Accelerated hardware object extraction and labelling: from object segmentation to connected components labelling

Kofi Appiah[a], Andrew Hunter[a], Hongying Meng[a], Patrick Dickinson[a]

[a]*Department of Computing and Informatics*
University of Lincoln
Brayford Campus
LN6 7TS

Abstract

This paper demonstrates the use of a single-chip FPGA for the segmentation of moving objects in a video sequence. The system maintains highly accurate background models, and integrates the detection of foreground pixels with the conversion into labelled objects using a connected component labelling algorithm. The background models are based on 24-bit RGB values and 8-bit greyscale intensity values. A multimodal background differencing algorithm is presented, using a single FPGA chip and four blocks of RAM. The real-time connected component labelling algorithm, also designed for FPGA implementation, has efficiently been integrated with the pixel level background subtraction to extract pixels of a moving object as a single blob. The connected component algorithm, run-length encodes the binary image output of the background subtraction, and performs connected component analysis on this representation. The run-length encoding, together with other parts of the algorithm, is performed in parallel; sequential operations are minimized as the number of run-lengths are typically less than the number of pixels.

Key words: Background differencing; Image Segmentation; Connected Component labelling; Object extraction; FPGA.

1. Introduction

Detection and tracking of objects in real-time using a stationary camera is an active field of research in the vision community with a wide range

of applications in the fields of monitoring and surveillance, smart rooms, vehicle tracking, biological image analysis and video compression. The first stage in processing for many video-based applications is the segmentation of (usually) moving objects with significant difference in colour and shape from the background, a very basic yet computationally expensive task for real-time purposes. The second stage is the connected-component labelling of the binary image; a highly sequential operation. This is then followed by extracting features of the moving object(s) to support tracking; for example, the centroid, colour histogram, width and height.

Using a Field Programmable Gate Array (FPGA) as an image accelerator, we demonstrate in this paper how real-time extraction of moving objects from image sequences taken under variable lighting conditions can be achieved efficiently on an FPGA architecture. The implementation has a number of uses in embedded systems and automated visual surveillance systems. Real-time image processing is difficult to achieve on a serial processor, due to the movement of large data sets and complex operations that need to be performed on the image [15]. Advances in semiconductor technology makes it possible to design complete embedded System-on-Chip (SoC) by combining sensor, signal processing and memory onto a single substrate [21]. New embedded vision systems have emerged as a result of this level of integration and are likely to proliferate in the coming years. The aim of such computer vision systems is to scan scenes and make judgements that remove or decrease the need for human observers – thus the need to develop imaging functions with performance comparable to trained human operators [28].

Field Programmable Gate Arrays (FPGAs), have been available for over two decades. Recent increases in programmable fabric density have made this an appealing platform for accelerating computer vision and image processing algorithms. The potential uses of FPGAs in areas like medical image processing, computational fluid dynamics, target recognition, embedded vision systems, gesture recognition and automotive infotainment have been demonstrated in [5, 9, 17, 18, 21]. Digital Image processing or computer vision algorithms can be broken down into three major stages [14]: early processing, implemented by local pixel-level functions; intermediate processing, which includes segmentation, motion estimation and feature extraction; and late processing, including interpretation and using statistical and artificial intelligence algorithms[1]. Typically algorithm sophistication is concentrated in the later stages, but processing demands dominate in the early stages [32].

In this paper, we preset the integration of two basic image processing al-

gorithms: background differencing (background subtraction) and connected component labelling, designed specifically for efficient FPGA implementation. Published background differencing algorithms tend to focus on robustness in varying lighting conditions, and on shadow extraction [1], while connected component labelling algorithms tend to focus on the resolution of equivalence table[19]. An integrated system, capable of extracting labelled regions of similar pixels (of a moving object) from real-time video sequence using FPGA is presented. The gate-rich capabilities of FPGAs offer the potential to efficiently implement both algorithms on a single chip for real time processing.

2. Binary Foreground Extraction

Image segmentation, defined as the grouping of pixels that are coherent in space, range, and time domain, is usually the first stage in processing for many video applications. The input image usually defines an algorithm suitable for segmentation. Comaniciu *et al* [7] presents a technique for segmenting a video frame into representative blobs detected in the spatial and colour domains. Similarly, Boykov *et al* in [6] demonstrates a general-purpose segmentation method for extracting objects using the s/t graph cuts, with reported success in segmenting photos, video and medical images. Fuzzy c-means algorithms based on the idea of clustering pixels with similar characteristics have successfully been used in segmentation [32]. Zhou *et al* [32, 31, 30] introduced the mean shift based fuzzy c-means algorithm, which addresses the computational requirement of the original fuzzy c-means algorithm with reported improved segmentation. Where the input is a video sequence and the camera is stationary, a natural approach is to model the background and detect foreground object by differencing the current frame with the background. A wide and increasing variety of techniques for background modelling have been described; a good comparison is given by Gutchess *et al* [11].

The most popular method is unimodal background modelling, in which a single value is used to represent a pixel, which has been widely used due to its relatively low computational cost and memory requirements [13, 29]. This technique gives a poor results when used in modelling non-stationary background scenarios like waving trees, rain and snow. A more powerful alternative is to use a multimodal background representation, which uses more than one process (mostly mixture of Gaussians) to represent a single

background pixels. However, the computational demands make multimodal techniques unpopular for real-time purposes; there are also disadvantages in multimodal techniques [10, 26, 29] including the *blending effect*, which causes a pixel to have an intensity value which has never occurred at that position (a side-effect of the smoothing used in these techniques). Other techniques rely heavily on the assumption that the most frequent intensity value during the training period represents the background. This assumption may well be false, causing the output to have a large error level.

