University College Cork, Ireland
Coláiste na hOllscoile Corcaigh

Multi-Spectral In-vivo FPGA-based Surgical Imaging*

Majed Alsharari[1,5][0000−0003−4755−4572], Lorenzo Niemitz[2], Simon Sorensen[2], Roger Woods[1][0000−0001−6201−4270], Ray Burke[2], Stefan Andersson Engels[2,3], Carlos Reaño[4], and Son T. Mai[1]

[1] Queen's University Belfast, Northern Ireland BT9 5AF, UK
(malsharari01, thaison.mai, r.woods)@qub.ac.uk
https://www.qub.ac.uk/schools/eeecs/
[2] Tyndall Institute, Lee Maltings Complex Dyke Parade, Cork, Ireland, T12 R5C
(lorenzo.niemitz,simon.sorensen, ray.burke)@tyndall.ie
https://www.ipic.ie/
[3] Department of Physics, Kane Science Building, University College Cork, Ireland
stefan.andersson-engels@tyndall.ie
http://research.ucc.ie/profiles/D006/stefan.andersson-engels@tyndall.ie
[4] Universitat de València, 46100, Spain
carlos.reano@uv.es
http://www.uv.es/caregon2
[5] Jouf University, Sakaka, 72341, Saudi Arabia
malsharari@ju.edu.sa
https://www.ju.edu.sa

Abstract. Intelligent and adaptive in-vivo, catheter-based imaging systems with enhanced processing and analytical capability have the potential to enhance surgical operations and improve patient care. The paper describes an intelligent surgical imaging system based on a 'chip on tip', which reduces the need for conventional imaging. The associated embedded system provides real-time, in-vivo imaging analysis and data display for surgeons, enhancing their ability to detect clinically significant tissue. The paper presents initial work on an field programmable gate array implementation of a contrast limited adaptive histogram equalization algorithm, Hessian matrix construction and region of interest function on the AMD-Xilinx's Kria KV260 board. It outlines optimizations undertaken to reduce the BRAMs by 38%, DSP48 blocks by 80%, flip-flops by 33% and LUTs by 36%, thus creating a design operating at 121 FPS.

Keywords: Surgical imaging · field programmable gate array.

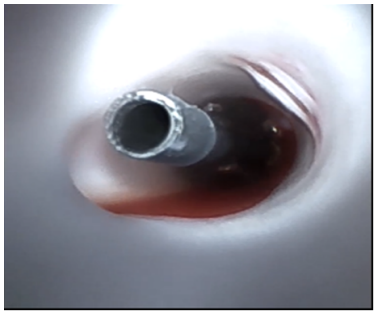
1 Introduction

In current surgical practice, surgeons still rely heavily on external, 'gold standard' imaging systems such as X-ray, CT and MRI. In-vivo imaging systems

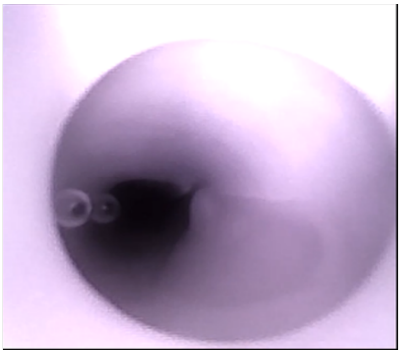
* Partially supported by Jouf University.

such as endoscopes, intravascular ultrasound are standard but can be further enhanced and using smart and integrated micro cameras. This offers the potential to enhance surgical procedures and outcomes by providing high quality, diagnostic images from deep within the body using micro-scale image sensors on a micro catheter platform.

As an example, the unprocessed in-vivo images in the femoral artery, distal to proximal, of a porcine model with different illumination RGB and Near Infra-Red (NIR) are shown in Fig 1. Fig 1(a) shows a clear field of the marker band of a balloon catheter, a commonly used medical device for cardiac procedures, and Fig 1(b) gives the same location illuminated using 940nm NIR. Commercial micro-camera integrated circuits are available, but they are not specifically designed for biophotonics applications such as surgical guidance, based on diffuse reflectance imaging, fluorescence and reflectance for specific biomarkers. Commercially available micro-cameras [8] are limited by resolution, image quality, sensitivity, field of view, etc,. thus limiting their use in-vivo.



(a) RGB image showing the marker band of a balloon catheter



(b) 940nM NIR illumination showing a diffuse image with some ability to image through the blood field.

Fig. 1. Multispectral images from inside the femoral artery of a porcine model (Courtesy of Tyndall National Institute.)

Using integrated image sensors in-vivo to successfully allow, for example, specular reflection and effective viewing of a beating heart, poses image processing and data analytics challenges. This can be resolved by employing smart, adaptive algorithms on an embedded system to enhance the image effectively, but requires adoption of a suitable low power technology and careful design. The Tyndall National Institute (TNI) in collaboration with clinicians and the medical device industry, are creating an intelligent surgical system (ISS) based on a custom CMOS image sensor and embedded processing unit which provides both image sensor power management and image processing capability to convert the detected signals at the edge or interface into clinical significant medical images,

in real time. This paper describes the collaboration with Queen's University to implement the image processing functionality on an field programmable gate array (FPGA) on the AMD-Xilinx's Kria KV260 AI board.

