This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCSVT.2015.2450371, IEEE Transactions on Circuits and Systems for Video Technology

A Resource-Efficient Hardware Architecture for Connected Components Analysis

Michael J. Klaiber, *Member, IEEE,* Donald G. Bailey, *Senior Member, IEEE,* Yousef O. Baroud and Sven Simon

***Abstract*—A resource-efficient hardware architecture for connected components analysis (CCA) of streamed video data is presented which reduces the required hardware resources especially for larger image widths. On-chip memory requirements increase with image width and dominate the resources of state-of-the-art CCA single-pass hardware architectures. A reduction of on-chip memory resources is essential to meet the ever increasing image sizes of high-definition and ultra-high-definition standards. The proposed architecture is resource-efficient due to several innovations. An improved label recycling scheme detects the last pixel of an image object in the video stream only a few clock cycles after its occurrence, allowing the reuse of a label in the following image row. The coordinated application of these techniques leads to significant memory savings of more than two orders in magnitude compared to classical two-pass connected component labelling architectures. Compared to the most memory-efficient state-of-the-art single-pass CCA hardware architecture, 42% or more of on-chip memory resources are saved depending on the features extracted. Based on these savings, it is possible to realise an architecture processing video streams of larger images sizes, or to use a smaller and more energy-efficient FPGA device, or to increase the functionality of already existing image processing pipelines in reconfigurable computing and embedded systems.**

***Index Terms*—Connected components analysis, connected components labelling, FPGA, parallel architecture, embedded image processing**

I. Introduction

CONNECTED *components analysis* (CCA) is a common step in image processing, extracting features such as area or size of arbitrary shaped objects in a binary image. It is based on *connected components labelling* (CCL) which creates a labelled image of the same dimensions as the original image where all pixels of each connected component are assigned a unique label. CCA is concerned with deriving the feature vector for each connected component from the binary input image I and does not output a labelled image. CCA and CCL are essential algorithms in computer vision and robotics. Increasing image resolutions beyond *high-definition* (HD) in consumer electronics [2] and frame rates above $100\ fps$ in high speed imaging [3] require high-performance hardware architectures. CCA is also used in image segmentation [4] and for evaluation of video surveillance footage [5]. For connected components analysis, a number of optimised hardware architectures and software implementations have been proposed in the recent past, all with the goal of avoiding the performance bottlenecks due to memory resources or memory bandwidth [6]–[13].

For hardware CCA architectures, the required resources are proportional to the image resolution [14]. This directly affects the throughput that can be achieved with a certain architecture or process technology. Any reduction in the hardware resources allows better performance to be achieved with the same technology or allows a switch to a more energy-efficient or less expensive hardware device.

A. Dedicated CCA HW architecture vs. SW implementation

A challenge for optimisation of CCA is that most algorithms are sequential and consist of a combination of compare, lookup and control operations [11]. A label is assigned to every pixel depending on its neighbourhood's labels. This data dependency on the current pixel's predecessors makes parallelisation non-trivial, but pipeline processing possible. From these data dependencies it follows that all operations for the currently processed pixel have to be finished before the operations on the subsequent pixel can be started. Therefore, the throughput depends on the execution time of the individual operations. Processing the pixel data of an image in real-time as it is streamed from an image source requires a high-throughput architecture, especially when a high-speed image sensor is used. Carrying out CCA on a general purpose processor (GPP) with a multi-core architecture requires a sequential execution of the comparison and control operations and several memory operations per pixel. If the size of data structures exceeds the size of on-chip memory of the GPP, slow off-chip memory has to be utilised. The execution time therefore can be dominated by the latency of the memory operations [15] limiting the overall throughput of the CCA algorithm and making the performance strongly dependent on the input data. Making use of a single-pass CCA algorithm on common GPP architectures might allow the required data structures to be stored in on-chip memory and solves the problem of the memory size. Nevertheless, the available on-chip memory bandwidth is usually shared among several processing cores reducing the performance for parallel memory access [16] which might limit the throughput. General purpose processors (GPP) are only a good choice for CCA or CCL as long as power dissipation or processing latency is of minor concern. Then, a high throughput and good scalability can be achieved by distributing the workload over a set of several GPP or GPGPU systems by either distributing parts of the pixel stream or assigning each image of the stream to a seperate processing unit. In contrast, when using a dedicated hardware architecture for CCA, all combinational operations for processing one pixel of the image can be carried out in a single clock cycle, some of them in parallel. Several on-chip memory structures ensure a low latency read and write of image labels at high bandwidth. This allows a faster processing of a single pixel,

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCSVT.2015.2450371, IEEE Transactions on Circuits and Systems for Video Technology

leading to a high processing throughput with low latency. The realisation of a dedicated hardware architecture is possible either as an application specific integrated circuit (ASIC) or on a field-programmable gate array (FPGA). Compared to a GPP architecture, both alternatives are typically superior in terms of power dissipation, which is especially important in embedded and mobile applications. Recent reconfigurable logic devices, FPGAs, consist of lookup tables (LUTs), registers and on-chip block-RAMs (BRAMs), which can be connected via a user-programmable connection network [17], [18]. For CCA, decisions and control operations are mapped to LUTs; for each operation requiring memory access a dedicated on-chip BRAM is assigned. The architecture proposed in this paper is customised for (but not limited to) a realisation as a hardware architecture on an FPGA. A speed-up is gained by distributing the computations to several pipeline stages working in parallel. The memory bandwidth is achieved by distributing the memory operations over several on-chip BRAMs. A high throughput by pipeline processing requires each pipeline stage to have a constant execution time to be able to keep up with the bandwidth of the image source. The goal of the processing architecture is to achieve a throughput of one pixel per clock cycle while maximising the clock frequency. When using BRAM resources having one clock cycle latency to represent data structures (e.g. directed graphs) only one lookup per clock cycle is possible. Recent CCA hardware architectures are close to the goal of one pixel per clock cycle by maintaining a rooted tree data structure of tree height of maximum one for labels to be processed in the current row. Labels already processed in the current image row may have a bigger tree height. A tree height of one at the beginning of the next image row is achieved by compressing the tree structure at the end of each row [1], [14]. This reduces the number of lookup operations to one lookup per clock cycle plus a maximum overhead of 18% at the end of the image row for tree compression, as shown in Section III-B.

B. Contributions of this paper

The architecture proposed in this paper reduces the resource requirements for connected components analysis by:

a) Detection and correct processing of not considered image patterns in previous publications: On the algorithm level, image patterns (e.g. Figure 10) were not taken into account in previous hardware architectures [14], [19] resulting in incorrect labelling. In the proposed architecture these patterns are detected and handled correctly, i.e. arbitrary image patterns can be analysed.

b) Using a novel control structure to detect the last pixel of an image object in the video stream at the earliest possible point in time: This allows the memory resources used by an object to be freed earlier.

c) Memory reduction by recycling of labels: On the architecture level a novel *label recycling* scheme is introduced. In combination with the proposed method for detecting the last pixel of an object, the memory for storing feature vectors is halved compared to [1] by eliminating redundant data structures.

d) A novel label translation scheme reducing the number of label lookups per pixel: The label translation scheme of [1] is simplified by reducing the number of lookups from two to one per label.

e) A method for out-of-order labelling for the efficient recycling of labels: As a consequence of the novel label recycling, augmented labelling is introduced, a technique to build consistent rooted tree data structures for components with out-of-order labels.

f) A reduction of memory resources for the entire architecture of 42%: The total memory required can be reduced by a factor of more than 200 compared to the classical connected component labelling algorithm [20]. Depending on the extracted feature vector and image size, 42% or more of memory resources can be saved compared to an optimised state-of-the-art architecture.

II. Related work

In classical *connected components labelling* algorithms an array L with the dimensions of the image is labelled to associate every pixel of an image with its connected component. The algorithm by Rosenfeld [20] first scans the binary image once using only local operations to assign an initial label to each pixel. If global dependencies are detected they are stored in an equivalence table. A second scan substitutes the labels assigned to a pixel by a representative label from the equivalence table [20]. When analysing the memory requirements, the labelled image and the equivalence table have to be taken into account. The sizes of both data structures depend on the maximum number of labels which is proportional to the number of pixels in the image. The general algorithm was improved by applying a union-find data structure [11], [21], [22] to achieve a quasi-linear scalability for the label substitution by applying heuristics to balance the height of the union-find data structure and path compression to avoid repetitive lookups [23], [24]. Khanna et al. proposed a two-pass algorithm applying a label reuse scheme to significantly reduce memory resources for the equivalence table [25]. The introduction of a single-pass approach eliminated the need to access each label several times [26], [27].

Execution performance can be improved by optimising the data access pattern to the memory hierarchy of the used processor (GPP or GPU) [12], [13], [28]–[31]. In most of these implementations, sequential data dependencies lead to sequential execution, and memory bandwidth is the limiting factor for a faster execution.

Memory resources were identified as a key issue for the scalability of CCA or CCL hardware architectures. The amount of memory required depends on the input image. Therefore, the image leading to the largest amount of memory (the worst case) has to be considered to cover all possible input images and is used to compare different architectures in the following. The principle of the algorithm by Rosenfeld [20] was adapted in several hardware architectures [29], [32], [33] to achieve real-time processing. However, the second pass requires the entire image to be stored. Single-pass connected components analysis methods eliminated the need to access

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCSVT.2015.2450371, IEEE Transactions on Circuits and Systems for Video Technology

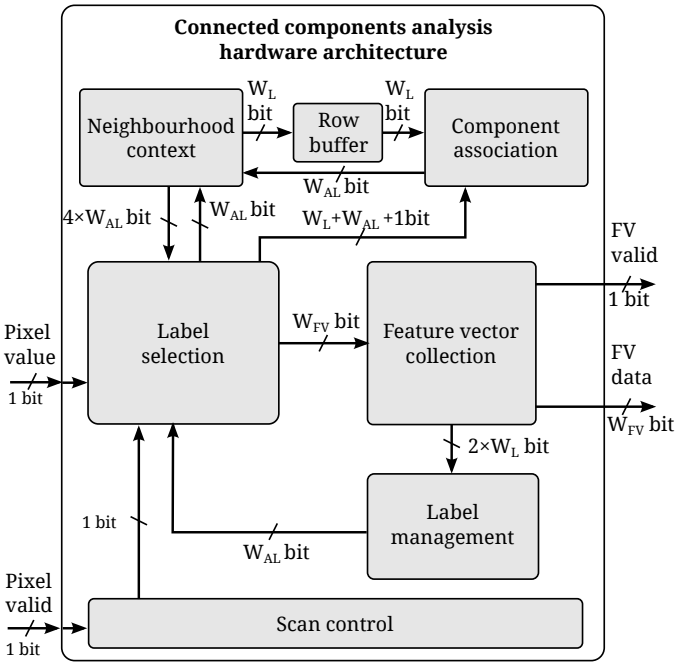

Fig. 1. Block diagram of the proposed hardware architecture for connected component analysis.

each label several times [26], [27] leading to reduced memory usage. Nakano et al. proposed a single-pass CCA architecture with the ability to analyse arbitrary connected components in an image which merge in the k rows above the current position. This architecture requires k image rows to be stored for a correct analysis [33]. To process a worst case image correctly, this single-pass architecture has to store the entire image. Bailey et al. proposed and implemented a single-pass hardware architecture which only requires a single row of labels to be stored to determine the current pixel's label [1], [19]. This is achieved by extracting the feature vector for each object so that the labelled image is not required. For a worst case image the architecture has to store equivalence relations for up to $\lceil \frac{W \times H}{4} \rceil$ labels in the equivalence table for an image W pixels wide and H pixels high. A throughput of up to 1 image pixel per clock cycle can be achieved. This architecture was optimised by Ma et al. by applying an aggressive relabelling scheme reusing memory resources after the last pixel of an image object is detected in the video stream [1]. This reduces the number of labels at the cost of additional hardware resources to translate the labels between the rows. The approach by Ma [1] is the most memory-efficient hardware architecture found in the literature and is therefore used in the following sections as a reference for comparisons.

