Comparative Study on Connected Component Labeling Algorithms for Embedded Video Processing Systems

R. Walczyk[1], **A. Armitage**[2], **and T.D. Binnie**[1]
[1]School of Engineering and the Built Environment, Edinburgh Napier University, Edinburgh, UK
[2]School of Computing, Edinburgh Napier University, Edinburgh, UK

Abstract— *The objective of this paper is to carry out a detailed analysis of the most popular Connected Component Labeling (CCL) algorithms for binary images. This study investigates their usability for processing streaming data and suitability for implementation using Field-Programmable Gate Array (FPGA) devices. The first part of this paper reviews popular CCL algorithms. Both capability for real-time video processing as well as memory requirements are taken into consideration. The second part of the paper describes an efficient implementation of the single pass labeling algorithm using a Virtex-II Pro FPGA. It is verified on the development board with an infrared camera module as a real-time video source. The system is capable of processing video stream with 640 $\times$ 480 pixels per frame at a speed of 30 fps limited by the bandwidth of the video source.*

Keywords: Connected Component Labeling, FPGA, Embedded Systems

1. Introduction

Connected Component Labeling (CCL) is a fundamental feature of many computer vision systems. It allows the assignment of unique identifiers (labels) to different, disjoint connected components. Hence it constitutes a significant stage in automated surveillance or pattern recognition systems.

Automated surveillance systems are very complex designs [1]. A typical real-time video processing embedded system contains the following stages: firstly, the video source is processed by one of the background separating algorithms. In this step the foreground objects (Regions of Interest) are subtracted from the background model. Results of this operation are forwarded to the filtering and thresholding unit where video acquisition noise is removed. By applying a thresholding filter to the processed data, the image is binarized, hence the amount of information is significantly reduced. From now on these groups of pixels refer to the detected objects, however to allow for tracking or classification, this data has to be further processed. CCL algorithms analyse binary images in order to distinguish disjoint groups of pixels (objects) and assign them with individual labels. Labeled objects are further processed to calculate their features used by tracking algorithms, such as position, width, height or centre of gravity.

Over the years, a wide variety of different connected component labeling algorithms have been proposed. They can be grouped as follows: algorithms processing an image frame in two consecutive passes through the image; multiple scan algorithms, where the number of passes depends on the image complexity; parallel algorithms processing a number of pixels at a time; contour tracing techniques following the contour of objects and single pass algorithms processing data sequentially in one scan through the image. This paper gives a brief description of these groups together with a detailed analysis of their suitability for video processing systems. The major bottleneck for this type of application is the memory requirement imposed by the resolution of the video source together with the real-time processing speed, hence detailed calculations will be provided.

The outline of this paper is as follows: Section 2 introduces the reader to the field of CCL; the most common algorithms will be presented. Section 3 analyses the memory requirements and execution time for these algorithms. The remainder of the paper gives implementation details using FPGA development platform of the single pass algorithm which proved to be the most suitable for embedded video processing systems. It also gives description of how to improve data administration in order to reduce memory requirement.

2. Connected Component Labeling Algorithms

Connected component labeling is an operation where groups of connected pixels (connected component) are classified as disjoint objects with unique identifiers (labels). This operation can be described as assigning a unique label l taken from a set of integral values $L \subset \mathbb{N}$, to each connected component. Thereby an input binary image frame $B \in \mathbb{Z}^2$, where all the pixels $p \in B$ correspond to the background or to the foreground objects ($F_b = 0$ or $F_f = 1$ respectively), is transformed into a frame where each pixel is represented by a decimal value (label) which is the identifier of the connected component CC_k it belongs to. Here $1 \leq k \leq K$ and K defines the total number of connected components within the frame. Labeling of B can be written as $g : B \mapsto \mathbb{N}$, where $g(x, y)$ is described as:

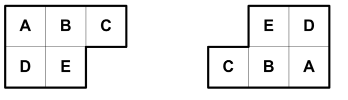


Fig. 1: 8-connected neighbourhood scan mask. (a) Forward scan mask M_f. (b) Backward scan mask M_b.