The work uses multispectral image processing to recover the best possible RGB images, including enhancement with at least two NIR wavelengths. A contrast limited adaptive histogram equalization (CLAHE) algorithm [5] is employed to help outline specific features such as tumours, by suppressing the contrast of each pixel based on its neighbouring values. A Region of Interest (ROI) algorithm highlights intra-operative ROIs to the clinician, by applying convolution procedures using the derivatives of Gaussian kernel to construct the Hessian matrix of each pixel with the eigenvalues used by an edginess or ROI function. In this paper, we undertake a number of optimisations for the implementation of this functionality using the AMD-Xilinx Vitis High-Level Synthesis (HLS) tools.

The paper is structured as follow: Section 2 briefly describes the ISS, followed by an explanation in Section 3 of the processes and algorithms used for the detection of blood vessels. The system architecture is then described in Section 4 and followed by the results in Section 5. Conclusions are given in Section 6.

2 Intelligent Surgical System

The proposed ISS consists of a front-end comprises a micro camera and light source, and a small embedded processor with intelligent image processing functionality, connected via a fiber optic cable to a transceiver (see Fig. 2). The front-end module needs to have a small footprint within a microcatether (3-6 Fr.) in order to allow surgeons to navigate to the narrow regions inside body organs. The challenge is to undertake the design of this functionality in a lower power, FPGA technology that provides the adaptive processing to support evolving requirements. The back-end comprises a high performance computing resource which, in the future, will incorporate additional intelligence (AI) capability, gleaned by surgeons from operations as they are performed.

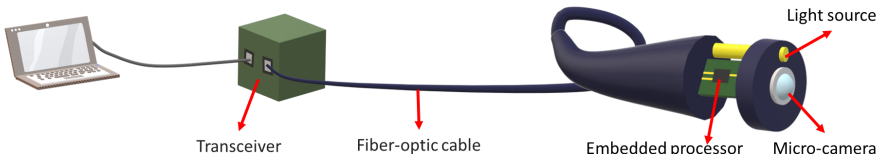


Fig. 2. Visualisation of the overall system design.

Images are captured by the micro-camera, and an automated multispectral light source helps to vary the illumination spectrum. The multispectral illumination is coupled to a single fibre and as part of the system control, the

illumination wavelengths are selected and synchronised with the detection by the imager and the video data transfer. The resulting raw video data is transmitted in a streaming-data fashion to the embedded processing unit which, in the future, will incorporate increasingly complex image analysis intelligence to assist the surgeon during the operation. Therefore, optimisation of the current implementation is essential in order to support future computation requirements.

Incorporating FPGA technology to surgical systems is an interesting choice when building innovation systems that are cost effective, have low-power consumption, and seek high performance. A multi-stage, FPGA-based customised design using similar image functionality was explored in [2] for the enhanced detection of blood vessels in retinal images. In another example, an FPGA platform was used in an endoscope imaging system [4] to provide a low-cost, high-performance implementation. A FPGA-based controller for robotic-assisted surgical system was developed in [7] to provide real-time control of a robot arms. Similar to our future plans, they aim to build the controller as a single chip, but clearly have different requirements.

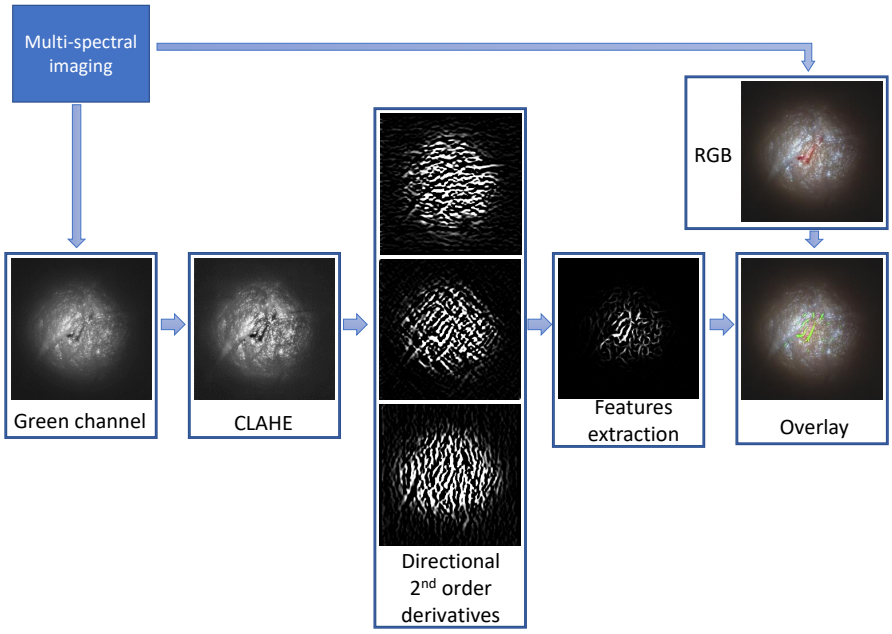


Fig. 3. Extraction of tissue features process from real imaging data captured by the multi-spectral imaging system developed by Tyndall Institute.

