Implementation of a Foveal Vision Mapping

Donald G. Bailey [#1], Christos-Savvas Bouganis [*2]

[#] *School of Engineering and Advanced Technology, Massey University*
Palmerston North, New Zealand
[1] `D.G.Bailey@massey.ac.nz`

[*] *Department of Electrical and Electronic Engineering, Imperial College London*
London, United Kingdom
[2] `christos-savvas.bouganis@imperial.ac.uk`

***Abstract*—Foveal vision reduces the data volume by utilising a spatially variant resolution. A high resolution is maintained in the fovea where it can be used by computer vision algorithms, while the resolution is reduced in the periphery where it is less important. This work focuses on the hardware architecture of a system that maps a conventional high resolution uniformly sampled sensor to a variable resolution output. The key feature of the proposed architecture is that it employs a separable forward mapping which requires a small amount of FPGA resources. This enables an efficient implementation of a continuously variable spatial resolution, requiring only 1000 LUTs on a Virtex-5, and runs at 104 MHz, enabling the mapping of a 512×512 window at over 300 frames per second.**

I. Introduction

Very high resolution sensors (up to 10 megapixels) are now a commodity item. The resulting increase in resolution usually has a positive impact on the overall performance of many computer vision algorithms. However, it also creates a computational burden in processing systems, especially when real-time constraints have to be met. This results in a trade-off between high resolution and processing power especially for embedded applications.

While it is possible to process such data in real time using dedicated hardware, any algorithm that requires multiple frames will require significant off-chip memory access, with its consequent bandwidth bottleneck. To enable on-chip storage, the volume of data, hence image size, must be reduced considerably. The simplest way to reduce the data volume is to reduce the image size, and hence resolution. The consequence of this is a loss of information that may be critical in many applications.

In many applications, such as tracking and pattern recognition, it is not so important to maintain the same resolution across the image sensor as to have a wide field of view. In such cases, high resolution is often only critical in small regions of the image. This implies a multi-resolution approach, or a spatially variant resolution within the sensor. A foveated window, inspired by the human visual system, maintains a high resolution within the fovea, with decreasing resolution towards the periphery. The variable spatial resolution provides a balance between high resolution, and large data volume. Compared to a uniform lower resolution image, the increase in resolution in the centre comes at the expense of a decrease in resolution at the periphery.

An important part of such systems is the use of active vision techniques (mechanical or electronic based systems) to ensure that the high resolution part of the sensor corresponds to the region of the scene where it can be most effective. The variable spatial resolution enables a significant data reduction (a factor of 22 in [1] and 64 in [2]) without a severe impact on the final performance of the application. Active vision requires minimising the latency between image capture and repositioning the fovea, because delays may degrade the performance of the application and limit the stability of this control loop.

A. *Structure of the paper*

Section II briefly reviews prior work on spatially variant imaging, provides an overview of the different approaches taken, and discusses the limitations of the various methods in the context of active vision. The architecture of our system is presented in section III, along with a discussion of the mapping issues associated with processing the image data as it is streamed from the camera. Section IV provides details of the implementation for a separable mapping (one that maps the horizontal and vertical coordinates independently). The logic requirements and operating speed for variations of the basic design are analysed. Finally, section V discusses the characteristics and performance of the design and compares our design with others in the literature.

B. *Novel Contributions*

Normally, for geometric transformations, a reverse mapping is used. This paper uses a forward mapping because it enables the image data to be transformed as it is streamed from the sensor. Symmetry enables a half width mapping to be used, and for efficiency, the forward mapping is derived from the smaller reverse map on the fly. The use of a single small table to define the map enables the mapping to be dynamically changed.

II. Prior Work

Many configuration topologies have been introduced in the literature for spatially variant imaging. Most are inspired by the human vision, having a resolution that decreases with the distance from the centre of the sensor. There are four main approaches for acquiring an image of variable spatial resolution.

1) *Optical Techniques:* Kuniyoshi et al. [3] achieved a variable spatial resolution optically by using a specifically designed lens that mimics the acuity of the human visual system. They concluded that the design of the optical system with the above characteristics was challenging and some digital processing was still a necessity. An alternative optical approach is to combine a low resolution and high resolution sensor with a beam splitter [4]. The two detectors were coupled using a two-axis MEMS scanner to enable the high-resolution sensor to be mapped anywhere within the field of view of the low-resolution sensor.

2) VLSI Approach: A second approach is to specifically design a VLSI sensor with variable spatial resolution. Etienne-Cummings et al [5] present a 2D foveated silicon retina with two static areas with different resolutions. The main drawback of the VLSI approach is the fixed topology (the fovea position must be scanned mechanically). This problem is overcome by Vogelstein et al. [6] by dynamically combining adjacent sensing elements to achieve varying spatial resolution. Their system was based on integrate-and-fire neurons. By pooling events, a lower resolution was achieved in the periphery.

3) Software Emulation: An alternative is to use a standard uniform resolution sensor, and emulating a variable resolution sensor by performing a mapping in software. Many researchers have adopted this approach due to the low cost and high flexibility that it offers. However, for real-time applications, the required processing time limits its applicability.