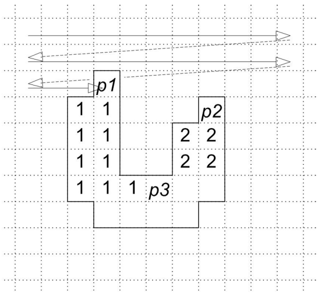


Fig. 2: A typical label collision

$$g(x,y) = \begin{cases} F_b & \text{if } B(x,y) = F_b, \\ l_k & \text{if } p(x,y) \in CC_k. \end{cases} \tag{1}$$

Due to the raster scanning nature of the image in digital systems, in order to label all the pixels within the cluster, most of the algorithms employ a scan mask M_f which is shifted pixel by pixel according to the present location (window filter manner). The scan mask, depicted in Figure 1(a), is used to check if there are any other pixels in the neighbourhood that need to be, or are already labeled. If there are no pixels in the neighbourhood of the pixel E and $E = F_b$, then the current value at location (x, y) stays unchanged. If the pixel $E = F_f$ and all the other neighbouring pixels are F_b, a new label l_k will be assigned. However, if one of the adjacent pixels was already labeled, copy this label. When there are more than one pixels labeled within the neighbourhood of the pixel E, these labels need to be merged.

The most common problem that the majority of algorithms struggle with is the *'u'* shaped cluster of pixels, depicted in Figure 2. Due to the raster scan, there is no momentary information that pixels $p1$ and $p2$ belong to one object. Once the pixel $p3$ is encountered, labels from positions C and D within the scan mask require to be merged. The major problem caused by the merging step is that all the previously scanned pixels need to be relabeled with one unique label per object. This requires at least one more scan through the image, also an auxiliary memory has to be used to store all the label ambiguities. Over the years, a wide variety of different techniques have been proposed in order to deal with this problem. Further subsections give general description of the most common labeling algorithms.

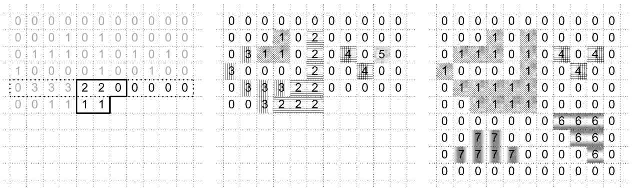


Fig. 3: Equivalence table based labeling. (a) Binary image input data, the line buffer marked by dotted rectangular box stores provisional labels from the previous line scan. (b) Provisional labels assigned simultaneously with input data. (c) Labeled image frame after the second scan.

2.1 Two Pass (Classical) Algorithm

One of the first publications describing this CCL algorithm was by Rosenfeld and Pflatz [2]. Over the years their algorithm has been significantly improved. The two pass algorithm, very often referred to in the literature as the classical algorithm, is used as a reference point in many benchmark tests. The key feature of this algorithm is the constant number of passes (two) through the binary image. Most of the two pass algorithms share similar features however they differ in data administration. The general concept is to assign preliminary labels while new foreground pixels F_f are appointed during the initial scan, see Figures 3(a) and 3(b). Once label ambiguity is encountered, the lower label is assigned and the equivalence table (ET) is updated as can be seen in Table 1(a). At the end of the image scan the equivalence table needs to be sorted, as depicted in Table 1(b). During the second scan all the preliminary labels are overwritten with their equivalences resulting in Figure 3(c). The initial scan can be described as:

$$g(x,y) = \begin{cases} F_b & \text{if } B(x,y) = F_b, \\ l_{k+1} & \text{if } \forall\{i,j \in M_s\} g(x-i, y-j) = F_b, \\ g_{min} \text{ \& ET} & \text{otherwise,} \end{cases} \tag{2a}$$

and

$$g_{min} = \min[\{g(x-i, y-j) | i,j \in M_s\}], \tag{2b}$$

where l_{k+1} indicates an increment of the label l_k, $M_s = M_f$ but the pixel $p(x,y) = F_f$, ET stands for equivalence table update and $\min(\cdot)$ denotes an operator calculating the minimum value.

