FPGA Implementation of a Single Pass Real-Time Blob Analysis Using Run Length Encoding

J. Trein*, A. Th. Schwarzbacher† and B. Hoppe°

*Department of Electronic and Computer Science, Hochschule Darmstadt, Germany
†School of Electronic and Communications Engineering, Dublin Institute of Technology, Ireland
° hoppe@eit.h-da.de

In the fields of machine vision and image processing applications the blob analysis became a well known method to detect objects in a digital image. Together with the increasing resolutions and frame rates of recent digital video cameras the requirements on the hardware to perform the blob analysis are very high. Software implementations may not be able to accomplish a satisfying performance. Furthermore, existing hardware solutions require a processing of the picture in multiple passes. This paper describes the development of a novel FPGA algorithm, performing a high speed real-time blob analysis using only one single pass. The input data are compressed using run length encoding. This allows the use of a full configuration camera link interface with data rates of up to 680MByte/s. Furthermore, object properties are passed to the host PC on-the-fly which reduces the latency to a minimum while at the same time allows the reuse of object labels thus achieving a memory efficient implementation.

1. Introduction

Since the development of the digital image processing during the 1980s it has always been very important to extract information from an image to allow decisions depending on the image content. These automated decisions improved many production and scientific processes in machine vision and computer vision applications [1]. An important step while gathering information from an image is the detection and description of objects. One of the most common techniques to detect objects from of a two-dimensional image is the blob analysis [1]. It describes the segmentation and mapping of neighbour foreground pixels to detect an object and the characterisation of shape parameters to determine the properties of the objects found. Afterwards, these object properties or object features can be used for the classification of objects which in turn leads to the possibility of making machine made decisions.

Blob analysis is basically used for the detection, counting and sizing of objects. Figure 1 shows a typical application of the blob analysis. In the original image, on the left, are five plugs where the two overlapped plugs have to be detected. The middle image shows the thresholded version of the original image. Now, every white pixel is assumed to belong to an object. Hence, this image is used to perform the blob analysis itself. The right image shows a visualisation of the results. In this example, the overlapping objects can be found easily by the sum of their foreground (white) pixels.

Respective applications of the blob analysis are the detection of surface defects in materials such as packaging material or on glass used for TFT monitors. Another application is the final inspection of production outcomes, for example the correct packing and shapes of paper carton boxes [2]. The blob...
analysis is also used in position compensation [1] and to trigger line-cameras. One of the most famous paradigms to demonstrate the blob analysis is the detection of the correct filling of blister [3].

The growing demands on speed together with the increasing resolutions of digital video cameras require a real-time blob analysis [4]. The high performance of today’s digital cameras may only be realised in microprocessor based systems with serious performance constraints and high effort. The problem of a processor based system is the serial processing of data which cannot reach the performance of a parallel hardware implementation. Even modern processors which operate in the gigahertz range cannot compensate this performance gap because of a limited RAM bandwidth [2].

The rapid development of field programmable gate arrays (FPGAs) during the last years, offers new possibilities. Complex algorithms can now be implemented into hardware without a high increase in design effort and cost. Furthermore, FPGAs enable almost unrestricted parallel processing of data and are flexible to use due to their re-programmability. Modern FPGAs facilitate built-in memory blocks and multipliers which enable the realisation of time critical and memory extensive applications.

This paper describes the FPGA investigation and implementation of a real time blob analysis algorithm. Digital images are transferred from the camera into the FPGA and are analysed in real time. The resulting object features are then passed to a host PC for further processing. Also, the algorithm facilitates the calculation of numerous object features such as area, centre of gravity, contour length, orientation and image moments [5]. The hardware is based on a Silicon Software [6] frame grabber system where the FPGA is a XILINX Spartan II or Spartan III [7] device. It operates with a core frequency of 50MHz. Through the parallel processing of up to 32 pixels and a pipelined architecture it is possible to process 1600Mpixels/s theoretically. This, however, is limited to 680Mpixels/s because of the input interface. The implementation is carried out in a low-cost FPGA and therefore a cost efficient realisation of the developed real-time single pass blob analysis algorithm is possible. Furthermore, the algorithm is suitable for general purpose blob analysis. This means, it is highly adaptable to various applications through many parameters which may set, for example the bit width or the required object features. Together with existing pre-processing operators provided by Silicon Software the resulting implementation is a powerful machine vision tool.

