A Flexible ASIP Architecture for Connected Components Labeling in Embedded Vision Applications

Juan Fernando Eusse*, Rainer Leupers*, Gerd Ascheid *,Patrick Sudowe†,Bastian Leibe†, Tamon Sadasue‡
*Institute for Communication Technologies and Embedded Systems (ICE) †Computer Vision Group
RWTH Aachen University, Germany
{eusse, leupers, ascheid}@ice.rwth-aachen.de {sudowe, leibe}@vision.rwth-aachen.de
‡6th Development Department, Work Solutions Development Division
RICOH Company LTD., Japan
Email: tamon.sadasue@nts.ricoh.co.jp

***Abstract*—Real-time identification of connected regions of pixels in large (e.g. *FullHD*) frames is a mandatory and expensive step in many computer vision applications that are becoming increasingly popular in embedded mobile devices such as smartphones, tablets and head mounted devices. Standard off-the-shelf embedded processors are not yet able to cope with the performance/flexibility trade-offs required by such applications. Therefore, in this work we present an Application Specific Instruction Set Processor (ASIP) tailored to concurrently execute thresholding, connected components labeling and basic feature extraction of image frames. The proposed architecture is capable to cope with frame complexities ranging from *QCIF* to *FullHD* frames with 1 to 4 bytes-per-pixel formats, while achieving an average frame rate of *30 frames-per-second (fps)*. Synthesis was performed for a standard *65nm* CMOS library, obtaining an operating frequency of *350MHz* and *2.1mm^2* area. Moreover, evaluations were conducted both on typical and synthetic data sets, in order to thoroughly assess the achievable performance. Finally, an entire planar-marker based augmented reality application was developed and simulated for the ASIP.**

I. Introduction

Computer vision applications are becoming increasingly popular in embedded devices such as tablets, smartphones and head mounted devices [1]. However, the processing elements integrated in these devices cannot yet provide the adequate efficiency/performance trade-off for the algorithms inherent to such applications [2]. Connected Components Labeling (CCL) [3]–[6] is one of the first steps in many computer vision applications such as marker-based augmented reality [1], video-based driver assistance [7], automated security systems [8] and geographical information services [9]. The algorithm consists on the detection of regions of connected pixels in an image, with the objective of producing a symbolic matrix that identifies each region with an individual label. Each individual region within the symbolic matrix is a potential object of interest. Therefore, the matrix is always post-processed and the features of each region are usually extracted for further use within the specific application [6].

Significant research efforts have been put into reducing the computational complexity of CCL, to enable real time execution (i.e. over *25fps*) of the applications that use it as a first step. Most efforts have focused on algorithmic optimizations, targeting execution either on General Purpose Processors (GPPs) or on Graphic Processing Units (GPUs) [3]–[6], [9]. However, with a few exceptions [6], [7], these algorithms are not optimally designed for architectures typically used in embedded systems.

Several Application Specific Integrated Circuit (ASIC) implementations of CCL have been proposed to increase the *throughput/efficiency* ratio [8], [10], [11]. These implementations are able to achieve processing rates from *70 Mega pixels-per-second* to *4.5 Giga pixels-per-second*, guaranteeing real time execution for most image formats [10]. Nevertheless, such architecture implementations lack the flexibility to adapt to algorithmic changes, require a dedicated frame memory communication infrastructure, and cannot be reused to perform other applications.

Application Specific Instruction Set Processors (ASIPs), are processors tailored for a specific application domain. They have proven to provide the adequate trade-off between customization, flexibility and efficiency, and are widely used in embedded systems [12]. Surprisingly, the approach has not been extensively used for CCL; Schewior et al. [7] being to our knowledge the only work on the subject. Here, a Tensillica LX2 extensible core was augmented with CCL-targeted Custom Instructions for pixel thresholding, image labeling and region feature extraction. They claim to achieve an operating frequency of *373MHz* and an average of *16.4 cycles-per-pixel (cpp)* performance (i.e. *10.9fps* for *FullHD* frames), when performing synthesis for a *90nm* standard library. Nevertheless this work limits the maximum number of detected regions per frame (i.e. potencial identified objects), thus sacrificing flexibility.

In this work we propose a tailored ASIP architecture based on the *division* of the input frame into several *slices*, which are then labeled independently [11], [13]. Our ASIP achieves real-time pixel thresholding, frame labeling and region feature extraction of a wide range of image resolutions and pixel formats, going from *QCIF* up to *FullHD*. In order to constrain the size of the ASIP, the maximum number of label regions per slice is restricted to an upper bound. To cope with this limitation we exploit the Software (SW) programmability of the ASIP, highlighting the scalability and flexibility of an ASIP based connected components labeling solution.