The major drawback of this algorithm is the memory consumption of the output labeled image - the number of labels used per image during the initial scan is highly dependent on the image complexity. An example hardware implementation of this algorithm was described in [3].

Table 1: Equivalence table

(a) During the initial scan

prov. label	eq. label
1	1
2	3
3	1
4	4
5	4
6	-
7	-
8	-

(b) After the second scan

prov. label	eq. label
1	1
2	1
3	1
4	4
5	4
6	6
7	7
8	7

2.2 Multiple Scan Algorithm

In 1981 Haralick introduced an iterative algorithm which does not require any auxiliary storage for label equivalences [4]. This technique involves multiple forward and backward raster scan passes through the image until no label change occurs. All the label collisions are solved on the local neighbourhood basis according to the equation (2a), however ET does not apply here. After the first scan through the binary image B, all the pixels will be assigned with the preliminary labels similarly as in the classical algorithm, however all the label ambiguities will be resolved on the local neighbourhood basis during the following multiple forward and backward scans with alternating scan mask $M_s = M_f$ and $M_s = M_b$ respectively according to:

$$g(x,y) = \begin{cases} F_b & \text{if } g(x,y) = F_b, \\ g_{min} & \text{otherwise.} \end{cases} \tag{3}$$

This algorithm was designed for systems with limited memory resources processing low resolution images. The performance of this algorithm is related to the size and the complexity of the binary image, thus it is not recommended for higher resolution images. Recent implementations improved processing time by introducing local equivalence tables, however the number of scans through the image frame is still dependent on the image complexity and is hard to predict. Hence these algorithms are not suitable for real-time video processing and will not be taken into consideration in further discussion. Two separate hardware implementations were described in [5], [6].

2.3 Parallel Processing Algorithm

Algorithms from this group are highly specified for parallel processing platforms and are not suitable for ordinary computer architectures. They often require one logical processing element per pixel. These algorithms, although suitable for FPGA implementation, require a large amount of logic resources. Due to their complexity, the size of input image has to be limited. More recent implementations proved to be much less resource consuming, however they are still too large for ordinary architectures. An FPGA-based processing platform proposed by Mozef *et al.* [7] is able to process an image frame while the image is loaded however it employs four XC4025 FPGA chips (32 × 32 CLBs each) and the image size is limited up to 32 × 32 size only. Due to the parallel processing nature, these algorithms are not efficient for streaming data video signals.

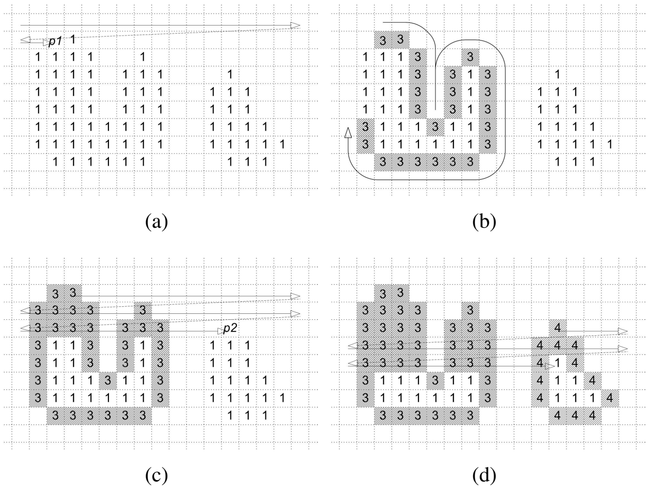


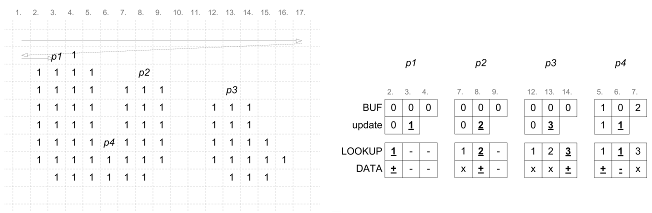
Fig. 4: Contour tracing based labeling. (a) Start tracing the contour. (b) Label contour pixels until pixel $p1$ is encountered again. (c) Keep scanning the image until unlabeled foreground pixel $p2$ is reached. (d) Label internal pixels with contour pixels labels.