3 Extraction of Tissue Features

A key need is to identify key features in generated images such as in the extraction of tissue features (Fig. 3). Therefore, the proposed system employs the

widely-used CLAHE algorithm followed by a features extraction stage. It determines the directional second order derivatives of the enhanced image by computing the eigenvalues of Hessian matrix to decide whether a pixel is of interest or not. Overlying detected features on top of the hyperspectral images helps the surgeons to identify the ROIs. With the aim to incorporate future additional functionality, it is vital to minimise the FPGA resources by mapping effectively the required functionality onto the parallel FPGA resources and employing system level optimisations to produce the smallest footprint.

3.1 Image Acquisition

With conventional micro cameras, data is acquired with a rolling electronic shutter at a frame rate of 100 frames per second (FPS). By experimentation of the detection of blood vessels, it was determined that only the green channel needed to be extracted, because it has unique representations of the dark background and the bright retinal blood vessels. Each pixel stores a charge proportional to the light intensity and is converted to a digital value between 0 to 255 within the imager.

3.2 CLAHE

CLAHE is applied to enhance the contrast of each pixel based on its spatial location and neighboring pixels and has been shown to map well to FPGA [3]. A key stage of the algorithm is to divide the image into predefined and equal sized tiles where each tile is independent and does not share pixels with its neighboring tiles. A histogram for each tile is then obtained with 256 bins and clipped to a threshold predefined by the user (Fig. 5). All exceeded amounts are accumulated and redistributed uniformly to each bin which prevents noisy pixels from being enhanced by the Adaptive Histogram Equalization (AHE) process [6].

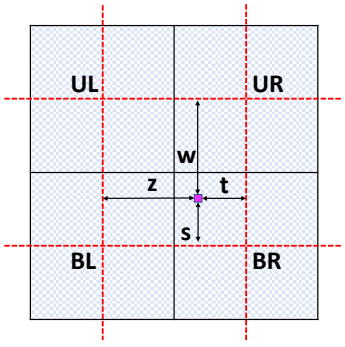


Fig. 4. Bi-linear Interpolation process

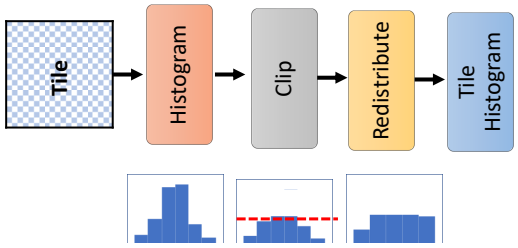


Fig. 5. Clipping, redistributing,and accumulating processes.

For each pixel value in the image, a cumulative distribution function (CDF) is generated using the cumulative sum of the redistributed histogram of each tile as below:

$$CDFt(p) = \sum_{n=0}^{p} \frac{h_t(n)}{z} \tag{1}$$

where p pixel in tile t, h_t is histogram of tile t, and z is the size of the tile. After that, a bi-linear interpolation of each pixel p, $I_{new}(p)$, is determined by three other CDFs of pixels from adjacent tiles, as shown in (2) and demonstrated graphically in Fig 4. This interpolation process reduces the impact of any interfering effects that will be generated at the framed boundaries.

$$\begin{aligned} I_{new}(p) = & \frac{s}{s+w}\left(\frac{t}{z+t}cdf_{UL}(p) + \frac{z}{z+t}cdf_{UR}(p)\right) + \\ & \frac{w}{s+w}\left(\frac{t}{z+t}cdf_{BL}(p) + \frac{z}{z+t}cdf_{BR}(p)\right) \end{aligned} \tag{2}$$

3.3 Convolution with Derivatives of Gaussian Kernel

Convolution with the second-order derivative of 2D Gaussian kernel estimates the directional gradients of the image and involves the construction of a Hessian matrix, H, of the image. The second-order partial derivative of 2D Gaussian kernels in x-direction, G_{xx}, xy-direction, G_{xy}, and y-direction, G_{yy}, are described in equations (3), (4) and (5) respectively.

$$\frac{\partial^2 G(x,y)}{\partial x^2} = \frac{1}{2\pi\sigma^4}\left[\frac{x^2}{\sigma^2} - 1\right] e^{-\frac{x^2+y^2}{2\sigma^2}} \tag{3}$$

$$\frac{\partial^2 G(x,y)}{\partial xy} = \frac{xy}{2\pi\sigma^6}\, e^{-\frac{x^2+y^2}{2\sigma^2}} \tag{4}$$

$$\frac{\partial^2 G(x,y)}{\partial y^2} = \frac{1}{2\pi\sigma^4}\left[\frac{y^2}{\sigma^2} - 1\right] e^{-\frac{x^2+y^2}{2\sigma^2}} \tag{5}$$