III. Hardware Architecture

In the following, the nomenclature defined in Table I is used. The top level block diagram of the proposed architecture is depicted in Figure 1. It processes a binary input image I in forward raster scan order, as shown in Figure 2. The input image I is of size $W \times H$ and consists of *object pixels* and *background pixels* represented by 1 and 0. Two object pixels p_1, p_2 are connected if one pixel is in the other pixel's 8-neighbourhood or a path of neighbouring object pixels between p_1 and p_2 exists. A set of object pixels of I is called a connected component if every pair of pixels in the set is connected. A subset of connected object pixels of a connected component is called a *component segment*. The *feature vector* (FV) of a connected component or component segment is an n-tuple composed by functions of the component's pattern [34]. Connected components analysis is concerned with deriving the feature vector for each connected component from the binary input image I. The hardware architecture associates every pixel with its connected component by assigning a label and extracts the component's feature vector. Label 0 is reserved for background pixels. A connected component in the image I is called *finished* when a label has been assigned to all of its pixels.

TABLE I
Nomenclature used in the following sections.

Abbreviation	Name
DT	Data table
E	Active tags
FV	Feature vector
H	Image height
I	Source image
L	Labelled image
LS	Label stack
M	Merger table
N_L	Number of labels
N_M	Number of merger patterns per row
R	Reuse FIFO
RB	Row Buffer
S	Stack
TT	Translation Table
V	Valid flags
W	Image width
W_L	Width of a label
W_{AL}	Width of an augmented label
W_{FV}	Width of a feature vector

For the selection of the label L_X assigned to the current pixel $I[X]$ at position X, the *neighbourhood context* provides the labels at position A, B, C and D labelled L_A, L_B, L_C and L_D as depicted in Figure 2. To simplify discussion, L_{AD} is introduced to refer to the label L_A or label L_D since if $I[A]$ and $I[D]$ are both object pixels they will always have identical labels: $L_{AD} = L_A = L_D$. If a label is used as a Boolean variable, *true* indicates an object label, *false* the background label.

There are three *label patterns* with either zero, one or two different object labels in the *neighbourhood context* of an object pixel which are referred to as *new label pattern*, *label copy pattern* and *merger pattern.* These patterns are handled by the *new label operation*, the *label copy operation* and the *merger operation* which change the content of the data structures in the neighbourhood context, the component association and the feature vector collection. Two different object labels in the neighbourhood context where $I[X] = 1$ obviously belong to the same connected component. For this case the merger pattern induces a merger operation: the smallest object label of the neighbourhood context is assigned

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCSVT.2015.2450371, IEEE Transactions on Circuits and Systems for Video Technology

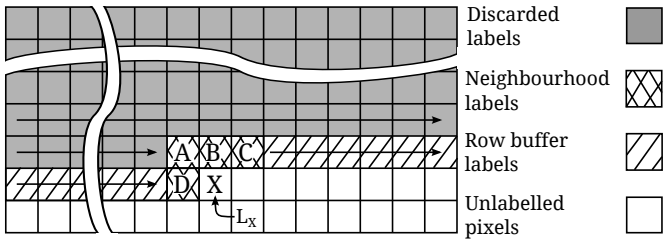

Fig. 2. The four different groups of image labels.

to L_X and merging labels are stored on the *merger table* M (see III-B). A merger operation on two labels l_0 and l_1 is referred to as merging l_0 and l_1. A merger operation on two component segments s_0 and s_1 of the same connected component is referred to as merging the component segments s_0 and s_1. The feature vector associated with each label is stored in the *data table* DT and is updated every time its object label is assigned to L_X. The new label operation and the label copy operation are discussed in Section III-B.

A. Neighbourhood Context and Row Buffer

The proposed architecture is based on the single-pass CCA algorithm from [14]. For this single-pass CCA algorithm, the decision as to which label to assign to L_X only depends on the labels of the previous image row from A to the end of the row and the labels of the current row left of X. In this architecture we distinguish between four different types of labels as shown in Figure 2:

- The *neighbourhood labels* (cross-hatched) L_A through L_D are required in the current clock cycle to determine the current pixel's label L_X.
- The *row buffer labels* (hatched) are required for labels associated with pixels processed in subsequent clock cycles.
- *Discarded labels* (marked grey) which are not required for further decisions.
- *Unlabelled pixels* (marked white) which have not been processed yet.

Figure 2 shows the source image I, where all pixels before X are already processed in raster scan order. Only the neighbourhood labels and the row buffer labels are relevant to determine the label L_X and must be stored for processing subsequent image rows. Since no labelled image is saved, the label of the current pixel is stored on the *row buffer RB* for one image row until it is required again for the decision process in the row below. The output of *RB* is connected to, and addresses the merger table which is discussed in Section III-B.

To parallelise and effectively accelerate the label selection, simultaneous read and write access to all labels of the neighbourhood context is required. This is realised by using a register for each of the labels L_A to L_D. After a merger operation, an update of L_B and L_C is required. The next cycle's L_B is assigned the current label L_X. When the next cycle's L_C is an object pixel it needs to be updated in case of a merger operation when $I[x+2, y-1] = 1$. These updates are realised by multiplexers at the input of the registers. The

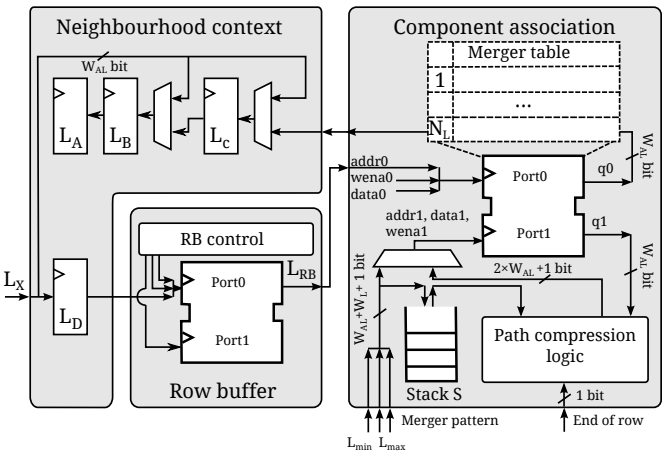

Fig. 3. Register-transfer level diagram of neighbourhood context, row buffer and component association unit.

size of the row buffer depends on the image width W. A label added to the row buffer is not accessed for $W-1$ cycles, i.e. it does not need to be read for the duration of processing one image row. This allows a realisation as a dual-port BRAM. Figure 3 shows the architecture of the *neighbourhood context* and *row buffer* on the register-transfer level.

B. Label Selection and Image Component Association

The *label selection* unit assigns the minimum object label of the neighbourhood context to L_X and generates control signals to update tables of the *component association* unit. When processing the image pixels in raster scan order, different initial labels may be assigned to different component segments of a connected component. To keep a record of merged labels, a rooted tree data structure containing vertices for the labels is used. Edges point from child vertices to parent vertices, as defined in [35]. This data structure is stored in a merger table M which is realised as a 1-D array. Each entry, $M[l]$, represents the directed edge from the vertex l to its parent $M[l]$. Every connected component and component segment is identified by the *root label* of its tree structure, which points to itself in M.

If the current pixel is an object pixel and all neighbour labels are background, a new label l is assigned to L_X and the merger table entry of l is initialised to point to itself, i.e. $M[l] := l$. A label copy operation assigns the object label in the neighbourhood to L_X. In the neighbourhood context of a merger pattern $L_{AD} \neq L_C$. To label each pixel correctly, the minimum label $L_{min} = min(L_{AD}, L_C)$ is assigned to L_X [14]. All pixels labelled $L_{max} = max(L_{AD}, L_C)$ processed before a merger pattern were already added to the row buffer RB and cannot be changed immediately, therefore the merger table entry of L_{max} is set to point to L_{min} so that the old labels in RB can be replaced by new labels after being read out, i.e. $M[L_{max}] := L_{min}$ which makes L_{min} the component segment's root label.

The merger table M is realised as a dual-port BRAM. One port is used as a read port to look up the labels output by the row buffer; a merger operation updates the rooted tree

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCSVT.2015.2450371, IEEE Transactions on Circuits and Systems for Video Technology

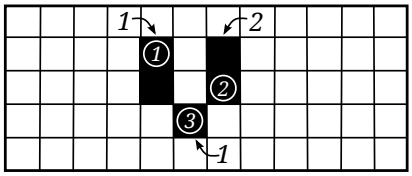

Fig. 4. This example image contains patterns inducing a new label ①, label copy ② or merger ③ operation. After the merger pattern at position ③ the merger table entry of 2 points to 1.

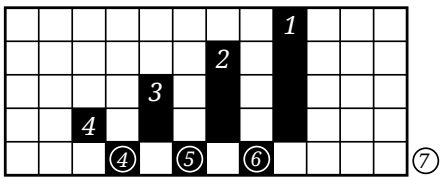

Fig. 5. Image with *chain* pattern. By saving the label pair of a merger operation on the stack S, the content of M $(4 \rightarrow 3 \rightarrow 2 \rightarrow 1)$ is updated at the end of the image row. Then the content of M is $4 \rightarrow 1$, $3 \rightarrow 1$, $2 \rightarrow 1$.

data structure on M via the second BRAM port. This enables continuous lookups in every clock cycle for the labels at the output of the row buffer, while the rooted tree data structure in M can be updated simultaneously via the write port.

All numbers in the figures used for the following examples represent the labels after the operations of the corresponding image pattern were carried out. Circled numbers in the examples, like ①, refer to positions in the image where patterns induce operations. In Figure 4 at positions ① through ③ an example for a new label pattern, a label copy pattern and a merger pattern are given.

A series of merger patterns of the same connected component where $L_C < L_{AD}$ creates a label *chain* in M, representing a path in the rooted tree, so that the labels stored in the row buffer do not yield the root label with a single lookup in M. A *chain* is resolved by making all its non-root label's vertices direct children of its root vertex. A reverse scan over the chains is executed to compress the tree in the merger table M to a height of one. In union-find algorithms this operation is called path compression [36]. The label pairs consisting of L_{min} and L_{max} for the reverse scan are pushed onto a *stack* S by every merger operation where $L_C < L_{AD}$ during the scan of the image. For every pair of labels popped off the stack S at the end of the row, the merger table is updated with $M[L_{max}] := M[L_{min}]$ which eventually resolves the chains.

An example of an image pattern resulting in a chain is illustrated in Figure 5. The chain generates 3 stack entries. The labels in the neighbourhood context of positions ④ to ⑥ lead to merger operations linking the component segments initially labelled 1 to 4 and push the label pairs $(3, 4)$, $(2, 3)$ and $(1, 2)$ on the stack S. At the end of the image row (position ⑦) the merger table M contains a chain where label 4 points to label 3, label 3 to 2 and label 2 to 1. For labels 3 and 4 a single lookup in M does not yield their component segment's root label 1 because of the chain $(4 \rightarrow 3 \rightarrow 2 \rightarrow 1)$. By popping the stack entries off S in reverse order, the content of M is compressed by updating the merger table entries of all labels to point to the root label 1.

