. For each sensor, gain, and temperature setting, the mean of the collected image stack is:

$$\mu_{x,y}(\bar{p}) = \frac{1}{N}\sum_{1}^{N} F_{x,y}(\bar{p}), \tag{21}$$

and the standard deviation of the stack was computed:

$$\sigma_{x,y}(\bar{p}) = \frac{1}{N-1}\sqrt{\sum_{1}^{N} F_{x,y}^2(\bar{p}) - N\mu_{x,y}^2(\bar{p})}, \tag{22}$$

where $\bar{p} = [T, \alpha, s_{ID}, t]$ is a parameter vector of the capture temperature (T), analog gain (α), sensor ID (s_{ID}), and exposure time (t). $F_{x,y}^2(\bar{p})$ and $\mu_{x,y}^2$ designates elementwise squaring of pixel values for Flat-field frame $F_{x,y}(\bar{p})$, and image stack mean frame $\mu_{x,y}(\bar{p})$. To reduce standard deviation of the temporal noise below 1 LSB, N = 4000 images for each parameter combination needed to be averaged, based on temporal noise measurements discussed in detail in section 3.8.1 and 3.8.2.

In order to precisely control the temperature of the sensor, the sensor PCBA was mounted on a Thermoelectric (TEC) heating/cooling plate, connected to a USB controllable power supply (Keysight E3634A). To control sensor temperature, a software defined PID controller was employed, using the built-in thermometer function of the sensor. Data collection took place in a temperature chamber (No Door α LST365W-PF), which could cool the sensor down to 0 °C. For flat-field light source, an LED panel (Imaging Tech Innovation model ITLB-ST-V1-100K) was used.

The thermoelectric heating/cooling plate and the sensor module were housed in a custom designed, 3D printed enclosure, which attached the sensor assembly to a flexible, collaborative robot arm (Universal Robot UR5e), programmed to move the sensor assembly on a closed, circular trajectory (Figure 23).

Figure 23 Sensor assembly positioned by articulated robot arm in temperature chamber

Introduction of motion while the image stack was recorded was necessary to blur any non-uniformity attributable to the light source. This was essential for the dataset with properly focused lens assemblies attached to the sensor. While many other results published [29] were based on measurements with integrating spheres, the experiments with the lens assembly attached and properly focused revealed both the low frequency (shading), and the high frequency (contamination) characteristics of the integrating sphere. Wang [30] also documented and addressed this issue.

The uniformity specifications of laboratory integrating spheres, typically in the 40dB range[11], are indeed well below the dynamic range of 12-bit sensors. With lens shading corrected and the image normalized for viewing, high frequency content of the integrating sphere images is revealed (Figure 24). If the image sensor is repositioned in the viewport, the PRNU component remains fixed, but smudges and other artifacts are shifted.

[11] Image Engineering CAL3, "CAL3 V2 data sheet":
https://www.image-engineering.de/content/products/equipment/illumination_devices/cal3/downloads/CAL3_data_sheet.pdf

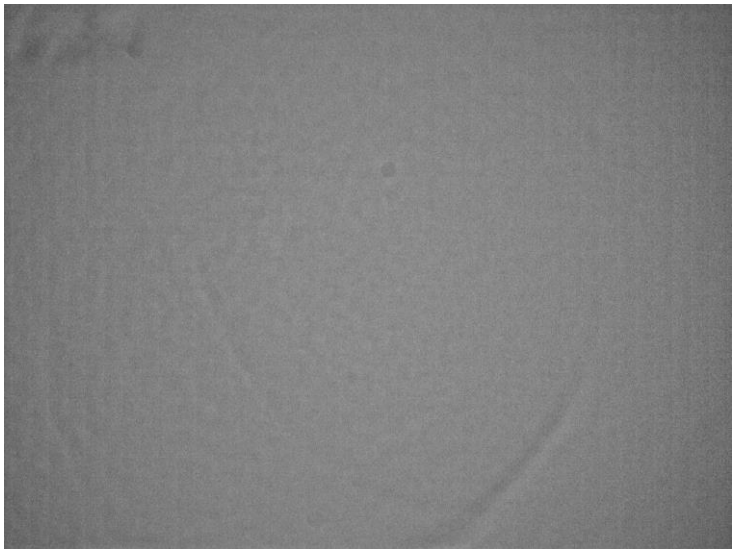


Figure 24. Static integrating sphere image artifacts

3.5 Hardware Implementation of Flat-Field Correction

In the pixel stream $N_{x,y}$ of equation (16), the signal is coupled with DSNU terms $d_{x,y}$ and PRNU term $r_{x,y}$. In order to correct frames, the additive DSNU and the multiplicative PRNU needs to be removed by performing the following correction:

$$O_{x,y}(\bar{p}) = G_g\left[N_{x,y}(\bar{p}) - Z_{in} - G_d(\bar{p})d_{x,y}(\bar{p})\right]\left[g_{x,y}(\boldsymbol{C},T)r_{x,y}(\bar{p})\right] + Z_{out}, \tag{23}$$

where $Z_{in} = \langle D \rangle$ is the black level of the sensor input frame, Z_{out} is the expected output black level, $G_d(\bar{p})$ is a temperature, gain, and sensor ($\bar{p}$) dependent coefficient, which may need to be re-evaluated every time parameters, such as temperature or analog gain, change. During DSNU measurement, and consequently during regular use, the input black level, Z_{in}, is typically set in the sensor to several times the expected standard deviation of DSNU to avoid clipping the measured FPN. Coefficients G_g and Z_{out} perform a linear transformation of the output pixel range, mapping values to the expected output range, 8 -16 bit per pixel data.

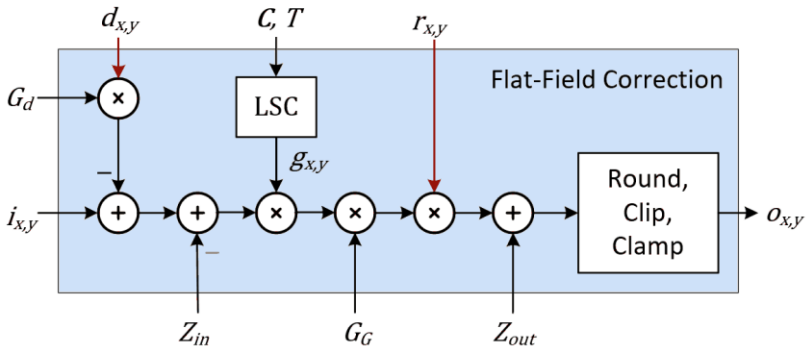


Figure 25. Flat-Field Correction module block diagram

Figure 25 shows a block diagram of a single channel FFC module. Input to the module are the sensor input pixels $i_{x,y} = [\alpha\rho N_{x,y}]$, where α is the analog gain applied, ρ is a charge to voltage coefficient, and [] denotes quantization, clipping and clamping of the sensor output.

Internal to the FFC module is an optional, parametric lens-shading correction (LSC) block, which can be configured with a 32x32 set of coefficient matrix ($\boldsymbol{C}$). Parameters G_d, G_G, $\boldsymbol{C}$, Z_{in} and Z_{out}, which only change between frames, are provided by ISP FW.

In the FPGA ISP implementations of stereo machine vision cameras, the proposed module maps efficiently to the DSP48 resources of novel Xilinx or Lattice FPGAs. Generic parameters controlling the instantiation of the DSNU, Lens Shading, and PRNU correction sections allow balancing FPGA logic resources with the performance requirement of the target application. For example, if temperature compensated lens shading correction (section 3.9) is not a requirement, lens shading correction can be performed by the PRNU correction multiplier if the frame buffer providing $r_{x,y}$ is initialized with the combined PRNU and Lens Shading correction image.

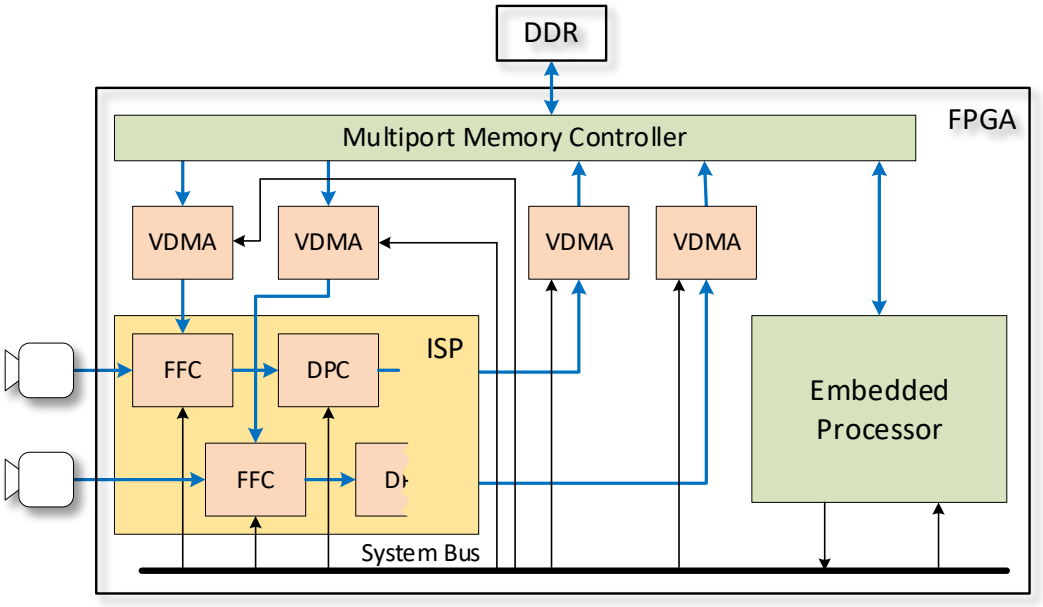


Figure 26. Uniformity correction solution for stereo cameras

All coefficients of the FFC hardware component need to be initialized before use, and coefficients $G_d(\bar{p})$, $\boldsymbol{C}$, $d_{x,y}$ and $r_{x,y}$ may need to be regularly updated to match sensor operating parameters. Initialization, and subsequent parameter updates are carried out by Firmware (FW), which in the proposed implementation is executed by an embedded processor co-located in the same Multiprocessor System on a Chip (MPSoC) FPGA as the ISP HW (Figure 26). Besides configuring FFC parameters, FW also configures the sensor(s), which includes programming the black level to the same Z_{in} value provided to the FFC module corresponding to the sensor.

Per-pixel DSNU and PRNU correction reference frames $d_{x,y}$ and $r_{x,y}$ are provided to the HW FFC modules from external memory (DDR). The Video Direct Memory Access (VDMA) modules in the system transfer video frames between the memory controller and other system components. The memory controller provides shared access to external memory, by arbitrating and prioritizing requests. The VDMAs are also configured by the system processor, and cyclically write, or read frames from predetermined memory address ranges. During initialization, one-, or many DSNU and PRNU calibration images, pertinent to different temperature and analog gain settings, can be loaded to DDR. During operation, FW programs exposure and analog gain settings into the image sensors for every frame (auto-exposure), and periodically reads sensor temperatures. Based on temperature and gain settings, it may also reconfigure the VDMAs to select PRNU and DSNU images best matched to the operating conditions and update parametric lens shading model coefficients ($\boldsymbol{C}$) based on temperature.

DDR bandwidth is a scarce resource, shared by the VDMA modules, the system processor, and other ISP modules and HW accelerators implemented in the MPSoC. Also, the high-speed components of the external memory interface subsystem are primary drivers of dissipation and power consumption. A secondary goal of an efficient FFC solution is to minimize DDR bandwidth.

Another design objective is to minimize manufacturing and calibration time. Even with proper automation, capturing thousands of images for PRNU calibration is time consuming. Capturing DSNU and PRNU images for large sets of gain-temperature combinations for each individual sensor can be prohibitively costly for mass manufacturing practical machine vision systems.

The subsequent analysis and summary focuses on how much DSNU and PRNU can be reduced over the entire gain and temperature range served by the ISP, while simultaneously minimizing access to DDR and the number of calibration images used.

3.6 Related Work

Generalized linear correction architectures similar to equation (23) were proposed in the seminal works of Seibert [22] and Snyder [31]. Based on application area, cost, and performance requirements, many solutions were proposed for FFC implementation and calibration methods.

As for implementation of FFC in an FPGA, Vasiliev [32] describe an FPGA based ISP for a VGA CMOS image sensor, including column based DSNU correction. For a basic FFC of the OmniVision OV5647 and Sony IMX219 sensors, Bowman [33] proposed a simple apparatus to counter lens-shading and establish color-correction coefficients.

To correct lens shading and sensor non-uniformity, many documented solutions propose static, single reference solutions. This is suitable for applications like microscopy, where at least temperature is expected to be relatively stable. Zhang [34] presents DSNU and PRNU reduction results, for back-illuminated scientific CMOS cameras (iXon 89, Dhyana 95 and Prime95B). However, temperature and gain dependence of DSNU and PRNU reduction efficiency, as well as the exact FFC method used is not discussed in their article. Our results corroborate theirs on the magnitude of DSNU and PRNU reduction achievable with single DSNU/PRNU reference images.

The static approach is also viable for IR FPAs, used by many consumer-grade (Teledyne-FLIR[12]), and aerospace (Hercules[13]) IR cameras, which perform NUC periodically during operation using a cold-plate mechanical shutter [35]. However, closing the shutter during use for a short period to capture FPN reference images may not be acceptable for defense or real-time process control applications. Another class of FFC solutions compensate for temperature but disregards the dependency of DSNU and PRNU on analog gain [36].

3.7 DSNU Analysis

DSNU measurements without the lens holder and lens assembly were conducted with a cover over the sensor. DSNU measurements with the lens assembly attached were conducted with the lens cap covering the lens.

---

[12] Teledyne-FLIR, "160x120 High-resolution Micro Thermal Camera: Lepton 3 & 3.5", 2022, https://flir.netx.net/file/asset/15529/original/attachment

[13] Silicon Devices, "Hercules 1280 X 1024, 15 µm pitch, digital InSb MWIR", 2021, https://scdusa-ir.com/wp-content/uploads/2021/11/Hercules-1280.pdf

3.7.1 DSNU vs Exposure Time

In the first dataset DSNU images for the IMX265LLR-C and IMX273LLR-C sensors were captured with 0.5ms and 2ms exposure times, while holding temperature and gain constant. Table 1 demonstrates almost perfect correlation between frames captured with different exposure times, with almost identical Standard Deviation (SD). SD is expressed in LSBs of 12 bit sensor data.

Dark current (thermal noise) is attenuated, but not fully cancelled by averaging. Note that the lower correlation values pertain to parameter sets with low SD, which amplifies the relative effect of quantization noise, as well as the residual temporal noise. Likely these sensors already contain advanced silicon process features to reduce dark current, as well as CMS, DDS, and automatic black level adjustment. These findings are consistent with the findings of Changmiao et al. [37]. Based on these results, for the rest of this analysis DSNU is treated invariant of exposure time.

Sensor Type	Temperature ['C]	Analog gain [dB]	Std. Dev. T_{int}=0.5us	Std. Dev. T_{int}=2.0us	Pearson Correlation
IMX265	10	2.0	1.423	1.416	0.974
IMX265	10	24.0	26.164	26.047	0.997
IMX265	50	2.0	3.275	4.982	0.983
IMX265	50	24.0	60.294	60.428	0.997
IMX273	10	2.0	4.430	4.416	0.988
IMX273	10	24.0	52.864	52.880	0.995
IMX273	50	2.0	4.799	5.376	0.962
IMX273	50	24.0	64.950	65.802	0.982

Table 1: DSNU Standard Deviation and Pearson correlation

3.7.2 Standard Deviation of Uncorrected DSNU

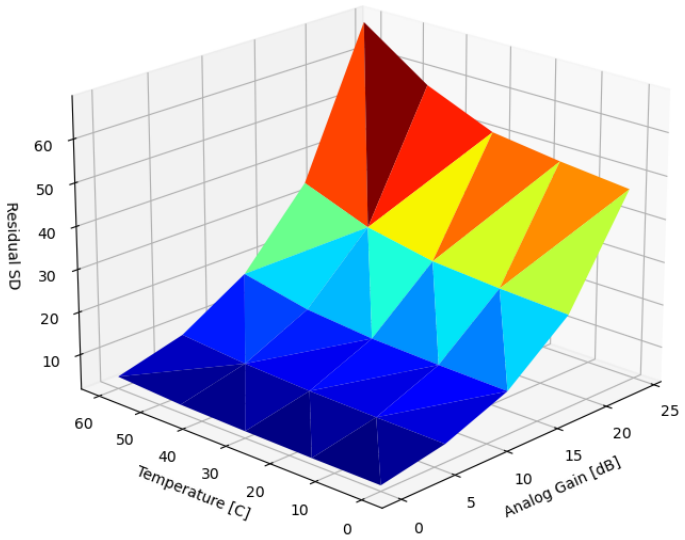


Figure 27. Standard deviation of uncorrected DSNU

Figure 27 presents the SD of uncorrected DSNU as a function of temperature and analog gain. Matching expectations and existing results, the magnitude of DSNU scales exponentially with both temperature and analog gain.

Equation (20) establishes the theoretical background for the exponential relation with temperature, while the definition, $\alpha = 20log_{10}A_g$ [dB] of analog gain consequently results to the A_g factor used by the programmable gain amplifiers in the sensor to scale exponentially with α.

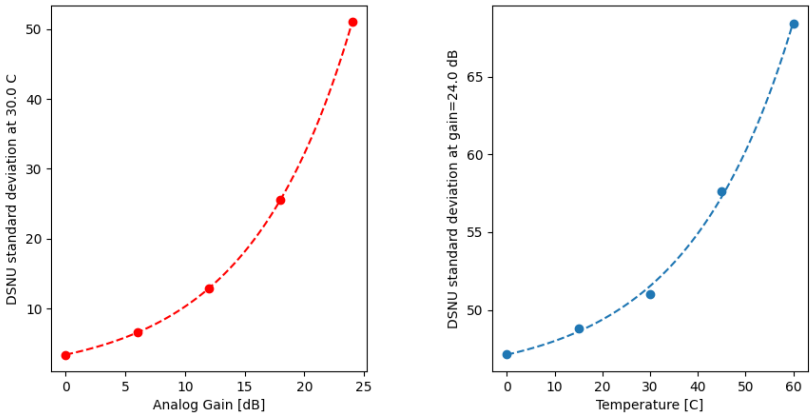


Figure 28. SD of DSNU at 30°C (left), and at 24.0dB gain (right)

Fitting exponentials along the axes (Figure 28), and modeling DSNU as a product of an FPN template, with magnitude approximated with a separable surface:

$$d_{x,y}(T,\alpha) \approx \hat{d}_{x,y} D(T,\alpha), \tag{24}$$

where $\hat{d}_{x,y}$ is a reference DSNU capture, normalized to $\sigma = 1.0$. For the IMX265,

$$D(T,\alpha) = c_0(c_1 e^{c_2\alpha} + c_3)(c_4 e^{c_5 T} + c_6), \tag{25}$$

with c_0= 0.0196, $c_{1,2,3}$= {3.1605, 0.1156, 0.2679} and $c_{4,5,6}$= {1.547, 0.0449, 45.5775}.

For optimal results $\hat{d}_{x,y}$ should be captured at a temperature and analog gain setting resulting to maximum correlation with DSNU images captured for the rest of the parameter space.

3.7.3 Single point correction

As a first approximation, a single prior, $\hat{d}_{x,y}$, was used without adjusting magnitude (G_d) to cancel DSNU across the entire temperature and analog gain range. Figure 29 presents the SD of residual DSNU when corrected with a static reference image captured at 45 °C and 18.0 dB. At these parameter values, SD of DSNU is 28.64 (Table 2), about half of the worst-case SD. As expected, due to the strong correlation between DSNU across temperatures and gain ranges, DSNU is almost perfectly cancelled at the parameter values where the reference image was captured. The non-zero residual noise is due to the fact that two sets of 2000 images for each parameter setting were collected, to correct one stack using the other as reference. This method also helps to quantify leftover temporal noise in the data. Worst-case DSNU is reduced considerably, by 37%, but DSNU is *significantly increased* in the range of the parameter space where SD of DSNU was lower than that of the reference image.

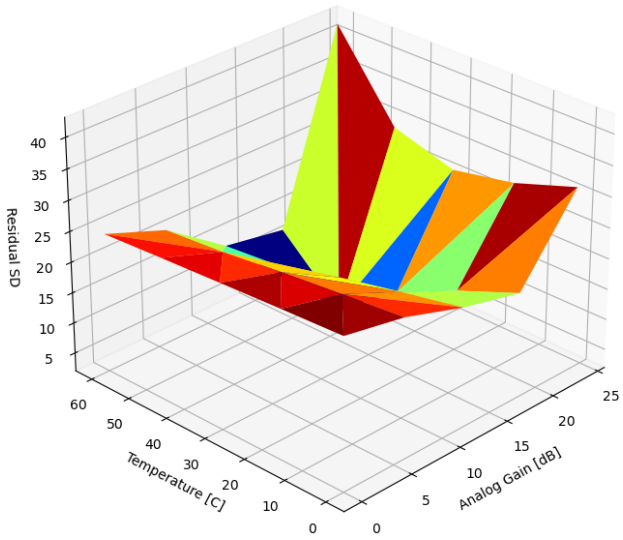


Figure 29. SD of DSNU corrected with a single, static image

Pearson correlation was chosen as a similarity metric between FPN captures due to its invariance to signal magnitude. Pearson correlation between a scaled reference frame and an actual dark input frame remains unaffected by scaling with (G_d).

Temp. ['C]	Analog gain [dB]	T_{int} [ms]	Original SD	Residual SD	Pearson Correlation
0	0	0.53	3.16	26.44	0.726
0	24	0.02	47.17	31.43	0.761
15	0	0.51	3.23	25.99	0.840
15	24	0.02	48.82	27.94	0.867
30	0	0.49	3.43	25.49	0.929
30	24	0.01	50.99	26.01	0.939
45	0	0.46	3.76	24.98	0.977
45	6	0.23	7.34	21.39	0.991
45	12	0.10	14.44	14.36	0.995
45	18	0.05	28.64	2.80	0.995
45	24	0.01	57.63	29.28	0.995
60	0	0.45	4.64	24.44	0.920
60	6	0.21	8.69	20.75	0.948
60	12	0.10	17.71	13.62	0.939
60	18	0.04	34.43	12.23	0.941
60	24	0.01	68.42	42.62	0.940

Table 2: SD and Pearson correlation with static reference

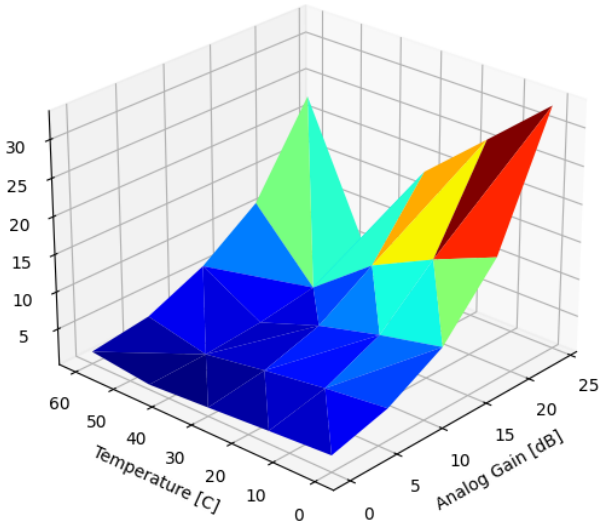


Figure 30. Residual SD of DSNU with a single, scaled image

Figure 30 presents the standard deviation of the residual DSNU after correction with a single reference that was scaled by

$$G_d(T,\alpha) = \frac{D(T,\alpha)}{\sigma(\hat{d}_{x,y})}, \tag{26}$$

where $D(T,\alpha)$ is the approximation introduced in equation (25), and $\sigma(\hat{d}_{x,y})$ is the standard deviation of the reference frame. This method reduced DSNU across the entire temperature and gain parameter space. The best DSNU reduction performance, 91.39% (21.3 dB), coincided with the temperature and gain identical to the reference frame parameters, alas the highest Pearson correlation between frame and reference, and the worst reduction performance, 25.1% (2.51dB) was measured at the parameter combination with the least Pearson correlation with the reference frame.

Establishing $D(T,\alpha)$ for each sensor instance may be prohibitively costly for mass manufacturing. While $D(T,\alpha)$ is fairly uniform for the same batch of sensors, a more accurate method is to read out the Optically Blanked Pixels (OBP) around the active region of the sensor image frame, then calculate the standard deviation of the OBP region. OBPs are affected by temperature and gain settings identical to regular pixels and provide an accurate value for in-situ assessment of SD magnitude. This method can be considered a digital post-processing step after the analog correction proposed by Zhu [9]. Reading out the OBP presents a small (~1%) overhead during regular operation, which is a trade to be considered with the manufacturing overhead of establishing $D(T,\alpha)$.

3.7.4 Multi-point correction

Results from correction with a single reference frame confirm that the key to improved FPN reduction performance is to use reference frames better correlated with the FPN characteristic to the temperature and gain parameters of the image frames to be corrected. For this purpose, many IR thermal imaging sensors and staring cameras use multiple sets of FPN reference images and apply the one best suited to actual operating parameters.

3.7.5 Correction via Linear Interpolation

A straightforward way to improve correlation is to calibrate at multiple gain and temperature settings, then interpolate the reference frame used by the FFC HW based on current temperature and gain settings (T, α). Suppose $n \geq 3$ reference points, $\{p_1, p_2 .. p_n\}$, in parameter space $\bar{p}$ are selected for calibration, for which corresponding DSNU reference images $\hat{d}_i \in \{\hat{d}_1, \hat{d}_2 .. \hat{d}_n\}$ have been captured.

To find the estimated $\hat{d}$ pertinent to $p = \{a, T\}$, the first step is to find p_A, p_B, p_C the three closest neighbors to p, preferably defining a triangle that contains p. The simplest way to do this is to define a distance operator $\|p - p_i\|$, based on which all candidate reference points can be rank ordered by proximity. A practical measure could be $c(\alpha - \alpha_i)^2 + (T - T_i)^2$, where α and T are the analog gain and temperature for p, and α_i and T_i are the analog gain and temperature for reference point p_i, and c is a constant weight. Note that the perfect distance metric would be the inverse of correlation between DSNU of the particular frame, and of the reference frame. However, the DSNU of the current frame is unknown. Based on Table 2 analog gain only scales DSNU, therefore DSNU along the gain axis is highly correlated, suggesting a low value for c.

The resulting reference frame to be used for FFC will be interpolated, using

$$\hat{d} = \sum_{i \in \{A,B,C\}} \lambda_i \, \hat{d}_i, \tag{27}$$

where λ_i are the barycentric coordinates of p, as defined by the triangle p_A, p_B, p_C in the analog gain and temperature parameter space $\bar{p} = \{a, T\}$. Coordinates λ_i can be found by solving

$$P\bar{\lambda} = \bar{p'}, \tag{28}$$

$$\begin{bmatrix} 1 & 1 & 1 \\ a_A & a_B & a_C \\ T_A & T_B & T_C \end{bmatrix} \begin{bmatrix} \lambda_A \\ \lambda_B \\ \lambda_C \end{bmatrix} = \begin{bmatrix} 1 \\ a \\ T \end{bmatrix} \tag{29}$$

for $\bar{\lambda}$, which yields

$$\begin{bmatrix} \lambda_A \\ \lambda_B \\ \lambda_C \end{bmatrix} = \begin{bmatrix} ((\alpha_B - \alpha_C)(T - T_C) + (\alpha - \alpha_C)(T_C - T_B))/D \\ ((\alpha_C - \alpha_A)(T - T_C) + (\alpha - \alpha_C)(T_C - T_A))/D \\ 1 - \lambda_A - \lambda_B \end{bmatrix}, \tag{30}$$

where

$$D = (\alpha_C - \alpha_B)(T_A - T_C) + (\alpha_A - \alpha_C)(T_B - T_C). \tag{31}$$

The magnitude of D can be thought of as the oriented area of a parallelogram defined by the $p_{\overrightarrow{AB}}$ and $p_{\overrightarrow{AC}}$ vectors. If p_A, p_B, p_C are on a line, $D = 0$, and the larger $|D|$ is the more orthogonal $p_{\overrightarrow{AB}}$ and $p_{\overrightarrow{AB}}$ are, thus the better for interpolating $\bar{p}$. Another consideration besides having $D \neq 0$ for selecting 3 candidates from the rank ordered list of reference candidates is to have positive barycentric coordinates $(\lambda_i > 0), \ \forall \, i \in \{A, B, C\}$, which ensures the $\hat{d}$ is not extrapolated in equation (27).

The SD of the resulting blended reference $\hat{d}$ can be calculated during evaluation of (27), or can be estimated using the precomputed SDs of the constituent references:

$$\sigma(\hat{d}) \cong \sqrt{\sum_{i \in \{A,B,C\}} \lambda_i \, \sigma^2(\hat{d}_i)}, \tag{32}$$

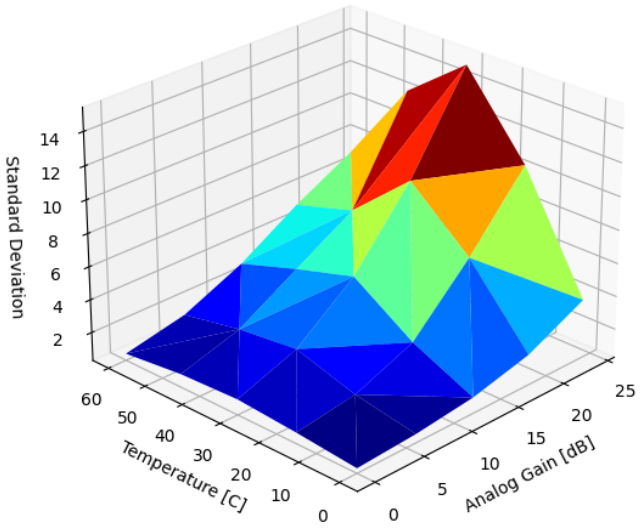


Figure 31. SD of DSNU corrected with interpolated reference, using 4 reference captures

Similar to the method introduced in section 3.7.3, the estimated SD can be used to scale interpolated reference frame $\hat{d}$ according to equation (26).

To evaluate the linear interpolation method, 4 reference DSNU images were captured at the 4 corners of the parameter space, using both extremes of temperature and gain. This intuitive selection ensures that for any parameter combination p in parameter space $\bar{p}$, a set of 3 references can be selected such that p is inside the triangle defined by the references.

As demonstrated by Figure 31, interpolating the reference image produced very good results, significantly reducing worst-case DSNU.

3.7.6 Optimizing reference selection

In order to interpolate between references, at least 3 reference DSNU captures are necessary. By each additional reference image captured, DSNU can be further attenuated around the parameter combination of the reference image, at the cost of additional calibration time, DDR allocation, and boot time.

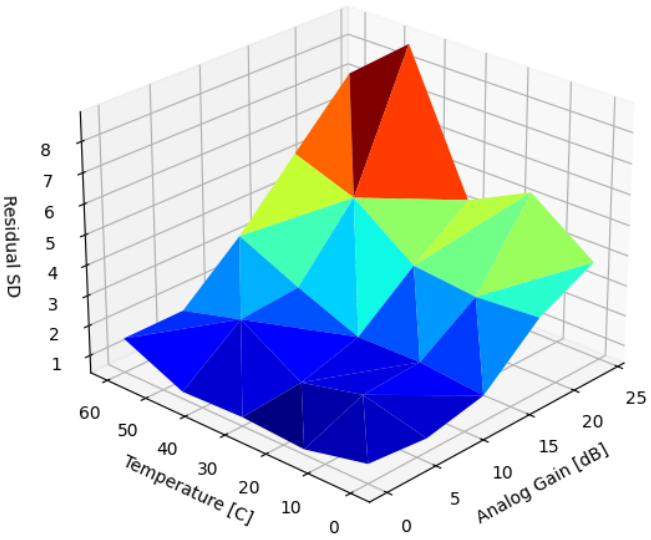


Figure 32. SD of DSNU corrected with interpolation between 5 reference captures

DSNU suppression quality can be also improved by optimizing the reference parameters for a given number of reference frames. Figure 32 presents the residual SD after correction with 5 reference images, captured at {(0°C, 0dB), (60°C, 0dB), (0°C, 24dB), (30°C, 24dB) and (60°C, 24dB)}. Corresponding Table 3 lists Pearson correlation values between the actual frames and the interpolated references. Besides improved DSNU suppression measurements, notice the improvement in correlation with respect to correlation values in Table 2.

Every time analog gain, or measured die temperature deltas exceed a predefined threshold, the embedded processor controlling the ISP (Figure 26) needs to re-compute the interpolated reference image $\hat{d}$. Selecting the 3 closest candidates around the current $p = \{\alpha, T\}$ from a set of $\{p_1, p_2 .. p_n\}$, reference parameters is trivial.

Temperature ['C]	Analog gain [dB]	Frame SD	Residual SD	Pearson Correlation	DSNU Reduction [%]	DSNU Reduction [dB]
0	12	11.90	1.28	0.994	89.3%	19.39
0	24	47.17	3.82	0.997	91.9%	21.84
0	18	23.61	2.89	0.993	87.8%	18.24
0	6	6.05	0.87	0.990	85.7%	16.88
0	0	3.16	1.06	0.944	66.6%	9.52
15	24	48.82	5.33	0.994	89.1%	19.24
15	18	24.40	2.73	0.994	88.8%	19.01
⋮	⋮	⋮	⋮	⋮	⋮	⋮
45	12	14.44	2.28	0.987	84.2%	16.02
45	6	7.34	2.12	0.958	71.1%	10.80
45	0	3.76	0.65	0.985	82.8%	15.30
60	24	68.42	7.10	0.995	89.6%	19.69
60	18	34.43	5.17	0.989	85.0%	16.47
60	12	17.71	3.21	0.984	81.9%	14.84
60	6	8.69	1.56	0.984	82.0%	14.90
60	0	4.64	1.55	0.944	66.6%	9.52

Table 3: Residual SD, Pearson correlation, DSNU reduction using interpolation with 5 references

The majority of the FFC related workload for the embedded ISP processor is to perform the actual interpolation on millions of pixels, which is dependent on the frame size, but invariant of the number of reference images, n.

At startup the embedded processor needs to load reference DSNU frames from non-volatile memory (NVME) to the system memory (DDR), which, depending on the NVME used may present a small penalty in terms of boot time, for each additional reference image. DDR or NVME size / cost typically is not a concern considering image sizes relative to current package capacities.

In order to optimize the locations of reference captures $\{p_1, p_2 .. p_n\}$ in parameter space $\bar{p} = \{\alpha, T\}$, the following quantities need to be introduced.

- Let $\psi_{\bar{\varepsilon}}(\alpha', T')$ denote the probability that during regular operation sensor temperature (T) and analog gain (α) is within a predefined range $|T - T'| < \varepsilon_T$ and $|\alpha - \alpha'| < \varepsilon_a$, such that

$$\sum_{a_{min}}^{a_{max}} \int_{T_{min}}^{T_{max}} \psi_{\bar{\varepsilon}}(\alpha, T) dT = 1.0 \tag{33}$$

  $\psi_{\bar{\varepsilon}}$ is essentially the 2D probability density function based on discrete parameter α, which is a register setting, and continuous parameter T, derived from camera usage statistics.

- Let $\omega(\alpha, T)$ denote the weight, or relative importance the user application, e.g., disparity mapping, associates with parameter combination (α, T). For high-gain scenarios increased temporal noise may reduce the importance of DSNU.

With these quantities, the optimum set of reference parameters, $\hat{p}$, can be selected as

$$\hat{p} = \underset{p \epsilon \bar{p}}{\operatorname{argmin}} \sum_{\alpha_{min}}^{\alpha_{max}} \int_{T_{min}}^{T_{max}} \sigma_p(\alpha, T)\ \omega(\alpha, T)\ \psi_{\bar{\varepsilon}}(\alpha, T) dT, \tag{34}$$

where $p = \{p_1, p_2 .. p_n\}$ is set of (α, T) parameters, at which reference images were captured. $\sigma_{p_n}(\alpha, T)$ is the SD of the residual DSNU on images corrected with reference set p.

The argmin operation can be implemented with simulated annealing (SA) [38], starting the p_n from a constellation of parameters distributed along the edges of the parameter space. This offline operation may be lengthy even on a powerful computer, but it only has to be performed once per reference set p during system design, assuming $\omega(a, T)$ and $\psi_{\bar{\varepsilon}}(a, T)$ are stable.

3.7.7 Correction VIA Logarithmic Interpolation

Since DSNU is an exponential function of both α and T, interpolating in logarithmic space intuitively would suggest improved results:

$$\widehat{d'} = \exp\left(\sum_{i \in \{A,B,C\}} \lambda_i\ \hat{d'}_i \right), \tag{35}$$

where $\hat{d'}_i = \ln(\hat{d}_i)$ are the reference DSNU images stored in logarithmic format.

My results unfortunately did not confirm this hypothesis. With the same set of reference parameters, the resulting DSNU residuals were slightly higher than that of linear interpolation.

3.8 PRNU Analysis

Noise patterns on the sensor output image depend on imaging conditions: illumination, exposure time, analog gain, as well as temperature, conducted, capacitive and inductive electronic interferences. FF images were captured for multiple IMX265LLR-C and IMX273LLR-C sensor instances at 0.5ms and 2ms exposure times, while holding temperature and gain constant.

PRNU VS Exposure Time

Analog gain and temperature were measured while holding the other two parameters constant. Measurements were performed with the lens assembly present over the sensor. Table 4 demonstrates almost perfect correlation between FF image stacks captured with different exposure times. For the IMX265, SD displays a slight dependence on exposure time and temperature, with almost no dependence (0.1%) at 0°C, and 2.1% increase at 60°C, while exposure time increased fourfold from 0.5ms to 2.0ms. These increases are under the residual noise floor after averaging temporal noise over 2000 images. The IMX273 did not display any measurable dependence on exposure time. These results confirm the expectation that PRNU in CMOS sensors is independent of exposure timing. The timing of digital control signals (e.g., SEL, TX, RST on Figure 19) signals is uniform across the imaging array, as well as the switching characteristics of the transistors involved. On/Off transition times (ns range) of the MOSFETs are also at least 4 orders of magnitude smaller than the exposure period (ms range).

Sensor Type	Temperature [°C]	Analog gain [dB]	Std. Dev. T_{int}=0.5ms	Std. Dev. T_{int}=2.0ms	Pearson Correlation
IMX265	0	2.0	175.69	176.00	0.999141
IMX265	20	2.0	174.47	176.33	0.999050
IMX265	60	2.0	171.03	174.16	0.998582
IMX273	35	0.2	182.69	183.31	0.996231
IMX273	35	12.0	180.74	183.32	0.995280
IMX273	60	0.2	177.80	181.75	0.999082
IMX273	60	12.0	154.77	155.31	0.996216

Table 4: FF Standard Deviation and Pearson correlation

Based on these results, for the rest of this analysis, just like DSNU, PRNU is treated invariant of exposure time.

In comparison with Table 1, SD of PRNU is up to two orders of magnitude larger than that of DSNU, and initially seems a lot less dependent on analog gain or temperature. A second set of measurements were performed with focus on temperature and gain dependence for the IMX265LLR-C, capturing PRNU at analog gain levels {0, 6, 12, 18 and 24} dB levels, at {0, 15, 30, 45, and 60} C° temperatures. For each sensor, gain, and temperature combination two sets of measurement data were recorded, based on $N = 2000$ frame captures in each set. Both sets contained mean images, $\mu_1(T, \alpha)$ and $\mu_2(T, \alpha)$, which were generated by averaging the captured images, as well as standard deviation $\sigma_1(T, \alpha)$ and $\sigma_2(T, \alpha)$ images, by calculating SD for each pixel across the stack of N images. Per-pixel SD allowed estimation of residual temporal noise present in the data.

3.8.1 Temporal VS FPN noise

Before continuing with the analysis of noise on image stacks captured with FF illumination, it is important to characterize noise sources. At higher illumination levels, temporal noise is dominated by shot noise, the collective effect of the quantum nature of light. The actual number of photons captured during the exposure period follows a Poisson distribution. If the mean number of photons captured is ν, then the SD of shot noise is $\sqrt{\nu}$. The maximum number of photons converted to electrons is limited by the full well capacity, ν_{max} of the sensor, as well as the saturation level of the ADCs following the PGAs. Thus, to get to an expected digital output level, such as 70% white level, with higher analog gains, fewer photons need to be captured. When using analog gain α, SD of the shot noise associated with the photon flux with be reduced by $\sqrt{\alpha}$ at the photodiode, but this noise, superimposed on the signal, is then also amplified by the PGA:

$$\sigma(\alpha) = \alpha\sqrt{\frac{\nu}{\alpha}} = \sqrt{\alpha\nu}. \tag{36}$$

Effectively, when comparing FF images captured with different analog gain settings resulting to similar output white levels, SD is expected to scale with the square root of gain applied. Figure 33, plotting the measured standard deviations $E[\sigma_1(T, \alpha)]$ on a lin-log scale confirms this expectation.

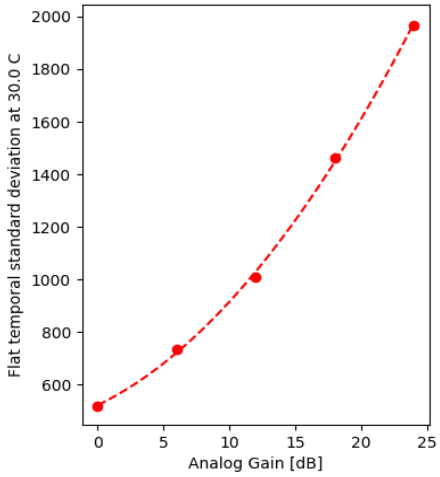


Figure 33. SD of Shot noise as a function of analog gain (α)

The 24.0dB (maximum gain for the IMX265 and IMX273) applies a factor of 16 amplification, for which a 4x increase in shot noise amplitude is expected.

Averaging over N images reduces the SD of shot noise by a factor of $\sqrt{N}$. Based on SD measurements, SD of temporal noise present on image stack means $\mu_1(\alpha, T)$ and $\mu_2(\alpha, T)$ is expected to range from 11.2 LSBs ($T = 0\,°\text{C}, \alpha = 0.0\,\text{dB}$) to 44.72 LSBs ($T = 60\,°\text{C}, \alpha = 24.0\,\text{dB}$), due to averaging.

It is also worth noting that the distribution of per-pixel standard deviation $\sigma_1(\alpha, T)$ is not necessarily Gaussian (Figure 34). Per the central limit theorem, the effects of multiple, uncorrelated noise sources (such as shot noise, thermal noise, and electronic noise) with different means and standard deviations superimposed on pixel outputs present as a single Gaussian even if the distributions of the individual noise sources were not Gaussians. However, if a color image sensor with Bayer Color Filter Array present over the pixel array was tested, a multi-modal distribution, with three distinct Gaussians pertinent to pixels with red, green, blue pigments is expected. Due to the differential spectral absorption of the dies, different sensitivities translate to different photon counts, in turn different means and standard deviations for shot-noise distributions. For a monochrome sensor, Figure 34 is proof of a continuum of different sensitivities, which effectively is the definition of PRNU present.

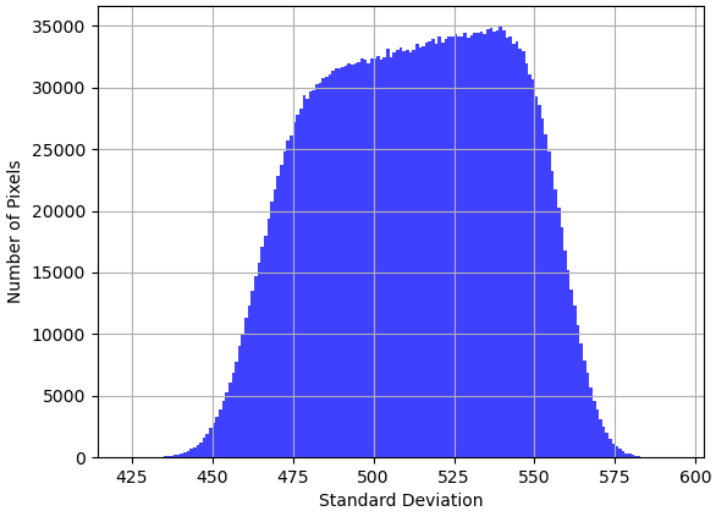


Figure 34. Distribution of per-pixel SD at T=30°C, α=0.0dB

To analyze noise variance on the pixel output, all temporal noise dependent on illumination and gain, like shot noise, is combined into $n_s(L,\alpha)$ and all other temporal noise sources, such as thermal-, reset-, and electronic noise is factored into $n_T(\alpha,T)$. Substituting this into equation (16), with $N'_{x,y}(\alpha,T)$ representing the output pixel value, assuming uniform gray illumination (L):

$$N'_{x,y}(\alpha,T) = T_{int}R_{x,y}L\alpha + D_{x,y}(\alpha,T) + n_s(L,\alpha) + n_T(\alpha,T). \tag{37}$$

For FF measurements exposure time T_{int} is set such that the expected output value remains constant (70% of the white value, N_{max}) for the chosen illumination intensity (L) and analog gain α. Hence $LT_{int}\alpha$= 0.7 N_{max} is a constant factored into the PRNU: $r_{x,y} = T_{int}R_{x,y}L\alpha = 0.7N_{max}R_{x,y}$.

Since DSNU for all temperature and analog gain combinations used in the PRNU analysis was also recorded, captured pixel values can be corrected with the DSNU:

$$n_{x,y}(\alpha,T) = N'_{x,y}(\alpha,T) - D_{x,y}(\alpha,T), \tag{38}$$

$$n_{x,y}(\alpha,T) = r_{x,y} + n_s(L,\alpha) + n_T(\alpha,T). \tag{39}$$

Assuming statistical independence between the noise sources, noise variance on the output is:

$$\sigma_n^2 = \sigma_r^2 + \sigma_s^2 + \sigma_T^2 = \sigma_r^2 + \sigma_t^2. \tag{40}$$

The first noise term, σ_r is pertinent to the fixed pattern PRNU, which is to be minimized. Suppression of temporal noise term σ_t is discussed in section A.2.2.

3.8.2 Analysis of Flat-Field image Stacks

SD of the flat-field image stack (σ_n) and of temporal noise (σ_t) are directly observable. When the two sets of mean images, $\mu_1(\alpha,T)$ and $\mu_2(\alpha,T)$, of the image stacks are combined, in the sum

$$s(T,\alpha) = \frac{\mu_1(\alpha,T) + \mu_2(\alpha,T)}{2}, \tag{41}$$

the SD of temporal noise is suppressed by a factor of $\sqrt{2N} \approx 63.24$, but the averaging does not affect the fixed-noise component:

$$\sigma_s^2 = \sigma_r^2 + \frac{\sigma_t^2}{2N}. \tag{42}$$

In the per-pixel *differences* of the two sets of mean images, the FPN term is eliminated,

$$d(T,\alpha) = \frac{\mu_1(\alpha,T) - \mu_2(\alpha,T)}{2}, \tag{43}$$

with the difference image the residual temporal noise after averaging:

$$\sigma_d^2 = \frac{\sigma_t^2}{2N}. \tag{44}$$

Figure 35 illustrates the SD of temporal noise as a function of temperature and analog gain used. As expected, thermal noise $n_T(\alpha,T)$ increases with temperature, and shot noise, $n_s(L,\alpha)$ increases with analog gain α.