2.1. Grimson's Algorithm

Grimson *et al* [26, 27] introduced a multimodal approach, modelling the values of each pixel as a mixture of Gaussians (MoG). The background is modelled with the most persistent intensity values. The algorithm has two variants, colour and gray-scale: in this paper, we concentrate on the gray-scale version. The probability of observing the current pixel value is given as:

$$P(X_t) = \sum_{i=1}^{k} \omega_{i,t} \eta(X_t, \mu_{i,t}, \sigma_{i,t}) \tag{1}$$

Where $\mu_{i,t}$, $\sigma_{i,t}$ and $\omega_{i,t}$ are the respective mean, standard deviation and weight parameters of the i^{th} Gaussians component of pixel X at time t. η is a Gaussian probability density function

$$\eta(X_t, \mu_{i,t}, \sigma_{i,t}) = \frac{1}{\sigma_{i,t}\sqrt{2\pi}} e^{\frac{(X_t - \mu_{i,t})^2}{2\sigma_{i,t}^2}} \tag{2}$$

A new pixel value is generally represented by one of the major components of the mixture model and used to update the model. For every new pixel value, X_t, a check is conducted to match it with one of the K Gaussian distributions. A match is found when X_t is within 2.5 standard deviation of a distribution. If none of the K distributions match X_t, the least weighed distribution is replaced with a new distribution having X_t as mean, high variance and very low weight. The update equations are as follows:

$$w_{i,t} = w_{i,t-1} + \alpha(m_{i,t} - w_{i,t-1}) \tag{3}$$

where α is the learning rate and

$$m_{i,t} = \begin{cases} 1 & \text{if there is a match} \\ 0 & \text{otherwise} \end{cases} \tag{4}$$

$$\mu_t = \mu_{t-1} - \rho(X_t - \mu_t) \tag{5}$$

$$\sigma_t^2 = (1-\rho)\sigma_{t-1}^2 + \rho(X_t - \mu_t)^T(X_t - \mu_t) \tag{6}$$

Only the matched distribution will have its mean and variance updated, all others remain unchanged. For

$$\rho = \alpha\eta(X_t|\mu_t, \sigma_t) \tag{7}$$

The first B distributions (ordered by ω_k) are used as a model of the background, where

$$B = arg_b\ min(\sum_{k=1}^{b} \omega_k > T). \tag{8}$$

The threshold T is a measure of the minimum portion of the data that should be accounted for by the background.

2.2. Temporal Low-Pass filtering Algorithm

Aleksej [20] introduced a method to avoid false alarms due to illumination, using a temporal filter to update the background model, while a global threshold value T was used to extract target regions. The background update he used is of the form

$$B(k,l,n) = \frac{(p-c)}{p}B(k,l,n-1) + \frac{c}{p}I(k,l,n)$$

where c is the number of consecutive frames during which a change is observed and is reset to zero each time the new value becomes part of the background; p is the adaptation time or insertion delay constant. The moving target is extracted on a pixel level with the following relation:

$$f(k,l,n) = \begin{cases} 1 & |I(k,l,n) - B(k,l,n)| > L \\ 0 & \text{otherwise} \end{cases} \tag{9}$$

where *f(k,l,n)*, *B(k,l,n)* and *I(k,l,n)* are the respective foreground, background and greyscale intensity value of pixel *(k,l)* for the n^{th} frame, and L is the global threshold value.

The low-pass filtering algorithm is attractive for two reasons. First it is very simple and hence updating the background information is computationally cheap and memory consumption is minimal. The use of single global threshold value as well as a single mode makes it unattractive for scenes with

varying lighting intensities. In contrast, Grimson's algorithm [26] is robust to outdoor environments where lighting intensity can suddenly change, and it handles multimodal backgrounds such as moving foliage (cyclical/oscillating motion) without manual initialisation. Unfortunately, accurate maintenance of mean, variance and weight for the MoG at every pixel location requires the use of floating-point numbers (unsuitable for hardware implementation [3]), or high precision fixed point numbers when implemented in hardware. To efficiently implement the MoG background subtraction algorithm on a hardware architecture for real-time processing, modifications have been made to the algorithm in section 2.3; trading off some level of accuracy for speed-up.

2.3. Our Approach

We present here a novel hybrid background modelling algorithm [1] that combines the attractive features of Grimson's algorithm [26] and the temporal low-pass filtering [20], with appropriate modifications to improve segmentation of the foreground image. The main contribution of the algorithm presented is its efficient implementation on a reconfigurable hardware platform such as Field Programmable Gate Array (FPGA). Following Grimson [26], we maintain a number of clusters, each with weight w_k, where $1 \leq k \leq K$, for K clusters. Rather than modelling a Gaussian distribution, we maintain a model with a central value, c_k of 11-bits (8 bits integer part and 3 bits fractional part). We use an implied global range, $[c_k - 15, c_k + 15]$, rather than explicitly modelling a range for each pixel based on its variance as in [26]. The weights and central values of all the clusters are initialised to 0.