The processing of each stack entry consists of two read and one write operation requiring in total three clock cycles. These operations can be pipelined, and with a dual-port BRAM for the merger table require on average one clock cycle per update [14]. A maximum chain consisting of $\lceil \frac{W}{2} \rceil$ merger patterns is possible in one image row, therefore the worst case stack depth is $\lceil \frac{W}{2} \rceil$. The image pattern creating $\lceil \frac{W}{2} \rceil$ merger operations is not the pattern that generates the maximum number of stack entries averaged over the whole image. The worst case creates a pattern for which a merger operation is carried out for every 5^{th} pixel of every row. In average for processing this worst case image, the number of stack entries after each image row is $\lceil \frac{W}{5} \rceil$ reducing the average throughput by 18% [14]. If the image source inserts a sufficiently long gap between two rows (e.g. the blanking period of an image sensor) then real-time processing is possible. Otherwise a buffer for the pixel stream at the input to cover bandwidth peaks allows real-time processing. For the stack S, write access is required during processing an image row and read access is necessary at the end of the row making the realisation as a single port BRAM sufficient.

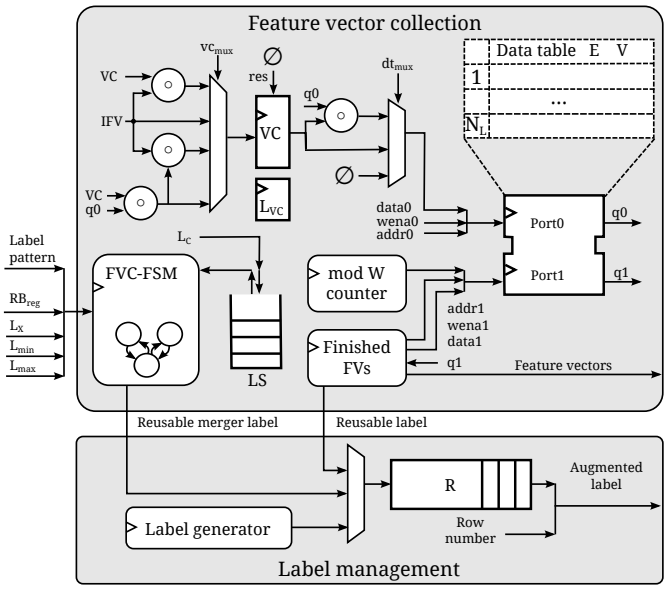

Fig. 6. Hardware units used for label recycling: the feature vector collection unit and the label management unit.

C. Label Recycling and Feature Vector Collection

In previous CCA architectures the memory requirements of M and DT are proportional to the image area, because in a worst case image a quarter of the pixels can be different connected components [14], [19]. However, at any time in the raster scan, the number of different labels assigned to pixels in an image row is only proportional to the image width [31], [25]. Memory requirements can be significantly reduced by recycling labels no longer in use, enabling entries of M and DT to be reused after a connected component is finished. The *label management* unit keeps a record of the unused labels on

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCSVT.2015.2450371, IEEE Transactions on Circuits and Systems for Video Technology

the *label reuse FIFO* R , which is initially filled with all labels, one through $\lceil \frac{W}{2} \rceil$. For each new label operation, an entry is read off R and assigned to the current pixel. For the reuse of already finished connected components two scenarios have to be distinguished:

- Recycling L_{max} after a merger operation
- Recycling the label of a finished connected component

The recycling of L_{max} requires that L_{max} is not contained in the row buffer anymore. This is the case one row after a merger operation was carried out, i.e. the label management unit has to delay the reuse of these labels until W more pixels have been processed.

Each connected component keeps its label until its end is detected. To detect finished connected components, every label which was assigned to a pixel in the previous row, but is not assigned to L_X in the current row, belongs to a finished object, i.e. the label and the associated memory resources can be reused immediately.

The position in the image where a label is recycled and added to R depends on the input image I, therefore the recycled labels on R are not necessarily in numerical order. The property from [14] that a merger operation always chooses the minimum label for L_X can therefore result in a corrupted merger table M not reflecting the actual label associations of the image. The image in Figure 7 demonstrates a case in which several merger operations create two root labels for a single connected component by always assigning the minimum label to L_X. To cover this case the concept of *augmented labels* (AL) is introduced. Labels are augmented with the row number they are generated in, i.e. each label consists of two parts: the row number and the index part. The row number is used to determine the minimum label during a merger operation: $L_{min} := L_C$ when $L_{AD}.row > L_C.row$, else $L_{min} := L_{AD}$. The index part is used to access tables such as the merger table M, e.g. $M[L_X]$ is realised as $M[L_X.index]$. Examples for augmented labels are given in Table III. Augmented labelling ensures that a merger operation always assigns L_{min} to the label created earlier in the scan process. With augmented labels, the rooted tree data structure on M always correctly reflects the current structure of unfinished connected components in the source image *I* detected up to the current position in the raster scan. The index parts of labels of finished connected components are written to the label reuse FIFO R. The label index of L_{max} of a merger pattern is added to R by a merger operation. The condition to delay reuse of L_{max} is implicitly fulfilled by using a FIFO to recycle labels.

To detect finished connected components, an *active tag* E is introduced for each connected component. If during the raster scan a label is assigned to L_X, its entry in E is updated with $y \bmod 3$, where y is the current row number. Any connected component for which its active tag is not updated in the current image row is finished. Their feature vectors are read out and their labels are recycled. A connected component is finished in row y, when its label does not appear in row $y+1$ of the labelled image L', therefore, the read-out and recycling is carried out in parallel to scanning row $y+2$. The active tag of a label ready to be recycled is, therefore, $y-2 \bmod 3$.

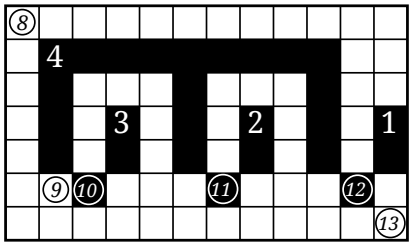

Fig. 7. Assigning the labels out of order creates a corrupted merger table.

TABLE II
DESCRIPTION OF DATA STRUCTURES AND COMBINING OPERATIONS FOR THE FEATURE VECTORS *bounding box*, *area* AND *first order moment*.

Feature	Data structure in DT	Initial feature vector $IFV(x,y)$	Combining $FV_a \circ FV_b$
Area	A	1	$A_a + A_b$
Bounding box	$\begin{pmatrix} x_{min} \\ y_{min} \\ x_{max} \\ y_{max} \end{pmatrix}$	$\begin{pmatrix} x \\ y \\ x \\ y \end{pmatrix}$	$\begin{pmatrix} min(x_{min,a}, x_{min,b}) \\ min(y_{min,a}, y_{1min,b}) \\ max(x_{max,a}, x_{max,b}) \\ max(y_{max,a}, y_{max,b}) \end{pmatrix}$
First Order Moment	$\begin{pmatrix} M_{10} \\ M_{01} \end{pmatrix}$	$\begin{pmatrix} x \\ y \end{pmatrix}$	$\begin{pmatrix} M_{10a} + M_{10b} \\ M_{01a} + M_{01b} \end{pmatrix}$

For a practical and efficient implementation the active tag E is mapped to a BRAM. This allows up to one label to be recycled per clock cycle and one feature vector to be read out in parallel with processing the following row. In this architecture the recycling process requires an additional 5 clock cycles of latency until the recycled label is available to be assigned to a new connected component. This is caused by pipeline registers and FIFO delays. The number of labels required for processing a worst case image is therefore $\lceil \frac{W+5}{2} \rceil$.

The feature vector *(FV)* for each connected component is accumulated during the raster scan and stored to the data table DT. For a new label operation the data table entry is initialised with the current pixel's feature vector referred to as the *initial feature vector* (IFV). A *label copy* operation combines L_X's DT entry with the *IFV* and a merger operation combines the DT entries of the two labels and the *IFV*. The operator $\circ$ is defined for combining the feature vectors. The combining operation, the data structure in DT and the *IFV* all depend on the feature vector, as shown in Table II for the *area*, *bounding box* and *first order image moment* feature vectors.

A new label operation requires one write operation for storing the feature vector, a label copy operation requires one read followed by a write operation and a merger operation requires two reads followed by a write and an invalidation operation of the data table DT and the active tags E. Additionally the read-out of feature vectors of finished connected components requires one read operation and one invalidation operation per finished connected component. Therefore, up to five memory operations per pixel are required. The proposed novel scheduling scheme makes a single BRAM port sufficient for updating feature vectors and a second BRAM port for read-out of finished connected components and invalidating entries no longer used. This reduces the memory resources required

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCSVT.2015.2450371, IEEE Transactions on Circuits and Systems for Video Technology

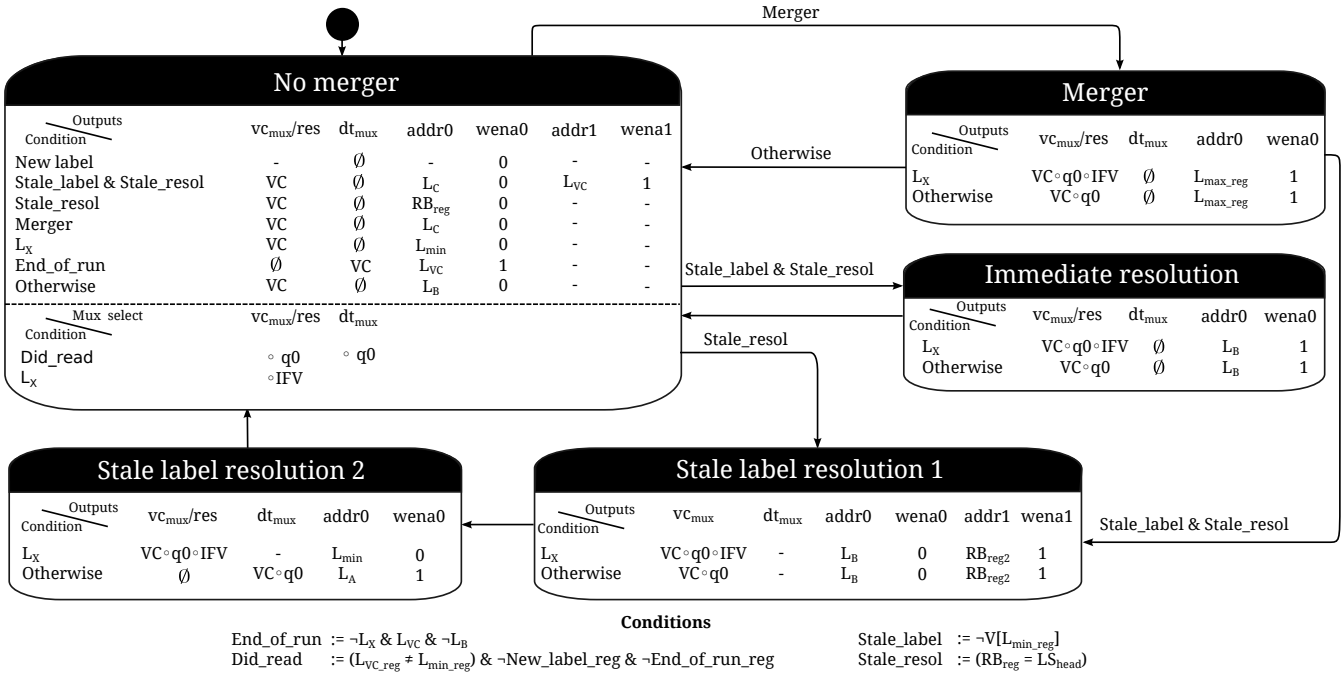

Fig. 8. Finite state machine for scheduling memory accesses in the *feature vector collection*.

for the data table DT by 50% (from two dual-port memories to a single dual-port memory) compared to Ma's architecture [1] and is a key improvement of the proposed architecture.