4) Hardware Emulation: The processing limitations may be overcome by performing the mapping with either a VLSI chip or an FPGA. This is the approach taken in this paper.

Camacho et al. [7] used an FPGA to interface to a progressive scan CCD camera. The FPGA was responsible for both mapping and processing. Their foveation mechanism was based on a pyramidal system, with successive levels within the fovea increasing the resolution by a factor of two, giving a rectangular log-Cartesian topology. This gives a stepped resolution, rather than one which is continuously variable.

Ovod et al. [8] used an FPGA for real-time image processing with multiple fovea. The limitation of their system was that it allowed only two levels of resolution.

Arribas and Macia [9] implemented a log-polar mapping using an FPGA that was able to achieve real-time performance. They used a large lookup-table to map each input pixel to a corresponding output pixel. The image was mapped using a forward mapping after capturing it in a frame buffer, rather than directly performing the mapping as the data was streamed in. Each input pixel was associated with only a single pixel in the foveal image, limiting the transition from the fovea to the periphery.

More recently, Martinez and Altamirano [1] proposed an FPGA pipelined architecture that transforms Cartesian images to a foveated image. They argue that the non-linearities introduced in the image representation resulting from a conventional log-polar transformation makes current computer vision algorithms such as correlation-based detection or recognition hard to apply. They address this problem by using a high-resolution rectangular sampling within the fovea, and in the periphery approximate a log-polar mapping by multiple square regions of different resolutions. They use a simple sub-sampling within the periphery, which can lead to aliasing, especially if there is significant fine detail in the periphery.

The proposal of [10] addresses many of the limitations of the systems described above. In particular, it provides a continuously variable resolution from the fovea to the periphery.

III. Foveal Mapping

A. Our Architecture

To exploit the high resolution available from commodity CMOS sensors, the architecture shown in Fig. 1 was proposed by the authors in [10]. The high pixel count enables a wide-angle lens to be used without a loss of resolution compared to that of a standard camera. However, the high pixel count also has the disadvantage of limiting the frame rate.

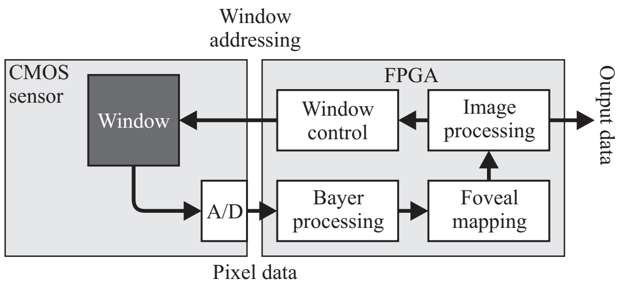

Fig. 1. System architecture

CMOS sensors may overcome this problem, by allowing any arbitrary rectangular region or window of the sensor to be read out. Since the readout window is programmable, it may be positioned anywhere within the field of view of the camera under programme control. Repositioning the window from one frame to the next provides the equivalent of a very fast pan and tilt. As the camera does not move, both the latency and motion blur associated with physically panning or tilting the camera is avoided.

Digital CMOS sensors have integrated analogue to digital conversion, enabling direct connection to an FPGA. Single chip colour sensors use a Bayer pattern or similar colour filter array. Adjacent pixels must be therefore interpolated to obtain a full colour image. The uniformly sampled image read from the camera is then resampled using a foveal mapping to give a variable resolution image that is significantly reduced in size. The small foveal image is then processed depending on the application to extract the required output. Part of this processing is to determine the next location for the fovea, which is used to control the position of the readout window in the sensor, closing the feedback loop shown in Fig. 1.

B. Definitions

In [10], a family of mappings was proposed that maintain Cartesian coordinates in the warped image. Let (x_u,y_u) be the position of a pixel in the input window, where the origin is defined as the centre of the window (the foveation point). Also let u be the distance of this point from the centre of the input window using some distance metric. Similarly, let (x_f,y_f) be the coordinates and f be the distance from the centre of the foveated image. The foveal mapping can then be defined either in terms of the forward mapping

$$f = \mathrm{map}_f(u) \tag{1}$$

or the reverse mapping

$$u = \mathrm{map}_r(f) \tag{2}$$

This paper explores the implementation of the separable mapping, where the mapping is applied independently to the x and y coordinates:

$$\begin{aligned} x_f &= \mathrm{sign}(x_u) \times \mathrm{map}_f\left(|x_u|\right) \\ y_f &= \mathrm{sign}(y_u) \times \mathrm{map}_f\left(|y_u|\right) \end{aligned} \tag{3}$$

The separable mapping was chosen for implementation, because of the different mapping schemes proposed in [10] it is simplest and most efficient to implement. Earlier simulations found that there is only minor difference in tracking performance compared to more complex radial mappings [2].