A pixel $X = I(i, j)$ (where X is 11-bit fixed-point) from an image I is said to match a cluster, k, if $X \geq c_k - 15$ and $X \leq c_k + 15$. The highest weight matching cluster is updated, if and only if its weight after the update will not exceed the maximum allowed value (i.e. $w_k \leq 64$, given the data width of the weight as 6 bits). The exponential moving average, which is easier to implement on FPGA has been used in all the update equations. Typically, more weight is given to the observed value to obtain a value closer to the average. The update for the weight is as follows:

$$w_{k,t} = \begin{cases} \frac{63}{64} w_{k,t-1} + \frac{1}{64} & \text{for the matching cluster} \\ \frac{63}{64} w_{k,t-1} & \text{otherwise} \end{cases} \tag{10}$$

The central values of all the clusters are also updated as follows:

$$c_{k,t,i,j} = \begin{cases} \frac{7}{8} c_{k,t-1,i,j} + \frac{1}{8} X_{i,j} & \text{matching cluster} \\ c_{k,t-1,i,j} & \text{otherwise} \end{cases} \tag{11}$$

Where $c_{k,t,i,j}$ is the central value for cluster k at time t for pixel (i, j). If no matching cluster is found, then the least weighted cluster's central value, c_K is replaced with X; its weight is reset to zero. The way we construct and maintain clusters make our approach gradually incorporate new background objects. The K distributions are ordered by weight, with the most likely background distribution on top. Similar to [26], K (usually set to 3) is the total number of clusters used to model every single background pixel (multimodal) to accommodate oscillating background objects. The first B out of K clusters are chosen as the background model, where

$$B = arg_b \; min(\sum_{k=1}^{b} \omega_i > T) \tag{12}$$

. The threshold T is a measure of the minimum portion of the data that should be accounted for by the background.

We classify a pixel as foreground pixel based on the following two conditions:

1. If the intensity value of the pixel matches none of the K clusters.
2. If the intensity value is assigned to the same cluster for two successive frames, and the intensity values $X(t)$ and $X(t-1)$ are both outside the 40% mid-range $[c_k - 6, c_k + 6]$.

The second condition makes it possible to detect targets with low contrast against the background, while maintaining the concept of multimodal backgrounds. A typical example is a moving object with greyscale intensity close to that of the background, which would be classified as background in [26]. This requires the maintenance of an extra frame, with values representing the recently processed background intensities.

2.4. Experimental Results

We evaluate the performance of our approach against that of [26] using $K = 3$, thus 3 cluster in our case and 3 distributions in [26] per pixel. We use twelve randomly selected video sequences; seven outdoor and five indoor scenes with a total of 1600 frames. One of the seven outdoor sequences is taken from a publicly available dataset (PETS 2000 camera 1) [23], two taken with a close range camera placed at a distance of about five meters and the last four taken in a reasonably challenging outdoor scene with oscillating background objects (river waves and foliage). The five indoor scenes

have varying lighting intensity, with two taken in a very reflective room. Reference standard segmentations on these sequences have been constructed by using manually marked frames; results of the algorithms are compared to this reference standard. The results of the pixel level comparison between the ground truth and the extracted foreground image for each frame of all the 12 sequences are based on true positive(TP), true negative(TN), false negative(FN) and false positive(FP) pixels. We have used three measures Sensitivity (SENS) expressed as $\frac{TP}{TP+FN}$, Jaccard Coefficient (JC) expressed as $\frac{TP}{TP+FP+FN}$ and Percentage Correct Classification (PCC) expressed as $\frac{TP+TN}{TP+FP+TN+FN}$ to evaluate the performance of the two algorithms. Table 1 shows the superiority of our algorithm against that in [26] in terms of sensitivity. The algorithm does produce more false positive errors; this is the side-effect of our approach in detecting targets with low contrast against the background. However, in our target application false positive errors of the type reported are more acceptable than false negative errors, as subsystem tracking stages can discard distracters such as shadows. Figure 1 gives sample images of the outputs generated by the two algorithms.

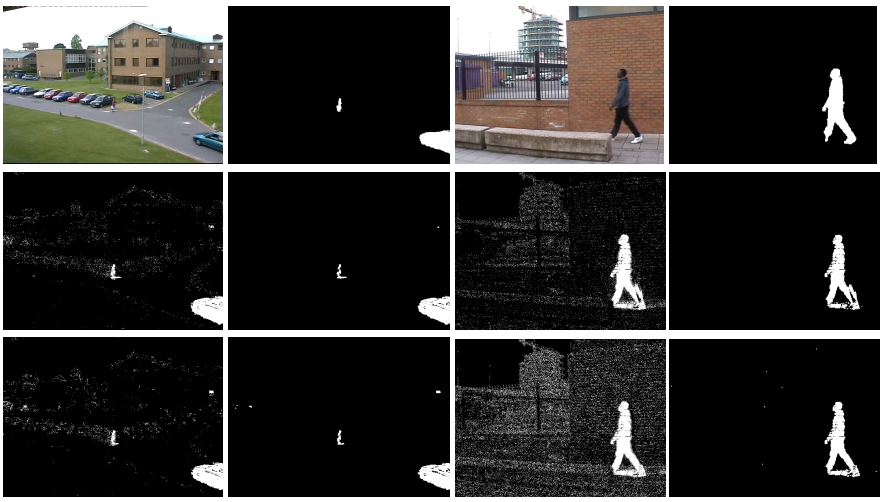

Figure 1: Sample outputs of the algorithms. The top row shows the original images with their corresponding manually marked ground truth. The middle and bottom rows show outputs of Grimson's algorithm and our approach respectively. Images on the second and fourth columns are the outputs after applying morphology.

