Real-Time Fisheye Lens Distortion Correction Using Automatically Generated Streaming Accelerators

Nikolaos Bellas[1] Sek M. Chai[2] Malcolm Dwyer[2] Dan Linzmeier[2]

Computer Engineering and Communications Department[1]
University of Thessaly, Volos,
Greece
nbellas@uth.gr

Motorola, Inc.[2]
Schaumburg, IL
USA
sek.chai@motorola.com

Abstract— Fisheye lenses are often used in scientific or virtual reality applications to enlarge the field of view of a conventional camera. Fisheye lens distortion correction is an image processing application which transforms the distorted fisheye images back to the natural-looking perspective space. This application is characterized by non-linear streaming memory access patterns that make main memory bandwidth a key performance limiter.

We have developed a fisheye lens distortion correction system on a custom board that includes a Xilinx Virtex-4 FPGA. We express the application in a high level streaming language, and we utilize Proteus, an architectural synthesis tool (described in [4][5]), to quickly explore the design space and generate the streaming accelerator best suited for our cost and performance constraints. This paper shows that appropriate ESL tools enable rapid prototyping and design of real-life, performance critical and cost sensitive systems with complex memory access patterns and hardware-software interaction mechanisms.

I. INTRODUCTION

Recent advances in reconfigurable technology have expanded the application domains for which a reconfigurable system offers the best performance versus cost ratio. The new paradigm of spatial computation enables the design and deployment of FPGA-based SoCs with complex hardware and software components [26]. In particular, modern FPGAs are used in "smart camera" systems to accelerate real-time, H.264/AVC video encoding [30], high image quality medical visualization [17], or image analysis and pattern recognition [5].

Fisheye lenses allow imaging a large sector of the surrounding space instantaneously. While ordinary rectilinear lenses map incoming light rays to a planar photosensitive surface, fisheye lenses map them to a spherical surface, which is capable for a much wider field of view (FoV). It is possible, and in fact very common, for fisheye lens to encompass a FoV of 180^{o}. Such hemispherical images have been traditionally used for specialized applications such as surveillance [23], robot navigation [13], content creation for immersive environments and virtual reality [27], photography [29], astronomy, etc.

Fisheye (or, generally, wide-angle) imaging can be used in more mainstream applications such as consumer digital imaging and video capture. By capturing a larger section of the surrounding space, a fisheye lens camera affords a wider horizontal and vertical viewing angle, provided that the distorted images at the fisheye space can be corrected and transformed into the perspective space in real time (Figure 1). Real-time distortion correction for megapixel frame resolution is possible using specialized hardware or powerful desktop graphics processors, but beyond the reach of today's embedded processors or DSPs, as we will show in section IV.D.

In this paper, we describe the design and implementation of a real-time fisheye lens distortion correction module using the Proteus architectural synthesis tool [4][5]. The module is part of an FPGA-based camera system which tracks the region of interest (ROI) within the frame using input from the user and then corrects the ROI in real-time before storing or transmitting the data to the other end.

We first lay out the geometric properties of the fisheye lens distortion correction, and we explain the algorithmic aspects of the computational flow in section II. The focus is on the challenges of mapping this application into efficient hardware, and, in particular, on the non-linear memory access patterns and the need for optimizations to increase the effective bandwidth between the accelerator and the main memory (section III.A). We make use of the on-chip memory bandwidth available in FPGAs by organizing the computations around tiled datasets, and then expressing the application in a streaming data flow language (section III.C).

Using the Proteus toolset and programming methodology, we produce hardware accelerators that follow the streaming architectural paradigm [1][4][5][8]. Automation allows us to quickly explore different Pareto-optimal implementations that span the whole area versus throughput design space as described in section IV.A. An implementation that meets the system requirements as well as results on an implementation in a Virtex-4 FPGA is described in section IV.B.

Not all tasks in a complete system should be accelerated to hardware. Tasks that are control-intensive or occur infrequently are left in software executed in the scalar processor of the FPGA. The hardware-software interaction is an important component of this design, and more so because optimizations such as tiling impose extra burdens on the software-hardware communication overhead (section IV.C).

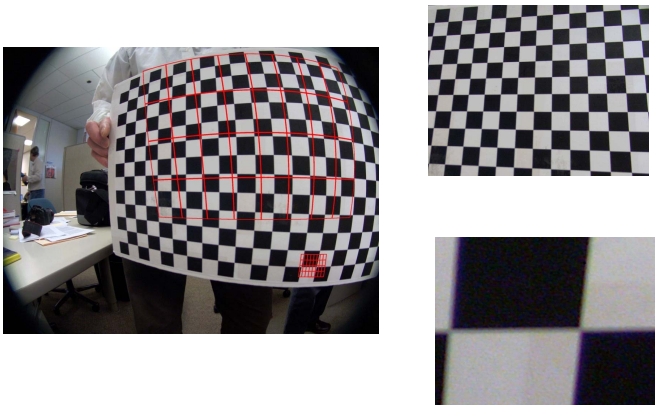

Figure 1. Fisheye lens distortion correction example for two windows of horizontal FoV=60° and FoV=8° The lenses cover a horizontal FoV=120°

By formulating the problem in the streaming domain we produce a design which more than 4x times faster than a high performance multi-core processor in how it utilizes its hardware resources measured in speedup per unit of clock frequency (section IV.D).

The main contribution of this paper lies on introducing a complex image processing application, explaining the source level optimizations on the original code to exploit the distributed memory architecture of modern FPGAs, and, finally, applying a design automation method to implement multiple versions of the module before the system architect selects the most appropriate implementation. One of the primary goals of the project was to show that a high level abstraction in the problem description can drastically reduce the development time of a highly complex, industrial-level system, without necessarily resulting to sub-optimal performance or area.