2.4 Contour Tracing Algorithm

A new variation of CCL was introduced by Chang and Chen [8], [9]. It is based on the contour tracing technique to detect contours of the object and also to fill in interior areas. A single pass through the binary image is sufficient to label all the objects. It was proved that this algorithm gives better performance than algorithms based on equivalence tables and requires less memory. With contour tracing, the label collision problem does not apply hence there is no need to scan the image multiple times, however it requires random access to all the image pixels.

In order to label a binary image frame using the contour tracing algorithm, an input image has to be stored in the memory. The image is raster scanned until a foreground pixel $p(x,y) = F_f$ is encountered. The complete trace of the contour is performed until the same pixel is reached again. The contour is labeled with index l_k for $L \subset \mathbb{N}$, where $3 \leq k \leq K$ and K defines the total number of connected components within the frame. Once the contour is labeled, l_k is incremented by 1 and the algorithm resumes scanning step. At this point, one of several pixels can be encountered:

- background pixel ($p(x,y) = F_b$)
- unlabeled foreground pixel ($p(x,y) = F_f$)
- already labeled contour pixel ($p(x,y) > 1$)
- horizontal border pixel ($x = H_{max}$)



(a) (b)

Fig. 5: Single Pass Algorithm. (a) Binary input image. (b) Memory registers, where values recently changed are underlined, "−" indicates empty cell, "x" previously assigned value, "+" stands for an update.

While background pixels are encountered, the algorithm keeps scanning the image and no further action is required. When $p(x, y) = F_f$, the algorithm starts the contour tracing procedure as described above for new object. Once the already labeled pixel $p(x, y) = l_k$ from the previously traced contour is encountered, the algorithm keeps scanning within the contour pixels while label l_k is assigned to all the pixels $p(x, y) = F_f$ until the second pixel $p(x, y) = l_k$ is reached. When the last pixel in a row is reached ($x = H_{max}$), the scan continues from the first pixel in the next row according to raster scan. The conceptual block diagram of this algorithm is depicted in Figure 4. To avoid tracing the same contour multiple times, the surrounding pixels are labeled with additional preoccupied label $l_k = 2$. Details regarding hardware implementation can be found in [10].

2.5 Single Pass Algorithm

Single pass algorithms are relatively new [11]. They were developed specifically for labeling connected component in streaming data systems [12], [13]. The labeling step is performed in a single scan while data is streamed to the system. This ensures real-time processing speed. Also there is no need for buffering the input image frame; this results in lower memory requirements. The most significant feature of this algorithm however is that it can extract most of the features of interest (position, size, etc.) for all the objects during the scanning step, so there is no need to store a labeled image frame. This significantly reduces the memory requirements. This data is kept in a separate data array. Although results of the single pass labeling are sufficient for most object counting and pattern recognition systems, this algorithm is not suitable for applications where a labeled object mask is required.

In order to label connected component in a single pass, the image has to be scanned in a raster scan. A general illustration of the algorithm operation is depicted in Figure 5. As can be seen, there are three separate memory modules in use:

- Row Buffer (`BUF`) - keeps labels assigned in the previous row;
- Lookup Table (`LOOKUP`) - gives pointers to label equivalences;
- Data Table (`DATA`) - gathers extracted features of interest.