2. Existing Solutions

In literature, such as in [8] and [9], numerous algorithms can be found to perform a blob analysis or an object labelling procedure. These algorithms are based on the idea that in a first step all pixels of objects are labelled with a respective object number. This operation requires two passes through the whole image. Afterwards, in a third pass, the features of the labelled pixels, and so the features of the objects, are calculated. Unfortunately, it is not possible to transfer this basic algorithm into hardware because of FPGA constraints. In a PC the captured image data is stored in the system memory. The microprocessor then has random access to this data. This is not feasible inside a FPGA, because the built-in block RAM is not sufficiently large for the required data volumes. Furthermore, a pipelined implementation for this algorithm is not possible as the data has to be stored first, before post-processing and performing the blob analysis can be carried out. Consequently, the analysis has to be performed in a raster-scan procedure at the rate of the camera transfer. Thus, random access on the image content is not possible. Moreover, it is not feasible to process the whole image twice because of a lack of available memory capacity as well as to reduce system latency. For example it is not possible to process the image from the bottom after the first pass like many algorithms require [9]. The novel proposed algorithm will address these problems and will successfully solve them as will be shown in the following section.

The main advantage of the developed algorithm is to be capable of performing the blob analysis in a pipelined single pass fashion. Through a compression, multiple pixels can be processed in parallel. The object features are passed on to the host PC as soon as they can be determined which reduces the latency to a minimum.

3. Developed Hardware Algorithm

The developed algorithm is based on the idea of directly capturing the frames from a digital camera, perform the blob analysis and afterwards pass the results on to a host PC. To transmit the frames between the camera and the FPGA the 'camera link' standard [10] is used as an interface. This interface allows the transmission of up to eight pixels in parallel at a clock frequency of up to 85 MHz and is used as a standard in machine vision applications. Furthermore, a timing synchronisation before data processing in the FPGA is required. Thus, the pixels are buffered in a SDRAM first. From this buffer, the pixels are read with a typical clock frequency of 50 MHz. As described, the clock frequency of the camera link is higher. To avoid a limitation of these data rates inside the FPGA a set
of subsequent pixels is read from the buffer in parallel. This may be up to 32 pixels which are transmitted in one clock cycle.

### 3.1. Run Length Encoding

The implemented algorithm is using a raster scan procedure to determine the objects and calculate their properties. This means every pixel of the image has to be scanned line by line to determine the objects. As described above, up to 32 subsequent parallel pixels are read from the buffer in one clock cycle. Hence a raster scan would not be possible. A run length encoding of these parallel pixels solves this problem. This procedure compresses the pixels and therefore allows a raster-scan procedure to determine the objects as it is shown in the next section.

### 3.2. Object Detection

The object detection is based on the idea that every foreground pixel is compared with its neighboured pixels. If one of the neighboured pixels is also a foreground pixel, they must belong to the same object. This is performed by scanning through the image line by line where every pixel is compared with its neighbours. The 'mask' in Figure 1 shows the relation of a pixel to its neighbours. If the current pixel has no other foreground pixels inside its mask it is assumed to be the start of a new object. Hence, the pixel is labelled with a new object number. If at least one of the neighbours of the current pixel is a foreground pixel, it has to belong to the same object and therefore gets the same label number as the neighboured pixel. These two operations are illustrated in Figure 2. In the left image the current pixel has no neighboured foreground pixels in its mask. Thus it is assumed to be the start of a new object and it is labelled with a new object number. In the right image, the current pixel has a neighboured pixel inside the mask. Hence, the current pixel belongs to the same object and is labelled with the same object number.

As explained, the pixels of this implementation are compressed by a run length code before this comparison. But the method of comparing pixels with its neighbours can also be used with runs. Here, a respective current run is compared with the runs in its previous line. If the start and end positions of the runs overlap, they have to belong to the same object. Thus they are labelled with the same object number. Figure 3 represents the same image such as the one shown in Figure 2. Here, the pixels are described by run length codes, hence described by the pixel-sequence start and end positions. In the left image, the current run does not overlap with any of the runs of the previous line. In the right image the current run overlaps with a run of the previous line. Therefore, they must belong to the same object.

As seen in this example, it is only necessary to compare a respective current run with the runs in the respective previous image line. The algorithm is scanning through the lines while comparing the runs of a respective current line with its antecedent line. Using this method the relations of the runs are determined and they can be associated to objects and marked with an object number.

