# SoC Processor for Real-Time Object Labeling in Life Camera Streams with Low Line Level Latency

Zhengqiang Yu[*], Luc Claesen[+], Yun Pan[$], Andy Motten[+], Yimu Wang[*], Xiaolang Yan[*]
[+]Faculty of Engineering Technology, Hasselt University, Diepenbeek, Belgium
[*]Institute of VLSI Design, Zhejiang University, Hangzhou, China
[$]Department of Information Science & Electronic Engineering, Zhejiang University, Hangzhou, China

***Abstract*—Image recognition systems implement a number of processing stages: preprocessing, segmentation and classification. In camera based video processing chains, usually several frame delays are incurred between the moment of capture and the actual availability of the classification results. Hardware architectures for stream based video processing have already been widely employed. In this paper, a new hardware architecture for accelerating the generic task of connected component analysis and object labeling in the segmentation step is presented. The architecture is specifically optimized for very low latency between image component capture by a camera and the detection in hardware. This latency constitutes only a few delay lines, thereby shortening the response time by a few orders of magnitude in comparison to traditional frame-buffer based methods.**

***Keywords—Connected Component Analysis; Video Processing; Vision; Blob Detection; Object Labeling; robotics***

## I. INTRODUCTION

The VLSI technological evolution has resulted in the availability of high resolution, high performance image sensors. Many consumer products such as smart phones, tablets and game consoles already include several cameras per device. Image recognition and computer vision methods constantly find new applications: security systems, traffic monitoring, intelligent cars, production lines, robotic systems, human computer interfaces, augmented reality etc... Intelligent vision systems operate in a number of processing stages such as: 1) image preprocessing, 2) segmentation and 3) object classification / recognition.

The image preprocessing step [1] conditions the captured images on a frame-by-frame base to emphasize the visibility and detectability of the objects or features of interest. In the segmentation stage a number of regions corresponding to possible objects of interest are identified in the image, starting from a binary map of the image that indicates candidate pixels of segmented objects. Fig. 1a shows a sample binary map resulting from the preprocessing.

The connected regions in the binary map represent regions of individual objects that need further processing, eventually using masking operations with the original image. A task in the segmentation is the identification of these regions by means of a connected component analysis, also called "object labeling". Fig. 1b illustrates the result of object labeling on the binary map of Fig. 1a. Here all of the connected regions are assigned with a unique label.

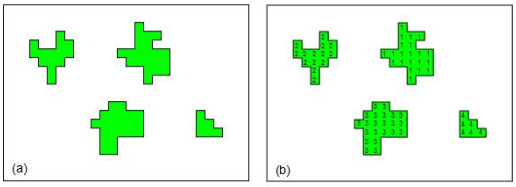


Fig. 1. (a) binary image (b) object labeling

Some connected component analysis algorithms [2] use two passes through the image data. They require to store entire images in memory. The temporary storage of image data not only adds to the cost of a hardware system and influences the scarce memory bandwidth, but impacts the latency of the object detection. Therefore the direct labeling in the scan line order of the camera data is very important. In [3,4] a one-pass hardware architecture for object labeling is proposed. Per scan-line, the eventual labeling of an incoming pixel is determined by the previous labels of the surrounding upper and left pixels. With "U" shaped features this can result in wrong labels. The method in [3,4] introduces the exploitation of the ordering properties of the label assignment. In case of conflicts, a backtracking method can be used during the blanking period for assigning the correct labels after the end of the scan line. Depending on the available blanking period, the amount of label corrections per line can be limited. In [5,6] an alternative single scan hardware architecture is proposed. In a preprocessing step, the streaming binary map camera information is converted to a run-length encoding (RLE) per line. This approach has the advantage that the consecutive object labeling has more clock cycles available per line. Per line, the storage for potential object labels is cleaned up in a garbage collection step. This makes this method interesting for line-based camera's e.g. used for object detection in production lines. In contrast to [3,4] this method does not have a fixed strategy for label assignment in case of conflicts during merging.

In this paper, a new object labeling architecture is proposed which combines the advantages of the one pass architectures as presented in [3,4] (the use of an ordering strategy for the label assignment in case of conflicts and the trace-back based resolution of it) and the efficient RLE based processing in [5,6]. The SoC architecture has been designed in a parameterized way, making it easy for integration in various automatic vision applications. The operation is demonstrated by the FPGA implementation connected to a digital streaming

video camera and the real-time display of the labeled components.