The contributions of this paper are threefold. First, we propose a highly specialized processor architecture that is able to execute thresholding, labeling and feature extraction of *FullHD* frames. Second, we exploit the programability of the ASIP to create a CCL variation that is able to adapt to the complexity of the input frame, without the need of any architectural modification. Third, we perform a comprehensive evaluation of the achievable performance of the ASIP, using publicly available image data sets specifically designed to test the CCL algorithm.

The organization of the paper is as follows. In Section II, the connected components labeling algorithm and its parallel variation are briefly introduced. Section III describes in detail the proposed ASIP. In Section IV a comprehensive study of the performance of the architecture is conducted using publicly available data sets. Finally, section V concludes the paper and presents a brief outlook.

II. Connected Components Labeling (CCL)

A. Algorithm Background

CCL deals with the identification of disjoint blocks of connected pixels within a 2-dimensional $m \times k$ image $\mathcal{I} = [p_{(0,0)}, p_{(0,1)}, \dots, p_{(m-1,k-1)}] \in \mathcal{M}_{m\times k}(\mathbb{P}^v \subset \mathbb{N}^v)$, where $p_{(c,r)} \in \mathbb{P}^v$, $c \in [0, m-1]$, $r \in [0, k-1]$ and $v \in [1, 4]$ depends on the color space. The input image $\mathcal{I}$ is first separated into foreground and background pixels via binary thresholding (TH), whose output is the matrix $\mathcal{B} = TH(\mathcal{I}) = [b_{(0,0)}, \dots, b_{(m-1,k-1)}] \in \mathcal{M}_{m\times k}([0, 1])$. After thresholding, each pixel in every block of connected pixels is identified with a unique value or label $l_{(c,r)} \leq N_{lbl} \in \mathbb{N}$, where N_{lbl} is the number of different CCs in an input image. Generated labels are stored within a symbolic $m\times k$ matrix (label matrix) $\mathcal{L} = [l_{(0,0)}, l_{(0,1)}, \dots, l_{(m-1,k-1)}]$. Classical CCL algorithms [3]–[6] perform a *raster scan* of the input image $\mathcal{I}$ and generate the binary matrix $\mathcal{B}$ (Fig. 1 (a)) on the fly. Label $l_{(c,r)} = CCL(p_{(c,r)})$ is assigned according to the thresholded value of the current pixel, and to a *8-connected* neighbour mask (Eq. 1), used to access already calculated predecessor labels, as shown in Fig. 1 (c).

$$NB_{8-mask}(c, r) = [l_{(c-1,r-1)}, l_{(c,r-1)}, l_{(c+1,r+1)}, \\ l_{(c-1,r)}] \in \mathcal{L} \tag{1}$$

Due to the local neighborhood, components may coalesce in the process (Fig. 1(b)), making two labels equal. Note, that this aspect leads to complications in implementation and architecture design. Therefore, classical algorithms utilize temporary data structures or tables to record equivalences between labels ($\mathcal{EQ}_{image} = [eq_0, eq_1, \dots, eq_{N_{lbl}}]$), which are later resolved (Eq. 2) to produce a consistent label matrix, as seen in Fig. 1 (d). During CCL, a vector $\mathcal{F} = [f_0, f_1, \dots, f_{N_{lbl}}]$ (see Sec. III-A) of properties for each connected region (e.g. clipping

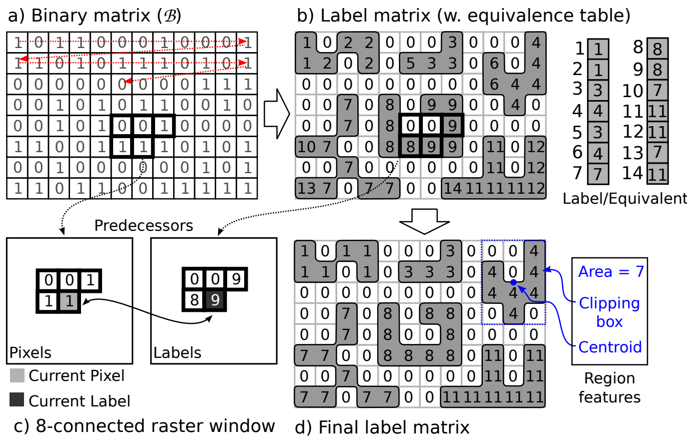

Fig. 1. Classical CCL algorithms a) Input Thresholded Image b) Imperfect label matrix and equivalence list c) Raster window applied in 8-connected CCL d) Final label matrix with resolved equivalences

box, area and centroid) is calculated. This facilitates subsequent processing steps in the application domain (Fig. 1 (d)). More detailed descriptions of the CCL algorithm can be found in [3]–[6], [9].

$$l_i(c_i, r_i) \equiv l_j(c_j, r_j) \leftrightarrow EQ_{image}(l_i) \equiv EQ_{image}(l_j) \tag{2}$$

B. Parallel Image Slicing Algorithm

Originally, CCL algorithm design focused on achieving maximum performance on existing hardware devices such as GPPs and GPUs. To facilitate image processing on mobile devices, both algorithmic performance and energy efficiency are of concern. Therefore parallel algorithms, intended to be implemented in dedicated architectures [10], [11], [13], have been created. Of these, a popular approach is the *divide-and-conquer* slicing algorithm [13] shown in Fig. 2 (a).