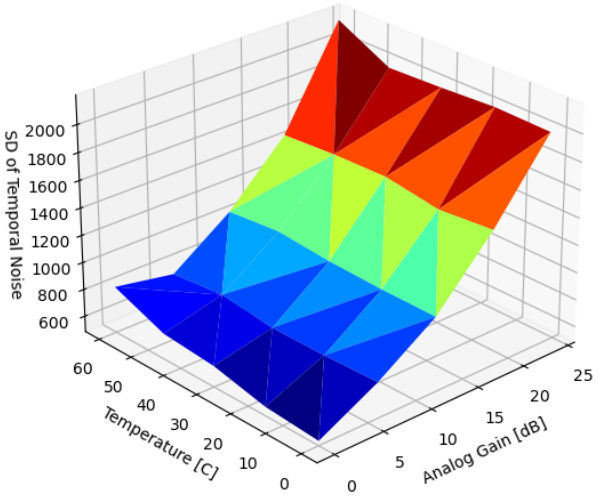


Figure 35. Standard deviation of temporal noise $\sigma_t(\alpha, T)$

3.8.3 Standard Deviation of Uncorrected PRNU

From variance measurements of the averaged means $\sigma_s(\alpha, T)$ and $\sigma_d(\alpha, T)$ the SD of the fixed pattern PRNU component can be computed:

$$\sigma_r = \sqrt{\sigma_s^2 - \sigma_d^2}\,. \tag{45}$$

Figure 36 illustrates the dependence of PRNU over analog gain and temperature. While the shape of the function may suggest that analog or digital processing techniques are used inside the sensor to improve uniformity at medium gains, the drop around 10dB is relatively insignificant in comparison to the average PRNU levels.

It is also worth noting that non-uniformity is only visible, though subtle, at very low gains where shot noise is at minimum. Above 0.5 dB gain a human observer could not perceive FPN on video as it was deeply buried in temporal noise.

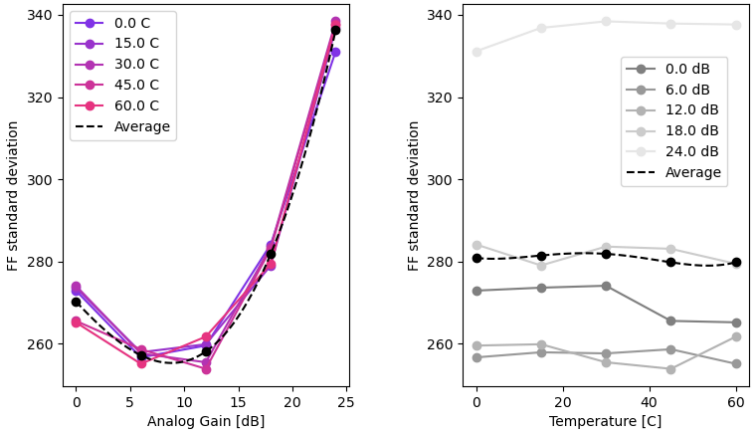


Figure 36. Projections and averages of $\sigma_r(\alpha, T)$

3.8.4 Single point correction

Selecting a single correction image is the simplest way to calibrate non-uniformity and has demonstrated good results [36]. For suppression of visible PRNU artifact, selecting a calibration frame in the middle of the temperature range, and at analog gain = 0.0dB removed all visible

artifacts (Figure 37). Analysis of PRNU recorded at different temperatures and analog gains (Table 5) reveal very high correlation across the entire temperature range. This finding augments the findings of Figure 36, suggesting that PRNU is stable across the operating temperature range.

Analog gain [dB]	Temperature [°C]	SD FF stack	SD temporal	SD PRNU	Pearson corr.	SD Residual	Residual PRNU
24	0	333.00	31.78	331.48	0.6533	255.42	253.44
18	0	281.70	23.36	280.72	0.7972	176.40	174.85
12	0	260.08	16.29	259.57	0.8630	139.52	138.56
6	0	258.91	11.78	258.64	0.9416	91.46	90.70
0	0	273.06	8.19	272.93	0.9967	22.06	20.48
24	15	341.27	31.94	339.77	0.6540	261.63	259.67
6	15	258.00	11.60	257.74	0.9403	92.46	91.73
0	15	271.51	8.68	271.37	0.9979	17.66	15.38
24	30	342.40	31.56	340.94	0.6369	267.24	265.37
6	45	255.11	12.37	254.81	0.9358	95.37	94.56
0	45	271.25	10.64	271.04	0.9982	16.06	12.03
24	60	351.97	34.39	350.29	0.6313	276.44	274.29
18	60	283.38	23.16	282.43	0.7790	184.00	182.54
12	60	259.39	16.55	258.86	0.8598	140.57	139.60
6	60	256.88	11.78	256.61	0.9343	97.19	96.48
0	60	257.24	13.03	256.91	0.9973	19.33	14.28

Table 5: SD of PRNU and residual, single reference correction

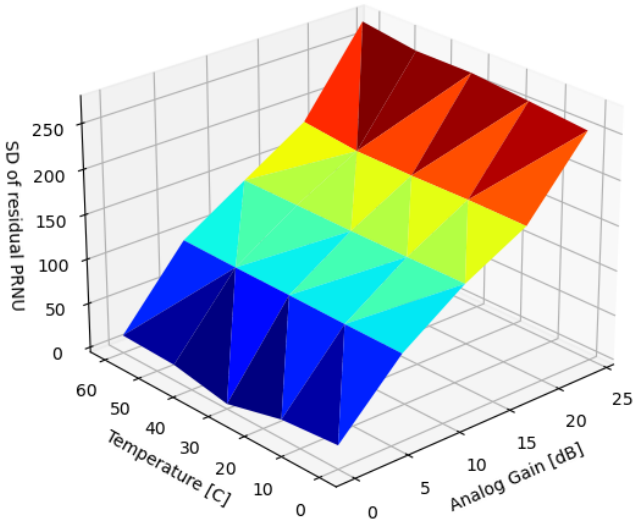


Figure 37. Residual SD of PRNU, single reference correction

3.8.5 Multi-point correction

High correlation along the temperature axis suggests that using multiple reference frames along the gain axis can minimize SD over the entire parameter range. Even though multi-point correction can reduce worst-case PRNU by a factor of 4 (Figure 38), a practical implementation of this method requires capturing multiple PRNU reference images in a temperature-controlled environment. The large number of images to capture for each image stack and reference image may be prohibitively expensive in a production environment.

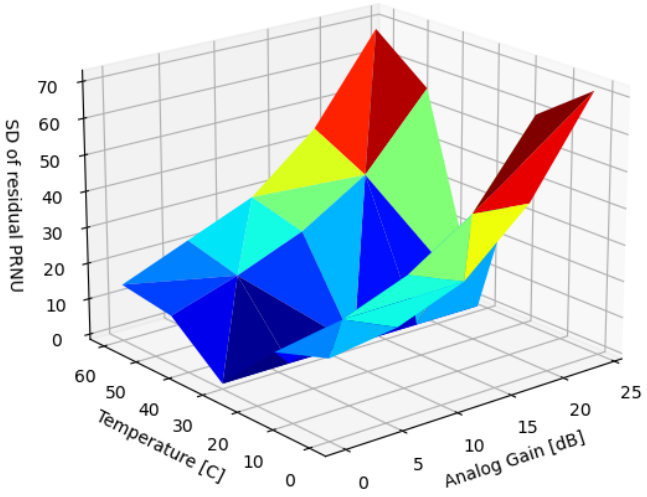


Figure 38. Residual SD of PRNU, multi-reference correction

Single point correction reduces PRNU related noise an order of magnitude under temporal noise under the same imaging conditions, which for most practical applications may be sufficient to achieve required fixed pattern noise performance.

3.9 Lens Shading

As mentioned in section 2.10, pixel vignetting is due to a mismatch between the Chief Ray Angle (CRA) of the lens system to the micro-lenses on top of the image sensor pixels. Micro-lenses over the sensor pixels at nadir are more efficient at focusing incident light to the buried photodiode than those closer to the edges of the sensor.

3.9.1 Temperature dependence

Lens spacers, the barrel, and lens elements undergo material expansion. The refractive indices of lens elements are also temperature dependent. Lenses made from materials with low Coefficient of Thermal Expansion (CTE), e.g., glass and metal, are more stable across the operating temperature range. For mobile consumer-, or airborne applications, to reduce cost and weight, plastic lens elements, with higher CTE are used.

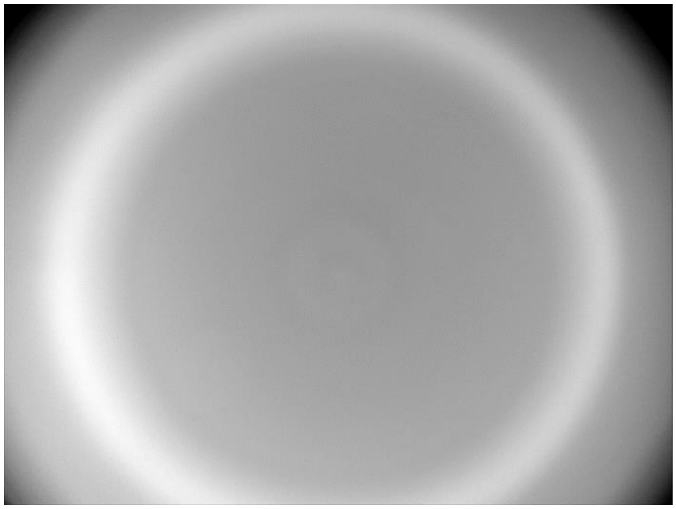


Figure 39. Hybrid lens at 0°C corrected with 60°C image

When a PRNU image captured with the lens assembly at one temperature is used to correct for lens shading at another temperature, the difference can introduce artifacts. Figure 39 demonstrates this problem on a hybrid (glass + plastic) lens assembly.

Figure 39, scaled to improve visibility, was captured by attempting to correct lens-shading of a FF image collected at 0°C, by using a lens-shading correction image collected at 60°C.

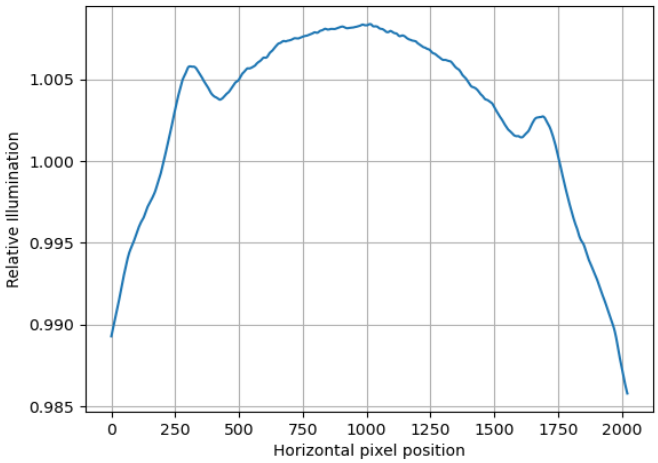


Figure 40. HIP of hybrid lens, 0°C corrected with 60°C

Figure 40 shows the Horizontal Intensity Profile (HIP) of the corrected image. The slight asymmetry of the profile is indicative of temperature dependent warping of the lens-sensor assembly. The ideal intensity profile would be perfectly flat at 1.0. While a 1.5% divergence may not seem much, the SD of the noise equivalent of this artifact of a 16-bit sensor image is 269 LSBs, which at low analog gains is significantly above uncorrected DSNU, and at par with uncorrected PRNU. Notably, DSNU and PRNU are distributed over the entire image area, and on single captures blend with temporal noise, but lens shading artifacts are concentrated and are discernable even on single captures.

This finding emphasizes the point that to suppress both high-, and low-frequency noise temperature compensated PRNU correction is necessary. As mentioned in section 3.5, the proposed Flat-Field Correction module can correct lens shading via $r_{x,y}$, with a combined LSC and PRNU correction image stored either in a frame buffer or generated by the dedicated LSC module.

PRNU is dependent mostly on analog gain, while LSC is a function of temperature only. For a solution that corrects both PRNU and LSC via $r_{x,y}(\alpha, T)$ an exhaustive set of FF image stacks may need to be recorded, which is often not practical for mass-manufacturing.

3.9.2 Complex Lens assemblies

For variable focus or zoom lens assemblies, where lens shading is dependent on temperature and the position of lens elements or groups, parametric characterization of the lens assembly may be necessary. To perform characterization, FF images for a single, module specific lens assembly, or a set of N_L sensor module samples are captured at combinations of different temperature (T), lens focus (f) and zoom (z) settings. The recorded images are low pass filtered, and if multiple samples were used, averaged over the sensors in the sample set:

$$ls_{x,y}(T,f,z) = \frac{1}{N_L}\sum_{s_{ID}}^{N_L} \hat{G}\big(F_{x,y}(s_{ID},T,f,z),\sigma\big), \tag{46}$$

Where $\hat{G}(F,\sigma)$ is a two-dimensional Gaussian Filter with SD σ, and $F_{x,y}(s_{ID},T,f,z)$ is a FF image of sensor module s_{ID}, captured at temperature T, with the lens assembly present with focus and zoom settings f,z. FF images are to be captured using minimum analog gain, and exposure time adjusted so the center of the resulting images are at 70% full-scale gray values.

The parametric LSC module takes a 32x32 set of coefficients $\boldsymbol{C}(T,f,z)$, derived from the scaled inverse of $ls_{x,y}(T,f,z)$, sampled at a spatially uniform sampling grid spanning the entire image. σ, and the size of the Gaussian filtering kernel $\hat{G}$ is set to equal the sampling grid size, to provide maximum suppression of temporal noise or contamination artifacts without interfering with the sampling intensity drop-off profile. The memory footprint of the resulting set of 32x32 LSC coefficients $\boldsymbol{C}(T,f,z)$ is small enough for FW to update between frames when temperature, or lens parameters change.

3.9.3 Use of Population Images

In order to expedite the manufacturing process, one option is to reuse a population image set, captured with multiple sensor samples, on all image sensor modules manufactured in a batch. However, due to manufacturing tolerances and temperature dependence in both the lens and sensor, unit to unit variations in lens shading and temperature dependence are introduced.

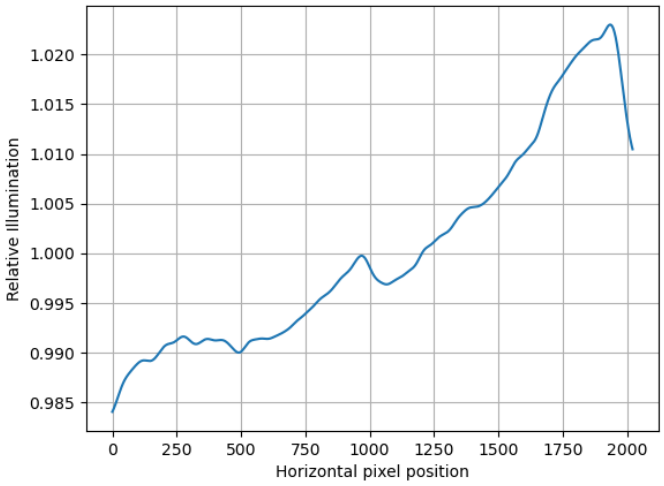


Figure 41. HIP of LSC without de-center correction

Figure 41 presents the Horizontal Intensity Profile of an FF image from one image sensor compensated for Lens Shading with a population image. While dozens of manufacturing parameters may vary module-to-module during manufacturing, optical de-center is the primary parameter to be controlled to enable use of population images.

The best way to control de-center is the use of Active Alignment (AA) during manufacturing, which bounds the tolerated de-center and angular alignment error of image sensor modules produced.

Another option, in lieu-, or on top of AA, is to measure optical de-center ($c_{x,y}$) as part of Image Quality Control (IQC) and adjust coefficients $\boldsymbol{C}(T,f,z)$ to compensate for $c_{x,y}$ before committing it to NVME storage on the image sensor module.

3.10 Chapter Summary

I provided an efficient FPGA implementation to suppress dark signal and photo-response non-uniformity that improved stereo matching results for machine vision cameras.

In the chapter introduction sources of non-uniformity in imaging systems were reviewed, defining DSNU, PRNU and Lens Shading. The dependency of DSNU and PRNU of two modern Sony CMOS machine vision sensors on exposure time, die temperature, and analog gain (Figure 27 and Figure 36) were analyzed.

In section 3.5 **I proposed FFC and ISP architectures, optimized for FPGA or ASIC implementation supporting different noise suppression performance and resource trades.**

I performed in-depth analysis of temperature and analog gain dependence of fixed pattern noise of modern CMOS image sensors. Noise suppression performance of four different algorithms were quantified to suppress DSNU (sections 3.7.3-3.7.7), along with the analysis of embedded software-, and calibration complexity of the different approaches. An algorithm was provided for optimizing the capture parameters of calibration images (section 3.6.6). Performance of two different options were explored to suppress PRNU (sections 3.8.3-3.8.5). Temperature dependence and correction options for lens shading were analyzed in section 3.9.

Based on different use case scenarios, the proposed FFC and ISP architecture can be configured for different performance tiers. For all performance tiers the proposed architecture introduces minimal latency as images are read out from attached image sensors.

For consumer products with inexpensive CMOS sensors and optics, such as webcams, a minimal ISP solution can use population images, and no correction for DSNU or PRNU. The most egregious non-uniformity problem is uncorrected lens-shading. Objectionable to human observers, and detrimental to machine vision and processing algorithms, lens shading can be compensated using just the parametric LSC module in the proposed FFC solution. This performance tier does not require an external frame buffer, VDMAs, or FW initialization of buffers $d_{x,y}$ and $r_{x,y}$.

For video applications where visible FPN is not acceptable, such as DSLR cameras, PRNU and DSNU has to be suppressed. This performance tier requires an external frame buffer to provide $d_{x,y}$ and $r_{x,y}$. If fixed focus optics is used, and temperature compensation of the lens is not a requirement, LSC can be performed by convolving intensity and PRNU correction in $r_{x,y}$.

As described in section 3.7.3 using a single, static image does not correct DSNU sufficiently. The top performance tier is suitable for high end machine vision cameras, studio equipment, or computational photography where motion compensated image stacks are registered to suppress temporal noise. For these demanding applications gain and temperature compensated DSNU, PRNU and LSC are all utilized. Parametric LSC is suggested with module specific, temperature compensated lens-shading parameters accounting for zoom and focus settings. For this tier, FW needs to either calculate $D(T, \alpha)$, or gather image statistics from the OBP region of the sensor and calculate $\sigma(T, \alpha)$ (section 3.7.4). Also, FW may dynamically adjust frame buffer contents to interpolate between DSNU and PRNU frames stored in DDR.

As demonstrated in sections 3.7.3 and 3.8.5, DSNU correction can be significantly improved by using the global DNSU amplifier (G_d) feature of the FFC. PRNU suppression can be improved by using gain dependent calibration images ($r_{x,y}$). For this performance tier, at initialization multiple $r_{x,y}$ images need to be deposited into DDR by FW. During use, FW also needs to read out sensor temperature T, and based on the current analog gain setting α update $G_d(\alpha, T)$ and reprogram the VDMA read controller to point to the $r_{x,y}(\alpha)$ best matched to operating conditions.

4 Defective Pixel Correction

To improve image quality raw sensor images are typically processed by FFC and DPC modules as the first stages of the ISP pipeline. FFC, described in detailed in the previous chapter corrects slight differences in pixel sensitivity and offset. As introduced in section 2.2, some sensor pixels are defective beyond correction. DPC replaces defective pixels with interpolated values using information from neighboring pixels.

Modern CMOS image sensors comprise a light sensitive array of pixels, analog amplifiers, ADCs, digital back end and timing circuitry (Figure 19). The pixel array is inherently monochrome and is sensitive to a broad spectrum of light. Color imaging systems may employ different color filters with multiple sensors, use a single sensor with a Color Filter Array (CFA) laid over the sensor pixels, or a single sensor with a three layer photodiode structure [39]. Most current color imaging devices use a single sensor overlaid with the Bayer CFA [40].

With improvements in process technology and novel architectures such as back-side illuminated pixels, pixel sizes and full-well capacities have been shrinking [16]. While defect densities are reduced, the expected number of pixel defects per sensor is rising, and the yield of defect-free imagers is deteriorating. Sensor manufacturers discard focal point arrays with cluster defects, defective rows, or columns, but tolerate a growing number of single pixel defects towards the periphery of the image.

The human visual system is highly sensitive to outlier pixels especially in flat areas of images. Furthermore, the CFA interpolation process [41],[42] may spread the effects of singular pixel defects to surrounding pixels by introducing color artifacts.

Some computer vision algorithms, such as mid-air collision warning and avoidance, are especially sensitive to pixel defects. This class of algorithms tracks small objects in a high frame-rate video stream. Objects in the 'sky' region of images which persistently observed at a constant angle (fixed x, y position on an imager) relative to a moving aerial vehicle (aircraft, drone, or missile) are on a crash trajectory. Defective Pixels (DPs), present stationary outliers which can cause false alarms or course corrections.

For every frame produced by the sensor, DP values have to be dynamically replaced with values derived from surrounding pixels. Which surrounding pixels can be used depends on the defect type and the sensor configuration; monochrome sensors can use the nearest neighbors, while sensors with Bayer CFA can use either a quincunx (green pixels) or rectangular (red, blue) grid of same color pixels.

4.1 Pixel Defect Types

In terms of spatial configuration, defects in image sensors may affect single pixels, pixel clusters, complete rows, or columns. Cluster defects are usually caused by doping contaminants, or particles shading the micro-lens array affecting multiple neighboring pixels. Defective rows or columns may be attributed to faulty row drivers, column amplifiers, shorted metallization, or a faulty element in the chain of charge coupled transistors in a CCD. Sensors exhibiting severe defects introduced during manufacturing can be discarded by quality checks performed by the vendor.

Single pixel defects arise over time due to high energy ionizing particles impacting photodiodes [43]. Generation rate of defects is a function of altitude and is especially a concern for aerospace camera applications.

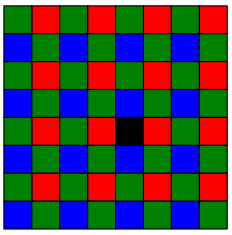


Figure 42. Example of a dead pixel in a sensor with Bayer

Pixel defects (Figure 42) may be characterized as dead (always dark), hot (always light, characterized by very high dark current), or flickers (excessively noisy). Using a linear model of photoconversion, pixels may be considered defective based on dark signal non-uniformity (DSNU), or photo-response non-uniformity (PRNU). FFC characterizes the DSNU and PRNU of each pixel in the array and establishes coefficient matrices correcting dark levels and photon-responses. When the photon-response correction coefficient larger than 1.0 is applied, shot and thermal noise is also amplified for the pixel.

A practical metric to determine if a pixel is defective or not is to estimate additional noise introduced by extrapolating the pixel value using FFC versus interpolating the pixel using neighboring pixel values (DPC). This criterium is usually a static rule manifested as a PRNU threshold established off-line.

Pixels defects may be permanent, or temporal, based on current ambient conditions, such as dependence of DSNU and PRNU values on temperature and sensor settings.

DPC is a two-step process: DPs need to be identified, then interpolated using information from surrounding pixels. Identification of DPs can be performed statically, or dynamically.

4.2 Static Identification

Static solutions take advantage of a calibration step, often performed as part of the FFC calibration, either during sensor-, or camera device manufacturing. In calibration mode the sensor is typically exposed to a dark, then a flat medium gray visual field. The dark field image is collected with the sensor covered. The flat-field is captured either in an integrating sphere or facing a light panel, with-, or without the optical assembly present. Exposure time and analog gain is usually set such that pixels are neither over nor underexposed. A global histogram of pixels is computed to quantify sensor quality, outliers (hot and dead pixels) are identified as DPs, and their coordinates are stored in a defective pixel table.

Figure 43 presents a simple DPC solution using a static defective pixel table and linear interpolation. As video pixels are streaming in scanline order, row-, and column counters keep track of the current pixel position of the frame. If the x, y coordinates of the pixel are flagged in the defective pixel table, the input pixel is replaced by a horizontally interpolated pixel in the output stream. For monochrome sensors, the immediate neighbors can be used. For color sensors, due to the structure of the Bayer pattern, same color values from two pixels over need to be used for the interpolation. For the sake of simplicity Figure 43 omits provisions for exception handling, e.g., interpolation of edge pixels, or cluster defects, when one of the neighbors should be excluded from interpolation.

Although simple in construction and robust in identification, static DPC solutions require an extra manufacturing step, and cannot account for temporary DPs. Most notably, static solutions cannot identify defects introduced during use, after the calibration step was performed.

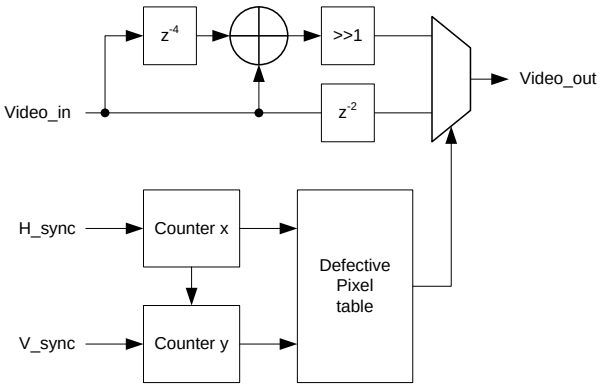


Figure 43. Static solution for Stuck Pixel Correction

4.3 Dynamic Identification

Dynamic identification of DPs is by far the harder problem and the primary focus of this paper. Dynamic solutions analyze the sensor data stream and look for outlier pixels, which are apparently stuck or flicker independent of neighboring pixel values and local motion.

4.4 Related Work

Early approaches for CCD DPC by Meynants [44] and Reinhant [45] used only a single line of pixels, with no frame or line buffers to reference against. These methods are computationally efficient but often produce visually poor results, particularly in regions with high horizontal frequencies.

More advanced methods utilize a two-dimensional interpolation kernel [46], calculate inter-frame differences, and cumulative variances. For all pixels, an external frame buffer is used to store previous values and accumulated variance. Pixel variance under a threshold indicates that the pixel is stuck (low or high). High variance above a threshold over local average indicate a flickering pixel. In both cases the pixel is concealed by 2D or 3D interpolation. In order to calculate and store inter-frame differences the frame buffer need to be accessed 4 times (Read/Write previous value, Read/Write variance). On-chip frame buffers are typically very costly in FPGAs. Access to an external frame buffer reduces memory bandwidth available for higher level processing algorithms. In [S4] I proposed a solution to this problem without addressing flickering pixels.

An edge-dependent adaptive filtering based solution has been proposed by Tanbakuchi et. al [47], identifying a predefined number of DPs. The intensities of possible DPs were compared with intensities of the same locations in a subsequent frame after a scene-change was detected. Possible DPs which did not change intensities over the scene-change were stored in a separate permanent memory.

Interpolation has extensive literature, with many trades offered between complexity and quality of results. Linear methods use bilinear, or higher order kernels using nearest neighbors. Non-linear methods employ median or rank-order filtering as proposed by Li [48]. A comparison between the efficacy of these methods is presented by Tchendjou in [49]. Complex interpolation methods, such as motion-compensated, three-dimensional interpolation described by Adil et. al in [50]. This

approach can suppress large cluster artifacts, using information from previous frames. However, a frame buffer, and sufficient computational resources are necessary to calculate optical flow.

Baharav et. al [51] introduced a dynamic detection solution without a frame buffer by detecting outliers first, but the solution lacks temporal tracking of DP candidates and immediately interpolates outliers if their nearest neighbors are not outliers. In video streams this may lead to flickering artifacts.

4.5 Proposed Solution

The typical spatio-temporal filtering method described identifies non-changing pixels first (temporal), then distinguishes between large stationary areas, which are likely to be non-changing parts of the image, and singular outliers, which are likely to be DPs. *Memory requirements can be significantly reduced by performing spatial filtering first*. Spatial filtering reduces the number of DP candidates, along with the amount of information needed for temporal filtering, therefore facilitates spatio-temporal filtering in embedded systems with limited-, or no access to external memory.

4.5.1 Algorithm

Two-dimensional spatial filtering first identifies potential DPs, and at the same time eliminates pixels which blend into their local neighborhoods, therefore may not need to be substituted even if they are defective.

At the core of the solution is a list of DP candidates, stored in scanline order. Each candidate entry in the list contains the:

- pixel scan-line index (de facto x, y coordinates),
- previous value of the pixel,
- age of the entry expressed as the number of frames since the pixel was entered into the list,

an optional flag (keep), to force retaining the entry.

If the input pixel is already in the candidate list, verify whether the current pixel value is within a predefined (ε_1) range of the previous value stored for the pixel. If the value is within range, increase the age of the entry, otherwise remove the pixel from the list unless the 'keep' flag is set. This flag enables using static identification results. Statically identified DPs are typically stored in non-volatile memory (NVM) next to the sensor and can be uploaded to the DPC module by the module FW driver.

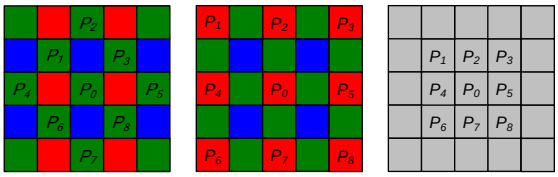


Figure 44. Nearest neighbors for Bayer CFA, and monochrome sensors

For each input pixel, two dimensional spatial filters evaluate the minimum distance between the pixel and its immediate neighbors, normalized by the standard deviation:

$$d = \frac{\min_k((P_0 - P_k)^2)}{\sqrt{\underset{k}{var}(P_k)}}, k = 1..8 \tag{47}$$

$$\underset{k}{var}(P_k) = \frac{1}{8}\sum_{k=1}^{8} {P_k}^2 - \mu^2, \qquad \mu = \frac{1}{8}\sum_{k=1}^{8} P_k \tag{48}$$

where P_k are the 8 nearest, same-color neighbors. Based on sensor configuration, the locations of the nearest, same-color neighbors may vary (Figure 44).

Figure 45 illustrates the algorithm for handling input pixels.

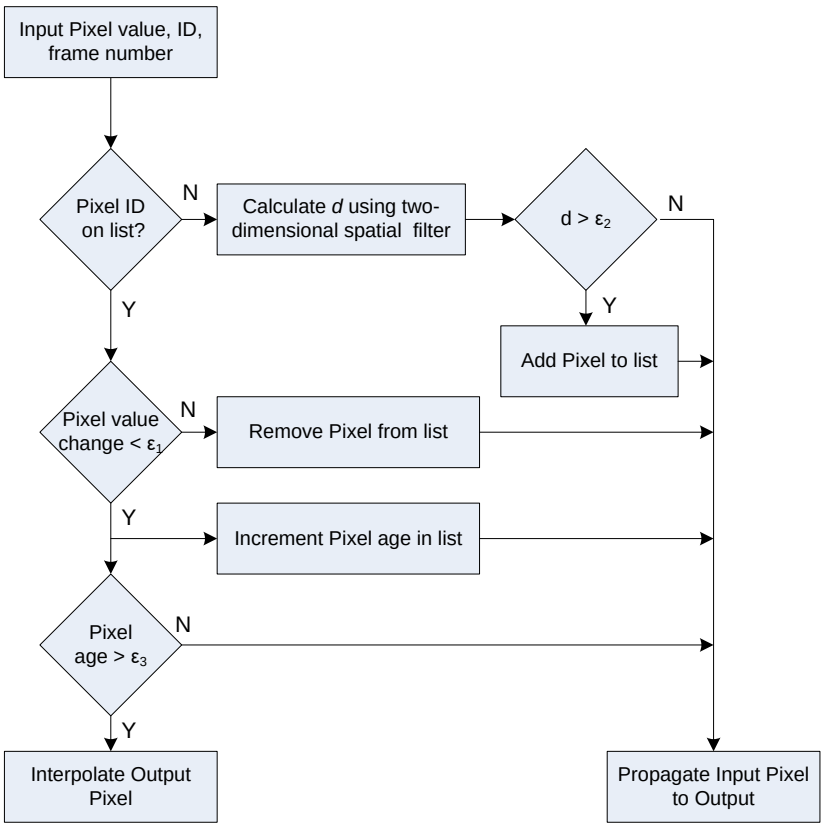


Figure 45. Simplified flow-chart of the proposed algorithm

4.6 Constrained Randomization

The number of DP candidates (spatial outliers) per frame could far exceed the number of actual DPs. As the proposed algorithm identifies outliers in a scan-line continuous fashion, pixels closer to the top of the image would be represented in the candidate list with higher probabilities than pixels closer to the bottom of the sensor image. If memory depth limitations of a hardware implementation prohibit monitoring all false positive candidates identified in an image, true DPs on the bottom of the image may never be added to the candidate list. Therefore, new DP candidates should only be added to the list after a random hold-off period if all memory available for list storage is consumed. This randomization de-correlates start of DP candidate insertion from frame coordinates, and results to a uniform distribution of candidates across all image regions. As false candidates are eventually removed from the list and actual DPs are located, the algorithm converges to identifying all true DPs, provided that the memory allocated can store the total number of DPs defined by parameters $\varepsilon_1, \varepsilon_2$ and ε_3.

The randomized hold-off period should be constrained as follows. If the period is less than one frame, then false positives on the list are likely not yet cleared. The longer the hold-off, the longer it takes for true positives to be added to the list, and interpolation to begin. Therefore, hold-off, expressed as pixel counts, should be randomized between one full-frame to tens of frames, dependent on video content. For relatively static content (e.g., security cameras) more frames may be necessary to remove false positives from the candidate list. For mobile sensors, such as sensors in autonomous vehicles, fewer frames with dynamic content may suffice to rid the list from false positives.

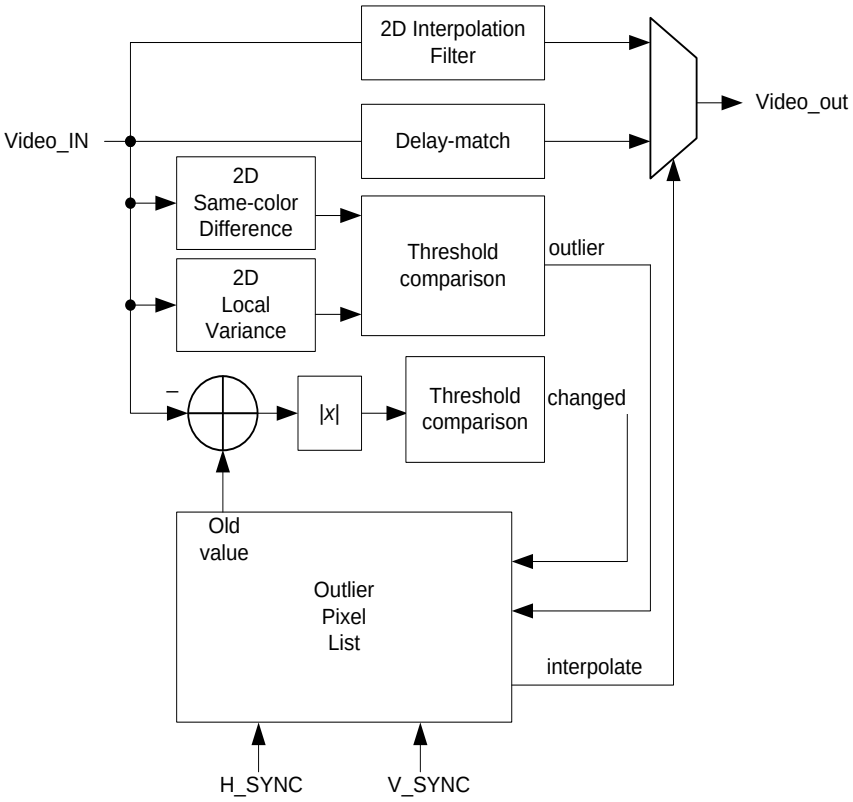


Figure 46. Block Diagram of an FPGA implementation

4.7 Flicker defects

As presented, the algorithm can detect stuck (dead, or hot) pixels, but not flickering pixels. If flicker detection is required, candidate list entries can be appended with a cumulative variance term, summing the squared differences between current and previous pixel values. A fourth thresholding operation, comparing the cumulated squared differences with a constant (ε_4) multiplied by the number of frames the candidate pixel had been tracked (age), can identify flickers.

4.8 Hardware Implementation

With some optimizations the proposed algorithm lends itself well for FPGA or ASIC implementation. The architecture defined in Figure 46 takes sensor stream video pixels (Video_In), as well as horizontal-, and vertical synchronization signals (H_SYNC, V_SYNC) as inputs. With a fixed delay it generates an output pixel synchronous to input synchronization signals. Output latency is a function of the 2D interpolation method used, described in detail in the next section.

The output pixel is either interpolated, using a 2D interpolation kernel, or fed forward from the input matching the delay of the interpolation kernel. The interpolation decision is based on whether the current input pixel is marked for replacement in the candidate list.

4.9 DP candidate list implementation

The candidate list is implemented as a dual-ported Block RAM based first word fall-through FIFO.

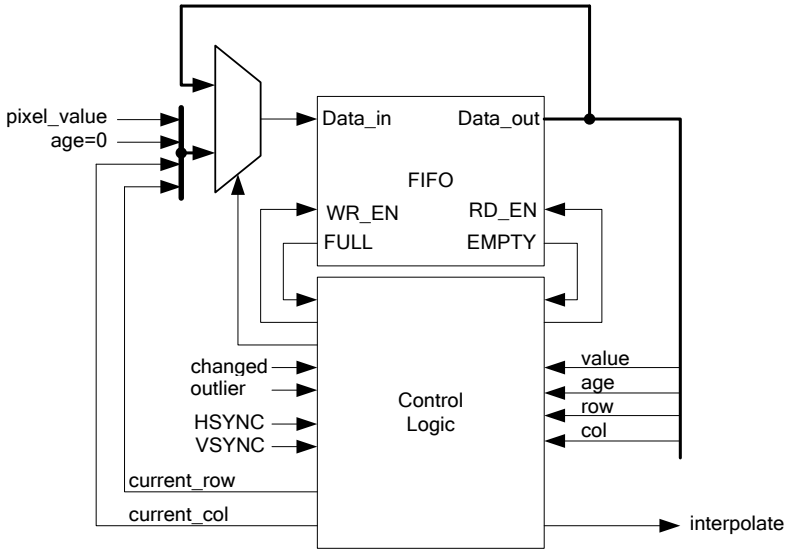


Figure 47. Block Diagram of an FPGA implementation

The output of the FIFO presents the row and column coordinates of the next pixel on the list. If the current row and column coordinates match the presented coordinates, the entry is read out (removed) from the FIFO. If based on absolute pixel value change the candidate pixel needs to remain on the list, the entry is written back to the FIFO. A multiplexer can either feed the FIFO output back into the input or insert a new pixel value into the FIFO based on the pixel qualifier control signals generated by the arithmetic and thresholding logic. If 2D local variance and color-difference indicates that the current pixel is an outlier which is not on the list, the pixel is written into the FIFO as a new entry if hold-off requirements are met. In each frame, all values from the FIFO are read out, with only those outlier candidates written back into the FIFO whose pixel values stayed in the range defined by ◎1. Also, new outlier candidates can be inserted into the appropriate positions between the existing FIFO elements, such that the scan-line continuous order of the pixels stored within the FIFO is maintained.

Software simulations confirmed that without a visible degradation in performance in (47) the minimum of squared differences operation can be effectively replaced by the minimum of absolute differences, while the standard deviation in (47) can be replaced by the sum of absolute differences (SAD) operator:

$$d = \frac{\min\limits_{k} |P_0 - P_k|}{SAD(P_k - \mu)}, k = 1..8 \tag{49}$$

where μ is the mean of the surrounding same color pixel values P_k. In FPGA and ASIC implementations the above changes reduce the arithmetic footprint considerably, replacing multipliers with adders. Combined with the thresholding operation $d < \varepsilon_2$, the condition can be rewritten as:

$$\min_k |P_0 - P_k| < \varepsilon_2 SAD(P_k - \mu), \tag{50}$$

replacing a block divider with a multiplier, which is available next to the Block RAM resources used for implementing the FIFO and the line buffers in current FPGAs.

Randomization is implemented in hardware with an LFSR based pseudo-random generator, which initializes a counter with a value between the frame-size and a constant multiple of the frame size. If and only if the candidate FIFO gets full the counter stars counting down by every input pixel. Irrespective of frame boundaries, new DP candidates are not written into the FIFO until the counter reaches its terminal value, however existing DP candidates are evaluated and may be removed from the FIFO.

4.10 Interpolation

The interpolation method chosen for DPC need to consider that nearest, same-color neighbors may be defective as well. Also, DPC interpolation artifacts may introduce false colors during subsequent CFA interpolation in the image processing pipeline. Nonlinear filters, such as 2D median filters, typically preserve edges well, but may introduce flickering to video streams when noisy samples in the ordered list are close to each other in value. Linear interpolation filters may introduce blurring when applied to regions with high spatial frequencies but considering the relatively small area (DPs) where the filters are applied the introduced artifacts are less visible than the flickering introduced by non-linear filtering. Two interpolation approaches, using similar hardware architecture can provide trade-off between performance and resource usage:

- Interpolation including only valid same-color neighbors
- Edge adaptive interpolation of valid, same-color neighbors

Both methods fit FPGA or ASIC implementation well, interpolate flat-, or gradient fields with at most two LSB error, and produces predictable results (no flickers based on low variance additive noise).

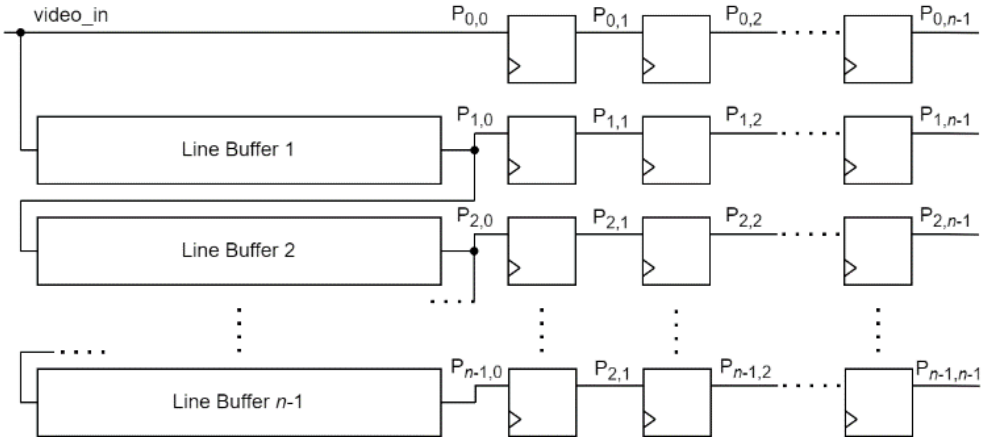


Figure 48. Input pixel matrix for the 2D interpolation kernel

The first part of the interpolator is the pixel-matrix presenting the input pixels around the DP candidate (Figure 48). Besides providing grayscale values, an additional input bit, indicating the validity of the corresponding pixel is provided with video_in. A pixel is considered valid if it is not on the list of actively interpolated pixels. If one of the 8 nearest neighbors is invalid, the diagonally opposite neighbor is considered invalid as well. This method results to symmetrical interpolation kernels, which in turn guarantee the flat-, or smooth gradient fields are interpolated within

rounding error. Pixel values and valid flags of the 8 nearest neighbors are passed to an adder tree, which combines pixel values. Each registered adder in the tree takes two pixels with a bit corresponding to the validity of the pixel and generates a summed output and a bit corresponding to the validity of that sum. Each adder node in the tree performs the following operation, based on the validity of input pixels:

A valid	B valid	output	output valid
true	true	A+B	true
true	false	2A	true
false	True	2B	true
false	False	don't care	false

Table 6: Output of adder stages based on input validity

Diametrically opposite nearest neighbors are paired for the first adder stage. The second adder stage combines averages from perpendicular directions, e.g., the horizontal average is combined with the vertical one. Method (B) optionally calculates not only the sum, but also the absolute difference between the pixel pair of stage one. If s_H and s_V are the horizontal and vertical sums, calculated by the first adder stage, and d_H and d_V are the absolute differences incremented by one to avoid division by zero, not only the validity, but differences can be factored in on stage two:

$$\frac{s_H d_V + s_V d_H}{d_H + d_V} \tag{51}$$

This method prefers the average across more similar pixel values, leading to edge adaptive interpolation. Equation (51) can be implemented in hardware without a division operator as described in [S6]. Similar to combining horizonal and vertical averages, diagonal averages can be combined weighing the corresponding absolutes differences.

The resulting weighed sum effectively implement edge adaptive interpolation, steering the filter to combine pixel values along edges, as opposed to across edges. This in turn reduces color artifacts introduced by CFA interpolation.

4.11 Parameterization

The robustness of an image processing algorithm or system often depends on how easily threshold values can be matched to the particular stimuli received. Parameters $\varepsilon_1, \varepsilon_2, \varepsilon_3$ can be set adaptively by firmware irrespective of the scene content, with little a-priori information of the sensor and its application.

Threshold value ε_1 defines how stable a pixel value needs to be in order to be classified as stuck. The lower the value, the lower the chance that slowly varying pixels get characterized as stuck (false positives). However, in high noise scenarios, or when blooming may modify readout values of dead pixels, ε_1 may need to be increased to identify all stuck pixels. Based on simulation results, as a practical value for ε_1, the square root of the maximum pixel value produces optimal results.

Threshold value ε_2 defines how different a pixel needs to be from the surrounding pixels in order to be classified as an outlier. A higher threshold value for ε_2 results to a lower number of outlier candidates and slower convergence time for identifying all DPs, but at the same time returns fewer false positives. If heuristics for the total number of DPs (N) are known, a feedback mechanism,

implemented in FW, may effectively adapt the filter, and dynamically tune ε_2 such that the number of DPs in the list approximates N.

Threshold value ε_3 defines the number of frames presumed outliers have to hold their values within the ε_1 range before replacement (interpolation) begins. The higher the value of ε_3, the less flickering due to DPC the algorithm produces, but also the longer it takes for the algorithm to converge and start replacing the pixels. A constant value in the range of 100-10000 allows virtually no flickering while identifying outliers within minutes. If fast convergence is more important than eliminating temporary DPs, ε_3 is suggested to track the total number of frames (T) since the sensor was turned on. In simulations a value of $T/5$ produced optimal results.

4.12 Results

Actual resource counts and maximum frequencies vary depending on the input frame size, the expected number of DPs (N) and the target FPGA family. For color sensor processing (5x5 kernel), 12 bit color representation, 4 line buffers take 3 BRAM36 primitives. The DP list, supporting up to 1024 DP candidates, can be stored in 4 Block-RAMs.

FPGA part	BRAM36s	DSP48s	FFs	Fmax
XC7V585T-1	3	2	2065	280 MHz
XC7K70T-1	3	2	2065	258 MHz
XC7A100T-1	3	2	2065	180 MHz

Table 7: Performance and resource use for Xilinx FPGAs, method (A)

In [49] Tchendjou proposes an algorithm with a similar structure (spatial detection-, and spatial interpolation filter, with a circular verification buffer). Their approach can detect 3x3 cluster defects (which in practice are rare due to manufacturer screening of image sensors) but cannot detect flicker defects. Compared to their implementation results obtained post synthesis targeting a Xilinx Zynq ZC702 part, my proposed algorithm and implementation uses the same number of BRAM primitives and registers, but 50% fewer DSP slices than their DPD_M implementation, and less than 25% of BRAM and registers than their higher quality DMD_R approach. In terms of operating frequencies my proposed algorithm is 2.5 times more performant (258MHz vs 106MHz) than DPD_M, their simpler approach.

4.13 Chapter Summary

An adaptive spatio-temporal filter algorithm and architecture for FPGA or ASIC implementation was proposed to identify and correct defective pixels in the video stream of an image sensor. **I provided an algorithm and architecture for automatic detection of single-pixel and cluster defects on focal-plane array images, optimized for FPGA implementation.**

In comparison with existing methods, the proposed architecture requires no external frame-buffer access, therefore lends itself well for FPGA and VLSI implementation of streaming ISPs. The solution can effectively identify slowly varying temporarily defective pixels, with excellent convergence speed and accuracy, can also detect flickering pixels, and supports monochrome or color (Bayer CFA) sensors. Results pertinent to this thesis were published in [S5].