## II. OBJECT LABELING ARCHITECTURE

The overall architecture for the component labeling architecture is illustrated in Fig. 2. It consists of six major blocks.

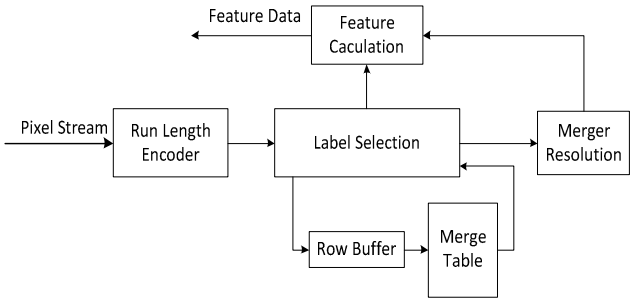


Fig. 2. Architecture of the algorithm

The "Run Length Encoder" compresses the pixels by means of a run length code. The "Label Selection" assigns labels for the current run. When two runs are merged, it also writes the merge links to the "Merger Resolution" module. The "Merger Resolution" module updates the "Merge Table" with the merge links after the label assignment of the current row. The "Row Buffer" is a FIFO which is used to cache the labeled runs of the previous row. The "Feature Calculation" module updates the object features or the object properties.

### A. Run Length Encoder

Hardware implementations of connected component labeling algorithms process the images pixel per pixel [3,4,7,8]. Because images are raster scanned, it is not easy to process the pixels in parallel. In run length encoding, a sequence of non-zero pixels in a line (also called a "run") is described by means of a start position value, an end position value and a row number.

Run length encoding allows the transmission of the information of multiple pixels at once. The algorithm as well as the camera can operate at different clock frequencies. If the average amount of pixels in a run is larger than the amount of clock cycles needed to process one run, the frequency of the algorithm clock can be less than the camera pixel clock.

### B. Label Selection

The "Label Selection" module assigns a label for the current run. This comes down to the comparison between two runs: one in the previous scan line and another in the current scan line. Because of RLE, the situation is different in comparison with previous methods [3,4,7,8]. First, the current run in the current row has no relationship with the previous run in the current row. The key factor is the relative relationship between the current run and a corresponding run in previous row. Therefore a RLE row buffer is needed to cache the runs processed in the previous line. Second, the label to be assigned to the current run is not only determined by the current comparison result but also by the previous result.

The relative relationships of two runs on the current line and on the previous line can be classified into five categories (shown in fig. 3). The relative relationships can be determined by comparing the start position value and end position value of the two runs. $y_{pre}$ is the end position value of the run in the previous row. $y_{cur}$ is the end position value of current run.

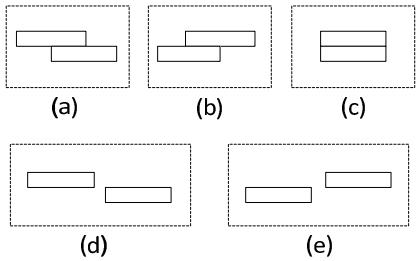


Fig. 3. Relative relationships of runs: (a) connected and $y_{pre} < y_{cur}$, (b) connected and $y_{pre} > y_{cur}$, (c): connected and $y_{pre} = y_{cur}$, (d): not connected and $y_{pre} < y_{cur}$, (e): not connected and $y_{pre} > y_{cur}$.

All complex relationships of connections can be described by the combination of the five basic relationships. We classify the connected runs by their end position values, because it determines which run should be updated after the comparison.

As mentioned above, the label of the current run is not only determined by the current comparison result but also by the previous one. Therefore after the current comparison, the algorithm has to decide the next actions to be taken according to both the current and the previous result, as shown in table I.