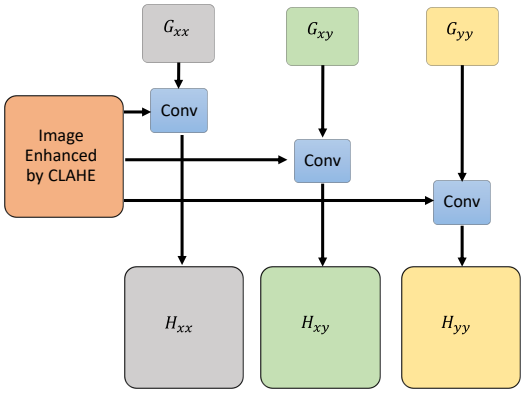


Fig. 6. Convolution with second-order derivatives of 2D Gaussian kernel

where x and y are integer values between $[-K, K]$, $K = 4\sigma + 1$ and σ is a scaling constant which affects the size and intensity of the Gaussian kernel. Any order derivative of a Gaussian kernel is also separable. This can be computation efficient since separable $(N \times N)$ kernels can be decomposed into horizontal and vertical kernels of size $(N \times 1)$ and $(1 \times N)$ respectively.

3.4 Constructing the Hessian Matrix

H is a square matrix which holds the directional second-order derivatives of an image, I, such as in equation (6). Each directional derivative of I can be determined by the convolution with the directional derivatives of Gaussian kernel (Fig. 6).

$$H = \begin{bmatrix} H_{xx} & H_{xy} \\ H_{yx} & H_{yy} \end{bmatrix} \tag{6}$$

The second-order derivative of I in the x-direction H_{xx} is given as equation (8) where $H_{xx} = I * G_{xx}$, $H_{xy} = I * G_{xy}$ and $H_{yy} = I * G_{yy}$. The Hessian matrix is symmetric since H_{xy} and H_{yx} are equal.

3.5 Eigenvalues of Hessian Matrix

For a given pixel point (x, y), the $H(x, y)$ is a 2×2 symmetric matrix which has two real eigenvalues. Therefore, determining eigenvalues at point (x, y) can be simplified using linear algebra as the following:

$$H(x, y) = \begin{bmatrix} h_{xx} - \lambda & h_{xy} \\ h_{xy} & h_{yy} - \lambda \end{bmatrix} = 0 \tag{7}$$

where $\lambda = \frac{h_{xx}+h_{yy} \pm \sqrt{(h_{xx}-h_{yy})^2 - 4h_{xy}^2}}{2}$, and $H_{xx}(x, y)$, $H_{xy}(x, y)$, and $H_{yy}(x, y)$ are represented as h_{xx}, h_{xy}, and h_{yy} respectively. From (7), it can be seen that

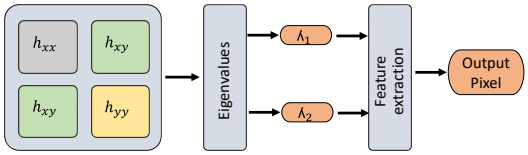


Fig. 7. Eigenvalues calculation and feature extraction

the two eigenvalues might be positive or negative real values and might be equal. Therefore, the inequality $|\lambda_2| \geq |\lambda_1|$ should always be satisfied for the purpose of this application before proceeding for a further process.

3.6 Feature Extraction or ROI Function

For images with darker features than the background, the edginess function, $F(x, y)$ in (8), is used to discriminate. It uses the eigenvalues of the corresponding Hessian matrix of a pixel (Fig. 7) to produces a value between [0,1], where values close to zeros are associated with the background and vice-versa. This gives,

$$F(x,y) = \begin{cases} (e^{-\frac{R_B^2}{2\beta^2}})(1 - e^{-\frac{S^2}{2c^2}}) & \lambda_2 > 0 \\ \\ 0 & otherwise \end{cases} \tag{8}$$

where $R_B = \lambda_2/\lambda_1$, $S = \sqrt{\lambda_2^2 + \lambda_1^2}$, $\beta = 0.5$, and $c = 15$ are used. We only implemented the traditional Hessian multi-scale filtering in [2] since the improved Hessian multi-scale enhancement filter requires calculations that involve all pixels of the image which impact the overall throughput and memory usage.

4 FPGA-Based Image Processing System Architecture

The AMD-Xilinx's Kria KV260 AI board and associated Vitis HLS 2021.2 toolset was used for initial implementation and design exploration. To optimise the image processing implementation, the data flow (DF) optimization was employed as it leads to solutions with lower memory usage and is applied by adding the DATAFLOW pragma in HLS. This ensures flawless data transfer from one function to the other and will support seamless integration of future, real-time functionality, as yet undefined.