This algorithm horizontally partitions $\mathcal{B}$ into a set of Q slices $\mathcal{S} = \{Sl_0, Sl_1, \dots, Sl_{Q-1}\}$, where $Sl_i = [b_{(0,i*k/Q)}, \dots, b_{(m-1,((i+1)*k/Q)-1)}]$ is a submatrix of $\mathcal{B}$. After division, CCL is used to independently label each slice. For every slice Sl_i, a Label-Matrix/Equivalence-Table (LM/EQ) pair is generated by CCL, $(LM_i, EQ_i) \xleftarrow{CCL} Sl_i$. After independent labeling, a merging operation is executed between the first two adjacent pairs (LM_i and LM_{i+1}) of neigbouring slices (Sl_i and Sl_{i+1}) to restore any possible label region connections lost due to slicing. The shaded circles of Fig. 2 (a) show the labels of the intermediate LMs used during merging. During merging an offset is added to the Sl_{i+1} equivalence table (EQ_{i+1}) to avoid label collisions. EQ_{i+1} is also updated with newly discovered equivalences, and EQ_i and EQ_{i+1} are added to the final equivalence table ($\mathcal{EQ}_{image}$). Merging is repeated *Q-1* times, until $\mathcal{EQ}_{image}$ is complete.

III. Architectural Design

Our main design goal was to create a customized ASIP based on the slicing approach, while supporting an average performance of *30fps* for *FullHD* frame sizes. However, as

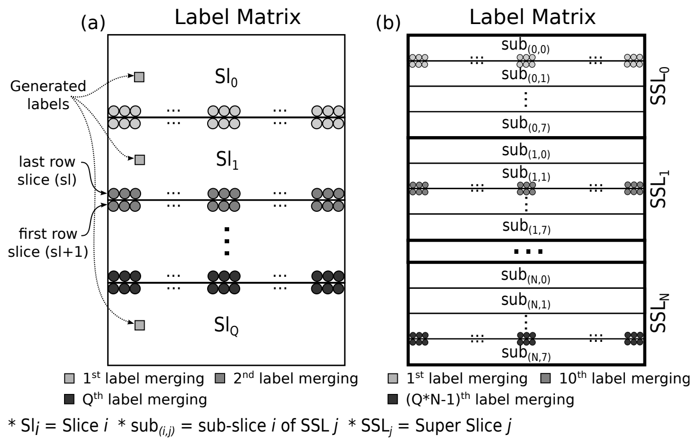

Fig. 2. Labeling slicing concept a) Fixed-slicing used in ASIC implementation (Section II-B) b) Extended SW supported super-slicing (Section III-B)

our design is targeted for integration into a Multi-Processor embedded vision platform, we considered the following additional design constraints:

- Perform CCL over frames stored in system memory.
- Support several image pixel formats and memory alignments.
- Be adaptable to the operating conditions of more than one application [1], [4], [7]–[9].
- Communicate via standard interfaces (e.g. 32bit buses) with the rest of the platform.
- Provide flexibility to support full applications, or to be reused by the platform as an extra-processing resource.

The ASIP is therefore constrained to execute all read and write operations to system memory sequentially. This means that a slicing implementation [10], [11], with Q pixels being labeled in parallel, is not easily attainable. To overcome this, a three step strategy was followed. First, system memory accesses are serialized via dedicated address generation logic. Second, thresholding, labeling and feature extraction are performed concurrently by custom logic for Q pixels ($p_{(c,r)} \in \mathcal{I}$, $c = 0, \ldots, m-1$, $r = i * k/Q, \ldots, (i+1) * k/Q - 1$) of each slice ($Sl_i$, $i = 0, 1, \ldots, Q$). Finally, intermediate CCL results for each slice (region features and equivalence tables) are stored in dedicated local scratchpads until slice labeling is completed, to avoid accesses to system memory. The merging process described in Section II-B is implemented by sequential software.

