FPGA ARCHITECTURE FOR REAL-TIME BARREL DISTORTION CORRECTION OF COLOUR IMAGES

Henryk Blasinski

ECE Deptartment
University of Rochester, NY
henryk.blasinski@rochester.edu

Wei Hai [*], *Frantz Lohier* [†]

Logitech Inc.
6505 Kaiser Drive
Fremont, CA

ABSTRACT

This paper presents a hardware architecture for real time barrel distortion correction of YUV 4:2:2 encoded color images. In our solution we are presenting an alternative implementation of the correction engine which is based on multiplications rather than trigonometric transforms. The system makes minimal use of resources thus being a good candidate for embedding into a single chip. Proposed solution is implemented in a cost-effective Spartan 3 Field Programmable Gate Array (FPGA). Our architecture is capable of processing QQVGA images at the rate of 30 frames per second (fps). Qualitative and quantitative examination confirm correct preservation of color information in processed images.

***Index Terms*—** real-time barrel distortion correction, frame buffer, Field Programmable Gate Array (FPGA), YUV 4:2:2

1. INTRODUCTION

In recent years digital cameras have become omnipresent in our daily lives either as stand alone devices or embedded into mobile phones, cars (back-up cameras) or surgical equipment (laparoscopes). Typically such cameras have a field of view (FOV) of about $60°$, which is only a third of the FOV of the human visual system. On the other hand there exist lenses covering much larger FOVs, such as wide-angle ($120°-130°$) or fish-eye ($\sim 180°$). They exhibit, however, a significant amount of barrel distortion, making them less appropriate for high volume, consumer applications. Large FOV optics is present in certain niche markets including: automotive, medical and surveillance industries [1].

The barrel distortion has been extensively studied and several correction algorithms have been proposed [1–3]. Also in recent years some work has been done on hardware implementations of those algorithms, however, authors focused on functionality rather than resource constraints [4–8]. Moreover only grayscale images have been taken under consideration, significantly reducing their impact on consumer applications.

[*]Wei Hai is currently with Contour Inc.; wei@contour.com

[†]Frantz Lohier is currently with Kudelski Group; frantz@lohier.com

In this paper we are presenting an FPGA architecture for real-time barrel distortion correction of colour images encoded in the YUV 4:2:2 format which is frequently used in the industry [9]. In addition to this we are proposing an alternative implementation of the correction engine, allowing for more efficient utilisation of resources available on modern FPGAs.

2. BARREL DISTORTION CORRECTION

Barrel distortion is one of the most common imperfection of lenses, it causes straight lines from the real world to be represented as curves in the image. It can be regarded as a geometric image transformation. It converts an undistorted image I with pixel coordinates (x, y) into a distorted image I_d with coordinates (x_d, y_d). The goal of the distortion correction is to perform an inverse operation that is to find a corrected image I_c with coordinates (x_c, y_c) such that $I_c \approx I$.

2.1. The polynominal method

According to [10] the barrel distortion can be decomposed into two primary components, tangential and radial. The radial component is affecting the distance R between any point in the image and the optical centre. The tangential component is affecting the angle θ between the line joining given point and the projection centre. The relationship between distorted values R_d and θ_d and corrected ones R_c and θ_c is best described by a polynominal of the n^{th} order:

$$\theta_c = \sum_{k=0}^{n} a_k \cdot \theta_d^k = p(\theta_d) \tag{1}$$

$$R_c = \sum_{k=0}^{n} b_k \cdot R_d^k = q(R_d) \tag{2}$$

Lenses can be considered symmetric with respect to their optical centre. This means that $a_0 = b_0 = 0$ and the tangential distortion is negligible: $\theta_c = \theta_d$, making the radial distortion the only component needed to be accounted for. Such assumptions have been made in many earlier works [3–8]

Hardware implementation of the barrel distortion correction algorithm can be done in several ways. The simplest approach is to explicitly encode the relationship between all distorted and corrected image coordinates in a form of a large look-up table [5]. This system may be very fast, however for larger image sizes it has significant memory requirements. The second approach uses a CORDIC engine to perform polar to rectangular conversion and implements polynominal mapping between distorted and corrected radii, as described by (2) [7, 8]. In this case, the main constraint is the FPGA chip size, since CORDIC engine requires significant amount of logic.