	Grimson's (%)			**Our Approach (%)**		
Seq.	SENS	JC	PCC	SENS	JC	PCC
1	78.92	**62.30**	**99.86**	**79.08**	60.04	99.84
2	80.99	76.67	99.34	**83.18**	**79.55**	**99.43**
3	64.34	59.93	95.86	**79.87**	**75.02**	**97.44**
4	82.20	72.27	98.06	**84.88**	**73.08**	**98.07**
5	68.55	**48.68**	**97.10**	**76.72**	41.76	95.71
6	68.70	49.51	96.11	**76.26**	**52.13**	**96.12**
7	45.96	32.99	**98.56**	**56.35**	**35.27**	98.40
8	75.58	**53.99**	**99.52**	**81.39**	51.23	99.42
9	74.33	64.75	97.50	**89.91**	**69.40**	**97.55**
10	48.60	37.48	95.60	**83.01**	**53.69**	**96.11**
11	68.50	**11.76**	**80.90**	**80.23**	10.64	74.95
12	48.47	**17.73**	**93.62**	**79.44**	17.58	89.43

Table 1: Per-pixel performance in terms of Sensitivity, Jaccard Coefficient and Percentage Correct Classification for the approach in [26] and our's. The three measures are computed on the outputs from the algorithms after applying morphological opening. For every sequence, the value in bold signifies the algorithm with the best measure for that particular sequence.

3. Connected Components

The foreground extraction algorithms described in section 2 are used to identify foreground pixels in each new frame while updating the description of each pixel's background data. This is followed by the labelling of the binary foreground image into regions using the connected components algorithm. We present a *run-length* based connect component algorithm, which can easily be integrated with the background subtraction algorithm to avoid the intermediate binary foreground data structure. The algorithm is similar to the two-pass connected component algorithm, but rather than pixels, we use run-lengths, which are much more compact that individual pixels. The original two-pass algorithm (Rosenfeld and Pfaltz [22, 24]), uses two raster scans of the image and an equivalence table resolution stage[19]. Details of the algorithm (using the 4-adjacency definition of connectivity) are as follows:

Initial labelling:-The binary input image is scanned in raster order, to label foreground pixels by reference to the left and upper neighbours. If the neighbours are background pixels, a new label is assigned. If they are both foreground with different labels, the left label is copied to the current pixel, and an entry is made in the *equivalence table* linking the upper and left labels. If only one neighbour is labelled, or they share the same label, the value is propagated.

Resolving Equivalences:-The initial labelling step is followed by equivalence resolution, after which the image is scanned again and each label is replaced by the label assigned to its equivalence class. This approach has been used in [16, 25], directly, or modified for implementation as the resolution of label equivalences has a great impact on run time. One drawback of this algorithm is the high dynamic storage required for the equivalence table.

To reduce storage requirements, Haralick [12] suggested a multi-pass labelling algorithm with increased computational time. In this algorithm, a foreground pixel with only background neighbours is assigned a new label. However, where there are any labelled neighbours, the minimum label is assigned. The algorithm alternates raster and reverse-raster scans until the labelling stabilises. The algorithm is highly sequential, and the repeated image scans make it computationally inefficient, as scan-times tend to dominate connected component cycles.

Crookes *et al* [8] successfully implemented the multi-pass connected component labelling algorithm on an FPGA using off-chip RAM. The implementation fits on a single Xilinx XC4010E chip with 20×20 CLBs and for an

image size of 256×256, the design runs at 76MHz. The implementation suffers from all the problems associated with the base algorithm and hence the time taken to process a frame is dependant on the complexity of objects in the scene. A similar implementation is presented in by Benkrid et al. ([4]) using a bigger device, Xilinx Virtex-E FPGA. For an image size of 1024×1024, the design consumes 583 slices, 5 Block RAMs and runs at 72MHz.

3.1. Our Approach

The labelling algorithm as presented in [24] can result in a very large equivalence table. Resolving the equivalence table has been the focus of most labelling algorithms, with little effort to implement such algorithms on hardware architecture for real-time processing. Our algorithm [2] is suitable for implementation on a hardware platform, and also minimises use of memory. Our key contribution is to process using a run-length encoding representation. This is easily parallelised by processing multiple rows in parallel, and can be integrated with the background differencing algorithm, avoiding the requirement to calculate a binary foreground image as an intermediate data structure. The run-length encoded format is also much more compact than a binary image (individual *runs* have a single label), and so the sequential label propagation stage is much faster than the conventional algorithm. Details of the algorithm are given below.

The stages involved in our implementation are as follows:

1. Pixels are converted to runs in parallel by rows,
2. Initial labelling and propagation of labels,
3. Equivalence table resolution and
4. Translating run labels to connected component.

The design is parallelised as much as possible. Although stages 2 and 3 are sequential, they operate on runs, which are far less numerous than pixels. Similar to stage 1, stage 4 can be executed in parallel by row. A run has the properties $\{ID, EQ, s, e, r\}$, where ID is the identity number of the run, EQ is the equivalence value, s the x offset of the start pixel, e the x offset of the end pixel, and r the row. The first stage involves row-wise parallel conversion from pixels to runs. Depending on the location and access mode of the memory holding the image, the entire image may be partitioned into n parts to achieve n run-length encoding in parallel. The use of runs rather

than pixels reduces the size of the equivalence table in some cases similar to figure 4 and hence makes it easier to resolve.