All of these memory modules must be true dual port where both read and write operations are possible at the time of a single pixel scan. The system needs to be pipelined where data read from the `BUF` points to the label in the `LOOKUP`, which gives an address to the `DATA`. The labeling step is similar to the classical algorithm with the difference that labels are not stored in the auxiliary memory, whereas an object's features of interest are calculated and updated simultaneously with the image scan.

According to Figure 5, the pixel $p1$ has no direct neighbours, it is assigned with a new label. This label is written into the `BUF` memory (`update`), simultaneously `LOOKUP` is updated so that the label points to itself. The `DATA` will be updated with the coordinates of this location. This procedure repeats for pixels $p2$ and $p3$. However, once the pixel $p4$ is encountered, labels `1` and `2` need to be merged. The `LOOKUP` at position `2` is updated with label `1` so each time a pixel labeled with `2` is encountered, the `LOOKUP` will be pointing to the label `1` in the `DATA` memory. Simultaneously, features of interest for both labels `1` and `2` are merged, for this particular situation bottom-right corner of label `2` are copied into the `DATA(1)`; location `2` in `DATA` and it will not be used again.

3. Performance Analysis

After general overview of most common labeling algorithms, this section analyses their processing time and memory requirements. For the memory analysis, algorithms are expected to extract and store at least information about an object's position and size (top-left and bottom-right coordinates of the smallest rectangle containing the detected object).

3.1 Processing Time

Time constraints for real-time video processing systems are very strict. The processing time has to be constant for each video frame and it should not exceed frame rate of the source.

Classical Algorithm

The classical algorithm requires two scans through the image. During the first scan preliminary labels are assigned, label ambiguities are stored in the equivalence table. The table needs to be pre-processed before the second scan. For embedded systems processing video in real-time, all the label ambiguities from the equivalence table can be sorted during the horizontal or vertical blanking periods.

Contour Tracing Algorithm

The contour tracing algorithm in its original form requires only one scan through the binary image. However, to trace contours, irregular memory access patterns are required hence there is a need for an initial image scan to buffer the input frame. The algorithm introduced by Chang and Chen [8] is not suitable for hardware implementation, however it can be used within an embedded system with minor changes [10]. This variation of the algorithm requires two scans through the image with a little overhead, it meets real-time performance.

Single Pass Algorithm

The single pass algorithm was developed to process streaming data. It is capable of labeling all the objects within a binary image frame in a single image scan. This gives the best performance of all the algorithms described here.

3.2 Memory Requirements

For embedded systems processing video streams, memory requirements are of great importance. This subsection gives calculations of the amount of memory required in order to label a binary image frame with R rows and C columns.

Assuming that every connected component encounters one label collision, the total amount of required memory for three different image sizes: 320×240, 640×480 and 1024×768 with $CC_{max} = 255$ objects per image was calculated and compared with other algorithms. This can be found in Table 2; Figure 6 gives a graphical representation.

Classical Algorithm

It is difficult to estimate the exact amount of memory required by the classical algorithm due to the fact it depends on the image complexity (number of label collisions). It can be calculated according to:

$$mem_{total} = \lceil log_2(CC_{max} + CC_{col} + 1)\rceil \cdot (R \times C) + \\ + mem_{ET} + mem_{FE}, \tag{4a}$$

where

$$mem_{ET} = \lceil log_2(CC_{max} + CC_{col})\rceil \cdot (CC_{max} + CC_{col}), \tag{4b}$$

and

$$mem_{FE} = (2 \cdot \lceil log_2(R)\rceil + 2 \cdot \lceil log_2(C)\rceil) \cdot (CC_{max}), \tag{4c}$$

where CC_{max} defines the maximum number of connected components, CC_{col} number of label collisions, $\lceil\cdot\rceil$ is an operator rounding $\cdot$ to the nearest upper integer and the $+1$ comes from the fact that 0 is a preoccupied label. Equations (4b) and (4c) calculate the amount of the memory required by the equivalence table and by the extracted features of interest respectively.