TABLE I. NEXT ACTION BASED ON BOTH THE CURRENT AND THE PREVIOUS RESULT

| Pre-result | Cur-result | Actions |
|---|---|---|
| a | a | Merge runs, assign smaller label to current run, push the merge link to "Merger Resolution", update pre-row run, |
| others | a | Assign label of pre-row run to current run, update pre-row run |
| a | b | Merge runs, assign smaller label to current run, push merge link to "Merger Resolution", update current run |
| others | b | Assign label of pre-row run to current run, update current run |
| a | c | Merge runs, assign smaller label to current run, push merge link to "Merger Resolution", update both pre-row and current run |
| others | c | Assign label of pre-row run to current run, update both pre-row and current run |
| all | d | Update pre-row run |
| c,d,e | e | Assign a new label to current run，push merge link to "Merger Resolution", update current run |
| a,b | e | Update current run |

### C. Merger Resolution

When a merge of two runs occurs, the merge link is pushed into a stack. The merge information is captured in a memory consisting of merge links. Merge link is a combination of the labels of the two runs. After the label assignment of the current

row (during the horizontal blanking period), each merge link is read in the opposite sequence in which the stack has been constructed. If the current merge link is the top data (the last), the smaller label is written to the merge table. If it is not the top data, the smaller label is compared with the previous merge link. If not equal, the smaller label is cached as the minimum label and written to the merge table at the same time. If the smaller label is equal to the labels of the previously merge link, the minimum label is written to the merge table. The merge table is updated at the address of the larger label.

### D. Feature Calculation

During the object labeling, the algorithm can also collect and calculate features of different objects, such as area, height, perimeter, bounding box, center of gravity, moments of inertia etc... The features are stored in a feature RAM. The feature RAM is indexed by the object label number. The input of the "Feature Calculation" module is divided into two parts. One part is the processed run from the "Label Selection" module. The other part is the merger information from the "Merger Resolution". The feature of the region with the larger label number is merged with the feature of the region with the smaller label and then discarded. A one bit memory with the same depth as the feature RAM indicates if the corresponding feature is valid or not. Because the "Label Selection" and the "Merger Resolution" operate during different time periods, there is no conflict between the two tasks.

## III. SoC Implementation Aspects

Whereas traditional video pre-processing algorithms such as de-Bayering, filtering, kernel based convolution operations, edge detection etc. [1] can be performed on a pixel-synchronous video stream, the algorithm presented in this paper consists of a number of connected finite state machines, performing specific tasks in the algorithm. Intermediate results of the algorithm are stored in dedicated memories. In fact they are "line-synchronous" with the video stream.

### A. Memory Utilization

When implementing algorithms on SoC's, the resource utilization is very important. In the hardware implementation, five memories are needed. They are discussed in what follows. We assume an input image resolution of M × N, where M is the amount of pixels in a row, and N the amount of rows. The width of a run is $2 * \lfloor \log_2 M \rfloor + \lfloor \log_2 N \rfloor$.

*1)* FIFO in "Run Length Encoder": The depth of this FIFO is related to the ratio of the system clock frequency $f_{sys}$ to the camera pixel clock frequency $f_{cam}$, the clock cycles needed to finish a comparison of two runs and the specific merge configuration at a specific line. The worst case for the depth of the FIFO is illustrated in fig. 4a. The maximum size is $\left\lceil (2 * \lfloor log_2 M \rfloor + \lfloor log_2 N \rfloor) * \frac{M}{2} * \frac{f_{cam}}{f_{sys}} * \frac{cycle}{2} / (\frac{f_{cam}}{f_{sys}} * \frac{cycle}{2} + 1) \right\rceil$.

*2) Merge table*: The labels which do not appear in the current row are released so that they can be assigned to new objects. The released labels include the obsolete labels after merging and the labels of the processed objects. The required memory size of the merge table is reduced. It just has to cache the merged labels of one row. The maximum amount of runs in a row is $\lfloor M/2 \rfloor$(shown in fig. 4b). The width of a label is $\left\lfloor log_2 \frac{M}{2} \right\rfloor$. Therefor the maximum size of the merge table is $\lfloor M/2 \rfloor * \left\lfloor log_2 \frac{M}{2} \right\rfloor$.

*3) Row buffer:* The depth of the row buffer is related to the amount of runs to be stored in one line. So the maximum depth of Row buffer is also $\lfloor M/2 \rfloor$ (shown in fig. 4b). The processed run includes the information of the run and the label. So the width of the processed run is $2 * \lfloor log_2 M \rfloor + \lfloor log_2 N \rfloor + \left\lfloor log_2 \frac{M}{2} \right\rfloor$. The maximum size of the row buffer is $\left(2 * \lfloor log_2 M \rfloor + \lfloor log_2 N \rfloor + \left\lfloor log_2 \frac{M}{2} \right\rfloor\right) * \lfloor M/2 \rfloor$.