The following sequential local operations are performed in parallel on each partition, for an image size $M \times N$ to assign pixels to runs:

Algorithm 3.1: PIXELTORUNS(T)

$\forall T : T(x, y) = I(x, y)$
$i \leftarrow 0$
$isBlock \leftarrow 0$
if $T(x, y) = 1$ **and** $isBlock = 0$

$$\textbf{then} \begin{cases} s_i \leftarrow x \\ isBlock \leftarrow 1 \end{cases}$$

if $isBlock = 1$ **and** $(T(x, y) = 0$ **or** $x = M)$

$$\textbf{then} \begin{cases} e_i \leftarrow (x - 1) \\ r_i \leftarrow y \\ ID_i \leftarrow EQ_i \leftarrow 0 \\ i \leftarrow i + 1 \\ isBlock \leftarrow 0 \end{cases}$$

where $isBlock$ is 1 when a new run is scanned for partition T and M is the width of the image. A run is complete when the end of a row is reached or when a background pixel is reached. The maximum possible number of runs in an $M \times N$ image is $2MN$. This worst case occurs when the image is a pixel-wise chequerboard pattern; see figure 2. The second stage involves initial labelling and propagation of labels. The IDs and equivalences (EQs) of all runs are initialized to zero. This is followed by a raster scan of the runs; assigning provisional labels which propagate to any adjacent runs on the row below. For any unassigned run ($ID_i = 0$) a unique value is assigned to both its ID and EQ.

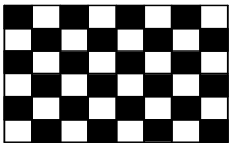

Figure 2: Worse case scenario.

For each run i with ID ID_i, excluding runs on the last row of the image; runs one row below run_i are scanned for an overlap. An overlapping run in

4-adjacency (ie. $s_i \leq e_j$ and $e_i \geq s_j$) or 8-adjacency (ie. $s_i \leq e_j + 1$ and $e_i + 1 \geq s_j$) is assigned the ID ID_i, if and only if ID_j is unassigned. If there is a conflict (if an overlapping run has assigned ID_j), the equivalence of run i, EQ_i is set to ID_j. This is summarized in algorithm 3.2.

Algorithm 3.2: InitLabelling($runs$)

$$
\begin{array}{l}
m \leftarrow 1 \\
\textbf{for } i \leftarrow 1 \textbf{ to } TotalRuns \\
\quad \textbf{do} \begin{cases}
\textbf{if } ID_i = 0 \\
\quad \textbf{then} \begin{cases} ID_i \leftarrow EQ_i \leftarrow m \\ m \leftarrow m + 1 \end{cases} \\
\textbf{for each } r_j \in r_{i+1} \\
\quad \textbf{do} \begin{cases}
\textbf{if } ID_j = 0 \textbf{ and } e_i \geq \\
s_j \textbf{ and } s_i \leq e_j \\
\quad \textbf{then} \begin{cases} ID_j \leftarrow ID_i \\ EQ_j \leftarrow ID_i \end{cases} \\
\textbf{if } ID_j \neq 0 \textbf{ and } e_i \geq \\
s_j \textbf{ and } s_i \leq e_j \\
\quad \textbf{then } \{EQ_i \leftarrow ID_j
\end{cases}
\end{cases}
\end{array}
$$

Where $TotalRuns$ excludes runs on the last row of the image. Applying $PixelToRuns()$ to the object in figure 3 (a 'U' shaped object) will generate four runs each with unassigned ID and EQ.

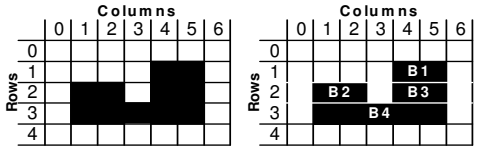

Figure 3: U shaped object with 4 runs after PixelToRuns()

The third stage is resolution of conflicts, similar to [24] as shown in algorithm 3.3. In the example above (figure 3 and table 2) a conflict occurs at $B3$; the initially assigned $EQ = 1$ in iteration 1 changes to $EQ = 2$ in iteration 3 due to the overlap with $B1$ and $B4$, see table 2. This conflict is resolved in $ResolveConflict()$ resulting in $ID = 2$ and $EQ = 2$ for all the four runs. Even though $ResolveConflict()$ is highly sequential, it takes half the total cycles as the two 'if statements' in the second loop are executed

IT		**B1**	**B2**	**B3**	**B4**
1	ID	1	0	1	0
	EQ	1	0	1	0
2	ID	1	2	1	2
	EQ	1	2	1	2
3	ID	1	2	1	2
	EQ	1	2	2	2

Table 2: Results for the object in fig.3 after 3 iterations.

simultaneously. The final IDs (final labels) are written back to the image at the appropriate pixel location, without scanning the entire image, as each run has associated s, e and r values.

Algorithm 3.3: ResolveConflict($runs$)

for $i \leftarrow 1$ **to** $TotalRuns$
do { **if** $ID_i \neq EQ_i$
then { $TID \leftarrow ID_i$
$TEQ \leftarrow EQ_i$
for $j \leftarrow 1$ **to** $TotalRuns$
do { **if** $ID_j = TID$
then $\{ID_j \leftarrow TEQ$
if $EQ_j = TID$
then $\{EQ_j \leftarrow TEQ$

To illustrate the performance of the algorithm, consider the stair-like 8-connected component illustrated in figure 4. Using the multiple pass approach, a stair-like component with N steps will require $2(N-1)$ scans to completely label. Using our approach, both images take two scans to completely label. As shown in figure 4, the runs are extracted in the first scan, while the 8-adjacency labelling is done in the second scan. Tables 3 and 4 show results after the first and second scan respectively. It is clear from table 4 that no further scans are required. The same image (figure 4) will result in an equivalence table with five entries if the two-pass algorithm[24] is used, due to the successive offsetting of rows to the left.

	B1	B2	B3	B4	B5	B6	B7
ID	0	0	0	0	0	0	0
EQ	0	0	0	0	0	0	0
ST	16	6	14	3	12	0	10
EN	17	8	15	5	13	2	11
RW	0	1	1	2	2	3	3

Table 3: Results after first image scan, where ST=start, EN=end and RW=row.