Contour Tracing Algorithm

As opposed to algorithms based on the equivalence table, the amount of memory is constant for a specified number of objects. It varies in direct proportion to the image resolution and it can be calculated according to:

$$mem_{total} = \lceil log_2(CC_{max} + 3)\rceil \cdot (R \times C) + mem_{FE}, \tag{5}$$

where $+3$ comes from the fact that labels 0, 1 and 2 are already preoccupied by background, foreground and reserved pixels respectively. Comparing equations (4a) with (5), assuming that there is one label collision per connected component ($CC_{max} = CC_{col}$), algorithms based on the classical approach require one more bit per pixel which results in additional $(R \times C)$ bits of data. Moreover, memory requirement for contour tracing based algorithms can be significantly reduced. The major advantage of the contour tracing algorithms is that features of interest can be extracted during the contour tracing step. Due to this fact, only two bits per pixel are sufficient in order to separate all the objects and extract their features of interest. Memory requirement for this approach can be calculated according to:

$$mem\_2bit_{total} = 2 \cdot (R \times C) + mem_{FE}. \tag{6}$$

This approach significantly reduces the amount of the required memory. It is proportional to the image size only, so for the same frame size mem_2bit_{total} will be constant for even very complex images. For further analysis, the $2bit$ variation will be taken into consideration. According to [10], implementation of contour tracing based algorithm is more complex and causes increase in hardware complexity compared to classical algorithms.

Single Pass Algorithm

Systems based on the single pass algorithms differ in data management from other labeling algorithms. Since they operate on the streaming data, there is no need to buffer an input image. As was mentioned in Section 2.5, they require three memory modules: `BUF`, `LOOKUP` and `DATA`. The algorithm also uses small amount of operating memory for label merging and data handling however this is small enough to be ignored. The total memory requirement can be calculated as follows:

$$mem_{total} = mem_{BUF} + mem_{LOOKUP} + mem_{DATA}, \tag{7a}$$

where

$$mem_{BUF} = \lceil log_2(CC_{max} + CC_{col} + 1)\rceil \cdot C, \tag{7b}$$

$$mem_{LOOKUP} = mem_{BUF}, \tag{7c}$$

Table 2: Memory requirements for labeling algorithms with $CC_{max} = 255$ objects per image

Resolution [pixels]	Classical [K bits]	Contour [K bits]	Single Pass [K bits]
320×240	1 280	158	12
640×480	3 355	619	18
1024×768	7 668	1 578	24

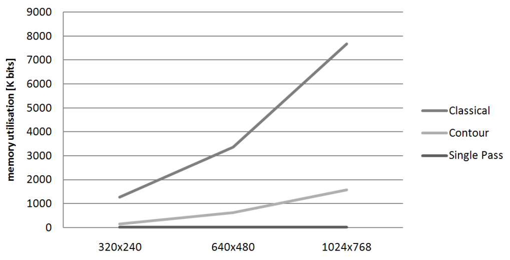


Fig. 6: Memory requirements for labeling algorithms with $CC_{max} = 255$ objects per image

and

$$mem_{DATA} = (2 \cdot \lceil log_2(R) \rceil + 2 \cdot \lceil log_2(C) \rceil) \cdot \cdot (CC_{max} + CC_{col}). \tag{7d}$$

Memory requirements for the algorithm described in [11] can be further reduced by improving data administration. This will be discussed in the following section.

3.3 Summary

The three most common algorithms for CCL were discussed. All of them meet real-time video processing criteria, however according to Table 2, there are large variations in memory requirement. The two pass (classical) algorithm requires the greatest amount of memory. Also, the memory demand grows with increasing image resolution much faster than other algorithms so will be of less interest for high resolution video signals. The contour tracing algorithm had much better performance. One of its key features is that its memory requirements do not increase with growing number of objects per image; it is proportional only to the image size. The last algorithm gives the best results. It can extract features in only one scan through the image and has much lower memory requirements. For the particular implementation with 640×480 pixels and up to 255 objects per scene, it utilizes almost 35 times less memory than the contour tracing algorithm and over 188 times less than the classical algorithm. For real-time video processing systems with limited memory resources, it is the best choice.

