*Projects*, Vol. 10, 2003 ISSN 1172-8426



# A REAL-TIME FPGA IMPLEMENTATION OF A BARREL DISTORTION CORRECTION ALGORITHM

**C T Johnston, D G Bailey**

**Abstract:** This paper presents a novel FGPA implementation of a barrel distortion correction algorithm with a focus on reducing hardware complexity. In order to perform real-time correction in hardware the undistorted output pixels must be produced in raster order. To do this the implementation uses the current scan position in the undistorted image to determine which pixel in the distorted image to display. The implementation employs the use of a look-up table with interpolation to reduce the complexity of the hardware design, without significant loss of precision. The paper details the background of the hardware and design environment used, and then barrel distortion and the model selected for correcting it. The implementation aspects are discussed and the results of the implemented design are shown. This leads to a discussion on future work and conclusions.

**Keywords:** FPGA, reconfigurable hardware, lens distortion, camera calibration

## 1 INTRODUCTION

Image processing is often used to make non-contact measurements for real-time measurement applications. It is therefore vital to ensure that accurate measurements can be made from the captured images. If the use of an analogue camera is employed, the camera must often be of greater resolution and quality than is needed for the particular application in order to compensate for losses incurred before digitisation of the image [1]. A digital camera facilitates early digitisation and therefore it is less crucial to compensate for these losses. This can lead to substantial cost savings since a digital camera with lower resolution can be used. However, the inexpensive and wide-angle lenses often used in low cost digital cameras are susceptible to barrel distortion, which can introduce significant errors into any measurements [2].

Barrel distortion occurs when the magnification at the centre of the lens is greater than at the edges. A higher quality lens can be used to correct for this but this comes at additional cost to the image capture system. Barrel distortion is primarily radial in nature, with a relatively simple one parameter model accounting for most of the distortion [3]. A cost effective alternative to an expensive lens is to algorithmically correct for the distortion using the model.

A microprocessor or DSP could be used as the implementation platform for such an algorithm if processing were done offline. However, in order to satisfy the operational constraints imposed by real-time processing at sixty frames per second the algorithm must be implemented in hardware. A fixed hardware approach using an application specific integrated circuit (ASIC) would have flexibility limitations making field programmable gate arrays (FPGA) a better choice.

An FPGA is a matrix of logic blocks that are connected by switching blocks. Both the logic blocks and the switching blocks can be programmed to build specific hardware. Some devices also have block RAM that can be used for on chip storage. These connections, BlockRAMs and logic functions can be reprogrammed for different applications to produce application specific hardware. The inputs and outputs to the logic can be routed to any I/O pin, allowing for a great level of flexibility when implementing the design. Figure 1 shows the main components of a Spartan II FPGA.

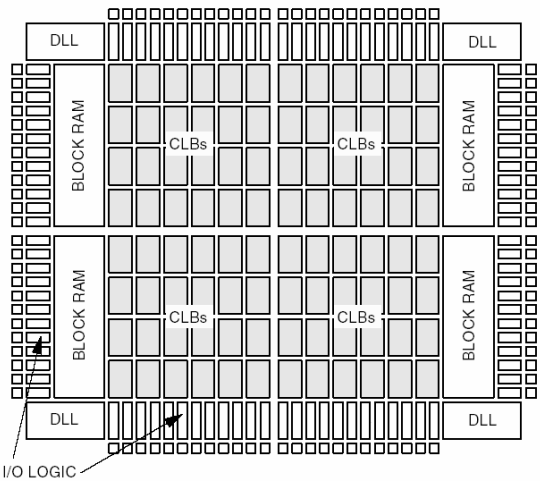


Figure 1 Main components in a Spartan II FPGA, taken from 4

As such, an FPGA offers a compromise between the flexibility of general-purpose processors and the

hardware-based speed of ASICs. Performance gains are obtained by bypassing the fetch-decode-execute overhead of general-purpose processors and by exploiting the inherent parallelism of digital hardware, while at the same time maintaining the ability to change the functionality of the system with ease.

Design entry can be achieved using low-level development methods such as schematic capture or hardware description languages. However, another alternative is to use system-level design languages such as Handel-C that take a more high-level approach [5].

### 1.1 HANDEL-C

Handel-C is a language developed by Celoxica that compiles algorithms written in a high-level C-like language directly into gate-level netlists. It is based on a subset of ANSI-C with syntax extensions for hardware design such as variable data widths, parallel processing and channel communication between parallel processing blocks. The language is designed to allow software engineers to express an algorithm without any knowledge of the underlying hardware.