Each processing step inside the system has to be linked to the next and/or previous processing step by internal streaming interfaces which will act as FIFO channels after C-synthesis. When using FIFOs, it is important to avoid deadlock, which can occur when depth is not specified correctly. A baseline system architecture was therefore established to allow stable system functionality and

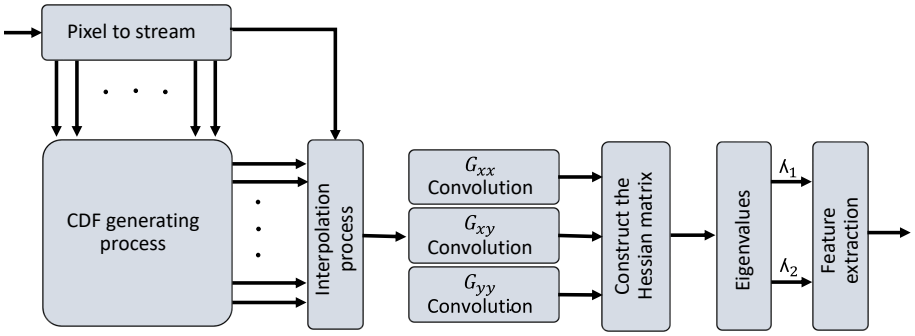


Fig. 8. Proposed FPGA-based image processing system architecture

then used as a reference design to evaluate the effectiveness of the applied optimisation techniques.

The baseline image processing system architecture (Fig. 8) comprises cascading processing elements (PEs) connected by streaming interfaces. The CLAHE is organised into the CDF generating processes, comprising large look-up tables (LUTs) which is connected to interpolating processes. The convolution process is applied as one PE, but it has small internal FIFO channels which help the separable convolution and border replication loops to be in-line when specifying the INLINE pragma in the DF optimisation flow. Finally, the Hessian matrix, determination and sorting of the eigenvalues, and the feature extraction functions, are combined into one PE, and termed the feature extraction process.

4.1 Experimental Setup

Vitis HLS 2021.2 is used for resource estimation and exporting RTL designs of the IP core while for place and route, we used Vivado 2021.2. For on-chip power, we used linux command "platformstats" and "timeit" package for timing analysis. The target FPGA platform is the Kria KV260 AI board (XCK26-SFVC784-2LV-C). For CPU/GPU evaluation, we used the Jetson Nano development kit which has a 1.43 GHz Quad-core ARM A57 as the CPU and 128-core Maxwell as the GPU. We chose to operate on 5W mode, and we used OpenCV/OpenCV-Cuda 4.1.1 implementations realised using Python on Jupyter notebook. For power analysis, we used "jetson-stats" package while for time analysis, we used "timeit" package. Both are imported as Python code to measure the performance for OpenCV implementations.

5 Evaluation

This section provides details of the baseline design and changes in resource utilisation and throughput after system- and algorithmic-level optimisations are ap-

plied. These are critical to ensure that sufficient FPGA real estate is available for future improved image analysis functionality and possible AI intelligence. The performance of the optimisations were investigated by assessing their impact in Matlab (Section 5.3).

As the CMOS sensor will be $240X240$ pixels and the system will need to operate at 100 FPS, each frame needs to be executed in 10 ms. If every pixel is executed for each clock cycle, this suggested a design with a 160 ns period time will need 9.216 ms for one $240X240$ frame to satisfy the design requirements.

Loops are pipelined using PIPELINE pragma with the minimum initiation interval (II) to satisfy period time by the HLS tool. Floating-point data types with single precision is the default for mathematical operations and variables in the baseline design. For CLAHE, the image is divided into a $8X8$ tile grid size, giving 64 independent regions. For the convolutions, we specify σ to be 2 which gives a $(19X19)$ filter size, based on Section 3.3.

5.1 Baseline design

The main hardware units in the FPGA are comprised of: a dedicated processing DSP48 (DSP) blocks including a 25-bit x 18-bit multiplier, a 48-bit adder and a 48-bit accumulator; a block RAM (BRAM) unit with 36 Kbits of data, configured as either two independent 18 Kb RAMs, or one 36 Kb RAM; a single bit flip-flip (FF) unit with pre-set/pre-clear functionality and; a 5-bit Lookup Table (LUT) which can be configured as logic, memory, or a shift register.

The baseline design of Fig. 8 was coded into three main functions, namely the CLAHE, convolution and the feature extraction processes. FIFOs were implemented as different dataflow objects. The resource breakdown resulted from the synthesis is given in Table. 1.