II. FISHEYE LENS DISTORTION CORRECTION ALGORITHM

The stereoscopic geometry of wide-angle photography does not comply with the conventional central perspective projection of Figure 2a. The central perspective model is based on the premise that the incidence angle of an incoming ray from an object point is equal to the angle between the ray and the optical axis[2]. According to this model, object points with incidence angle close to 90° would be projected to a point at infinite distance from the principle point[3], thus limiting the field of view to angles close to the optical axis. To enable a larger field of view, a different projection model is necessary.

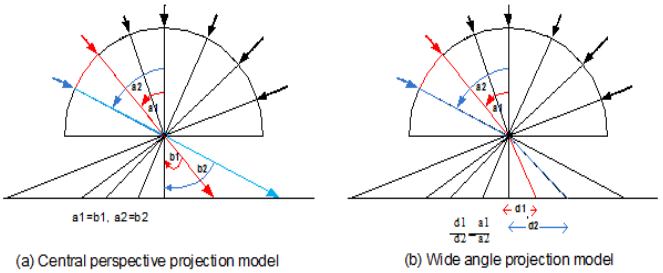

Figure 2. Projection model of fisheye lens

The fisheye projection is based on the principle that the incidence angle of the ray is linearly proportional to the distance of the projection point to the principle point. According to this equation, the incoming object rays are refracted towards the optical axis, so that even an incidence angle of 90° can be projected onto a finite distance from the principle point: $\frac{r}{R} = \frac{a}{90°}$, where $r = \sqrt{x^2 + y^2}$ is the distance between the projection point and the optical axis, α is the angle of incidence, R is the image radius, and x, y are the fisheye image coordinates. To reconstruct the projection of an object point into the 3D camera space, the object and image coordinates have to refer to the same coordinate system. After some algebraic transformations [22], the resultant equations that describe the projection on the image plane when using fisheye lens are given by:

$$x = \frac{\frac{2R}{\pi} a\tan\left[\frac{\sqrt{(Xc)^2 + (Yc)^2}}{Zc}\right]}{\sqrt{\left(\frac{Yc}{Xc}\right)^2 + 1}} + d_x + x_h$$

$$y = \frac{\frac{2R}{\pi} a\tan\left[\frac{\sqrt{(Xc)^2 + (Yc)^2}}{Zc}\right]}{\sqrt{\left(\frac{Xc}{Yc}\right)^2 + 1}} + d_y + y_h \quad (1)$$

where X_c, Y_c, Z_c are object point coordinates on the 3D camera coordinate system, d_x, d_y are lens-distortion parameters, and x_h, y_h are the coordinates of the principle point.

Equation (1) provides the method to convert the perspective space coordinates (X_c, Y_c, Z_c) of an object back to the 2D distorted fisheye space (inverse mapping). First, the 2D central perspective image coordinates *i, j* are converted to the 3D space:

$$\begin{bmatrix} Xc \\ Yc \\ Zc \end{bmatrix} = \begin{bmatrix} r11 & r12 & r13 \\ r21 & r22 & r23 \\ r31 & r32 & r33 \end{bmatrix} \times \begin{bmatrix} i \\ j \\ 1 \end{bmatrix}$$

Equation (1) can be broken into elementary mathematical functions and the lens-distortion parameters can be folded into the following equations.

$$d = \sqrt{Xc^2 + Yc^2} \quad (1)$$

$$Du = \frac{d}{Zc} \quad (2)$$

$$Ru = a\tan(Du) \quad (3)$$

$$P = k(1) * Ru^4 + k(2) * Ru^3 + k(3) * Ru^2 + k(4) * Ru + k(5) \quad (4)$$

$$x = \frac{P}{d} * Xc + x_h, \qquad y = \frac{P}{d} * Yc + y_h \quad (5)$$

Equation (4) models lens distortion based on the sensor parameters *k[]*. Given the (Xc, Yc, Zc) coordinates of a each point, we compute the corresponding coordinates of this point

[2] The imaginary vertical line that passes through the center of curvature in Fig. 2.

[3] The point which the optical axis intersects the projection surface

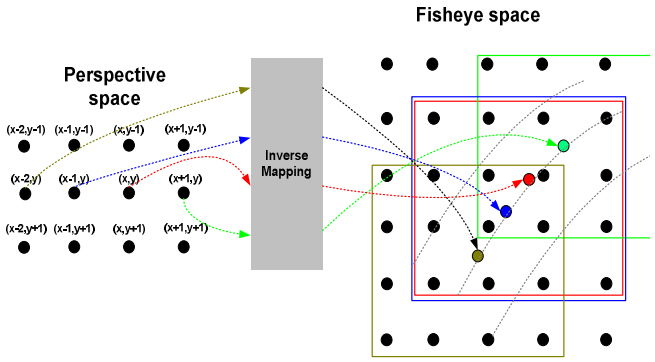

Figure 3. Inverse Mapping is used to convert the coordinates from the perspective space back to the fisheye space. A 4x4 pixel neighborhood around the mapped pixels is used to perform bicubic interpolation and compute the pixel values at the fractional points.

at the 2D fisheye space. Note that (5) produce a fractional pair of coordinates at the fisheye plane, and the pixel value at that point has to be interpolated based on the values of the pixels at neighboring integer positions.