Without loss of generality, the reverse mapping

$$\mathrm{map}_r(f) = f + \tfrac{7}{32} f^2 \tag{4}$$

will be considered throughout this paper. This particular mapping gives a 64-fold reduction in data volume (from 512×512 to 64×64). The effect of eq. (4) as a separable mapping is illustrated in Fig. 2. High resolution is maintained in the centre of the image, and for this mapping, a pixel in the periphery corresponds to approximately 15×15 pixels in the input image.

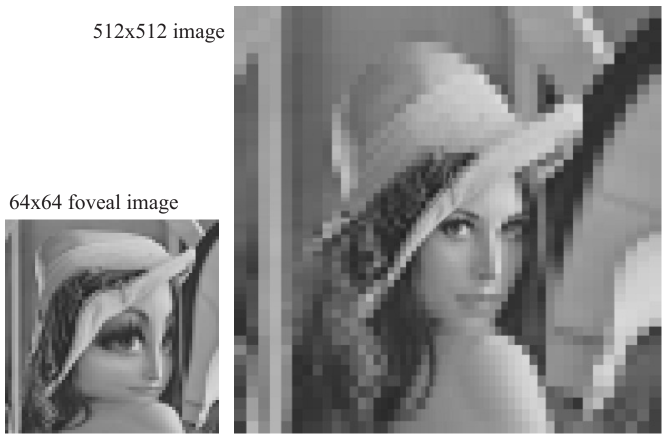

Fig. 2. Example foveal image from the mapping of eq. (4)

A. Reverse Mapping

The reverse mapping, defined by eq. (2), is usually used to warp an image [11]. This determines the corresponding location in the input image for each output pixel, using interpolation to handle fractions of pixels. The reverse mapping is advantageous when the output pixels must be produced in a particular order, for example when streaming the output for a display. However, there are several problems with using a reverse mapping for implementing the foveal mapping.

First, it requires random access to the input image to select the corresponding input pixels for each output pixel. Implementing the mapping in this way would require significant memory for frame buffering, which was contrary to the goal of foveated image processing. Storing the image in an intermediate frame buffer also introduces latency into the transformation, which will reduce the ability to rapidly control the positioning of the fovea within an active vision system.

The foveal mapping results in down-sampling the image which can result in significant aliasing, particularly in the periphery. This must be overcome by pre-filtering with an anti-aliasing filter. With the spatially variant resolution, this requires a spatially variant filter, with little or no filtering required in the fovea, and a large window in the periphery.

B. Forward Mapping

These problems may be overcome by using the forward mapping, defined by eq. (1), which determines where in the output image each input pixel maps to. A forward mapping enables the input pixels to be processed as they are streamed from the camera. This minimizes the need for frame buffering, gaining the maximum benefit from the reduction in data volume from the foveal mapping. Processing each pixel as it arrives can minimize the latency of the mapping stage.

Since the foveal mapping involves a reduction in resolution, it results in a many-to-one mapping. Aliasing may be reduced by simulating a low resolution sensor by averaging all of the input pixel values associated with each output pixel. Such averaging is implicitly implementing a spatially variant filter.

A continuous mapping function is required to give a smooth transition between the fovea and periphery. This implies that it is possible for an input pixel to fall on the boundary between adjacent output pixels. In such cases, the input pixel value must be apportioned to the corresponding outputs.

IV. Separable Mapping Implementation

A separable mapping enables the X and Y directions to be mapped separately, which reduces the required logic.

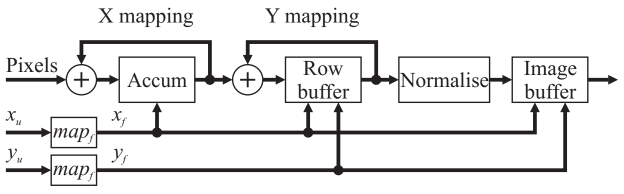

Fig. 3. Basic structure of the implementation

The basic structure of the implementation is shown in Fig. 3. The mapping is effectively a two-pass algorithm similar to the form first described by Catmull and Smith [12]. The first pass, or X mapping, assigns pixels from the incoming pixel stream into the correct output column as defined by the forward mapping. The accumulator accumulates the pixel values and areas until the width of the corresponding output pixel is completed, at which point it is passed to the Y mapping circuit. The second pass, or Y mapping, then operates on the column to place each pixel in the correct output row.

Rather than operate on one column at a time, the Y mapping is performed on all of the columns in parallel. This processes the pixels in the order that they are streamed out from the first pass. The row buffer caches the accumulators for each column of the output image. As each output row is completed, the accumulated pixel values are normalised and both saved into the image buffer and streamed for subsequent processing.

A. X Mapping Details

The detailed implementation of the X mapping is shown in Fig. 4. The forward mapping needs to map each pixel from the raster-scanned input stream onto its corresponding output position. The need to map multiple input pixels to a single output pixel, and to split pixels that are on the border of two output pixels, complicates the creation of a forward mapping table.