5 White Balance Correction

In photography and image processing, color-, or white-balance adjustment is the correction of color intensities to render specific colors, particularly neutral colors, correctly. White balancing corrects the colors of a scene to appear as if the scene was imaged with natural lighting, irrespective of the illumination source used. Real-time color-balancing is becoming increasingly challenging as resolutions and frame-rates of industrial, consumer and automotive video applications increase. The method I developed illustrates how a custom hardware accelerator gathering image statistics, a pixel-level color correction module, and an algortihm with $o(1)$ algorithmic complexity implemented on an embedded processor can achieve excellent results. The algorithm has no dependencies on any particular library, or HW features, and can be ported to a soft processor like MicroBlaze, or an advanced ARM core to control custom image / video processing logic.

The measured color and intensity of reflections from a small, uniform surface element with no inherent light emission or opacity depend on the following functions:

- The spectral power distribution of the illuminant $I(\lambda)$
- The spectral reflective properties of the surface material $S(\overline{x}, \lambda)$
- The spectral photo-sensitivities of the imager $P(\lambda)$

where $\overline{x}$ denotes location expressed in camera coordinate space, such as an angle subtended by a Lambertian patch of a scene object to the imager, and λ denotes wavelength of light. The signal power measured by a pixel can be expressed as

$$p = \int_{\lambda_m}^{\lambda_M} I(\lambda)P(\lambda)S(\overline{x}, \lambda)\mathrm{d}\lambda, \tag{52}$$

where $\lambda_M$and λ_m are the maximum and minimum wavelength sensitivity limits of the imaging system. The trichromatic aggregate spectral power measured by image sensor pixels inherently mixes the native color of objects with the spectral content of the illuminator. Illuminant estimation is inherently under-constrained [52], and the identification of illuminant spectrum is based on heuristics about the color content of the scene. Color correction subsequent to illumination estimation aims to restore colors that appear correct regardless the illuminant, such as incandescent or fluorescent lighting.

In order to get a color image, human vision, photographic and video equipment uses multiple, adjacent sensors with different spectral responses. Human vision utilizes 3 different types of light sensitive cone cells, to formulate color perception. CIE has defined a set of three color-matching functions, $\bar{x}(\lambda)$, $\bar{y}(\lambda)$ and $\bar{z}(\lambda)$ which can be thought of as the spectral sensitivity curves of three linear light detectors that yield the CIE XYZ tristimulus values P_x, P_y, and P_z. The tabulated numerical values of these functions are known collectively as the CIE standard observer [53].

Digital image sensors predominantly measure tristimulus values either by a Color Filter Array [40] overlay above inherently monochromatic photodiodes, or by stacked photodiodes [39] measuring the absorption depth of photons, which is proportional to the wavelength λ. Neither method creates spectral responses similar to the human eye, which leads to different color measurements between human observers and image sensors, as well as image reproduction equipment when photographing the same scene.

The purpose of camera color calibration is to transform / correct the tristimulus values measured by a camera or image sensor such that the spectral responses match those of the CIE standard observer (Figure 49, left).

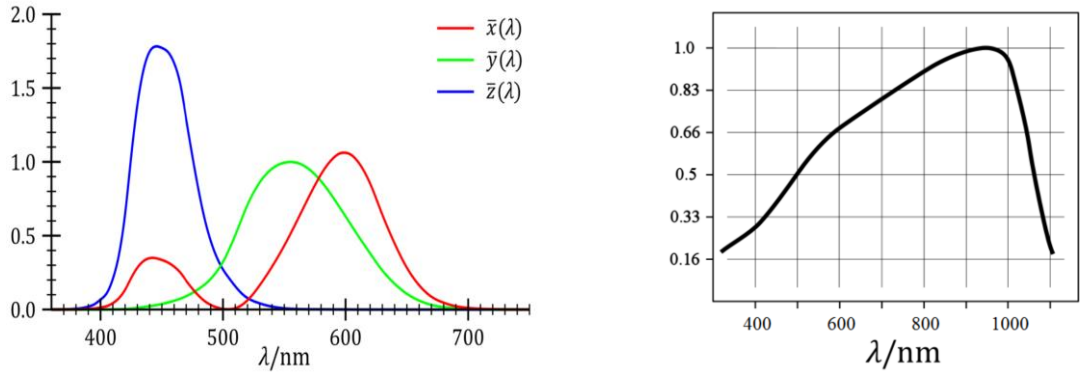


Figure 49. Normalized spectral responses of the CIE Standard Observer, and silicon photodiode

With $\{R(\lambda), G(\lambda), B(\lambda)\} > 0$, $\lambda_m < \lambda < \lambda_M$ representing the sensor specific spectral transmittance properties of the red, green, and blue pigments used in the Bayer CFA of a color image sensor, and $p(\lambda)$ denoting the sensitivity of a silicone photodiode (Figure 49, right), the actual spectral sensitivities of the R,G,B channels can be defined.

The camera correction factors pertinent to color channels, given uniform white illumination $I(\lambda) = i,$, and uniform Lambertian reflectance $S(\overline{x}, \lambda) = s$, are expressed as

$$c_x = \int_{\lambda_m}^{\lambda_M} \frac{\bar{x}(\lambda)}{p(\lambda)R(\lambda)} \mathrm{d}\lambda, \qquad c_y = \int_{\lambda_m}^{\lambda_M} \frac{\bar{y}(\lambda)}{p(\lambda)G(\lambda)} \mathrm{d}\lambda, \qquad c_z = \int_{\lambda_m}^{\lambda_M} \frac{\bar{z}(\lambda)}{p(\lambda)B(\lambda)} \mathrm{d}\lambda \tag{53}$$

5.1 White Balance

Regardless lighting conditions, and the type of illumination used, human vision aims to perceive the objects with their respective colors (object permanence). This feature of the visual system is called chromatic adaptation. Color constancy has been modeled by the Retinex theory [54].

A camera with no adjustment or automatic compensation for illuminant spectrum may register objects with different colors if illumination spectra changes. White balance correction refers to the adjustment of colors within the ISP such that a scene object with surface reflectance independent of wavelength

$$S(\lambda) = s, \qquad \lambda_m < \lambda < \lambda_M, \tag{54}$$

where λ_m=380nm and λ_M=750nm are the minimum and maximum wavelengths pertinent to the sensitivity of the human eye.

According to equation (52), the spectra of the illuminant(s), the reflective properties of scene object(s), and the spectral sensitivity of the detector all contribute to the resulting color measurement. Therefore, even with the same image sensor measurement results will mix information from innate object colors and the spectrum of the illuminant. White balancing therefore is a two-phase operation: illuminant estimation, and color correction. Illuminant estimation, the separation of innate reflective properties $S(\lambda)$ from the spectrum of the illuminant $I(\lambda)$, is possible only if:

- Some heuristics about the spectrum of the illuminant, such as the colors of reference objects, are known.
- Illumination within a scene, or image is assumed uniform. In the related work examined a common assumption is that changes across the image in illuminant spectral composition are negligible in comparison with changes of object reflectance properties within the scene. For example, when photographing a scene with natural sunlight, the spectra of the illuminant remain constant over the entire image. Conversely, when a cinematic image is projected onto a white screen, sharp changes in illuminant spectra are observed, while the reflective properties of the scene (the canvas) remain constant. When both illuminant and reflective properties change abruptly isolation of the scene objects and illuminants is very difficult.
- Detector sensitivity $S(\lambda)$ and the illuminant spectrum $I(\lambda)$ do not have zeros in the range of spectrum observed. No information can be gained about the reflective properties of objects outside the illuminant and detector sensitivity spectrum. For example, when a scene is illuminated by a monochromatic red source, a blue object will look just as black as a green one.

5.2 Related Work

5.2.1 Color Constancy

In digital imaging systems the problem of camera calibration for a known illuminant can be represented as a discrete, 3 dimensional vector function:

$$\underline{x'} = F(\underline{x}), \tag{55}$$

where $F()$ is the mapping vector function, $\underline{x}$ is the discrete (typically 8, 10 or 12 bit) vector of R,G,B principal color components. Based on whether mapping is performed linearly, and whether color-components are corrected independently, the mapping function can be categorized into four quadrants (Table 8).

Color Correction Method	Linear	Non-linear
Independent	Von-Kries	Component Correction
Dependent	Color-Correction Matrix	Full Look-Up Table

Table 8. Characterization of color correction methods

5.2.2 The Von Kries Hypothesis

The simplest, and most widely used method is based on the Von Kries Hypothesis [55], which transforms colors to the LMS color space, then performs correction using only 3 multipliers on a per channel basis. The hypothesis rests on the assumption that color constancy in the human visual system can be achieved by individually adapting the gains of the three cone responses. Cone responses are to be adapted based on the spatio-temporal sensory context, that is, the color history and perceived colors of surrounding objects. Cone responses corresponding to two radiant spectra, $\alpha(\lambda)$ and $\beta(\lambda)$, can be matched by appropriate choices of diagonal adaptation matrices $\boldsymbol{A}$ and $\boldsymbol{B}$ such that

$$\boldsymbol{A}S\alpha(\lambda) = \boldsymbol{B}S\beta(\lambda) \tag{56}$$

where S is the cone sensitivity matrix.
In the LMS (Long-, Medium-, Short-wave sensitive cone-response space),

$$D = \boldsymbol{A^{-1}B} = \begin{bmatrix} L_b/L_a & 0 & 0 \\ 0 & M_b/M_a & 0 \\ 0 & 0 & S_b/S_a \end{bmatrix} \tag{57}$$

is a diagonal matrix, enabling independent adjustment of color channel gains. The advantage of this method is its simplicity and compact implementation with 3 multipliers either as part of the image sensor, or the ISP:

$$\begin{bmatrix} L' \\ M' \\ S' \end{bmatrix} = \begin{bmatrix} k_L & 0 & 0 \\ 0 & k_M & 0 \\ 0 & 0 & k_S \end{bmatrix} \begin{bmatrix} L \\ M \\ S \end{bmatrix} \tag{58}$$

In a practical implementation, instead of using the LMS space, the RGB color-space is often used to adjust channel gains such that one color, typically a light gray tone, is represented by equal R,G,B values. However, adjusting the perceived cone responses or R,G,B values for one color does not guarantee that other colors are represented faithfully.

5.2.3 Independent Component Correction

For any particular color component, the Von-Kries Hypothesis using independent channel gains can only represent linear relationships between input and output. Assuming similar data representation (e.g. 8, 10, or 12 bits per component), unless $k_L = k_M = k_S = 1.0$, some of the output dynamic range is unused, or some of the input values correspond to values that need to be clipped / clamped, resulting in piecewise linear transfer functions with abrupt changes.

Instead of multipliers, use channel specific look-up tables can capture arbitrary transfer functions defining smooth input-output mappings. This way logarithmic compression or gamma correction as well as color correction can be addressed in one block. In the example FPGA ISP implementation of Figure 4, the tone curve correction block can be used to perform this operation.

5.2.4 Color-Correction Matrix

Considering the differences between the cone sensitivities of the human visual system, and the spectral absorption properties of the pigments used in the image sensor Bayer CFA [40], obviously some spectral components reflected back to the image sensor are captured by different from that of human cone cells. This wavelength dependent cross-contamination between color channels can be compensated by using a 3x3 matrix multiplier. The advantage of this method over the Von-Kries approach is that all 3 color channels are involved in the calibration process, e.g., information from the red and blue channels can be incorporated when adjusting green channel gains. For camera calibration, color responses of calibrated red, green, and blue light sources are captured, and a coordinate transformation aiming to orthogonalize measured red, green, and blue components is established by inverting the matrix of recorded responses.

This solution lends itself well for simultaneous camera calibration and white balance correction using the same CCM (Figure 4), updating matrix coefficients dynamically to match changing illuminants on a frame-by-frame basis.

5.2.5 3D Look-Up Table

Camera calibration assigns an expected value to all possible camera input tristimulus values. A brute-force approach to represent this mapping could be a large look-up-table, containing expected values for all possible input RGB values. This solution raises two practical problems:

- Memory size. For 10 bit input color components mapped to 10 bit output components, the correction table would be 2^{30} words deep, and 30 bits wide, allocating 4 GBytes
- Initialization values. Typically, only tens-, or hundreds of camera input / expected value pairs are established via calibration measurements. The rest of the sparse look-up-table values have to be interpolated. This interpolation task is not trivial, as the heterogeneous component input to output functions are neither monotonous nor smooth.

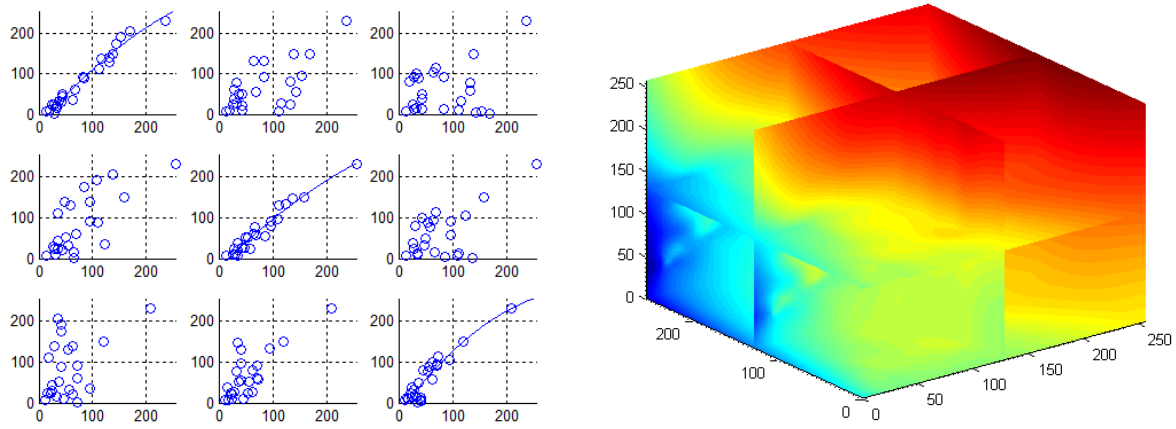


Figure 50. RGB measured vs expected values (left: cartesian plot, right: R channel interpolation)

The left side of Figure 50 presents the measured vs the expected color component values, for R,G and B channels. The right side figure visualizes the 3D mapping function or the Red channel color output based on R,G,B inputs, using a 3D interpolation volume smoothly fitted over the measured input-output data captured.

While this method may not be feasible for FPGA implementation, offline simulation of the results is certainly possible. However, visual evaluation of empirical results using 3D interpolation did not show significant quality improvements in contrast with a properly tuned Color Correction Matrix and Tone Curve Correction based solution.

5.3 Dynamic illuminant colorimetry

Camera calibration focused on accurate color representation with a known illuminant. Dynamic illuminant colorimetry aims to identify the spectral properties of the illuminant using measured pixel values and priors about scene composition.

5.3.1 RGB methods

The two simplest algorithms for illuminant color estimation and corresponding white balance correction operate in the RGB color-space.

The Gray World [56] algorithm is based on the heuristics that the average of all scene colors, the average of red, green, and blue values should result to a neutral, gray color. Consequently, the differences in average red, green and blue values provide information about the illuminant color

and independent color channel gain correction should transform colors such that the resulting color averages are identical.

In a random, dynamic scene, however, there is no guaranty that the scene with different, distinctly colored objects will be balanced. While this solution is simple to implement in conjunction with independent channel specific integrators, normalization [S6], and multipliers, visually objectionable errors can be introduced, or inherent scene colors removed in the presence of large, vivid objects.

The White Point / White Patch [56] algorithm is based on the assumption that the lightest pixels in an image must be associated with the illuminant or are reflections from white objects. The ratios of red, green, and blue channel averages of pixels with maximal brightness provide information about the illuminant color, and correction should transform colors such that the resulting color R,G,B maxima are equal. However, to find the white point, ranking of pixels by luminance values is necessary, and spatio-temporal filtering of the ordered list may be necessary to suppress noise artifacts and aggregate ranked results into a single, white color triplet. More importantly, in most cameras auto-exposure aims to balance the ratio of saturated and underexposed image areas. Therefore, even on well-balanced images a small portion of pixels are saturated. Saturation of individual color channels distorts the perceived (measured) ratios of maximal channel values, leading to incorrect or insufficient channel gains.

5.3.2 LUV, YCC methods

More complex methods take advantage of color-space conversions, where hue can be easily isolated from color saturation and luminance, which reduces 3 dimensional color correction to a 1 dimensional problem.

Color Gamut mapping methods [56] build a two dimensional histogram in the YCC, YUV, L*a*b* or LUV/YUV color-spaces, and fits a convex hull around the base of the histogram. The UV or (Cr, Cb) averages are calculated and used to correct colors, such that the resulting color UV or CbCr histograms are centered on the neutral, or gray point in the YUV, YCC, Luv or Lab space.

The advantage of operating in a color-space with explicit chrominance channels is better color performance. All methods described above may suffer from artifacts due to incorrect exposure settings, or extreme dynamic ranges in scene illumination. Saturated pixels in an image illuminated by a bright light source with inherent hue, such as a candlelit picture with the flame in focus, may result to fully saturated, white pixels present on the image.

5.4 Heuristics for color constancy

5.4.1 Foreground-background separation

Auto-focus (AF) coupled with multi-zone metering allows spatial distinction of pixels in focus around the center and the background around the edges. The assumption that background is more likely to contain the illuminant, and a larger variety of distant objects may improve be closer to the conditions where the gray world or white point hypotheses hold.

5.4.2 Face or object detection

Face, or skin color detection helps cameras identify image content with expected hues. In this case illuminant detection can be limited to pixels with known expected hues. Color correction will take place to move the colors of these pixels closer to the expected colors. In a practical ISP

implementation this is only feasible if downstream processing blocks, such as face detection and image segmentation, extract regions of interest which are provided to the ISP.

5.4.3 Complex methods

Most commercial applications combine multiple methods using a strategy adapting to image contents and photographic environment [57],[58].

5.5 Proposed Algorithm

The proposed algorithm has the following distinct phases:

1. **Camera calibration**. Performed in a laboratory setting with controlled illumination, calibration of image sensors and optical assemblies to establish color correction coefficients and offsets (CCM_k), for all known illuminators (N_i), $k = \{1..N_i\}$.
2. **Illuminator gamut mapping**. Identify color gamuts representative of illumination sources.
3. **White Balancing**. Real-time illuminant estimation and application of color correction with the embedded camera system.

5.5.1 Experimental Setup

For my research the ISP was implemented using the Xilinx Zynq Video and Imaging Kit, based on the XC7Z020 CLG484-1 AP SoC, the Agile Mixed Signal (AMS) camera interface board, and the On-Semi VITA 2000 Color Image Sensor module[14]. The 10-bit global shutter sensor, targeting machine vision applications, can provide an image stream up to 92 frames per second at 1920x1080 (1080p) resolution. The ISP modules (Figure 4) relevant to white balance correction include:

- the Image Statistics HW module, which gathers zone based statistical data on a frame-by-frame basis,
- the CCM HW module, which performs color transformations on a pixel-by-pixel basis,
- the application software, which analyzes statistical information and programs the CCM on a frame-by-frame basis.

To calibrate the colors of the sensor using standard illuminants and uniform, isotropic illuminator, I used an off-the-shelf color-viewing booth (X-Rite Macbeth Judge II). This light-box has 4 standard illuminants with known spectra:

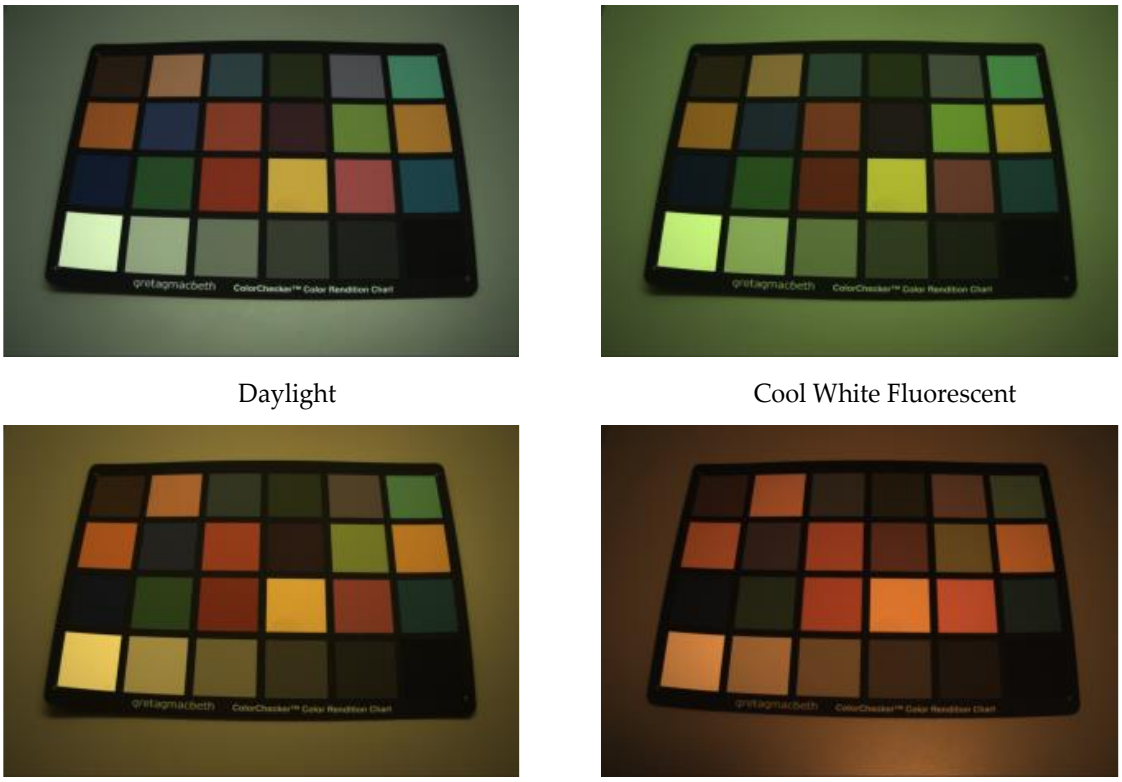
- An illuminant with simulated Daylight spectrum,
- Cool White Fluorescent illuminant,
- Warm Fluorescent illuminant,
- Incandescent illuminant.

As typical for color constancy quality measurements [59], I used the X-Rite ColorChecker 24 Patch Classic target, with color patches of known reflective properties, and expected RGB and sRGB values.

[14] VITA 2000 2.3 MP Global Shutter CMOS Image Sensor https://www.onsemi.com/pdf/datasheet/noiv1sn2000a-d.pdf

5.5.2 Camera calibration

Camera (image sensor with the optical assembly attached) calibration can be performed on each camera unit, or, during mass manufacturing, a batch specific set of correction matrices can be established and used on all units. The X-Rite color target is placed in the lightbox against the gray background of the light box, such that illumination from all light sources is as even as possible. Test images are captured by the sensor with all illuminants, using the ISP, with no color correction (identity matrix loaded to the color-correction matrix). Figure 51 illustrates the captured images without geometric lens distortion and



Daylight | Cool White Fluorescent

Warm Fluorescent | Incandescent

Figure 51: Sensor images with different illuminants before lens correction

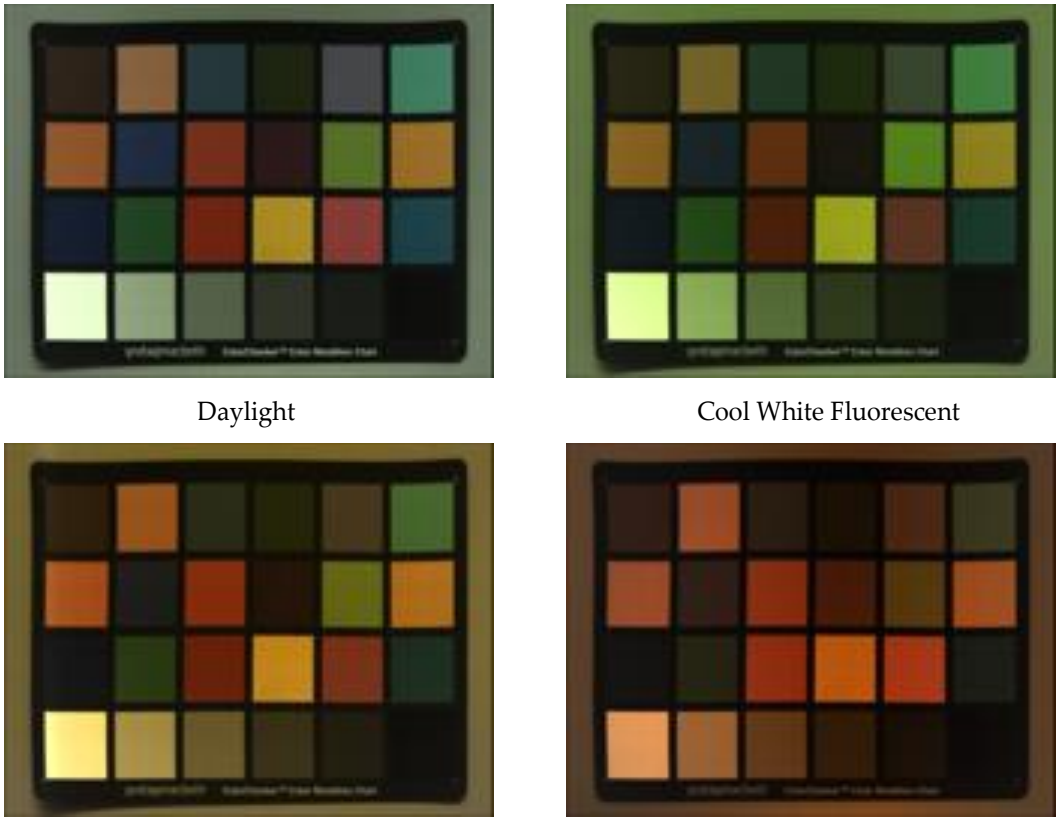
Image parameters for geometric distortion correction (barrel distortion) and lens shading correction (vignetting) can be extracted from this image set, and the images can be post-processed on the workstation performing calibration, or alternatively the distortion parameters can be programmed into the ISP, and a new set of regularized images can be captured for color calibration (Figure 52).

Using the regularized images, the calibration script identifies the patches by edge detection in the HSV color-space. In order to attenuate noise, rectangular regions within each color patch are averaged to represent each color patch with an $\boldsymbol{x}_i^T = \{R_i, G_i, B_i\}$ triplet. The expected RGB values of each color patch $\hat{\boldsymbol{x}}_i^T = \{\hat{R}_i, \hat{G}_i, \hat{B}_i\}$ are available from X-Rite.

The next step of the algorithm is to establish an optimal mapping for each illuminator *k*,

$$\boldsymbol{C}_k = \underset{\boldsymbol{C}}{\operatorname{argmin}} \sum_{i=0}^{n} \boldsymbol{a}_i (\boldsymbol{C}\boldsymbol{x}_i - \hat{\boldsymbol{x}}_i)(\boldsymbol{C}\boldsymbol{x}_i - \hat{\boldsymbol{x}}_i)^T, \tag{59}$$

which minimizes squared error between the transformed x_i values to the expected values $\hat{x}_i$ for all patches $i \in \mathbb{Z} < n$, where n represents the number of color patches. For the X-Rite ColorChecker 24 Patch Classic, n = 24. Coefficient vector $\boldsymbol{a}_i$ allows application specific weighting the importance of color patches. For n = 24 patches, and three color components per patch the generic problem is over-defined. The optimization error can be distributed if certain patches, e.g., gray patches or the ones close to human skin tones, are prioritized over others by setting the corresponding $\boldsymbol{a}_i$ values accordingly.



Daylight | Cool White Fluorescent

Warm Fluorescent | Incandescent

Figure 52: Lens corrected sensor images with different illuminants before color calibration

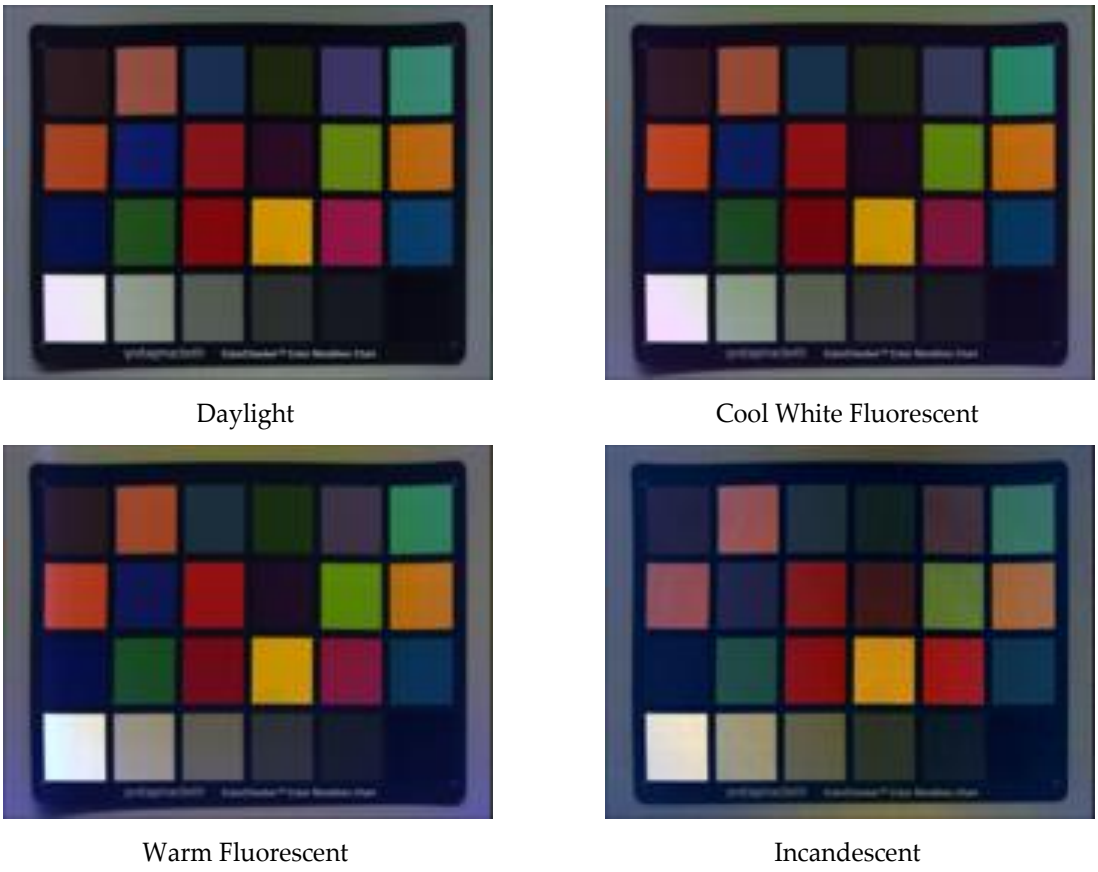
To perform the argmin optimization, the Simulated Annealing (SA) method was used [38], which can minimize a scalar valued error function defined in (59). To accurately model quantization effects inside the CCM, color correction coefficients and offsets were represented in fixed point format, and results were quantized back to the 10 bits/color channel native format of the image sensor data stream:

$$\boldsymbol{x'}_i = \left\lfloor \begin{bmatrix} k_{11} & k_{12} & k_{13} \\ k_{21} & k_{22} & k_{23} \\ k_{31} & k_{32} & k_{33} \end{bmatrix} \boldsymbol{x}_i + \begin{bmatrix} R_{offs} \\ G_{offs} \\ B_{offs} \end{bmatrix} + 0.5 \right\rfloor, \tag{60}$$

where $\lfloor\,\rfloor$ denotes truncation to the nearest integer. The argmin optimization algorithm was performed in 4 different homologous setups:

1. sum of squared error (59), with $\boldsymbol{x}_i$ and $\hat{\boldsymbol{x}}_i$ in the RGB domain,
2. sum of absolute error, in the RGB domain,
3. sum of squared error (59), in the YUV domain, Y (luminance) ignored in optimization,
4. sum of squared error (59), in the L*a*b* domain, L (lightness) ignored in optimization.

No significant difference was observable between the quality of optimization results, except, as expected, for the YUV and L*a*b* domain higher lightness value error variances were traded for lower variances in chrominance/hue. Along with selecting the patches of interest with coefficient vector $\boldsymbol{a}_i$, ISP designers can tune the CCM correction matrices $\boldsymbol{C}_k$ for each illuminant k to results perceived optimal for the application area.



Daylight | Cool White Fluorescent

Warm Fluorescent | Incandescent

Figure 53: Color Calibrated, lens corrected sensor images with different illuminants

Figure 53 illustrates colors calibrated with the illuminant specific correction matrices $\boldsymbol{C}_k$.

5.5.3 Illumination estimation by gamut mapping

The next step of the algorithm is to identify color gamuts characteristic to each illuminator to be used later for illumination estimation. To perform this step, for all illuminants $k = \{1,2,3,4\}$, I recorded the two-dimensional chrominance histograms using pixel values from the gray colored patches of Figure 52. As opposed to the more perceptually uniform CIELUV or CIELAB spaces, YUV or YCbCr can be used as these color spaces also capture the area around the white/gray/D65 point, and are significantly simpler to compute in HW.

5.5.4 FGPA Hardware accelerator

In order to support illuminator estimation, the image statistics (section 2.4) module of the ISP contains specific hardware components to construct a two-dimensional U-V histogram (Figure 54).

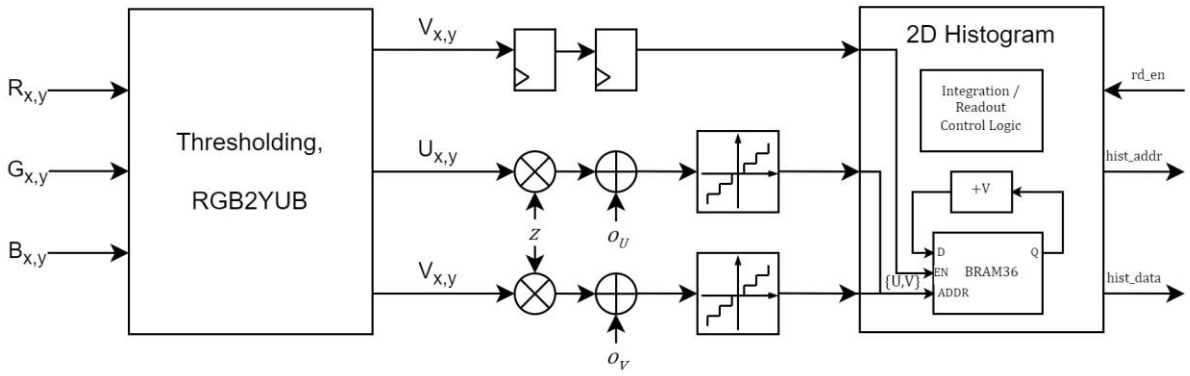


Figure 54: Illuminator estimation specific hardware components in the Image Statistics ISP Module

YUV (YCbCr, L*u*v* or L*a*b*) histograms of the scenes by binning pixel values by chrominance (UV, or CbCr, or a*b* components). In the physical implementation (Figure 54) the RGB pixel values are first transformed to the YUV domain by a module introduced in section 2.7. For pixels with saturated R, G, or B channels, the output luminance (Y) value is set to zero. Chrominance (U and V) values are transformed by

$$u_{x,y} = [zU_{x,y} + o_U], \\ v_{x,y} = [zV_{x,y} + o_V], \tag{61}$$

which for ($z > 1$) effectively zooms into a square area of the chrominance plane (Figure 55). The 2D histogram is stored in a BRAM36 in 1024x36 bit configuration. Read and write address to the BRAM are formulated by concatenating the transformed ($u_{x,y}$, $v_{x,y}$) chrominance values quantized to 5 bits, resulting to a 10 bit address. For $u_{x,y}$ and $v_{x,y}$values outside the 0-31 range, the corresponding Y value is set to 0, effectively excluding the pixel from the 2D histogram integration. For included pixels, instead of simply counting the number of pixels with chrominance values within bin boundaries, bin values are incremented by the luminance value. This results to a luminance weighed chrominance histogram $H(u, v)$, which de-prioritizes dark pixels, where small

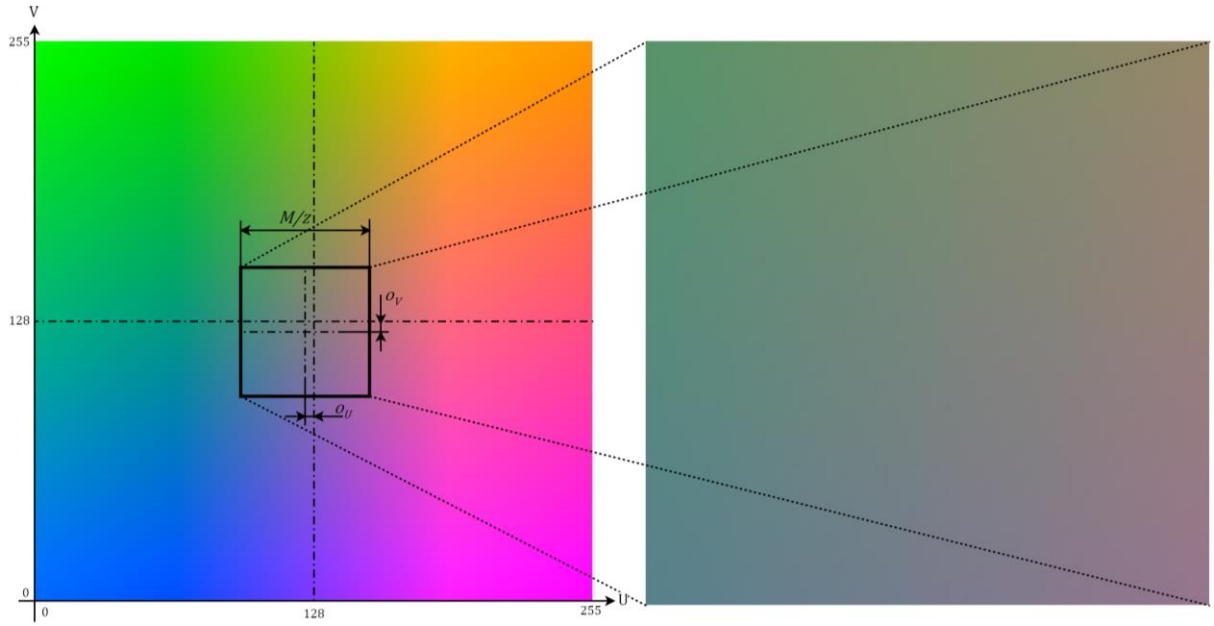


Figure 55: U,V chrominance plane (left), zooming on the center region (right)

differences in R,G,B values result to more noise in the chrominance domain. For a perfectly flat image with a single color, all pixels contribute their quantized luminance (Y) value. To avoid overflows, each bin, or BRAM memory location, should be able to accumulate 24, or 32 bit wide histogram data.

It is worth noting that typical latency of BRAMs is 2 CLK cycles (using the internal output registers of the primitive), and adder stage cumulating the +V values requires at least one additional pipeline stage, considering the width of the adder. In order to accommodate latency of k clock cycles, a predictive, k stage deep pipeline is performing additions in the histogram module. Without the predictive pipeline, if 3 pixels with the same quantized $u_{x,y}$ and $v_{x,y}$ values, and corresponding Y_1, $Y_2$and Y_3 are input to the sub-module, the BRAM output $Q(\{u_{x,y}, v_{x,y}\})$ read value would not reflect the increments by Y_1 before incrementing Q by Y_2 and then Y_3, resulting to an erroneous cumulative value of only $Q + Y_3$. In contrast, the predictive pipeline evaluates the next k pixels in parallel. If corresponding addresses (based on $u_{x,y}$ and $v_{x,y}$ values) are identical, only a single read-modify-write cycle is executed, with memory contents updated by $Y_1 + Y_2 + Y_3$. If the addresses are all different, the operation is safe, and 3 pipelined read-modify-write operations are performed in sequence. In my implementation k=4, and the predictive pipeline handles all five possible address relations: all identical, 3 identical but 1 different, 2 pairs of identical, 2 identical and 2 different, or all different addresses.

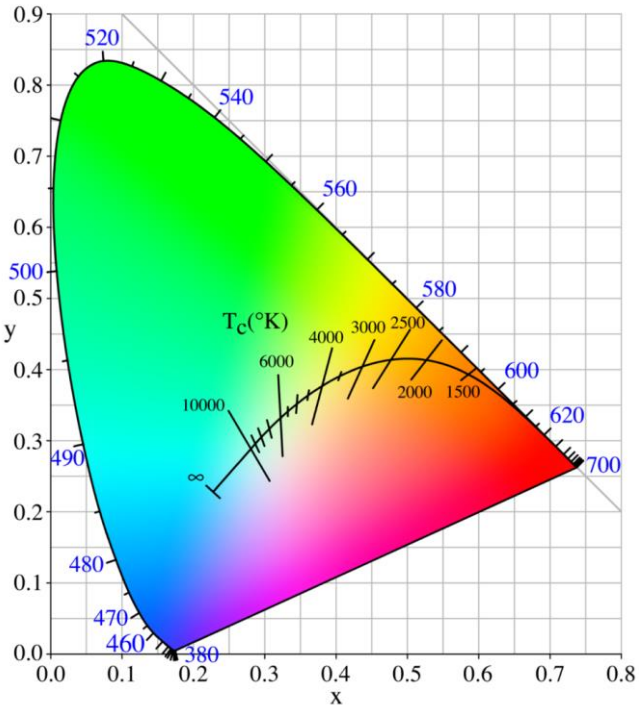


Figure 56: x and y coordinates of illuminants with different temperatures in CIE color-space

The center portion of the Y,U plane contains neutral, gray colors. Similar to the CIE color-spaces, radiant black body illuminants with different color temperatures (Figure 56) are located in the central, neutral region. The periphery contains vivid, vibrant colors, which are more representative of the innate spectral absorption and reflection properties of scene objects. Therefore, for illuminant identification, focusing on the central, neutral region of the color-space improves the likelihood of correct illuminant identification as this region is more likely to reflect the perceived color of the illuminant.

In the final phase of color calibration, a set of reference images were recorded, with four different illuminators. Some of the images were captured in the lightbox, the same scene imaged with each illumination source. Some of the images were captured with the central region of the UV histogram with different scene contents e.g., the top-left image of Figure 57 captured with incandescent lighting. The rest of the reference images were captured in settings typical for the illuminators (natural sunlight, warm or cool fluorescent lighting), making sure that neutral, or white color objects were present in the scenes.

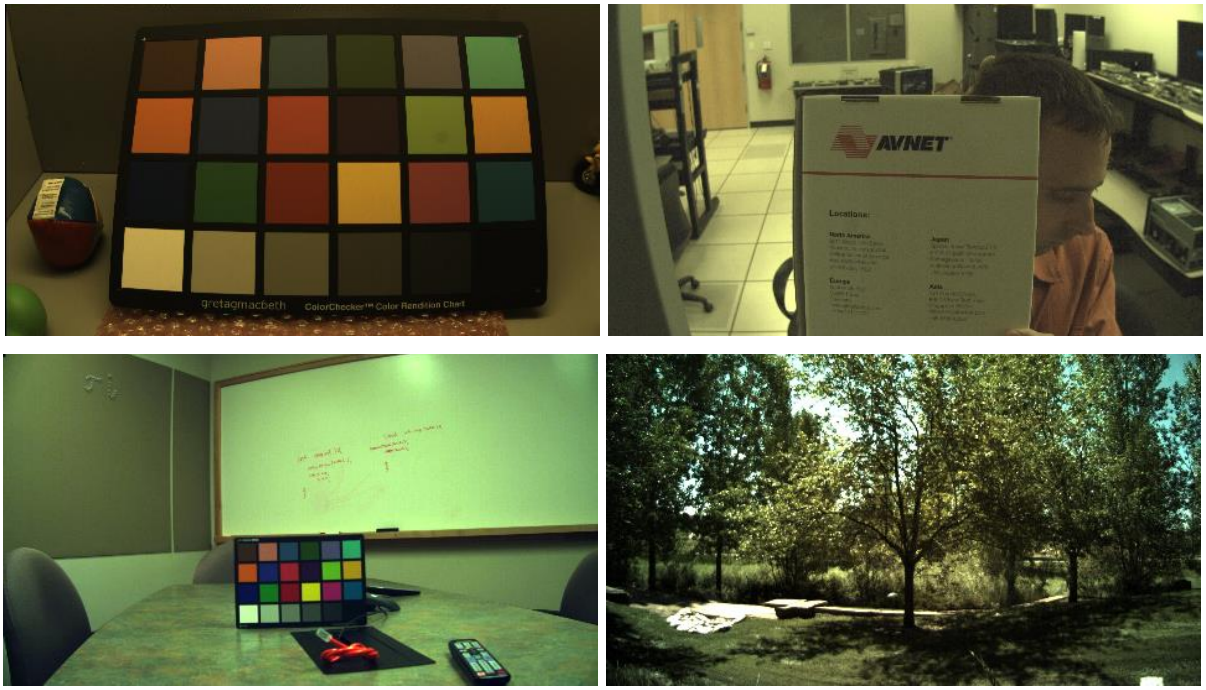


Figure 57: Reference images with four different illuminators
top-left: Incandescent, top-right: warm fluorescent, bottom left: cool fluorescent, bottom-right: daylight

The reference images were captured with the target camera system, without any color, geometric, or lens shading correction, matching how the target systems statistical module receives input images.

The UV chrominance histograms of reference images pertinent to each illuminator were averaged, smoothed with a 2D Gaussian kernel of $\sigma = 1.4$, then normalized such that $\sigma^2(H_k(u,v)) = 1$. Focusing on just the center of the UV plane, with $z = 4$, $o_U = 4$, $o_V = 12$, the center regions of the UV chrominance histograms show loci characteristic to different illuminators (Figure 58). Each locus carries information on how the illuminator was reflected from a variety of materials, by the R,G, and B channels of the image sensors, also a function of the spectral filtering properties of the color-filters used by the sensor manufacturer.