4. Hardware Implementation and verification

This section gives a general description of the single pass connected component algorithm together with its VHDL implementation details using XUP V2P Development Board. The real-time video source is provided by a thermal infrared camera. Results of the processing are sent to the host PC via BlueSMiRF Gold Bluetooth wireless transmission module. They are also displayed as bounding boxes on the monitor display for visual verification. A general overview block diagram of the processing system is depicted in Figure 7; a detailed description can be found in [14]. The labeling module was designed as a fully customizable generic module that can be easily included into the project. It was fully tested using embedded and external logic analysers. For development test purposes a variety of static images stored in the Block RAM memory were used.

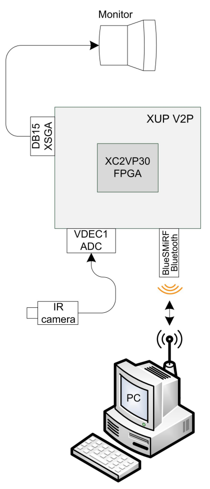


Fig. 7: A block diagram of the processing platform

4.1 Development Platform

This subsection gives an overview of the system processing platform. A XUP Virtex-II Pro Development System was used for the processing unit. This is a well equipped FPGA development board with a powerful FPGA chip and a wide range of peripherals. The Virtex-II Pro (XC2VP30) FPGA chip from Xilinx was used as a base of this board. It features 30,816 Logic Cells, 18×18-bit multiplier blocks, two PowerPC processor cores and 2,448 K bits of block RAM (136 blocks). The video source was provided by the FLIR Systems Thermacam PM595. The analog video signal was digitised by the VDEC1 Video Decoder Board with ADV7183B Video Decoder chip from Analog Devices into the ITU-R BT.656 format which is decoded and provided as a source into the labeling unit.

4.2 Algorithm Implementation

A detailed description of the single pass algorithm can be found in [11] and it will be used as a reference point in further analysis.

The major problem for all labeling algorithms are label collisions. A typical label collision was depicted in Figure 5

Table 3: Resource utilisation of the single pass algorithm implementation using XC2VP30 FPGA

Resource	in use	total	[%]
Slice Flip Flops	166	27,392	0,61
4 input LUTs	434	27,392	1,58
Occupied Slices	230	13,696	1,68
Block RAMs	4	136	2,94

and described in Section 2.5. Since the single pass algorithm operates on streaming data, once label collision is encountered, a straightforward solution would be the immediate update of the `LOOKUP`. However, in a single line more than one label collision can occur and this will not solve the problem (labels will be pointing to the wrong equivalences). To handle this task, all the consecutive label collisions are stored on the stack and are resolved during the horizontal blanking period in reverse order. This procedure ensures that the `LOOKUP` table is always up to date and gives pointers to the correct label equivalences. Once labels are merged, the label with higher index is assigned with its equivalence in the `LOOKUP`, the `DATA` is cleared at this address and this index is never used again. This approach can cause significant waste of the memory resources; the number of empty entries in the `DATA` table will grow in direct proportion to the image complexity. The proposed solution is to push the higher index after the merger into the FIFO so that it could be reused for new objects. To ensure this index will not be used in the row of the merger, it is also set as a flag. Once a new object is encountered and the FIFO is not empty, the current label index is assigned with the value read from the FIFO. However for this label the `LOOKUP` will not be updated immediately as it takes place for labels assigned from the label counter. The label and its pointer (here the same value) need to be pushed into the merger stack, the `LOOKUP` will be updated during the horizontal blanking period. This lets spare indices be used in the next line after the merger occurred.