However since there are fewer output than input pixels, the table for the reverse mapping requires significantly less storage than that for the forward mapping. The reverse mapping effectively specifies the position of the borders of the foveal pixels in terms of input pixel coordinates. Therefore, exploiting the fact that input pixels are streamed in sequentially, the forward mapping may be implemented by using the reverse mapping to detect when the input address matches the boundary between the current and next output pixel. The reverse mapping table is not only smaller, but it also manages the splitting of input pixels more naturally.

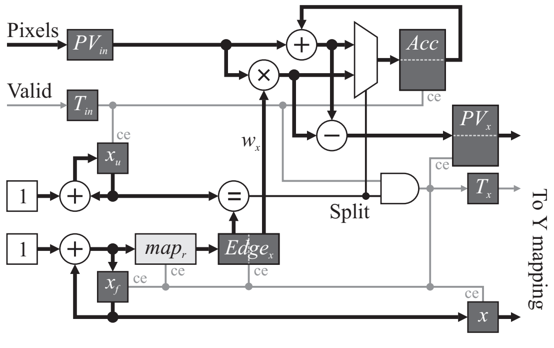

Fig. 4. Detailed implementation of the X mapping.

The implementation of the forward mapping process is as follows: The next foveal pixel location (x_f+1) is looked up in the reverse mapping, and the edge stored in the register $Edge_x$. When the integer part matches the input pixel location, it indicates that the current input pixel must be split between two output pixels. The fractional component of the mapping, w_x, is used to determine the proportion of the input pixel value that maps to the next foveal pixel. Since the weight is small, typically only a few bits, it can be implemented efficiently in the FPGA fabric using adders rather than dedicated multipliers.

If the pixel is not split, it is simply added into the accumulator, *Acc*. If the pixel is split, the fraction belonging to the next foveal pixel is stored in the accumulator, and the accumulated pixel value is passed to the Y mapping.

Note that the accumulator, *Acc*, has two components: the total accumulated pixel value, *V*, and the area or number of input pixels accumulated, *A*. Dividing one by the other will enable the output pixel value to be normalized. Defining *Acc*[*V*,*A*] to be the composite accumulator, it is updated as

$$Acc\left[V,A\right]=\begin{cases}\left[w_x\times PV_{in},w_x\right] & \text{if split}\\ Acc\left[V,A\right]+\left[PV_{in},1\right] & \text{otherwise}\end{cases}\tag{5}$$

where PV_{in} is the current pixel value streamed in from the sensor. The corresponding output if the input pixel is split is

$$\begin{aligned}PV_x[V,A]&=Acc\left[V,A\right]+(1-w_x)\times\left[PV_{in},1\right]\\&=Acc\left[V,A\right]+\left[PV_{in},1\right]-\left[w_x\times PV_{in},w_x\right]\end{aligned}\tag{6}$$

Since the output will occur sporadically (especially in the periphery), a token passing mechanism is used to indicate when valid data is available. This is connected to the clock enable inputs of the various registers. The token passing and enabling path is shown in grey in Fig. 4. The accumulator is latched and the input position incremented whenever a valid pixel is input. When a pixel is split, the foveal position is incremented, and the next edge is obtained from the reverse mapping. The token is passed to the Y mapping as T_x, along with the accumulated value and corresponding *x* position.

B. Reducing the Mapping Size Using Mirroring

Since the mapping is symmetrical, only one half of the mapping needs to be stored and represented. This requires mirroring the input and foveal image coordinates to enable the reduced map to be used. While this will increase the logic, it will simplify the specification for the mapping, especially in the case where the mapping itself is changed dynamically from one frame to the next. Fig. 5 updates Fig. 4 with the additional logic needed for mirroring.

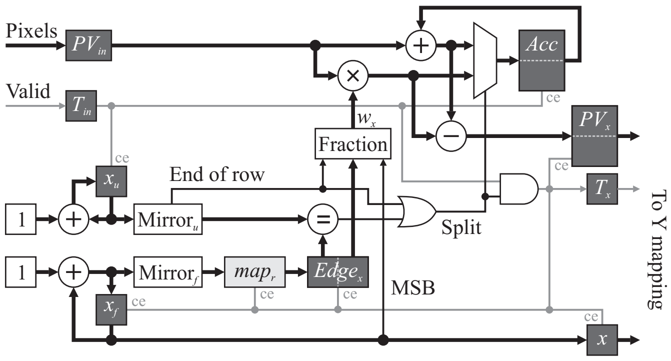

Fig. 5. Introducing mirroring into the X mapping.

The two blocks, Mirror$_u$ and Mirror$_f$, perform the mirroring on the input and foveated image pixel positions respectively to take into account the symmetry. The mapping of 16 input pixels onto 8 pixels in the foveal image will be used to illustrate the requirements. One such mapping is

$$\text{map}_r(f)=f+\tfrac{1}{4}f^2\tag{7}$$

This mapping is illustrated in Fig. 6, with the corresponding mirror functions tabulated in Table I.