The architecture of the *feature vector collection* unit is shown in Figure 6. It contains the finite state machine scheduling the feature vector update process (FVC-FSM). Its Mealy state diagram is shown in Figure 8. The label pattern at the current position X serves as a condition to determine the FSM's outputs and the next FSM state. To save space in the figure the new label pattern, label copy pattern and merger pattern are abbreviated by new label, copy and merger. The condition at the top has the highest priority, if it does not match, the subsequent condition is evaluated. A *stale label pattern* is detected by the condition *Stale_label* which results from comparing RB_{reg}, a register at the output of the row buffer, and the label at the head of the label stack LS, the resolution of a stale lable is detected by the condition *Stale_resol*. Details on stale label processing are introduced in Section III-D. The *FVC-FSM* controls the BRAM ports *addr0*, *wena0, addr1 and wena1* directly, the ports *data0* and *q0* are connected with the *feature vector cache VC* and the *IFV* via the multiplexers vc_{mux} and dt_{mux} depending on the label pattern. Port *data1* is always $\emptyset$ since it is only used for invalidations. The feature vector cache VC is realised as a register to store the feature vector associated with the current label L_X to delay a write access to the data table DT. The label associated with the accumulated feature vector on VC is L_{VC}. The register L_{max_reg} contains the label of L_{max} of the pixel processed one clock cycle earlier. The BRAM port for feature vector updates can either be used for writing feature vectors or for read requests. The result of a read request appears on the output $q0$ in the next clock cycle. To accumulate the feature vectors with as few memory accesses as possible, the *feature vector cache VC* is updated with the feature vector of label L_X while the current connected component is scanned. A new label operation requires *VC* to be filled with the initial feature vector *IFV*, a label copy operation on L_D requires the feature vector on the *VC* to be combined with *IFV* and a label copy operation applied on L_A, L_B or L_C requires to combine *VC* and the feature vector at the $q0$. The feature vector on *VC* is written to its data table entry when a background pixel is reached at the end of a run when the condition *end_of_run* is true.

Performing a merger operation combines the feature vectors of L_{min}, L_{max} and *IFV*. This is scheduled over three clock cycles carried out when processing the current, the previous and following pixel. In general for every object pixel the *IFV* is combined with the feature vector cache *VC*. Additionally the following operations are required: In the first cycle L_C is applied to $addr0$ requesting L_C's feature vector. The feature vector of L_{AD} was either already read in the previous cycle as L_B if the previous pixel was background or is already on the *VC* if the previous pixel was an object pixel. In state *merger* the feature vector at $q0$ is combined with the *VC* and the data table entry associated with the previous L_{max} is invalidated. Depending on the following pixel the combined feature vector on *VC* is either written to the data table or further accumulated.

The new value assigned to the feature vector cache VC in each cycle is either IFV, $IFV \circ VC$, $VC \circ q0$ or $VC \circ IFV \circ q0$, by using the reset signal VC is cleared to $\emptyset$. For the input of the data table it is either an empty feature vector to clear a DT entry $\emptyset$, VC or $VC \circ q0$. If the current pixel is an object pixel, *IFV* is always combined with VC. A read request issued in the previous clock cycle is indicated by the *did_read* condition, which triggers the combination of VC with the output $q0$ in the current clock cycle. For the state *no merger* these two

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCSVT.2015.2450371, IEEE Transactions on Circuits and Systems for Video Technology

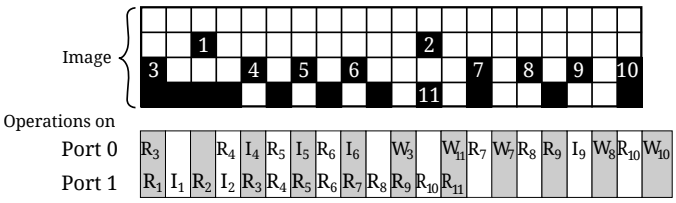

Fig. 9. The image shows the read (R), write (W) and invalidate(I) operations for BRAM port 0 and 1 for the last image row.

degrees of freedom are represented by additional conditions below a horizontal dashed line extending the conditions above the line. For example: If the current pixel in an object pixel, a read request was issued in the previous clock cycle and the current pixel is a merger pattern, the multiplexer vc_{mux} is set to $VC \circ q0 \circ IFV$.

The example in Figure 9 in the lower part illustrates the read, write and invalidation operations induced by the patterns of the image in the upper part of Figure 9 for each cycle on memory port 0 for feature vector collection and the operations for reading out the feature vectors of finished image objects on memory port 1. The index of each *read* (R), *write* (W) or *invalidate* (I) operation indicates the address it is applied on.

The second BRAM port is used to read out feature vectors of finished connected components and invalidate operations. An invalidate operation on *addr1* of the second port is carried out when *wena1* is one, during this cycle the read-out process is paused. There can be up to $\lceil \frac{W}{2} \rceil$ different connected components in a single image row. If all of them finish in the same image row, $\lceil \frac{W}{2} \rceil$ read and $\lceil \frac{W}{2} \rceil$ invalidation operations have to be carried out. Each feature vector and each active tag is associated with exactly one connected component, therefore E can be packed together into the same BRAM as DT.

Table III shows how augmented labels and the reuse FIFO R are used to assign the correct labels and to extract the feature vectors for each connected component of the image in Figure 7. In this example the augmented labels are represented by two digit numbers. The first digit represents the row number and the second digit the index. The changes of table and FIFO entries requiring a write operation to a memory are highlighted in grey. Before processing the image (position ⑧) all tables are initially blank, the label reuse FIFO R contains the reused label *4* at the head followed by *3, 2, 1*.

For a new label operation the connected components are labelled with augmented labels, i.e. label 4 becomes 14, label 3 becomes 33, etc. At ⑨ DT contains a feature vector for each of the component segments labelled 31, 32, 33 and 14.

The object pixel at ⑩ has two object labels 14 and 33 in its neighbourhood, i.e. a merger pattern. At this position $L_{min} = 14$ and $L_{max} = 33$, making $M[3]$ to point to augmented label 14 and recycling label 3 to R. The feature vector at $DT[3]$ is combined with the feature vector of $DT[4]$ and stored to $DT[4]$, $V[3]$ is set to *false* and $DT[3]$ is invalidated.

The patterns at positions ⑪ and ⑫ lead to merger operations which update the entries of labels 32, 31 and 14 in tables M, V, DT and E and return non-root labels to FIFO R.

TABLE III
CORRECT LABELLING OF IMAGE IN FIGURE 7 BY USING AUGMENTED LABELS. A # IN THE DATA TABLE DT INDICATES THAT THE CORRESPONDING ENTRY CONTAINS MEANINGFUL FEATURE VECTOR DATA, ∅ INDICATES THAT THE ENTRY IS EMPTY.

	Tables				FIFO R
⑧		1	2	3	4
	M	00	00	00	00
	V	f	f	f	f
	DT	∅	∅	∅	∅
	E	0	0	0	0
		↓			
		...			
		3			
		4			
		↓			
⑨		1	2	3	4
	M	31	32	33	14
	V	t	t	t	t
	DT	#	#	#	#
	E	1	1	1	1
		↓			
		...			
		...			
		5			
		↓			
⑩		1	2	3	4
	M	31	32	14	14
	V	t	t	f	t
	DT	#	#	∅	#
	E	1	1	0	2
		↓			
		3			
		...			
		5			
		↓			
⑪		1	2	3	4
	M	31	14	14	14
	V	t	f	f	t
	DT	#	∅	∅	#
	E	1	0	0	2
		↓			
		2			
		...			
		5			
		↓			
⑫		1	2	3	4
	M	14	14	14	14
	V	f	f	f	t
	DT	∅	∅	∅	#
	E	0	0	0	2
		↓			
		1			
		...			
		5			
		↓			
⑬		1	2	3	4
	M	14	14	14	14
	V	f	f	f	f
	DT	∅	∅	∅	∅
	E	0	0	0	2
		↓			
		4			
		...			
		5			
		↓			

At position ⑬ the connected component labelled 14 is detected as finished so its label is returned to *R* as well.

D. Stale labels

A label is called *stale* if a single lookup in M does not yield the root label. This has not been taken into account in previous hardware architectures [14], [19] and requires further processing. A *bridge pattern* is a component segment in which an object label appears two times in the current image row separated by background pixels. The bridge pattern's object pixels in the current row are referred to as its piers. The pixels belonging to the bridge which are above the current row are referred to as the bridge's arc. In the following figures the arc is either shown as a group of pixels or as a dashed line indicating a path of object pixels. Merging a bridge pattern's pier with a smaller label requires a lookup in M for the other pier to be labelled correctly in the neighbourhood context. Thus the height of the connected component*'s* tree structure becomes 1. At the beginning of each image row the height of all tree structures is $\leqq 1$ due to chain resolution. A merger pattern with a bridge pattern's left pier in the current image row which merged another component segment in the previous

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCSVT.2015.2450371, IEEE Transactions on Circuits and Systems for Video Technology

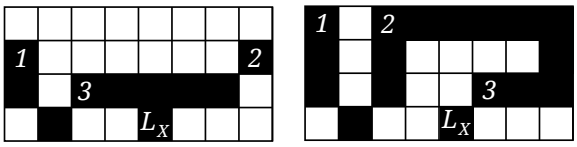

Fig. 10. Two basic examples for stale labels: At position X the content of the merger table M is: $3 \rightarrow 2 \rightarrow 1$. Label 3 is looked up to label 2, which has been merged with label 1 earlier in the current row.

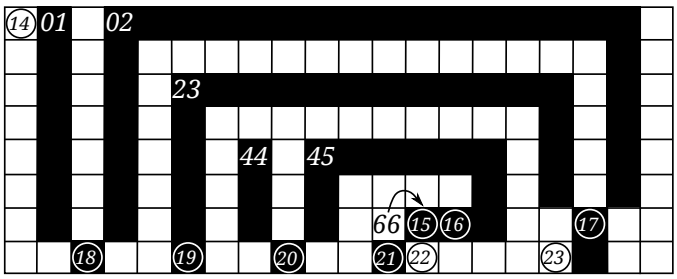

Fig. 11. Image containing nested connected components with stale labels.

row results in a tree height of 2 in the merger table M (Figure 10). A single lookup of a label which has a distance of 2 to its root label in the rooted tree structure in M does not yield the root label. If such a non-root label appears as the minimum label in the neighbourhood of an object pixel, the wrong label is assigned to L_X.

A root label on M can be detected by using an additional lookup to check if the label points to itself. A *valid flag* V for each label allows stale labels to be detected without this additional lookup: A new label operation sets the new label's V flag to *true*, a merger operation sets the V flag of L_{max} to *false*. Reading out V along with DT tells whether the assigned label is a root label. If a stale label is output by the row buffer, the label assigned to L_X is not a root. Therefore, L_X's feature vector is stored to DT until its root label appears in the neighbourhood context and their feature vectors are combined.

If there are nested bridge patterns, several stale labels can be detected before their root labels appear in the neighbourhood context. To keep a record of the stale labels which have to be merged with their root labels a *label stack LS* is introduced. The label L_X is pushed to LS whenever its V flag is *false*. Its feature vector is temporarily stored on the data table entry of L_X, which is unused for any non-root label. When LS_{head}, the label at the head of the label stack, is equal to the output of the row buffer, it is popped off LS to combine the feature vector of LS_{head} and its root label. In the FSM in Figure 8 the states *Stale label resolution 1* and *Stale label resolution 2* handle merger patterns of feature vectors of non-root labels. If L_X is detected to be stale and its resolution is detected simultaneously, the feature vectors are handled as shown in state *immediate resolution* of Figure 8. The V flag and the data table DT each have one association per label, i.e. they can be mapped to the same logical BRAM resource. The maximum number of stale labels which can appear in an image row is up to $\lceil \frac{W}{10} \rceil$. Depending on the image size, stack *LS* can either realised as BRAM or distributed RAM.