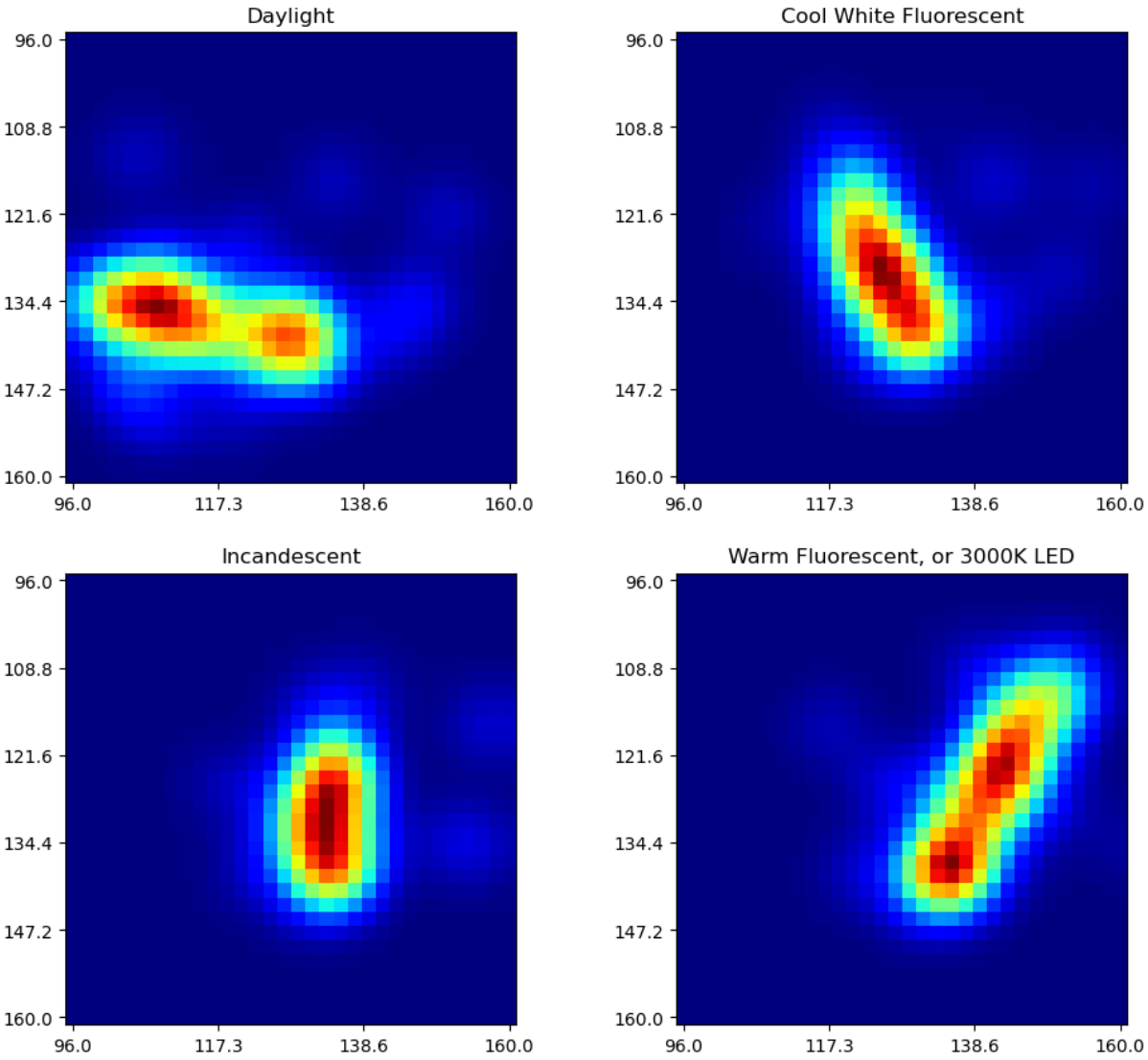


Figure 58: averaged 2D UV histograms, $H_k(u, v)$, with 4 different illuminants

5.5.5 White Balance Correction Algorithm

The white balancing algorithm, implemented in software running on an embedded processor, performs the following operations on a frame-by-frame basis:

1. Using statistical information, estimate the illuminant probabilities ($p_{k,f}$)
2. Adaptive filtering of illuminant probabilities to establish weights ($w_{k,f}$),
3. Combine $\boldsymbol{C}_k$ values according to $w_{k,f}$, into a scene specific correction matrix $\boldsymbol{C}_f$, and program the Color-Correction Matrix module with $\boldsymbol{C}_f$

One advantage of this method is that a linear combination of calibration $\boldsymbol{C}_k$ values will implicitly bound color artifacts in the presence of significant illuminant estimation errors. For example, in case of underwater photography, where a strong blue tinge is present, a simple approach, such as Gray World would compensate to remove all blue, severely distorting innate scene colors.

The masked two-dimensional histogram values $H_k(u, v)$, as well as color-correction matrix $\boldsymbol{C}_k$ coefficients corresponding to illuminant k are compiled to the FW of the embedded processor as coefficient files established during static calibration.

1. During real-time operation, the statistical module of the ISP collects two-dimensional, luminance weighed chrominance histograms of the scene. The statistical module collecting 2D histograms is configured identical to the parameterization used during reference image collection ($z = 4$, $o_U = 4$, $o_V = 12$), focusing on the central region of the UV histogram. The resulting 32x32 histogram is low pass filtered, and normalized by the firmware, resulting to signature $H_f(u, v)$ specific to frame *f*. The first step of the white balancing algorithm is classification of this signature, to one of the *k* illuminant classes. Sum of squared differences is calculated between the four stored histograms and the measured two-dimensional histogram:

$$D_{k,f} = \sum_{u=0}^{31} \sum_{v=0}^{31} \Big(H_k(u, v) - H_f(u, v) \Big)^2, \tag{62}$$

where the resulting D_k is proportional to how *dissimilar* the signature specific to frame f is to the pre-calculated references pertaining to known illuminants. Based on the measured histogram differences D_k, normalized similarity measures are calculated using:

$$p_{k,f} = \frac{1}{s(D_{k,f} + \varepsilon)} \ , \ s = \sum_{i=1}^{4} \frac{1}{(D_{i,f} + \varepsilon)} \tag{63}$$

where ε is a small constant to avoid division by zero. Normalized similarity measures $p_k[f]$ can be interpreted as probabilities representing the likelihood that the scene of the last image frame *f* was illuminated with illuminant *k*.

2. To avoid abrupt frame-by-frame tone changes, objectionable to viewers, an adaptive, temporal low-pass filter is applied to the illuminant probabilities. During scene changes, used a simple IIR filter, implementing

$$w_{k,f} = cp_{k,f} + (1 - c)w_{k,f-1} \tag{64}$$

where $0 < c < 1$ controls the impulse response of the IIR filter. Lower *c* values result to smoother transitions. The larger the value of *c*, the quicker the filter responds to changes in lighting conditions. Values of *c* can be controlled dynamically by the ISP based on changes in global luminance (Y) and chrominance (U,V) of the entire image frame, which are calculated by the image statistics module. Large, sudden changes in global values typically indicate either a scene change, or a change in illumination such as turning on lights in a room to complement fading natural sunlight. Explicit treatment of chrominance allows detection of large, colorful objects entering, or exiting a scene, even if the concurrent autoexposure algorithm minimizes luminance changes.

$$c = c_m + (1 - c_m - c_M) \operatorname{erf}\left(\frac{|\Delta Y|}{Y_{TH}} + \frac{|\Delta U| + |\Delta V|}{C_{TH}} \right), \tag{65}$$

where c_m and c_M are lower and upper bounds for the convergence constant driving the IIR filter (64), ΔY, ΔU, ΔV are the frame-to-frame differences in global (average) YUV values, Y_{TH} is threshold value controlling sensitivity to luminance changes, and C_{TH} is a threshold value controlling sensitivity to chrominance changes.

Based on the illuminator specific weights $w_{k,f}$ computed (64), the color correction coefficients best suited to a mixed illuminant can be found as the linear combination of the pre-calculated color-correction coefficients and offsets ($\boldsymbol{C}_k$):

$$\boldsymbol{C}_f = \sum_{k=1}^{4} w_{k,f} \boldsymbol{C}_k. \tag{66}$$

Finally, FW programs the CCM of the ISP (Figure 4) with the aggregated, frame specific color correction coefficients ($\boldsymbol{C}_f$).

As an additional feature, suitable for applications where the resulting video stream is intended for human consumption, the coefficients of $\boldsymbol{C}_f$ pertinent to the chrominance channel outputs can be multiplied with a constant. Setting this constant to a value less than 1.0 reduces tonality, with zero resulting to grayscale images, while setting the constant to values above 1.0 increases color brightness.

5.6 Results

The proposed algorithm was tested indoors and outdoors, at two illumination levels ~1000 Lux and ~65000 Lux. Real-time white-balance implementation results, even from scenes illuminated by both natural daylight and fluorescent light (Figure 59) demonstrated significant improvement in perceived image quality and color representation.

Figure 59: Conference room image with dual illumination with AWB disabled (left), enabled (right)

Algorithms for illuminant estimation are often evaluated by quantifying the estimation error in terms of an angular error [52]. Using a benchmark dataset with a gray reference image (masked during scene illuminant estimation), the color of the light source is estimated for every image of the image set and compared to the ground truth established from the gray reference object. The result is expressed then as an angular error:

$$d_{angle}(\boldsymbol{v}_e, \boldsymbol{v}_g) = \cos^{-1}\left(\frac{\boldsymbol{v}_e \cdot \boldsymbol{v}_g}{\|\boldsymbol{v}_e\| \cdot \|\boldsymbol{v}_g\|}\right), \tag{67}$$

where $\boldsymbol{v}_e \cdot \boldsymbol{v}_g$ is the dot product of the estimated illuminant $\boldsymbol{v}_e$, and the ground truth $\boldsymbol{v}_g$ and$\| \| \cdot \| \|$ denotes the Euclidean norm of a vector. Unfortunately, to evaluate end-to-end results of the proposed approach against the benchmark sets, the camera calibration coefficients for known illuminants ($\boldsymbol{C}_k$) is necessary, which is not available at this time.

5.7 Chapter Summary

At the beginning of this chapter, I reviewed the background of colorimetry and image sensor color formation, as well as existing basic approaches to illuminant detection and automatic white balance correction. The purpose of my research to this application area of the ISP was to provide methodology and FPGA circuitry that allows the FPGA user community to implement an ISP with white-balance functionality [S7], minimizing FW complexity and HW footprint, while maximizing quality of results.

I provided a method for color calibration, an algorithm suitable for low-power embedded processors and its efficient implementation on FPGAs for robust white balancing of image streams. The proposed algorithm leverages a hardware accelerator module, implemented in FPGA logic fabric, which calculates focused, luminance weighted 2D Chrominance histograms. Illuminant estimation is performed by real-time classification of 2D chrominance histograms of image frames, comparing against illumination specific signature patterns established offline. Classifier output probabilities are filtered using temporal, adaptive IIR filters, which in turn control the mixing of color-calibration matrixes specific to illuminants. Besides efficient use of FPGA resources and low embedded processor load, one advantage of the proposed method is that correction errors due to illuminant misclassifications are bound, preventing gross distortion of image colors.

6 Non-linear filter optimizations

6.1 Median and Rank order Filtering

Rank order filtering is a non-linear filtering technique, which selects an element from an ordered list of samples. Median filters are examples of rank order filters where the element in the middle of the ordered list is selected. Compared to Finite-, or infinite Impulse Response (FIR, IIR) linear filters, rank order filters can effectively remove impulse-like noises while preserving high frequency content of the original stream. For image and video processing, rank order filters can be extended to two-dimensional (2D) filters, which remove "salt-and-pepper" noise while preserving edges in the original image. This can be useful for removing transmission (bit-flip) artifacts, especially as pre-processing for edge detection. The median of a set of samples is the sample in the middle of the rank ordered list of the samples:

$$y = \mathrm{med}(x_0, x_1 \dots x_{2N}) = \hat{x}_N, \tag{68}$$

where $\hat{x}_k, k \in \{0, 2N\}, N \in \mathbf{Z}$, is the magnitude (rank) ordered list of input samples, when the filter size (2*N*+1) is odd. For even sized kernels, median is defined as:

$$y = \mathrm{med}(x_0, x_1 \dots x_{2N-1}) = \frac{\hat{x}_N + \hat{x}_{N-1}}{2}, \tag{69}$$

To perform median filtering on a set of *n* input samples, resulting to a set of *n* output samples, the input set is typically padded on both ends with *N* samples by repeating the first and last samples. To perform the non-recursive median filtering operation, the sampling window is shifted along the sample set, with the least recent sample removed from the kernel, and a new sample entering the kernel. For a kernel size 2**N*+1, the actual input sample values in the kernel are x_{k-N}, x_{k-N+1},.., x_{k-1},x_k, x_{k+1},.., x_{k+N-1}, x_{k+N}. For any $i < 0, x_i = x_0$ and any $i \geq n, x_i = x_{n-1}$. The filter output is defined as:

$$y_k = \mathrm{med}(x_{k-N}, .., x_{k+N}), \tag{70}$$

In case of recursive median filters half of the median sorting kernel contains *N* recent output samples, and *N*+1 input samples, so the non-ranked kernel contents are: y_{k-N}, y_{k-N+1},.., y_{k-1}, x_k, x_{k+1},.., x_{k+N-1}, x_{k+N}. As opposed to IIR linear filters, in response to a step function the output of the recursive median filter converges to a stable signal in finite number of steps [60]. These stable signals extracted from the input signal are called the roots. The advantage of recursive median filters is the ability of abstracting the roots from input signals in one run. As shown below the hardware realization of the standard and recursive filters are almost identical.

Outputting a sample other than the center of the rank ordered kernel can be useful if the input signal is contaminated with non-zero mean noise. In case of image filtering, rank order filtering may change the brightness of the image. Hardware implementation of rank order filters is almost identical to median filters [S8], with median filter being a sub-class of rank order filters with constant indexing of the rank ordered list.

Two-dimensional median-, and rank order filters are commonly used in image processing, where occasional transmission noise spikes, such as Inter-Symbol Interference (ISI) affecting the MSB of samples should be removed from an image, while sharp edges should be retained.

The complexity of the ordering operation is strongly affected by the size of the kernel size N, with complexity scaling with $\theta(N\log(N))$. General purpose DSP processors exhibit sharply decreasing performance with increasing N due to inefficient pipelining and prediction misses caused by data dependent jumps in sorting algorithms. Due to predominance of comparison operators and subsequent branch misprediction median filtering is not efficiently carried out by GP processors or ALUs either.

6.1.1 Related Work

Bit serial approaches [61],[62] provide the most compact FPGA / ASIC solutions, but do not lend themselves well for high-dynamic range (10/12/14 bit per color channel) CMOS sensor processing as filtering performance is proportional to the precision of the input data. However, the processing rate typically does not depend on the number of new samples the filter can handle in a single clock cycle.

A large number of word parallel architectures implement sorting networks, which may employ bubble sorting [63], odd-even merge sorting [64], or other architectures optimized for resource efficacy [65]. Insert-delete architectures store the samples ordered, as proposed in the 1D rank filter core introduced in section 6.1.1, the least recent sample is discarded, and the most recent input is inserted into the magnitude sorting structure at the appropriate location. This architecture does not natively support inserting multiple samples per clock cycle. A cumulative histogram-based architecture is proposed in [66]. This architecture exploits the relatively large on-chip memories in modern FPGAs, however synthesis results showed significant speed burdens which renders this class of filters less applicable for filtering high sample rate videos with large kernels.

6.1.2 Efficient FPGA Implementation of a One dimensional Rank Filter

FPGAs can implement real-time median or rank order filters in high speed video processing applications efficiently with dedicated HW that scales with N, processing each input sample in a single clock cycle. Because arithmetic operations and data transfers can be constrained to local, neighbor-to-neighbor operations, FPGA or ASIC implementation is compact, efficient, with easily predictable performance.

My proposed implementation (Figure 60) for a 1D filter with T=2N+1 taps is a systolic array of T simple Processing Elements (PE) [S9],[S10], [S11]. Each PE contains data registers d_k to hold an input sample value and its corresponding index i_k, with the most recent, and least recent samples having indexes 0, and T-1 respectively. The pipelined architecture is similar to a doubly linked list

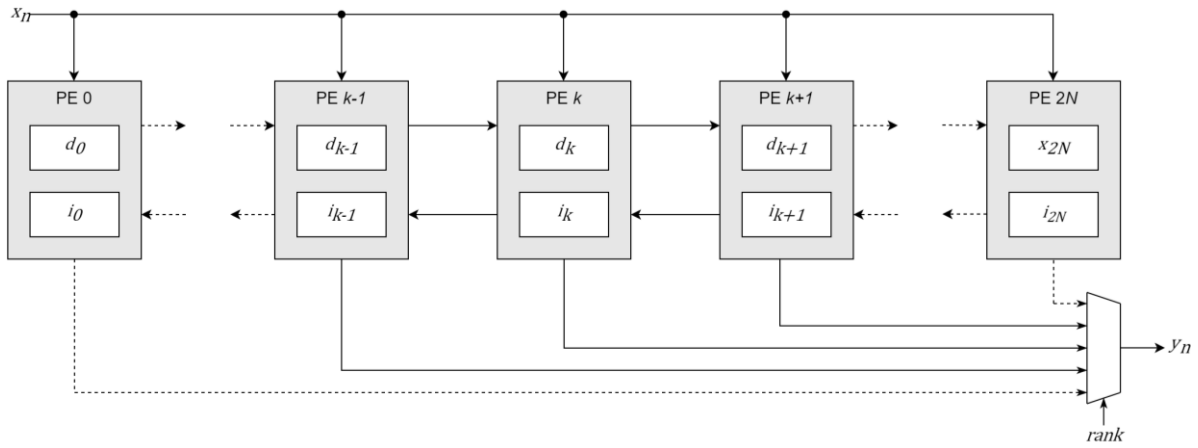


Figure 60: Block Diagram of a 1D rank-ordering architecture

implemented in HW. During operation the linked PEs maintained an ordered list of the input samples. For non-recursive mode of operation, for each clock cycle, the least recent sample is discarded, and the new input sample (x_n) is inserted to the appropriate PE. The index of each sample in the list is incremented, while the index pertinent to the new sample is initialized with zero. The filter output is derived directly from one of the PEs, selected by input signal *rank*. For median filtering *rank* = *N*, the output sample (x_n) is selected from PE in the center of the linked PE pipeline.

Each PE has access to the new input sample (x_n), and the sample values it stores (d_k) and performs the comparison $g_k = (d_k < x_n)$. Each PE shares the one bit comparison result g_k with its neighbors. Using a control table based on the 3 bits $\{g_{k-1}, g_k, g_{k+1}\}$ nodes can uniquely identify where the new sample needs to be inserted.

Besides the comparison output bit, all PEs receive from, and provide to their respective neighbors a bit indicating whether the insertion point is to the left or to the right. Once the insertion point is identified, the one bit location asynchronously ripples through the list in both directions.

Similarly, the PE holding the least recent sample can identify itself as the one discarding a sample, and this information is broadcasted left and right to both neighbors, in turn to the list of PEs in both directions.

Let $0 \leq k \leq T-1$ denote the position of the PE storing the least recent sample, with index *T-1*. Let $0 \leq l \leq T-1$ denote the position of the PE where the new sample needs to be inserted. Four cases are possible:

- $k = l$. The new sample just replaces the least recent one, no other samples are moved.
- $k > l$. PEs *l+1* to *k* load values from their left neighbors, PE l loads the new sample.
- $k < l$. PEs *k* to *l+1* load values from their right neighbors, PE l loads the new sample.
- The insertion point can't be determined. In case all PEs contain the same value, and $x_n = x_k$, the filter output remains the same and no operation takes place.

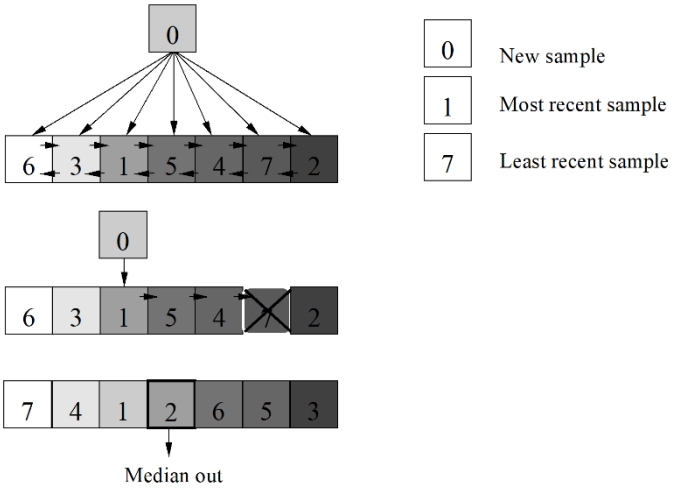


Figure 61: Inserting a new sample to the magnitude ordered list

Figure 61 illustrates this concept for T=7 (*N*=3), $k > l$, with the shade of the rectangles representing the magnitude of the stored sample (d_k) and the number inside each rectangle reflecting the index (i_k) corresponding to d_k. The new sample, with index = 0 is first compared to all samples in the ordered list in parallel. Based on the one-bit signals from neighbors conveying information on the comparisons each PE can determine whether it needs to perform deletion and the insertion, or if the location of the deletion and the insertion is left or right. In the example above, based on sample

index values, PE#5 will discard the least recent sample, and based on comparison results PE#2 will load the new sample. Based on this information each PE can determine whether shifting left, shifting right, or no loading is necessary for each clock cycle, so PE#2 loads the new sample, PEs #3,#4,#5 will load samples from their left side neighbors, PE#2, PE#3, PE#4. At the end of the cycle all indexes are incremented, and for median operation, the output of the center PE, PE#3, is selected. Using synchronous sequential logic, all operations take place in a single clock cycle.

This architecture can also implement recursive median filters in two clock cycles to process an input sample. For recursive operation, bit 0 of the index (i_k) is designated to indicate whether the corresponding sample (x_k) is new input (0), or recurrent output (1). In phase 1, the least recent input sample is discarded, and the current input sample is inserted into the list. A new output sample is identified. In Phase 2 the least recent output sample is discarded, and the recently identified output sample is inserted into the list.

A limitation of the proposed 1D architecture is that daisy-chained control signals form a deep combinatorial network with many logic and routing stages, which is difficult to pipeline. As the number of PEs increases, the critical path through the control signals severely limits the operating frequency and throughput of the filter.

6.2 2D Rank Order Filtering

2D filtering takes place on the contents of a rectangular window, which slides across the image. Every time the h pixel high by w pixel wide window is moved by one pixel, h obsolete pixels are discarded, and h new pixels are inserted to the kernel. The filter has to sort hw pixels to generate an output pixel.

A 2D median filter can be evaluated using a 1D median filter with a $h \cdot w$ size kernel [S12], [S13]. When the 2D kernel is shifted by one pixel, h new samples (pixels) are entering the filter, and the h least recent elements are discarded respectively. Only after h processing steps can the output selected from the ordered list. If pixel clock frequencies are prohibitively high to run the 1D filter at a multiple of the pixel clock frequency, a more parallel approach can be used. Using multiple, parallel instances of certain key components of the filter, the filter may accept h number of new input samples every clock cycle, allowing operation at the pixel frequency. If the pixel rates of image sensor, video stream clock is relatively low compared to the operating frequency achievable in novel FPGAs, a fully parallel implementation may be sub-optimal due to inefficient resource utilization.

A solution capable of ingesting a configurable number of input pixels (S) spanning between fully parallel ($S = h$) and word serial ($S = 1$) allows tuning the filter core to the maximum clock frequency allowed by the target chip while minimizing resource counts. From the vertical size of the filter window (h), the sampling (pixel) frequency of the input (f_s), and the number of input samples (S) the filter can process in parallel the necessary operating frequency of the filter core (f_{max}) can be determined:

$$f_{max} = f_s \frac{h}{N_i} \tag{71}$$

The next section describes a word serial approach, similar to the architecture introduced in section 6.1.1, which can process one input sample in every clock cycle. Section 6.2.3 presents an extension which allows processing a configurable number of new input samples between in one clock cycle.

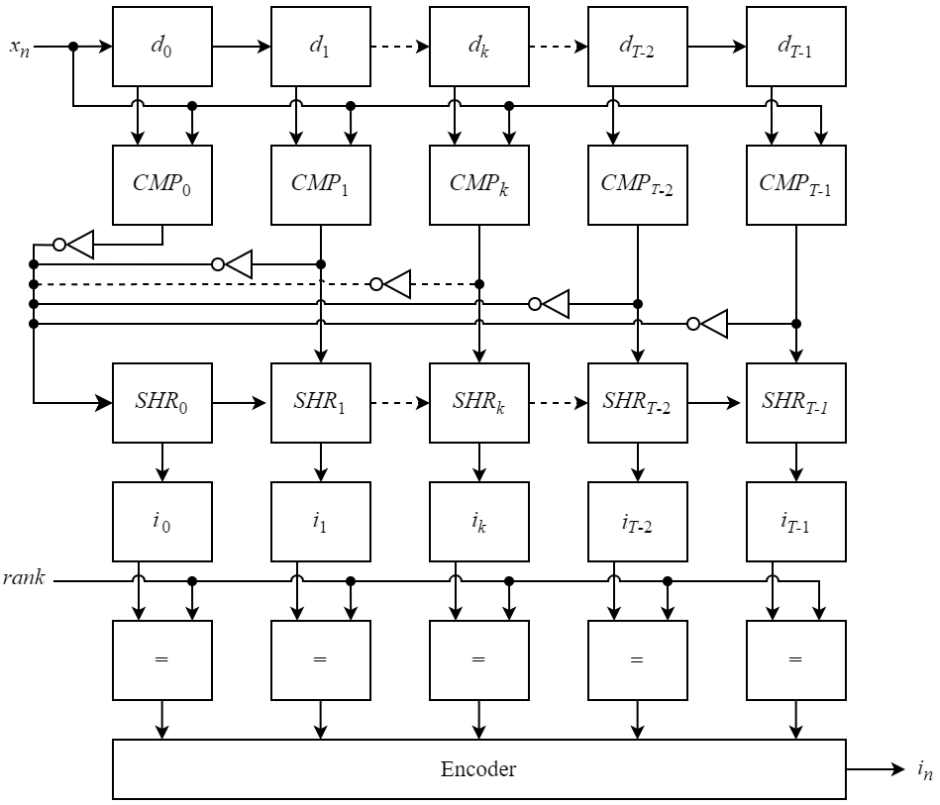


Figure 62: Word-serial filter core architecture

6.2.1 Word-serial rank filter core

To improve timing of the systolic 1D architecture introduced above, samples are stored in the order they were received. The word-serial filter core architecture is presented on Figure 62.

A new sample (x_n) is entered into a set of data registers holding samples in the kernel ($d_0 - d_{T-1}$). While the samples are stored unordered, index i_k pertinent to sample d_k in the filter core indicates the index of d in the magnitude ordered list of ($d_0 - d_{T-1}$). As in case of the systolic 1D architecture, x_n is compared with all samples already in the kernel. The 1 bit result of each comparison (CMP_k) is entered into a T-1 bit wide register SHR_k. For all new samples x_n, SHR_0 captures inverted bits of all comparisons. The rest of the registers ($SHR_k, k > 0$) load bits 0 to T-2 of output of SHR_{k-1}, so shift register contents stay aligned with data registers, shift contents left by 1 bit, and add CMP_k. E.g., if the new sample is greater than all samples in the kernel, then all bits of $SHR_0$are initialized by 1s, while all $SHR_k, k > 0$ latch in a zero bit.

$$SHR_k = \lfloor (SHR_{k-1} \ll 1) + CMP_k \rfloor_T, \tag{72}$$

where $\lfloor \rfloor_T$ denotes truncation to T bits.

Index i_k is derived from SHR_k by summing the bits of SHR_k

$$i_k = \sum_{b=0}^{T-1} SHR_k[b], \tag{73}$$

where $SHR_k[b]$ denotes bit b of SHR_k. Index i_k, pertinent to sample d_k, indicates the number of samples in the kernel smaller than d_k.

Finally, indexes are compared with input integer $rank$, resulting to a T bits wide one-hot encoded bit vector, which can be encoded to a binary index i_n. This output index can be used as an address to select a sample from the list of input samples stored in a shift register or circular buffer.

6.2.2 Color image processing

When processing a color video stream, filtering the color channels independently may introduce color artifacts, generate new color values which were not present in the input image. Also, filtering components independently increases computational requirements, therefore the preferred solution is to use a single indexable aggregate value per pixel. For most practical applications this value is brightness, or luminosity (Y). If the input format already has a luminosity component, as for YUV or YCbCr, it can be used directly, otherwise the indexable value can generated within the 2D filter.

The 2D color processing architecture illustrated on Figure 63 implements a 2D kernel with h=3 pixel rows, and the word-parallel filter core processing $N_i = 3$ samples per clock cycles. The line buffers store all color channels (R,G,B or Y,Cb,Cr) per pixel. The RGB Pixel Buffer holds all T pixels currently processed by the core, and is outputting y_n a color pixel addressed by the filter core output index, i_n.

If required, the RGB to Y modules compute magnitude values (such as luminance) for filtering. For RGB input, luminance, a typical magnitude value can be calculated by

$$x_n = 0.299\,R_n + 0.587\,G_n + 0.114\,B_n \approx 0.25\,R_n + 0.625\,G_n + 0.125\,B_n. \tag{74}$$

The first method requires 3 DSP48 tiles per filter core input, while the approximation only requires 4 fabric based adders. The RGB line buffers are implemented using BRAM based FIFO primitives.

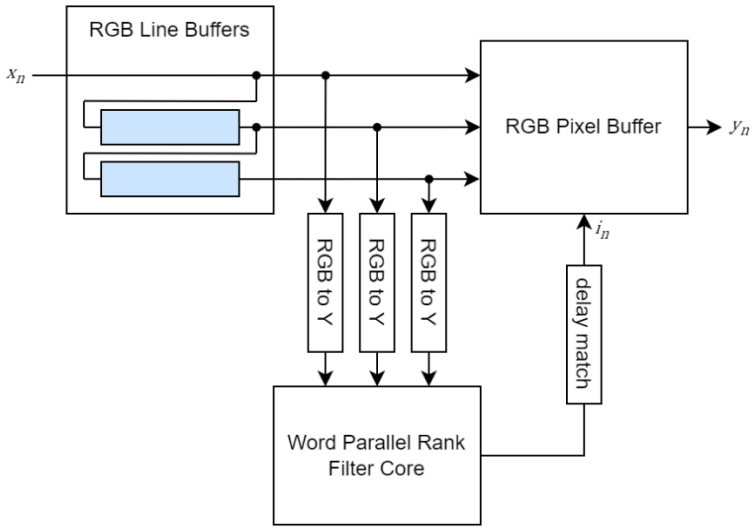


Figure 63: Word parallel rank-filter architecture for RGB color image processing

If the FPGA design is not constrained by BRAM availability, a single RGB to Y module can be used directly on the input RGB components, and all 4 components can be stored in the line buffers, trading DSP48s for extra BRAM storage.

The complexity and latency of the RGB to Y module does not change the filter architecture, but the additional latency of the module need to be considered when synchronizing i_n with the buffered RGB pixels.

6.2.3 Word-parallel extension of the rank filter core

The word-serial filter core architecture can be easily extended to ingest multiple input samples in one clock cycle. Instead shifting by one, data registers ($x_0 \ldots x_{T-1}$), and comparator results registers ($SHR_0 \ldots SHR_{T-1}$) can shift by S positions to process S samples per clock cycles (word parallel architecture). In order to compare each new sample ($x_n \ldots x_{n+S-1}$) with all previous samples in the kernel, as well as the new samples with each other, the number of comparators necessary are:

$$S(T-S)+\frac{1}{2}S(S-1) = S(T-0.5S-0.5). \tag{75}$$

Similarly, recursive rank filters can be implemented by feeding back the filter output along with the filter input, as two new samples, and configuring the filter core for $S = 2$. One way to easily visualize extension of the word-serial architecture to word-parallel is replicate all functional modules S times as processing layers (Figure 64). Processing columns in each layer pass samples along the delay pipeline one sample per clock cycle. Shift register (SHR) inputs in each layer accept $S(T-S)$ inputs, combining information from all layers.

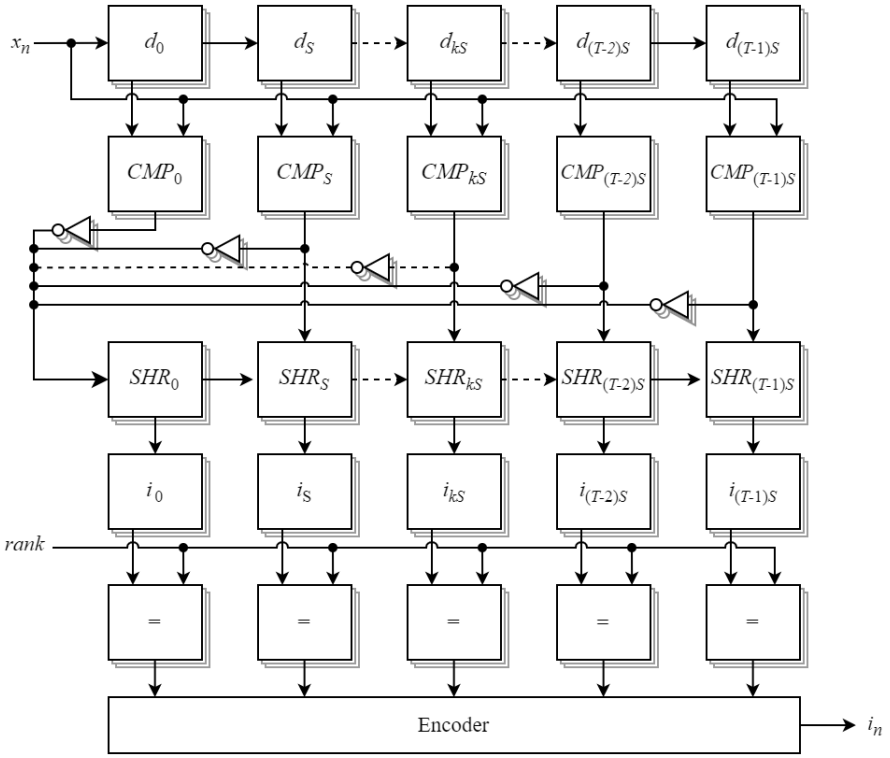


Figure 64: Word-parallel filter core architecture

For simple two-dimensional filtering ($S = h$ and $T = w$), h new samples are provided from the line buffer in every clock cycle. However, considering the high operating frequency (350+ MHz for Ultrascale devices) of the filter core in comparison to typical streaming video pixel clock frequencies (f_s), logic resources may be conserved by running the parallel architecture at a multiple of f_s. As shown in equation (71), to filter streams with pixel frequency f_s using a 2D kernel with vertical size h, processing one pixel per clock cycle, a minimum operating frequency of $f_{min} = hf_s$ is necessary. Conversely, if the target FPGA can implement the word parallel filter core at $f_{max}(S)$,

$$S \geq \left\lceil \frac{hf_s}{f_{max}} \right\rceil \tag{76}$$

samples need to be processed per clock cycle, where $\lceil\ \rceil$ denotes rounding up to the nearest integer.

6.3 Virtual and non-rectangular kernels

To support vertical kernel sizes (h) not an integer multiple of S, the filter can be enhanced to process a dynamically changing number of new samples, by inserting multiplexers into the appropriate data paths, in front of the data (d_k) and comparator result shift registers (SHR_k). Two-to-one multiplexers are always sufficient, as there are only two different scenarios while h input samples are entered to the filter with S inputs in subsequent clock cycles. Either all S filter inputs are valid, or there are only h mod S valid samples available. Even though the size of multiplexers is limited to 2:1, numerous multiplexers are required to facilitate dynamic shifting.

Another, simpler solution to enable arbitrary vertical kernel sizes (h) is to pad with pixels, so for all clock cycles S new samples are entered [S14], [S15]. All registers may contain valid or padding samples during operation. Comparisons are performed using all data registers d_k, irrespective sample validity, therefore the number of comparators required scales with the size of the virtual filter window (h_v). Padding samples are tracked and masked in the shift registers before summing to generate i_k. Mask values are periodic with h and can be generated by a circular shift register.

Figure 65 illustrates such virtual window for the $T = w = 3,\ \ S = 2$ case. Valid and padding samples in the window are marked with light blue and light grey respectively. As the window is sliding across the input image, samples x_{15}, x_{16}, and x_{17} need to be entered to calculate y_{11}. In the first clock cycle x_{15} and x_{16}, and in the second clock cycle x_{17} and x_{18} are latched in.

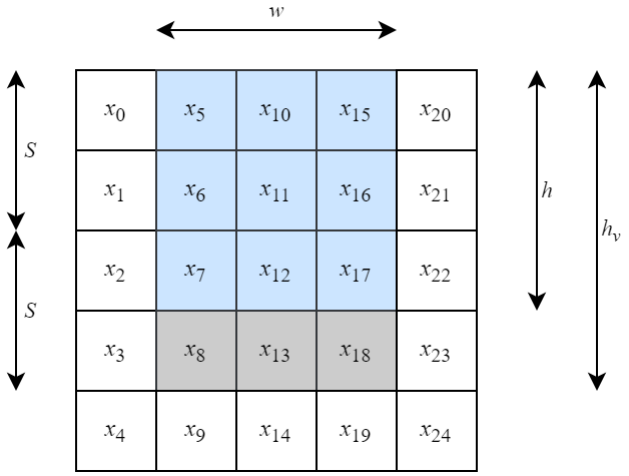


Figure 65: Virtual filter window with padding pixels

Building on the idea of a virtual filter kernel, pixels other than one on the bottom of a kernel can be masked, extending applications to circular-, or elliptical filter kernels [S16], [S17]. To support arbitrary kernel shapes, circular shift registers generating masking bits for each SHR_k stage need to be expanded to wh_v bits and need to be pre-initialized according to the desired window shape.

6.4 Results

The 2D architecture proposed in section 6.2.3 stores samples in the order of arrival and selects the appropriate output sample by calculating the location of the output sample dynamically. This architecture is easy to pipeline and to adapt to multiple samples per clock cycles. Implementation results were obtained using 24-bit RGB input, targeting Xilinx devices. The filter was configured to calculate luminance using the approximation (74) and output a 10-bit result. Table 9 summarizes achievable maximum processing frequencies (f_{max}).

Target Device	f_{max} per kernel configuration		
	3x3	5x5	7x7
XC5VSX50T-4	480 MHz	480 MHz	480 MHz
XC4VSX35-10	400 MHz	400 MHz	350 MHz
XC2V1000-4	235 MHz	225 MHz	215 MHz

Table 9. Maximum word-serial processing frequencies, 24bit RGB processing

Based on the pixel clock frequency of the video stream to be processed, the number of samples to be processed in parallel can be inferred. E.g. for 1080p60, the video clock rate is 148.5 MHz, allowing fully serial ($S = 1$) implementation of the 3x3 filter kernel on 7 series, Ultrascale, or Ultrascale+ devices. For the least recent, XC2V-5 part, only the fully parallel $S = h$ path is viable, resulting to a larger footprint and increased routing complexity.

For characterization purposes, implementations were tested with comparators in general purpose logic-fabric with the output register forced to be placed into the CLB where the carry chain ends. Since the design uses considerable slice-based arithmetic, Table 10 presents approximate LUT counts for different kernel and filter configurations.

kernel	$S = 1$	$S = 2$	$S = 3$	$S = 4$	$S = 5$	$S = 6$	$S =7$
3x3	190	442	345	N/A	N/A	N/A	N/A
5x5	630	1320	1770	3270	2100	N/A	N/A
7x7	1370	2630	4150	4400	7500	10500	6000

Table 10. Approximate LUT usage for 3x3, 5x5, and 7x7 configurations

Adder trees for the new samples were pipelined dynamically. The number of pipeline stages used is a function of T. At the first stage 4 bit sets of the SHR register are masked and summed, while every other stage summarizes two elements of the previous stage. Accordingly, the number of stages is $log_2 T - 1$. The N:1 decoder at the end of the filter consists of two pipeline stages. The first stage processes 8 bit parts, while the second stage is simply a wide OR gate.

6.5 Chapter Summary

In section 6.1.1 **I demonstrated that FPGAs are ideal implementation platforms for 1D median filters, and I proposed an architecture that resulted in a compact, high-performance solution.**

In sections 6.2.1 and 6.2.3 the 1D architecture is extended to 2D by allowing ingestion of a vector of samples instead of a single scalar. **I proposed an efficient, generalized structure for 2D rank order filtering, capable of processing a configurable number of pixels in a single clock cycle.**

Rank order filtering color channels separately may introduce color artifacts along edges. In section 6.2.2 the rank order filter architecture was extended to operate on indexed data, enabling efficient color image processing without introduction of color artifacts. Also, with the introduction of virtual or padding samples, non-rectangular, e.g., circular kernels can be processed efficiently.

7 Efficient Implementations of Fast Fourier Transform processors

The Fast Fourier transform (FFT) is a computationally efficient algorithm for computing a Discrete Fourier Transform (DFT) [67]. Appendix section A.3 reviews definition of the one-dimensional DFT, the derivation of FFT, decimation in frequency and time, compares the benefits of higher radix algorithms. Appendix section A.3.7 discusses efficient hardware implementations of the FFT on FPGAs and explores different pipelining and numerical representation options.

7.1 Design considerations

General requirements on HW FFT accelerators usually aim to minimize resource allocation, power and transform time. Numerous architectural solutions represent trade-offs between design goals.

For most FPGA users the goal is to meet certain performance requirements while fitting the design into the smallest (cheapest) FPGA possible. Historically, FFT architectures that included fewer multipliers were better suited to ASIC/FPGA implementation, however, multiplier count is no longer a good benchmark for FFTs. Instead, the total envelope of the implementation, slice count, BRAM count, and DSP slice count should be compared with the I/O rate attainable with different architectural options.

As transfer size grows, the number of operations per I/O sample scales logarithmically. For large transfer sizes, where FPGAs can outcompete DSP processors due to inherent data parallelism, FFT solution sizes are dominated by memory footprint. Any improvements in memory management that can reduce memory allocation in turn reduces the overall footprint, which can translate to cost and power saving in the overall design.

Practical implementations can be characterized on how the algorithm, multipliers, butterflies or dragonflies, twiddle sources and scaling logic, collectively referred as Processing Elements (PEs), map to FPGA resources. Design balance come into focus as columns of BRAMs and DSP48s were weaved into the logic fabric of FPGAs (Figure 66).

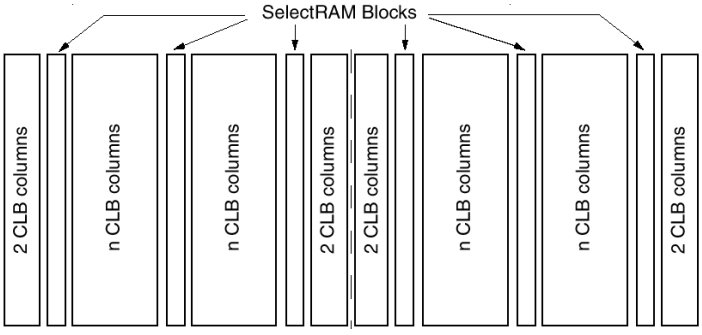


Figure 66. Dedicated resources inserted into the logic fabric

The two most widespread implementations are the loop engine (Figure 67.) and the pipeline.

In the loop engine implementation, one or more PE(s) are working concurrently on the same FFT stage. Only when a stage is complete, work on the next stage commences, so the FFT is signal flow is traversed horizontally, on a stage-by-stage basis.

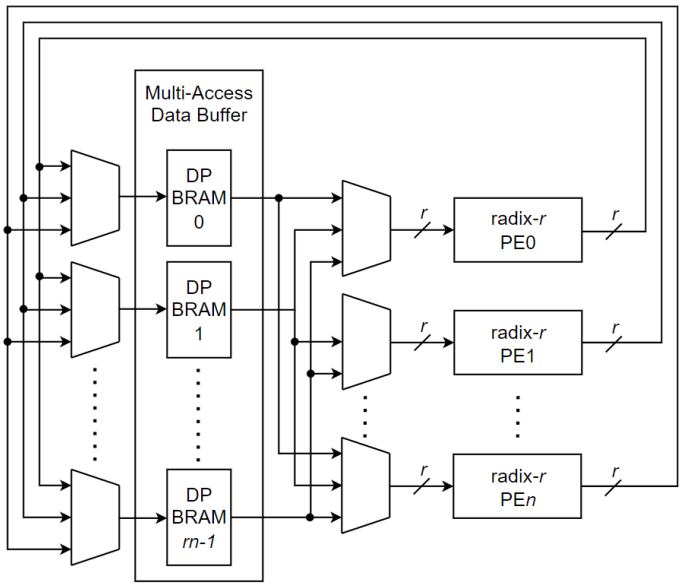


Figure 67. Loop-engine implementation

In the pipeline implementation (Figure 68) the PEs are working concurrently on different stages, with buffers between the stages, so the signal flow graph is traversed vertically.

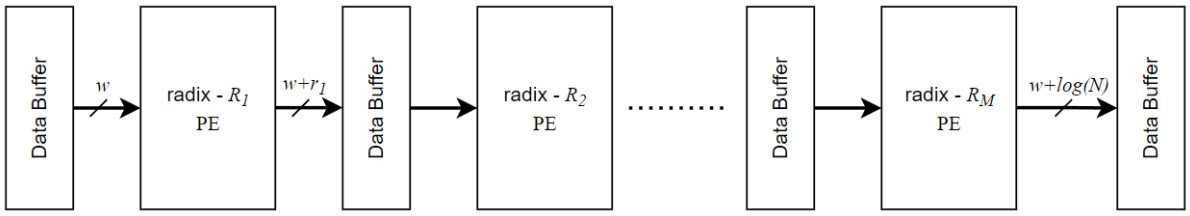


Figure 68. Pipeline implementation

7.2 Memory Segmentation

A loop engine with a parallel radix-*r* PE processes the *r* points simultaneously. To supply and store data to/from the PE at least *r* dual-ported, or 2*r* single-ported memory blocks are necessary (Figure 67). To enable single ported memories, ping–pong buffering can be used.

In this scheme, a PE reads data (operands) from one buffer and writes results back to the other buffer, and after each rank the read and write buffers are swapped. On the PE input side, the *N* point frame has to be segmented into *r* memories, each *N*/*r* samples deep. The segmentation problem is illustrated through a 16-point, radix-4 example (Figure 69).

This FFT has 2 ranks and the radix-4 FFT engine performs four radix-4 FFTs (dragonflies) per rank. Data points are accessed in the following order:

	pass 1	**pass 2**	**pass 3**	**pass 4**
rank 1	x_0, x_1, x_2, x_3	x_4, x_5, x_6, x_7	x_8, x_9, x_{10}, x_{11}	x_{12}, x_{13}, x_{14}, x_{15}
rank 2	w_0, w_4, w_8, w_{12}	w_1, w_5, w_9, w_{13}	w_2, w_6, w_{10}, w_{14}	w_3, w_7, w_{11}, w_{15}

Table 11. Data accesses, radix-4 FFT engine, *N*=16

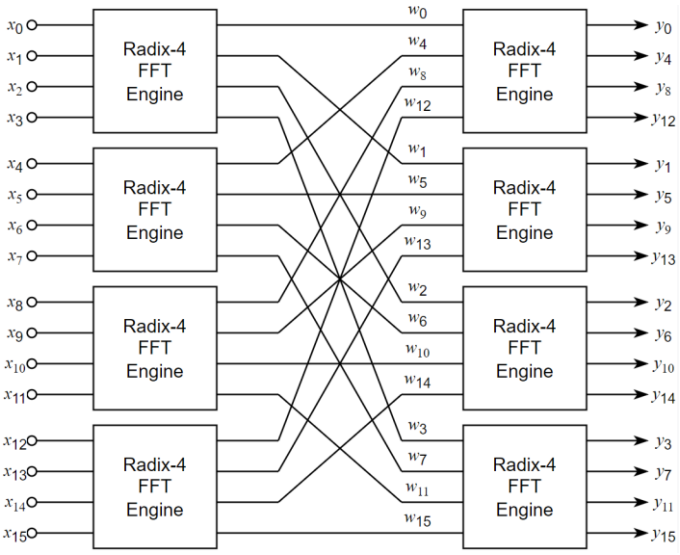


Figure 69. 16 point, radix-4 FFT

Suppose in a radix-4 loop engine implementation four buffers, buffers B0-B3 store input frames (x) and temporary results (w), with buffers B0..B3 feeding inputs 0..3 of the single radix-4 PE. Initially, B0 stores input sample x_0, x_4, x_8 and x_{12}, buffer B1 stores sample x_1, x_5, x_9 and x_{13}, etc., facilitating simultaneous access of the samples for the first rank. If during the evaluation of the first rank, results $w_0..w_3$ were written back to the original addresses and buffers of operands $x0..x3$, and so on for all passes, then w_0, w_4, w_8 and w_{12} would all end up in the same buffer, preventing simultaneous access to these operands for the second rank.

If the results were written back to locations of x_0, x_4, x_8 and x_{12}, then yet unprocessed x_4, x_8 and x_{12} would be overwritten before use. Data allocation/segmentation to buffers without contention in any of the ranks is a non-trivial problem. Successive ranks of the FFT access memory locations in different order, causing segmentation to change from rank to rank. Existing FFT solutions solved this issue by doubling the size of the buffers and alternated using the lower/upper banks for each successive rank. In an FPGA implementation this may effectively double BRAM allocations, which for large transform sizes is the bottleneck for implementation.

In [S18], [S19] I propose a static memory segmentation scheme which allows the FFT to use single buffering, saving 50% of BRAM resources (Figure 70). The solution assumes a radix-r FFT engine, performing an FFT/IFFT on a frame of N data points, where $r = 2^t$, $N = r^c$, c and t being positive integers, which fits the majority of practical hardware FFT implementations.