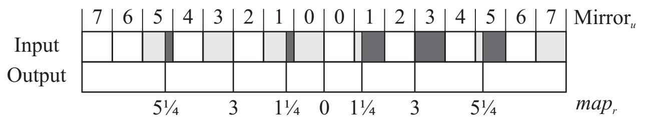

Fig. 6. Simplified example using the mapping of eq. (7). Light shaded input pixels are on or span the border of the output pixels (split pixels). The dark shaded regions represent the fraction of the split pixel assigned to the next output.

TABLE I
EXAMPLE MIRROR FUNCTION AND FRACTION CALCULATIONS FOR EQ. (7)

x_u	Mirror$_u$	EoR
0	7	0
1	6	0
…	…	0
6	1	0
7	0	0
8	0	0
9	1	0
…	…	0
14	6	0
15	7	1

x_f	x_f+1	Mirror$_f$	map_r	MSB	Fraction
0	1	3	101.01	0	0.01
1	2	2	011.00	0	0.00
2	3	1	001.01	0	0.01
3	4	0	000.00	0	0.00
4	5	1	001.01	1	0.11
5	6	2	011.00	1	1.00
6	7	3	101.01	1	0.01
7	0	-	-	1	0.00

The mirror function of the input pixel position is given by

$$\text{Mirror}_u(x_u) = \text{LSBs}(x_u) \oplus \overline{\text{MSB}(x_u)} \tag{8}$$

with the most significant bit used to invert the least significant bits. Note that this labels the pixels symmetrically from the centre, with two input pixels labelled as zero: one negative zero, and the other positive zero.

The mirror function for the foveal image can only have one zero, corresponding to the centre (as shown in Fig. 6). Therefore the foveal mirror function must be:

$$\text{Mirror}_f(x_f+1) = \left(\text{LSBs} \oplus \overline{\text{MSB}}\right) + \overline{\text{MSB}} \tag{9}$$

Mirroring also requires the fractional component to be mirrored, as illustrated in Fig. 6 and Table I.. The corresponding weight, as implemented by the Fraction block in Fig. 5 is given by

$$w_x = \begin{cases} \text{Fract}(Edge_x) & \text{if MSB}(x_f)=0, \\ 0 & \text{if } x_u = \text{End of Row} \\ 1-\text{Fract}(Edge_x) & \text{otherwise} \end{cases} \tag{10}$$

C. Y Mapping Details

The Y mapping logic is very similar to that for the X mapping (refer to Fig. 7). However, the input to the Y mapping is streamed horizontally rather than vertically. Therefore, all of the columns must be mapped in parallel. This requires maintaining a separate accumulator for each output column. Each input value from the X mapping corresponds to a single column, so only one accumulator needs to be accessed at a time. Therefore rather than store each accumulator directly in a separate register, the accumulators are most efficiently stored in a memory. The memory is indexed by the column, x, with the accumulated value written back on the following clock cycle. A dual-port memory separates the read and write operations.

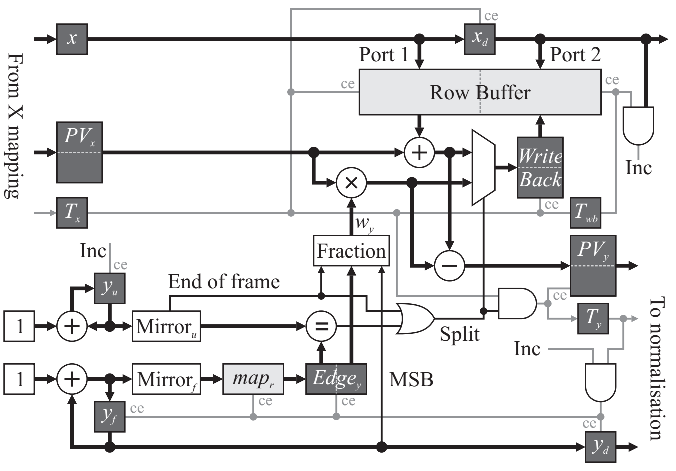

Fig. 7. Detailed implementation of the Y mapping.

The input row address is incremented at the end of each row (when x_d is all 1s). The foveal row address only needs to be incremented at the end of split rows, when a new output row is begun.

The mapping lookup table is only read and stored in the $Edge_y$ register when the foveal row address is changed. This enables the lookup table and associated logic (Mirror$_f$ and incrementing) to be shared between the X mapping and Y mapping.

On a split row, every incoming pixel is split between the new row and the completed accumulated row. The weight of the input pixel, w_y, is calculated in the same way, using eq. (10). The difference is that two multiplications are required, one for the value, and one for the area. Again, the multiplications are small (the fraction is only a few bits wide), enabling them to be implemented efficiently using adders. Equation (5) therefore becomes

$$Acc[x][V,A] = \begin{cases} w_y \times PV_x[V,A] & \text{if split} \\ Acc[x][V,A] + PV_x[V,A] & \text{otherwise} \end{cases} \tag{11}$$

with the accumulated output value for the corresponding column given by

$$\begin{aligned} PV_y[V,A] &= Acc[x][V,A] + (1-w_y) \times PV_x[V,A] \\ &= Acc[x][V,A] + PV_x[V,A] - w_y \times PV_x[V,A] \end{aligned} \tag{12}$$

Both the delayed x_d and y_d positions are also output from the Y mapping stage to provide the address of the foveal pixel generated.