Table 1. Utilization estimates of baseline design on the Kria KV260 SOM involving a Zynq UltraScale+ (% of the resource is listed)

Process	BRAM	DSP	FF	LUT
FIFO	12 (4.2%)	0 (0.0%)	2836 (1.2%)	2695 (2.3%)
CLAHE	26 (9.0%)	76 (6.1%)	6179 (2.6%)	26689 (22.8%)
Convolutions	57 (19.8%)	565 (45.3%)	10209 (4.4%)	40141 (34.3%)
Feature ext,	0 (0.0%)	58 (4.6%)	1043 (0.4%)	5811 (5.0%)
Total	95 (32.0%)	699 (56.0%)	20267 (8.7%)	75336 (64.3%)

The convolution process has the highest resource allocation as a lot of processing is required. It uses the majority of DSP blocks and in addition, a large amount of BRAMs in order to implement the buffers used for the FIFOs and the efficient separable convolution processes. FIFOs consume almost 4.2% of BRAMs since a large depth is specified to avoid deadlock issues which can cause FIFOs with a small depth size between these loops and functions to be filled and thus blocking writing or reading to the FIFOs.

5.2 Convolution optimisations

In this section, optimisations were identified at the system level and algorithmic level with the aim of reducing the FPGA resources. At algorithmic level, we focused around exploiting common coefficients to employ common factor optimisation (CFO) to reduce the computational complexity [1, 9] and also change the order of the computation (re-ordering).

Hardware sharing: With flip-flops readily available in both the programmable logic and DSP48 blocks, pipelining can be used to increase the speed beyond that required and folding then applied to reduce the resources usage. This optimisation is available within the Vitis tools and was employed in the convolution function by applying feature II of pipelined loops. The unroll function produces multiple copies of the same function which are then folded onto one PE, leading to a reduction in the reources. This reduces the number of DSPs to 15%, FFs to 6%, and LUTs to 38% of the baseline design. However it also results in a reduction in the clock rate well below the desired value, in this case 27 MHz.

Common factor optimisation: The Hessian matrix requires the computation of three second-order directional derivatives of enhanced image (Fig. 9) requiring $3N$ multiplications and $3(N-1)$ additions and associated row buffers. The coefficients of $(N \times N)$ Gaussian kernels are determined by equations (3), (4) and (5) and can be decomposed into $(1 \times N)$ horizontal and $(N \times 1)$ vertical kernels. These coefficients will be fixed for the convolution process. If we expand the horizontal convolution expression in X direction, then for N is 19, this gives,

$$\begin{aligned} h_{xxh}x_n &= h_{xxh1}x_1 + h_{xxh2}x_2 + h_{xxh3}x_3 + h_{xxh4}x_4 + h_{xxh5}x_5 + h_{xxh6}x_6 \\ &+ h_{xxh7}x_7 + h_{xxh8}x_8 + h_{xxh9}x_9 + h_{xxh10}x_{10} + h_{xxh9}vx_{11} + h_{xxh8}x_{12} + h_{xxh7}x_{13} \\ &+ h_{xxh6}x_{14} + h_{xxh5}x_{15} + h_{xxh4}x_{16} + h_{xxh3}x_{17} + h_{xxh2}x_{18} + h_{xxh1}x_{19} \end{aligned} \tag{9}$$

This will require 19 multiplications and 18 additions. However, we can exploit the separability and symmetrical proprieties of Gaussian kernels and eliminate zero values. For the specific xx direction, $h_{xxh6} = h_{xxh7}$ and $h_{xxh8} = 0$. Exploiting this and exploiting the symmetry in equation (9), we can reorganise this specific computation into equation (10) as follows:

$$\begin{aligned} h_{xxh}x_n &= h_{xxh1}(x_1 + x_{19}) + h_{xxh2}(x_2 + x_{18}) + h_{xxh3}(x_3 + x_{17}) \\ &+ h_{xxh4}(x_4 + x_{16}) + h_{xxh5}(x_5 + x_{15}) + h_{xxh6}(x_6 + x_{14} + x_7 + x_{13}) \\ &+ h_{xxh9}(x_9 + x_{11}) + h_{xxh10}x_{10} \end{aligned} \tag{10}$$

This optimisation reduces the computation by 50% as only eight multiplications and sixteen additions are needed. When this is applied to the other horizontal and vertical directional convolutions, h_{xy} and h_{yy}, it reduces the number of DSPs by 50% compared to baseline process in Table 2 while still providing a throughput of 114 FPS. The figures for this revised implementation are listed as **CFO** in Table 2.

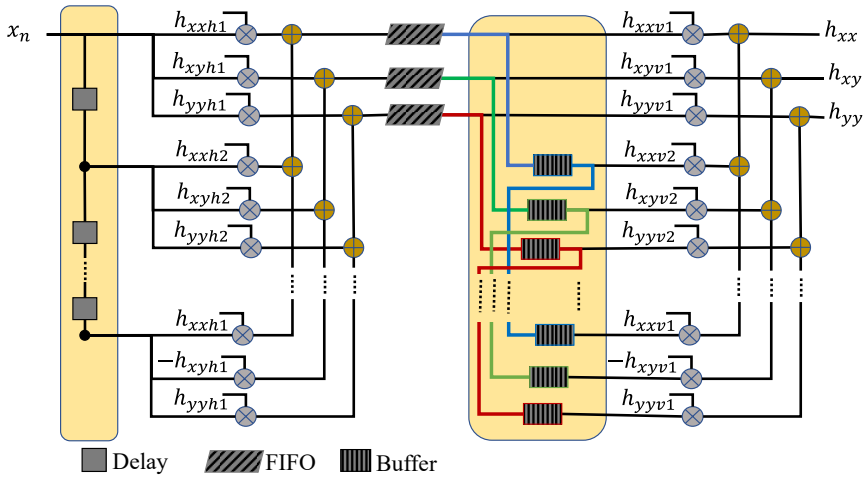


Fig. 9. Baseline separable convolution design for three different kernels combined