A. Processor Architecture

Our customized architecture is a collection of parameterizable CCL Functional Units (CCL-FUs) integrated within the Synopsys Processor Designer RISC (PD_RISC) [14]. The PD_RISC is a load-store 6-stage pipeline architecture described using the Language for Instruction Set Architectures (LISA) [12], and provided as an example with Synopsys tools. A total of $Q = 8$ CCL-FUs were instantiated into the PD_RISC, enabling us to speedup the labeling of up to 8 slices ($\mathcal{S} = \{Sl_0, \ldots, Sl_7\}$). The main parameters of a CCL-FU are related to the maximum amount of supported labels per slice (max_{lbl}) and to the input image size ($m \times k$), which dictate both the size for local memories and datapath. Through application profiling for our target input data (video streams from a smartphone camera), we fixed $max_{lbl} = 512$ ($lbl_{bit} = 9$), and a maximum frame size of 2048×2048 pixels. Fig. 3 shows a generalized block diagram of the customized logic of the ASIP.

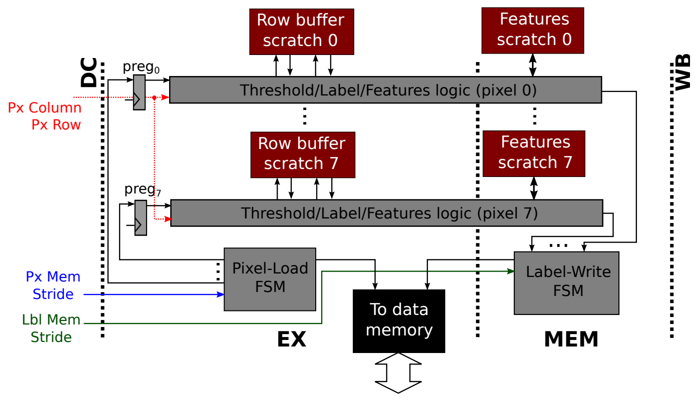

Fig. 3. 8-way CCL architecture customizations added to the EX and MEM pipeline estages of the PD_RISC [14]

The operation of the CCL-FUs is distributed among the execute (EX) and memory access (MEM) stages of the pipeline (Figs. 3 and 4). Two local scratchpad memories are used within each CCL-FU:

- **Row Buffer Scratchpad (RBS):** Logically banked memory that stores the history of the generated labels for one row of its corresponding slice (Sl_i). In a single clock cycle, the labels for both the previous and current rows must be read and written concurrently. Therefore, the RBS is implemented as a double ported memory with separated read and write interfaces (both data and addressing). Each memory word stores the labels for four different pixels, with the memory size calculated as $size_{rbs} = lbl_{bit} * k$. Its indexing is handled by the label row addressing logic unit shown in Fig. 4.
- **Features Scratchpad (FS):** Each word of this memory stores a vector with the area, clipping box, and accumulated row and column indexes for each discovered label region ($f_i = [A_i, start_{rowi}, end_{rowi}, start_{coli}, end_{coli}, \sum row_i, \sum col_i] \in \mathcal{F}, i = 1, \ldots, N_{lbl}$). The contents of this memory are modified automatically by the features update logic (Fig. 4) of the CCL-FU during the EX and MEM stages of the pipeline, in a *read-modify-write* fashion. It has a single read/write port. Furthermore, its size is related to the image dimensions (m and k) and the number of supported labels per slice (max_{lbl}), and can be calculated using Eq. (3).

$$size_{fs} = (\lceil \log_2 (m * k) \rceil + 2 * \lceil \log_2 (m) \rceil + 2 * \lceil \log_2 (k/Q) \rceil + \left\lceil \log_2 \left(m * \sum_{j=0}^{k/Q} j \right) \right\rceil + \left\lceil \log_2 \left(k/Q * \sum_{j=0}^{m} j \right) \right\rceil) * max_{lbl} \quad (3)$$

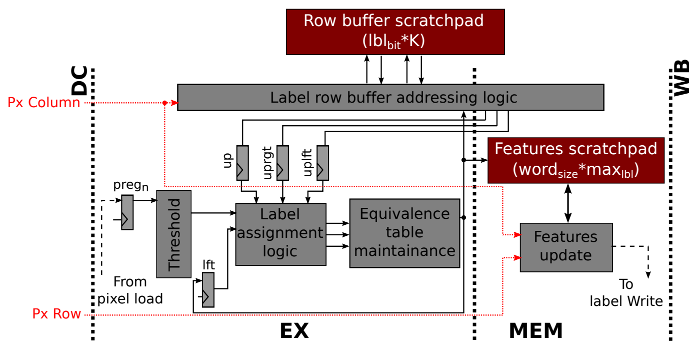

Fig. 4. CCL custom logic for slice (SL_i)

The Pixel-Load Finite-State Machine (FSM) in Fig. 3 represents the aforementioned serialization of the pixel loading from system memory. This process is triggered via a Custom Instruction (CI) and is in charge of loading 8 pixels from data memory and writing them into the $preg_n$ state register. The unit receives as parameters the base address for pixel $p_{(c,r_0)}$ in slice (SL_0), and a memory access stride $pixel_{stride} = k * m/Q * b$,where $b = size_{bytes}(p_{(c,r)})$ is the pixel format size in bytes *(1-4)*. Using the stride, the unit is able to address system memory to load pixels $(p_{(c,r_i)}, i = 0, \ldots, 7)$ for each slice (SL_i). Read accesses to row buffer scratch memories are also initiated by this stage, hiding their latency.