Until now, runs only have been marked with an object number, but it should be noted that it is not the aim of a blob analysis to get images of the found objects. The blob analysis only has to determine the object properties. Furthermore, it is not possible inside the FPGA to keep all marked runs of an image in a memory. Therefore, a new method is developed. First, every run is allocated a specified object number, as was explained above. However, this is only applied to a respective current line and the previous line. After the respective previous line has been compared with the current line its data is discarded. Hence, the runs...
which have been compared with other runs are not available anymore. But how can the object properties being determined if the runs are discarded. The solution is the object property calculation or object feature calculation right after the comparison of a run and its allocation to an object number. If a new object is detected while comparing the runs, an object number is allocated to this run like explained before. This number is equal to a memory address in a special object feature RAM. For every run which has to be added to an object, the properties are calculated. Next, the properties are added to the properties calculated before in the object feature RAM. Thus, the object properties ‘grow’ with every new run added to the existing properties in the RAM. If an object in the RAM is not updated any more it is assumed to be completed and its properties are passed to the host PC. The procedure can be summarised as follows:

1. comparison of a run with runs of the previous line
2. association with an existing object or creation of a new object
3. allocation of an object number
4. calculation of the properties of the run and update of the properties in the object feature RAM

### 3.3. Object Merges

So far, the algorithm works for objects with simple shapes. More complex objects may have shapes which merge or divide. Two objects which are assumed to be individual when comparing early rows can merge together into one object later. An example of this is shown in Figure 4. While performing a blob analysis these merges cause problems. This is a reason why several algorithms need to proceed through the image a second time. These algorithms ‘clean up’ the object associations by proceeding through the image in inverted direction [9].

The innovative algorithm presented here solves this problem in a novel way. As the properties are calculated while detecting the objects, the problems of merges can be solved without a second pass through the image.

In Figure 4 the current run is compared with the run, marked in the previous line. As they overlap, they must belong to the same object. However, they are labelled with different object numbers, because it was assumed that they belong to different objects while scanning the previous lines. To solve this, all properties of object “two” which have been calculated so far, are updated into the properties of object “one”. To avoid further updates of the obsolete object “two”, a pointer in the object feature RAM at address “two” is set which redirects all further updates to object “one”. The flow chart in Figure 5 summarises the object detection algorithm.

This example shows that all object properties can be determined correctly. A second pass is therefore not necessary. Hence, the algorithm can be performed in only a single pass, without the need of complex storage operations.

### 3.4. Object Completion

In the previous section the ‘growing’ of object properties in the object feature RAM has been described. Now the completed objects have to be passed to a host PC. This has to be done immediately after the object properties in the RAM are completed, even if the object detection scan through the image is still in progress and other objects may still need to be detected. So the latency and the required memory size of the object feature RAM are reduced because memory addresses and object labels can be reused for other objects.

A garbage collection mechanism searches in the object feature RAM for completed objects and obsolete objects. An obsolete object is an object which points to another object after a merge. An object is completed if it has not been updated during the last 2 lines. The garbage collector passes these completed objects to a host PC and makes the memory addresses in the RAM available for new objects.
FPGA Implementation of a Single Pass Real-Time Blob Analysis Using Run Length Encoding

MPC-Workshop February 2008

3.5. Feature Calculation

The calculation of object properties or object features can be divided into two parts. On one side, the features calculated so far and stored in the object feature RAM have to be updated by new runs which are added. On the other side, the two object features caused by an object merge have to be added. Here the features area, bounding box and centre of gravity are explained. The area is the sum of all pixels of an object. A recurrence relation which describes the update of a run to the area calculated so far can be described by

\[ A^* = A + (e - s + 1) \]  

(1)

where \( e \) is the end-position of a run and \( s \) the start-position. Thus, \( e - s + 1 \) is the length of a run. The centre of gravity is determined for the \( x \)-axis and \( y \)-axis.

\[ x = \frac{1}{A} \sum_{(u,v) \in R} u \quad y = \frac{1}{A} \sum_{(u,v) \in R} v \]  

(2)

where \( A \) is the area. \( u \) and \( v \) are coordinates of the object \( R \). The recurrence relation will then be

\[ x^* = x + \frac{e^2 + e - s^2 - s}{2} \quad y^* = y + (e - s + 1) \cdot l \]  

(3)

where \( l \) is the line number of the run. \( x \) and \( y \) have to be divided by \( A \) afterwards to complete the calculation. The bounding box is the minimum paraxial rectangle which includes all object pixels. It can be easily determined by comparisons. More features are defined in [8].

4. Hardware Used

The implementation of the algorithm is performed on a XILINX Spartan FPGA [7] which is embedded to a Silicon Software frame grabber system. This frame grabber is a PCI-express or PCI64-bit PC extension card [6]. The card has up to four external camera link interfaces for the connection between the camera and the frame grabber. After performing the blob analysis, the results are transferred over the PCI-bus into the main memory of the host PC. The complete configuration of the frame grabber is shown in Figure 6.

The presented algorithm fits into a low-cost FPGA which allows an economic implementation of the blob analysis. Embedded multipliers are used to calculate complex object features such as the centre of gravity and image moments. True dual-port FPGA block RAM is used for memory access and FIFOs.