*4) The stack in "Merger Resolution"*: In the worst case, every run in the "Row Buffer" is merged, as occurs in the middle row of the situation in fig. 4c. So the maximum size of the stack is $2 * \left\lfloor log_2 \frac{M}{2} \right\rfloor * (\left\lfloor \frac{M}{2} \right\rfloor - 1)$.

*5) RAM in "Feature Calculation":* The maximum amount of objects in a row is $\lfloor M/2 \rfloor$, the same as the amount of runs. The width of the RAM is unknown and varies in different situations.

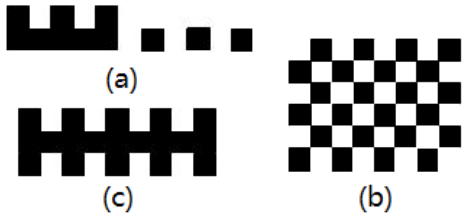


Fig. 4. Worst case for memory utilization :(a) FIFO in "Run Length Encoder", (b) row buffer and merge table, (c) the stack in "Merger Resolution".

The worst case memory utilization of various algorithms for a 1024*1024 image is shown in table II. Here the system clock frequency $f_{sys}$ is set equal to the camera pixel clock frequency $f_{cam}$. The RAM in "Feature Calculation" is 40 bits wide, which is the same to the width of data table in [4].

TABLE II. WORST CASE MEMORY (BITS) UTILIZATION FOR A 1024*1024 IMAGE

| | Classic | Single pass | Algorithm in [4] | This paper |
|---|---|---|---|---|
| FIFO | | | | 256*30 |
| Row buffer | 18874368 | 18432 | 9216 | 512*39 |
| Stack | | 18432 | 4608 | 511*18 |
| Merger table | 4718592 | 4718592 | 9216 | 512*9 |
| RAM | | 10485760 | 40960 | 512*40 |
| Total | 23592960 | 15241216 | 63000 | 61934 |

The worst case requirement as described above is not always realistic or necessary in specific applications. The actual requirements can be lower if the amount of expected objects is limited. The input pixels are filtered and the extreme pixels are treated as noise and discarded.

### B. Performance

The algorithm compresses the input pixels by a run length encoding. At the same clock frequency, it is faster than algorithms that process images on a pixel per pixel basis [3,4]. The key part that determines the performance is the "Label

Selection". To achieve a better performance, the read clock of the RAM and the clock of the "Label Selection" are shifted by 180 degrees. So if the "Label Selection" gives the read enable and the address, the runs will be updated during the next clock cycle. The run resulting from the row buffer has to be looked up in the merge table to correct the label. This action needs one more clock cycle. In [5], the run length encoding algorithm needs 6 clock cycles to process one comparison. In our approach, processing one comparison requires only 2 clock cycles. But it does not mean that it finalizes the computation in one run. If a merger occurs, more comparisons are needed to complete the assignment.

## IV. FPGA IMPLEMENTATION

To demonstrate the operation of the hardware architecture presented in this paper, a sample application using the Altera DE2-70 board [9] and the TRDB-D5M camera [10] is illustrated here. The camera is used with a resolution of $1280 \times 1024$ and a pixel clock frequency of 60MHz. In this application, objects in the input image stream are boxed and displayed on a VGA screen in real time. The global structure of the application is shown in fig. 5. Lights emitted by 940nm infrared LEDs are used as objects to be segmented and labeled. The VGA_controller combines the preprocessed image from SDRAM and the features from connected components module. The output of the application is shown in fig. 6.