After loading, a subsequent CI has to be called in order to activate the CCL-FU logic shown in Fig. 4. The EX stage of CCL-FU for Sl_i thresholds its corresponding pixel register $(preg_i = p_{(c,r_i)})$, to get its binary value. A temporary value for the pixel label is generated by the assignment logic using the binary pixel $(b_{(c,r_i)})$), previous label values loaded from RBS and the previously calculated label for $p_{(c-1,r_i)}$, which is stored in the *lft* state register (Fig. 4). The storage and consistency of the equivalence table (EQ_i) are handled by its maintenance block, which generates the final label $(l_{(c,r_i)})$ for the current pixel. In this stage, the read operation for the FS memory is also triggered using $l_{(c,r_i)}$ as the address.

During the MEM stage of the CCL-FU, the word coming from its feature memory is split into its fields (A, $\sum_{row}$, $\sum_{col}$, $start_{row}$, end_{row}, $start_{col}$, and end_{col}). Then, the update logic calculates their new values and triggers the corresponding FS memory write operation. During this stage, the labels for each slice are serially written back to system memory by the write FSM (Fig. 3). To do so, the CCL-FU receives as input parameters the base address for label $(l_{(c,r_0)})$ of slice Sl_0 and a label access stride $label_{stride} = k*m/Q*q$, where $q = size_{bytes}(l_{(c,r)})$. The CIs must be called for every pixel in one slice $(m * k/Q\ times)$, which effectively labels the whole image.

At this point, the equivalence tables (EQ_i) and the region features (f_i) for each slice are stored either inside the equivalence maintenance logic or in the feature scratchpad for Sl_i. These tables are first read from the custom HW blocks of the processor via CIs, and then the merging steps described in Section II-B are performed in SW.

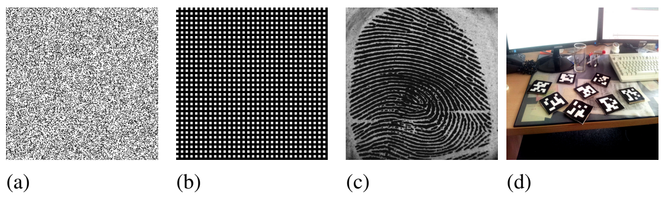

Fig. 5. (a,b) Synthetic images; (c,d) Natural images

B. Exploiting the ASIP Inherent Flexibility

ASIPs are customized according to the input data characteristics of the specific application domain in mind (e.g. driver assistance) [7]. In such a constrained architecture, it is possible that the capacities of the customized hardware are overwhelmed by highly unusual input data, or that the support of a second application domain with different characteristics (i.e. bigger frame size, more complex data, lower/higher required frame rate) is required. For such a case, we exploit the flexibility of the ASIP approach by extending the slicing concept [13]. In our implementation, we divide the input frame into a set of N super slices $\mathcal{SSL} = \{SSL_0, SSL_1, \ldots, SSL_{N-1}\}$, each of size $size(SSL_i) = m/N$, and composed by 8 sub-slices $(sub_{(i,j)}, i = 0, 1, \ldots, N, j = 0, 1, \ldots, 7)$. The labeling of each individual sub-slice is handled by customized hardware, and a subsequent merging with $Q * N - 1$ steps is performed in software. In the case that super slicing is needed, such flexibility comes with an associated performance penalty due to the extra merging steps, which will be analyzed in section IV-C. Fig. 2 (b) illustrates the concept of super slicing.

IV. Results and Discussion

A. Experimental Setup

Regardless of the specific algorithm, CCL is strongly data dependent, and its algorithmic performance is usually evaluated using extensive sets of input frames. Among these, two main categories exist. Synthetic data sets are artificially generated frames targeted to stress the algorithm. Natural data sets on the other hand, are images taken with an electronic sensor, and represent more realistic input scenarios [3]–[6]. Fig. 5 shows several examples of both types of input data.