The problem of segmenting the data frame to r memory blocks, is analogous to storing the frame of N data points in a matrix ($\boldsymbol{A}$) with r rows and n columns, $n = N/r$. Rows of $\mathbf{A}$ correspond to different DPMs in Figure 70. By introducing the base r representation of index k:

$$k = \sum_{i=0}^{c-1} r^i d_i, \tag{77}$$

the constraints of simultaneous access of samples can be expressed simply as follows. If in the base r representation of any two indices all corresponding digits except for d_q are the same, then the samples have to reside in different rows of $\mathbf{A}$. Simplifying further, no two indices p and q can reside in the same row if any c-1 digits in the base r representation of p and q are the same.

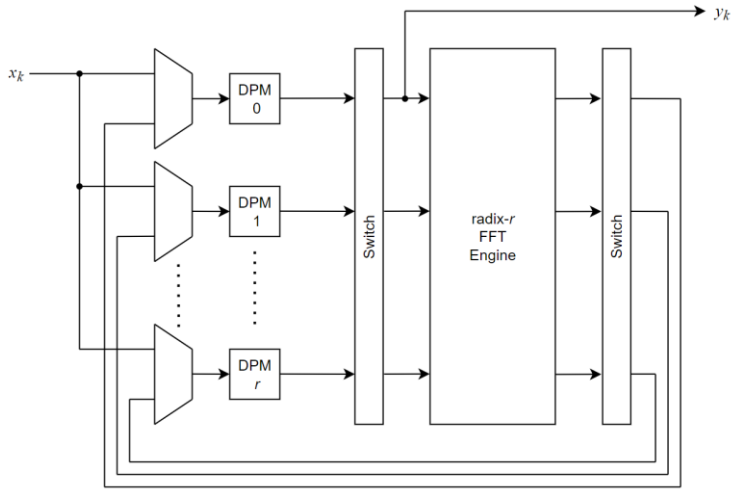


Figure 70. Single buffer Radix-4 FFT loop engine

Although there could be numerous solutions to satisfy this set of constraints, I proposed a straightforward systematic solution, which also minimizes complexity of the addressing logic. Within matrix **A**, input sample x_k should be in row a_r and column a_c:

$$a_r = \left\{ \sum_{i=0}^{c-1} d_i \right\} \bmod r \tag{78}$$

$$a_c = k \bmod n \tag{79}$$

When designing the addressing and segmentation scheme for a HW implementation, a_c is the static address of input sample x_k in BRAM $a_r(k)$. With slight modifications, the findings can be applied to cases where *N* is not an integer power of *r*.

Table 12 shows an example of the static memory segmentation for an $N = 64$, $r = 4$.

DPM0:	0	49	34	19	52	37	22	7	40	25	10	59	28	13	62	47
DPM1:	16	1	50	35	4	53	38	23	56	41	26	11	44	29	14	63
DPM2:	32	17	2	51	20	5	54	39	8	57	42	27	60	45	30	15
DPM3:	48	33	18	3	36	21	6	55	24	9	58	43	12	61	46	31

Table 12. Static memory segmentation for $N = 64$, $r = 4$

In rank 0, the radix-4 PE accesses data points 0,16,32,48. In rank 1, the radix-4 PE accesses data points 0,4,8,12. In rank 2, the radix-4 PE accesses data points 0,1,2,3.

As all these points are in different rows of the table, they physically reside in different memories, so it is possible to construct a system that accesses the data points simultaneously.

The solution can be extended to cases where $r = 2^t$, $p = 2^s$, $N = pr^c$, $s < t$, enabling run-time configurable transform lengths using the bypass option introduced in section A.3.11A.3.7. For example, a 128 point FFT can be calculated using a radix-4 PE (N=128, r=4, s=1).

Before defining row and column addresses, sample index *k* needs to be decomposed by

$$l = \frac{N}{p \cdot r}, \tag{80}$$

$$k = p\left\{\sum_{i=0}^{c-1} r^i d_i\right\} + b, \tag{81}$$

and the proposed segmentation for the run-time configurable cases becomes:

$$a_r = \left\{b + \sum_{i=0}^{c-1} d_i\right\} \bmod r \tag{82}$$

$$a_c = (k \bmod l) + l(k \bmod p) \tag{83}$$

To illustrate the segmentation, Table 13 shows an example for N=32, k=4, p=2, implemented with a radix-4 dragonfly, with an option to bypass a butterfly stage.

DPM0:	0	26	20	14	25	19	13	7
DPM1:	8	2	28	22	1	27	21	15
DPM2:	16	10	4	30	9	3	29	23
DPM3:	24	18	12	6	17	11	5	31

Table 13. Static memory segmentation for N = 32, k = 4, k'=2

In this scenario, two radix-4 ranks are followed by a radix-2 rank. The first radix-4 rank accesses samples {0,8,16,24}, {1,9,17,25}, etc. The second radix-4 rank accesses samples {0,2,4,6}, {1,3,5,7}, etc. The final radix-2 rank, with bypass active, accesses {{0,1} and {2,3}}, {{4,5} and{6,7}}, etc.

7.3 Natural to Digit-reversed reordering

An $N=r^m$ point, radix-r streaming FFT with r parallel inputs and outputs produces a continuous flow of data frames, each frame processed in N/r cycles. In most applications, FFT and IFFT blocks access data in natural order. Hence, either at the input (DIT) or at the output (DIF) is digit-reversed, and conversion to natural order may be necessary (Figure 71).

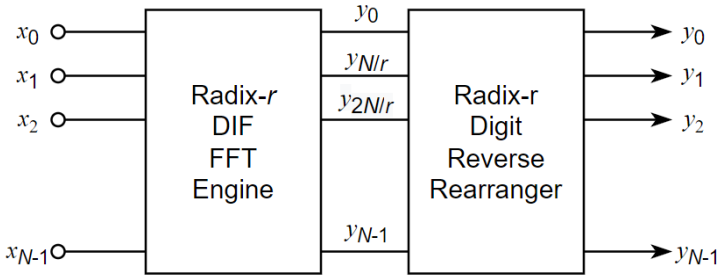


Figure 71: Radix r DIF FFT with data reorganizer

For example, a radix-2 DIF FFT consumes inputs in natural order, such as $\{x_0, x_{N/2}\}$ in the first clock cycle, and $\{x_1, x_{1+N/2}\}$ to $\{x_{N/2-1}, x_{N-1}\}$ in subsequent clock cycles. At the end of the pipeline the output spectra are presented in $\{y_0, y_{N/2}\}$, $\{y_{N/4}, y_{3N/4}\}$, etc. order.

To reorganize the output, N samples have to be stored in r memories to enable parallel access to data without contention or data loss. Following the radix-2 example, reading out of spectra y can start after outputs $\{y_1, y_{N/2+1}\}$ had been generated, effectively when half of the frame was processed. The first half of the spectra frame therefore had to be buffered. Once the pipeline is full, r samples are written, and r samples are read from the reorganizer, such that both $\{y_0, y_{N/2}\}$ and $\{y_0, y_1\}$ need

to be written to separate BRAM modules to facilitate parallel read/write access. In earlier FFT pipeline implementations data-reorganization employed double buffering: one buffer being written in digit-reversed order, while the other read out in natural order. In [S20] I proposed a method that eliminates the need for double buffering, therefore reduces memory allocation by 50%.

Data reordering (Figure 72) takes place by employing two radix-r commutators, switches with r inputs and r outputs, before and after the frame buffer. The commutator performs a circular shift on the input data-set, so output sample received on input i is assigned to commutator output (i-sel) mod r, where input sel is driven by the bit-reversed, most significant digit of address $addr$.

When $r = 2^t$, in a practical HW implementation $addr$ can be generated by an N-r+1 bit wide counter, with the MSB tracking even/odd data frames. For each subsequent frame, based on the MSB, BRAM addresses alternate between natural and digit-reversed order.

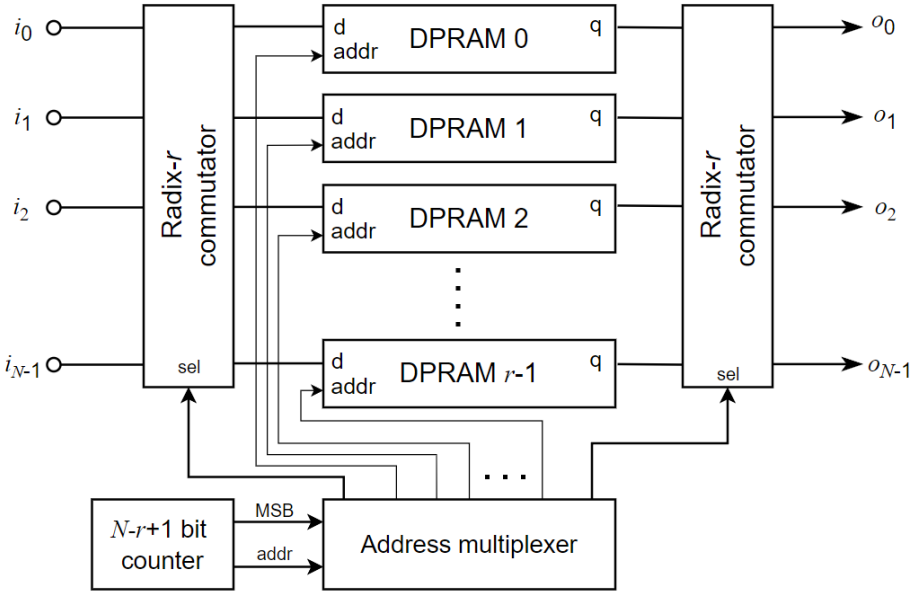


Figure 72: Example of a r input, r-output, data reorganizer for radix-r FFTs

Following the nomenclature introduced in the Memory Segmentation section above, similar to (78) and (79), sel for the input commutator, and address for BRAM k for sample n is defined as

$$sel = \left\{ \sum_{i=0}^{c-1} d_i \right\} \bmod r, \tag{84}$$

$$addr_k = addr \bmod \left(\frac{N}{r} \right), \tag{85}$$

where $addr$ alternates between natural and digit reverse order per frame.

The sel input of the output commutator is the modulo r inverse of sel driving the input commutator.

Like in the Memory Segmentation section above, the reorganizer solution can be extended to cases where $r = 2^t$, $p = 2^s$, $N = pr^c$, $s < t$, matching the input (DIT) / output (DIF) ordering of FFTs with run-time configurable transform length. For these cases, a cascade of binary counters may be used effectively implementing the addressing scheme defined in (82),(83).

7.4 Implementation Results

Tables 6-8 summarize resource usage for implementations in Xilinx 7 series devices.

Architecture / N	64	128	256	512	1024	2048	4096	8192	16384
Radix-2 pipeline	8	10	12	14	16	18	20	22	24
Radix-4 pipeline	6	9	9	12	12	15	15	18	18
Radix-2 loop, 1 PE	3	3	3	3	3	-	-	-	-
Radix-4 loop, 1 PE	9	9	9	9	9	9	9	9	9
Radix-4 loop, 2 PEs	9	9	9	9	9	18	18	18	18

Table 14. Multiplier usage by point-size and FFT implementation type (16 bit precision)

Architecture / N	64	128	256	512	1024	2048	4096	8192	16384
Radix-2 pipeline	4	6	9	101	11	161	25	43	79
Radix-4 pipeline	0	3	3	4	6	9	18	26	65
Radix-2 loop, 1 PE	0/315	0/3	3	3	3	-	-	-	-
Radix-4 loop, 1 PE	7	7	7	7	7	7	11	22	44
Radix-4 loop, 2 PEs	19	19	19	19	19	22	38	76	152

Table 15. BRAM usage by point-size and FFT implementation type (16 bit precision)

Architecture / N	64	128	256	512	1024	2048	4096	8192	16384
Radix-2 pipeline	1225	1425	1545	1851	1995	2151	2435	2700	2945
Radix-4 pipeline	1735	1855	1927	2111	2197	2351	2461	2620	2767
Radix-2 loop, 1 PE	978	1138	818	818	818	-	-	-	-
Radix-4 loop, 1 PE	2016	2036	2056	2076	2096	2116	2136	2156	2176
Radix-4 loop, 2 PEs	3072	3092	3112	3132	3152	3812	3832	3852	3872

Table 16. Slice usage by point-size and FFT implementation type (16 bit precision)

Table 17 presents transform times for 16 bit precision in Xilinx Virtex Ultrascale Plus devices [3] targeting 571 MHz clock speeds, as characterized by the vendor with the latest speedfiles[16].

Architecture / N	64	128	256	512	1024	2048	4096	8192	16384
Radix-2 pipeline	0.06	0.11	0.22	0.45	0.90	1.79	3.59	7.17	14.35
Radix-4 pipeline	0.03	0.06	0.11	0.22	0.45	0.90	1.79	3.59	7.17
Radix-2 loop, 1 PE	0.34	0.78	1.79	4.04	8.97	19.73	43.04	93.25	200.85
Radix-4 loop, 1 PE	0.08	0.20	0.45	1.01	2.24	4.93	10.76	23.31	50.21
Radix-4 loop, 2 PEs	0.04	0.10	0.22	0.50	1.12	2.47	5.38	11.66	25.11

Table 17. Transform times [us] as a function of architecture and point-size[*N*]

Table 18 demonstrates savings of BRAM allocation attributable to single buffering enabled by the proposed memory addressing schemes, using one processing engine and BRAMs for data buffering. In order to quantify savings, the BRAM configurations should be considered. The 18 kbit

[15] Distributed RAM / Block – RAM Implementation

[16] https://www.xilinx.com/html_docs/ip_docs/pru_files/xfft.html#virtexuplus

Xilinx and Lattice BRAM primitives can be configured as 512 x 36bit, which can store 512 16bit complex samples. For a radix-k FFT samples are stored in k dual-ported memories (Figure 67). E.g., a radix-2 loop-engine or pipeline stage uses at least two BRAM primitives. For 16 bit samples and $N \leq 512$, the radix-2 pipeline can support double-buffering with single BRAM primitives.

Architecture	Point size (N) / Data representation [bits]							
	512 / 16	**512 / 32**	**1024 / 16**	**1024 / 32**	**2048/16**	**2048/32**	**4096/16**	**4096/32**
Radix-2 [17]	0%	50%	50%	50%	50%	50%	50%	50%
Radix-4 [15]	N/A	N/A	0%	50%	N/A	N/A	50%	50%
Radix-2 [18]	0%	50%	50%	50%	50%	50%	50%	50%
Radix-4 [16]	N/A	N/A	0%	50%	N/A	N/A	50%	50%

Table 18. Reduction in BRAM allocation compared to naïve double-buffered implementation.

7.5 Chapter summary

Section A.3.1 introduces the DFT and the FFT along with well documented implementation optimizations to reduce the complexity of the signal flow graph and the corresponding serial, or parallel implementations. 2D FFT and IFFT operations are important preprocessing steps for image classification [S21], [S22]. Section A.3.7 discusses mapping FFT signal flow graphs to FPGA resources and identifies complex multipliers as well as local memories for sample and twiddle factor storage as critical resources to be optimized for FPGA implementations. I have shown that with a large number of parallel multipliers and adjacent local memories, FPGAs can implement FFT/IFFT operations with configurable transform lengths efficiently, providing trade-offs between resource allocation and transform time [S23]. Silicon area in HW implementations with $N > 4096$ points are dominated by memory. In previous implementations data frames were double buffered both in pipelined and loop-engine configurations to facilitate random access to samples without output samples overwriting input samples. In section 7.2 **I proposed methods and architectures to reduce the depth of memory buffers by 50% for in-place FFT implementations by eliminating double buffering [S18].**

Section 7.3 discusses accessing input and output samples. DIT processing structures expect the input data frame in bit/digit reversed order and generate an output frame in natural order. Conversely, a DIF FFT expects the input frame in natural order, but the output frame is in bit/digit reversed order. To access input and output samples in natural order, a reorganizer stage is necessary, which previously double-buffered the frame. **I proposed a method to reorganize input data to an FFT from natural to digit-reversed order, or results of an FFT from digit-reversed order to natural order without double-buffering frames.[S20]**.

[17] Pipeline or loop-engine architecture, no sample reordering, no block floating point

[18] Pipeline architecture, with natural sample reorder stage, parallel multi-channel load/unload

8 Direct Digital Synthesis

8.1 Introduction

The typical DDS block introduced by Tierney *et al.* in [68] consists of a phase accumulator and a phase to sinusoid converter. In its simplest form, the phase accumulator is an unsigned, registered adder, and the phase-to-amplitude converter is a Look-Up-Table (LUT). The DDS takes a clock reference and a phase increment ($\Delta\varphi$) or frequency control word input and generates samples of a sine with frequency:

$$f = f_{ref} \frac{\Delta\phi}{2^M}, \tag{86}$$

where $\Delta\varphi$ is the phase increment, M is the width of the phase accumulator and f_{ref} is the frequency of the reference clock. The DDS maintains the frequency stability of the reference signal, has unparalleled frequency switching rates, ensures phase continuity on switching, and has frequency resolution typically in the sub-Hertz range [68], which makes it an ideal building block in radar and communication systems such as transceivers, modems, and up/down converters.

Requirements of DDS designs are small size, low power consumption, and high level of spectral purity. Apart from timing (jitter) issues, noise on the output can be attributed to phase accumulator quantization error, and phase-to-amplitude quantization error. The phase accumulator quantization error can be easily combated by widening the accumulator. However, the LUT size grows exponentially by M and linearly by the width of the output.

With trivial operations typically referred as coarse rotation, the symmetries of the sine function can be exploited by storing one quadrant of the complex plane only. However, stringent memory limitations may not allow a fully look-up-table (LUT) based solution even when only one quadrant has to be stored.

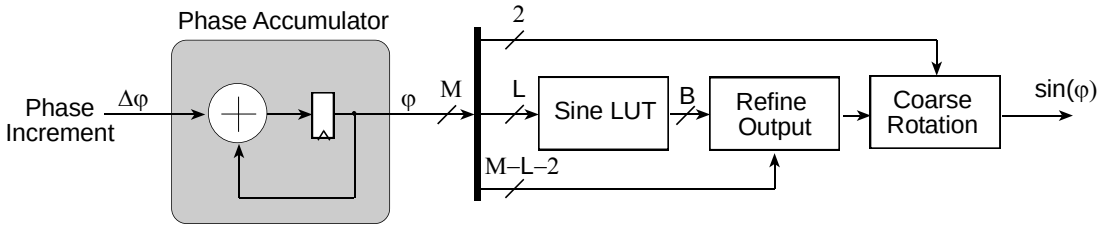


Figure 73. Typical DDS block diagram

Consequently, for low-noise and high period length DDS applications, where a fully LUT based solution would be prohibitively large, a B bits wide, 2^L deep LUT, addressed by L bits ($L < M - 2$) can be used with extra circuitry (Figure 73) to generate samples missing from the LUT. Output values are refined using the less significant bits of the phase accumulator (φ). The arithmetic operations used during this operation significantly contribute to output noise. To measure output phase and amplitude noise, define the output of the DDS as

$$f(n) = a\, sin(\frac{2\pi f_{out}}{f_{ref}} n + \varphi) - \varepsilon(n), \tag{87}$$

where a is the amplitude and φ is the initial phase of the expected output signal, and $\varepsilon(n)$ is the error. To find the phase and amplitude of a sinusoid which fits best an arbitrary discrete set of N values, $y(n)$, minimizing error $\varepsilon(n)$:

$$\begin{bmatrix} y(0) \\ y(1) \\ y(2) \\ \vdots \\ y(N-1) \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ \cos\omega & \sin\omega \\ \cos 2\omega & \sin 2\omega \\ \vdots & \vdots \\ \cos(N-1)\omega & \sin(N-1)\omega \end{bmatrix} \underline{x} + \begin{bmatrix} \varepsilon(0) \\ \varepsilon(1) \\ \varepsilon(2) \\ \vdots \\ \varepsilon(N-1) \end{bmatrix}, \tag{88}$$

where $$\underline{x} = [0,1,..2]^T, \text{ and } \omega = \frac{2\pi}{N} \tag{89}$$

or $\underline{y} = \boldsymbol{A}\underline{x} + \underline{\varepsilon}$, for which the solution can be found as

$$\underline{x} = (\boldsymbol{A}^T\boldsymbol{A})^{-1}\boldsymbol{A}^T\underline{y}, \tag{90}$$

$$\varphi = \tan^{-1}\frac{x_1}{x_0},\ a = \sqrt{{x_0}^2 + {x_1}^2} \tag{91}$$

The least squares method minimizes noise by associating phase and amplitude errors with the signal, thereby revealing the effect of time domain noise to the phase and amplitude of the quadrature signals if said noisy sinusoidal was used for Fourier analysis.

A simpler way to assess the quality of the DDS output is using the DFT. The N point, complex valued DFT of the output series was defined in (117). Since $f(n)$ values are real,

$$F\left(k + \frac{N}{2}\right) = \overline{F\left(\frac{N}{2} - k\right)}, \qquad k = 0, \ldots, \frac{N}{2} - 1, \tag{92}$$

and the power spectral density (PSD) is

$$P(k) = F(k)\overline{F(k)}, \tag{93}$$

The energy of the signal is concentrated to one bin $P(0)$. The rest of the bins contain the PSD of the noise $\varepsilon(n)$. The benchmarks used to measure DDS output quality and noise are the Signal to Noise Ratio (SNR),

$$SNR = 10\,log_{10}\left(\frac{P(0)}{\sum_{i=1}^{\frac{N}{2}-1} \mathrm{P}(i)}\right), \tag{94}$$

and the Spurious Free Dynamic Range (SFDR), defined as:

$$SFDR = 10\log_{10}\left(\frac{P(0)}{\max(P_k)|k > 0}\right), \tag{95}$$

An upper bound on the SFDR based on the number of bits in the sine LUT representation (B) can be shown to be

$$SFDR \leq 6.02B - 3.92\text{dB} \tag{96}$$

8.1.1 FFT specific application

The twiddle factors used during the DFT/FFT computation are:

$$W_N^{nk} = \cos\left(\frac{2\pi nk}{N}\right) - \mathrm{j}\sin\left(\frac{2\pi nk}{N}\right) = e^{\mathrm{j}\frac{2\pi nk}{N}} \tag{97}$$

The twiddle factor access sequence is always monotonous but depends on the actual FFT structure used. In pipelined FFT processors, multipliers in each PE have designated twiddle factor generators. In the in-place (loop-engine) structure the PE needs different twiddle factors for each rank. However, all W_N^{nk} (k>1, $0 \le n < N/r$) series are sub-sets of W_N^n. Therefore, the ultimate challenge is to generate high-quality twiddle factors pertinent to W_N^n.

Although the FFT algorithm reorders the computation of (119), fundamentally each *X(k)* is a convolution of the data *x(n)* and the twiddle factor series W_N^{nk}, therefore noise present in the twiddle factor series has similar effects as noise in the data series. For this reason, high point-size, low-noise FFTs need high quality, spurious-free twiddle factors. The purpose of this analysis is to compare current DDS algorithms and propose an efficient and compact solution for twiddle factor generation for FFT and IFFT implementations. The comparison criteria are precision, maximum operating frequency and resource usage in FPGAs.

8.1.2 Research Goals

My goal was to design a twiddle factor generator for FFTs optimized in the following terms:

- Support for $2^{12} \le N \le 2^{20}$ samples period length,
- SFDR >150 dB,
- capable of high-speed implementation in FPGAs or ASICs,
- minimal footprint in FPGA implementation.

In addition, because twiddle factors are quadrature signals, both cos(φ) and sin(φ) should be available at the same time. The ideal Quadrature Direct Digital Frequency Synthesizer (QDDFS) twiddle factor generator must support accessing samples (twiddle factors) in both forward and reverse direction, to calculate FFTs and IFFTs.

SFDR and SNR measure the quality of results. A widely used target function to measure the success of compressing the DDS design is compression ratio [69]. Compression ratio measures the reduction in size of a particular DDS structure compared to the direct implementation (with coarse-rotation) for the same SFDR/SNR performance. Historically transistor-, or equivalent gate counts had been used to compare the footprint of different architectures [70],[71]. For FPGAs, the concept of *compression* can be generalized if besides memory allocation, the footprints of all arithmetic components are considered. In many FPGAs [2][3],0[73] CLBs, BRAMs and DSP slices/multiplier blocks constitute a tile (Figure 74). Vertically connected tiles form a bank, and horizontal regions in the chip form a local CLK region.

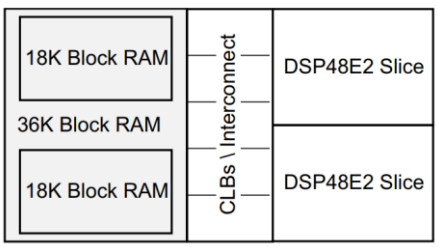


Figure 74: Xilinx Ultrascale BRAM, CLB and DSP48 slice tile

For a particular target device, the number of CLBs (slices, registers, LUTs), BRAMs and DSP slices/multiplier blocks can establish the exact conversion ratios between resources. While in Xilinx FPGAs the exact ratio of registers/BRAMs/DSP slices is family and package dependent, based on Virtex-7 averages[19] in this analysis a 18kbit BRAM is treated equivalent to 400 registers, and 1 DSP slice or 18 bit multiplier. The compression ratio thus can be redefined as the quotient of the aggregated equivalent register count allocated by a particular DDS structure, versus the direct memory implementation with coarse rotation, designed for the same numerical performance. For N=2^{16} and B=24 bits, a LUT of 2^{14} ×24 bits = 384 kbit is required, which uses 24 BRAMs, plus the coarse rotation logic, having an equivalent register count of 9648.

Since twiddle factors have a fixed order in the FFT algorithm, the phase increment is constant. This allows recursive calculation of samples, which can lead to significant savings in logic complexity. Also, the DDS can exploit that phase-increment is always a power of 2 for twiddle factor generation.

8.2 Current Solutions

The numerous approaches currently available for phase to amplitude mapping (sine generation) can be classified into the following categories:

- piecewise-linear interpolations [71], [74], [75]
- Taylor-series approximations [76], [77]
- Chebyshev approximation [70],
- trigonometric decomposition [68], [78][77], [79]
- recursive solutions, [80]
- coordinate rotations [81], [82]

The piecewise-linear, Taylor-series, Chebyshev, and trigonometric decomposition algorithms all fit under the umbrella of polynomial approximations. The difference is only in the method of coefficient computations.

Recursive solutions are based either on a resonator, or a complex multiplier in a feed-back loop, which has the added benefit of generating quadrature signals without extra cost.

Some solutions [69] are combinations of the above techniques. For 12 bit precision (SNR ~ 85 dB), Table 19, modified from [74], summarizes results of different implementations. The implementation details of the solutions above are discussed further in Appendix section A.4.1.

8.3 Digital Resonators

An intuitive solution for the twiddle generation problem is using an IIR filter / oscillator [79] with poles very close to the unit circle, so the impulse response approximates the desired sinusoid function. The z-transform of sin $(n\omega_0)$ is:

$$y(z) = \frac{z^{-1}\sin\ \omega_0}{1-2z^{-1}\cos\ \omega_0+z^{-2}}, \tag{98}$$

which can be realized as

[19] 7 Series FPGAs Data Sheet, Xilinx Inc, DS180 (v2.6.1) September 8, 2020, Table 8: Virtex-7 FPGA Feature Summary. https://www.xilinx.com/content/dam/xilinx/support/documents/data_sheets/ds180_7Series_Overview.pdf

$$y(z) = x(z)H(z), \text{ where} \tag{99}$$

$$x(z) = \delta(z)\, sin\, \omega_0 \text{ and}$$

$$H(z) = \frac{z^{-1}}{1-2z^{-1}\, cos\, \omega_0 + z^{-2}}, \tag{100}$$

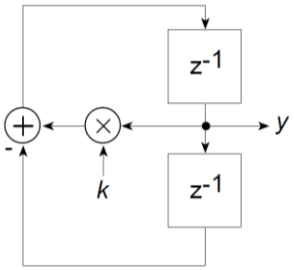


Figure 75. Two-pole resonator

Direct form II, second order IIR stages [83] lend themselves well to low-cost FPGA implementation. Ideally the LUT could be removed from the design and a delta pulse can be incorporated into the initial conditions of the filter.

Figure 75 shows the direct form II realization of the 2 pole filter defined by (100). The filter contains only one constant coefficient ($k = 2\, cos\, \omega_0$) multiplier, which makes it an appealing candidate for DDS implementation on FPGAs. Also, regardless quantization of k the poles are constrained to lie on the unit circle, given that $-2<k<2$.

Quantization of k results only to deviation from the expected frequency. Having poles on the unit circle makes the filter prone to error accumulation. However, for twiddle factor generation, the circuit can be reset, or re-initialized periodically to suppress noise accumulation.

8.4 Quadratic interpolation

Quadratic interpolation drastically reduces the number of segments required to represent $f(\varphi)$ with the required precision. If uniform quantization is used only 71 points are necessary to reduce spurious content under 150 dB, or 49 points if non-uniform quantization is used. However, similar to linear interpolation, for non-uniform quantization the reduction in coefficient memory depth is offset by increased arithmetic and control complexity as well as storage requirements for 4 coefficients per segment. Reduction in memory is traded for arithmetic components: one BRAM is sufficient to store the 3x71x32 bits of information, but multiplier count is doubled (8 primitives), and at least two 32 bit adders are necessary to perform the $y = a_i x^2 + b_i x + c$ operation. In addition, quadratic constants and terms have larger dynamic ranges, which may further widen data-paths for storage and arithmetic components. In [S24] I provided a method to calculate the quadratic formula with B_d precision by using only B_d adders, if a_i and b_i were represented as B_d bits wide CSD coefficients.

8.4.1 Differentiator-Integrator structures

To introduce a simpler structure for second-order approximation, linear interpolation has to be examined in more detail. A discrete function $f(\varphi)$ for $\varphi_i \leq \varphi \leq \varphi_{i+1}$ can be interpolated by

$$f(\varphi) = \frac{f(\varphi_{i+1}) - f(\varphi_i)}{\varphi_{i+1} - \varphi_i}(\varphi - \varphi_i) + f(\varphi_i) = a_i(\varphi - \varphi_i) + b_i, i \in \mathbb{Z}, i < 2^L \tag{101}$$

where φ_i is the down-sampled phase input to the quadrature LUT introduced in Figure 73, and $f(\varphi_i)$ is the LUT output.

However, this definition of a_i and b_i forces the interpolant to the convex curvature side of $f(\varphi)$, resulting to sub-optimal interpolation errors. More advanced methods, such as least squares (90) can be used to determine a_i and b_i to distribute and minimize total error.

If the domain of $f(\varphi)$, $\pi/2,$ is sliced to 2^L uniform sections, so $S = \frac{N}{2^L} = 2^{M-L-2}$,

$$\frac{1}{S} = \frac{1}{\varphi_{i+1} - \varphi_i} \tag{102}$$

is constant, then successive samples from $f(\varphi_i)$ to $f(\varphi_{i+1})$ can be obtained by preloading an integrator with $b_i$and then accumulating a_i over S cycles (Figure 76).

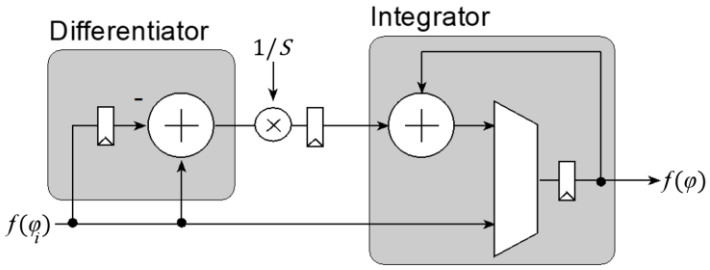


Figure 76. DI structure with preload

This structure exploits that output samples are accessed in a continuous monotonously increasing order.

If the sample-rate conversion between the input and the output is $S = 2^{M-L-2} = 2^{B_S}$, then the constant multiplication in $a_i = \frac{1}{S}(f(\varphi_{i+1}) - f(\varphi_i))$ can be replaced by bit-shifting, or more precisely, by shifting the binary point. To generate the complex valued twiddle factor series W_N^n (97), the two target functions are:

$$f_s(n) = \sin\left(\frac{2\pi n}{N}\right) \text{ and} \tag{103}$$

$$f_c(n) = \cos\left(\frac{2\pi n}{N}\right) \tag{104}$$

Both $f_s(n)$and $f_c(n)$are periodic with a period length of N. If mid-period reinitialization is not a requirement, the multiplexer based pre-load logic can be replaced by synchronous set/reset. Retaining the pre-load logic to facilitate piece-wise interpolation offers a trade-off between the extra cost of the multiplexer and the coefficient memory but allows increased operating rates due to reduced accumulator width. Overall resource use vs operating speed and noise can be optimized by simulating interpolator noise accumulation.

8.4.2 Continuous Quadratic Interpolation

The above steps to interpolate $f(n)$ can be used to interpolate the first order derivative $f'(n)$ as well. A second order interpolator can be built by nesting two DI stages (Figure 77).

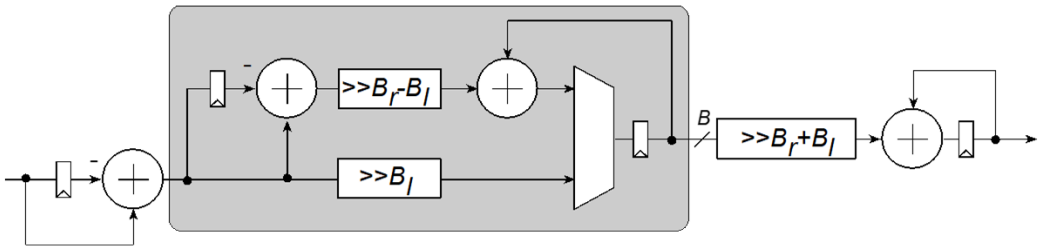


Figure 77. DDII structure

Quadratic interpolation drastically reduces algorithmic error compared to linear interpolation, allowing interpolation through longer sections while keeping an upper bound on the interpolation error. Distributed memory may be sufficient to store samples since only $S=N/M$ samples have to be pre-calculated and stored. As for linear interpolation, the first differentiator stage can be removed from the DDII structure if discrete differentials (first order derivative) of $f(\varphi)$ are pre-calculated and stored. Section A.4.3 provides additional information on optimization options and FPGA implementation results for continuous quadratic interpolators.

To study the arithmetic error of the interpolator, data were represented using the same word-length (B) through all arithmetic components. Noise plotted on the two-dimensional graphs (Figure 78) clearly show the two regions, one dominated by cumulated arithmetic noise ($B < 40$), and the one dominated by the interpolation error (B>48).

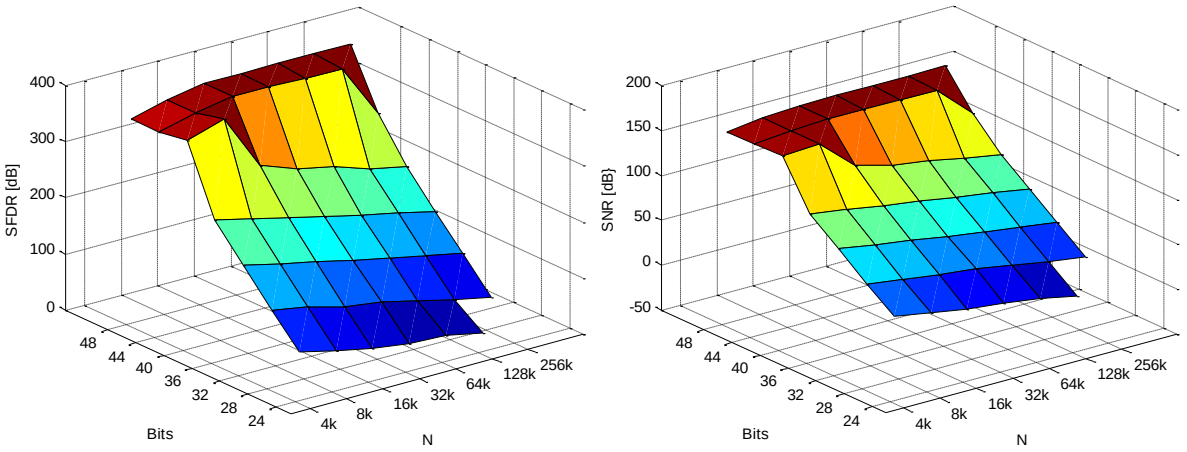


Figure 78. SFDR (left) and SNR (right) as a function of N and B, M=128

8.5 Cascaded Resonator Interpolator structure

To generate W_N^n, $N>2^{17}$, the BRAM based LUT providing samples for the interpolator structure can be replaced by a pipelined resonator. Generally, the resonator structure requires at least one multiplier, which has half the cost of a BRAM primitive in Xilinx FPGAs. However, continuous quadratic interpolation provides S output samples for every sample retrieved from or generated by the coarse LUT of Figure 73. Choosing S sufficiently large allows generating the input series using *a resonator with a bit-serial multiplier*.

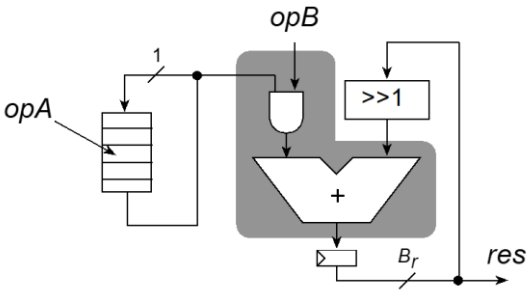


Figure 79. Bit-serial multiplier

The bit-serial multiplier maps to FPGAs efficiently as CLBs are typically designed to include XOR and multiplexer primitives to accelerate full adder and fast carry chain implementations. Also, the output register is present next to each LUT in slices. In Ultrascale parts each slice can implement an 8 bit adder. Shifting by one bit can be hard-wired, therefore the bit-serial structure allocates only $B_r/8$ slices. In Xilinx FPGAs slices can be configured as 16 bit shift registers which can be cascaded to form the circular shift registers holding operand A.

For wide operands, or low cost parts (e.g. Lattice MachX03), operating speed can be improved by breaking the carry chain and pipelining the adder (Figure 80).

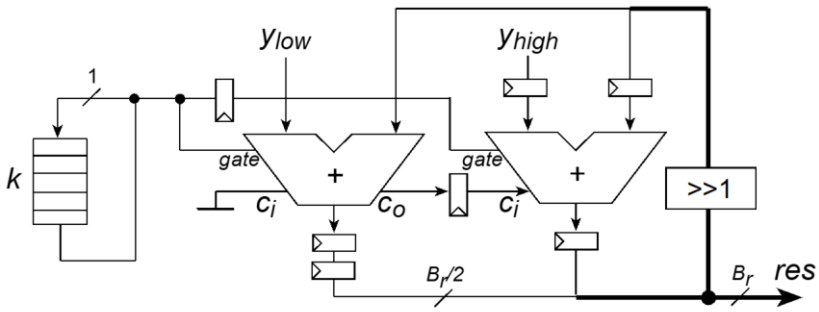


Figure 80. Pipelined bit-serial multiplier

The structure illustrated on Figure 80 can perform multiplications $k_1 y_1$ and $k_2 y_2$ in $2B_k$ CLK cycles, where B_k is the number of bits representing k. The circular shift-register holding k should store $k_1$and k_2 in a bit-interlaced fashion, and operand y should be equal to y_1 for even and $y_2$for odd CLK cycles.

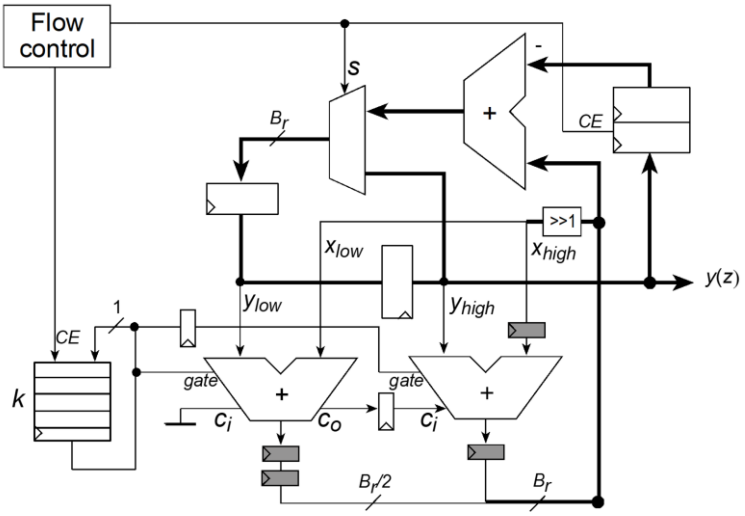


Figure 81. Pipelined bit-serial resonator

If $k_1 = k_2 = k$, then bits of k at the output of the circular shift-register have to be available for two CLK cycles. Figure 81 presents a two-stage (p=2) pipelined resonator structure with a bit-serial multiplier. The structure works as follows. Assuming $B_k = 63$, k being represented in a 63.61 bit fixed-point format, the pipelined bit-serial multiplier calculates $ky_1$and ky_2 in 2(B_k+1)=128 CLK cycles. During multiplication, adders are accumulating y_1 and y_2, while the circular shift-register supplies subsequent bits of k (LSB to MSB) every second CLK cycle. At CLK cycle 126 and 127, ky_1 and ky_2 are ready to be latched into the circulating registers when Clock Enable (CE) generated by the flow-control modules is set appropriately. These registers, along with the subtractor are responsible for forming the transfer function of the resonator (100).

During the last two CLK cycles, the registers holding partial product results (grayed out on Figure 81) must be re-initialized. Initialization to a non-zero value allows rounding of multiplier results. When the $B_k + B_r$ bits wide result is fed back to the delay-registers, B_k bits are dropped. Had the result been simply truncated, on average -½ LSB error would be accumulated with each resonator output sample. Due to the accumulating error the resonator output would quickly diverge.

If $y_1(0)$ and $y_2(0)$ are chosen such that $y_1(0) = 0$, and $y_2(0) = 1.0$, the structure generates quadrature signals in every 128 CLK cycles and can replace the BRAM based LUT. The interpolators must use uniform domain quantization, which equals the sample rate conversion of the bit-serial resonator, due to the fact that the input sample rate is fixed by the representation of k in the resonator. Also, the resonator does not need any addressing, which allows merging control components for the interpolators and the resonator. Control logic for the structure can be implemented using a counter with decoders signaling CLK cycles $2B_k$ and $2B_k$+1, which control the multiplexers in the resonator, loads the integrators, enables register loading in the resonator, and initializes partial products with ½ LSB rounding constant.

8.5.1 Results

Noise in DDS / QDDFS is introduced by algorithmic approximation and arithmetic errors. The approximation error is a function of the sinusoid period length (N) and the uniform interpolation length (S). As S increases the polynomial approximation increasingly deviates, resulting to increased noise in the generated quadrature signals. If N is doubled, but section length S is kept constant, effective ω is halved. Since the omitted higher order derivates in the Taylor series decomposition of $f(\varphi)$ are inversely proportional to powers of ω, as ω is reduced the omitted series is less significant, resulting to a better fit, and in turn attenuated approximation noise, and increased SNR and SFDR.

Arithmetic errors can be introduced by the resonator or the integrators in the interpolator, chiefly due to quantization errors after multiplication and data scaling. Simulation results for the cascaded resonator + quadratic interpolator (B=36, S=128) are illustrated on Figure 82. The 1/128 sample-rate sinusoid source for the quadratic interpolator was the digital resonator, with outputs quantized to 36 bits. Noise introduced by the resonators can be controlled by proper quantization, the selection of B_k and B_r. For a resonator with pipelined bit-serial multipliers, which takes $2(B_k + 1)$ cycles to generate quadrature signals, selection of B_k is bound by $2(B_k + 1) = S = 2^{B_s}$. Increasing S is constrained by the quadratic interpolation error, decreasing B_k is constrained by arithmetic error introduced by the resonator. The practical solutions are S={32, 64, 128, 256}, B_k = {15, 31, 63, 127}. Simulation results showed that B_k = 63 was optimal to maximize SFDR while minimizing implementation footprint. After B_k is assigned, B_r can be estimated by

$$SNR \approx 7.5B - 170\text{dB} \tag{105}$$

given that $B_r \leq B_k$. The resonator is designed to insert less noise than a LUT / BRAM with 36 bit quantization (SNR=218.7), therefore $B_r > 52$.

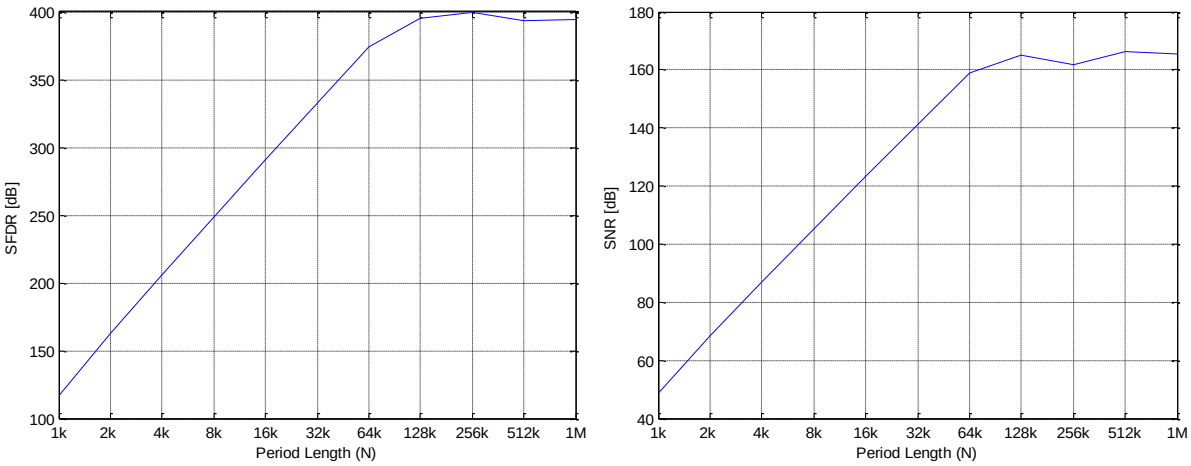


Figure 82. SFDR (left), and SNR (right) for the resonator + quadratic interpolator, as a function of N

Selecting $B_r = B_k = 63$ results to cumulated quantization noise of arithmetic components orders of magnitude smaller than the input quantization noise, which in turn allows generalization of noise characteristics of Figure 82 to the interpolated resonator.

Existing results on the field used ROM compression ratio to compare architectures. For FPGA implementation, I generalized compression ratio to take into account all FPGA resources used (section 8.1.2). In order to compare the proposed solutions to existing solutions and meta-analysis results, this generalized compression ratio is calculated as the equivalent register count of a direct implementation yielding the posted SFDR.

For FPGAs, the coordinate rotation (CORDIC) IP for (N=2^{16}, B=32) occupies 3620 registers[20] in a XC7K70T. The DDS core currently available from Xilinx [76] employs Taylor-series approximation, and for SFDR=140 dB, implemented in a XC7K70T, N=2^{25}, B=25 the design occupies 372 registers, 3 BRAM18s, 5 DSP slices primitives, resulting to an equivalent register count of 372+8*400=3572.

Method	ROM bits	Equivalent Registers	Compression Ratio	SFDR [dB]
Direct, ROM-based solution	16384x24	9648	1	144
Xilinx CORDIC (32 stages, 32 bit)	-	3620	2.67	160
Xilinx DDS	512x32	3572	2.7	140
Angular Decomposition [79]	832	1260	7.65	96
Optimized Linear Interp. [74]	960	1370	7.04	96.2
5th degree Taylor [70]	-	2496	3.87	82.5
Proposed resonator QDDFS	16384*1	1150	8.38	173
Proposed Quadratic Interpolator	512*36	1100	8.77	412
Proposed cascaded QDDFS	-	550	17.54	370

Table 19. Algorithmic Techniques (Memory Compression and SFDR).