Re-ordering: The separable 2D convolutions can be decomposed into a horizontal followed by a vertical 1D-convolution each involving N multiplications and $N-1$ additions. The baseline design computes the horizontal followed by the vertical 1D-convolution, requiring the intermediate storage of $(3(N-1))$ buffers of length equal to an image row (Fig. 9), corresponding to $(N-1)$ buffers for each of the three different intermediate results produced by the horizontal convolutions. However, if we reverse the order of operation such that we start with vertical convolution, this will require only $(N-1)$ buffers since enhanced pixels coming from CLAHE process are shared between the different directional derivatives (Fig. 10). For (19×19) kernels, the number of BRAMs are reduced from 54 to 18 which more than 60% saving in convolution process. This **Reorder** design provides a 40% overall reduction in the baseline design in Table 2.

5.3 Combined Designs

In this section, we explore more about combined optimisations that would eventually build efficient designs that achieve high performance with minimal resource usage.

Combined: A more efficient design can be achieved by combining a number of these algorithmic optimisations. We first apply the **Re-order** optimisation to save in BRAMs usage and then the **CFO** option to reduce the complexity of the convolution function. This provides an additional saving in DSPs usage. The combined design (shown as **Combined** in Table. 2) achieved a 114 FPS which fulfils the design requirement of 100 FPS and with a lower BRAMs and

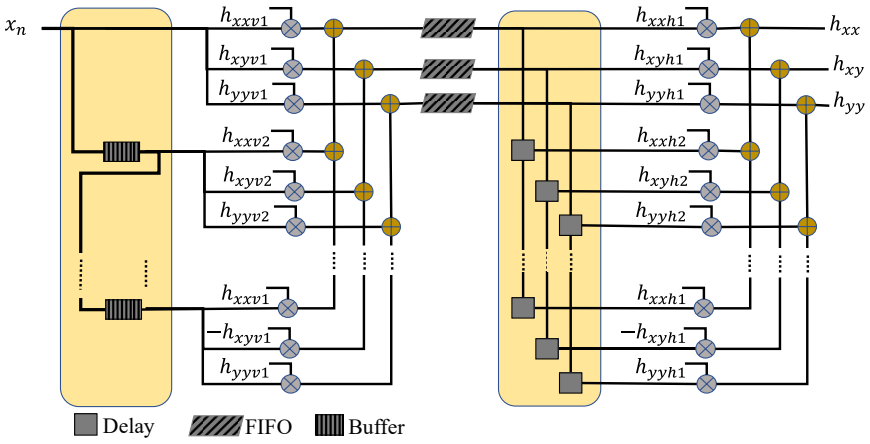


Fig. 10. Re-ordered separable convolution implementation for three kernels

DSPs resource usage. However, we expect a higher throughput after hardware implementation due to high-level optimisations by the Vivado tool.

Combined$^+$: Another obvious optimisation is to trade-off wordlength against resolution due to the small size of the original image. It is clear that a floating-point representation is unnecessarily large for circuit parts of the processing chain. For this reason, we changed the data type for the **Combined** design from 32-bit floating-point to 18-bit fixed-point arithmetic in the convolution only. This is organised into a 10-bit integer with the remaining 8 bits used for the fractional part.

Sufficient performance quality or quality of results (QoR) was ensured by assessing experimentally the visual impact and also measuring the structural similarity index measure (SSIM) and image quality degradation by peak signal-to-noise ratio (PSNR). Our results indicate that the 18-bit fixed-point representation scored on average 0.985 SSIM index and 40dB PSNR value compared to 32-bit floating-point results. The resource utilization labelled as **Combined$^+$** is presented in Table. 2 with the biggest saving is in DSP units where the 5 DSPs of the floating-point can be reduced to a single DSP for the 18-bit fixed-point representation. There is also been a small reduction in the number of flip-flops and LUTs needed. As expected, there has only been a minimal change in throughput, 121 FPS, due the consistent use of pipelining.