5. Performance

The performance of the implementation in terms of speed depends on the image content. This is because the run length encoding and the number and size of objects in the image. As explained in Section 3.2 the input pixels are compressed by a run length code to process on average multiple pixels in one clock cycle. Because of block RAM accesses and feature calculation processes the implementation requires on
average 6 clock cycles to process one run. If the compression of the parallel input pixels has a rate of less than 1:6 the object detection will inhibit the input and therefore, cannot perform the blob analysis in real-time. However, extensive tests have shown that only for images with an artificial strong noise this gap cannot be compensated for. Furthermore, the compression results in the fact that the algorithm is independent on the input image resolution. Thus, the content and the number of objects is the determining factor for the required clock cycles.

The performance of the blob analysis is limited to the maximum input speed of the transmission of the raw data between the camera link interface and the run length encoder. An image with a resolution of 8k pixels i.e. an image with a resolution of 8.192 by 8.192 pixels has 67MPixels. At a clock frequency of 58MHz and the parallel transmission of 32 pixels a frame rate of

\[
\frac{32 \cdot 58 MHz}{67 MPixels} \approx 28 \text{ fps}
\]

(4)

can be achieved. If images with a resolution of 4k are used a frame rate of

\[
\frac{32 \cdot 58 MHz}{17 MPixels} \approx 109 \text{ fps}
\]

(5)

can be achieved. As explained, only for images with an added strong noise the detection mechanism will inhibit the input. Therefore, the bottleneck is the transmission of the incoming data and the camera link interface and not the blob analysis algorithm itself.

An example of a simulation is shown in Figure 7. The 1MPixels sized input image consists of three main objects. A very strong noise is added to simulate a near worst case scenario. A cut-out of the image after the binarisation is shown in the figure. The algorithm calculates the area, the bounding box, the centre of gravity as well as the orientation of the objects. The blob analysis detects a total of 60.069 objects in the image, due to the high level of added noise. Only object properties with an area greater than 100 pixels are shown in the Figure. The bounding boxes show that the real objects are then detected correctly. The algorithm requires 37,7023 clock cycles for the blob analysis. Therefore, the maximum frame rate will be still 132fps. This shows the ability of the algorithm to handle strong image noise while maintaining the high frame rates and still operating correctly.

Figure 7: Simulation output of an analysed image

After having presented the image detection performance, the resources are now discussed. They are dependent on the image width, the maximum number of runs and the number of features which have to be calculated. For a normal configuration with input images of a resolution of 4k by 4k pixels the implementation requires 60kBits of block RAM and about 5.678 look up tables (LUT). This is including the blob analysis itself, the DRAM-, the camera link- and the PCI interface. The blob analysis itself only requires about 1.600 LUTs. The XILINX Spartan II XC2S600E device has a total of 15.552 LUTs and 288k block RAM bits. The implementation can be placed and routed up to a frequency of 66MHz in the Spartan II and Spartan III devices.

6. Conclusions

This paper has presented the development of a real-time blob analysis algorithm for a FPGA implementation. The incoming frames from a digital video camera are transferred to the FPGA via the camera link interface. After performing the blob analysis in the FPGA, the determined object features are passed to a host PC. By use of a run length encoding the data rates of the parallel incoming pixels can be met.

The memory inside a FPGA is not sufficient to hold a whole frame. Therefore, the image is processed in a raster scan procedure where only two antecedent rows are stored at any given time. By comparing these two rows, the objects and their properties are determined on-the-fly. This method minimises the amount of memory necessary in the FPGA. Object properties are passed to the host PC as soon as they
are completed while scanning the picture, overcoming a possible communications bottleneck and saving FPGA memory. The problem of merging objects is solved by combining their properties and setting pointers on objects which are out-of-date. Hence, a second pass through the image is not necessary and the detection of all objects is performed in only a single pass. Obsolete object labels can be reused by the algorithm which allows a memory efficient implementation.

The consistent use of dual port block RAM in the FPGA together with the algorithm focused on performance allows the real-time analysis. Furthermore, the algorithm fits in a low-cost XILINX Spartan II/E device requiring a block RAM size of about 60kBits and 5.678 LUTs. Using the run length compression, the implementation can process the data rates of a full configuration camera link interface which is 680MByte/s. Resolutions of up to 8k by 8k pixels and frame rates of up to 400fps are possible.

This blob analysis is implemented into a Silicon Software microEnable III or IV frame grabber system which allows the combination with numerous existing pre-processing operators. This makes the blob analysis a powerful tool for machine vision applications.

7. Acknowledgement

Thanks to Silicon SOFTWARE GmbH [6] which supports the research as well as provides the frame grabber hardware and software environment.

8. References