	B1	B2	B3	B4	B5	B6	B7
ID	1	2	1	2	1	2	1
EQ	1	2	1	2	1	2	1

Table 4: Results after second scan. The start, end and row remain unchanged

3.2. Analysis of our algorithm

We compared our implementation and that presented in [4] using a set of complex images of various sizes. The testing environment is an Intel Pentium IV 2.8GHz personal computer with 2.0GB SDRAM running MATLAB/C. A graph showing the processing time (in seconds) of the two implementations, for 200 naturalistic images each of size 720×576 is shown in figure 5. In the worse case scenario, where there are no continuous pixels in a row as in figure 2, our approach incurs extra writing overheads. The implementation in hardware running at real-time is a significant advantage.

4. Segmentation and Labelling in Hardware

In this section we present a hardware implementation of the multi-modal background modelling algorithm [1] in section 2, followed by the labelling of the foreground objects using the run-length encoding algorithm [2] presented

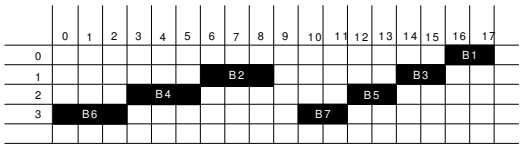

Figure 4: Example of a 3 and 4-stairs connected component

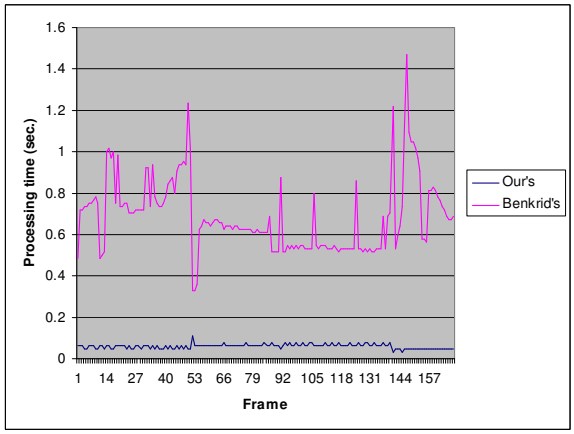

Figure 5: Graph showing processing time for 200 images of size 720x576

in section 3. The run-length encoding is integrated with the detection of foreground pixels during segmentation, avoiding the intermediate data structure as well as reducing the latency of the entire implementation on FPGA. Multimodal background modelling is required to model scenes with non-stationary background objects, so reducing false positive alerts. To successfully implement our algorithm on a Field Programmable Gate Array, access to $17n$ bits of background data is required every clock cycle, where n is the total number of background clusters and 17 bits is the total number of bits require for the cluster's weight (6 bits) and central value (11 bits). The hardware setup is composed of a digital video camera, two display units (one to output a binary foreground image for diagnostic purposes, and the other to output the connected component labels) and an FPGA prototyping board. The RC340 board is packaged with Xilinx Virtex IV XC4VLX160 FPGA, 4 banks of ZBT SRAM totalling 32MBytes and two DVI output ports. Note that the maximum achievable clock frequency of the FPGA chip, is constrained by the connected external devices.

Camera data is sent to the FPGA chip every clock cycle. The address of the pixel data from the camera is used to address the correct background information from the external RAM. The RGB data from the camera is converted into greyscale intensity and compared with the background data. The resulting binary pixel from the comparison is then converted into run-length for further processing into a connected component label. Figure 6 is a high-level diagram of the setup. The design on FPGA has six different blocks;

the rgb2gray, binarization, background update, initial labelling, conflict resolution and the output blocks. The initial labelling and conflict resolution blocks are triggered when an entire frame has been binarized. All the other four blocks run in parallel.

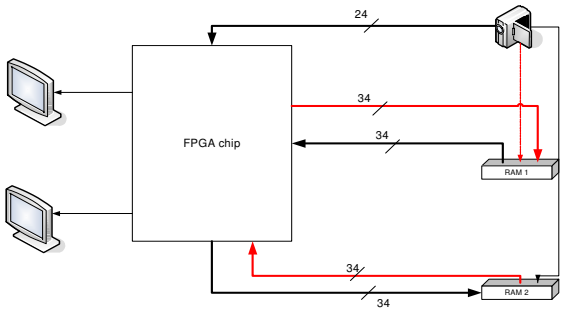

Figure 6: A high-level block diagram of the FPGA implementation

4.0.1. rgb2gray Block

The rgb2gray block reads 24 bit camera data, converts it into 8 bit grayscale intensity and further converts the 8 bits into a 11 bit fixed-point value with 8 bits integer part and 3 bits fractional part. The corresponding 34 bit background data for the current pixel is read from the external RAM onto the FPGA chip in 2 clock cycles. The 34 bit data is for the 2 background components, each 17 bits long. This block is fully pipelined and it takes exactly three clock cycles to complete. Note the external RAM is a single port RAM, hence the need to use two blocks for double buffering; background data is read from the first and the updated background data written onto the other.

4.0.2. Binarization/run Encoding Block

This block identifies foreground pixels and encodes the runs; it also generates the foreground image for verification purposes (this part of the implementation could be discarded in a completed system). The 34 bit background data from the rgb2gray block is split into two 17 bit bimodal background components. Each of the 17 bits is then further split into their corresponding background intensity value of 11 bits and weight value of 6 bits. A range is then defined around the 11 bit background value. If the current pixel value falls within either of the defined ranges, the pixel is considered background;

else it is a foreground pixel. If the current pixel is the first foreground pixel in the row or the first foreground pixel after the last run-length in the same row, a new run-length encoding starts with this pixel as the first in the run-length. If the pixel is a background pixel or the last in the row, any run-length encoding already in progress will terminate with the previous pixel as the last in the run-length.