2.2. The coefficient method

The major disadvantage of the polynominal method is the necessity of performing rectangular to polar and polar to rectangular transforms [8]. We would like to propose a different formulation of the polynominal method allowing to avoid those transitions. It is based on the assumption that the tangential distortion is negligible ($\theta_c = \theta_d$).

The polynominal method describes the relationship between a point in the disorted domain and its equivalent in the corrected domain: $(x_d, y_d) \rightarrow (x_c, y_c)$. For hardware applications it is much more convenient to perform an inverse operation: $(x_c, y_c) \rightarrow (x_d, y_d)$ that is for a particular point in the corrected image, calculate its coordinates in the distorted image space. This approach, called the back mapping [7], relates radii as: $R_d = m(R_c)$. In many applications the function $m(R_c)$ is also approximated by polynominals. Additionally the back mapping is more convenient for implementing image interpolation techniques [7]. From the congruence of coordinate triangles (Fig. 1) in both domains we may write:

$$\frac{R_d}{x_d} = \frac{R_c}{x_c} \tag{3}$$

$$\frac{R_d}{y_d} = \frac{R_c}{y_c} \tag{4}$$

after some rearrangements one obtains:

$$x_d = x_c \cdot \frac{R_d}{R_c} = x_c \cdot C_f|_{(R_c, R_d)} \tag{5}$$

$$y_d = y_c \cdot \frac{R_d}{R_c} = y_c \cdot C_f|_{(R_c, R_d)} \tag{6}$$

Above equations reveal that in order to obtain (x_d, y_d) coordinates in the distorted image space it is enough to multiply both undistorted image coordinates (x_c, y_c) by the same coefficient C_f. The value of this coefficient depends upon the ratio between distorted and undistorted radii. In our approach we propose to create a function relating the square of undistorted radius R_c with the correction coefficient C_f:

$$C_f = g(R_c^2) \tag{7}$$

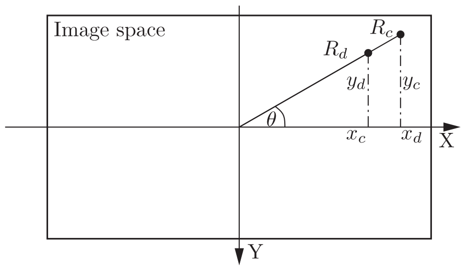


Fig. 1. Relationship between corrected (x_c, y_c) and distorted (x_d, y_d) coordinate spaces.

Such an approximation, while not resulting from mathematical transformations, will greatly facilitate hardware implementation. R^2 can be easily calculated from the Pythagorean Theorem. The function $g(R_c^2)$ should be approximated with such an expression that would facilitate hardware implementation. If for example a polynominal function is used all calculations of the C_f coefficient, and distorted image coordinates are brought down to muiltiplications and additions only, replacing complex polar-rectangular coordinate system transforms.

3. SYSTEM ARCHITECTURE

The overall system architecture may be decomposed into two parts. Image acquisition stage manages writing image data into the frame buffer. On the basis of sensor signals, for each incoming byte its coordinates (x_d, y_d) are computed. They are used in later stages to calculate frame buffer addresses. The image regeneration stage is responsible for creating the corrected image and sending it to the external ASIC. Firstly corrected image coordinates (x'_c, y'_c) are generated, then they are mapped to their distorted counterparts (x'_d, y'_d). On their basis the frame buffer address is calculated and data read. The final block is responsible for recreating appropriate sensor-ASIC interface signals (Fig. 2). In general case there is no dependance between input and output image resolutions, however for simplicity it shall be assumed that they are the same. As a result the output image, though without distortion, will have a slightly lower FOV.

3.1. YUV 4:2:2 subsampling

In many computer applications instead of encoding colour images using 24 bits per pixel (8 bits per channel) the colour information (chrominance) has a twice lower sampling frequency than luminance thus reducing the bandwidth by a third [9].