State of the art works concerning ASIP accelerated CCL lack a comprehensive evaluation of the achieved performance regarding varying input data [7], [8], [10], [11]. Therefore, for our evaluation we chose publicly available data sets that have both synthetic (6000) images [6], and natural (5398) images [15]–[18]. The synthetic image set consists of:

- **Homothety:** Set of 6 images (1024 × 1024) having a variable number (16, 64, 256, 1024, 4096, 16384) of square shaped foreground regions.
- **Random Percolation:** 1000 randomly generated images (1024 × 1024) with foreground pixel densities from 0 to 1.0.
- **Morphology:** 4000 images (1024 × 1024) generated based on the previous set, with a determined *3x3* mor-

phological operator applied (opening, closing, erosion, dilation).

- **Closing 1920:** Extra image set composed with 1000 random percolation images (1920 × 1080) after the *3x3* closing operator has been applied.

All the images have been input to a Cycle Accurate (CA) simulator of our processor, and the performance in terms of spent processor *cycles-per-pixel (cpp)* has been obtained. It should be noted that all our simulation results include the time spent within the computation of label region features, and that the obtained *cpp* would decrease if they are neither read from FS memories nor subjected to merging. Furthermore, based on simulation results we have chosen a subset of the images that exhibit the *worst/average/best* behavior. This set has been input to a pure SW implementation of the algorithm both running on the base PD_RISC [14] and on a Texas Instruments *TMS320C64x+* DSP, and a performance comparison has been made.

B. Implementation Results

All CCL-oriented architectural customizations were implemented using LISA [12], and were added to the base Instruction Set Architecture (ISA) for the PD_RISC processor. As a result we added 8 new instructions to the base ISA, together with the custom logic depicted in Figs. 3 and 4. HDL code for the architecture was generated using Synopsys Processor Designer. Moreover, post-synthesis models for all local scratchpad memories were generated for a *65nm CMOS standard library*. According to our design parameters $(9\ bits/slice, 2048 \times 2048\ bits/frame, 8\ slices)$, the total size for RBS memories is *18kB* and for FS memories is *62kB*. Synthesis was performed using worst case conditions for the chosen *65nm* CMOS library, and area results are shown in Tab. I. Our ASIP is able to achieve a maximum clock frequency of *350MHz@65nm*, with an overall area of *2.1mm*^{2}. As shown in Tab. I, only 2.7% of the total area is used by the original PD_RISC, 50.5% by the local memories and the remaining 40.9% by the custom CCL logic. Although a complete energy consumption evaluation is relevant, only an initial estimate of the power consumption for the synthesized VHDL description of our ASIP has been obtained. Through it we obtained an estimated power consumption of 228*mW*, which gives us a clear idea of the achievable energy benefits when using the proposed implementation.

C. Performance Evaluation

We chose *cycles-per-pixel (cpp)* as the performance measurement metric given that it represents the suitability of an architecture (GPP, GPU, ASIC or ASIP) for a certain CCL implementation, while being independent of the architecture clock frequency. Given the *cpp* achieved by an architecture with frequency $arch_f$, its achievable *frames-per-second (fps)* rate for a determined frame size $(m \times k)$ can be calculated as $fps_{arch} = arch_f/(m * k * cpp)$. Tab. II shows the obtained simulation results. In the table we show the achieved minimum, average and maximum *cpp* for each test set. We also

TABLE I
SYNTHESIS RESULTS @ *65nm* CMOS

Logic Component	**Pipeline Stage**	**Instance Qty**	**Area (*μm*^{2})**	**Total (*μm*^{2})**	**%**
PD_RISC Base ISA	All	1	50,364	50,364	2.7
Pixel load FSM	EX	1	1,653	1,653	0.09
Label write FSM	MEM	1	490	490	0.03
Threshold & Labeling	EX	8	9,464	75,711	4.0
Equivalence table maintenance	EX	8	96,236	769,890	40.9
FS update	MEM	8	2,135	17,080	0.9
RBS addressing	EX/MEM	8	1,765	14,120	0.8
FS addressing	MEM	8	115	920	0.05
RBS Memory	NA	8	58,000	464,000	24.6
FS Memory	NA	8	61,000	488,000	25.9
Total Area (*μm*^{2})					1,882,228

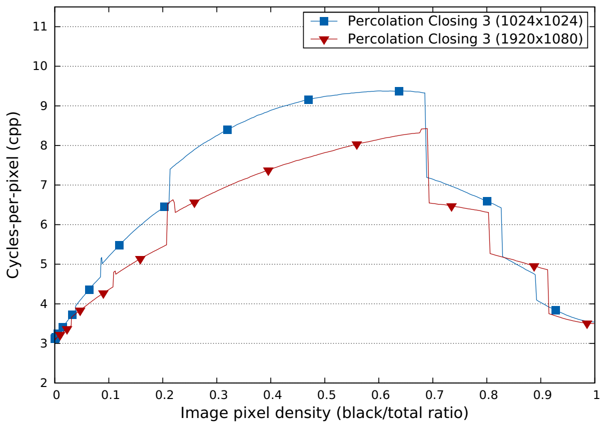

Fig. 6. *cpp vs. foreground pixel density* for the *Percolation Closing 3* data sets

calculated the standard deviation, which shows how sensitive the architecture is to changes in image complexity. We can see from the table that for all test sets we achieved the target *fps* in the best and average cases. In the worst case, we get from *26fps* to *5fps*, depending on the input image.