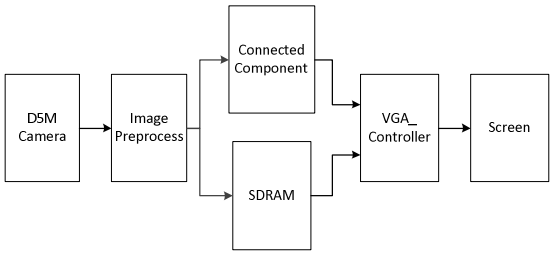


Fig. 5. Architecture of the application

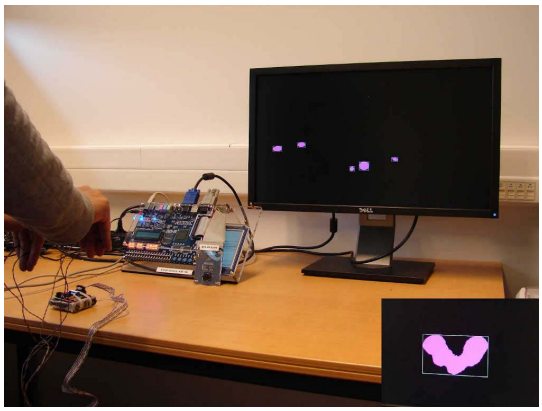

Fig. 6. Application demonstration: five 940nm infrared LEDs that are pointed by hand from the left side towards an infrared camera attached to the FPGA board. The screen displays the binary image in real-time as well as the boxes surrounding the detected connected components in the binarized camera stream. The inset in the lower right corner shows the connected component detection and corresponding boxing of the LEDs when held in a "U" shape.

## V. CONCLUSION

The classic two-pass connected components algorithm wastes a lot of hardware resources when implemented in hardware as such. It requires a frame buffer memory to cache images and in addition results in a latency of one or more frame periods. In this paper, a single pass, real time connected components algorithm for hardware implementation, using run length encoding, has been presented. The architecture has been implemented and demonstrated on an FPGA. The processing of one comparison of runs only requires 2 clock cycles. Object features are updated during the label selection and the merger resolution and are sent out as soon as they are completed. This means that instead of latencies of a number of frame periods (several msec) the latency is reduced by a few orders of magnitude, depending on the preprocessing operations and the camera shutter times, to a number of line periods (several μsec).

A sample application has been implemented on an educational FPGA board. This implementation demonstrates the algorithm and its hardware implementation while processing digital camera data in real time.

## ACKNOWLEDGMENT

The research in this paper was sponsored in part by the Chinese MOST (Ministry of Science and Technology) and the Belgian FWO (Flemish Research Council) bilateral cooperation project number G.0524.13.

## REFERENCES

[1] R.C. Gonzales, R.E. Woods, "Digital Image Processing", Addison-Wesley Pub, 3rd Edition, Aug. 2007, ISBN 978-0131687288

[2] R.V. Rachakonda, "High-Speed Region Detection and Labeling using an FPGA-based Custom Computing Platform", Proc. of the 5th International Workshop on Field-Programmable Logic and Applications, FPL'95, pp. 86-93.

[3] D.G. Bailey, C.T. Johnston, "Single Pass Connected Component Analysis", Proc. of Image and Vision Computing New Zealand 2007, Hamilton, New Zealand, December 2007, pp. 282-287.

[4] D.G. Bailey, C.T. Johnston and Ni Ma, "Connected components analysis of streamed images", Field Programmable Logic and Applications, Heidelberg ,679-682(2008).

[5] J. Trein, A. Th. Schwarzbacher and B. Hoppe, "FPGA Implementation of a Single Pass Real-Time Blob Analysis Using Run Length Encoding", In MPC-Workshop, Ravensburg-Weingarten, Germany, February 2008.

[6] J. Trein, A. Th. Schwarzbacher, B. Hoppe, K.-H. Noffz and T. Trenschel, "The FPGA implementation and investigation of a real-time blob analysis algorithm", Proc. Irish Systems and Signals Conference, Derry, N. Ireland, September 2007, pp. 121-126.

[7] C. Grana, D. Borghesani, P. Santinelli and R. Cucchiara, "High Performance Connected Components Labeling on FPGA", in Workshops on Database and Expert Systems Applications, 221-225(2010).

[8] M. Jablonski and M. Gorgon, "Handel-C implementation of classical component labelling algorithm", in Euromicro Symposium on Digital System Design(DSD 2004), Rennes, France, 387-393 (2004).

[9] http://www.terasic.com.tw/cgibin/page/archive.pl?Language=English&CategoryNo=53&No=30&PartNo=1

[10] http://www.terasic.com.tw/cgibin/page/archive.pl?Language=English&CategoryNo=68&No=281&PartNo=1