In a properly encoded YUV 4:2:2 image, there is a predefined sequence of Y (intensity) and Cb, Cr (chrominance) data. Each pixel has its own intensity value Y and one of the two remaining components Cb or Cr, the other being as-

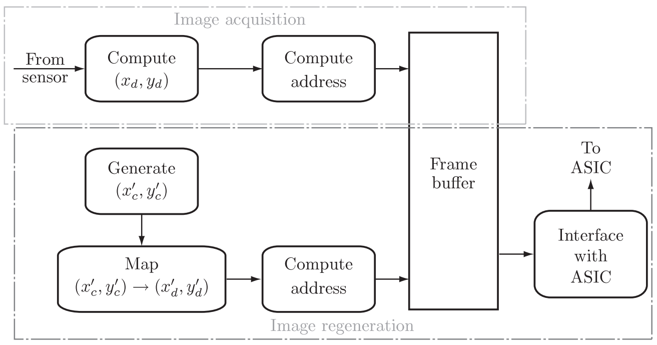


Fig. 2. Principal blocks and data flow of the barrel distortion correction arhitecture.

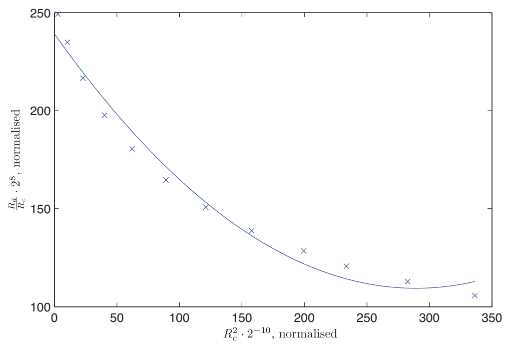


Fig. 3. An example implementation of equation 8 . Points mark experimental data, solid line is their 2^{nd} order polynominal approximation.

Table 1. Mapping engine coefficients, QQVGA resolution.

Lens type	Scaling		2^{nd} order polynominal		
	m	n	a	b	c
Wide-angle	8	-6	0.0031	-1.0331	251.32
Fish-eye	8	-10	0.2272	-10.863	239.24

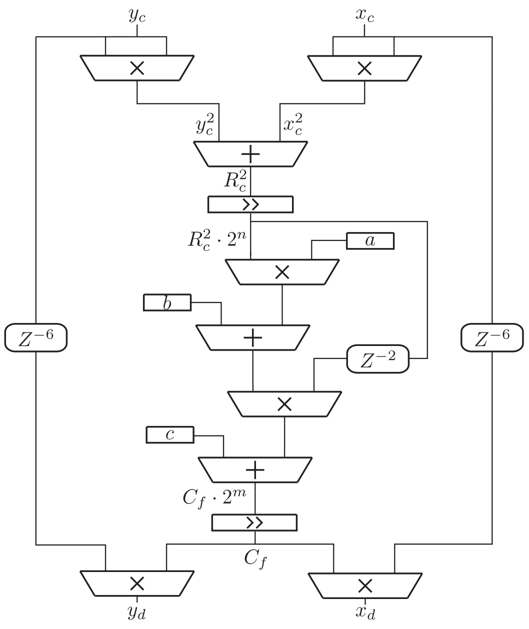


Fig. 4. Hardware realisation of the mapping engine. 'x' - multiplications; '+' - additions, '>>' - logical shift right; 'Z^{-n}' - n cycle delay element.

signed to the neighboring pixel. For an image line this gives a sequence of bytes: $Y_1, Cr_1, Y_2, Cb_1, Y_3, Cr_3, Y_4, Cb_3, ...$ A typical frame buffer stores one pixel per address, hence when accessing pixels independently, as in the case of barrel distortion correction, it often happens that this sequence is no longer restored. The most common situation is the replacement of Cr data by Cb (or vice versa). An example sequence of bytes which results in colour conversion is: $Y, Cr, Y, Cr, Y, Cr, ...$

3.2. Radius mapping

Our architecture uses the radii mapping method described in section 2.2. Targeting hardware implementation, the function (7) is approximated with a 2^{nd} order polynominal. It is important to notice that the correction coefficient C_f is always smaller than 1 and input may be as high as 10000 (maximum R_c^2 for QQVGA resolution). For this reason it is better to represent the mapping function as:

$$C_f \cdot 2^m = h(R_c^2 \cdot 2^n) \qquad (8)$$

In this way polynominal coefficients will have higher numerical values which makes them less sensitive to finite precision arithmetics and easier to compute. Polynominal approximation of function (8) can not be found analitically, numerical computations are necessary. Starting from (2) we compute several sample points, they serve as reference data for least squares approximation with a polynominal (Fig. 3). Table 1 lists coefficients found for two lens types (wide-angle and fish-eye) and values of m and n used. Even though the function (8) does not produce a zero C_f for zero radius, this does not matter since in such a case $x_c = 0, y_c = 0$ and equations (5) and (6) still hold.