The two patterns in Figure 10 generate tree structures of height 2 in M resulting in stale labels. In both images label 2 and 3 are merged in the previous image row. After merging label 1 and 2 in the current image row, all pixels labelled 3 leaving the row buffer are translated to label 2 by M assigning a non-root label to L_X.

Table IV reflects the steps for processing the image in Figure 11 making use of the label stack *LS*. At first we consider the connected component consisting of the component segments initially labelled 01, 02 and 23. The merger pattern at (17) induces a merger operation updating $M[3]$ to label 02 and sets the valid flag of label 23 to false, i.e. $V[3] = false$ indicates that label 23 is not a root label. The feature vectors of the component segments 02 and 23 are combined and stored to $DT[2]$, the data table entry of label 3 is cleared, $DT[3] \coloneqq \emptyset$. Therefore, the tree height of the component segment labelled 02 is one before reaching position (18). The merger operation induced by the merger pattern at position (18) updates $M[2]$ to label 01 and sets $V[2] \coloneqq false$. The tree structure of the connected component labelled 01 is therefore of height two which makes label 23 stale. At position (19) L_B is 02 as a result of a single lookup of label 23. Since $V[2]$, was set to false at (18), a non-root label is detected and assigned to L_X. From this it follows that the feature vector of the current pixel is stored to $DT[2]$ and label 2 is added to the label stack *LS*. At position (23) the label stored on the register attached to the output of the row buffer RB_{reg} is equal to the label at the head of the label stack *LS*. The feature vector of the non-root label 02 temporarily stored at $DT[2]$ is combined with the feature vector of its root label 01 and stored to $DT[1]$. Simultaneously $DT[2]$ is cleared and label 2 is popped off *LS*. For the inner connected component consisting of the component segments 44, 45 and 66, the merger operation induced by the merger pattern at (16) updates $M[6]$ to label 45 and sets $V[6] \coloneqq false$. The feature vectors of the component segments 45 and 66 are combined and stored to $DT[5]$, the data table entry of label 66 is cleared. As before, label 66 becomes stale because of the merger operation at (20), which increases the height of the tree structure of the connected component labelled 44 from one to two and sets $V[5] \coloneqq false$. The feature vectors of the component segments 44 and 45 are combined and stored at $DT[4]$, the data table entry of label 45 is cleared, $DT[5] \coloneqq \emptyset$. At (21) the non-root label 45 (which is looked up from label 66) is assigned to L_{min}. Therefore, the feature vector of the pixel at (21) is stored at DT[5] and label 5 is added to the label stack *LS*. At position (22) the label stored on the register attached to the output of the row buffer RB_{reg} is equal to the label at the head of the label stack *LS*, the feature vector of the non-root label 45 temporarily stored at $DT[5]$ is combined with the feature vector of its root label 44 and stored in $DT[4]$. Simultaneously $DT[5]$ is cleared and label 5 is popped of *LS*.

E. Validation

The architecture relies on the assumption that a single lookup in M is always sufficient to assign a label to L_X which associates the pixel at position X with its connected

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCSVT.2015.2450371, IEEE Transactions on Circuits and Systems for Video Technology

TABLE IV
FEATURE VECTOR EXTRACTION FOR THE CONNECTED COMPONENTS OF FIGURE 11 WHICH CONTAINS SEVERAL STALE LABELS. AUGMENTED LABELS IN M ARE REPRESENTED BY A TWO DIGIT NUMBER - THE FIRST DIGIT IS THE ROW, THE SECOND THE INDEX. THE VALID FLAGS IN V ARE EITHER *(t)rue* OR *(f)alse*.

	Tables							LS
		1	2	3	4	5	6	
(14)	M	00	00	00	00	00	00	↓↑
	V	f	f	f	f	f	f	
	DT	∅	∅	∅	∅	∅	∅	
	E	0	0	0	0	0	0	∅
(15)	M	01	02	23	44	45	66	↓↑
	V	t	t	t	t	t	t	
	DT	#	#	#	#	#	#	
	E	0	0	2	1	1	0	∅
(16)	M	01	02	23	44	45	45	↓↑
	V	t	t	t	t	t	f	
	DT	#	#	#	#	#	∅	
	E	0	0	2	1	0	0	∅
(17)	M	01	02	02	44	45	45	↓↑
	V	t	t	f	t	t	f	
	DT	#	#	∅	#	#	∅	
	E	0	0	0	1	0	0	∅
(18)	M	01	01	02	44	45	45	↓↑
	V	t	f	f	t	t	f	
	DT	#	∅	∅	#	#	∅	
	E	1	1	0	1	0	0	∅
(19)	M	01	01	02	44	45	45	↓↑
	V	t	f	f	t	t	f	
	DT	#	#	∅	#	#	∅	
	E	1	1	0	1	0	0	2
(20)	M	01	01	02	44	44	45	↓↑
	V	t	f	f	t	f	f	
	DT	#	#	∅	#	∅	∅	
	E	1	0	0	1	1	0	2
(21)	M	01	01	02	44	44	45	↓↑
	V	t	f	f	t	f	f	
	DT	#	#	∅	#	#	∅	
	E	1	0	0	1	1	0	5 2
(22)	M	01	01	02	44	44	45	↓↑
	V	t	f	f	t	f	f	
	DT	#	#	∅	#	∅	∅	
	E	1	0	0	1	1	0	~~5~~ 2
(23)	M	01	01	02	44	44	45	↓↑
	V	t	f	f	t	f	f	
	DT	#	∅	∅	#	∅	∅	
	E	1	1	0	1	0	0	~~2~~

component. To achieve this, the tree structures in the merger table M are compressed to a height of one by different means: Merger operations can either affect the tree structure in M of labels to the left or to the right of position X. Label chains create tree structures with height > 1 for labels left of the current position X which are relevant for the processing of the next image row. Stale labels create a tree height of 2 for labels right of X which are relevant for processing the current image row. With the stacks S and LS both cases, chains and stale labels, are handled by memorising temporary differences in

TABLE V
VALIDATION OF POSSIBLE COMBINATIONS OF MERGER PATTERNS. *Don't cares* IN THE TABLE ARE MARKED BY '-'.

Combinations of Merger Patterns							
Merger Pattern #	Stale		Bridge		$L_{AD} > L_C$	Pattern Possible	Correct labelling
	AD	C	AD	C			
0	0	0	0	0	0	✓	✓
1	0	0	0	0	1	✓	✓
2	0	0	0	1	0	✓	✓
3	0	0	0	1	1	✓	✓
4	0	0	1	0	0	✓	✓
5	0	0	1	0	1	✓	✓
6	0	0	1	1	0	✓	✓
7	0	0	1	1	1	✓	✓
8	0	1	0	0	0	✗	
9	0	1	0	0	1	✗	
10	0	1	0	1	0	✗	
11	0	1	0	1	1	✓	✓
12	0	1	1	0	0	✗	
13	0	1	1	0	1	✗	
14	0	1	1	1	0	✗	
15	0	1	1	1	1	✓	✓
16-19	1	0	0	-	-	✗	
20	1	0	1	0	0	✓	✓
21	1	0	1	0	1	✗	
22	1	0	1	1	0	✓	✓
23	1	0	1	1	1	✗	
24-31	1	1	-	-	-	✗	

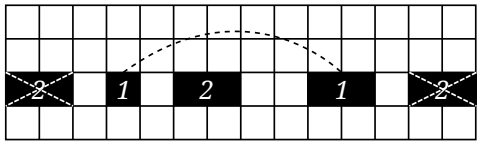

Fig. 12. Impossible scenario: Label between bridge piers appears to the right or left of the bridge.

M and re-establishing the single lookup principle by updating the merger table M after a chain or a stale label.

To ensure that each pixel is associated with its connected component, the different combinations for stale label and bridges are enumerated in Table V. Chains and stale labels are formed by merger patterns or combinations of merger patterns and bridge patterns. In an image neighbourhood not all combinations of stale labels and bridge patterns are possible. This is analysed in the following. The cases not possible are marked with a ✗ in Table V. There is no need to further examine whether labelling is correct for these impossible combinations. All combinations that can occur are marked by a ✓ and were examined to assign the correct label to the current label L_X. In the image in Figure 12 the current image row contains two piers of a bridge pattern labelled 1 and a connected component between the two bridge piers labelled 2. In the already scanned image part no pixels can be labelled 2 outside the bridge. Therefore, a pixel's label between two bridge piers of the same bridge is always larger than the bridge's label. Merger patterns *10, 14, 21, 23* of Table V are therefore impossible.

By the definition of Section III-D a stale label is always created by a bridge. This excludes merger patterns *8, 9, 12, 13, 16-19* from the possible patterns, as marked in Table V. For label L_C to be stale, one bridge is necessary, for the label

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCSVT.2015.2450371, IEEE Transactions on Circuits and Systems for Video Technology

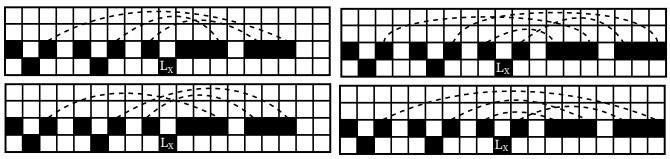

Fig. 13. All failing attempts to make the two object labels of a merger pattern stale.

L_{AD} to be stale, two bridges are required. The four different combinations to form a pattern making both L_{AD} and L_C stale are shown in Figure 13, bridges are indicated by dashed lines. All four attempts to make both L_{AD} and L_C stale fail because of intersecting bridges making them a connected component, therefore the merger patterns *24-31* of Table V can never exist. Table V shows that all cases of merger patterns for stale labels and bridges which are possible are labelled correctly.

To ensure the functional correctness of the implementation of the proposed architecture we streamed all possible pixel combinations for a small size test image into the architecture and verified the outputs against a reference implementation [37]. The size of the test image chosen for this full verification is a trade-off between the complexity of the pixel patterns in the image and processing time which grows exponentially with the number of pixels. For the chosen image size of 9×5 pixels, all of the possible 2^{45} different image patterns were successfully verified against the reference implementation applying the classical two-pass connected components labelling algorithm. To reduce the duration of the verification process an *on-chip verification environment* realised in hardware on an FPGA was used. It contains 75 parallel working instances of the proposed CCA architecture and the reference implementation, enabling an accelerated verification process, from more than one year of verification time on a single instance down to seven days for all instances processing in parallel. The exhaustive verification of all 9×5 image covers the cases of Table V, still parts of the implementation are not exercised by images of this size. This did require further validation to make sure the implementation realises all scenarios of the previously described architecture correctly, e.g. for nested stale labels such as in Figure 11. These parts of the implementation were validated by checking the code coverage of the VHDL code in behavioural simulation to ensure a correct functionality [38].

IV. Experimental Results and Discussion

In this section the results for the realisation of the proposed connected components analysis architecture is evaluated and benchmarked. Its performance and memory-efficiency is compared to other connected components analysis hardware architectures for different image sizes from *VGA* with 640×480 pixels per image to *ultra-high-definition (UHD)* with image sizes up to 7680×4320 pixels [2].

A. Memory Requirements

Table VI compares the number of bits required for the memories integrated in the different processing blocks of the CCA architectures for an image of the size $W \times H$ pixels for

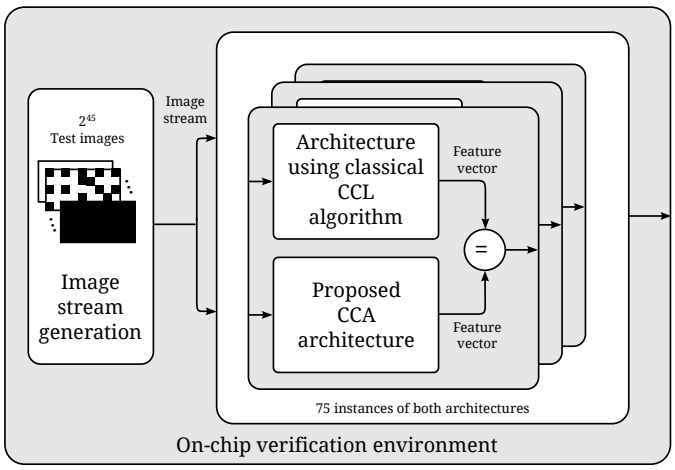

Fig. 14. Block diagram of the on-chip verification environment which successfully verified all combination of a 9×5 image against a reference implementation.

both the architectures in [1] and the proposed architecture as well as the classical CCL algorithm [20]. Both architectures require the row buffer RB and the stack S of the same size for the connected components analysis.