To generate N=2^{17} data points with SFDR=150 dBs, a 2^{15} deep by 26 bit (96) wide LUT is necessary, exploiting quarter wave symmetry, which would take 52 BRAM primitives in a Xilinx FPGAs. For

[20] https://www.xilinx.com/htmldocs/ip_docs/pru_files/cordic.html

this performance, the proposed resonator structure uses one BRAM primitive and a 18x18 multiplier (one DSP slices), register banks, as well as several wide (32 to 53 bit) adders, some of which can be absorbed into the DSP slice, altogether accounting to 350 registers. The equivalent register count therefore is 1150, resulting to a compression ratio of 8.38.

In comparison with the first order Taylor-series, or linear interpolation based DDS, the proposed quadratic interpolation architecture trades two 18x18 Block Multiplier primitives for 6 adders. This is definitely favorable for ($N \geq 2^{14}$, or $B \geq 24$) FFT designs already block-RAM and multiplier heavy. Moreover, the proposed architecture extends the point-size (N) range supported by the current solution, support dynamic adjustment of twiddle frequency, while the attainable SFDR > 400 dB. Equivalent register count for this architecture is 1100.

8.6 Chapter Summary

Pipeline FFT solutions require static twiddle factor sources for each rank, generating only one twiddle (quadrature) series, with k based on the rank and multiplier position within the processing element (Figure 92). In contrast, as introduced in section 8.1.1, compact, in-place (loop-engine) FFT implementations require twiddle factor sources which generate different sequences for each rank.

Twiddle factors (W_N^{nk}) for FFT computations are traditionally pre-computed and stored in a single LUT with coarse-rotation logic. For 36 bit coefficients, sufficient for most applications, this solution maps efficiently to Lattice and Xilinx FPGA BRAMs for $N \leq 2^{11}$. For $N > 2^{11}$, however, block-RAM counts grow proportional to N. Targeting these applications, I adapted two existing twiddle factor generation QDDFS architectures for FFTs for optimal implementation in FPGAs, then analyzed and optimized their performance.

I proposed a novel Quadrature Direct Digital Frequency Synthesizer architecture for sine / cosine synthesis optimized for twiddle factor generation (W_N^{nk}) for FFTs, using a quadratic differentiator – integrator structure [S25]. This QDDFS can supply coefficients with SFDR>410 dB for $2^{12} \leq N \leq 2^{16}$, using only one BRAM primitive and six 36 bit adders. With configurable subsampling- and interpolation length ($S = 2^p$, $1 \leq p \leq 7$), as well as configurable phase addressing of the BRAM based LUT, the solution can provide twiddle factors for in-place processing engines, dynamically configuring ranks for $N \leq 2^{16}$ FFTs. **I proposed a cascaded architecture using a digital resonator as the primary, and a continuous quadratic interpolating secondary stage as a high-quality, compact twiddle-factor source for large (N>2^{17}) FFTs. [S26].**

The bit-serial digital resonator solution, which replaces the BRAM based LUT to provide samples to be interpolated to the quadratic interpolator stage, computes 63-bit complex twiddle factors in 128 CLK cycles. For in-place FFTs, the angular frequency of the generated sequence can be selected by rank dependent selection of coefficient k. The solution demonstrated superior compression ratios and very low noise compared to other methods available, using slice based logic only: four 32 bit adders, six 36 bit adders and 3 multiplexers allocating less than 550 registers. The cascaded architecture demonstrated SFDR>370 dB and SNR>160 dB for $N \geq 2^{17}$ points. For comparable numerical performance, the CORDIC algorithm requires 2 banks of 32 x 32bit adders and control logic, allocating more than 2000 registers. The direct (LUT) + coarse rotation implementation for this performance level would require 72 BRAMs.

Appendix

A.1 Processor and storage component granularity and binding

This section reviews:

- single-, and multi-processor systems,
- multi-core and superscalar systems,
- memory access hierarchy,
- concepts of reconfigurable systems,
- basic optimization concepts such as pipelining and predictive execution,
- FPGA implementation optimizations of DSP primitives.

A.1.1 Single Processor Systems

Single, Application Specific Standard Parts (ASSPs) provide minimum cost solutions for low computational performance problems, such as audio processing and low resolution imaging. The simplest processors are scalar Single Instruction Single Data (SISD) processors, where instruction fetching, decoding, operand fetching, instruction execution and result storage are carried out sequentially. Modern processor architectures, such as the ARM or PowerPC cores in Xilinx FPGAs, are highly pipelined resulting to higher operating clock rates, but increasingly complex control units to guarantee data integrity.

One way to increase throughput is operating on multiple data items simultaneously, fetching 128-, or 256 Very Long Instruction Words (VLIW). VLIWs are typically processed on Single Instruction Multiple Data (SIMD) architectures, where each instruction may operate on multiple operands, an example of which is Intel's SSE[21]. In a superscalar processor each instruction processes one data item in a pipelined fashion (MIMD), with multiple redundant functional units within the CPU so that multiple instructions can be processing separate data items concurrently.

With VLIW, the task of dependency checking by dedicated hardware logic at run time is removed and delegated to the compiler. Also from the software side, simultaneous multithreading (SMT), is a technique which improves the overall efficiency of superscalar CPUs. SMT permits multiple independent threads of execution to better utilize the resources provided by modern processor architectures. Current FPGAs without a dedicated, hardened processor system have sufficient logic resources to include a "soft" processor[22], customized to the needs of the application. For example, a control-heavy application may not need a Floating Point Processing Unit (FPU), which can be disabled during configuration. All 3 major FPGA vendors (Xilinx, Altera, and Lattice Semiconductor) offer soft processors, some providing multiple performance tiers.

A.1.2 Multi-processor Systems

To further increase performance, simultaneous threads / processes can be assigned to separate processor units which may execute instructions in parallel. This solution can be particularly attractive for certain configurable Signal Processing solutions, previously employing multiple processor cards[S19],[S22] to match the computational load. In terms of reliability, multi-processor systems are more exposed to failures if all processing nodes have to perform for system-level functionality. On the other hand, highly fault-tolerant multiprocessor systems can be designed when redundancy is provided by either hot-, warm-, or cold standby cores as backup.

Superscalar processors offer finer grain redundancy with redundant functional units, not entire processors. A single processor is composed of functional units such as the ALU, integer multiplier,

[21] https://www.intel.com/content/www/us/en/support/articles/000005779/processors.html

[22] https://www.xilinx.com/support/documentation/selection-guides/ultrascale-plus-fpga-product-selection-guide.pdf

integer shifter, floating point unit, etc. On a superscalar design there may be multiple versions of each functional unit which can simultaneously be harnessed to process instructions concurrently. This differs from a multi-core CPU that concurrently processes instructions from multiple threads, one thread per core. A hybrid solution between multi and single processor systems is a multi-core system, where multiple processor cores are integrated onto the same silicon. Multicore systems can be homogenous, or heterogeneous such as the Xilinx Zynq Ultrascale+ processor [3].

A.1.3 Hierarchical Memory organization

Very Large Scale Integrated (VLSI) Dynamic RAMs (DRAM) provide applications Gigabytes of storage at the cost of access latency to long bursts of data. Hierarchical memory organization utilizes progressively smaller but faster memories closely coupled to processing units to supply data for most transactions. In a processor core, the memory element closest to the ALU is the processor's internal register array. Operands not found in the register array are still kept close to the processor, accessible from one, or multiple levels of cache memories (L1 and L2). Caches are integrated on the same silicon as the processor and provide fast and wide access to memory. External to the processor, system (DDR or QDR) memory stores data. To increase access efficiency, DDR memory is accessed in long bursts of continuous address ranges.

Modern FPGAs also provide a diverse set of memory resources, from distributed small, but fast registers to fewer, but larger, memories in the same package. Specifically, each CLB or slice contains 16 bit Look-Up-Tables (LUT), which are integrated together with the arithmetic components (gates, adders, multipliers, etc.). 18kb (Xilinx) to 20kb (Altera) BRAM resources, with up to 36bit wide, dual-ported access to primitives provide slightly deeper memories, suitable as line buffers for video processing. In package DRAMs, often on a separate silicon die, but in the same ceramic package, provide larger buffers suitable as frame buffers for video processing algorithms.

A large FPGA device may have millions of registers and LUTs, thousands of BRAMs and dozens of eSRAM (Altera, 18Mb) UltraRAM (13.5Mb) units. Understanding bandwidth and access time limitations of these resources is key to successful implementation of high-performance ISP algorithms.

A.1.4 Reconfigurable Computing Platforms

Reconfigurable hardware devices enabled by FPGAs have opened a range of signal processing solutions between the extremes of ASSPs and ASICs. ASICs bind functions to silicon at fabrication time, resulting to chips with a single designated function. ASSPs bind functions to silicon only for the duration of a single processor cycle, limiting what the processor can accomplish in a single cycle. Reconfigurable hardware binds function to silicon dynamically within the final system, depending on the needs of the application [S27], [S28]. Binding flexibility improves resource utilization in reconfigurable hardware.

Configuration may take place at deployment time, between execution phases, or during execution. In a typical reconfigurable system, a bit stream is used to program the device at deployment time. The more flexibility a configurable system offers the greater the configuration time as more elements needed to be addressed and programmed. Therefore, coarse-grain architectures gain from potential lower energy requirements with less silicon space allocated for configurable switches and routing. Partial reconfiguration allows part of the device to be reprogrammed while other parts are still performing active computation. Partial reconfiguration also requires smaller configuration bit streams.

Modern FPGAs support partial reconfiguration of device areas, making them the enabling platform for configurable hardware acceleration [S29], [S30]

A.1.5 Basic DSP operations in FPGAs

This section reviews some of the basic arithmetic and signal processing primitives used in 1D and 2D filtering and image pre-processing functions. The optimized FFT, DDS, median-, rank-order filters are used in Chapters 6-8.

The cost of programmable resources is that the silicon footprint of a functional module in an FPGA is larger than in an ASIC. Hence propagation delays are longer, which in turn makes timing closure more challenging.

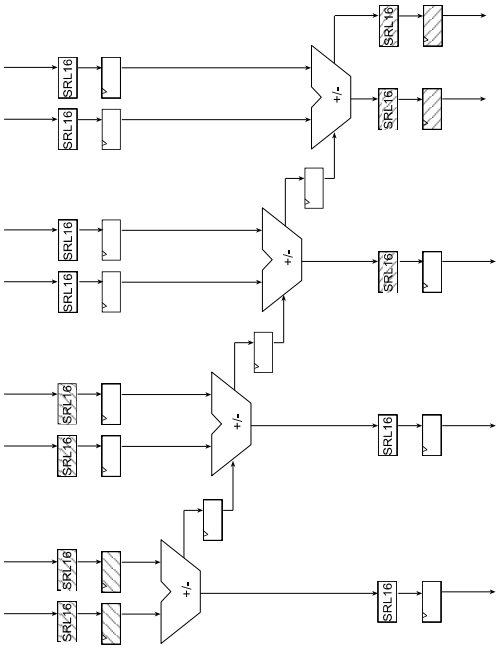


Figure 83. Pipelined adder / subtractor

Specific routing, such as dedicated carry logic, is necessary to speed up additions and subtraction operations. Long carry chains in adders/subtractors wider than 18 bits have to be registered. Figure 83 shows an example for such a pipelined structure. Modules cross-hatched left are used only if input pipelining is used. Modules cross-hatched right are used only if output pipelining is used. The delay components before and after the functional modules are configurable in depth using dedicated shift-register blocks common in most FPGA slices. In order to further accelerate wide additions and subtractions, carry-select, or carry look-ahead architectures are employed. Xtreme DSP slices also contain a configurable number of registers to facilitate pipelined daisy chaining of the modules to implement wide arithmetic operands, such as the ones used in Chapter 2.

A.1.6 Optimized rounding

In a DSP system – especially if the system contains feedback, such as the pipelined resonator described in Chapter 2 – the word length growth through arithmetic operands should be offset by quantizing the results. Quantization, or reduction in word-length, results in data loss, introduces quantization noise and may introduce bias. For best results it is favorable to select a quantization method which introduces zero mean noise and minimizes noise variance.

For truncation, the probability density function (PDF) of the noise is:

$$p(e) = \begin{cases} \frac{1}{\Delta} & -\Delta < e < 0 \\ 0 & \text{otherwise} \end{cases} \tag{106}$$

Therefore, the mean and the variance of the noise introduced are:

$$m_e = \int_{-\Delta}^{0} ep(e)de = \frac{1}{\Delta}\int_{-\Delta}^{0} ede = -\frac{\Delta}{2} \tag{107}$$

$$\sigma_e^2 = \int_{0}^{\Delta} e^2 p(e)de = \frac{1}{\Delta}\int_{0}^{\Delta} e^2 de = \frac{\Delta^2}{3} \tag{108}$$

For rounding the (PDF) of the noise is:

$$p(e) = \begin{cases} \frac{1}{\Delta} & -\Delta/2 < e < \Delta/2 \\ 0 & \text{otherwise} \end{cases} \tag{109}$$

the mean and the variance of the noise introduced are:

$$m_e = \int_{-\Delta/2}^{\Delta/2} ep(e)de = \frac{1}{\Delta}\int_{-\Delta/2}^{\Delta/2} ede = 0 \tag{110}$$

$$\sigma_e^2 = \int_{-\Delta/2}^{\Delta/2} e^2 p(e)de = \frac{1}{\Delta}\int_{-\Delta/2}^{\Delta/2} e^2 de = \frac{\Delta^2}{12} \tag{111}$$

Truncation has no cost in hardware, but it introduces significant bias. The ideal rounder should introduce no DC bias to the signal flow. To implement rounding, 0.5 has to be added to the full product, which may introduce positive bias. Therefore, 0.5 should be rounded up/down with 50% probability dithering the exact rounding threshold. To mitigate accumulation of quantization errors Xilinx DSP slices support rounding towards infinity (based on the MSB of the input), towards 0 (based on the LSB), or can use any 50% duty-cycle signal statistically independent of the rounding operand.

A.1.7 Parallel multiplication

Modern FPGAs contain 18x18 and/or 25x18 block multipliers to accelerate and reduce the footprint of DSP implementations. However, for large FIR or IIR filter banks with fixed coefficients, Distributed Arithmetic (DA), [83] may be used to trade block multipliers with slice LUT based logic. A good use case example for this optimization is the Sobel operator, with powers of two coefficients. Slice based implementation of fixed coefficient multipliers can benefit from Canonic Signed Digit (CSD) code representation of the fixed coefficient. CSD code belongs to the family of Signed Binary Number Representation (SBNR), in which the coefficient value is given as a sum of signed power-of-two terms. Without loss of generality coefficient values are assumed to be integer as fractional coefficient values can be scaled to integer values, with the resulting product scaled back respectively.

$$x = \sum_{r=0}^{B-1} s(r)2^r \tag{112}$$

where $s(r) \in \{-1,0,1\}$. CSD code contains at most $B/2$ nonzero digits, and the average number of non-zero digits are B/3. For constant, inner product multipliers CSD offers significant logic resource savings [S24]. The algorithm exploits the available digit pattern coincidences between the CSD codes of the coefficients to further reduce footprint in FPGA or ASIC implementations.

Many digital signal processing algorithms are based on calculations with complex numbers. Multiplication of complex numbers requires 4 real multiplications and two real additions/subtractions:

$$(P + jQ) = (X + jY)(x + jy) \tag{113}$$

Complex multiplication is an atomic operation in FFTs or resonator based filters.

A.1.8 Complex Multipliers

The most straightforward way to implement Givens rotation is direct implementation with complex multipliers. There are two basic architectures to implement the complex multiplication. The 4-multiplier solution (Figure 84) computes the multiplication using only DSP slices, which results to very high performance.

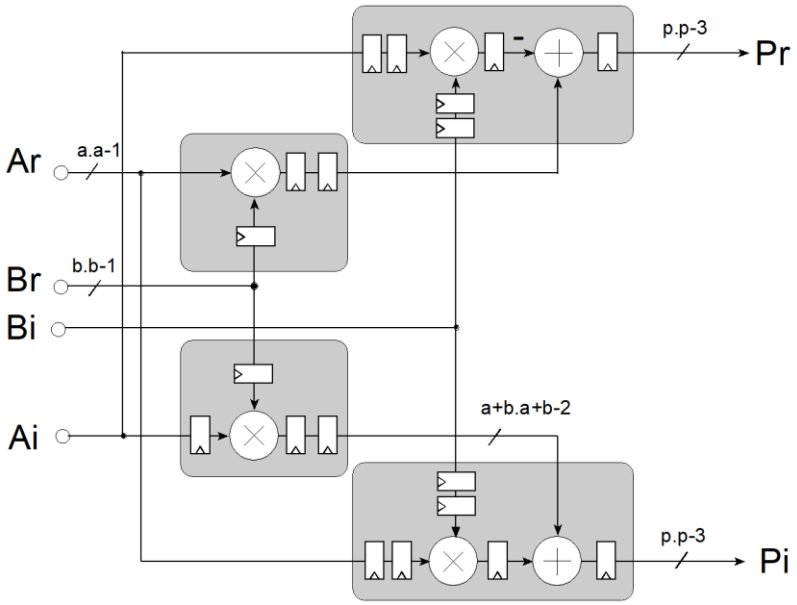


Figure 84: Complex multiplier with 4 real multipliers

The 3-multiplier solution offers a trade of saving 1,2 or 4 DSP slices for 18x18, 18x35 and 35x35 operand sizes respectively, requiring pre-combination of data. In earlier FPGA families with DSP48 primitives (Figure 85), this required using logic-fabric based resources. In recent Xilinx FPGAs the pre-adder function of DSP48A*x* and DSP48E*x* primitives can be used for the pre-combining step.

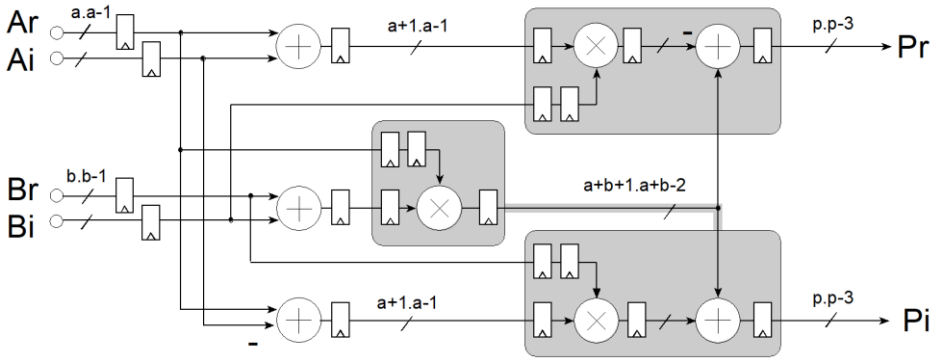


Figure 85: Complex multiplier with 3 real multipliers

A.2 Additional information on ISP module design

The following two sections provide addition information on my work on ISP modules.

A.2.1 Color Filter Array Interpolation

The simplest way to avoid the introduced zipper artifacts is to use an optical low-pass (OLP) filter as part of the lens assembly. Gratings of the OLP remove spectral components above $f_{R,B}$, in the continuous signal (light) domain before sampling by the CFA. Consumer imaging devices (cell-phones, laptops) often use small, high-resolution sensors with pixel pitches below the resolution capabilities of the compact lens assembly employed, which is another way to low-pass filter the image in the optical domain. Post-processing, aimed at the detection and suppression of the zipper artifacts can also be employed to improve image quality after the CFA interpolation module.

If the input image has not been low-pass filtered, CFA algorithms need to use heuristics to recreate information lost in the sub-sampling process. When interpolating a color component using its neighbors, the interpolation kernel can weigh constituents based on their presumed similarity of the interpolated pixel. On the boundary between two regions, a pixel needs to be combined with like color components from the region it belongs to.

Therefore, high quality Bayer CFA interpolation algorithms typically recover the green channel, or luminance image first, by spatially convolving the image with kernel

$$\boldsymbol{A} = \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}, \tag{114}$$

which approximates luminance with

$$\boldsymbol{Y} = \boldsymbol{R} + 2\boldsymbol{G} + \boldsymbol{B} \tag{115}$$

Note that the luminance image does not need to be segmented to avoid color artifacts. Most artifacts can be avoided if the color interpolation kernels are steered using edge orientation information derived from the luminance image. If the recovered edge map indicates that the local region:

- is mostly flat: interpolate with all 4 neighbors to suppress noise,
- is on a well-defined edge: steer the interpolation kernel along the edge,
- contains high frequency components with ill-defined edges: chrominance should be suppressed.

Figure 86 illustrates the typical block diagram of a CFA interpolator[23], as part of the ISP implemented in FPGA or an ASIC. From the single channel sensor stream the G channel is interpolated fist. Using the recovered G pixels and the original input pixels, the R and B channels are recovered by the same module [42].

[23] Color Filter Array Interpolation v7.0, PG002, Xilinx Inc 2015, https://docs.xilinx.com/v/u/en-US/pg002_v_cfa

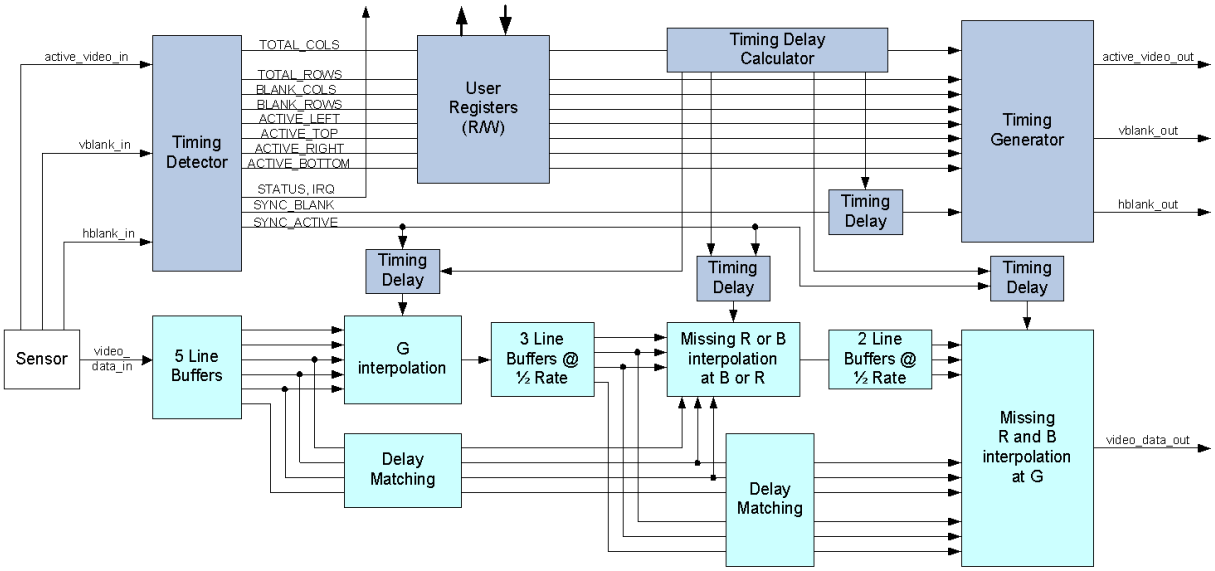


Figure 86. Block diagram of a Color Filter Array Interpolator

A.2.2 Gaussian Noise Removal

Sources of imaging noise in order of significance are shot-, electronic-, and thermal-noise. All 3 noise sources are uncorrelated and can be modelled as additive Gaussian noise. Several effective methods have been adopted widely to denoise images and video streams:

- Spatial filtering techniques, such as bilateral filtering [84],
- Temporal (motion compensated, or motion adaptive) IIR filtering [85],
- Deep learning based methods [86].

While novel FPGAs can implement any of these approaches, for streaming ISPs without access to an external frame buffer, pure spatial filters are the most efficient option. Temporal and neural approaches to noise reduction have extensive literature, but here I'd like to mention a simple and effective method for temporal noise reduction, which can be efficiently implemented in FPGAs.

A.2.3 Motion Adaptive Noise Reduction

Spatial noise reduction combines values from neighboring pixels. Temporal filtering aims to combine information from the same pixel between subsequent frames. In order to avoid combining pixels corresponding to different objects which moved from one frame to the next, temporal filtering is enabled by dense optical flow calculation, which creates one to one correspondence of pixel locations between frames. Optical flow calculation in turn requires feature detection and tracking, which are typically a computationally intensive function out of the reach of ISPs.

On stationary scenes temporal noise reduction can produce impressive results without compromising image quality. But when pixels pertinent to different objects, typically a moving foreground object and the background, are erroneously combined trailing motion artifacts are introduced. It is also important to note that the human visual system is more sensitive to noise on stationary regions of an image than on edges of moving objects. Therefore, it is feasible to construct a module, which feeds back information from the same pixel location from previous frames if motion has not been detected in the vicinity of the pixel, but limits this feedback when motion is detected.

Motion Adaptive Noise Reduction (MANR) is a recursive temporal filter with a programmable motion transfer function trading off noise suppression with trailing artifacts. Pixel variance can be

computed from frame-to-frame pixel differences, in turn calculated using the previous image buffered. The motion transfer function maps measured variance to a pixel specific feedback factor $k_n(x,y)$, such that the output pixel $r_n(x,y)$ is computed from input pixel value $s_n(x,y)$ as

$$s_n(x,y) = r_n(x,y)\big(1 - k_n(x,y)\big) + s_{n-1}(x,y)k_n(x,y), \tag{116}$$

for frame *n*, effectively defining a two-tap, adaptive low-pass IIR filter for each pixel.

Augmenting MANR and the method proposed in [S1][S2] for suppression of Gaussian noise, Chapter 6 presents my work on the field of non-linear filter implementations on FPGAs.

A.3 Review of the Fast Fourier Transform

This Appendix section introduces the Discrete, and Fast Fourier Transforms, their inverse, different and optimization options such as higher-, and split radix algorithms.

A.3.1 The Discrete Fourier Transform

The DFT $X(k), \rightleftharpoons k = 0, \dots, N-1$ of a sequence $x(n), \rightleftharpoons n = 0, \dots, N-1$ is defined as

$$X(k) = \sum_{n=0}^{N-1} x(n)\, e^{-\frac{jnk2\pi}{N}}, \qquad k = 0, \dots, N-1 \tag{117}$$

where *N*is the transform size and $j = \sqrt{-1}$.

The inverse DFT (IDFT) is

$$x(n) = \frac{1}{N}\sum_{k=0}^{N-1} X(k)\, e^{\frac{jnk2\pi}{N}}, \qquad n = 0, \dots, N-1 \tag{118}$$

Homology in the definitions of (117) and (118) allows merging them into the form

$$X[k] = \sum_{n=0}^{N-1} x[n] W_N^{kn}, \tag{119}$$

where $W_N^k = e^{\frac{2\pi jk}{N}}$ is often referred as phase-, or twiddle factors. In embedded applications the factors are often pre-computed and stored in memory. If the twiddle factors are stored in memory, the DFT or IDFT computation differs only in the twiddle addressing sequence.

A.3.2 Derivation of the FFT

Direct form computation of a single FFT bin (sample *x*(*k*), or spectrum *X*(*k*)) requires *N* multiplications and *N*-1 additions. Symmetry of the DFT calculation can be exploited to drastically speed up execution. Creating separate sums for even and odd values of *n:*

$$X[k] = \sum_{r=0}^{\left(\frac{N}{2}\right)-1} x[2r] {W_N}^{2rk} + \sum_{r=0}^{\left(\frac{N}{2}\right)-1} x[2r+1] {W_N}^{(2r+1)k}, \tag{120}$$

which, in turn can be written as

$$X[k] = \sum_{r=0}^{(N/2)-1} x[2r] W_{N/2}^{rk} + W_N^k \sum_{r=0}^{(N/2)-1} x[2r+1] W_{N/2}^{rk}, \tag{121}$$

by utilizing that $W_N^2 = e^{-j2\pi 2/N} = e^{-j2\pi/(N/2)} = W_{N/2}$, and $W_N^{2rk+1} = W_N^k W_{N/2}^{rk}$.

This result facilitates the computation of an N point DFT to the computation of two $N/2$ DFTs followed by a combination step. The decomposition and post-combination step can be used on the $N/2$ point DFTs recursively, eventually breaking down the whole structure to two-point atomic DFTs. The 2 point DFT operation is defined as:

$$\begin{aligned} X[0] &= x[0] + x[1] \\ X[1] &= x[0] + e^{-\frac{j2\pi}{2}} x[1] = x[0] - x[1] \end{aligned} \tag{122}$$

This operation is generally referred as the butterfly operation (Figure 87). It is important to note, that the butterfly operation is free of multiplications.

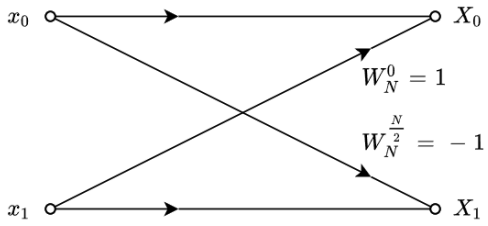


Figure 87. Butterfly operation

The optimized signal-flow graph of the 8 point FFT (Figure 88) is a radix-2 decomposition of the DFT operation, using 2 input butterflies and complex multipliers as basic elements. The graph has 3 columns, generally referred as ranks or stages, with 4 butterfly operations in each rank.

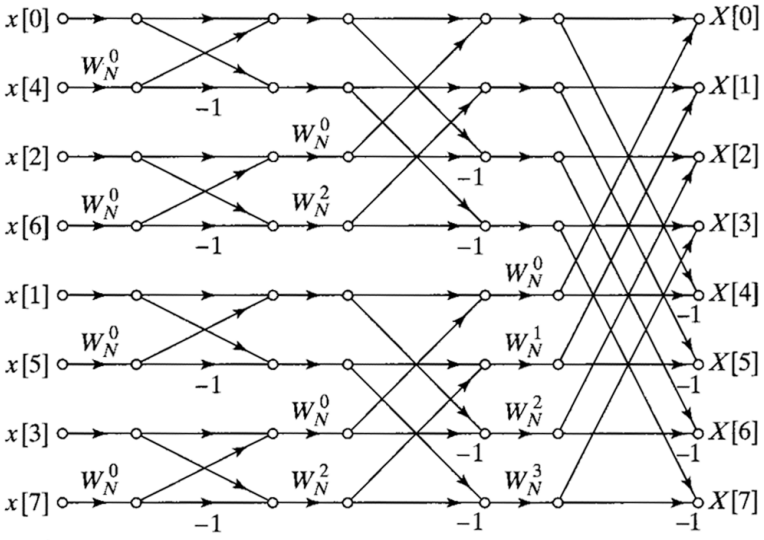


Figure 88. Eight point DIT FFT

Note that the spectra samples on the output are in natural order, but the input time-domain samples are in bit-reversed order.

A.3.3 Decimation in Time or Frequency

Figure 88 describes the signal flow graph of a Decimation In Time (DIT) algorithm for computing the FFT, proposed by Cooley and Tukey in [67]. Because of the homology of DFT and IDFT, the reversed signal flow graph (Figure 89) also performs an FFT operation. Alternatively, this approach is referred as Decimation In Frequency (DIF).

A radix-2 DIF FFT first splits the frame to an even and an odd part by a radix-2 rank, then calculates the DFT of the even and the odd halves. Conversely, a radix-2 DIT FFT performs two $N/2$ point

DFTs, then combines the results in the last rank. The approaches have the exact same complexity and resource requirements, however, there are some key differences, which may render one or the other more favorable for a particular application:

	DIT	DIF
I/O ordering	Output is in natural order	Input is in natural order
Multiplier free stage	First Stage	Last Stage
Multiplier position	Pre-butterfly	Post-butterfly

Table 20. sample ordering per FFT configuration

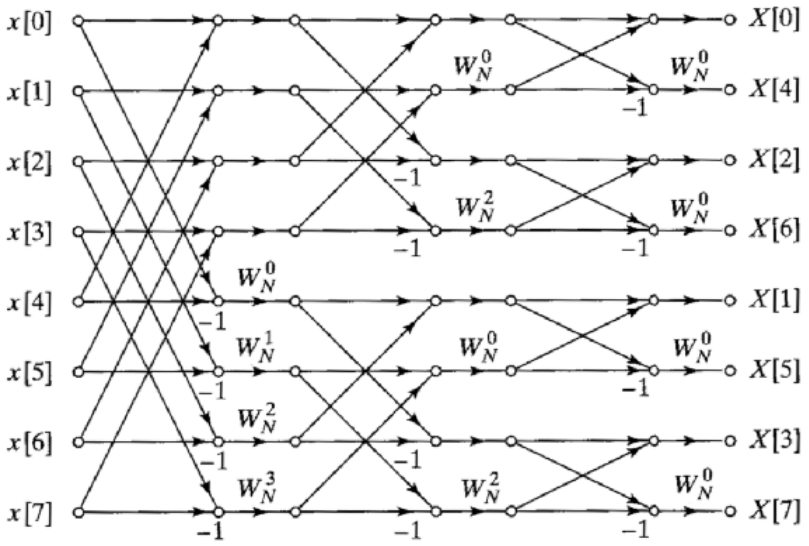


Figure 89. Eight point DIF FFT

Cyclic convolution in the frequency domain is used extensively to accelerate 1D and 2D convolutions [S21], [S19], [S22]. Cyclic convolution can be implemented using a DIF FFT and a DIT IFFT (Figure 90), without input or output reordering. However, the coefficient memory holding *H(z)* should either be re-ordered or addressed in non-natural order.

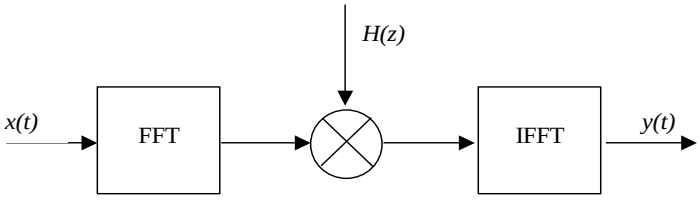


Figure 90. Cyclic Convolution in the Frequency Domain

It is important to note that each time the data passes through the adders or subtractors its dynamic range may expand. In case an FFT implementation uses dedicated multipliers with fixed operand sizes, having the multipliers in front of the butterflies may reduce quantization noise. E.g., using 16 bit input data, the DIT algorithm first does the complex multiplications (4 real multiplications) on 16 bit data, whereas it DIF was used, the multipliers should be at least 17 bits wide. In case of DIT, the subsequent additions and subtractions must be performed on wider data, however the benefits of smaller multipliers usually overweight the cost of wider adders and subtractors.

A.3.4 Alternative implementations

There are custom FFT architectures which trade the regular structures of DIT or DIF for different advantages, while using the same resources. Most notably, it is possible to construct FFT algorithms (e.g. 8 point FFT in Figure 91), in which the FFT ranks are all similar, therefore one instance can be reused several times to perform the FFT [87], [88].

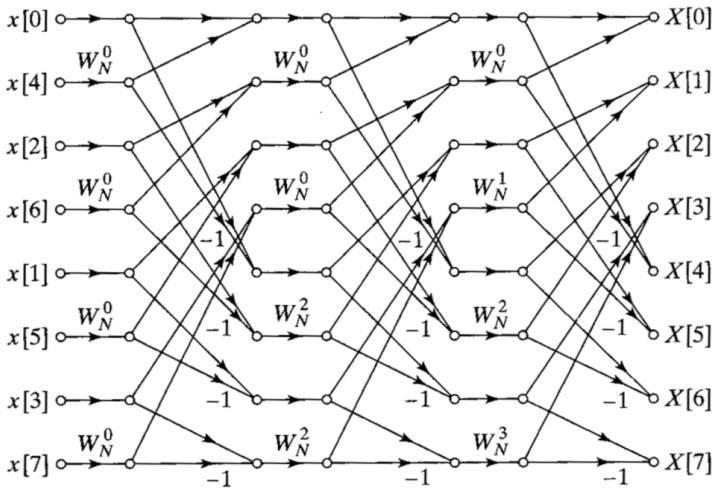


Figure 91. FFT architecture with similar stages

In a DIF radix-2 algorithm, most of the non-trivial factors are in the first stages of the algorithm. In DIT, most multiplications are in the final stages. Therefore, multipliers can be saved, if the recursive radix-2 breakdown of an $N=2^M$ point starts with a DIF, then at $M/2$ all subsequent $n=2^{M/2}$ point FFT sub-modules are broken down as DIT FFT. This approach is known as Decimation in Time and Frequency (DIFT). At the transition layer, outputs of DIF butterflies (post-multiply by W_N^k) will drive DIT butterflies (pre-multiply by W_N^l), so some of these multipliers can be merged into a single multiplier (multiplying by W_N^{k+l}, which is also a twiddle factor). In DSP processors the overhead introduced by the irregularities of twiddle factors in the transition layer may be well compensated by the multiplications saved. FPGA implementations are affected by irregular transition layer, the two different kinds of butterflies used and the extra control logic necessary for complicated twiddle address generation.

There are numerous other ways to approximate the DFT results [88]. For the computation of real-valued convolution and correlation kernels, a very important use case for image compression and processing, only the real valued DFT bins are required. For this sub-class the Discrete Sine or Cosine Transform can be used, which are analogous in FPGA implementation to the DFT.

A.3.5 Higher radixes, algorithmic complexity

In general, if the transform size N can be constructed as a product of prime factors

$$N = \prod_{k=0}^{M-1} p_k \tag{123}$$

the N point DFT operation can constructed using M stages, each having N/p_k radix p_k butterfly instances. In each radix-p butterfly, multipliers are associated with all but the first input, totaling in $\sum_{k=0}^{M-1}(p_k-1)\frac{N}{p_k}$ complex multiplications for in the FFT. In case of the $N=8=2^3$ example, 12 non-trivial complex multiplications are necessary, which is a significant saving compared to the 8*8 complex multiplications required for the direct computation of (119). Furthermore, omitting

the trivial multiplications with $W_N^0 = 1$, the total number of complex multiplications necessary for an FFT drops to:

$$\sum_{k=0}^{M-1}(p_k - 1)(\frac{N}{p_k} - 1) \approx O(N\log(N)) \tag{124}$$

The vast majority of DFTs used in signal processing applications are ones where $N=2^M$, M being a positive integer. In these cases, the FFT algorithm derived can be further simplified by merging neighboring ranks together. Instead of using two point (radix-2) FFTs as basic elements, four point (radix-4), or eight point (radix-8) dragonflies can be used. The radix-4 DIT FFT can be derived the same way as the radix-2. First, perform an N/4 point DFTs on the four quarters of $x(n)$:

$$\hat{X}(l,q) = \sum_{m=0}^{N/4-1} x[4m + l]W_{\frac{N}{4}}^{m,q}, \tag{125}$$

With $l = 0,1,2,3$ and $q = 0,1,2,\ldots,\frac{N}{4} - 1$, then do four point DFTs on the results:

$$X(\frac{N}{4}p + q) = X(p,q) = \sum_{l=0}^{3}[W_{\frac{N}{4}}^{lq}\hat{X}(l,q)]W_4^{lp} \tag{126}$$

Using a matrix representation, equation (126) can be rewritten as:

$$\begin{bmatrix} X_{0,q} \\ X_{1,q} \\ X_{2,q} \\ X_{3,q} \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -j & -1 & j \\ 1 & -1 & 1 & -1 \\ 1 & j & -1 & -j \end{bmatrix} \begin{bmatrix} W_N^0\hat{X}(0,q) \\ W_N^q\hat{X}(1,q) \\ W_N^{2q}\hat{X}(2,q) \\ W_N^{3q}\hat{X}(3,q) \end{bmatrix} \tag{127}$$

Factorizing the matrix to create sparse matrices leads to

$$\begin{bmatrix} X_{0,q} \\ X_{1,q} \\ X_{2,q} \\ X_{3,q} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & -j \\ 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & j \end{bmatrix} \begin{bmatrix} 1 & 0 & 1 & 0 \\ 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & -1 \end{bmatrix} \begin{bmatrix} W_N^0\hat{X}(0,q) \\ W_N^q\hat{X}(1,q) \\ W_N^{2q}\hat{X}(2,q) \\ W_N^{3q}\hat{X}(3,q) \end{bmatrix} \tag{128}$$

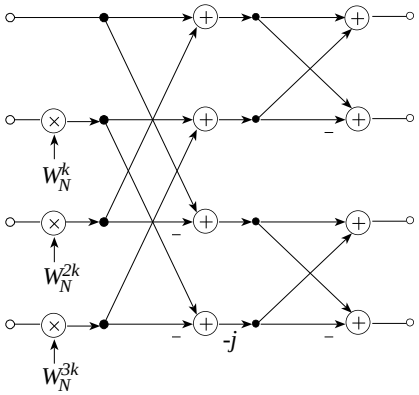


Figure 92. Radix-4 DIT dragonfly

In a practical implementation, the number of complex multiplications can be further reduced as multiplications by 0, 1, –1, *j*, and –*j* come at no cost in a parallel hardware implementation. Therefore (128) can be directly transformed to the signal flow-graph of the radix-4 DIT element, referred as a dragonfly (Figure 92).

A radix-8 FFT can similarly be derived. Apart from the seven multipliers necessary to carry out twiddle factor multiplications, the radix-8 dragonfly has two complex multipliers *within* the dragonfly structure.

It looks obvious that using a higher radix decomposition of the DFT is beneficial to minimize the silicon area allocated for block multipliers, which is typically the case in ASIC designs. However, in most FPGA designs the goal is to map the design such that the allocated slice / multiplier ratio matches that of the target device. In fact, image processing designs often deplete slice resources (registers and LUTs) before running out of block multipliers.

Also, if run-time configurable transform length with minimal control logic is necessary, a high radix FFT implementation covers fewer point-sizes. If available memories permit storing 1024 bins, a radix-2 implementation can address all powers-of-2 FFT sizes from 2 to 1024. Radix – 4 FFTs can cover only 4, 16, 64, 256, 1024, while radix-8 can cover only 8, 64 and 512.

It is important to note, that partitioning *N* along its prime factorization may not lead to the most optimal implementation either. E.g., for *N* = 64, radix-2, radix-4, and radix-8 implementations are all possible. Looking at the number of multipliers necessary on Table 21 shows that both radix-4 and radix-8 algorithms use less multipliers than radix-2.

	Number of stages	**butterflies per stage**	**multipliers within the butterfly**	**multipliers per butterfly**	**Total number of multipliers**
radix-2	$\log_2 64 = 6$	64/2 = 32	-	1	6*32*1=192
radix-4	$\log_4 64 = 3$	64/4 = 16	-	3	2*16*3=144
radix-8	$\log_8 64 = 2$	64/8 = 8	2	7	2*8*2+8*7=144

Table 21. Multiplier counts for different radixes

By taking advantage of one of the stages having only trivial pre-, or post-multiplications, the actual multiplier counts drop to 160, 96, and 88 respectively.

A.3.6 The Split-Radix algorithm

The first set of butterflies in the DIF dataflow (Figure 89) split computation of even and odd results. It is well visible here, but also noticeable after all the other stages, that there are no multipliers in the even butterfly branches. Based on this observation, one can construct a radix-4 like butterfly, comprising 3 radix-2 butterflies and 2 multipliers which carry out the multiplications required for the odd branches. In terms of arithmetic complexity, the split radix algorithm is superior to the pure radix-2, radix-4 or radix-8 FFTs, which can be exploited on DSP processors. Nevertheless, the L shaped butterfly poses timing challenges in silicon and FPGA implementation as partial results need to be propagated to match delay via the longer path; the trade-off is between control-, and data-flow complexity. E.g., for *N*=1024, the Split-Radix algorithm saves about 8% of multiplications, but pipelining the implementation is difficult. Although a loop-engine type of implementation is possible it requires a more sophisticated control block. The split-radix butterfly contains only 2 complex multipliers, a 33% savings in multiplier count, however the total transform time increases about 33% (Table 22, N/A: configuration not supported by the architecture).

N	Real Multiplications				Real Additions			
	Radix-2	Radix-4	Radix-8	Split Radix	Radix-2	Radix-4	Radix-8	Split Radix
16	24	20	N/A	20	152	148	N/A	148
32	88	N/A	N/A	68	408	N/A	N/A	388
64	264	208	204	196	1032	976	972	964
128	712	N/A	N/A	516	2504	N/A	N/A	2308
256	1800	1392	N/A	1284	5896	5488	N/A	5380
512	4360	N/A	3204	3076	13566	N/A	12420	12292
1024	10248	7856	N/A	7172	30728	28336	N/A	27652

Table 22. Comparison of computational requirements

Due to the options and factors described above, most FFT implementations are radix-2, radix-4, or split-radix. As processors are hindered less by the sophisticated control logic, split-radix is the preferred solution for DSP, GPPs or ASSPs, while most FPGA and ASIC implementations are radix-2 or radix-4 based [87].

A.3.7 FPGA based FFT implementations

This section on FPGA implementation reviews different considerations for pipelining and numerical representation, as well as basic signal processing structures used in efficient FPGA implementation of configurable FFTs.

The two obvious extremes on the performance-footprint spectrum are the full-parallel and the one butterfly implementation. In the former, a custom designed, optimized architecture is implemented. A full parallel implementation is only feasible for small point sizes. It performs an *N*-point DFT in a single clock cycle, but I/O requirements scaling linearly with operating frequency, numerical representation and transform size quickly exhaust available bandwidth at the FPGA perimeter.

The single butterfly solution on the other end of the spectrum has minimal arithmetic resource requirements, however it has significant control logic overhead (address generators), with performances below DSP processors on same technology process node.

As samples are combined in stages the width of operands grows by $r_k = log_2(R_k)$ bits. Note that for a pipelined implementation the width of arithmetic operators (adders / multipliers) can be custom tailored to the width of the data, while for the loop engine implementation operators must support the widest operand, namely the representation used for output data. Conversely, intermediate results can be quantized / rounded back to the input representation, offering a trade between quantization noise and footprint.

To explore implementation options in current FPGAs, I designed and implemented three parametrizable FPGA architectures. All architectures were targeting Xilinx FPGAs with DSP slices (built in parallel, signed integer multipliers with pre-, and post-add capabilities) adjacent to BRAMs.

A.3.8 Transform time

The transform time for an *N*-point FFT using the loop-engine is:

$$T_{FFT} = log_r(N)\frac{N}{rP}\frac{1}{f_{CLK}}, \tag{129}$$

where *r* is the common radix of the PEs, *P* is the number of PEs working in parallel, and f_{CLK} is the clock period time (assuming each PE completes a butterfly operation in 1 CLK cycle).

The transform time of the pipeline architecture is simply

$$T_{FFT} = \frac{MN}{r_{min}f_{CLK}} \tag{130}$$

Where *M* is the number of stages (Processing Engines), and r_{min} is the smallest radix used for any of the Processing Engines.

A.3.9 Scalability

Another important consideration besides pure processing time is input / output transfer time. A loop-engine type implementation uses one side of the dual ported BRAM resources to read operands, and the other is used to store results. For loop-engines the I/O phase can't overlap with the processing phase. Assuming one I/O operation, loading one complex operand and unloading one complex operand per CLK cycle, during back-to-back processing a continuous data stream for the loop-engine, the ratio of time spent processing versus moving data is defined as:

$$\frac{T_{FFT}}{T_{I/O}} = log_r(N)\frac{1}{rP}, \tag{131}$$

which means as the number of ranks increases, increasingly more time is spent processing. Conversely, for the pipeline architecture

$$\frac{T_{FFT}}{T_{I/O}} = \frac{1}{p_{min}}, \tag{132}$$

which remains the same regardless point-size changes.

If the FFT is expected to process frames with different point-sizes, unless compensated by the number of PEs in the system, T_{FFT} may grow over $T_{I/O}$, which thwarts stream data processing efforts. An FFT architecture scales well if it is relatively easy to adjust the allocated resources (mostly in PEs) to meet performance requirements and allow point-size changes. Assuming the number of PEs in a loop engine is a power of two, PEs are relatively easy to reconfigure to work together seamlessly to meet performance goals even when point-size varies.

The pipeline architecture does not support dynamic point-size changes easily: each pipeline stage is processing a certain rank of the FFT, so the number of stages must be equal to the maximum number of ranks to allow streaming data processing. If the point-size along with the number of ranks decreases, the FFT operation is performed by bypassing some of the unnecessary stages, resulting in idle silicon.