Fig. 4 presents a block diagram of the hardware realisation of the coordinate mapping block. Each coordinate is raised to the 2^{nd} power, they are added together to obtain the square of the radius. Logical shift operations are obtained by hard wiring appropriate bus lines of adder outputs. Two final multipliers modify input coordinates and produce distorted coordinate pair.

3.3. Half-frame SRAM buffer

The correction scheme does not require access to all image pixels at any given instance of time. Storing just a part of the image, for example a half, reduces the memory requirement

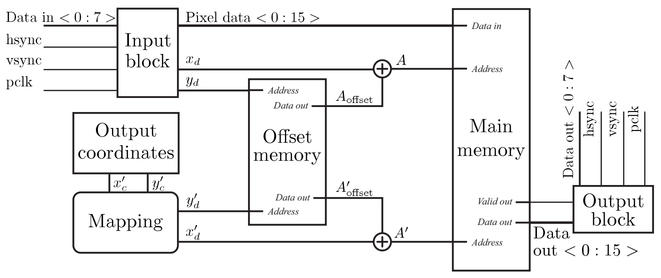


Fig. 6. SRAM memory buffer architecture. For each incoming pixel (x_d, y_d) its main memory address A is calculated. For every output image pixel (x'_c, y'_c) its counterpart in the distorted domain (x'_d, y'_d) is found and address A' determined.

with no influence on the algorithm.

Suppose that the frame has X columns and Y lines. In the half-frame buffer approach we are proposing to create a buffer having the size of $\frac{Y}{2} \cdot X$ memory words. When using YUV 4:2:2 encoding each word is 16 bits long. The starting address of an image line is calculated as y modulo $\frac{Y}{2}$. Incoming image lines from 1 to $\frac{Y}{2}$ (upper half of the image) are stored in assigned memory blocks. The line $\frac{Y}{2}+1$ (1^{st} line of the lower image half) is stored in the memory block assigned to line 1, the line $\frac{Y}{2}+2$ in memory location assigned to line 2 etc..., replacing the contents of the buffer with the lower image half (Fig. 5).

In order to implement this memory allocation scheme, a special address calculation method is necessary. Suppose that an incoming data corresponds to a pixel placed at (x, y) in the image, where x denotes the column and y the line. The address A may be determined as the sum of the line offset from the beginning of the frame buffer A_{offset}, and the column offset from the beginning of the line. The column offset is actually equal to the x coordinate, hence:

$$A = A_{offset} + x \tag{9}$$

Offset values A_{offset}, although constant for a particular resolution, are a function of the line number y, which is why a simple adder is insufficient. They may be stored in a small look-up table, accessed on the basis of the current line number value (Fig. 6).

This system does not have to operate at elevated frequencies. The input pixel clock *pclk* rate is about 27 MHz. All modules within the design operated at a slightly elevated frequency of 30 MHz. The last block in the data flow *Output block* used an even higher frequency of 37 MHz. Since every next block in the data flow is not slower than its predecessor FIFO and buffer overflows are prevented.

3.4. SRAM Colour restoration

To correctly restore colours it is necessary to assure appropriate arrangement of chrominance information in the data stream. A traditional 16 bit wide memory block should be replaced with two 8 bit memories, the total capacity does not change. Each of those blocks has the same addressing space as a single 16 bit block. Both of those blocks will be connected to the same address line, however one would store luminance information only (first 8 bits of the 16 bit word) while the other chrominance information (remaining 8 bits).This memory rearrangement does not influence the image acquisition stage of the system.