For the CCA architecture of [1] the following on-chip memories are required: Every second pixel can be a different connected component or be a component segment merging another segment later in the image. Therefore, the number of labels N_L is only dependent of the image width. The architecture requires two merger tables M, one to store the label pairs for each merger pattern of the previous row and one to store each label pair of the merger pattern of the current row. To store a relation between two labels, each entry of the merger table M is as wide as a label (W_L). The aggressive relabelling scheme requires a translation table TT with N_L entries of width W_L. For the feature vector collection two data tables DT, one for the feature vectors of the previous row's labels and one for the feature vectors of the current row's labels are required. The width W_{FV} of each entry of the data table DT is dependent on the feature to be extracted.

The proposed CCA architecture requires the following on-chip memory: The number of labels N_L depends on the image width plus a constant to compensate for the 5 clock cycles latency of the label recycling process, and is therefore $\lceil \frac{W+5}{2} \rceil$. The augmented labelling (AL) requires the merger table of the proposed architecture to be as wide as the width of an augmented label W_{AL}. The label reuse FIFO *R* of the label management unit needs to be able to store all labels, i.e. requires a depth of N_L labels. For the feature vector collection a single data table DT with N_L entries of width W_{FV} is sufficient.

Table VII compares the amount of on-chip memory required between the classical CCL algorithm [20], the single-pass architecture by Ma [1] and the proposed architecture for extraction of the bounding box feature vector of images of different sizes. We compare between on-chip memory required for the *label assigning* process, such as the row buffer RB, the stack S, the merger table M, the valid flags V, the FIFO

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCSVT.2015.2450371, IEEE Transactions on Circuits and Systems for Video Technology

TABLE VI
COMPARISON OF THE MEMORY BITS REQUIRED FOR COMPONENTS ANALYSIS FOR THE CLASSICAL CCL ALGORITHM [20], THE SINGLE-PASS ARCHITECTURE FROM [1] AND THE PROPOSED ARCHITECTURE FOR DIFFERENT IMAGE SIZES.

	Classical [20]	Single-pass [1]	This work
N_L	$\lceil \frac{W \times H}{4} \rceil$	$\lceil \frac{W}{2} \rceil$	$\lceil \frac{W+5}{2} \rceil$
N_M	–	$\lfloor \frac{W-1}{2} \rfloor$	$\lfloor \frac{W-1}{2} \rfloor$
W_L	$\lceil \log_2(N_L) \rceil$	$\lceil \log_2(N_L) \rceil$	$\lceil \log_2(N_L) \rceil$
W_{AL}	–	–	$W_L + \lceil \log_2(H) \rceil$
L	$N_L \times W_L \times W \times H$	–	–
RB	–	$N_L \times W_L$	$N_L \times W_L$
S	–	$2 \times W_L \times N_M$	$2 \times W_L \times N_M$
M	$N_L \times W_L$	$2 \times N_L \times W_L$	$N_L \times W_{AL}$
R	–	–	$N_L \times W_L$
DT	–	$2 \times N_L \times W_{FV}$	$N_L \times W_{FV}$
TT	–	$N_L \times W_L$	–
LS	–	–	$N_L \times \lceil \frac{W}{10} \rceil$
V	–	–	N_L
E	–	–	$2 \times N_L$

TABLE VII
COMPARISON OF ON-CHIP BRAM BITS REQUIRED FOR COMPONENTS ANALYSIS FOR THE CLASSICAL CCL ALGORITHM, THE SINGLE-PASS ARCHITECTURE FROM [1] AND THE PROPOSED ARCHITECTURE FOR DIFFERENT IMAGE SIZES. THE SIZES OF DATA TABLE DT CORRESPOND TO EXTRACTING BOUNDING BOX AND AREA FEATURES SIMULTANEOUSLY.

	VGA 640 × 480	DVD 720 × 576	HD720 1280 × 720	HD1080 1920 × 1080	UHD3k 3840 × 2160	UHD4k 4096 × 2160	UHD8k 7680 × 4320
Rosenfeld's classical two-pass algorithm [20]							
L	5M	7M	16M	39M	174M	194M	763M
M	1.3M	1.7M	4M	9M	43M	48M	190M
DT	4.3M	5.9M	13.8M	32.6M	143.0M	157.0M	622.0M
$\sum$	10.8M	14.7M	34.5M	81.9M	0.36G	0.40G	1.57G
Ma and Bailey's optimised single-pass architecture [1]							
M	5760	6480	12800	19200	42240	45056	92160
TT	2880	3240	6400	9600	21120	22528	46080
RB	5760	6480	12800	19200	42240	49152	92160
S	2880	3240	6400	9600	21120	24576	46080
DT	35840	41040	76800	120960	264960	290816	576000
$\sum$	53120	60480	115200	178560	391680	425984	852480
This work							
M	5760	6840	12800	20160	44160	49152	96000
R	2907	3267	6430	9630	21153	24612	46116
LS	576	648	1280	1920	4224	4920	9216
RB	5760	6480	12800	19200	42240	49152	92160
S	2880	3240	6400	9600	21120	24576	46080
DT	18243	20883	39043	61443	134403	147459	291843
E	646	726	1286	1926	3846	4102	7686
V	323	363	643	963	1923	2051	3843
$\sum$	36772	42084	80039	123879	271146	297829	589101

for reused labels R, the translation table TT and memory required for the feature vector collection, such as the data table DT, the label stack LS and the active tags E. The values of the table of the proposed architecture and the values of the architecture of [1] are depicted in the diagram in Figure 15. For all image sizes from *VGA* to *UHD8k* the proposed CCA architecture requires fewer on-chip memory resources than the architecture of [1]. By halving the resources for feature vector collection, the resources required for the entire architecture can be reduced by up to 31% for extracting the bounding box and the area feature vector for each connected component. The width of the bounding box, the area and the first order moment feature vector is dependent on the image dimension. For the simultaneous extraction of multiple features, the width of the data table increases accordingly. The feature vector for the three features, bounding box, area and first order image moment adds up to 175 bits per connected component. When realising a CCA architecture extracting those three features simultaneously, up to 42% of memory resources are saved compared to [1]. For a wider feature vector even more memory resources are saved.

As discussed in the introduction, the access time to the memory structures, especially the latency, was identified to be the most important criteria for processing a pixel stream with a high throughput and low latency. Therefore, offloading the memory structures to off-chip memory to save chip area counteracts the key idea of the architecture. If implementing the proposed architecture on an ASIC, there are a number of possibilities to realise the memory structures, either based on SRAM or DRAM cells [39]. While DRAM cells require fewer transistors per bit than SRAM cells, in general, any data stored in DRAM must be refreshed periodically. In CCA the data structures are accumulated from one row to the next and are therefore only up to date for a maximum of two image rows before they are either changed or read out. For typical image sizes this process is less than a millisecond which is significantly lower than the refresh rate of a typical DRAM cell. This allows the smaller DRAM cells to be used without refresh. This applies to all internal memory structures except the reuse FIFO R which needs to store the unused labels for the duration of up to one frame.

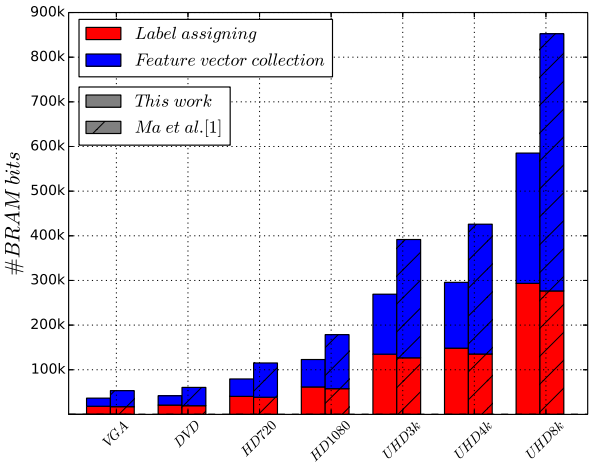

Fig. 15. The bar diagram shows the number of on-chip BRAM bits for the label assigning in red, the feature vector collection in blue. The hatched bars on the right shows the memory required for the architecture in [1], the left bars show the memory required for the proposed CCA architecture for different image sizes.

B. Benchmark

The performance of the architecture is measured in a manner similar to the benchmark in [11] with test images from the

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCSVT.2015.2450371, IEEE Transactions on Circuits and Systems for Video Technology

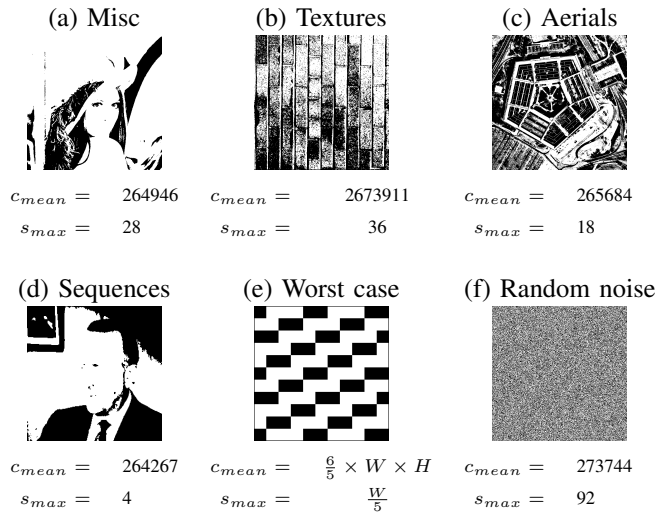

Fig. 17. This figure gives the mean number of processing cycles c_{mean} and the maximum stack size s_{max} for (a) - (d) which are test images from USC-SIPI Image Database [40]. (e) Worst case image [14] with the maximum number of merger patterns. (f) Random noise image with 50% of object pixels as in Figure 16(f).

Standard Image Database (SIPI) [40] and random images with different densities of object pixels. All images for this performance evaluation are of size 512×512 pixels. Greyscale images are binarised using the threshold value determined by Otsu's method [41]. All results in this section are acquired by behavioural simulation of the implementation of the CCA architecture implemented in VHDL.

Every image pixel can be processed in one processing cycle; additional processing cycles at the end of the image row result from chain resolution. From this, it follows that the worst case processing time occurs by maximising the number of entries on stack S. To analyse the worst case processing time for different image sizes from 64 pixels to 4 megapixels, images with the worst case pattern of Figure 17(e) are evaluated. Figure 16(e) shows that the number of processing cycles scale linearly for the examined image sizes.

Random images were used to evaluate the execution time against the number of object pixels in an image. Figure 16(f) shows the execution time for processing a random image as a function of the density of object pixels in the image for 512×512 images. For 0% and 100% density of object pixels, the image contains either zero or one connected component; the highest number of connected components is at 50%. The diagram in Figure 16(f) shows that the execution time is maximum between 40% and 50% image object density. This corresponds to an overhead due to stack processing at the end of the row of less than 5%, which is significantly lower than in the worst case. To evaluate the architecture's performance for processing natural images representative image series from *SIPI database* containing 215 typical images are used divided to the categories *misc*, *textures*, *aerials* and *sequences*. In Figure 17 the results for the mean processing cycles per image for each image series and the maximum number of stack entries for processing each image series is shown.