5.4 Implementation Comparison

It is worth considering the performance issues when compared to a GPU implementation. For this reason, the same design was implemented on GPU and

Table 2. Resource utilization of optimized designs on (Kria KV260 SOM)

Design	BRAM (288)	DSP (1248)	FF (234240)	LUT (117120)	FPS
Baseline	95 (32%)	699 (56%)	20507 (8%)	76704 (65%)	114
CFO	95 (32%)	417 (33%)	14879 (6%)	58623 (50%)	114
Re-order	59 (20%)	684 (54%)	25055 (10%)	76240 (65%)	114
Combined	59 (20%)	422 (33%)	19479 (8%)	59480 (50%)	114
Combined$^+$	59 (20%)	142 (11%)	13687 (5%)	49426 (42%)	121

initial results generated and compared to the best FPGA realisation. In the GPU realisation, all optimisations were applied to ensure a high quality design. For example, conditional statements used for sorting eigenvalues and in feature extraction function had to be implemented solely using OpenCV-Cuda commands as it does not have ready-to-use functions for this. This allowed the GPU implementation to have a very fast execution compared to CPU.

The resulting performance figures are shown in Table 3. As expected, the GPU outperforms the CPU. However, the optimisation implemented in mapping the design to FPGA has resulted in higher throughput when compared to the GPU. The solid FPGA performance is largely achieved due to small image size. The smaller size has resulted in an effective utilisation of the on-board FPGA resources and avoided having to undertake off-chip memory accesses which would compromised the performance. Use of the efficient design allows a throughput rate that is nearly 1.4× as fast as the GPU. The lower power performance of the FPGA device comes to the fore, but the overall System-on-Module (SoM) power consumption results in comparable GPU FPS/W figure. This would be much better if we use the single FPGA figure.

Table 3. Throughput and power comparison

	Image size	Jetson Nano Developer Kit		Kria Vision AI Starter Kit	
		ARM A57 (CPU)	Maxwell (GPU)	Zynq UltraScale+MPSoC (FPGA)	
				Baseline	Combined$^+$
Time (ms)	240X240	20.50	12.10	8.73	8.26
FPS	240X240	48.85	82.41	114.4	121
FPS/W	240X240	17.0	29.42	30.59	33.15

6 Conclusions

The paper presents details of an FPGA implementation of an intelligent surgical imaging system based on a 'chip on tip' camera system. The key challenge is to be able to implement a low power embedded processing unit to be able

to enhance the image quality and in the future, provide increased intelligence. Results were presented on the implementation of the contrast limited adaptive histogram equalization to suppress the contrast of each pixel based on its neighbouring values, convolution procedures using the derivatives of Gaussian and the construction of the Hessian matrix of each pixel with the eigenvalues used by a ROI function. We are able to demonstrate savings in DSP processor resources by up to 80% without a non-discernible loss in image quality. The work to date has been important in ensuring that available FPGA real estate is created so that the user can incorporate future functionality. Future work is targeted at providing much more intelligence into the embedded system which will provide detection capability for the surgeon. This will focus around building up a knowledge of existing operations and providing embedded training on the device.

References

1. Bailey, D.G.: Design for embedded image processing on FPGAs. John Wiley & Sons (2011)
2. Elbalaoui, A., Fakir, M., Taifi, K., Merbouha, A.: Automatic detection of blood vessel in retinal images. In: 2016 13th International Conference on Computer Graphics, Imaging and Visualization (CGiV). pp. 324–332 (2016). https://doi.org/10.1109/CGiV.2016.69
3. Honda, K., Wei, K., Arai, M., Amano, H.: Clahe implementation and evaluation on a low-end fpga board by high-level synthesis. IEICE Transactions on Information and Systems **E104D**(12), 2048–2056 (2021). https://doi.org/10.1587/transinf.2021PAP0006, publisher Copyright: Copyright © 2021 The Institute of Electronics, Information and Communication Engineers
4. Liu, X., Li, L.: Fpga-based three-dimensional endoscope system using a single ccd camera. In: 2015 IEEE International Conference on Information and Automation. pp. 614–618 (2015). https://doi.org/10.1109/ICInfA.2015.7279360
5. Pizer, S., Johnston, R., Ericksen, J., Yankaskas, B., Muller, K.: Contrast-limited adaptive histogram equalization: speed and effectiveness. In: [1990] Proceedings of the First Conference on Visualization in Biomedical Computing. pp. 337–345 (1990). https://doi.org/10.1109/VBC.1990.109340
6. Pizer, S.M., Amburn, E.P., Austin, J.D., Cromartie, R., Geselowitz, A., Greer, T., ter Haar Romeny, B., Zimmerman, J.B., Zuiderveld, K.: Adaptive histogram equalization and its variations. Computer vision, graphics, and image processing **39**(3), 355–368 (1987)
7. Taghizadegan, A., Piltan, F., Sulaiman, N.B.: Design high frequency surgical robot controller: Design fpga-based controller for surgical robot manipulator simscape modeling. International Journal of Hybrid Information Technology **9**(5), 431–474 (2016)
8. Wäny, M., Voltz, S., Gaspar, F., Chen, L., Tecnopolo, A.L.M.: Ultrasmall digital image sensor for endoscopic applications. In: Proc. Int. Image Sensor Workshop. pp. 1–3 (2009)
9. Woods, R., McAllister, J., Lightbody, G., Yi, Y.: FPGA-based implementation of signal processing systems. John Wiley & Sons (2008)