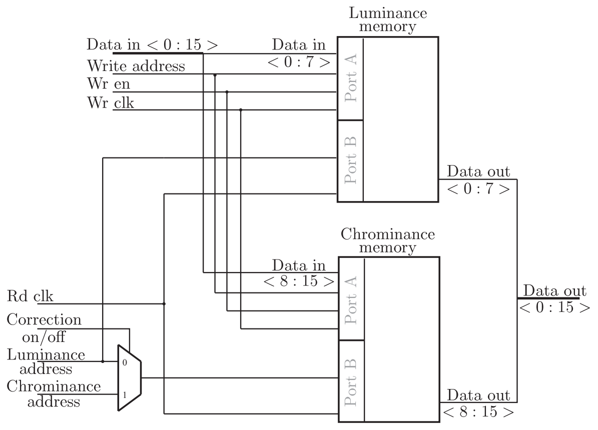


Fig. 7. Half frame buffer memory separation for luminance and chrominance data. Write address is common for both blocks, for reading data there are two separate addresses: *Luminance address* and *Chrominance address*. If the colour correction is off, both memory blocks are connected to *Luminance address* line.

More changes are necessary to be introduced to the image regeneration stage. Since we know the pixel location in the output image (x_c, y_c), we can easily determine if Cb or Cr data is needed for this particular pixel. This depends upon the parity of the x_c coordinate only, even numbers correspond to Cb and odd to Cr. Once the coordinate transformation is performed a pair of distorted image coordinates (x_d, y_d) is obtained. We can now compare the parities of x_c and x_d. If they are the same, appropriate colour information will be read from the buffer, if they are not, chrominance data will get replaced. Placement of luminance and chrominance into separate memories allows for an independent access to data, as a function of the parity accordance signal (Fig. 7). If parities of x_c and x_d match, then luminance and chrominance addresses do not change, in an opposite case, the chrominance address is incremented by 1.

4. RESULTS

4.1. Experimental setup

The architecture described above was implemented in hardware. The setup consisted of commercially available parts: a 2 megapixel 1/3.2 inch MT9D111 sensor by Aptina, a Xilinx

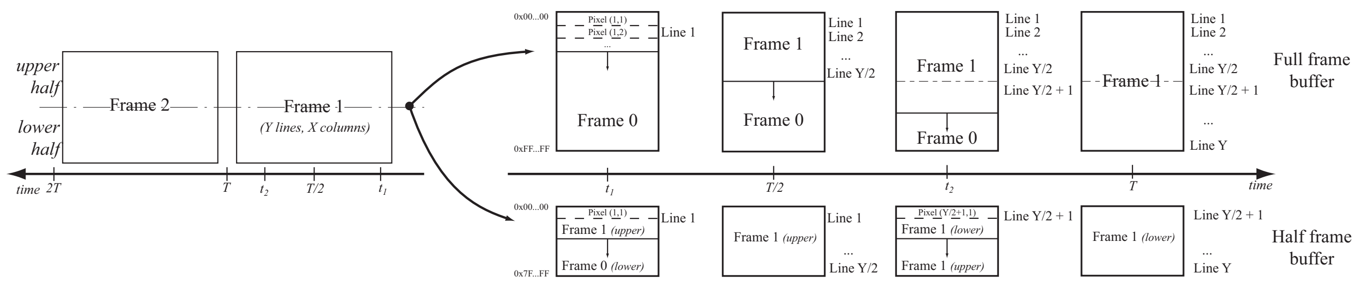


Fig. 5. Comparison between traditional full frame and a half frame buffer. In the latter case, the memory is written into twice per frame time.

Table 2. Comparison between mapping methodologies (in pixels).

Lens type	$\epsilon = R_{class} - R_{coeff}$		
	$\overline{\lvert\epsilon\rvert}$	$\sigma_{\lvert\epsilon\rvert}$	$max(\lvert\epsilon\rvert)$
Wide-angle	0.35	0.2	2.28
Fish-eye	1.51	0.89	2.5

Spartan 3 FPGA and an ASIC chip for USB interface with the PC. Designs were created using Xilinx ISE 9 IDE with System Generator plugin for Matlab. The Spartan 3 family is a low cost low power solution designed for high volume applications. The chip used in our design, Spartan-3 700AN, has about 13000 logic cells, 20 18 bit embedded multipliers and 20 memory blocks 18 kbits each, giving a total of 360 kbits of memory.