C. Hardware Resources

The hardware architecture for the proposed CCA was described in *VHDL* and implemented for the Xilinx Virtex 6 VLX240T-2 (speedgrade -2, 40 nm technology), Xilinx Spartan 6 SLX150T-2 (speedgrade -2, 45 nm technology) and Xilinx Kintex 7 K325T-2L (speedgrade -2L, 28 nm technology) to explore the performance on different FPGA devices. To acquire comparable mapping and timing results, for the implementation on all FPGAs the *PlanAhead 14 default* implementation strategy was used and nothing but the CCA architecture was implemented on the FPGA devices.

In the diagrams of Figure 16(a)-(c) the resources required for the implementation of the proposed CCA architecture is shown for a number of typical image sizes from VGA to UHD8k for Kintex 7, Virtex 6 and Spartan 6. The diagram in Figure 16(a) shows the number of lookup tables (LUTs) which realise logic functions with up to 6 inputs [17], [42]. The number of registers is shown in the diagram in Figure 16(b). Both the number of LUTs and registers increase quasi-logarithmically with the image width. The number of slice registers is nearly identical for the three examined FPGA devices, the number of LUTs varies between the device families depending on the image size. The Kintex 7 and Virtex 6 devices provide 18kBit and 36kBit BRAM resources, Spartan 6 BRAMs are 8kBit and 16kBit. Since the unused memory resources of a partially used BRAM are not available to other components on the FPGA they are considered to be used for the comparison. This results in a different number of required BRAM bits for Spartan 6 and Virtex 6 or Kintex 7. The diagram in Figure 16(c) shows the number of used BRAMs for different image sizes. The number of required on-chip memories scales linearly with the image width. The throughput of the CCA architecture is mainly proportional to the maximum operating frequency f_{max} which is shown in the diagram in Figure 16(d). For the implementation of the CCA architecture on Kintex 7 and Virtex 6 f_{max} is almost twice that implemented on the Spartan 6 which has a direct impact on the throughput.

The throughput can be classified in two parts: a static part with one pixel per clock cycle which is completely independent of the image content and a data-dependent part for resolving the label pairs of merger patterns (stored on S) depending on the image content. The data-dependent part lasts between 0 clock cycles if the stack S has no entries and $\lceil \frac{W}{5} \rceil$ clock cycles per image row for the worst case pattern of Figure 17(e). Thus considering the worst case pattern an image stream of up to 166 megapixels per second can be processed in real-time (for VGA resolution).

D. Comparison to Other Hardware Architectures

In Tables VIII and IX the algorithms and the implementations of several published CCA hardware architectures are compared. These publications suggest a diverse variety of methods on the algorithmic level as well as on the architectural level and the used technology for implementation. The differences of these architectures on the algorithmic level include the connectivity, either 4-connectivity of 8-connectivity, the scan

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCSVT.2015.2450371, IEEE Transactions on Circuits and Systems for Video Technology

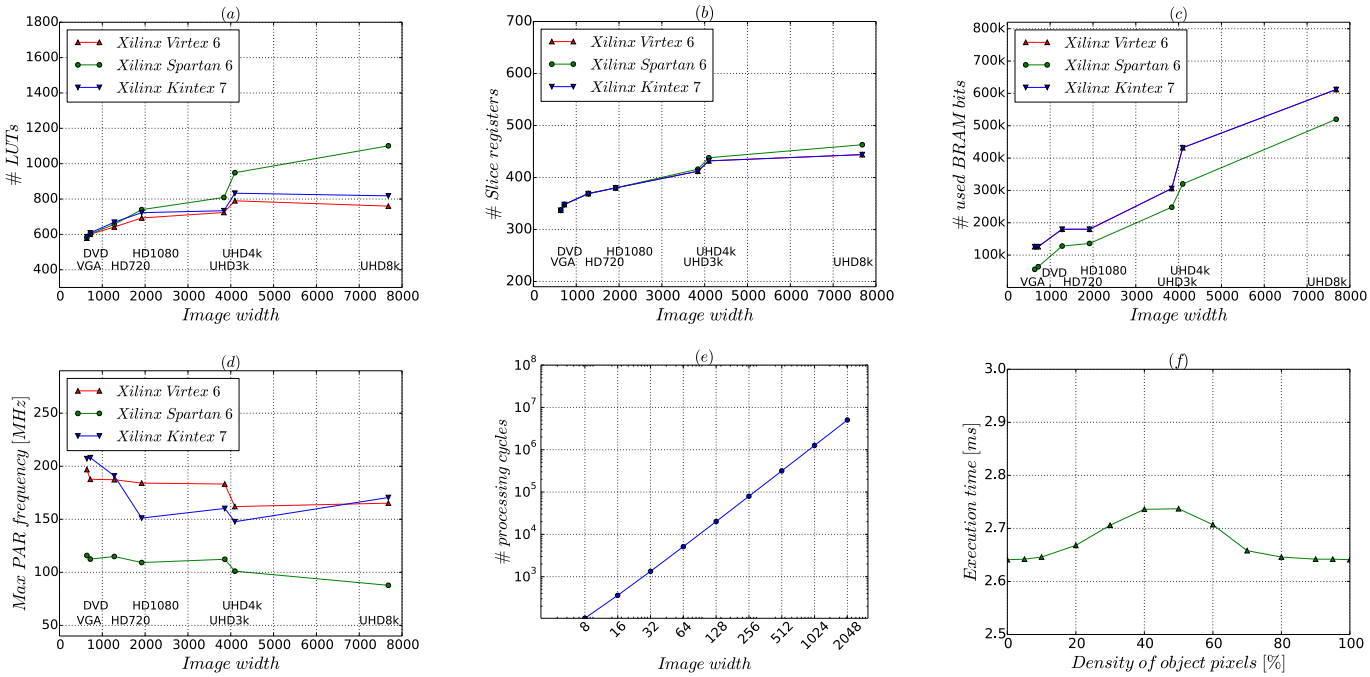

Fig. 16. The diagrams in (a) through (c) show the number of lookup tables (LUTs), slice registers and on-chip BRAM bits required by the implementation of the proposed connected components analysis architecture for different image sizes and different FPGA families. Diagram (d) shows the maximum operation frequency after the place&route (PAR). In (e) the number of clock cycles for processing square images of different sizes with the worst case pattern from Figure 17 is shown, (f) shows the execution time of the proposed CCA architecture operated at 100 MHz for 512×512 images filled with random noise for different densities of object pixels.

TABLE VIII
COMPARISON OF THE ALGORITHM PROPERTIES OF CCA HARDWARE ARCHITECTURES.

Algorithm	# Passes	Scan method	Connectivity	Worst case identified
[1]	Single-pass	Pixel-based	8	True
[10]	Single-pass	Run-based	4	False
[14]/ [19]	Single-pass	Pixel-based	8	True
[33]	Single-pass	Run-based	4	True
[43]	Two-pass	Run-based	8	True
This work	Single-pass	Pixel-based	8	True

method, either pixel by pixel processing or run processing, and the number of scans, either single-pass or two-pass. On the architectural level they differ in image sizes, extracted feature vector and the device technology used.

All of these factors directly affect the maximum frequency the circuit of the CCA architecture can be operated at, which plays a major role in the achievable performance. As a basis for comparing the maximum throughput we chose the throughput of a worst case image stream, because it provides a true upper bound for the processing time and, therefore, the applicability of the architecture for real-time processing. Depending on connectivity, scan method and the number of scans the worst case image differs; some publications lack the identification of a worst case scenario. These aspects make a direct comparison difficult. For this reason the results of each architecture from Table IX are compared to the architecture proposed in this paper individually.

Comparison to [1]: The architecture of [1] is the most resource efficient architecture reported in the literature to date. The key weakness of [1] is the requirement for two tables for merger management and to translate labels due to aggressive relabelling. This also requires use of two data tables, one for the old labels and one for the new labels. Memory resources scale linearly with image width, therefore, are more critical for scalability. In Section IV-A we have shown that the BRAM resources required for this work are fewer for all image sizes compared to [1]. In this work the maximum throughput is more than three times higher, some of which will be a result from using a newer FPGA. The main advantage of the proposed algorithm is label reuse reducing the memory requirements for storing feature vectors.

Comparison to [10]: In [10] a CCA architecture is presented processing the input image after a transformation to a representation as runs. The architecture of [10] considers 4-connectivity which will give a different result to 8-connectivity used in this paper. Four-connectivity requires fewer comparisons per pixel providing a shorter critical path in the resulting hardware circuit. For an image size of 256×256 the authors state a throughput around 90 megapixels per second by eliminating the additional processing at the end of the image row. For 8-connectivity CCA such a method cannot be found in the literature. The architecture proposed in this paper requires significantly fewer LUTs and registers for an architecture processing the same image size. Another major advantage of the proposed algorithm compared to [10] is the identification and analysis of the worst case image pattern which makes it applicable for real-time processing.

Comparison to [14]/ [19]: The CCA architecture in [14]/ [19] is a basic version of [1]. It provides memory for 256

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCSVT.2015.2450371, IEEE Transactions on Circuits and Systems for Video Technology

TABLE IX
COMPARISON OF SEVERAL CCA HARDWARE ARCHITECTURES.
(A) AREA, (C) COMPONENT COUNT, (FOM) FIRST-ORDER MOMENT, (BB) BOUNDING BOX

Implementation of architecture	Technology	Image size $[pixel]$	Extracted FV	LUTs	Registers	BRAM $[bit]$	f_{max} $[MHz]$	Worst case throughput $[\frac{MPixel}{s}]$
[1]	Virtex 2	640×480	A, C	1757	600	72k	40.64	32.5
[10]	Virtex 2	256×256	A, FOM	4587	3154	234k	95.7	N/A
[14]/ [19]	Spartan II	670×480	A, C	810	286	16k	N/A	N/A
[33]	Stratix	$2k \times 2k$	N/A	1.5k-10k LE		53k-409k	61-72	61-72
[43]	Virtex 4	640×480	N/A	649	641	1142	49.73	$\leq$24.86
This work	Kintex 7	256×256	BB	493	296	108k	185.59	148.47
		UHD8k	BB	818	444	548k	170.53	136.42

labels, i.e. does not cover a worst case scenario. Therefore, a meaningful comparison to this work is not possible.

Comparison to [33]: In [33] a measure for the level of concavity in image components is introduced, therefore the output results for a complex input image differs from the output of the proposed architecture. The authors state their architecture requires between 1.5k and 10k logic elements to process a four megapixel image, where one logic element is equals to a 4-input LUT and a flip-flop, which is more than the architecture in this work requires to process a comparable image size. In [33] all information are obtained by local operations propagating tags to the next image row. In the proposed algorithm the connection between distant component segments is identified by a global equivalence table which allows feature vectors to be extracted for arbitrarily shaped components.

Comparison to [43]: The architecture in [43], based on a two-pass algorithm, requires the complete image to be stored before the labelling process starts, i.e. large images cannot be processed completely on an FPGA due to a lack of sufficient on-chip memory. The two-pass algorithm requires a minimum of 2 clock cycles to process a single pixel. For stream processing an additional buffer is required to store the pixels received while the second pass of the previous image is carried out. The proposed architecture uses a single-pass algorithm which does not require the complete image to be stored and, therefore, requires significantly less memory resources.