Again, a token passing mechanism is used to synchronise the logic. T_{wb} delays the write back of the accumulated result to the row buffer to the following clock cycle. The output of an accumulated foveal pixel is indicated by T_y.

D. Normalisation

The final stage before writing the accumulated pixel value to the output frame buffer is normalisation. This divides the accumulated value by the accumulated area as shown in Fig. 8.

$$\text{Image}(yd, xd) = \frac{PV_y[V]}{PV_y[A]} \quad (13)$$

Since the output is a pixel value (typically 8 bits), the division required for this may be implemented efficiently by using 8 iterations of a non-restoring division algorithm [13]. This uses approximately the same logic as required for the corresponding sized multiplication [13], although the propagation delay is significantly longer because of carry propagation.

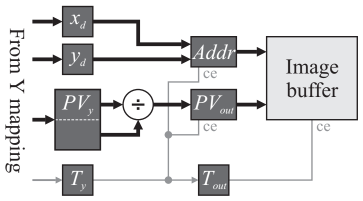

Fig. 8. Detailed implementation of normalisation and image saving.

Although the division is implemented within a single clock cycle in the baseline implementation, the operation can readily be pipelined over two or more clock cycles [13], and this option was also evaluated.

E. Implementation Performance

A number of parameters need to be determined in order to evaluate the logic required. The baseline implementation is for greyscale images with 8 bits per pixel. Adapting to colour triple the width of the value accumulators, although only a single area accumulator would be required. It would also triple the number of multipliers and dividers needed. Since these use most of the logic, it is expected that the logic requirements for colour images would be slightly less than triple that for greyscale images. For comparison, designs from 4 bits per pixel to 12 bits per pixel were also implemented.

As multiple input pixels are accumulated, it is necessary to allow sufficient bits in the accumulator to prevent overflow. The mapping of eq. (4) averages 15 pixels along a row in the periphery. The row accumulator therefore requires an extra 4 bits, and the column accumulators require an extra 8 bits to avoid overflow. A larger fovea would have even lower acuity in the periphery, so variations of the circuit with 5 row and 9 column bits, and 5 row and 10 column bits were also implemented for comparison.

A further parameter is the number of bits to represent the fractional pixel position of the foveal pixel boundaries. The fractional component limits the degree to which the resultant mapping has a continuously variable spatial resolution. In the periphery, where a large number of pixels are averaged, the fractional component is less important, however, in the fovea, a fine division is necessary to give a smooth transition. With 8 bit pixels, it is unnecessary for the fraction to also be more than 8 bits. In the baseline implementation, a 4 bit fractional component was arbitrarily chosen for each of the X and Y mappings. This means that the X mapping accumulators have 4 fractional bits. Since these get multiplied when a pixel spans two output rows, this requires a total of 8 fraction bits for the Y mapping stage. Designs with 2, 3 and 5 fractional bit components in the mapping were also implemented for comparison.

All of the implementations compressed a 512×512 window to a 64×64 foveal image. This represents a 64 fold reduction in data volume. Table II lists the different implementations, and gives the number of bits required for the accumulator registers at the different stages of the design. The three key parameters associated with each design are **P**/**A**/**F** where **P** is the number of bits per pixel in the input image, **A** is the number of bits required for accumulator overhead, and **F** is the number of fraction bits in the mapping.

TABLE II
NUMBER OF BITS REQUIRED FOR ACCUMULATOR REGISTERS AT VARIOUS STAGES OF THE DESIGN FOR THE DIFFERENT IMPLEMENTATIONS.

Scheme	**X Mapping**		**Y Mapping**	
P/A/F	**Value (V)**	**Area (A)**	**Value (V)**	**Area (A)**
Baseline 8/8/4	8+4+4 16	4+4 8	8+8+8 24	8+8 16
Pixel size 4/8/4	4+4+4 12	4+4 8	4+8+8 20	8+8 16
Pixel size 6/8/4	6+4+4 14	4+4 8	6+8+8 22	8+8 16
Pixel size 10/8/4	10+4+4 18	4+4 8	10+8+8 26	8+8 16
Pixel size 12/8/4	12+4+4 20	4+4 8	12+8+8 28	8+8 16
Colour pixel 24/8/4	3×(8+4+4) 48	4+4 8	3×(8+8+8) 72	8+8 16
Accumulator 8/9/4	8+5+4 17	5+4 9	8+9+8 25	9+8 17
Accumulator 8/10/4	8+5+4 17	5+4 9	8+10+8 26	10+8 18
Fraction bits 8/8/2	8+4+2 14	4+2 6	8+8+4 20	8+4 12
Fraction bits 8/8/3	8+4+3 15	4+3 7	8+8+6 22	8+6 14
Fraction bits 8/8/5	8+4+5 17	4+5 9	8+8+10 26	8+10 18

These eleven designs were implemented using Handel-C and were mapped onto a Xilinx Virtex-5 FPGA [14]. Three additional designs that introduced pipelining into the divider of the baseline and colour implementations were also considered. The logic utilisation summaries for the different designs as reported by the Xilinx ISE (using medium place and route effort) are listed in Table III. The baseline design uses fewer than 1000 LUTs, and will run at a clock speed of 64.9 MHz.