Bicubic interpolation is a robust, yet computationally expensive technique used to approximate intermediate points of a continuous event given the interpolation nodes, or sample points [16]. Although other techniques such as nearest neighbor or bilinear are simpler and more widely used in hardware implementations, the high PSNR[4] requirements of the fisheye correction module makes this the method of choice.

The inverse mapping and two-dimensional bicubic interpolation flows are shown in Figure 3**.** The bicubic interpolation method uses cubic sampled functions to approximate an intermediate value based on the fundamental property that the sample function *f* is equal to the interpolation function *g* in the sample points.

The following equation approximates the value of a function *f* at point *x*, based on known sampled point values $C_i = f(x_i)$:

$$g(x) = C_1 * U_1(s) + C_2 * U_2(s) + C_3 * U_3(s) + C_4 * U_4(s)$$
$$U_1(s) = (-s^3 + 2s^2 - s)/2$$
$$U_2(s) = (3s^3 - 5s^2 + 2)/2$$
$$U_3(s) = (-3s^3 + 4s^2 + s)/2$$
$$U_4(s) = (s^3 - s^2)/2$$

where the point *x* is such that: $x_1 \le x_2 \le x \le x_3 \le x_4$ and $s = x - x_2$.

The two-dimensional bicubic interpolation method shown in Figure 3 is accomplished by one-dimensional interpolation in each coordinate. The method requires the use of the 16 pixel values of a 4x4 window around the interpolated point.

The interpolation function *G* at point *(x,y)* is given by:

$$G(x,y) = g_1(x) * V_1(t) + g_2(x) * V_2(t) + g_3(x) * V_3(t) + g_4(x) * V_4(t)$$
$$V_1(t) = (-t^3 + 2t^2 - t)/2$$
$$V_2(t) = (3t^3 - 5t^2 + 2)/2$$
$$V_3(t) = (-3t^3 + 4t^2 + t)/2$$
$$V_4(t) = (t^3 - t^2)/2$$

where $t = x - \lfloor x \rfloor$, and $g_k(x), k = 1,2,3,4$ is the row-wise interpolation for the rows at $\lfloor y \rfloor - 1, \lfloor y \rfloor, \lfloor y \rfloor + 1, \lfloor y \rfloor + 2$

In order to eliminate high frequency artifact noise on the image, we apply a 5-tap vertical and a 5-tap horizontal low pass filter on the corrected image as the final step of the algorithm to downscale to a VGA (640x480) output resolution.

We have developed an optimized software version of the algorithm in C running on a Core 2 Quad desktop processor with 2GB of RAM for baseline comparison and statistics gathering. The bicubic interpolation was responsible for 75% of the execution time, the inverse mapping for 3.6%, and the low pass filter for 21.4%. The low contribution of the inverse mapping is due to the efficient FP units of the x86 architecture; the FPGA implementation has to expend a high amount of gates and cycles to implement this module for real-time operation. As we will examine later in section IV.D, real time correction with acceptable video quality is beyond the reach of today's high performance multi-core processors.

III. Architectural Approach

A. Need for architectural optimizations

To achieve real-time, high-quality fisheye lens distortion correction, a very large amount of pixel data has to be streamed from an image sensor to the accelerator, be processed, and transferred back to the DRAM memory before transmitted to the end user. The main observation from the algorithmic analysis of section II is that the pixel coordinates at the fisheye space that have to be retrieved at each step follow a complicated non-linear pattern that, although static (not dependent on pixel values), is difficult to be pre-computed.

Figure 3 shows the mapping of pixel coordinates from the perspective to the fisheye space. The resulting non-linear trace of fractional coordinates determines a square 4x4 neighborhood of pixels, which are needed to approximate the pixel value at each fractional sub-pixel point. The exact trace shape depends on a variety of factors, such as the field of view of the ROI, the exact location of the pixel in the fisheye space, and also on a set of parameters modeling lens distortion. Although the trace is not data dependent, and thus, can be theoretically pre-computed ahead of the interpolation, a complex pre-load unit would be needed to prefetch the pixels for the bicubic interpolation and low pass filtering.

A general purpose caching mechanism in lieu of an exact streaming interface is not efficient in a lot of cases. Using the software version of the algorithm we can assess the efficiency of a caching mechanism. Less than 1% of successive fractional points fall within the same 4x4 neighborhood (e.g. like the two pixels in the middle of the trace in Figure 3), for a large field of view (FoV) equal to 60°, when the pixels are near the optical center of the image. This percentage grows up to 30% when the pixels are close to the edge of the frame for the 60° FoV, and to 80% for a much smaller FoV of 10°. Therefore, besides lack of

[4] Peak Signal to Noise Ratio is frequently used to measure signal quality in images

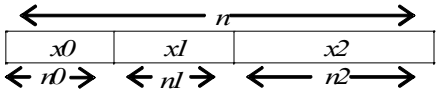

Figure 4. The partition of the input x in three fields in the BT method

temporal locality, this algorithm implementation suffers from lack of spatial locality when the field of view is large (and a large FoV is the reason that fisheye lens are used, in the first place).

B. Arithmetic Operations and Numerical Accuracy

Inverse mapping uses the inverse tangent operation, divisions, and square root, which are expensive in hardware resources. Popular methods to approximate such elementary mathematical functions in hardware are the CORDIC algorithm [2] and the Bipartite Tables (BT) [12]. The BT method decomposes a function into a sum of two functions with a smaller input size, and requires two smaller look-up tables (LUTs) rather than a larger one. It has been reported in [20] that the BT method offers a compression factor of one to two orders of magnitude compared to a straightforward LUT implementation. We settled on the Bipartite Tables because we can make use of the on-chip memories of the FPGAs, without expending a lot of logic slices.