CONCLUSION

In this paper a resource-efficient hardware architecture for connected components analysis on an FPGA is presented. Due to its efficient design, it is possible to reduce memory resources by a factor of more than two orders of magnitude compared an implementation of the classical connected component labelling algorithm. Compared to the most memory-efficient state-of-the-art single-pass connected component analysis architectures, 42% or more of the on-chip memory resources are saved depending on the selected feature vector. To achieve this a novel principle to detect finished image objects is used allowing memory resources to be efficiently recycled. On the algorithm level, auxiliary data structures were added to detect image patterns which were not taken into account in the algorithms of several previous hardware architectures. Therefore, arbitrary image patterns are analysed correctly with the proposed architecture. For processing a video stream consisting of worst case patterns, the proposed architecture achieves a throughput of up to 166 megapixels per second. To the best of our knowledge this is the highest throughput of a connected component analysis architecture for 8-connectivity processing one pixel per clock cycle which has been achieved on an FPGA. Further improvements can be obtained by parallel processing of several pixels per clock cycles, e.g. by image slicing. This requires a basic unit to process the individual image slices, which the proposed architecture can be used for, too.

REFERENCES

[1] N. Ma, D. Bailey, and C. Johnston, "Optimised single pass connected components analysis," in *International Conference on Field Programmable Technology (FPT)*, Dec. 2008, pp. 185 –192.

[2] *Recommendation ITU-R BT.2020-1 Parameter values for ultra-high definition television systems for production and international programme exchange*, International Telecommunication Union, Jun. 2014.

[3] J. C. Lasheras, E. Villermaux, and E. J. Hopfinger, "Break-up and atomization of a round water jet by a high-speed annular air jet," *Journal of Fluid Mechanics*, vol. 357, pp. 351–379, Sept. 1998.

[4] J. Kim and T. Chen, "A VLSI architecture for video-object segmentation," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 13, no. 1, pp. 83–96, Jan. 2003.

[5] S. Chan, S. Zhang, J.-F. Wu, H.-J. Tan, J. Ni, and Y. Hung, "On the hardware/software design and implementation of a high definition multiview video surveillance system," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 3, no. 2, pp. 248–262, Jun. 2013.

[6] A. Agrawala and A. Kulkarni, "A sequential approach to the extraction of shape features," *Computer Graphics and Image Processing*, vol. 6, no. 6, pp. 538–557, Dec. 1977.

[7] M. Klaiber, L. Rockstroh, Z. Wang, Y. Baroud, and S. Simon, "A memory-efficient parallel single pass architecture for connected component labeling of streamed images," in *International Conference on Field-Programmable Technology (FPT)*, Dec. 2012, pp. 159–165.

[8] J. F. Eusse, R. Leupers, G. Ascheid, P. Sudowe, B. Leibe, and T. Sadasue, "A flexible ASIP architecture for connected components labeling in embedded vision applications," in *Design, Automation and Test in Europe Conference and Exhibition (DATE)*, Mar. 2014, pp. 1–6.

[9] Z. Yu, L. Claesen, Y. Pan, A. Motten, Y. Wang, and X. Yan, "SoC processor for real-time object labeling in life camera streams with low line level latency," in *IEEE International Symposium on Circuits and Systems (ISCAS)*, Jun. 2014, pp. 345–348.

[10] F. Zhao, H. Lu, and Z. Zhang, "Real-time single-pass connected components analysis algorithm," *EURASIP Journal on Image and Video Processing*, vol. 2013, p. 21, 2013.

[11] L. He, Y. Chao, K. Suzuki, and K. Wu, "Fast connected-component labeling," *Pattern Recognition*, vol. 42, no. 9, pp. 1977–1987, Sep 2009.

[12] L. Lacassagne and B. Zavidovique, "Light speed labeling: efficient connected component labeling on RISC architectures," *Journal of Real-Time Image Processing*, vol. 6, no. 2, pp. 117–135, Jun. 2011.

[13] F. Nina Paravecino and D. Kaeli, "Accelerated connected component labeling using CUDA framework," in *Computer Vision and Graphics*, ser. Lecture Notes in Computer Science, L. Chmielewski, R. Kozera, B.-S. Shin, and K. Wojciechowski, Eds. Springer International Publishing, 2014, vol. 8671, pp. 502–509.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCSVT.2015.2450371, IEEE Transactions on Circuits and Systems for Video Technology

[14] D. Bailey and C. Johnston, "Single pass connected components analysis," in *Proceedings of Image and Vision Computing New Zealand*, 2007, pp. 282–287.

[15] D. Burger, J. R. Goodman, and A. Kägi, "Memory bandwidth limitations of future microprocessors," in *23rd Annual International Symposium on Computer Architecture*, vol. 24, no. 2, Philadelphia, Pennsylvania, May 1996, pp. 78–89.

[16] D. Hackenberg, D. Molka, and W. E. Nagel, "Comparing cache architectures and coherency protocols on x86-64 multicore SMP systems," in *Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO 42. New York, NY, USA: ACM, 2009, pp. 413–422.

[17] *7 Series FPGAs Overview DS180 (v1.16)*, Xilinx, Inc., Oct. 2014.

[18] *Stratix V Device Handbook - Volume 1: Device Interfaces and Integration*, Altera Corporation, Jul. 2014.

[19] C. Johnston and D. Bailey, "FPGA implementation of a single pass connected components algorithm," in *4th IEEE International Symposium on Electronic Design, Test and Applications*, Jan. 2008, pp. 228 –231.

[20] A. Rosenfeld and J. L. Pfaltz, "Sequential operations in digital picture processing," *Journal of the ACM*, vol. 13, pp. 471–494, Oct. 1966.

[21] J. T. Schwartz, M. Sharir, and A. Siegel, "An efficient algorithm for finding connected components in a binary image," Robotics Research Technical Report 38. New York Univ. New York, Tech. Rep., Feb. 1985.

[22] M. B. Dillencourt, H. Samet, and M. Tamminen, "A general approach to connected-component labeling for arbitrary image representations," *Journal of the ACM*, vol. 39, no. 2, pp. 253–280, Apr. 1992.

[23] R. Seidel and M. Sharir, "Top-down analysis of path compression," *SIAM Journal on Computing*, vol. 34, no. 3, pp. 515–525, Mar. 2005.

[24] R. Tarjan and J. van Leeuwen, "Worst-case analysis of set union algorithms," *Journal of the ACM*, vol. Volume 31 Issue 2, pp. 245 – 281, 1984.

[25] V. Khanna, P. Gupta, and C. Hwang, "Finding connected components in digital images by aggressive reuse of labels," *Image and Vision Computing*, vol. 20, no. 8, pp. 557 – 568, 2002.

[26] L. Ni, K. Wong, and D. Yen, "Single pass method for labeling black/white image objects (pipeline processing)," *IBM Tech. Disclosure Bull.;(United States)*, vol. 10, 1984.

[27] A. Abubaker, R. Qahwaji, S. Ipson, and M. Saleh, "One scan connected component labeling technique," in *IEEE International Conference on Signal Processing and Communications. ICSPC.*, Nov. 2007, pp. 1283–1286.

[28] J. De Bock and W. Philips, "Fast and memory efficient 2-D connected components using linked lists of line segments," *IEEE Transactions on Image Processing*, vol. 19, no. 12, pp. 3222–3231, Dec. 2010.

[29] X. Yang, "Design of fast connected components hardware," in *Computer Society Conference on Computer Vision and Pattern Recognition*, 1988, pp. 937–944.

[30] P. Chen, H. Zhao, C. Tao, and H. Sang, "Block-run-based connected component labelling algorithm for GPGPU using shared memory," *Electronics Letters*, vol. 47, no. 24, pp. 1309 –1311, Nov. 2011.

[31] R. Lumia, L. Shapiro, and O. Zuniga, "A new connected components algorithm for virtual memory computers," *Computer Vision, Graphics, and Image Processing*, vol. 22, no. 2, pp. 287 – 300, 1983.

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

[33] Y. Ito and K. Nakano, "Low-latency connected component labeling using an FPGA," *International Journal of Foundations of Computer Science*, vol. 21, no. 03, pp. 405–425, 2010.

[34] R. M. Haralick and L. G. Shapiro, "Glossary of computer vision terms," *Pattern Recognition*, vol. 24, no. 1, pp. 69 – 93, 1991.

[35] D. Knuth, *The Art of Computer Programming: Fundamental algorithms*, ser. The Art of Computer Programming. Addison-Wesley, 1997, vol. 1, pp. 372–373.

[36] J. Hopcroft and J. Ullman, "Set merging algorithms," *SIAM Journal on Computing*, vol. 2, no. 4, pp. 294–303, 1973.

[37] B. Bässler, "Implementation and hardware accelerated verification of a connected component architecture," Master's thesis, University of Stuttgart, 2014.

[38] S. Tasiran and K. Keutzer, "Coverage metrics for functional validation of hardware designs," *Design Test of Computers, IEEE*, vol. 18, no. 4, pp. 36–45, Jul 2001.

[39] P. G. Emma, W. R. Reohr, and M. Meterelliyoz, "Rethinking refresh: Increasing availability and reducing power in DRAM for cache applications," *IEEE Micro*, vol. 28, no. 6, pp. 47–56, 2008.

[40] USC-SIPI, "USC-SIPI image database," 2014. [Online]. Available: http://sipi.usc.edu/database, accessed 2014-07-21.

[41] N. Otsu, "A threshold selection method from gray-level histograms," *IEEE Transactions on Systems, Man and Cybernetics*, vol. 9, no. 1, pp. 62–66, 1979.

[42] *Spartan-6 FPGA Configurable Logic Block User Guide UG384 (v1.1)*, Feb. 2010.

[43] K. Appiah, A. Hunter, P. Dickinson, and J. Owens, "A run-length based connected component algorithm for FPGA implementation," in *International Conference on Field Programmable Technology (FPT)*, Dec. 2008, pp. 177 –184.

Michael J. Klaiber received the diploma degree (Dipl.-Ing.) in Electrical Engineering and Information Technology from the University of Stuttgart, Germany in 2011.
Since 2011 he has pursued a doctorate degree and worked as a research associate at the Institute of Parallel and Distributed System, department for Parallel Systems of University of Stuttgart, Germany.
His research interests include image processing on FPGAs, computer engineering and hardware architectures.

Donald G. Bailey received the B.E. (Hons) degree in Electrical Engineering in 1982, and the PhD degree in Electrical and Electronic Engineering from the University of Canterbury, New Zealand in 1985. From 1985 to 1987, he applied image analysis to the wool and paper industries within New Zealand. From 1987 to 1989 he was a Visiting Research Engineer at University of California at Santa Barbara. Dr Bailey joined Massey University in Palmerston North, New Zealand as Director of the Image Analysis Unit at the end of 1989. He is currently an Associate Professor in the School of Engineering and Advanced Technology, and leader of the Image and Signal Processing Research Group. His primary research interests include applications of image analysis, machine vision, and robot vision. One area of particular interest is the application of FPGAs to implementing image processing algorithms.

Yousef O. Baroud received the B.Sc. degree in Electronics and Communication Engineering in 2006, and M.Sc. in Information Technology (INFOTECH Program) with embedded systems specialisation from University of Stuttgart, Germany in 2011. In 2012 he joined the Institute of Parallel and Distributed Systems, department for Parallel Systems at University of Stuttgart as a research assistant. His research interests include hardware architectures, image processing and data compression.

Sven Simon received the diploma from RWTH Aachen (1992) and the Ph.D. from Technische Universität München (1996) both in Electrical Engineering. In 1996, he joined Siemens AG and Infineon Technologies AG in 1998 focusing on hardware architectures and digital signal processing. In 1998 he became project manager and was nominated for the Infineon Inventors Award in 2000. In 2001, he became professor at Hochschule Bremen, Germany, heading a research group for hardware architectures and sensor systems. In 2007, he became full professor and head of the Parallel Systems Department at the Institute of Parallel and Distributed Systems of the University of Stuttgart. His research interests include parallel algorithms, hardware architectures and sensor systems. He has numerous publications as well as a number of patents.