V. DISCUSSION AND CONCLUSIONS

First, the logic utilisation of the baseline configuration will be considered, and then the other implementations will be compared (refer to Table III for the data).

The single BlockRAM used by the baseline design is required to hold the output foveated image. The fact that a whole image is compressed into a single BlockRAM enables multiple frames to be both stored and processed on chip.

TABLE III
COMPARISON OF LOGIC UTILISATION AND CLOCK FREQUENCY FOR A RANGE OF IMPLEMENTATION OPTIONS.

		Baseline	**Pixel size**				**Accumulator**		**Fraction bits**			**Pipeline stages**		**Colour**	
		8/8/4	**4**/8/4 4 bpp	**6**/8/4 6 bpp	**10**/8/4 10 bpp	**12**/8/4 12 bpp	8/**9**/4 9 bits	8/**10**/4 10 bits	8/8/**2** 2 bits	8/8/**3** 3 bits	8/8/**5** 5 bits	8/8/4 2 stage	8/8/4 3 stage	**24**/8/4 3×8	**24**/8/4 2 stage
Flip flops		245	221	234	257	269	253	257	217	231	258	286	321	437	526
LUTs	**Logic**	784	520	652	916	1048	825	850	583	685	880	784	784	1998	1998
	DP RAM	80	72	76	84	88	84	88	64	72	88	80	80	176	176
	SP RAM	8	8	8	8	8	8	8	6	7	8	8	8	8	8
	Shift regs	2	2	2	2	2	2	2	2	2	2	14	20	2	14
	Route thru	69	69	68	69	69	69	71	46	55	82	70	71	123	126
	Total	943	671	806	1079	1215	988	1019	701	821	1060	956	963	2307	2322
Block RAMs		1	1	1	2	2	1	1	1	1	1	1	1	3	3
Max clock, MHz		64.9	108.6	87.0	49.9	42.6	62.3	62.0	70.0	66.5	64.2	103.9	107.0	61.8	103.9

The row buffer was implemented using dual-port RAM made from the LUTs. The wide bitwidth (40 bits total for the baseline implementation) combined with the relatively shallow depth (64 accumulators) makes the use of BlockRAM inefficient for this.

Similarly, the relatively small size of the reverse mapping meant that this was also best implemented in LUT RAM. In this design it was single port, although in applications where the mapping is changed dynamically, it would be better to implement it using dual-port memory, with the second port reserved for setting the mapping. Alternatively, if several preset mappings were required, these could be stored in a single BlockRAM, with the selected mapping indexed via a register. Using a single small table to specify both the X and Y mapping is a strong feature of the presented design.

Of the parameters tested, the number of bits per pixel had the most significant effect on both in terms of the resources required and the resultant clock speed. Designs with more than 8 bits per pixel require two BlockRAMs to store the resultant image. The strong dependence of the clock speed on the pixel width can be attributed to the number of stages required to implement the division within the normalisation. This implies that the division is the bottleneck, and the baseline design could be improved significantly by pipelining the normalisation. This is in fact borne out in the results – spreading the division over a two stage pipeline almost doubled the maximum clock speed. Spreading over three stages had very little extra effect, indicating that a two stage pipeline is sufficient to bring the division in line with the rest of the design. Pipelining had only a minor effect on the resources required. The logic requirements are unchanged, however more flip-flops are required for the pipeline registers. The increase in number of flip-flops was smaller than expected because the place and route tools made use of LUTs as shift registers.

Changing the number of bits required for both the accumulator overhead and the fraction representation had a small effect on the resources required. There were minor differences with clock speed, generally in proportion to the number of bits processed, which is consistent with the division being the bottleneck.

Moving from monochrome images to colour increased the resource requirements by about two and a half times (note that this does not take into account any logic that might be needed for colour filter array interpolation). However, since each colour plane can be processed in parallel, the effect on the clock rate is insignificant.

A typical pixel clock rate for streaming from a high resolution sensor is 48 MHz, so most of the designs tested were suitable without modification. However, if increased precision was required, for example 10 or 12 bits per pixel, then it would be essential to pipeline the division to maintain an adequate clock speed. A higher speed sensor, such as the 5 megapixel Micron M9TP031 [15], has a maximum clock speed of 96 MHz. This would require using pipelining for all of the designs. Running at 96 MHz, a 512×512 window could be read, and processed, at over 300 frames per second.

The design presented in this paper uses a fixed-point representation that is independent of the position within the image. An alternative would be to adjust the fixed point representation based on location in the image. In the fovea, fraction bits are more important, and there are fewer pixels accumulated. In the periphery, it is important to have a larger overhead to avoid accumulator overflow, and the fraction bits are less important. This implies that a dual fixed-point representation [16] would reduce the size of the storage and the width of both the multiplications and divisions. However, the logic required to maintain and switch between multiple representations could easily undo any savings made.