4.2. Coefficient method evaluation

The coefficient method has two main sources of errors. The first one is related to fixed point arithmetics and generally can be avoided if sufficiently wide buses are used. In our case we established that this error is negligible if the fractional part is stored on 10 bits.

The second, more important source of error is related to the mapping methodology. The function (8) is not a result of mathematical transformations of (2) but it is obtained via curve fitting to sample points. For every pixel in the image its corresponding distorted radius was calculated using two methods: classical R_{class} (section 2.1), and the coefficient method R_{coeff} (section 2.2). For each pixel i the error was calculated as: $\epsilon_i = (R_{class})_i - (R_{coeff})_i$. The first column in table 2, $\overline{|\epsilon|}$, is the average of the absolute error values, the second one $\sigma_{|\epsilon|}$, is the corresponding standard deviation. The final column $max(|\epsilon|)$ is the maximal absolute error. Results confirm that this methodology is best suited for wide-angle type distortion correction.

4.3. Buffer implementation

The rationale behind half-frame buffer is to maximally limit the memory requirement. If sufficiently small it might be

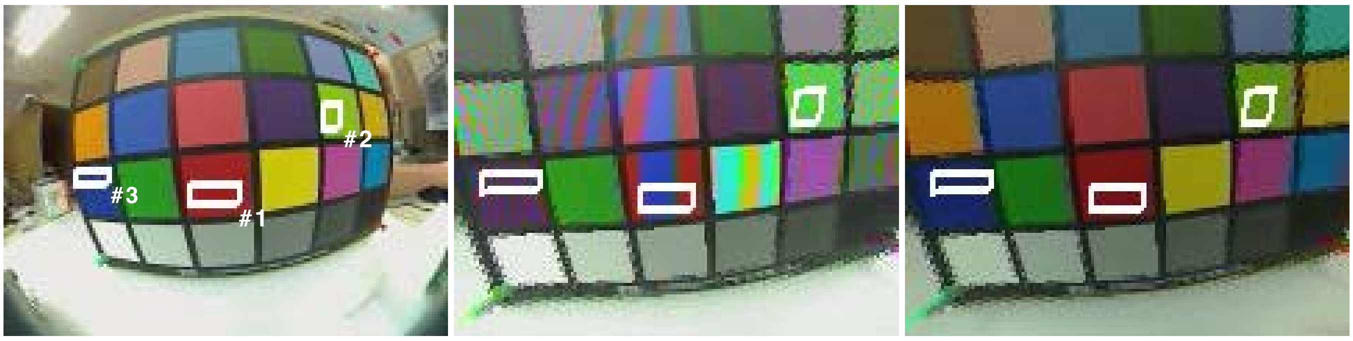
Table 3. Spartan-3 700AN resources utilisation.

Module	Logic slices [%]	Memory blocks [%]	Multipliers [%]
Input block	2%	5%	0
Data offset	<1%	0	0
Offset memory	0	5%	0
Data address	<1%	0	0
Main memory	<1%	60%	0
Coordinate generator	7%	0	50%
Output block	3.5%	10%	0
Total	**14%**	**80%**	**50%**

possible to benefit only from FPGA on-chip memory blocks. This particular circuit has very limited memory resources, hence it was only possible to provide support for QQVGA (120 x 160) resolution at 30 fps. Table 3 summarises FPGA utilisation, where the memory is clearly the bottleneck. It is used in 80% for the frame and I/O buffers (a total of 288 kbits). 10 multipliers are needed to implement the coordinate mapping scheme. The remaining circuitry uses only 14% of FPGA resources (about 1800 logic cells).

4.4. Colour restoration

Colour restoration scheme assures appropriate preservation of original colours. Fig. 8 presents three images, original one obtained with a fish-eye lens (Fig. 8(a)), an image with barrel distortion removed and without colour restoration (Fig. 8(b)), finally, the third image implements the colour restoration mechanism (Fig. 8(c)). Images show three regions of interest used for quantitative analysis. Each of them has a single channel dominating over the remaining ones. Quantitative data for those regions, listed in table 4, confirms qualitative results and proves that the colour representation of the proposed mechanism is more faithful. The improvement for region #3 is minor, this however may be the result of pixel repetitions in the distortion correction algorithm due to nearest neighbour interpolation, perceptually the improvement is much more pronounced.