As many algorithms are prototyped in a higher-level programming language like C they must be ported into VHDL or Verilog for implementation, which increases the risk of errors. Handel-C avoids this problem by using a high-level language to design the algorithms and then directly compile them to hardware [6].

Assignment statements in Handel-C take exactly one clock cycle. All other language statements such as control logic constructs are free and add zero additional clock cycles although they can increase combinatorial delays to the extent that the system clock cycle may need to be lengthened [5].

Section two describes the algorithm used for correcting barrel distortion. Section three describes a novel approach to a hardware-based implementation of a real-time barrel distortion correction.

The hardware used to support this implementation is the RC100 prototyping and development board from Celoxica, which incorporates a Xilinx Spartan-II FPGA, video decoder, offchip RAM and video DAC.

## 2 BARREL DISTORTION CORRECTION

### 2.1 ALGORITHM

The barrel distortion model [2] is:

$$r_u = r_d(1 + kr_d^2) \tag{1}$$

where $r_u$ and $r_d$ are the distance from the centre of distortion in the undistorted and distorted images respectively, as shown in figure 1, and $k$ is the distortion parameter.

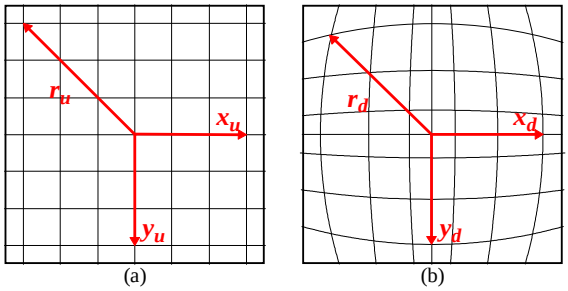


Figure 1 Barrel distortion Coordinate system

A block diagram of the complete system is shown in Figure 2. The system is driven entirely from the scan position of the display. Distortion correction is performed by using the current scan position and a magnification factor held in a look-up table (LUT) to calculate the address of the corresponding distorted pixel that is located in video RAM and the pixel is output to the display

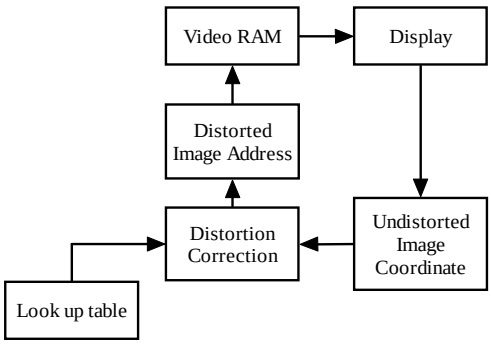


Figure 2 System diagram

This barrel distortion model effectively gives the coordinates in the undistorted image as a function of those of the distorted image. This form is unsuitable for real-time correction because it is necessary to produce the undistorted output pixels in raster order. The coordinates in the undistorted image must be used to determine which pixel in the distorted image should be displayed. Therefore the equation needs to be of the form:

$$r_d = F(k, r_u) \tag{2}$$

As $r_u$ is calculated as:

$$r_u = \sqrt{x_u^2 + y_u^2} \tag{3}$$

which involves two multiplications and one square root, this function is very costly in terms of resources. Therefore it is preferable to have the model

in terms of $r_u^2$. Equation (1) may be rewritten in terms of a radial dependent magnification as:

$$x_d = x_u M(k, r_u^2)$$
$$y_d = y_u M(k, r_u^2) \quad (4)$$

where the magnification factor, M is:

$$M = \frac{r_d}{r_u} = \frac{1}{1 + k r_d^2} \quad (5)$$

Unfortunately this equation is still in terms of $r_d^2$ rather than $r_u^2$, but this can be solved by substituting equation (4) into equation (5) and obtaining:

$$M = \frac{1}{1 + k M^2 r_u^2} \quad (6)$$

Equation (6) may be used to iteratively solve for $M$ in the following manner. First, $M$ is set to 1, and substituted into equation (6). This gives a revised value of $M$, which is again substituted into the equation. This is iterated until $M$ converges to the desired precision. Figure 1 shows the resultant magnification function. Clearly, the mapping depends on the product $kr_u^2$, so this avoids having to have a separate mapping for each $k$. Having a single mapping allows $M(kr_u^2)$ to be precalculated and put into a lookup table.

By normalising $x$ and $y$ coordinates based on the size of the image, $r_u^2$ can be made to be in the range of zero to one. After normalisation $k$ is also in the range of zero to one, where a value of one corresponds to severe distortion. This normalisation results in $kr_u^2$ also being in the range zero to one.