4.3 Results

The system was developed and tested in Matlab then implemented in the FPGA. The hardware implementation was a non trivial task due to the highly pipelined architecture. The labeling unit was designed to be a self contained IP block, fully customizable by generic parameters, with clock, reset, pixel data and horizontal/vertical counter inputs. There are four vector data outputs from the module giving coordinates of the bounding box (top-left and bottom-right points) for each object once it is detected. The implementation details can be found in the Table 3. Due to the very low resource utilisation the single pass labeling algorithm can be implemented within much smaller, lower cost FPGA devices.

5. Conclusions

This paper gives a general description of the most common labeling algorithms. A detailed analysis of these algorithms was provided in order to choose the most suitable for labeling and feature extraction from real-time video streams. The single pass CCL algorithm proved to have the best capabilities. This algorithm was developed for processing streaming data, there is no need to buffer an input image frame. Additionally, labeled objects do not have to be stored in an auxiliary memory in order to extract their features of interest - they can be extracted while data is processed. This guarantees very low memory utilisation. Memory requirements for the single pass CCL can be further reduced as described above. The successful hardware implementation of the labeling unit features very low resource utilisation making it optimum choice for low cost embedded video processing platforms.

References

[1] F. Kristensen, H. Hedberg, H. Jiang, P. Nilsson, and V. Öwall, "An embedded real-time surveillance system: Implementation and evaluation," *J. Signal Process. Syst.*, vol. 52, no. 1, pp. 75–94, 2008.

[2] A. Rosenfeld and J. Pfaltz, "Sequential operations in digital picture processing," *J. ACM*, vol. 13, no. 4, pp. 471–494, 1966.

[3] M. Jablonski and M. Gorgon, "Handel-C implementation of classical component labelling algorithm," in *Digital System Design, 2004. DSD 2004. Euromicro Symposium on*, pp. 387–393.

[4] R. Haralick, "Some neighborhood operations," *Real Time/Parallel Computing Image Analysis*, pp. 11–35, 1981.

[5] D. Crookes and K. Benkrid, "FPGA implementation of image component labelling," *Reconfigurable Technology: FPGAs for Computing and Applications*, 1999.

[6] K. Appiah and A. Hunter, "A single-chip FPGA implementation of real-time adaptive background model," in *2005 IEEE International Conference on Field-Programmable Technology, 2005. Proceedings*, 2005, pp. 95–102.

[7] E. Mozef, S. Weber, J. Jaber, and G. Prieur, "Parallel architecture dedicated to image component labeling in O (n Log n): FPGA implementation," in *Proceedings of SPIE*, vol. 2784, 1996, p. 120.

[8] F. Chang and J. Chen, "C., A Component-Labelling Algorithm Using Contour Tracing Technique," in *IEEE Proc. 7th International Conference on Document Analysis and Recognition (ICIDAR 2003), 0-7695-1960-1/03*, 2003.

[9] F. Chang, C. Chen, and C. Lu, "A linear-time component-labeling algorithm using contour tracing technique," *Computer Vision and Image Understanding*, vol. 93, no. 2, pp. 206–220, 2004.

[10] H. Hedberg, F. Kristensen, and V. Owall, "Implementation of a labeling algorithm based on contour tracing with feature extraction," in *IEEE International Symposium on Circuits and Systems, 2007. ISCAS 2007*, 2007, pp. 1101–1104.

[11] D. Bailey and C. Johnston, "Single pass connected components analysis," in *Image and Vision Computing New Zealand*, 2008, pp. 282–287.

[12] C. Johnston and D. Bailey, "FPGA implementation of a single pass connected components algorithm," *Electronic Design, Test and Applications*, pp. 228–231, 2008.

[13] J. Trein, A. Schwarzbacher, B. Hoppe, K. H. Noffz, and T. Trenschel, "Development of a FPGA Based Real-Time Blob Analysis Circuit," in *Irish Systems and Signals Conference, 2007. Derry, N. Ireland*, 2007, pp. 121–126.

[14] R. Walczyk, A. Armitage, and T. Binnie, "An Embedded Real-Time Pedestrian Detection System Using an Infrared Camera," in *IET Irish Signals and Systems Conference, 2009. IET ISSC 2009.*