(a) Original fish-eye (b) Distortion removed, no colour restoration (c) Distortion removed, with colour restoration

Fig. 8. Qualitative evaluation of colour restoration mechanism.

Table 4. Quantitative analysis of colour restoration algorithm.

Region	Average intensity	R	G	B
#1	Original	**156**	34	32
	Not corrected	85	55	113
	Corrected	**113**	19	20
#2	Original	147	**149**	33
	Not corrected	93	199	77
	Corrected	92	**140**	14
#3	Original	101	102	**110**
	Not corrected	78	53	90
	Corrected	21	40	**91**

5. CONCLUSIONS

This paper proposes an architecture for real time barrel distortion correction of YUV 4:2:2 encoded colour images. It is not specific to this particular application, and may be used in other geometric image processing algorithms on resource constrained platforms. By using the FPGA as a co-circuit it was possible to use standard sensor and USB-ASIC chips. As a result the final system preserved the plug and play capability of a typical computer peripheral.

The architecture aims at reducing the external memory size by implementing a half-frame buffer using FPGA internal memory blocks only. This solution was tested for QQVGA resolution at 30 fps. It is important to note that for this particular approach not all image data is available at any given instance, which may limit the applicability of other image processing mechanisms. The colour correction scheme counteracted the misplacement of Cb and Cr data in the output stream. Experimental results proved a very good quality of colour restoration, both qualitatively and quantitatively.

Finally we have proposed an alternative implementation of the coefficient mapping engine, as opposed to logic or memory intensive solutions it aims at profiting from multipliers available in large majority of current FPGA chips. The coefficient method delivers comparable image quality as previously developed implementations.

6. REFERENCES

[1] K. Wonjun and K. Changick, “An efficient correction method of wide-angle lens distortion for surveillance systems,” in *in Proc. IEEE Intl. Symp. on Circuits and Systems ISCAS*, May 2009, pp. 3206–3209.

[2] F. Devenray and O. Faugeras, “Straight lines have to be straight,” *Machine Vision and Applications*, vol. 13, no. 1, pp. 14–24, August 2001.

[3] J. Courbon, Y. Mezouar, L. Eck, and P. Martinet, “A generic fisheye camera model for robotic applications,” in *in Proc. IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems*, October-November 2007, pp. 1683–1688.

[4] H. Ngo and V. Asari, “Developing a FPGA-based high performance, power-aware architecture for real-time radial lens distortion correction of video stream,” *ICGST Intl. J. on Programmable Devices, Circuits and Systems, PDCS*, vol. 7, pp. 33–41, 2007.

[5] A. Hernandez, A. Gardel, L. Perez, I. Bravo, R. Mateos, and E. Sanchez, “Real-time image distortion correction using FPGA-based system,” in *in Proc. IEEE Annual Conf. on Industrial Electronics, IECON*, November 2006.

[6] K. Gribbon, C. Johnston, and D. Bailey, “A real-time FPGA implementation of a barrel distortion correction algorithm with bilinear interpolation,” in *in Proc. of Image and Vision Computing New Zealand*, November 2003, pp. 408–413.

[7] H. T. Ngo and V. K. Asari, “A pipelined architecture for real-time correction of barrel distortion in wide-angle camera images,” *IEEE Trans. on Circuits and Systems for Video Technology*, vol. 14, no. 3, pp. 436–444, March 2005.

[8] L. Qiang and N. Allinson, “FPGA implementation of pipelined architecture for optical imaging distortion correction,” in *in Proc. IEEE Wkshp. on Signal Processing Systems Design and Implementation SIPS*, oct. 2006, pp. 182 –187.

[9] K. Jack, *Video Demystified: a Handbook for the Digital Engineer*. Elsevier, 2007.

[10] S. Shah and J. Aggarwal, “A simple calibration procedure for fish-eye (high distortion) lens camera,” in *in. Proc. IEEE Intl. Conf. on Robotics and Automation*, no. 4, May 1994, pp. 3422–3427.