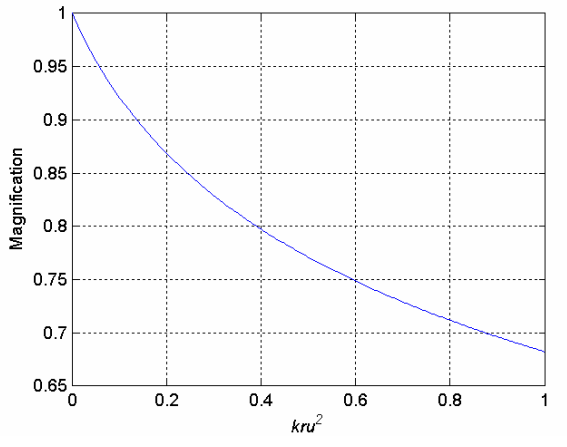


Figure 3: Magnification factor from pixel radius

## 3 IMPLEMENTATION

### 3.1 HARDWARE REDUCTION

There are two advantages gained by reducing the hardware required the first is to reduce the amount of CLBs which are required to implement the function. The second is that complex functions take a longer time to compute, this is due to the propagation delays caused by the construction logic function. These expressions with a large logic depth will slow the whole operation of the FPGA down, as Handel-C requires all operations to take one clock cycle. For a real time system the system must be able to run a fast enough to drive the output.

The calculation of $x_u^2$ and $y_u^2$ can avoid using a multiplier by making use of the fact that

$$(x_u + 1)^2 = x_u^2 + 2x_u + 1 \quad (7)$$

and the image is scanned in a raster fashion.

In hardware this is equivalent to $x_u$ with a 1 appended to the bottom bit added to the previous $x_u^2$.

$$(x_u + 1)^2 = x_u^2 + x_u @ 1 \quad (8)$$

In Handel-C functions can be shared by multiplexing the input and output. As $y^2$ and $x^2$ are never calculated at the same time the hardware for equation 8 can be shared. It was found that there was no difference in gate count when sharing the function.

Due to the complications and large number of logic gates needed to perform floating-point operations in hardware, a fixed point representation was chosen. Originally the Handel-C fixed point library was used for performing arithmetic operations, however using this library resulted in excessive compile times. Therefore all variables were represented as signed or unsigned integers and were commented to indicate the position of the binary point. When arithmetic operations were performed operands were shifted to ensure alignment.

The main advantage of using fixed point notation is that each step can have a different bit length depending on the precision required. As the image size is 503 by 480 pixels only 9 bits are required to represent $x$ and $y$. This means that 18 bits are required for $y^2$ and $x^2$. To give adequate resolution for the required correction it was found 10 bits are required. Only 16 bits are used for $kr_u^2$ as the top 8 bits are used for looking up the desired magnification in a 256 element look up table, dicussed in section 3.2. The lower 8 bits are used to estimate the distance the actual magnification is from the looked up one. The final output pixel address is 18 bits consisting of 9 bits for both $x$ and $y$. A tenth bit of $x$ is used to choose which of the two pixels stored in the memory location is displayed.

### 3.2 CALCULATING THE MAGNIFICATION

Due to the raster nature of the output $y_u^2$, is constant for a line but $x_u^2$, and therefore $M$, changes for each

pixel in the output. Obviously it is impractical to evaluate M directly for each pixel and therefore a lookup table implementation was considered using BlockRAM resources on the FPGA. Each BlockRAM is sufficient to provide 256 lookup entries. Analysing the mapping in Figure 1 indicates that 256 entries covering the range of $kr_u^2$ from zero to one will only provide 8 bits of accuracy. To get 10 bits requires four BlockRAMs; while 12 bits will use 16 BlockRAMs, more than what is available on the targeted device.

To overcome this problem, a single 256 entry lookup table is used, with interpolation between the table values used to improve accuracy. The top 8 bits of $kr_u^2$ are used to get the magnification value from the look up table. The next entry is also retrieved (the BlockRAM is dual port). The difference between these gives the slope, which is scaled by the lower 8 bits of $kr_u^2$:

$$M(kr_u^2) \approx M([kr_u^2]_{MSB}) + (M([kr_u^2]_{MSB}+1) - M([kr_u^2]_{MSB})) \times [kr_u^2]_{LSB} \times 2^{-8} \quad (9)$$

Since the most significant bit of $M$ is always 1, the look up table does not need to store this, allowing the table to contain $M$ to 17 binary places. When combined with interpolation the accuracy of the approximation is about 15.5 bits for 16 bits precision of $kr_u^2$.