The aforementioned reduction in performance particularly appears in images that must be heavily super sliced (32 SSLs) due to a high appearance rate of label regions, or in images in which many regions have to be merged. The overhead of performing merging in a pure SW implementation can be seen in Fig. 6, which shows the obtained *cpp* for two of the synthetic data sets. The discontinuties shown in each plot correspond to the increase or decrease of the number of super slices. By examining Fig. 6, it becomes clear that the architecture benefits from wide input images, as data parallelism was exploited *column-wise*.

As expected, our ASIP outperforms the base PD_RISC by a factor that ranges between **5** and **359**. Furthermore, we are between **6x** and **38x** more time efficient than the *TMS320C64x+* DSP when executing CCL. Note that for images close to those we intend to process with the ASIP (Flickr data set), we get an average *speedup* of **33x** for the PD_RISC and of **11x** for the TI DSP. Tab. III shows the obtained *cpp* for each one of the evaluated architectures. Finally, to prove the flexibility of the processor, we developed a complete Planar-

TABLE II
OBTAINED *cycles-per-pixel (cpp)* FOR THE CHOSEN INPUT DATA SETS

Data Set	***Size***	***# Images***	***min***		***avg***		***max***		***std. deviation(cpp)***
			lbl	***cpp***	***lbl***	***cpp***	***lbl***	***cpp***	
Homothety (paving) [6]	1024x1024	6	16	3.14	3,640	4.10	16,384	7.5	1.72
Random Percolation [6]	1024x1024	1,000	1	3.32	23,231	16.39	78,609	28.95	10.36
Percolation Closing 3 [6]	1024x1024	1,000	1	3.11	3,234	4.99	11,963	9.37	2.19
Percolation Closing 3 (1920)	1920x1080	1,000	1	3.09	3,460	4.53	12,710	8.42	1.71
Percolation Opening 3 [6]	1024x1024	1,000	1	3.31	3,381	5.04	30,363	13.47	2.8
Percolation Dilation 3 [6]	1024x1024	1,000	1	3.11	6,591	5.58	22,385	13.41	2.84
Percolation Erosion 3 [6]	1024x1024	1,000	1	3.31	877	4.38	14,156	9.81	1.79
Finger [17]	448x478	80	52	3.61	549	4.68	1,855	6.49	0.61
Medical [18]	2048x2048	248	387	3.21	1,007	3.52	2,976	4.58	0.27
Textures [15]	1024x1024	64	2	3.12	2,578	5.41	14,336	15.59	2.08
Flickr [16]	512x512	5,000	1	3.30	299.92	4.32	4,019	12.29	0.66

Marker detection augmented reality application using the OpenCV and ARToolKitPlus C++ image processing libraries. The CCL routines inside ARToolKitPlus were modified to use our custom instructions, the SW libraries were compiled using an LLVM-based C++ compiler provided by Synopsys and the generated binary was simulated. For the application, we were able to achieve an average of *54fps* for 320x240 frames and up to *24fps* for 640x480 frames.

TABLE III
COMPARISON OF THE OBTAINED *cpp* FOR THE PD_RISC-BASE, THE TMS320C64X+ AND THE PROPOSED ASIP

		Ours	**PD_RISC Base (cpp)**	**Gain**	**TMS320C64x (cpp)**	**Gain**
Homothety	*min*	3.14	22.0	7.01	39.76	12.66
	avg	4.11	26.28	6.39	42.70	10.38
	max	7.54	40.88	5.42	50.60	**6.71**
Random Percolation	*min*	3.31	17.16	**5.18**	25.59	7.73
	avg	16.39	3,524	215	394.42	24.06
	max	28.9	10,384	**359**	1,107.53	**38.32**
Finger	*min*	3.61	21.8	6.04	35.81	9.2
	avg	5.03	84.07	16.71	53.79	10.69
	max	6.49	167.48	25.8	64.87	10.0
Textures	*min*	3.12	19.75	6.33	30.86	9.89
	avg	7.5	493.69	65.82	106.12	14.15
	max	15.59	2,219.36	142.35	303.20	19.45
Flickr	*min*	3.3	22.38	6.78	33.71	10.21
	avg	6.49	215.3	33.17	75.25	11.60
	max	12.29	1,080.14	87.88	158.56	12.9