The fractional part of input x is partitioned in three non-overlapping fields (Figure 4) and the function is approximated as:

$$f(x) = f(x0 + x1 + x2) \approx a0(x0, x1) + a1(x0, x2)$$

The two tables provide the coefficients *a0* and *a1*, given the smaller inputs *(x0,x1)* and *(x0,x2)*, respectively[12].

The only constraint is that the input x to function *f(x)* be normalized to within the [1,2] range before the tables are accessed. This is done by detecting the number of leading zeros of the input x, and then, shifting left or right the input x by that amount. The output *f(x')= f1(x')+f2(x')* is corrected back to produce the elementary function *f(x)*. All calculations are done in the streaming accelerator using fixed-point arithmetic hardware.

The numerical accuracy of the inverse mapping function is important for high-quality image processing. The bicubic interpolation algorithm creates severe spatial distortions if the granularity of the sub-pixel fractional coordinates is not fine enough. In our implementation, we use 25 bits for the fixed-point representation of each one of the two coordinates, 12 of which are used for the fractional part.

We use two tables for each one of the *atan(x)*, $\sqrt{x}$, *1/x* mathematical functions as follows: a 256x16 and a 512x7 table for *1/x*, a 256x17 and a 256x7 table for $\sqrt{x}$, and a 256x17 and a 256x7 table for *atan(x)*. We use 15-bits fractional accuracy for the output of each of the tables.

C. Tiling and Pipelining

A key observation is that the algorithm has a large degree of data reuse but not necessarily across a row or column of the frame. For instance, in case of the row-wise inverse mapping (Figure 3), some fisheye space pixels that have already been

Figure 5. Tiling allows each block of pixels to be corrected before the correction of the next block.

fetched to interpolate the perspective space pixels at row y can be used again for the next row *y+1*. Reuse is maximized by applying two-dimensional tiling in each frame, a technique used by optimizing compilers to improve cache hit rate. We partition the output frame in blocks of equal size, and we seek to produce all the pixels of one block before we start producing the pixels of the next block.

Figure 5 shows a block of output pixel data taken from the frame in Figure 1, and the corresponding rectangular bounding block of pixels at the distorted fisheye space needed to interpolate the output pixels. Adjacent rectangular boxes in the fisheye space will partially overlap due to the curvature of the distorted boxes (shown in red in Figure 5). The partial overlap implies that pixels close to the edges of the bounding box are being fetched from the memory more than once.

By re-arranging the order of the computations to exploit reuse at the block level, we also allow distributing the data in low-latency on-chip memories, which are readily available in modern FPGAs. The advantages of tiling are as follows:

- All pixel data are kept within a very small and constant latency from the computational units, instead of in off-chip DRAMs.
- We exploit the high bandwidth capabilities of on-chip SRAMs, which is necessary to achieve high frame rate, and high image quality, and
- We can trade-off control and communication overhead between the accelerator and the scalar processor with FPGA memory capacity. Larger, more costly FPGAs with more on-chip memory can hold larger, and fewer tiles, and increase the accelerator dataset.

Additional optimizations can be applied if we pipeline the fisheye transformations, and allow multiple tiles to reside in the computational pipeline at any moment. Pipelining increases the effective computational bandwidth, and better utilizes the high bandwidth of on-chip SRAMs. In our scheme, each pipeline stage is dedicated to a single transformation so that successive tiles are processed simultaneously. For example, while all pixels of tile N are being processed in the bicubic interpolation stage, the tile N+1 is in the inverse mapping stage. We call this optimization macro-pipelining because it typically takes thousands of cycles for each stage to finish its task, and the size of each tile is thousands of pixels.

The accelerator architecture in Figure 6 shows the pipelined block diagram of the streaming accelerator for real-time fisheye lens distortion correction. The pipeline is partitioned in four stages:

1. A DMA (prefetch) stage accesses the main memory and stores the next tile of pixel data[5] to on-chip SRAMs (tile N). Multiple SRAMs are used to increase the bandwidth from the main memory, and to the next pipeline stage. Figure 3 indicates that one of the performance bottlenecks of the system is due to the need to read 16 pixels[6] from the fisheye space to interpolate the value of a single pixel at a sub-pixel location. The availability of dual-ported SRAMs in modern FPGAs also helps to increase the bandwidth.
   At the same pipeline stage, inverse mapping transforms coordinates of tile N from the perspective to the fisheye space using the bipartite table method as explained previously. The fractional sub-pixel coordinates are stored in on-chip SRAMs to be used in the next stage.
2. At the same time, the bicubic interpolation unit retrieves each coordinate pair (x,y) of tile N-1, and uses these coordinates as address pointers to read the 4x4 pixel area from the input buffers. For each pair (x,y), we first compute the coordinates $(\lfloor x \rfloor, \lfloor y \rfloor)$ of the closest integer pixel at the top left direction of the sub-pixel, and we then use these coordinates as the basis to retrieve the remaining pixels in the 4x4 vicinity. For instance, all the integer pixels within the four colored bounding boxes of Figure 3 are retrieved, 32 bytes at a time, in order to compute the pixel values at locations *(x-2, y), (x-1, y), (x, y),* and *(x+1, y)* of the perspective space. These values are, in turn, stored in the next pipeline buffers.
3. The next two stages are used to perform a 2D convolution-based low pass filter both vertically and horizontally on tiles N-2 and N-3, respectively. The filtering is necessary to reduce the high-frequency noise artifacts particularly needed when the requested field of view is small. The final stage also streams the data out to the main memory.