### 3.3 COORDINATE TRUNCATION

Once the magnification has been found it is multiplied by the $x_u$ and $y_u$ position, equation (4). When doing this calculation the coordinate for the pixel location to be displayed is seldom an integer number. The approach that has been taken for this implementation is to truncate the result, discarding the fractional components. The results of truncation or the alternative rounding can introduce substantial error in pixel location. This caused distortion in lines in the image, creating jagged edges. Figure 4 shows the effect of the used of truncation, the circles represent actual pixels and the cross the calculated position with fractional component, in (a) there is only a small error, however in (b) the pixel is closer to the lower right pixel rather than the resulting left pixel.

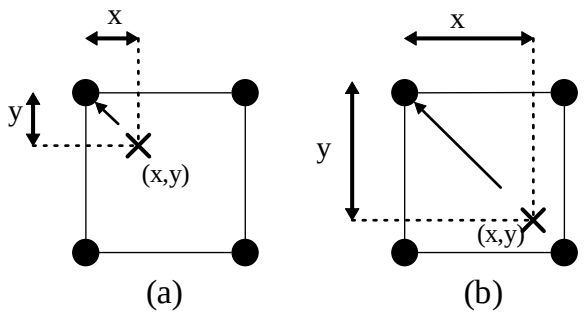


Figure 4 Effect of truncation

### 3.4 PIPELINING

Due to the constraint of real time processing all operations have to occur in a single clock cycle. This can be achieved by creating a pipeline where each operation uses the results of the previous clock cycle but resulting in a delay in output equal to the length of the pipeline. This delay can be accommodated by starting the pipeline calculations during the horizontal blanking period. A five-stage pipeline shown in Figure 5 is used for determining the coordinates in the distorted image of the pixel to be output. The dotted lines in Figure 4 represent registers.

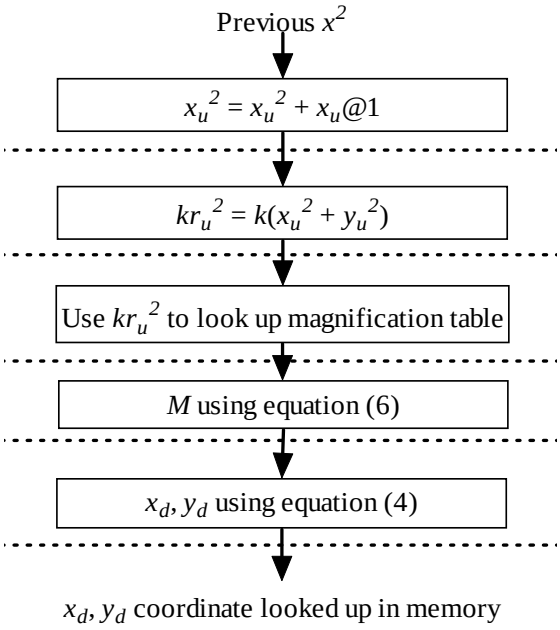


Figure 5: Pipeline for coordinate calculation

The $x_d$ used in the fifth stage is not the present $x_d$ but one due to the delay in the pipeline it needs to be the $x_d$ from four clock cycles ago, this can be achieved by registering the $x_d$ at each stage, but as we agre moving is a raster order the desired $x_d$ will be the present $x_d$ with four subtracted from it

## 4 RESULTS

The barrel distortion correction algorithm has been successfully implemented and tested on the RC100 development board, figure 6.a the distorted image and figure 6.b corrected image. This correction algorithm performs the correction in real time at a rate of 60 frames per second. The utilisation of the device is shown in Table 1.

Table 1: Resource utilisation of device (XC2S200-5FG456)

| | Resource Utilisation | |
|---|---|---|
| | **CLBS (1172 total)** | **BlockRAM (14 total)** |
| **Keyboard interface** | 129 (11%) | 1 |
| **Video decoder / VGA** | 235 (20%) | 4 |
| **Correction algorithm** | 270 (23%) | 1 |
| **Total** | 642 (54%) | 6 (42%) |

The correction algorithm uses 23% of the device with most of this being used to implement the large multipliers. If this were implemented on an FPGA such as the Virtex-II that incorporates built-in multipliers the resource utilisation would be significantly reduced.

The other resources used are not directly part of the correction algorithm but support it. The keyboard interface is only used to allow the correction factor to be changed, if a fixed lens configuration was to be used the correction factor could be fixed and the keyboard interface removed. The video decoder and the VGA sections are required for the capture and display of the image.

Although it cannot be clearly seen in figure 6.b there are some jagged edges in the image that are caused by the coordinate truncation process.