The design presented has a small resource footprint, leaving significant resources for the remainder of the vision application. The use of a forward mapping, rather than the more usual reverse mapping, enables the input image to be processed directly as it is streamed in from the sensor. This not only reduces the footprint, but also reduces the memory requirements, and gives the design a low latency. The last pixel is written to the image buffer only a few clock cycles after the last pixel is read from the sensor.

The variable data rate that results from the spatially varying resolution is effectively handled using a token passing control mechanism. There is nothing stopping an application from further processing the pixel stream as it is saved to the image buffer. However, any such processing would also have to manage the variable data rate. The significant reduction in data volume would also allow an application to process the small foveal image with reduced latency after it has been loaded into the image buffer.

The foveal vision mapping presented in this paper has a number of advantages over the already published work. First, the mapping is able to provide continuously variable resolution, rather than the stepped approaches of many of the other systems. Several of the competing implementations require that the image first be captured into a frame buffer prior to performing the foveal mapping. Our design is able to operate directly on the pixel data as it is streamed from the sensor, significantly reducing both the latency and resource requirements. All of the other designs we have found in the literature had a fixed mapping function (this is separate from the ability to move the position of the fovea). In contrast, our mapping function is dynamic, specified by a single reverse mapping table. This allows our design to change the mapping from one frame to the next if necessary.

The low latency in combination with the ease with which the mapping may be modified enable high-speed and high resolution active vision applications to be readily implemented.

References

[1] J. Martinez and L. Altamirano, "FPGA-based pipeline architecture to transform cartesian images into foveal images by using a new foveation approach," in *IEEE International Conference on Reconfigurable Computing and FPGA's*, San Luis Potosi, Mexico, 2006, pp. 1-10.

[2] D. G. Bailey and C. S. Bouganis, "Tracking performance of a foveated vision system," in *International Conference on Autonomous Robots and Agents (ICARA 2009)*, Wellington, New Zealand, 2009, pp. 414-419.

[3] Y. Kuniyoshi, N. Kita, K. Sugimoto, S. Nakamura, and T. Suehiro, "A foveated wide angle lens for active vision," in *IEEE International Conference on Robotics and Automation*, Nagoya, Japan, 1995, pp. 2982-2988.

[4] H. Hua and S. Liu, "Dual-sensor foveated imaging system," *Applied Optics,* vol. 47, pp. 317-327, 2008.

[5] R. Etienne-Cummings, J. Van der Spiegel, P. Mueller, and M. Z. Zhang, "A foveated silicon retina for two-dimensional tracking," *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing,* vol. 47, pp. 504-517, 2000.

[6] R. J. Vogelstein, U. Mallik, E. Culurciello, R. Etienne-Cummings, and G. Cauwenberghs, "Spatial acuity modulation of an address-event imager," in *11th IEEE International Conference on Electronics, Circuits and Systems (ICECS 2004)*, Tel-Aviv, Israel, 2004, pp. 207-210.

[7] P. Camacho, F. Arrebola, and F. Sadoval, "Multiresolution sensors with adaptive structure," in *24th Annual Conference of the IEEE Industrial Electronics Society (IECON '98)*, Aachen, Germany, 1998, pp. 1230-1235.

[8] V. I. Ovod, C. R. Baxter, M. A. Massie, and P. L. McCarley, "Advanced image processing package for FPGA-based re-programmable miniature electronics," in *Infrared Technology and Applications XXXI*, Orlando, Florida, USA, 2005, pp. 304-315.

[9] P. C. Arribas and F. M. H. Maciá, "FPGA implementation of a log-polar algorithm for real time applications," in *Conference on Design of Circuits and Integrated Systems*, Mallorca, Spain, 1999, pp. 63-68.

[10] D. G. Bailey and C. S. Bouganis, "Reconfigurable foveated active vision system," in *International Conference on Sensing Technology*, Tainan, Taiwan, 2008, pp. 162-169.

[11] G. Wolberg, *Digital image warping*. Los Alamitos, California: IEEE Computer Society Press, 1990.

[12] E. Catmull and A. R. Smith, "3-D transformations of images in scaline order," *ACM SIGGRAPH Computer Graphics,* vol. 14, pp. 279-285, 1980.

[13] D. G. Bailey, "Space efficient division on FPGAs," in *Electronics New Zealand Conference (EnzCon'06)*, Christchurch, New Zealand, 2006, pp. 206-211.

[14] Xilinx, *Virtex-5 FPGA User Guide* vol. UG190 (v4.4): Xilinx Inc., 2008.

[15] Micron, *MT9P031 5MP Image Sensor Product Brief*: Micron Technology Inc, 2008.

[16] C. T. Ewe, "Dual fixed-point: an efficient alternative to floating-point computation for DSP applications," in *International Conference on Field Programmable Logic and Applications*, Tampere, Finland, 2005, pp. 715-716.