To be able to operate all stages simultaneously, we make use of dual buffering schemes in each stage. We use the dual-port capability of the FPGA on-chip memories exclusively within the same pipeline stage to increase bandwidth, and we use two such memories to interleave reads and writes at successive stages.

Referring to Figure 6, at any point during computation, the inverse mapping stage writes coordinates to SRAM A of dual buffer named *Coordinates*, and the bicubic interpolation stage reads the coordinates of the previous tile from SRAM B. In the next accelerator invocation for the next tile, the roles of SRAMs A and B are reversed. The eight buffers between the DMA and the bicubic interpolation stage are all dual-ported and double buffered and provide a collective bandwidth of 16 bytes/cycle for read and 16 bytes/cycle for write. Our Proteus tool is parameterized to instantiate any combination of single or dual-ported SRAM, in a single or dual-buffered configuration.

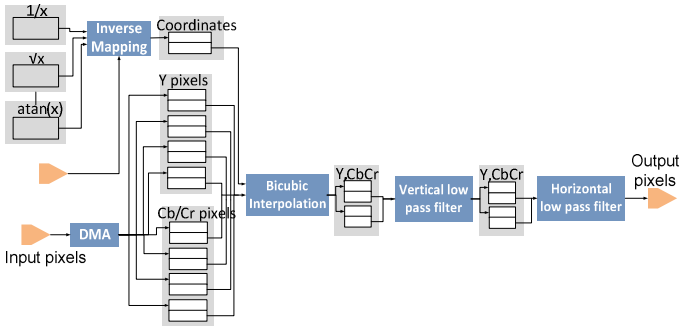

Figure 6. Block diagram of the lens distortion correction pipeline.

Finally, special attention is needed for the first and the last few invocations of the accelerator in every frame when the macro-pipeline is filled and when it is drained. We make use of the valid bit mechanism [4][8] to force parts of the macro-pipeline to inactivity. For example, the reads out of the pixel buffers between the DMA and the BI are set to invalid in the first invocation of the accelerator for each frame, because they have not been filled yet with pixel data. Likewise, the two input streams are de-activated (by setting their stream descriptor size to 0), at the last three invocations of the accelerator in order to drain the macro-pipeline. The scalar processor is responsible for managing the setting of valid bits for the buffers, and the I/O stream units.

IV. FPGA IMPLEMENTATION

The fisheye lens distortion correction module is part of an FPGA-based SoC which uses a fisheye lens camera. The SoC has been implemented in a Virtex-4 LX-80 FPGA and also contains a real-time tracking mechanism to detect a remote-control device used to set the pan, tilt, and zoom parameters in the frame. It includes a Microblaze microprocessor, an image sensor interface, a multi-ported memory controller, and a number of I/O peripherals. The real-time video compression/decompression takes place outside the FPGA part. The FPGA system operates at 62.5 MHz, and produces 22 VGA frames/sec given a megapixel input fisheye image.

A. Architectural automation

We use Proteus, our streaming accelerator generation tool to quickly perform architectural exploration and hardware generation [4][5]. Proteus generates accelerators that use the memory-to-memory stream-oriented approach to vector processing. The architecture decouples and overlaps data accesses and computations and removes the requirements from programmers to explicitly schedule memory accesses [8]. This approach has several independent load/store units (called stream interface units, SMIFs) used to prefetch data from wide, slow memories and turn it into narrow, high-speed streams of vector elements. A natural consequence is that vector alignment and size become irrelevant.

The Proteus tool allows us to quickly explore different architectural scenarios and evaluate the quality of the design in terms of computational bandwidth, clock frequency and size. By varying system constraints such as the number and size of available functional units (multipliers, ALUs, on-chip SRAMs, etc.), the average bandwidth and latency to the main memory and so on, we obtain a large set for different implementations in a few minutes.

The application code for the fisheye lens distortion correction is very compact, and has size of around 800 lines. The code is explicitly operating on tiled data, i.e. the tiling and pipelining source level transformations are applied manually to the functional C code, and not through an optimizing compiler. The Proteus tool has been extended to provide facilities for random, and data-dependent memory accesses for on-chip memories.

[5] Pixel data are in YCbCr 422 format
[6] Or 32 bytes: 16 Y bytes, and 16 CbCr bytes

These extensions do not break one of the characteristics of the streaming computing, namely the decoupling between data fetching and computations. This is because they are only limited to local, constant latency, on-chip memories, and not to accesses to external memories or peripherals. The logic and the related on-chip memory are part of the data path, and the related load and store instruction only support the access of a single item at a time.

Of particular interest is the mechanism to start and terminate the accelerator. The user triggers the accelerator by writing to a memory-mapped "Start" register, and the accelerator starts reading the input streams and writing the output streams according to the stream specifications. These specifications, such as the starting address, shape and size are also set before the accelerator is invoked by the user.

Each of the pipeline stages of Figure 6 can start and finish independently to each other. This not only useful, but also critical in this design because the back-end stages should not be enabled before the pipeline is filled, and the front-end stages should not be enabled when the pipeline is flushing.

Mechanisms that cause an execution thread running on the accelerator or on a stage of the accelerator to terminate include:

- reading all the stream elements of an input stream,
- writing all the stream elements of an output stream to the memory, and receiving an ACK signal from the system bus when the last bus write transaction terminates ,
- performing all the required writes to the on-chip SRAMs
- finishing all the accumulator iterations required in the DFG.

When a termination mechanism is activated, an interrupt is generated to an accelerator interrupt controller. The controller combines all the relative interrupts from the various sources to generate an accelerator interrupt to the processor interrupt controller. For instance, the accelerator of Figure 6 has sixteen interrupt sources: two from the input stream units, one from the output stream unit, and thirteen for the writes to the on-chip SRAMs.