V. CONCLUSIONS

In this work we applied the ASIP paradigm to the design of an architecture tailored for the efficient labeling of connected components, widely used in computer vision applications. The proposed architecture/algorithm pair is able to achieve up to *45/30/5 frames-per-second* in the *best/average/worst* case, when labeling *FullHD* images. The ASIP was synthesized for a *65nm* standard CMOS library, occupying an area of *2.1mm*^{2} with a frequency of up to *350MHz*. The flexibility of the ASIP enabled us to efficiently support complex images within a constrained processor architecture, and to implement an entire augmented reality application within the ASIP. For our future work we will focus on further optimization of the processor architecture to achieve higher performance and on the power consumption evaluation of the architecture, to fully determine its suitability for integration into consumer devices.

ACKNOWLEDGMENT

This work has been supported by the UMIC Research Centre, RWTH Aachen University (www.umic.rwth-aachen.de).

REFERENCES

[1] T. Vriends, "Evaluation of High Level Synthesis for the Implementation of Marker Detection on FPGA," Master's thesis, Eindhoven University of Technology, 2011.

[2] H. Liao, M. Asri, T. Isshiki, D. Li, and H. Kunieda, "A Reconfigurable High Performance ASIP Engine for Image Signal Processing," in *Parallel and Distributed Processing Symposium Workshops PhD Forum (IPDPSW), 2012 IEEE 26th International*, 2012, pp. 368–375.

[3] L. He, Y. Chao, K. Suzuki, and K. Wu, "Fast Connected-Component Labeling ," *Pattern Recognition*, vol. 42, no. 9, pp. 1977 – 1987, 2009.

[4] C. Grana, D. Borghesani, and R. Cucchiara, "Optimized Block-Based Connected Components Labeling With Decision Trees," *Image Processing, IEEE Transactions on*, vol. 19, no. 6, pp. 1596–1609, 2010.

[5] F. I. Y. Elnagahy, "Connected Components Algorithm Based on Span Tracking," *Canandian Journal on Image Processing and Computer Vision*, vol. 2, no. 7, pp. 72–81, 2011.

[6] 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, 2011.

[7] G. Schewior, H. Flatt, C. Dolar, C. Banz, and H. Blume, "A Hardware Accelerated Configurable ASIP Architecture for Embedded Real-Time Video-Based Driver Assistance Applications," in *2011 International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation, SAMOS XI*. IEEE, 2011, pp. 209–216.

[8] K. Appiah, A. Hunter, P. Dickinson, and H. Meng, "Accelerated Hardware Video Object Segmentation: From Foreground Detection to Connected Components Labeling," *Comput. Vis. Image Underst.*, vol. 114, no. 11, pp. 1282–1291, Nov. 2010.

[9] P. Netzel and T. F. Stepinski, "Connected components labeling for giga-cell multi-categorical rasters," *Computers & Geosciences*, vol. 59, no. 0, pp. 24 – 30, 2013.

[10] 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 *Field-Programmable Technology (FPT), 2012 International Conference on*, 2012, pp. 159–165.

[11] C.-Y. Lin, S.-Y. Li, and T.-H. Tsai, "A Scalable Parallel Hardware Architecture for Connected Component Labeling," in *Image Processing (ICIP), 2010 17th IEEE International Conference on*, 2010.

[12] P. Ienne and R. Leupers, *Customizable Embedded Processors: Design Technologies and Applications*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2007.

[13] V. Kumar, K. Irick, A. Maashri, and V. Narayanan, "A Scalable Bandwidth-Aware Architecture for Connected Component Labeling," in *VLSI 2010 Annual Symposium*, ser. Lecture Notes in Electrical Engineering. Springer Netherlands, 2011, vol. 105, pp. 133–149.

[14] Synopsys Inc. Synopsys Processor Designer. [Online]. Available: http://www.synopsys.com/Systems/BlockDesign/ProcessorDev/

[15] The USC-SIPI Image Database: Textures. [Online]. Available: http://sipi.usc.edu/database/database.php?volume=textures

[16] M. J. Huiskes and M. S. Lew, "The MIR Flickr Retrieval Evaluation," in *MIR '08: Proceedings of the 2008 ACM International Conference on Multimedia Information Retrieval*. New York, NY, USA: ACM, 2008.

[17] FVC2000 Fingerprint Verification Database: DB3. [Online]. Available: http://bias.csr.unibo.it/fvc2000/databases.asp

[18] The Standard Digital Image Database. [Online]. Available: http://www.jsrt.or.jp/jsrt-db/eng.php