A.3.10 Memory Fragmentation

Memory resource most relevant to FFT data processing are the 36kbit (Xilinx) or 20kbit (M20K Altera) dual-ported RAMs (BRAMs) embedded into the logic fabric. As BRAMs can be read and written in the same clock cycle, they fit both the loop-engine and the pipeline architecture as buffers between ranks.

A high radix butterfly utilizes the high memory bandwidth of FPGAs, as it is accessing data from multiple BRAMs simultaneously. For example, a radix-4 PE is reading and writing 4 complex data words from 4 different BRAM instances in one CLK cycle. On the low-point size end of the spectrum this may lead to fragmentation. For a 64-point FFTs, used for OFDM extensively, a radix-2 butterfly needs 2 memories for data storage, and 1 for twiddle storage. A faster radix-8 implementation needs 8 memories for data, and 7 for twiddle storage to enable simultaneous data access, which is an inefficient use of BRAMs. In contrast with several MBs of cache on ASSPs, FPGA BRAMs are relatively shallow (18-36kbit), which may limit the maximum transform size (N) an architecture can efficiently support. On the high end, for N>4096, data buffers feeding one input of a PE are constructed out of many BRAM primitives, with only one primitive actively handling data, which is an inefficient utilization of RAM bandwidth.

Pipeline architectures can match the size of the buffer used to the rank being processed, each stage using different memory depths, and optionally, widths. These memories can be either tiled with BRAMs, or, if shallow, buffers can be constructed of distributed memory.

A.3.11 Run-time configurable transform length

As higher radix implementations are slightly more resources efficient (Table 22), designers are motivated to use higher-radix PEs. Often times an FFT accelerator is required to service any transform length $N = 2^k < N_{max}$, dynamically re-configuring the accelerator module between data frames.

Higher radix ($r > 2$) PEs can be augmented with bypass logic to service lower point sizes. To illustrate the point, consider $N = 32$, $r = 4$. A radix-4 FFT natively supports N=16, or N=64 points. The same signal flow graph can be used to perform two 32 point FFTs by bypassing one of the butterfly stages in the first of the ranks of the DIF, or last rank of the DIT FFT.

It is possible to construct a higher radix PE, which can be split into multiple smaller radix PEs, by inserting multiplexers into the data-flow graph (Figure 93). The multiplexers bypass the some of the adder network, therefore the higher radix dragonfly is split to many smaller radix dragonflies / butterflies.

While this solution has minimal logic overhead, actually impact on twiddle factor memory can be significant. Usually, unless twiddle factors are run-time computed, twiddle factor memories, adjacent to dedicated complex multiplier c ($0 < c < r$), store only W_N^{ck}, which is a significant saving compared to storing all N twiddle factors for all complex multipliers.

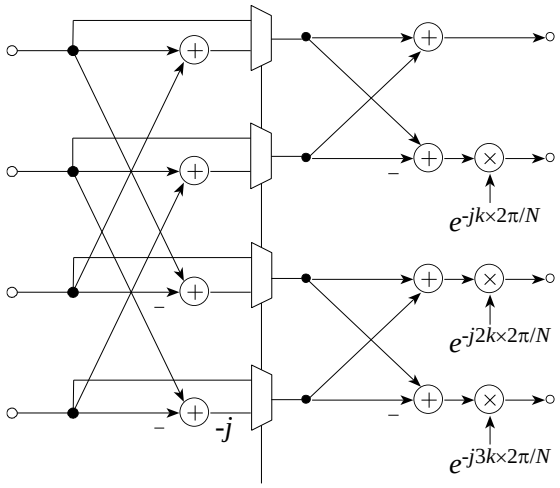


Figure 93. Radix-4 PE with bypass-option

However, if the radix-4 PE on Figure 93 works in bypassed mode – either as a radix-4, or as two radix-2 PEs – twiddle factor memories 1 and 3 must store values other than general radix-4 twiddle factors, which may result to an increase in BRAM count.

A.3.12 Support for forward and inverse transforms

There are three strategies to carry out both forward and inverse transforms using the same architecture with minimal extra hardware.

- In the direct approach, twiddle factors are conjugated when an IFFT is performed. If only one quarter of the unit circle is stored, inverters on the imaginary part are readily available.
- Alternatively, the *address* to the twiddle memory can be inverted. If *data* inverters would be more costly, for high-resolution twiddle factors with the whole unit circle is stored in a memory table, inverting the twiddle address may be a more efficient way.
- Conjugate the data. This approach is based on the simple equation of $a \cdot \bar{b} = \overline{\bar{a} \cdot b}$. As the output of one rank is the input of the next, conjugations cancel each other, this option requires inversion of the imaginary parts of the input and the output data streams.

A.3.13 Data scaling

Selecting numerical representation along the FFT affects resource usage and attainable dynamic range. Each time data passes through adders and subtractors in PEs, data range may grow over the maximum range permitted by the data format. Unless data growth is addressed by proper selection of data representation, potential overflow or saturation can degrade precision.

A strategy must be employed to accommodate this dynamic range expansion. An obvious solution is using floating-point data representation akin to general-purpose and DSP processors with built-in floating point ALUs. It is possible to implement the required floating-point operators, adders, subtractors, and multipliers in FPGAs but carefully chosen Fixed Point Representation (FPR) operators can provide similar numerical performance with significantly less resources.

To optimize data formatting at various stages along the pipe, let *B.F* designate *B* bits wide signed data with *F* fractional bits, e.g. a 16.15 notation describes 16 bit wide integers, representing values in the range [–1,1[.

Without loss of generality, assume input data to be rendered to $B_x.B_{x-1}$ format, values ranging from $x_{min}$to x_{max}. The first butterfly (Figure 87) of a radix-2 DIF FFT contains one layer of adders/subtractors and one complex multiplier. Consider the worst-case input operands, with values $x_0 = (1 + \mathrm{j})x_{max}$, $x_1 = (1 + \mathrm{j})x_{min}$ on the butterfly. Output values will be $y_0 = 0$, $y_1 = (1 + \mathrm{j})(x_{max} - x_{min})$, with y_1 as the input operand of a complex multiplier. Assuming twiddle factor $e^{-\mathrm{j}\frac{\pi}{4}}$as the other multiplicand, the result is $\sqrt{2}(x_{max} - x_{min})$, which is potentially a growth by a factor of $2\sqrt{2} \approx 2.8284$. For a radix-4 FFT, the values computed in the first butterfly rank can experience a growth to $4\sqrt{2} \approx 5.657$. To accommodate this growth and completely avoid overflows output data width should grow by 2 bits for a radix-2, and 3 bits for a radix-4 PE. In case of DIF FFTs, as dynamic range is expanded sufficiently by the first rank, for all subsequent ranks expanding the data path by 1 bit for radix-2 PEs, or 2 bits for radix-4 PEs is sufficient to avoid overflows, as proven in [89] which address this topic.

For DIT FFTs, the first rank does not contain multipliers therefore the maximum growth is 1 bit for radix-2 and 2 bits for radix-4. In the second rank though, multipliers may contribute an additional integer bit. To accommodate range expansion an additional 2 bits for radix-2, or 3 bits for radix-4 PEs is necessary. As in the DIF case, for successive ranks expansion by 1 bit for radix-2 PEs, or 2 bits for radix-4 PEs is sufficient. Total bit-growth through the FFT is $log_2(N)+1$, regardless radix or decimation scheme used.

Noise at the FFT output is a combination of:

- quantization noise present in the input,
- quantization and phase noise of the twiddle factors applied,
- quantization (rounding and/or truncation) applied by arithmetic components.

Oversizing the integer portion of the data representation at any stage does not confer benefits. Quantization noise at each rounding operator accumulates across the pipeline, which can be combated by carrying additional fractional bits. For FPGA implementations where data representation can be selected arbitrarily to balance resource footprint with performance, the goal is to find the minimum necessary integer bits to avoid over-, or under-flows, and the minimum number of fractional bits to contain quantization noise.

A.3.14 Evaluating numerical performance

Standard ways to evaluate FFT numerical performance is to model different twiddle / data representations along the data path and compare slot noise or single tone spectral results. For the slot noise test a random frame of real-, and imaginary samples are generated with Gaussian distribution. A small, continuous range of input samples are set to zero, then the frame is processed by an ideal (double-precision floating point) FFT model. Optional windowing (e.g. using a Harris window function) is applied. Then the test data frame is reverse transformed (IFFT) with both the double precision and the evaluated fixed-point model.

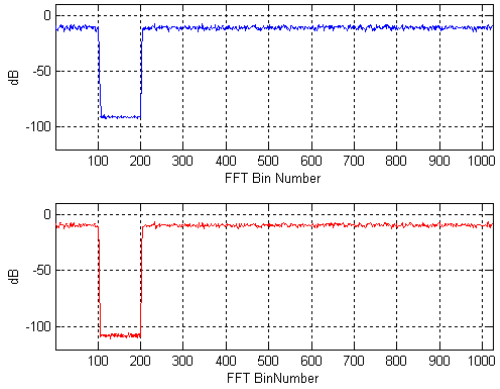


Figure 94. FFT Dynamic range (*N*=1024, blue: 18-bit fixed-point, red: double precision)

Noise level at the bottom of the resulting slot is indicative of the quantization noise characteristic to the numerical representation chosen. A single tone test is performed similarly, but with a complex sinusoid quantized to the input precision used as input to an IFFT, and dynamic range measured as peak to sidelobe ratio (PSR) on the spectral output. Figure 94 presents simulation results for an 18-bit radix-4 pipeline implementation with 18.17 twiddle factors, versus the double precision model. 18.17 representation of the input samples limit the dynamic range to about 18 * 6 = 108 dB.

As shown on Figure 95, increasing internal precision has diminishing return as output noise is lower bound by quantization noise present in input samples.

Implementing data-path components in a pipeline FFT allows customizing data and twiddle representation along the stages. In loop-engine type implementation PE(s) calculate multiple ranks, therefore data-paths and twiddle sources have to support the widest/highest fractional precision data formats expected at the last stage of the transform.

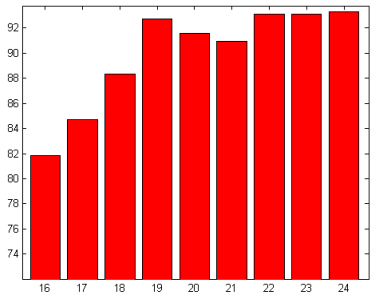


Figure 95. Dynamic range as a function of fractional bits carried

If the output width/precision is constrained, as usually output data representation matches the input, it makes little sense to carry all the bits through the transform (unscaled transform) only to drop the less significant bits at the end.

The same dynamic range can be attained with fewer resources by scaling data after each PE. It is more advantageous to use the same data-width, often dictated by the width of BRAM and block multiplier primitives, along the data-paths within the FFT and support the dynamic range constrained by the input and the output data widths (Figure 96).

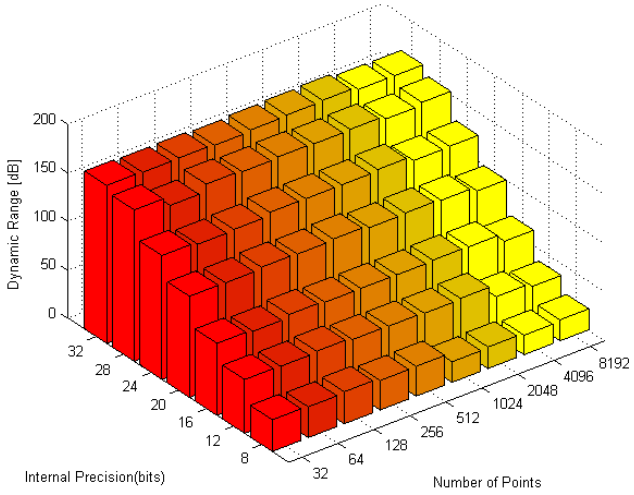


Figure 96. FPR dynamic range as a function of transform size (*N*) and internal precision

A.3.15 Block Floating point representation

Scalers in each PE can dynamically trade range with precision. E.g., if 16 bit data is processed by the first stage of an FFT, and results can be committed to 18 bit wide BRAMs, scaling can be postponed until the data frame is accessed in the subsequent frame. If the range of all input samples process indicate that no samples used the upper bits, no scaling is necessary at the next stage, preserving dynamic range. Data dependent scaling will transform the FPR to Block Floating Point representation, as the FFT output should contain not just the data but information about the binary point position, which changes every time the data is scaled. This output, which is basically the number of right shifts performed on the data – is the block exponent.

A.3.16 Twiddle factor storage, computation

Twiddle factors (W_N^k) are either pre-calculated and stored in local memories or calculated during the transform. For N<64 distributed memory can be used for storage. For $64 \le N \le 512$, all twiddle factors can be fit into a single BRAM primitive. For $512 < N$ it is favorable to store only one quarter of the complex unit circle, $W_N^k, 0 < k < N/4$, and dedicate logic resources to allow inverting and

swapping the real and imaginary components to extend twiddle factors along the full unit circle. Also, the twiddle factor LUT may contain only the real (cosine) values, as both distributed memory and BRAMs allow accessing two values simultaneously, so by appropriate inversions the complex exponential can be restored.

For *N*>1024, especially in pipeline implementations, twiddle factor storage poses potential problems as all PEs need to access twiddle factors simultaneously from multiple LUT instances. Unfortunately, at large point-sizes data storage also demands large memory buffers, rendering pipeline implementations memory heavy and incentivizing runtime calculation of twiddle factors. Twiddle factors can be calculated on the fly by using the CORDIC algorithm [81], or Direct Digital Synthesis (Chapter 8).

A.4 Additional Information for Direct Digital Synthesis

A.4.1 Alternative DDS Implementations

This section presents a summary of the operating principles and implementation footprints of published DDS solutions.

A.4.1.1 Polynomial approximation

The piecewise-linear, Taylor-series, Chebyshev, and trigonometric decomposition algorithms all fit under the umbrella of polynomial approximations. The difference is only in the method of coefficient computations.

As sin(φ) and cos(φ) are smooth, continuous, single-valued functions, the Taylor-series decomposition can be used for any $f(x)$ continuously differentiable around ϕ_k as

$$f(\varphi) = \sum_{n=0}^{N} \frac{f'^{n}(\varphi_k)}{n!}(\varphi - \varphi_k)^n = \sum_{n=0}^{N} a_{n,k}\,(\varphi - \varphi_k)^n, \tag{133}$$

Practical implementation limits the number of terms used, therefore $f(\varphi)$ can only be approximated. Also, for higher order polynomials the dynamic range of the power series and small coefficient values require wide arithmetic operands. With limited number of terms and arithmetic precision results tend to deviate as the difference between φ and φ_k increases. To limit $|\varphi - \varphi_k|$, piecewise polynomial approximation is typically used by slicing the phase domain into sections where separate, pre-computed coefficients ($a_{n,k}$) can be used for more accurate approximation.

If *N* terms are used to evaluate the sum of the products for each *f(φ)*, *N*-1 additions and multiplications have to be computed for each output sample. In case the phase domain is split into uniform sections, and the phase increment is constant the power series itself can be calculated iteratively without multiplication, using:

$$(\varphi + 1)^n = \sum_{k=0}^{n} \binom{n}{k} \varphi^k, \tag{134}$$

$(\varphi - \varphi_k + 1)^2$ can be computed iteratively from $(\varphi - \varphi_k)^2$ and $\varphi - \varphi_k$ such that

$$(\varphi + 1 - \varphi_k)^2 = (\varphi - \varphi_k)^2 + 2(\varphi - \varphi_k) + 1, \tag{135}$$

My analysis for optimizing Taylor series / polynomial approximation for FPGA implementation is focused on practical first or second order approximation structures.

A.4.1.2 Piecewise Linear interpolation

This approximation method fits line-segments on $f(\varphi)$, such that the error between the approximated values and $f(\varphi)$ are minimized. If discontinuities are tolerated, the implementation requires storage of coordinates $[\varphi, f(\varphi)]$. If the phase domain is divided into uniform sections, storage of φ can be eliminated.

In order to reach 150 dB SFDR, using uniform phase sections and double precision logic, at least 1407 sections were necessary. Using 24 bit representation, this structure needs at least 1407 x 24 bits memory, which equals to a *memory* compression ratio of 11.6 in contrast with a pure LUT based solution. Choosing the number of segments $D = 2^d, d \in \mathbb{Z}$ efficiently uses of BRAMs. However, uniform sectioning of the phase domain may result to an uneven distribution of approximation errors (Figure 97), which are proportional with the curvature of the approximated function.

Another approach is to set section boundaries such that the interpolation error remains under a predefined threshold, resulting to an approximately flat error function. By employing non-uniform quantization of φ, the number of sections could be reduced to 880. However, the LUT size requirement increases to 880·(2·24+14) bits since the quantization of φ has to be stored as well. Also, extra multipliers are necessary to evaluate the $(\varphi - \varphi_k)^n$ power series.

Using Xilinx or Lattice parts, a sine generator based on non-uniform linear interpolation allocates 5 Block-ROMs and one 24x24 bit multiplier (4 mult18x18 primitives), which is not a significant saving compared to a LUT-based implementation with coarse rotation.

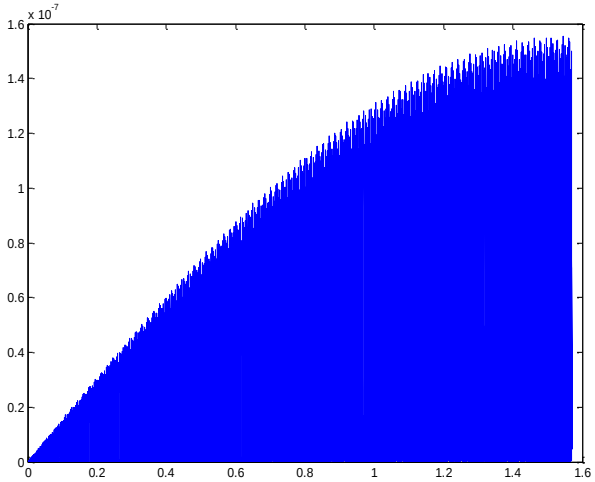


Figure 97. 1407 point linear interpolation, time domain error

The optimized linear interpolation algorithm proposed in [74] is configurable to trade segments (initialization points) with SFDR, here I consider the highest SFDR (96.2) option published, which uses only 960 ROM bits, and claims no multiplier use. Instead, slope coefficients are coarsely quantized, and numerous adders are used to replace the multipliers. Since the authors only post transistor counts, I estimate the equivalent register count at 1370 based on the author's comparison to the CORDIC implementation (8400 vs. 54000 transistors), for which the register count is known.

A.4.1.3 Angular decomposition

The angular decomposition algorithm proposed in [79] is based on the fact that for low angles β ($\beta \ll \alpha < \pi/4$) $\cos(\beta) \approx 1$, so

$$\sin(\alpha + \beta) \approx \sin(\alpha) + \sin(\beta)\cos(\alpha), \tag{136}$$

$$\cos(\alpha + \beta) \approx \cos(\alpha) - \sin(\beta)\sin(\alpha) \tag{137}$$

By approximating with $\sin(\beta) \approx \beta$, results to a variant of piece-wise linear interpolation. However, for optimal noise performance vs. QDDFS area, the authors propose to store $\sin(\alpha)$, $\cos(\alpha)$ and $\sin(\beta)$ in 3 LUT ROMs. Based on simulation results they surmise that along an SFDR target (phase resolution), overall LUT area is minimized if the total phase resolution (e.g. 14 bits), are split equally between α and β. The 3 small ROMs map well to distributed memory in FPGAs. Using two multipliers and several multiplexers, total resource use is estimated at 832+400+44 equivalent registers, considering that the 18x18 multipliers can perform two 8x8 multiplications simultaneously.

A.4.1.4 Taylor Series Decomposition

The Taylor series decomposition proposed in [70], uses 3rd, or 5th order polynomials to approximate trigonometric functions. The solution trades the number of interpolation sections, and the necessary initialization values which are stored in ROMs, with arithmetic complexity required for the evaluation of the polynomial.

Considering that $(k+1)^2 = k^2 + 2k + 1$, k^2can be calculated recursively using accumulated values of $2k+1$. For a 3rd quadratic interpolator 6 multipliers, and at least 4 wide adders evaluating the higher order polynomials are necessary, for an estimated total 6*400 + 4*24 = 2496 equivalent registers.

A.4.1.5 Coordinate rotation with CORDIC

CORDIC-based algorithms are widely used for the calculation of trigonometric functions, square root, polar-rectangular coordinate conversion. A detailed description of the algorithm and its many applications can be found in [81]. The algorithm performs vector rotations iteratively using only addition and shift operations:

$$\begin{aligned} x' &= x\cos\varphi - y\sin\varphi \\ y' &= y\cos\varphi + x\sin\varphi \end{aligned} \tag{138}$$

rotates by ϕ the two-dimensional vector (x,y) in a Cartesian plane.

By factoring out cos ϕ:

$$\begin{aligned} x' &= \cos\varphi\,(x - y\tan\varphi) \\ y' &= \cos\varphi\,(y + x\tan\varphi) \end{aligned} \tag{139}$$

If rotation angles are restricted so that $\tan(\phi) = \pm 2^{-i}$, the multiplication by the tangent term is reduced to a simple shift operation. Arbitrary angles of rotation are obtainable by performing a series of successively smaller elementary rotations. If at each iteration stage only the direction of rotation is configured, the cosϕ term becomes a constant, as $\cos(\phi) = \cos(-\phi)$. Therefore, the iterative rotation becomes:

$$\begin{aligned} x_{i+1} &= K_i\left(x_i - y_i d_i 2^{-i}\right) \\ y_{i+1} &= K_i\left(y_i + x_i d_i 2^{-i}\right), \end{aligned} \tag{140}$$

where $K_i = \cos\left(\tan^{-1}(2^{-i})\right) = \sqrt{\frac{1}{1+2^{-2i}}}$, and $d_i = \pm 1$.

Removing the scale factor from the iterative equations yields a shift and add algorithm for vector rotation. The product of the K_i:

$$A_n = \prod_{i=1}^{n} K_i, \tag{141}$$

can be applied elsewhere in the system. If the CORDIC operates with a fixed iteration length n, and is used solely for rotating the same initial vector (x_0, y_0), the scalar A_n can be incorporated into x_0 and y_0. For twiddle factor generation, x_0 and y_0 are chosen as ($1/A_n$,0), therefore all rotated results will have a length of 1.

SFDR [dB] of the CORDIC can be approximated with $6n$, where n is the number of iteration stages. The width of x,y and z accumulators (or adders in a pipelined implementation) and the number of iterations necessary for a particular angular resolution are closely related and proportional. The angle (z) is refined in each iteration step as well as x and y, therefore the angular (input) and the data resolutions are closely coupled. Choosing the adder-width equal to the number of iterations.

In a fully parallel, high speed implementation with 150 dB SFDR 25 layers of adders/subtractors are needed, totaling $2 \cdot 225^2$ adders, as for each iteration step a new layer of adders is needed. Besides large slice count, the large number of stages result to substantial latency between start of a frame to twiddle factor availability, necessitating deep data buffers to match the data flow to the twiddle factors in pipelined FFT implementations. Nevertheless, both issues are addressed in [82]. Instead of using a constant initial vector ($1/A_n$, 0), Janiszewski et.al. sliced the phase domain to multiple regions, each having different initial vectors. Initial x_0, y_0 and z_0 of each stage is stored in LUTs. This solution replaces the first adder stages with LUTs which may be implemented as block-RAMs, effectively trading off slices with block-RAMs. The tradeoff helps balancing slice-count with block-RAM usage in FPGAs and reduces the overall latency of the solution. For all algorithms observed before, the Janiszewski solution of splitting the phase to amplitude converter into two or more cascaded sections may reduce the overall cost. The first module supplies coarse values, while subsequent module(s) refine / approximate values in-between the widely spaced values. The goal is to maximize compression factor and minimize spurious content and overall resource usage.

A.4.2 Optimizing the resonator FPGA implementation

This section provides additional considerations and analysis of optimization options for implementing the Direct Form 2 IIR filter on programmable logic. Architectural solutions are proposed to improve the tradeoff between design size and output quality.

A.4.2.1 Optimizing multiplier use

Assuming $\omega_0 = \frac{2\pi}{N}, N = 2^n, n > 14$, the value of $k = cos\,\omega_0 \approx 1$. Introducing

$$k' = 1 - cos\,\omega_0 = 1 - k \tag{142}$$

allows replacing the multiplication xk by x-xk', where $k' \ll 1$ can be exploited. Knowing that the t top (MSB) bits of k' are all zeros, it is sufficient to multiply by only the portion of k' which is under the first non-zero bit, thus saving $B_s = -\lceil \log_2(k') \rceil$ bits. Assuming B bits wide data-paths through the resonator, using the above simplification will result to a B by B_k multiplier, where B_k is the number of bits representing k', and with the signed product $B + B_k - 1$ bits wide (Figure 98)

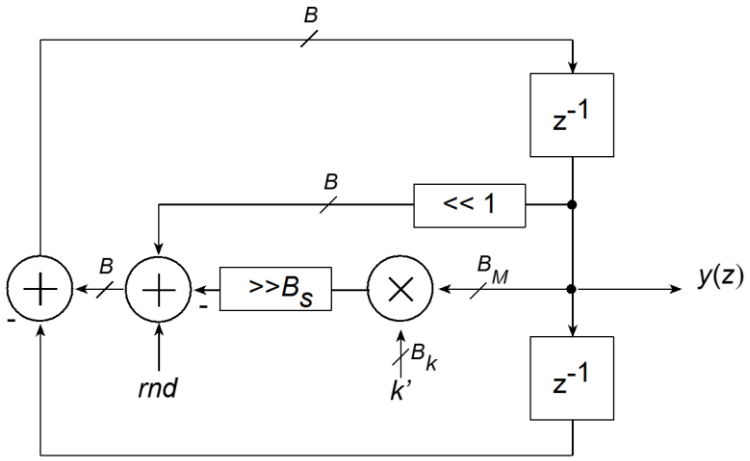


Figure 98. Constrained Multiplier

Operand sizes can be further reduced without impacting overall precision as long as the product width is significantly larger than B, since the product needs to be quantized back to B bits before being written back to the registers. Simulation results indicated that in order to reach the 150dB clarity goal (~24-bit precision), at least $B = 24 + \log_2(N)$ bits were necessary. FPGA devices have dedicated 18x18 or 18x25 bit multiplier blocks, which can be combined to build wider pipelined multipliers. The digital resonator implemented uses $B_M = 35$ bits for representing x and B_k=18 bits representing k' (the 18 bits under the first non-zero bit in k), allocating only two mult18x18 primitives (Figure 99). The 35 bit wide results of 18x18 primitives, or the 53 bit results of the 35x18 multipliers have to be rounded before being added to the B bit wide partial results. This operation is facilitated by the addition of a rounding constant $rnd = 2^{-B+1}-1$ value to the multiplier result. With the addition of the sign of the multiplier result through the carry-input of the adder, the round towards zero method can be implemented, which does not introduce DC bias and ensures filter stability.

A.4.2.2 Polyphase implementation

To ensure high operating frequencies the design has to be pipelined. However, pipelining changes the transfer function of the filter. Pipelining the arithmetic components and latching data only when valid data is present at the output of the last arithmetic component allows higher clock frequencies, but it does not increase the overall output data rate. The easiest way to overcome this problem is by using a polyphase filter, as if the resonator structure was evaluating p independent filters with the same transfer function. By initializing all registers involved in pipelining with valid partial results pertinent to the first p operands, each sub-band filter produces a new sample in p clock cycles, while the overall polyphase structure produces a valid output for every clock cycle.

In the pipelined solution, the z^{-1} registers have to be replaced by p deep parallel shift-registers (Figure 99), so output samples can be collated to form a continuous output stream. This concept helps determining the initial states of pipe registers.

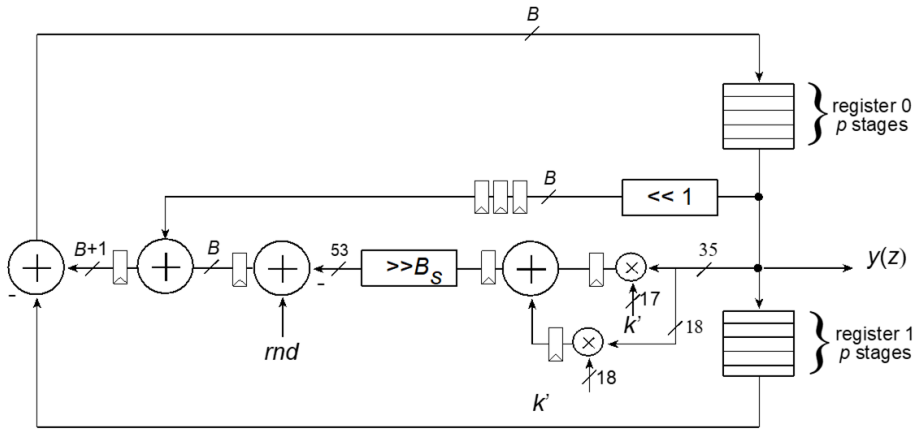


Figure 99. Pipelined resonator using block multipliers

Assuming p pipeline stages, two sets of p initialization coefficients needed, describing p parallel filters, each having the same poles (defined through k'), but different initial phases.

By pipelining the adder / subtractor components (Figure 83) in order to reduce the carry chain lengths, further improvements can be achieved in system performance. By employing maximum pipelining in all arithmetic components, the closed loop with a 35x18 multiplier has an overall latency of 8 cycles, easily attaining CLK rates over 300 MHz in Virtex4 devices, 400 MHz in Virtex7 devices, and 500MHz in Virtex Ultrascale devices.

Whether pipeline stages are concentrated in shift registers or distributed in registers next to the arithmetic components, the system needs to be properly initialized to synchronize the p phase-shifted filters. Once the initial coefficient vectors for registers 0 and 1 (Figure 99) are determined, it is easy to find the distributed coefficients by simply propagating the concentrated coefficients through the pipe-stages until the corresponding register location is reached. For a pipelined resonator generating the sine component of the quadrature signals, the initialization vector for register 0 is:

$$\sin(\omega l), \tag{143}$$

and for register 1 is:

$$2^{B-3}[0..0, \{2\,sin(\,\omega l)\,cos(\,p\omega) - sin(\,(l+p)\omega)\}], \tag{144}$$

where l is a vector of integers from 0 to p-1. The initialization vector of register 1 has to be prefixed with zeros to ensure proper alignment of signal flows at the last adder.

A.4.2.3 Quantization

Determining the proper word-length at different points of the data-path sets the balance between resource utilization and output quality. This section focuses on the relations between the output error, and different design parameters. I analyzed the effects of the following parameters to the fidelity and noise level of the generated sinusoidal:

- precision (representation) of k
- word-length throughout the design
- run-length of the feed-back system
- rounding / truncation method

The resonator architecture was implemented using Xilinx System Generator for DSP version 6.2, which supports rapid prototyping [S31] and easy analysis of performance and resource usage [S32]

in the MATLAB environment. Coefficient-, and product quantization are the primary drivers of quantization noise. For block-multiplier implementation, where either 18x18 or a 35x18 primitives were used, multiplier operand and the product sizes are fixed.

The second largest noise-source in the filter is the quantization to B bits after the $kx = 2(1-k')x = 2x - 2k'x$ subtraction. Figure 100 and Figure 101 present the SFDR and SNR values corresponding to different data path widths (B), for $N=2^{16}$. These simulation results indicate that increasing fix-point resolution does help lowering noise, therefore improving spectral purity, until quantization noise introduced by the multiplier becomes the dominant noise source.

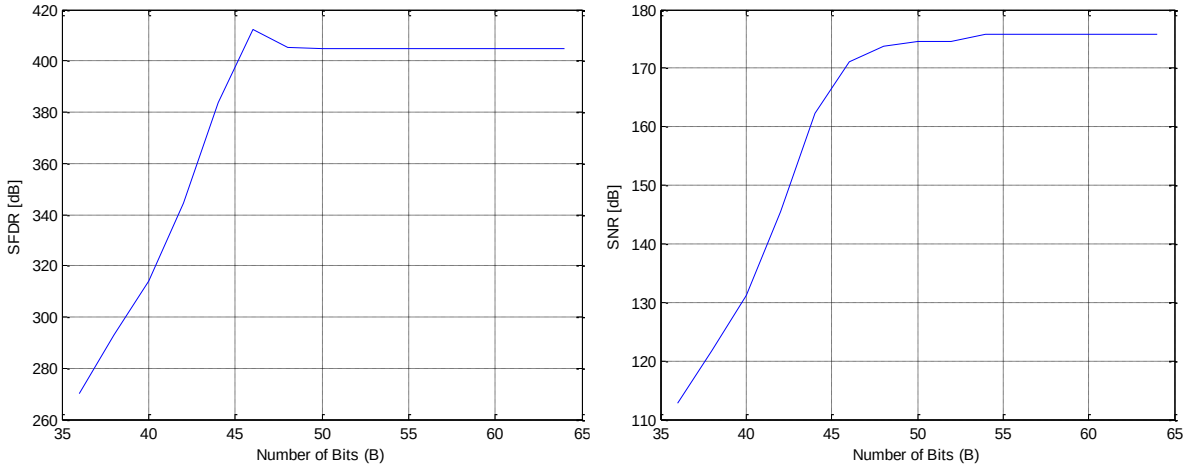


Figure 100. SFDR (left) and SNR (right) as a function of B, 35x18 bit multiplier

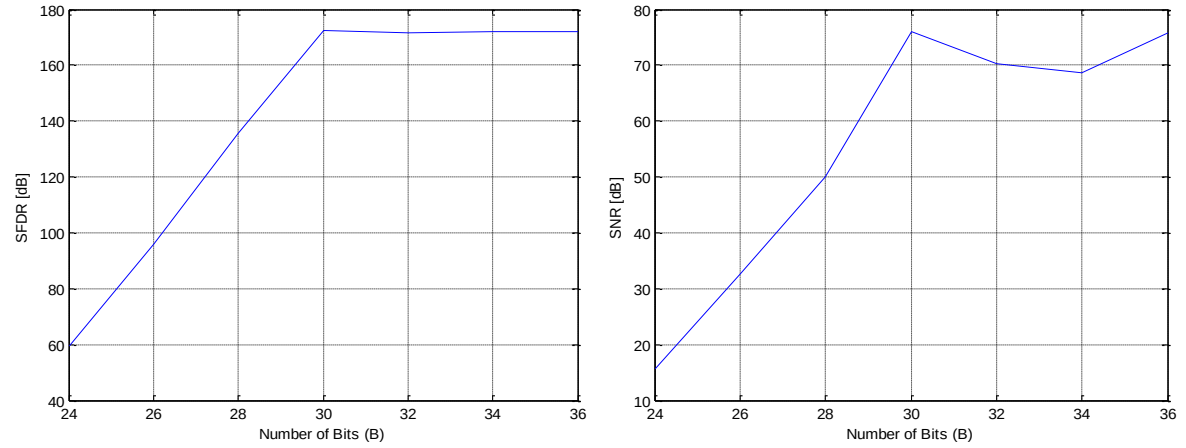


Figure 101. SFDR (left) and SNR (right) as a function of B, 18x18 bit multiplier

A.4.2.4 Frequency dithering

As the block multiplier input restricts k' to 18 bits, the excessive quantization of k' may introduce a frequency shift, manifesting in accumulated phase error. Instead of allocating more bits for k', and therefore doubling the size of the multiplier, one effective way to fine-tune the output frequency is to dither the constant multiplicand k'. This introduces frequency modulation, which in an FFT may cause signal power to leak to adjacent bins. Whether frequency dithering can be used or not depends on the target application; if phase noise around the target frequency is more acceptable than a constant frequency error, then dithering can be used to gain additional SFDR. Figure 102 illustrate the effects of frequency dithering, with parameters $N=2^{16}$ and $B=36$. Both SFDR and SNR measurements improved from 237.98 and 100.01 dB to 270.34 and 112.73 dB respectively. The

graphs on Figure 102 present the time-domain errors of the polyphase interpolator phases, with corresponding spectra on Figure 103.

While peak to sidelobe ratio is mostly unaffected by frequency dithering, maximal time domain error is reduced by a factor of eight, and correspondingly the average noise floor is about 48dB below the 18x35 multiplier solution without dithering.

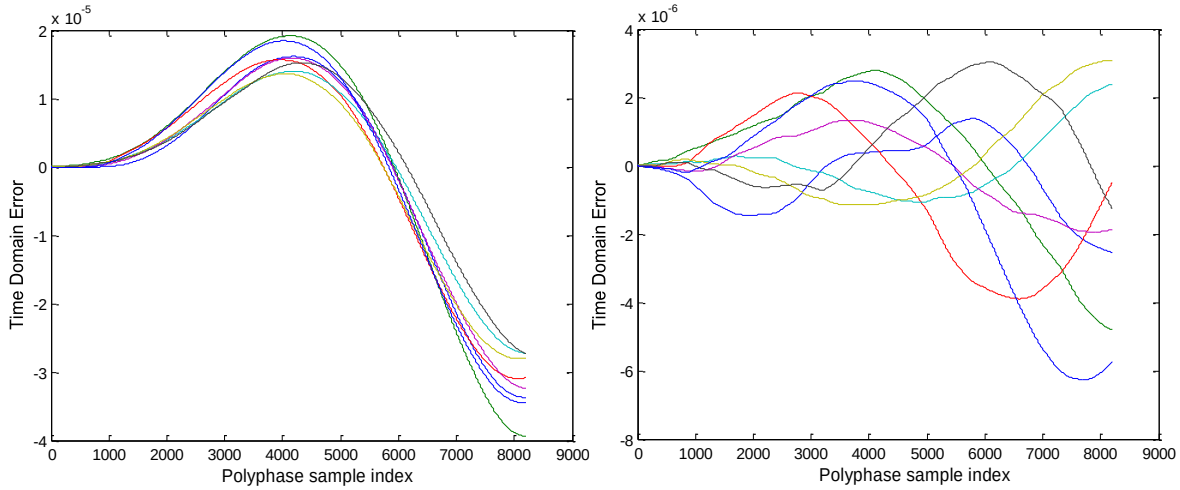


Figure 102. Time Domain error without (left) and with (right) frequency dithering, $N = 2^{16}, p = 8$

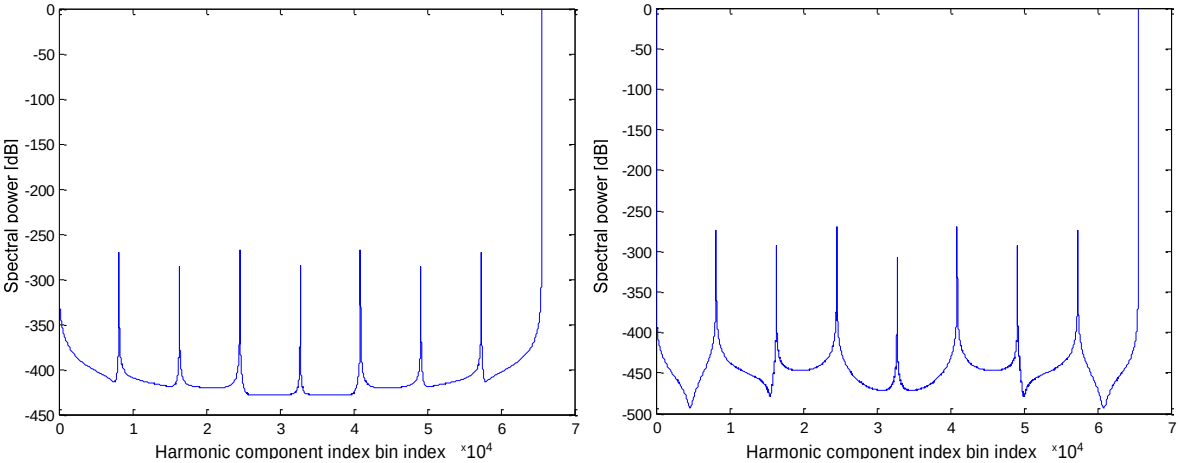


Figure 103. Output spectrum [dB] without (left) and with (right) frequency dithering

As discussed in the pipelining section, the pipelined resonator is evaluating p, phase-shifted filters simultaneously. To shape the phase noise added during frequency dithering, a simple ΔΣ converter was used.

A.4.2.5 Run-length reduction

Due to accumulation of quantization noise output values deviate increasingly from the ideal waveform (Figure 102). If the design goal is to keep spurious content under a predetermined level, the free-run length of the resonator can be adjusted by periodically reinitializing the resonator (Figure 104), trading initialization LUT size with data width. The more often the resonator is reinitialized, the more memory is necessary to store initial values, but the less precise the arithmetic operators can get. Block-ROMs and multipliers in Xilinx FPGAs support word lengths of 18 and 36 bits, so simulations were performed with B=36. In this configuration, 1 BRAM can store 512 words. $2p$ words are necessary to initialize the resonator for each phase, allowing 256 sets per BRAMs used. Figure 105 presents the relation between SFDR, SNR and the period length. The local maxima

on Figure 106 suggests that aggregate noise has at least two components, one increasing, and one decreasing with run-length (N). Obviously with longer run lengths noise accumulates, and SNR and SFDR is expected to fall. However, ω, k and k' are all functions on N. For shorter run-lengths ($N<2^{11}$), k' is larger in absolute value.

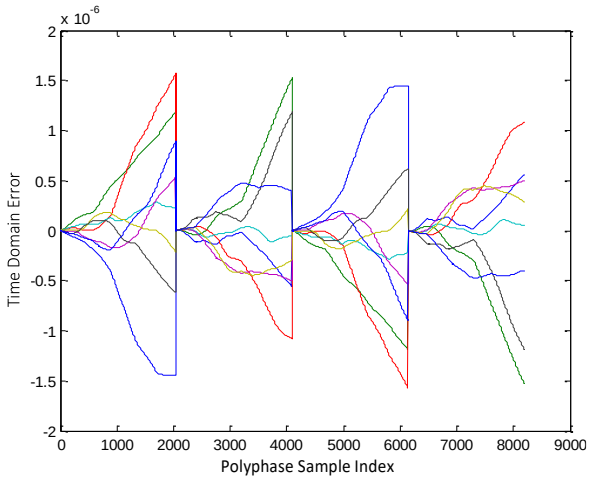


Figure 104. Noise reduction by re-initialization

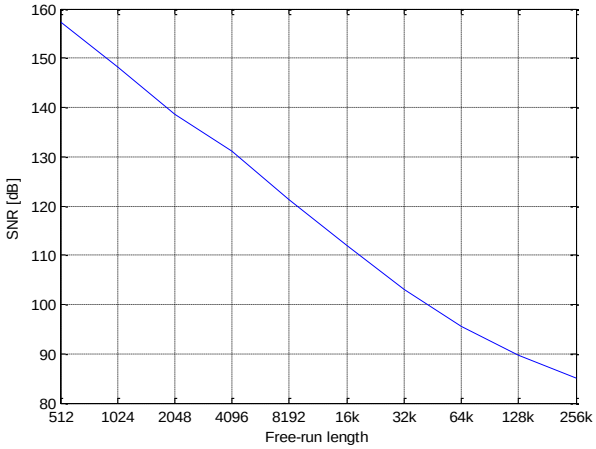
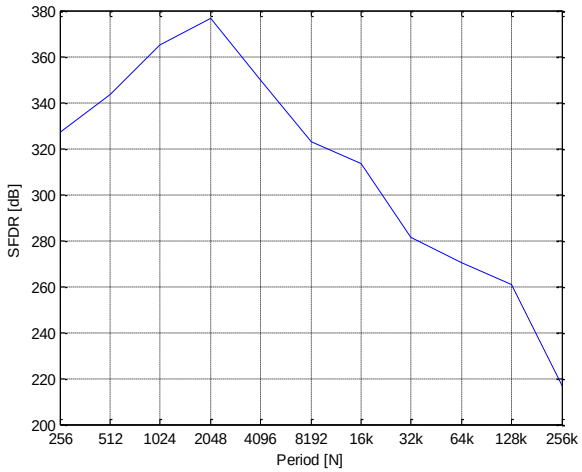


Figure 105. SNR as a function of free-run length



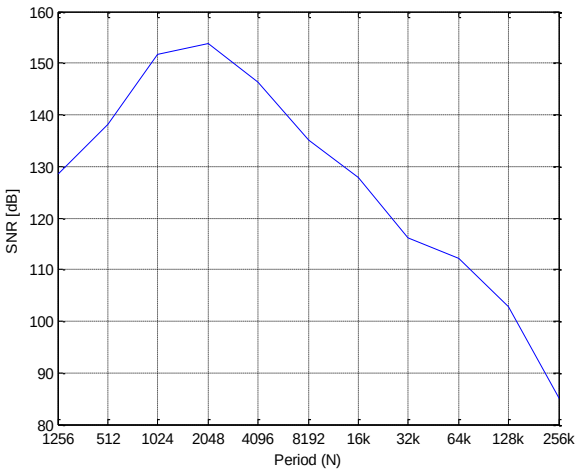


Figure 106. SFDR (left) and SNR (right) as a function of N (35x18 bit multiplier)

Since k' is represented in either 18 or 36 bit fixed-point format, quantization of k' is the noise source responsible for increasing noise as N is decreased, with the aggregate noise reaching an optimum at $N=2^{11}$. The resonator architecture with re-initialization is presented on Figure 107.

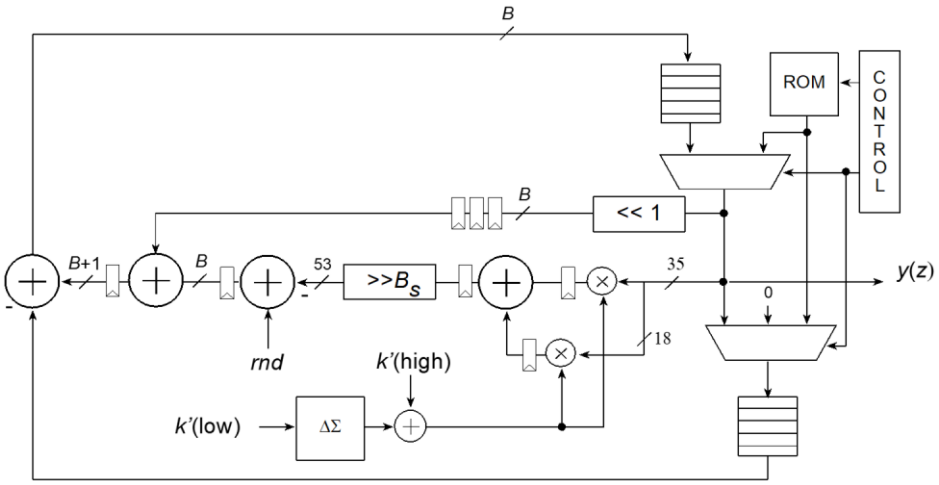


Figure 107. Resonator structure with re-initialization support

A.4.2.6 Error feedback to the registers

Since the resonator is a deterministic structure, error accumulation is predictable at design time by simulation. The LUT size can be further reduced, or the number of initializations along the SFDR can be increased if instead of complete re-initialization of B bits wide operands, only the difference between ideal output values and actual resonator outputs are stored. Errors between ideal and actual values after P sample long free-running periods ($p << P < N$) are saved to the LUT (ROM). Re-initialization multiplexers are replaced by adders in the resonator structure, to correct for the accumulated error, returning the filter to a known-good state. The reduction in ROM size is due to the compressed dynamic range as the error is typically several orders of magnitude smaller than the signal.

In FPGAs this reduction can be used to select a different BRAM aspect ratio (1024 x 18 bits vs 512x36) and use the increased depth to reduce free-run length P, in turn resulting to improved SNR and SFDR. In an ASIC, the reduced word-length can save silicon area.