The Proteus toolset generates automatically all the necessary interrupt circuitry based on the structure of the DFG. It also generates all the appropriate registers in the accelerator memory map needed to monitor and control the accelerator through the Microblaze processor.

The tool provided a very fast path to architectural exploration and final implementation. For the final implementation, we started with a streaming DFG description of the application of around 800 lines of code, and we generated approximately 100,000 lines of code of synthesizable Verilog. Moreover, Proteus generates the testbench to drive and monitor the Verilog code. The testbench includes facilities to initialize internal registers of the hardware accelerator, to drive the input streams, and to monitor and verify the correctness of the output streams. For more information on the Proteus tool the reader may refer to [4][5][8]. Readers are referred to [1][6][10] more information on the streaming programming model.

B. Hardware details

The size of available on-chip SRAM memory in the FPGA device determines the tile size in the fisheye lens distortion correction algorithm. The LX-80 part has 200, 18 Kbit dual-ported on-chip SRAMs, and each one can be configured in any "aspect ratio" from 16Kx1 to 512x36 [24]. If the output tile is Nx8 bits, then the buffer between IM and BI is Nx50[7], the buffers between BI and LPF vertical will be Nx8 and Nx16, and the buffers between LPF vertical and LPF horizontal will be (N/2)x8 and (N/2)x16. The output data are down-sampled horizontally and vertically by two and the last two buffers store the pixels after vertical down-sampling only.

Moreover, we use 8x2=16 buffers at the input to store 422 pixels and to increase the number of available read ports at the BI stage. Four of the eight dual-buffers store Y pixel data, and four store Cb/Cr data. The exact number of pixels stored in these buffers is dependent on FoV settings and the location of the block within the frame. We need to make sure that the buffers are large enough to store the pixels for the largest FoV allowed in the system, and for all block locations within the input image. As Table 1 indicates, the buffer size N for the application was 6864 pixels, which corresponds to a 128x48 tile size. Four additional rows and four additional columns which were added to accommodate the boundary conditions of the 3-tap low pass filter, increased the tile size to 132x52. The output stream is (128/2 x 48/2) = 1536 pixels (or 3072 bytes) per tile, and the VGA output is partitioned in (640*480)/1536 = 200 tiles (20 tile rows, 10 tile columns). The total processing time per frame amounts to 203 blocks, due to the need to fill and flush the accelerator pipeline at the beginning and end of each frame.

To be able to sustain an acceptable frame rate without temporal artifacts, the module should process at least 15 VGA frames/sec at 62.5 MHz. Based on the above constraints, the accelerator should spend no more than 328us (20526 cycles) for each block processing a 6864-pixel tile. In order to achieve this performance, the accelerator is scheduled according to modulo scheduling with an iteration interval of two cycles.

We measured our performance on the Virtex-4, LX-80 FPGA board at 224us (13995 cycles) per 6864-pixel tile, which accounts for 22 fps frame rate. This measurement includes both the time for the accelerator to process the tile and for the Microblaze to set up the accelerator for tile processing. Up to about 400 instructions are executed in parallel in every clock cycle at the steady state portion of the modulo schedule of the streaming accelerator.

A high performance, priority-based, multi-port, memory controller serves the two I/O ports of the accelerator and additional peripherals with real time requirements such as the image sensor interface. A high-speed dedicated PLB bus drives the I/O streaming data from and towards the multi-ported memory controller. In our design, a PLB write burst length of 16 is used to transfer one 128-byte row of the output tile[8]. The read accesses, which dominate the accelerator bandwidth, are also burst, but the burst size is variable depending on the size of the row of each input tile. Typically, 2-3 bursts of size 16 are needed to read an input. Each write burst transaction requires 20 cycles (4 cycles overhead), and each read burst transaction requires 33 cycles (17 cycles overhead). Using bursting to access data from the DRAM is critical to meeting the real time requirements of the application, and to reduce memory latency.

An arbitration module regulates the access of the I/O streaming units to the dedicated PLB bus that connects the

[7] Each entry stores 2, 25-bit fractional coordinates

[8] Note that the PLB bus size is 8 bytes and each output tile is 64x24 pixels

Table 1. Percentage resource utilization and BRAM sizes for the Virtex-4 LX-80 FPGA

Logic Slices	11082 (30%)
DSP48 units	71 (88%)
BRAMs	109 (54.5%)
BRAM types (Number per type)	4096 x 8 (16) 13728 x 50 (1) 6864 x 8 (2) 6864 x 16 (2) 3432 x 8 (2) 3432 x 16 (2) Bipartite Tables (see 1))

accelerator to one of the ports of the memory controller. The arbitration unit uses round-robin and is also generated automatically by the Proteus tool to match the characteristics of the streams.

The dataset of each accelerator invocation is a tile of pixels and the accelerator is called 203 times for every frame. Table 2 shows the sequence of events between the scalar processor and the Proteus accelerator. The scalar processor sets up the accelerator for the next tile in the frame by modifying the stream descriptors (e.g. starting address of the input and of the output stream, span, skip of the input stream, etc.), and triggers the start by setting a memory-mapped register in the accelerator. At the end of the tile processing, the accelerator generates an interrupt back to the processor.

C. Hardware-Software interaction

A new setting of the region of interest (ROI) or the FoV (zoom factor) disrupts this execution flow. The scalar processor first reads the coordinates of the ROI center point (u_{cp}, v_{cp}) from the tracker and transforms them in the perspective space using forward mapping. The forward mapping operation is only carried out for a single point when the ROI settings change, and is therefore out of the critical path.