4.0.3. Background Update Block

The current pixel value is used to update the exiting background value as shown in equation 11, thus if the pixel is a foreground pixel, the least weighted background model is replaced with the current pixel value. This block takes exactly 2 clock cycles to complete. The updated value is then written onto the external RAM block and used when the next frame is read from the camera.

4.0.4. Initial Labelling Block

This block is only triggered when an entire frame has been processed. The run-lengths generated from the binarization/run encoding block are used in this block. Initial run-length labels which may propagate onto overlapping runs are assigned in this block. Two runs overlap if they are a pixel away from each other, either horizontally or vertically (for 4-adjacency) and diagonally (for 8-adjacency). This block takes as many clock cycles as there are runs in the current image.

4.0.5. Conflict Resolution Block

This block is used to resolve any entries in the equivalence table. There is an entry in the equivalence table if the ID and EQ of a run-length are not equal after the initial labelling. This is the most sequential part of the implementation and can take the same number of clock cycles as the square of the number of runs in the image to complete in the worse case. However, such a case is unusual in real-world images – with the use of run-lengths there are normally few or no entries in the equivalence table.

4.1. Output Block

This block is used to control the two external VGA displays and runs at the same frequency as the display units, typically 60Hz. The binary image generated during the image binarization is written onto a dual-port Block

RAM for display for visual verification. The output of the connected component label is also buffered onto a different dual-port Block RAM for display purposes. Three different colours are used to display the connected components to distinguish between different regions.

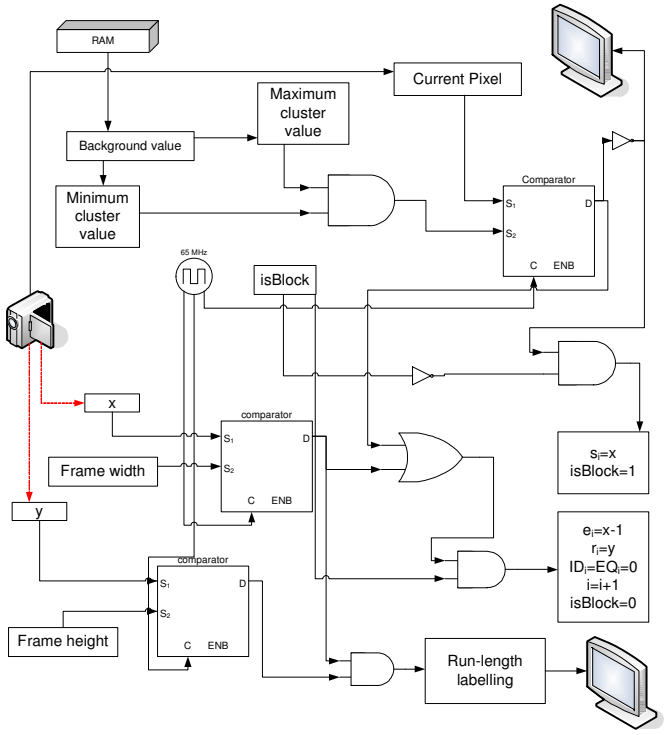

Figure 7: A block diagram of the entire architecture as implemented on the FPGA fabric.

A block diagram of the FPGA implementation is shown in figure 7. The outputs from the design have been compared with that of the software implementation in MATLAB/C++ to verify the correctness. Sample outputs from the implementation are shown in figure 8. The entire design implemented on the Xilinx Virtex IV FPGA has been clocked at a frequency of 65MHz. At this reported frequency, the designed is capable of processing 35 frames per second for a standard image size of 640×480 from binarization to connected component labelling. Note, the clock frequency includes logic for controlling all the external devices like the VGA, memory and camera. Resource utilization of the entire implementation is summarized in table 5.

Resource	Total Used	Per.
Flip Flops	1,316 out of 135,168	1%
4 input LUTs	1,232 out of 135,168	1%
Block RAMs	19 out of 144	6%
bonded IOBs	239 out of 768	31%
Occupied Slices	1,351 out of 67,584	1%
SSRAM (VGA)	20 out of 256 Mbits	7.8%

Table 5: Resource utilization of the bimodal greyscale implementation, using *XC4VLX160*, package *ff1148* and speed grade *-10*.

5. Conclusion

We have demonstrated how a single chip FPGA can effectively be used in the implementation of a highly sequential digital image processing task in real-time. This paper presents hardware implementations of two different algorithms, combined effectively for FPGA implementation. We have presented an architecture for extracting connected components from a binary image generated using a multimodal (bimodal) background modelling on an FPGA without a frame buffer when using a camera as the input source.

The connected component algorithm that we have successfully implemented is an extension of the very first algorithm present in [24], exploiting the desirable properties of run-length encoding, combined with the ease of parallel conversion to run-length encoding. The processing speed and resources used in the implementation make room for other vision analysis algorithms to be implemented on the hardware platform.

References

[1] K. Appiah and A. Hunter. A single-chip fpga implementation of real-time adaptive background model. IEEE International Conference on Feild-Programmable Technology, 2005.

[2] K Appiah, A. Hunter, P. Dickinson, and J. Owens. A run-length based connected component algorithm for fpga implementation. IEEE International Conference on Feild-Programmable Technology, 2008.