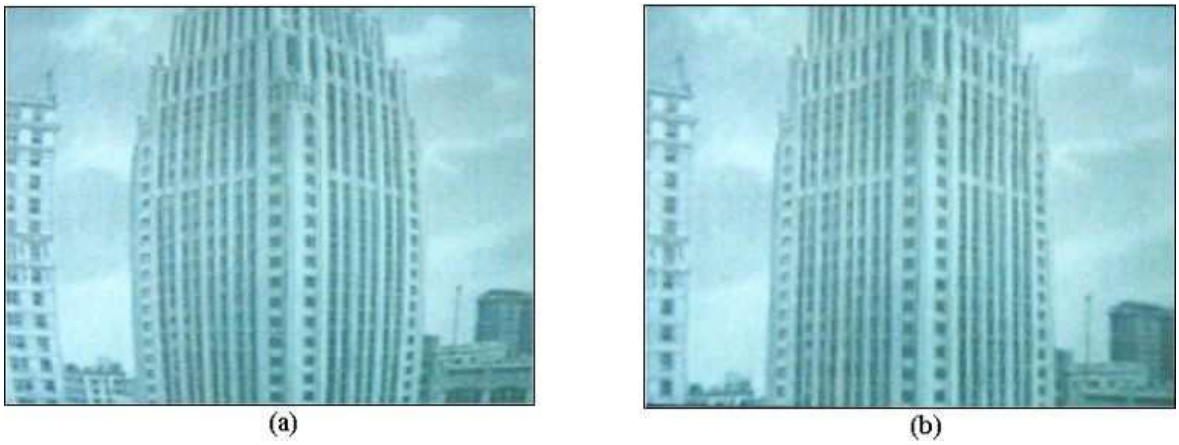


Figure 6: Results from correction algorithm

## 5 FUTURE WORK

It can be seen form the resource utilisation table that almost half of the FPGA is not utilised. This space can be used to implement bilinear interpolation between pixels to remove the effects of coordinate truncation, and improve the quality of the corrected image. This requires the used of row buffering to allow a number of pixels to be accessed in one clock cycle and is discussed in [7].

An investigation in to whether a CMOS optical sensor could be used as the memory element for the distorted image need to be done. If this can be successfully implemented an FPGA could be placed between the sensor element and a memory bank with the barrel distortion being corrected for before the image is stored. This would then enable other image processing functions to be done on the corrected image by either another FPGA or a microprocessor

The types of functions that can be implemented on FPGA hardware needs to be investigated.

## 6 CONCLUSION

A barrel distortion algorithm has been successfully implemented on a FPGA, using Handel-C.

To allow the algorithm to be preformed buffering of the image is required, for simplicity a whole frame is buffered in to RAM

The conversion from a software algorithm to one that runs in hardware at real-time presents a number of difficulties.

This includes the inability to do offline processing; this is inherent in all real time applications.

As only one off chip memory access is possible per clock cycle any complex functions require the buffering of pixel values in the FPGA.

So that hardware can be mapped to the desired FPGA there needs to be a minimisation of the logic gate count.

The use of a look up table with interpolation can reduce the complexity of the hardware design without significant loss of precision compared to calculating values at run-time.

Even though Handel-C was designed to raise the level of abstraction for hardware design and shift the focus to algorithmic design, in our experience it has been beneficial to maintain a data flow approach at the register transfer level. The data flow approach has many advantages when it comes to bit manipulation, logic reduction and pipeline design.

## 7 ACKNOWLEDGEMENTS

The Celoxica University Programme for generously providing the DK1 Design Suite.

## 8 REFERENCES

1 Bainbridge-Smith, A. & Dunne, P., "FPGAs in computer vision applications", *Proceedings Image and Vision Computing New Zealand 2002*, pp 347-352 (2002).

2 Bailey, D.G., "A new approach to lens distortion correction", *Proceedings Image and Vision Computing New Zealand 2002*, pp 59-64 (2002).

3 Li, M., Lavest, S.M., "Some aspects of zoom lens camera calibration", *IEEE Trans on PAMI*, 18(11): 1105-1110 (1996).

4 Xilinx®, *Spartan-II 2.5V FPGA Family: Complete Data Sheet,* (September 3, 2003)

5 Alston, I., Madahar, B., "From C to netlists: hardware engineering for software engineers?", IEE Electronics & Communication Engineering Journal, pp 165-173 (August 2002).

6 Celoxica Ltd., *HANDEL-C Language Overview*, (2002).

7 K.T. Gribbon, C.T. Johnston, and D.G. Bailey, *"A Real-time FPGA Implementation of a Barrel Distortion Correction Algorithm with Bilinear Interpolation"* accepted for IVCNZ 2003 conference.