Table 2. Control and Monitor code running on the Microblaze processor.

```
Power Up Sequence : Load the Bipartite Tables
while (1) {
 Read the center point (CP) coordinates of the fisheye ROI from
 the tracking module and call forward_mapping() to compute
 the rotation matrix (r_ij);
 Call inverse_mapping() to compute the fisheye space
 coordinates of the top left pixels (u_0,0, v_0,0) for each of the 200
 blocks of the perspective space;
 Compute the sizes and locations of the bounding boxes for the
 input tiles;
 /* Main loop for each frame */
 while (values for Pan, Tilt, Zoom remain unchanged) {
  tile = 0;
  for (i = 0; i < 203; i++) {
      Set up stream descriptors and accelerator for tile;
      Trigger accelerator;
      Wait for interrupt from accelerator;
      if (tile < 200) tile++;
  }
  /* End of frame checks */
  Check for new values for Pan, Tilt, or Zoom;
 }
}
```

The accelerator then proceeds to use the scalar version of the inverse mapping algorithm to compute the fisheye space coordinates of the top-left pixel *TL* for each of the 200 tiles of the frame. This is necessary because these addresses will be fed to the accelerator to be used as a starting address for the input streams. This part of the code is also used to determine the bounding boxes, and therefore, the skip and span settings of the input pixel stream. For example, the width of the bounding box *BBi* is given by:

$$W_{BBi} = \lceil \max(U_{TL(i+1)}, U_{TL(10+1+i)}) - \min(U_{TL(i)}, U_{TL(10+i)}) \rceil + dw$$

where $U_{TL(i)}$ is the column coordinate of the top left pixel of the BBi at the fisheye space.

A. Performance evaluation

Using the bit-exact, software version of the fisheye lens distortion correction algorithm, we compared the performance of the hardware accelerator to the performance of the Intel Core 2 Quad multi-core processor. In order to make a fair comparison, we optimized the code to exploit the quad-threaded, SIMD architecture of Core 2 Quad by using the OpenMP library and by manually rewriting the inner loops of the IM, BI and LPF kernels using the SSE ISA extensions. We used Intel's *icc* compiler with the O3 optimization flag to enable aggressive optimizations. Moreover, we make full utilization of the processor's FP units for the computation of the fractional coordinates.

Table 3 shows the execution time of the software implementation running on the 2.5 GHz Core 2 Quad processor and our FPGA hardware running on the Virtex-4 LX-80, 62.5 MHz FPGA. The fully optimized Core 2 Quad version cannot meet real-time requirements at only 5.26 fps processing rate and is very inefficient in handling such streaming workload.

Table 3 also shows that the FPGA solution is 167.2 times more efficient in utilizing its hardware resources than the high performance processor, and 668.8 times more efficient per each thread running on a single core. These numbers are a testament of the superiority of reconfigurable over general purpose computing for application specific platforms.

V. RELATED WORK

Algorithms for fisheye lens distortion correction have been proposed for over two decades [11]. Most implementations that we know of are based on software running on a desktop processor, but there are embedded systems for fisheye lens distortion correction using FPGAs[14].

There has been an intense interest in the research community in the last decade to automate the architectural process for ASIC of FPGA tool flows starting from a high level representation like C, Java, Matlab, DFGs etc. [9].

Relevant projects and commercial tools include the OCAPI tool from IMEC [21], the DEFACTO compiler from USC [28], the ASC streaming compiler effort from the Imperial College [18], and the CASH compiler from CMU that maps a C application onto asynchronous circuits [25]. The Impulse-C [19] and Handel-C [7] languages are efforts to utilize C with extensions as a high level RTL language for FPGA design. At an even higher level of abstraction, AccelChip [3] is

Table 3. Performance comparison between the Core 2 Quad and the FPGA implementation

Implement.	Frame rate (fps)	Speedup over SW	Speed Up per Hz	Speed Up per Hz per thread
Software	5.26	1	1	1
Hardware (FPGA)	22	4.18	167.2	668.8

commercializing a compiler to automatically generate gates from Matlab code. The PICO project [15] incorporated a lot of concepts from earlier work on VLIW machines, and described a methodology to generate a VLIW engine along with an accelerator optimized for a particular application. The C2H tool from Altera Corporation and the Catapult-C from Mentor Graphics use an ANSI-C to gates methodology to automate architectural synthesis.

VI. CONCLUSIONS

In this paper, we have presented a pipelined architecture in a reconfigurable platform for a fisheye lens distortion correction algorithm. We follow a methodology that includes coding a given functional code with source-level optimizations in a streaming language. This is done so that our ESL tool, Proteus, can efficiently map the streaming application into hardware accelerators. Design issues such as memory bandwidth are alleviated by tiling and pre-fetching image data from memory.

Using a design automation tool, we were able to quickly scan the space of Pareto-optimal implementations for a variety of area, throughput and clock frequency settings, and meet the system frame rate requirements.

The most interesting future direction is to investigate the automation of such source level optimizations with a minimal user feedback. Although there have been considerable academic research in this subject, and some commercial tools have appeared in the market, no solution for an efficient software-oriented, C-to-gates compiler has become mainstream.