The ROM could be utilized even better with deeper quantization of the error words. Using the Block ROMs in 16k x 1 bit mode, data can be corrected towards the ideal value by a fixed constant $\pm\varepsilon$ error every N/16384 CLK cycles, which substantially reduces the length of the free-running periods. In a typical case, N=2^{18}, p=8, each sub-process is corrected in every P=128 CLK cycles. Figure 108. presents the regulated noise. The significant, two orders of magnitude drop in time domain error is clearly visible in comparison with Figure 104. Since exhaustive simulations for all possible (2^{16384}) combinations of error ROM contents is not feasible, I developed a proxy algorithm to determine correction coefficients. As shown on Figure 104, accumulation of quantization-, and approximation errors for each sub-process can be thought of as a random walk away from zero during the free running periods. Each adjustment by $\pm\varepsilon$ to the register banks is aimed to correct the sub-sampled discrete differential ($f'(\varphi)$) such that the resonator output error is minimized. Absolute output error is affected by all previous corrections. The goal is to provide an algorithm to minimize the squared error of the output error functions $f(\varphi)$.

Assuming a decision-depth of D, the following backtracking algorithm can deliver adequate results:

```
resonator_state = initialize_resonator_register_banks(SIN_OR_COS, reset)
for (n=0, n<N, n+= 1) :
    err_min, resonator_state, c = iterate(resonator_state, d=0, corr=0)
    ROM[n] = c

iterate(resonator_state, d, corr):
    resonator_state.output += corr
    resonator_state_next, cum_sqr_err = simulate_resonator(resonator_state, cycles=P)
    if (d<D) :
        err_min_0, resonator_state_0, c_0 = iterate(resonator_state_next, d+1, corr = -ε)
        err_min_1, resonator_state_1, c_1 = iterate(resonator_state_next, d+1, corr = ε)
        if (err_min_0 < err_min_1):
            return resonator_state_0, cum_sqr_err + err_min_0, 0
        else:
            return resonator_state_1, cum_sqr_err + err_min_1, 1
    return  resonator_state_next, cum_sqr_err, 0
```

To establish which direction the resonator output need to be corrected, the resonator is run for $2^D \cdot P$ cycles, exploring the D bit binary combinations for each sub-process. Then for each 2^D combination, the mean-square error of the generated $2^D \cdot P$ data series is calculated, and the correction direction for the best (lowest-error) combination is selected. The overall algorithmic complexity is $N \cdot 2^D$ steps.

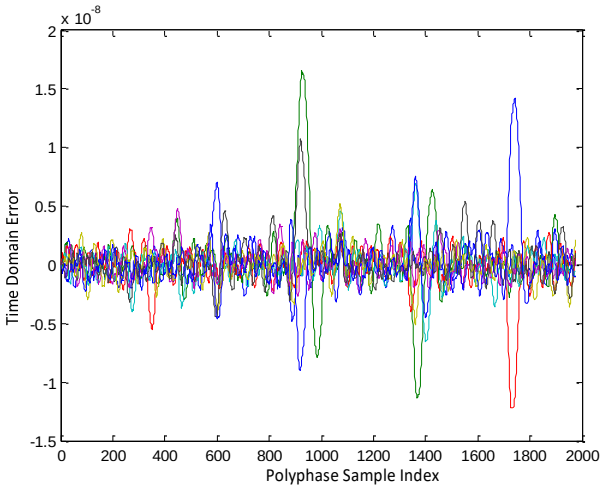


Figure 108. Amplitude errors of p=8 phases

The time-domain noise on Figure 108 shows the results of the algorithm, with 1 bit error feedback to registers, and D=1. Applying this simple algorithm, for parameters N=2^{16}, B=36, p=8, SFDR = 441.124 and SNR = 174.22 was achieved.

A.4.2.7 Error feedback to the ΔΣ converter - Results

If abrupt jumps by correction factors ±ε are unacceptable, feedback can be applied using a ΔΣ converter (Figure 109), resulting to smooth corrections. Feedback can be incorporated either to z^{-1} stages (time domain) or to the phase increment of the resonators (frequency domain). This method also requires a 1 bit only ROM.

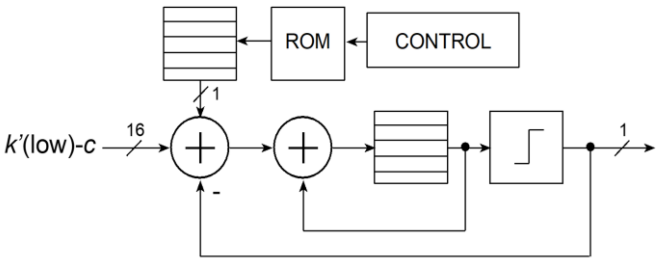


Figure 109. Modified ΔΣ converter

Results of this method for parameters N=2^{16}, B=36, p=8, with 1 bit error feedback to the ΔΣ converter are illustrated on Figure 110.

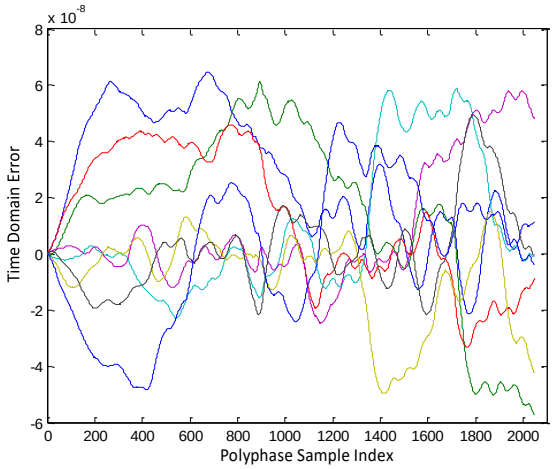


Figure 110. Amplitude errors of p=8 sub-processes

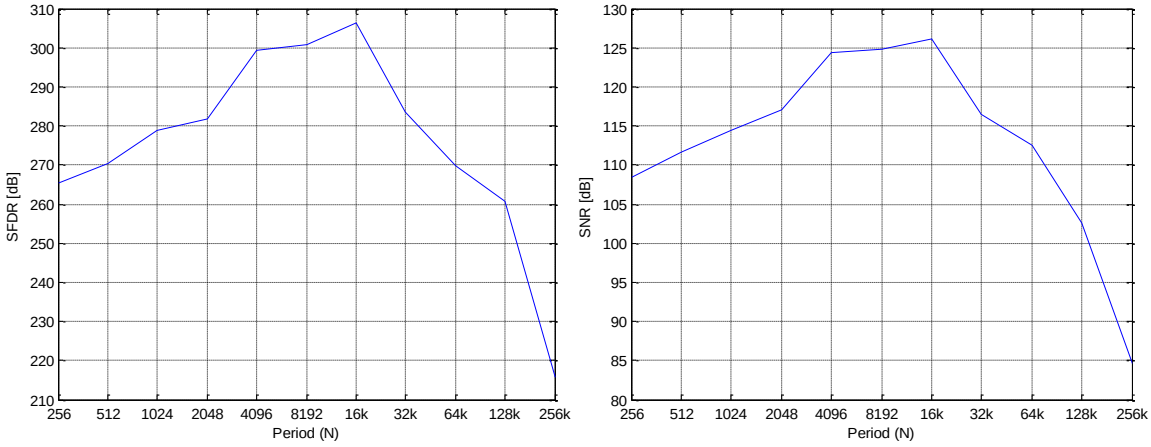


Figure 111. SFDR (left) and SNR (right) as a function of N, 18x18 bit multiplier

If the target design is multiplier heavy, as in the case of FFTs with wide complex multipliers, and sliced based logic is available, the block-multipliers can be replaced by pipelined CSD adders/subtractors. The CSD multiplier based resonator structure can be compared directly against a CORDIC implementation, as both architectures are built out of pipelined adders. The advantage of CSD representation is at least 50% reduction in adder/subtractor count, which allows a B x B_k bits wide multiplier to be implemented using ½ B B_k adders. **Maintaining the same output quality, the resonator structure uses only about 25% of the resources of the CORDIC algorithm, where B^2 full-adders are needed.**

A.4.3 Implementation of Quadratic Interpolator

This section provides additional information on optimization options and FPGA implementation results for continuous quadratic interpolators.

A.4.3.1 Dynamic range considerations

It is important to take range changes into consideration as data passes through the differentiator – integrator stages. The range of the discrete differentials of a sinusoid is less than the range of the stored sinusoids, since

$$\frac{d}{dx}[A\sin(\omega x+\phi_0)] = A\omega\, cos(\omega x+\phi_0). \tag{145}$$

where $\omega = \frac{\pi}{2^{L+1}} \ll 1$, $x < 2^L, x \in \mathbb{Z}$. Range is compressed by

$$B_l = -\left\lfloor log_2\left(\frac{\pi}{2^{L+1}}\right)\right\rfloor \tag{146}$$

bits, however additional fractional bits to represent the discrete differentials are necessary to reduce quantization error accumulation in the subsequent integrator stage – which is best expressed as a binary point shift by B_l bits to the left. On Figure 112 shift right operations indicate that the binary point is shifted left, implementing the constant multiplication, while maximizing the available dynamic range. After the first integrator data have to be scaled back by B_r bits to avoid arithmetic overflows and to ensure correctly scaled output values.

If the twiddle factor generator DDS has to generate W_N^{nk} for all $k = 2^\kappa$, $0 < \kappa \leq \log_r N$ instead of storing pre-calculated discrete differentials, samples of $f(\varphi)$ need to be stored.

If samples of $f(\varphi)$ are stored in BRAM, the number of interpolation periods may be determined by the efficient configuration of the BRAM. Assuming the design goal is to maximize compression / minimize BRAM use, the widest (36 bits) block-RAM configuration supported by Xilinx devices is suggested, allowing 512 words per primitive.

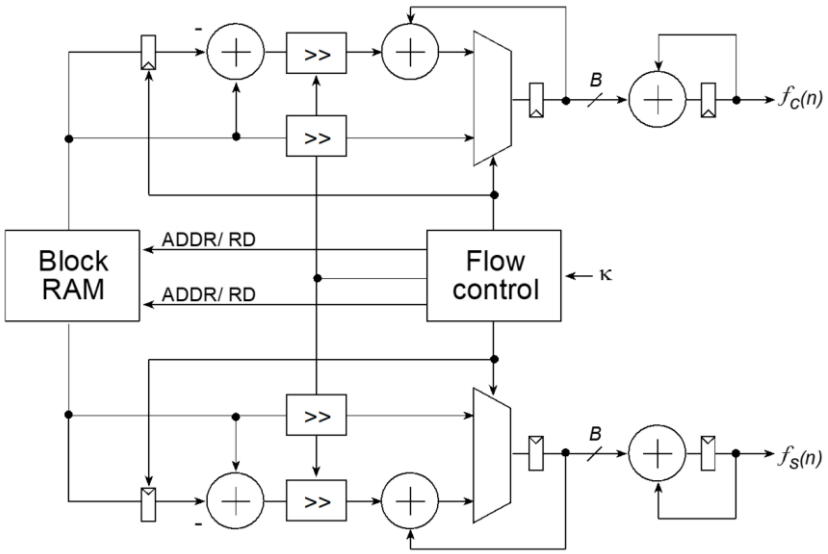


Figure 112. Phase factor source for in-place FFTs based on DII structure

A.4.3.2 Quadratic Interpolation Results

Both integrators and the BRAM contents have to be initialized correctly to get optimal results from the interpolator.

Assuming the expected waveform is

$$f_n = A\,sin\left(\frac{2\pi n}{N} + \phi_0\right), \tag{147}$$

to ensure that the first two output values are f_0 and f_1, the initial contents of the output integrator can be chosen as

$$f_0 = A\,sin(\phi_0) + \varepsilon_0, \tag{148}$$

and the initial contents of the first integrator can be chosen as

$$f_1 - f_0 = A\left[sin\left(\phi_0 + \frac{2\pi}{N}\right) - sin(\phi_0)\right] + \varepsilon_1, \tag{149}$$

where ε values provide extra freedom to fine-tune the output in order to reduce noise. The contents of the LUT should be a sinusoid with amplitude

$$A\frac{2\pi}{M}(1 + \varepsilon_A), \tag{150}$$

and initial phase

$$\frac{2\pi}{M} + \frac{\pi}{2}c + \varepsilon_\phi, \tag{151}$$

As opposed to ε_0 and ε_1 which can be used to suppress DC bias and have relatively small effects on the output noise spectrum, amplitude, and phase errors ε_A and ε_ϕ as small as 10^{-6} affect output noise by as much as 40dB.

The errors of the first DI stage are accumulated by the second integrator. If the error of the DI stage has the same sign over half the sine period, the integrated output is skewed. Tuning amplitude by ε_A allows the piece-wise linear first order derivative to extend over the expected bounding curve, significantly reducing noise accumulation.

To find optimal values for ε_A, ε_φ ε_0 and ε_1, first the amplitude and phase need to be tuned by simulations. Then ε_1 needs to be found so the derivative transitions smoothly after the first multiplexer loads the first integrator stage. Finally, ε_0 needs to be set to cancel out the mean error. The optimization problem was tackled iteratively, one variable at a time, with the Newton-Raphson algorithm, for *N*= 2048, 4096 and 8192.

Based on simulation results, ε_φ can be approximated by the linear function

$$\varepsilon_\phi(N) = \frac{\pi}{N}. \tag{152}$$

To minimize noise, ε_A can be approximated by a hyperbolic function

$$\varepsilon_A(N) = c_0 - \frac{c_1}{x - c_2}, \tag{153}$$

where c_0= 3.1892673178E-6, c_1= 1.0971156E-3, and c_2= 1.1802123120E3.

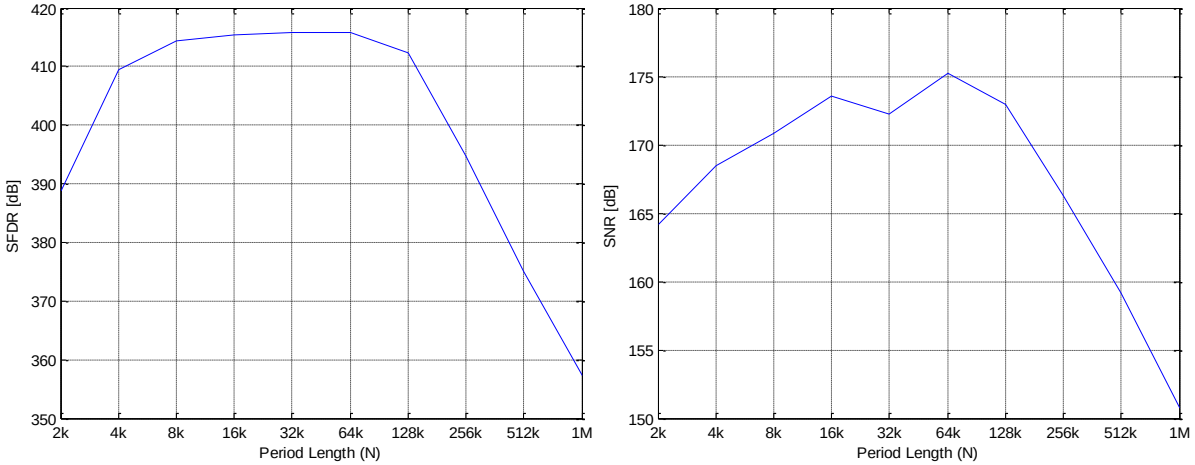


Figure 113. SFDR (left) and SNR (right) of the quadratic interpolator as a function of *N*, *B*=36, *M*=1024

Figure 113 presents the relation between the attainable SFDR, SNR and the period length, using a single BRAM primitive as a LUT. For N<2048, $f(\varphi)$ can be generated by bypassing the interpolators. When flow-control selects no bit-shifting and continuous loading of the first interpolator, choosing interpolation length = 1, LUT contents are practically fed forward to the output directly.

9 Bibliography

[1] Manoel E.de Lima, David J.Kinniment: "Sea-of-gates architecture", *Microelectronics Journal*, Volume 26, Issue 5, July 1995, Pages 431-440. DOI: 10.1016/0026-2692(95)98945-N

[2] AMD/Xilinx Inc, "Zynq-7000 SoC Data Sheet", Product Specification DS190 (v1.11.1) July 2, 2018 https://www.xilinx.com/content/dam/xilinx/support/documentation/data_sheets/ds190-Zynq-7000-Overview.pdf

[3] AMD/Xilinx Inc, "Zynq UltraScale+ MPSoC Data Sheet", Product Specification DS891 (v1.9) May 26, 2021, https://www.xilinx.com/support/documentation/data_sheets/ds891-zynq-ultrascale-plus-overview.pdf

[4] Allaoui, R., H. H. Mouane, Z. Asrih, S. Mars, and I. El Hajjouji. "FPGA-based implementation of optical flow algorithm." In 2017 International Conference on Electrical and Information Technologies (ICEIT), pp. 1-5. IEEE, 2017. DOI: 10.1109/EITech.2017.8255246

[5] Jammoussi, Ameni Yengui, Sameh Fakhfakh Ghribi, and Dorra Sallami Masmoudi. "Implementation of face recognition system in Virtex II Pro platform." In 2009 3rd International Conference on Signals, Circuits and Systems (SCS), pp. 1-6. IEEE, 2009. DOI: 10.1109/ICSCS.2009.5412313

[6] Menke, Jan. "Improving the image quality of contrast-enhanced MR angiography by automated image registration: A prospective study in peripheral arterial disease of the lower extremities." European journal of radiology 75.3 (2010): pp 1997-2009. DOI: 10.1109/TMI.2017.2725644

[7] European Machine Vision Association, "EMVA Standard 1288 Standard for Characterization of Image Sensors and Cameras", 2016, https://www.emva.org/wp-content/uploads/EMVA1288-3.1a.pdf

[8] David Perry, Eustace Dereniak, "Linear theory of nonuniformity correction in infrared staring sensors", SPIE Optical Engineering 32(8), August 1993. DOI: 10.1117/12.145601

[9] Zhu, Yajun, et al. "A 160× 120 ROIC with Non-uniformity Calibration for Silicon Diode Uncooled IRFPA." 2019 IEEE International Conference on Electron Devices and Solid-State Circuits (EDSSC). IEEE, 2019. DOI: 10.1109/EDSSC.2019.8754184

[10] Purnawarman Musa, Farid A. Rafi, Missa Lamsani, "A Review: Contrast-Limited Adaptive Histogram Equalization (CLAHE) methods to help the application of face recognition," *Third International Conference on Informatics and Computing (ICIC)*, 2018, pp. 1-6, doi: 10.1109/IAC.2018.8780492.

[11] Pablo Arbeláez, Michael Maire, Charless Fowlkes and Jitendra Malik, "Contour Detection and Hierarchical Image Segmentation" in *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 33, no. 5, pp. 898-916, May 2011. DOI: 10.1109/TPAMI.2010.161

[12] Xingxing Li, et al. "Accurate and Automatic Extrinsic Calibration for a Monocular Camera and Heterogenous 3D LiDARs", *IEEE Sensors Journal*, vol. 22, no. 16, pp. 16472-16480, 15 Aug.15, 2022, DOI: 10.1109/JSEN.2022.3189041

[13] Andrew W. Fitzgibbon, "Simultaneous linear estimation of multiple view geometry and lens distortion" in *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition* (CVPR). IEEE. DOI:10.1109/CVPR.2001.990465.

[14] Jack Keith, "Video Demystified", 5th Edition, Elsevier Inc, 2011, ISBN-13: 978-8190935661, pp. 15-22.

[15] Jang, Wonwoo, et al. "An image signal processor for ultra-small HD video sensor with 3A in camera phones." *Digest of Technical Papers International Conference on Consumer Electronics*. IEEE, 2009. DOI: 10.1109/ICCE.2009.5012325

[16] Isao Takayanagi, Junichi Nakamura, "Image Sensors and Signal Processing for Digital Still Cameras" 1st Edition, CRC Press, 2005, ISBN 0849335450

[17] Min-Woong Seo, Keita Yasutomi, Keiichiro Kagawa, Shoji Kawahito, "A Low Noise CMOS Image Sensor with Pixel Optimization and Noise Robust Column-parallel Readout Circuits for Low-light Levels", *ITE Transactions on Media Technology and Applications*, 2015, (4):258-262 DOI: 10.3169/mta.3.258

[18] Kim, Min-Kyu, Seong-Kwan Hong, and Oh-Kyong Kwon. "A Fast Multiple Sampling Method for Low-Noise CMOS Image Sensors with Column-Parallel 12-bit SAR ADCs." *IEEE Sensors* (14248220) 16.1 (2016). DOI:10.3390/s16010027

[19] Jonathan M. Mooney, Freeman D. Shepherd, William S. Ewing, James E. Murguia, Jerry E. Silverman, "Responsivity nonuniformity limited performance of infrared staring cameras", *Optical Engineering* 28(11), 1151—1161 (1989). DOI: 10.1117/12.7977112

[20] David Perry, Eustace Dereniak, "Linear theory of nonuniformity correction in infrared staring sensors", *Optical Engineering* 32(8), (1 August 1993), DOI: 10.1117/12.145601

[21] Schulz, Max, and Larry Caldwell. "Nonuniformity correction and correctability of infrared focal plane arrays." *Infrared Physics & Technology* 36.4 (1995): 763-777. DOI: 10.1016/1350-4495(94)00002-3

[22] James A. Seibert, John M. Boone, and Karen K. Lindfors. "Flat-field correction technique for digital detectors.", *Medical Imaging* 1998: Physics of Medical Imaging, vol. 3336, pp. 348-354. SPIE, 1998. DOI: 10.1117/12.317034

[23] L. Mei, L. Zhang, Z. Kong, H. Li "Noise modeling, evaluation and reduction for the atmospheric lidar technique employing an image sensor." *Optics Communications* 426 (2018): 463-470. DOI: 10.1016/j.optcom.2018.05.072

[24] Teledyne (e-c), "EV76C661 1.3 Mpixel Color CMOS Image Sensor Datasheet", E2V Inc, 2019. Available online: https://imaging.teledyne-e2v.com/content/uploads/2019/02/DSC_EV76C661.pdf

[25] Scharstein, Daniel, Szeliski, Richard. "High-accuracy stereo depth maps using structured light", *IEEE Computer Society Conference on Computer Vision and Pattern Recognition* (CVPR 2003), vol 1, pp 195-202, Madison, WI, June 2003. DOI: 10.1109/CVPR.2003.1211354

[26] Liu, Li, and Tianxu Zhang. "Optics Temperature-Dependent Nonuniformity Correction Via l0-Regularized Prior for Airborne Infrared Imaging Systems." *IEEE Photonics Journal* 8.5 (2016): 1-10. DOI: 10.1109/JPHOT.2016.2602059

[27] Guan, Juntao, et al. "Fixed pattern noise reduction for infrared images based on cascade residual attention CNN." *Neurocomputing* 377 (2020): 301-313. DOI: 10.1016/j.neucom.2019.10.054

[28] Karaküçük, Ahmet, and Ahmet Emir Dirik. "Adaptive photo-response non-uniformity noise removal against image source attribution." *Digital Investigation* 12 (2015): 66-76. DOI: 10.1016/j.diin.2015.01.017

[29] Burggraaff, O., Schmidt, N., Zamorano, J., Pauly, K., Pascual, S., Tapia, C., Spyrakos, E. and Snik, F., "Standardized spectral and radiometric calibration of consumer cameras". *Optics Express*, 27.14 (2019), pp.19075-19101. DOI: 10.1364/OE.27.019075

[30] Wang, Ye, et al. "PRNU Estimation of Linear CMOS Image Sensors That Allows Non-Uniform Illumination." *IEEE Transactions on Instrumentation and Measurement* (2021). DOI: 10.1109/TIM.2021.3088484

[31] Donald L. Snyder, Dennis L. Angelisanti, William H. Smith, and Guang-Ming Dai. "Correction for nonuniform flat-field response in focal plane arrays.", *Digital Image Recovery and Synthesis* III, vol. 2827, pp. 60-67. SPIE, 1996. DOI: 10.1117/12.255089

[32] Vladimir Vasiliev, Fedor Inochkin, Sergey Kruglov, and Igor Bronshtein. "Real-time image processing inside a miniature camera using small-package FPGA.", *2018 International Symposium on Consumer Technologies* (ISCT), pp. 5-8. IEEE, 2018. DOI: : 10.1109/ISCE.2018.8408916

[33] Richard Bowman, Boyko Vodenicharski, Joel Collins, Julian Stirling, "Flat-field and colour correction for the raspberry pi camera module.", *arXiv preprint* arXiv:1911.13295 (2019). DOI: 10.5334/joh.20

[34] Zhang, Zhaoning, Yujie Wang, Rafael Piestun, and Zhen-Li Huang. "Characterizing and correcting camera noise in back-illuminated sCMOS cameras." *Optics Express* 29, no. 5 (2021): 6668-6690. DOI: 10.1364/OE.418684

[35] Orżanowski, Tomasz. "Nonuniformity correction algorithm with efficient pixel offset estimation for infrared focal plane arrays." *SpringerPlus 5.1* (2016): 1-8. DOI: 10.1186/s40064-016-3534-1

[36] Yao, Pingping, et al. "Non-uniformity calibration method of space-borne area CCD for directional polarimetric camera." *Optics Express* 29.3 (2021): 3309-3326. DOI: 10.1364/OE.410768

[37] Hu, Changmiao, Yang Bai, and Ping Tang. "Denoising algorithm for the pixel-response non-uniformity correction of a scientific CMOS under low light conditions." *The International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences* 41 (2016): 749. DOI:10.5194/isprsarchives-XLI-B3-749-2016

[38] Kirkpatrick, Scott, C. Daniel Gelatt, and Mario P. Vecchi. "Optimization by simulated annealing." *Science* 220.4598 (1983): 671-680. DOI: 10.1126/science.220.4598.671

[39] Paul M. Hubel, John Liu, and Rudolph Guttosch, "Spatial Frequency Response of Color Image Sensors: Bayer Color Filters and Foveon X3"; *2004. Proc. SPIE* Vol. 5301, pp. 402-4, DOI:10.1117/12.561568

[40] Bryce Bayer, "Color imaging array", 1976, US Patent 3971065

[41] Jin Wang, Jiaji Wu, Zhensen Wu, Gwanggil Jeon,"Filter-based Bayer Pattern CFA Demosaicking", *Circuits Systems, and Signal Processing* 36, 2917–2940 (2017). DOI:10.1007/s00034-016-0448-7A.

[42] Keigo Hirakawa and Tomas W. Parks, "Adaptive homogeneity directed demosaicing algorithm," *IEEE Transactions on Image Processing*, vol. 14, no. 3, pp. 360–369, Mar. 2005.

[43] Dragulinescu, L. Lizarraga, S. Mir, G. Sicard, "Defect and fault modelling of a CMOS n-diffusion photodiode", *Advanced Topics in Optoelectronics, Microelectronics, and Nanotechnologies* III 2007 May 31 (Vol. 6635, pp. 301-309). SPIE. DOI: 10.1117/12.742105

[44] Guy Meynants, Bart Dierickx, "A circuit for the correction of pixel defects in image sensors", 1998, *Proceedings of the 24th European Solid-State Circuits Conference*, ESSCIRC '98 pp: 312 – 315. DOI: 10.1109/ESSCIR.1998.186271

[45] Craig Reinhart, Manjunath Bhat, David Standley, "Automatic bad pixel correction in image sensors", 2006, US Patent 7034874

[46] Jeehoon An, Wonjae Lee, Jaeseok Kim, "Adaptive Detection and Concealment Algorithm of Defective Pixel", *Proceedings of the 2007 IEEE Workshop on Signal Processing Systems*, pp 651-656. DOI: 10.1109/SIPS.2007.4387626

[47] Anthony Tanbakuchi, Arjen Sijde, Bart Dillen "Adaptive Pixel Defect Correction", *Proceedings of SPIE 5017, Sensors and Camera Systems for Scientific, Industrial, and Digital Photography Applications IV,* (2003). Vol. 5017. pp 360-370. DOI: 10.1117/12.499223

[48] Jim S. Jimmy Li, Sharmil Randhawa, "Adaptive order-statistics multi-shell filtering for bad pixel correction within CFA demosaicking", *TENCON 2009* - 2009 IEEE Region 10 Conference. DOI: 10.1109/TENCON.2009.5395817

[49] Ghislain T. Tchendjou, Emmanuel Simeu, "Detection, Location and Concealment of Defective Pixels in Image Sensors,", *IEEE Transactions on Emerging Topics in Computing*, vol. 9, no. 2, pp. 664-679, 1 April-June 2021, DOI:10.1109/TETC.2020.2976807.

[50] Omer F. Adil, Huseyin S. Demir, "Scene-based bad pixel detection using interframe registration.", *Electro-Optical and Infrared Systems: Technology and Applications* XV. Vol. 10795. International Society for Optics and Photonics, 2018. DOI: 10.1117/12.2324957

[51] Izhak Baharav, Ramekrishna Kakarala, Xuemei Zhang, Dietrich W. Vook, "Bad Pixel Detection and Correction in an Image Sensing Device", 2004. US Patent 6737625

[52] Arjan Gijsenij, Theo Gevers, and Joost Van De Weijer, "Computational color constancy: Survey and experiments." *IEEE transactions on image processing*, 20.9 (2011): 2475-2489. DOI: 10.1109/TIP.2011.2118224

[53] Min Huang, Yonghui Xi, Jie Pan, Ruili He, Xiu Li, "Colorimetric Observer Categories for Young and Aged Using Paired-Comparison Experiments", *IEEE Access*, vol. 8, pp. 219473-219482, 2020, DOI: 10.1109/ACCESS.2020.3042817.

[54] Marc Ebner, "White Patch Retinex, Color Constancy". Wiley & Sons, 2007. ISBN 978-0-470-05829-9

[55] Hamilton. Y. Chong, Steven. J. Gortler and Todd Zickler, "The von Kries Hypothesis and a Basis for Color Constancy," 2007 IEEE 11th International Conference on Computer Vision, 2007, pp. 1-8, DOI: 10.1109/ICCV.2007.4409102.

[56] Francesca Gasparini, Raimondo Schettini, "Color Correction for Digital Photographs", Proceedings of the 12th International conference on Image Analysis and Processing (ICIAP '03), (pp. 646-651), DOI: 10.1109/ICIAP.2003.1234123

[57] Peter V. Gehler, Carsten Rother, Andrew Blake, Tom Minka and Toby Sharp, "Bayesian color constancy revisited," 2008 IEEE Conference on Computer Vision and Pattern Recognition, 2008, pp. 1-8, DOI: 10.1109/CVPR.2008.4587765

[58] Simone Bianco, Francesca Gasparini, and Raimondo Schettini, "Combining strategies for white balance." In Digital photography III, vol. 6502, pp. 113-121. SPIE, 2007. DOI: 10.1117/12.705190

[59] Dongliang Cheng, Dilip K. Prasad, and Michael S. Brown. "Illuminant estimation for color constancy: why spatial-domain methods work and the role of the color distribution." *Journal of the Optical Society of America*, A 31.5 (2014): 1049-1058. DOI: 10.1364/JOSAA.31.001049

[60] Yongok Han and Iickho Song, "Some useful weights of recursive weighted median filters," *1991 IEEE International Symposium on Circuits and Systems (ISCAS)*, 1991, pp. 204-207 vol.1, DOI: 10.1109/ISCAS.1991.176309

[61] Roberto Roncella, Roberto Saletti, Pierangelo Terreni, "70-MHz 2-mm CMOS Bit-Level Systolic Array Median Filter", *IEEE Journal of Solid State Circuits*, vol 28, 1993. DOI: 10.1109/4.229403

[62] Barun K. Kar and Dhiraj K. Pradhan, "A new algorithm for order statistic and sorting", *IEEE Transactions on Signal Processing*, vol 41, August 1993. DOI: 10.1109/78.229899

[63] Mustafa Karaman, Levent Onural and Abdullah Atalar, "Design and implementation of a general purpose median filter in CMOS VLSI" *IEEE Journal of Solid-State Circuits*, vol. 25, April 1990. DOI: 10.1109/4.52178

[64] Chaitali Chakrabarti, "Sorting network based architectures for median filters". *IEEE Transactions on Signal Processing*, March 1994. DOI: 10.1109/82.251840

[65] Chaitali Chakrabarti and Li-Yu Wang, "Novel Sorting Network-Based Architectures for Rank Order Filters", *IEEE Transactions on VLSI Systems*, vol. 2, December 1994. DOI: 10.1109/92.335027

[66] Suhaib A. Fahmy, Peter Y. K. Cheung, Wayne Luk: "Novel FPGA-Based Implementation of Median and Weighted Median Filters for Image Processing", *Field-Programmable Logic and Applications* (FPL 2005), 2005. DOI: 10.1109/FPL.2005.1515713

[67] J. W. Cooley and J. W. Tukey, "An Algorithm for the Machine Computation of Complex Fourier Series", *Mathematics of Computation*, Vol. 19, pp. 297-301, Apr. 1965.

[68] Joseph Tierney, Charles M. Rader, and Bernard Gold, "A digital frequency synthesizer", *IEEE Transactions on Audio Electroacoustics*, vol. AU-19, pp. 48-57, Mar 1971. DOI: 10.1109/TAU.1971.1162151

[69] Florean Curticapean, Kalle I. Palomaki, Jarkko Niittylahti, "Quadrature direct digital frequency synthesizer using an angle rotation algorithm", ISCAS '03. *Proceedings of the 2003 International Symposium on Circuits and Systems*, 2003, vol 2, pp 81-84, DOI: 10.1109/ISCAS.2003.1205895

[70] Kalle I. Palomaki, Jarkko Niittylahti, “Direct digital frequency synthesizer architecture based on Chebyshev approximation”, *Conference Record of the Thirty-Fourth Asilomar Conference on Signals, Systems and Computers*, 29 Oct.-1 Nov. 2000, vol. 2, pp :1639 – 1643. DOI: 10.1109/ACSSC.2000.911267

[71] Liu, S.-I.; Yu, T.-B.; Tsao, H.-W.; “Pipeline direct digital frequency synthesizer using decomposition method”, *IEEE Proceedings on Circuits, Devices and Systems*, June 2001, vol. 148, issue: 3, pp :141 – 144. DOI: 10.1049/ip-cds:20010158

[72] Richard C. Meitzler, Wesley P. Millard, “A Direct Digital Frequency Synthesizer Prototype for Space Applications”, *11th Annual NASA Symposium on VLSI Design*, Coeur d'Alene, ID, May 28-29, 2003. https://bit.ly/39JOS7Q

[73] Lattice Semiconductor, “CrossLink-NX Family Datasheet”, FPGA-DS-02049-1.2.1, October 2021 https://www.latticesemi.com/view_document?document_id=52780

[74] Pierre Langlois, Dhamin Al-Khalili, “Novel Approach to the Design of Direct Frequency Synthesizers Based on Linear Interpolation”, *IEEE Transactions on Circuits and Systems – II: Analog and Digital Signal Processing*, vol. 50#9, Sept, 2003. DOI: 10.1109/TCSII.2003.815020

[75] Francisco Cardells-Tormo, Javier Valls-Coquillat, "Area-optimized implementation of quadrature direct digital frequency synthesizers on LUT-based FPGAs," *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, vol. 50, no. 3, pp. 135-138, March 2003, doi: 10.1109/TCSII.2003.809716.

[76] Xilinx DDS Compiler v6.0, PG141, January 21, 2021, Vivado Design Suite LogiCore IP Product Guide. https://docs.xilinx.com/v/u/en-US/pg141-dds-compiler

[77] Kalle I. Palomaki, Jarkko Niittylahti, “Methods to improve the performance of quadrature phase-to-amplitude conversion based on Taylor series approximation”, *Proceedings of the 43rd IEEE Midwest Symposium on Circuits and Systems*, 8-11 Aug. 2000, vol 1 , pp :14 – 17, DOI: 10.1109/MWSCAS.2000.951576

[78] Jouko Vankka, “Methods of Mapping from Phase to Sine Amplitude in Direct Digital Synthesis”, *IEEE Transaction on Ultrasonics, Ferroelectrics, and Frequency Control*, Vol 44. No 2, March 1997. DOI: 10.1109/58.585137

[79] Florean Curticapean, Jarkko Niittylahti, “A hardware efficient direct digital frequency synthesizer”, *The 8th IEEE International Conference on Electronics, Circuits and Systems*, 2001. ICECS 2001. 2-5 Sept. 2001, vol. 1, pp: 51 – 54. DOI: 10.1109/ICECS.2001.957664

[80] Benoit R. Veillette, Gordon W. Roberts, “High frequency sinusoidal generation using delta-sigma modulation techniques”, ISCAS '95, 1995 *IEEE International Symposium on Circuits and Systems*, Volume: 1 , 28 April-3 May 1995, Page(s): 637 -640 vol.1. DOI: 10.1109/ISCAS.1995.521594

[81] Ray Andraka, “A survey of CORDIC algorithms for FPGA based computers”, *Proceedings of the 1998 ACM/SIGDA sixth international symposium on Field programmable gate arrays*, Feb. 22-24, 1998, Monterey, CA. pp191-200. DOI: 10.1145/275107.275139

[82] Ireneusz Janiszewski, Bernhard Hoppe, Hermann Meuth, “VHDL-Based Design and Design Methodology for Reusable High Performance Direct Digital Frequency Synthesizers” *Proceedings of the 38th Design Automation Conference*, DAC 2001, June 18-22, 2001, Las Vegas, NV, USA, pp 573-578. DOI: 10.1145/378239.379026

[83] Thomas Holton, “Digital Signal Processing: Principles, Algorithms and Applications”, Cambridge University Press, 2021, ISBN-13:978-1108418447

[84] Tomasi, Carlo, and Roberto Manduchi. “Bilateral filtering for gray and color images.” *Sixth international conference on computer vision* (IEEE Cat. No. 98CH36271). IEEE, 1998. DOI: 10.1109/ICCV.1998.710815

[85] Robert Szeliski, James Coughlan, “Spline-Based Image Registration.” *International Journal of Computer Vision* 22, 199–218 (1997). DOI: /10.1023/A:1007996332012

[86] Juntao Guan, et al. “Fixed pattern noise reduction for infrared images based on cascade residual attention CNN.” *Neurocomputing* 377 (2020): pp 301-313. DOI: 10.1016/j.neucom.2019.10.054

[87] Shousheng He and Mats Torkelson, “A New Approach to Pipeline FFT Processor”, *Proceedings of the 10th International Parallel Processing Symposium*, 1996, DOI: 10.1109/IPPS.1996.508145

[88] Alan Edelman, Peter McCorquodale, Sivan Toledo, ”The Future Fast Fourier Transform?”, 1999 Society for Industrial and Applied Mathematics, *SIAM Journal on Scientific Computing*, Volume 20, Number 3, pp. 1094-1114. DOI: 10.1137/S1064827597316266

[89] William. R. Knight, “A Simple Fixed-Point Error Bound for the Fast Fourier Transform”, *IEEE Trans. Acoustics, Speech and Signal Proc*., 1979, Vol. 27, No. 6, pp. 615-620, DOI: 10.1109/TASSP.1979.1163314

9.1 Publications related to the dissertation

[S1] Gabor Szedo, Jose R. Alvarez. "Methods of reducing aberrations in a digital image", US Patent 8400533, Filed: Feb 23, 2010, Granted: Mar 19, 2013.

[S2] Gabor Szedo, Vanessa Y. Chou. "Circuits for and methods of generating a digital image", US Patent 9237257, Filed: June 14, 2013, Granted Jan 12, 2016.

[S3] Gabor Szedo Becker, Róbert Lovas, "Uniformity Correction of CMOS Image Sensor Modules for Machine Vision Cameras", Sensors vol 22. no. 24: 9733. 2022. DOI: 10.3390/s22249733

[S4] Gabor Szedo, Jeffrey D. Stroomer, Jose R. Alvarez, "Determining outlier pixels of successive frames", US Patent 8774544, Filed: Jul 23, 2010, Granted: Jul 8, 2014.

[S5] Gabor Szedo Becker, "Low-Cost Spatio-Temporal Algorithm for Defective Pixel Identification and Correction", in Proceedings of the 2022 IEEE 19th International Symposium on Intelligent Systems and Informatics (SISY), September 2022, accepted for publication

[S6] Gabor Szedo. "Weight normalization in hardware without a division operator", US Patent 8484267, Filed: Nov 19, 2009, Granted: July 9, 2013.

[S7] Gabor Szedo, Steve Elzinga, Greg Jewett. "Image Sensor Color Calibration Using the Zynq-7000 SoC", *Xcell Magazine, Issue 81, Fourth quarter* 2012

[S8] Gábor Szedő, Béla Fehér. "High Speed FPGA implementation of Median Filters" *NDES'98 Nonlinear Dynamics of Electronic Systems*, Technical University of Budapest, Hungary 16-18. July 1998 Proceedings, pp. 191-193.

[S9] Gábor Szedő. "Noise Filter Applications on Reconfigurable Hardware", *XI. Conference on Application of Microprocessors in Automatic Control and Measurement*, Warsaw, Poland 13-14. October 1998. Proceedings, Vol 1. pp. 3-11.

[S10] Béla Fehér, Gábor Szedő. "Nonlinear Filters on Reconfigurable Processors", *Tempus Symposium on Intelligent Systems in Control and Measurement*, Miskolc, Hungary, 21-27. November 1998. Proceedings, pp. 162-167.

[S11] Gábor Szedő, Béla Fehér. "Noise Filter Applications on Reconfigurable Hardware", *PACT '98. International Conference on Parallel Architectures and Compilation Techniques, Workshop on Reconfigurable Computing*, Paris, France October 13. 1998. Proceedings, pp. 95-100.

[S12] Szedő Gábor, Fehér Béla: "Nagy sebességű medián szűrők megvalósítása FPGA áramkörök segítségével", Mérés-Automatizálás'98 Konferencia az Alkalmazott Informatika Jegyében 1998. November 3-5.Proceedings, pp. 39-47

[S13] Gábor Szedő, Péter Szántó. "Cost-Effective Two-Dimensional Rank-Order Filters on FPGAs", *Xilinx DSP Magazine*, Apr 2007, pp 34-37

[S14] Péter Szántó, Gábor Szedő, Béla Fehér. "High Performance Timing-Driven Rank Filter", *ICECS 2006, 13th IEEE International Conference on Electronics, Circuits and Systems. Special Issue, IEEE, 2006.* pp 752-755, DOI: 10.1109/ICECS.2006.379898

[S15] Péter Szántó, Gábor Szedő, Béla Fehér. "Implementing 2D Median Filters in FPGAs", *Transactions of the University of Mining and Metallurgy of Ostrava.* (1210-0471 1804-0993): 1 pp 179-184 (2006) ICCC 2006.

[S16] Peter Szanto, Bela Feher, Gabor Szedo. "Scalable Architecture for Rank Order Filtering", US Patent 8713082, Filed on Mar 2, 2007, Granted Apr 29, 2014.

[S17] Peter Szanto, Gabor Szedo, Bela Feher, Wilson C. Chung. "Scalable architecture for rank order filtering", US Patent 8005881, Filed: Mar 2, 2007, Granted: Aug 23, 2011.

[S18] Hemang Parekh, Helen Tarn, Gabor Szedo, Vanessa Chou, Jeffrey Graham, Beth Cowie. "Method of and Circuit for Buffering Data", US Patent 7669017, Filed Sept 27, 2006, Granted: Feb 23, 2010.

[S19] Gabor Szedo, Helen Hai-Jo Tarn. "Memory segmentation for Fast Fourier Transform", US Patent 7395293, Filed: July 23, 2004, Granted: Jul 1, 2008.

[S20] Gabor Szedo, Hemang Parekh. "Data reorganizer for Fourier transformation of parallel data streams", US Patent 8572148, Filed: Feb 23, 2009, Granted Oct 29, 2013.

[S21] Gábor Szedő, Ted Bapty, Sandeep Neema, Jason Scott. "Reconfigurable Target Recognition System" *Eighth ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, FPGA 2000, February 10-11, Monterey, CA, Proceedings pp 221.

[S22] Sandeep Neema, Jason Scott, Theodore Bapty, Gabor Szedo, Bela Feher. "Real Time Reconfigurable Image Recognition System", *18th IEEE International Conference on Instrumentation and Measurement Technology*, May 21-23, 2001, Proceedings pp. 351-355.

[S23] Gabor Szedo; Vanessa Yang; Chris Dick. "High-performance FFT processing using reconfigurable logic", *2001. Conference Record of the Thirty-Fifth Asilomar Conference on Signals, Systems and Computers*, 4-7 Nov. 2001, Proceedings, pp:1353 - 1356 vol.2.

[S24] Béla Fehér, Gábor Szedő. "Cost effective 2x2 inner product processors", *Field Programmable Logic and Applications*, 8th International Workshop, FPL'98, Tallin, Estonia, 3-5 September 1998. Proceedings, pp. 348-355.

[S25] Gabor Szedo. "Quadratic Approximation for Fast Fourier Transform", US Patent 7984091, Filed: Oct 14, 2005, Granted: July 19, 2011.

[S26] Gábor Szedő. "A Compact High Resolution Phase-Factor Generator for Large Point-Size FFTs" *Global Signal Processing Conference on Pervasive Signal Processing, GSPx 2005* Oct 24-27, 2005, Santa Clara, CA USA, Proceedings on CD ROM, track: FPGA based DSP.

[S27] Gábor Szedő: "FPGA Based Reconfigurable Logics" Mini-symposium, Technical University of Budapest Department of Measurement and Instrumentation, Feb. 1997, Proceedings, pp. 38-39.

[S28] Gábor Szedő: "Potentials of Dynamic Reconfiguration", *Conference on the latest results of information technology*, Technical University of Budapest department of Process Control, 1997 Proceedings pp. 14-15.

[S29] Gábor Szedő: "An Application Example of Reconfigurable Logics", *Mini-symposium, Technical University of Budapest department of Measurement and Information Systems*, February 1998 Proceedings pp. 38-39.

[S30] Gábor Szedő: "An Application of Reconfigurable Hardware: The Finite Element Model" *microCAD'98 International Computer Science Conference*, February 25-26, 1998. Proceedings, Section E. pp. 43-45.

[S31] Gabor Szedo, Singh Vinay Jitendra, L. James Hwang. "Generation of a high-level simulation model of an electronic system by combining an HDL control function translated to a high-level language and a separate high-level data path function", US Patent 7684968, Filed: Dec 9, 2004, Granted: Mar 23, 2010

[S32] Sean A. Kelly, Gabor Szedo. "Interface for managing multiple implementations of a functional block of a circuit design", US Patent 8181149, , Filed: Sept 3, 2009, Granted: May 15, 2012

9.2 Other, non-related publications

[SX1] Joachim Bauer, Gabor Szedo Becker, US Patent 11321873, "Calibrating and detecting vibration of stereo ranging systems", Filed: February 4, 2019, Granted May 3, 2022

[SX2] Gabor Szedo Becker, US Patent 11076085, "Detecting interference between time-of-flight cameras using modified image sensor arrays", Filed: June 20, 2018, Granted: July 27, 2021

[SX3] Gabor Szedo Becker, Chengwu Luke Cui, Anirudth Nambirajan. "Rolling shutter motion and depth sensing", US Patent 10812777, Filed: Oct 31, 2018, Granted: Oct 20, 2020

[SX4] Gabor Szedo Becker. "Synchronizing time-of-flight cameras", US Patent 10674063, Filed: Jun 20, 2018, Granted: Jun 2, 2020.

[SX5] Joachim Bauer, Gabor Szedo Becker, "Calibrating and detecting vibration of stereo ranging systems", US Patent 11321873, Filed February 4, 2019, Granted: May 3, 2022

[SX6] Gabor Szedo, Christopher J. Martin, Ted N. Booth. "Image Stabilization", US Patent 9305362, Filed: Feb 28, 2014, Granted: Apr 5, 2016.

[SX7] Gabor Szedo, Steven P. Elzinga, Jose R. Alvarez, "Method and device for generating a digital image based upon a selected set of chrominance groups", US Patent 9013611, Filed Sept 6, 2012: Granted: Apr 21, 2015.

[SX8] Gabor Szedo, Jose R. Alvarez. "Color filter array alignment detection", US Patent 8441562, Filed: Sept 2, 2010, Granted: May 14, 2013.