REFERENCES

[1] Amarasinghe S., Thies B. Architectures, Languages and Compilers for the Streaming Domain. Tutorial at the 12th Annual International Conference on Parallel Architectures and Compilation Technique (PACT), New Orleans, LA

[2] Ray Andraka. A survey of CORDIC algorithms for FPGA based computers. International Symposium on FPGA, February 1998, Monterrey, CA

[3] Banerjee P. et. al. A MATLAB compiler for distributed, heterogeneous, reconfigurable computing systems. Proceedings of the IEEE Symposium on Field Custom Computing Machines (FCCM), April 17-19, 2000, pp. 39-48, Napa Valley, CA

[4] Nikolaos Bellas, Sek Chai, Malcolm Dwyer, Dan Linzmeier. Template-based generation of streaming accelerators from a high level representation. International Symposium on Field-Programmable Custom Computing Machines (FCCM), April 24-26, 2006, Napa Valley, CA

[5] Nikolaos Bellas, Sek Chai, Malcolm Dwyer, Dan Linzmeier. FPGA implementation of a license plate recognition SoC using automatically generated streaming accelerators. 13th Reconfigurable Architectures Workshop (RAW), 25-26 April 2006, Rhodes, Greece

[6] E. Caspi, et. al.. Stream Computations Organized for Reconfigurable Execution (SCORE). Proceedings of the International Conference of Field Programmable Logic (FPL), pp. 605-614, August 2000

[7] Celoxica Corporation, Handel-C language reference manual, www.celoxica.com

[8] Compton K., Hauck S.. Reconfigurable Computing: A Survey of Systems and Software. ACM Computing Surveys, vol. 34, No. 2, June 2002, pp. 171-210

[9] W.J. Dally, et. al.. Merrimac: Supercomputing with streams. Proceedings of the SuperComputing SC'03 Conference, pp. 35-43, November 2003, Phoenix, AZ

[10] N. Greene. Environment Mapping and Other Applications of World Projections. IEEE Computer Graphics and Applications. Nov 1986, vol. 6, no. 11, pp. 21-29.

[11] Hannes Hassler, Naofumi Takagi. Function Evaluation by Table Look-Up and Addition. Proceedings of the 12th Symposium on Computer Arithmetic, pp. 10-16, July 1995, Bath, UK

[12] N. D. Jankovic, M. D. Naish. Developing a Modular Active Spherical Vision System. Proceedings of the 2005 IEEE International Conference on Robotics and Automation, Barcelona, Spain, April 18-22, 2005

[13] J. Jiang, et. al, "Distortion correction for a wide-angle lens based on real-time digital image processing", Optical_Engineering, July 2003, Vol 42, No. 7, pp 2029-2039

[14] V. Kathail, S. Aditya, R. Schreiber, et. al., "PICO: automatically designing custom computers", *Computer*, vol. 35, no 9, Sept. 2002, pp. 39 – 47.

[15] Robert Keyes. Cubic Convolution Interpolation for Digital Image Processing. IEEE Transactions on Acoustics, Speech ,and Signal Processing. Vol. ASSP-29, No. 6, December 1981.

[16] Miriam Leeser, Shawn Miller, Haiqian Yu. Smart Camera Based on Reconfigurable Hardware enables Diverse Real-Time Applications. Proceedings of the 12th IEEE Symposium on Field Custom Computing Machines (FCCM), April 2004, Napa Valley, CA

[17] Mencer O., Pierce D. J., Howes L.W., Luk W. Design Space Exploration with a Stream Compiler. Proceedings of the IEEE International Conference on Field Programmable Technology (FPT), December 2003, Tokyo, Japan

[18] Pellerin D., Thibault S. Practical FPGA Programming in C. Prentice Hall, 2005

[19] Michael Schulte, James Stine. Symmetric Bipartite Tables for Accurate Function Approximation. Proceedings of the 13th Symposium on Computer Arithmetic, pp. 175-183, 1997, Los Alamitos, CA

[20] Schaumont P., Vernalde S., Rijnders L., Engels M., Bolsen I. A programming environment for the design of complex high speed ASICs. Proceedings of the 35th Design Automation Conference (DAC), June 1998, pp. 315-320, San Francisco, CA

[21] E. Schwalbe, "Geometric Modeling and Calibration of FishEye Lens Camera Systems," Proceedings of the ISPRS Working Group, Panoramic Photogrammetry Workshop, Berlin, Germany, Feb 2005, ISBN 1682-1750, vol 34-5/W8

[22] Wang, Ming-Liang, et. al. An Intelligent Surveillance System Based on an Omnidirectional Vision Sensor, IEEE Conference on Cybernetics and Intelligent Systems, June 2006, pp. 1-6.

[23] Virtex-4 Handbook, www.xilinx.com, August 2004

[24] Budiu M., Venkataramani , Chelcea T., Goldstein S.C. Spatial Computation. Proceedings of the 11th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS), October 9-13, 2004, pp. 14- 26, Boston, MA

[25] Vidiu M., Venkataramani , Chelcea T., Goldstein S.C. Spatial Computation. Proceedings of the 11th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS), October 9-13, 2004, pp. 14- 26, Boston, MA

[26] Tsuyoshi Yamamoto, Munehiro Doi. Design and Implementation of Panoramic Movie System by Using Commodity 3D Graphics Hardware. Computer Graphics International (CGI) p. 14-19, July 2003, Tokyo, Japan

[27] H. Ziegler H., Hall M. Evaluating Heuristics in Automatically Mapping Multi-Loop Applications to FPGA Proceedings of the 13th International Symposium on FPGAs, February 2005, pp. 184-195, Monterey, CA

[28] S. Zimmermann, D. Kuban. A video pan/tilt/magnify/rotate system with no moving parts. IEEE Digital Avionics Systems Conference, Oct. 1992, pp.523 – 531.

[29] News Release, www.4i2i.com, April 2005

A patent is pending that claims aspects of items and methods described in this paper.