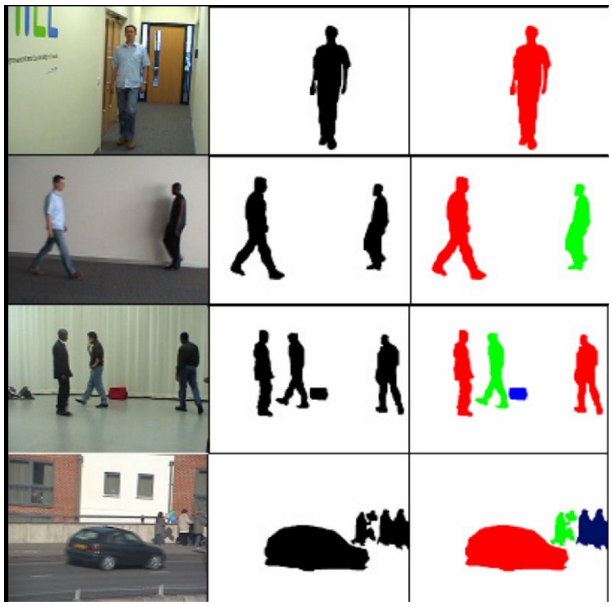

Figure 8: Sample outputs of the FPGA implementation. Original images to the left, binary images after segmentation in the middle and the results after connected component labelling to the right.

[3] E. Ashari. *FPGA Implementation of Real-Time Adaptive Image Thresholding*. SPIE–The International Society for Optical Engineering, December, 2004.

[4] K. Benkrid, S. Sukhsawas, D. Crookes, and A. Benkrid. An fpga-based image connected component labeller. In *Field-Programmable Logic and Applications*, 2003.

[5] V. Bonato, A. Sanches, and M. Fernandes. *A Real Time Gesture Recognition System for Mobile Robots*. International Conference on Informatics in Control, Automation and Robotics, Portugal, August, 2004.

[6] Yuri Boykov and Gareth Funka-Lea. Graph cuts and efficient n-d image segmentation. *International Journal of Computer Vision*, 70(2):109–131, 2006.

[7] Dorin Comaniciu, Visvanathan Ramesh, and Peter Meer. The variable bandwidth mean shift and data-driven scale selection. In *in Proc. 8th Intl. Conf. on Computer Vision*, pages 438–445, 2001.

[8] D. Crookes and K. Benkrid. An fpga implementation of image component labelling. In *Proceedings SPIE Configurable Computing: Technology and Applications*, pages 17–23, September 1999.

[9] P. Ekas. *Leveraging FPGA coprocessors to optimize automotive infotainment and telematics systems.* Embedded Computing Design, Spring, 2004.

[10] A. Elgammal, D. Harwood, and L. Davis. *Non-parametric Model for Background Subtraction.* Proceedings of the 6th European Conference on Computer Vision, Dublin, Ireland, 2000.

[11] D. Gutchess, M. Trajkovic, E. Cohen-Solal, D. Lyons, and A. K. Jain. *A Background Model Initialization Algorithm for video Surveillance.* IEEE, International Conference on Computer Vision, 2001.

[12] R. M. Haralick. *Real time Parallel Computing Image Analysis.* Plenum Press, New York, 1981.

[13] I. Haritaoglu, D. Harwood, and L. Davis. *W^4: Who? When? Where? What? A real time system for detecting and tracking people.* IEEE Third International Conference on Automatic Face and Gesture, 1998.

[14] J.Batlle, J. Martin, P. Ridao, and J. Amat. *A New FPGA/DSP-Based Parallel Architecture for Real-Time Image Processing.* Elsevier Science Ltd., 2002.

[15] C. T. Johnston, K. T. Gribbon, and D. G. Bailey. *Implementing Image Processing Algorithms on FPGAs.* Proceedings of the eleventh electronics New Zealand Conference, ENZCON'04, Palmerston North, Nov, 2004.

[16] V. Khannac, P. Gupta, and C. J. Hwang. Finding connected components in digital images. *Proceedings of the International Conference on Information Technology: Coding and Computing*, 2001.

[17] M. Leeser, S. Miller, and H. Yu. *Smart Camera Based on Reconfigurable hardware Enables Diverse Real-time Applications.* Proc. of the 12th annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM'04), 2004.

[18] B. Levine, B. Colonna, T. Oblak, E. Hughes, M. Hoffelder, and H. Schmit. *Implementation of a Target Recognition Application Using Pipelined Reconfigurable Hardware*. Int. Conf. on Military and Aerospace Applications of Programmable Devices and Technologies, 2003.

[19] N. Ma, D. G. Bailey, and C. T. Johnston. Optimised single pass connected components analysis. IEEE International Conference on Feild-Programmable Technology, 2008.

[20] A. Makarov. *Comparison of Background extraction based intrusion detection algorithms*. IEEE Int. Conference on Image Processing, 1996.

[21] S. McBader and P. Lee. *An FPGA Implementation of a Flexible, Parallel Image Processing Architecture Suitable for Embedded Vision Systems*. Proceedings of the International Parallel and Distributed Processing Symposium, IPDPS'03, 2003.

[22] J. Park, C. G. Looney, and H. Chen. *Fast Connected Component Labeling Algorithm Using A Divide and Conquer Technique*. Technical report, 2000.

[23] PETS2000. First ieee international workshop on performance evaluation of tracking and surveillance, March.

[24] A. Rosenfeld and J. Pfaltz. Sequential operations in digital picture processing. In *Journal of the ACM*, pages 241–242, 1966.

[25] C. Schmidt and A. Koch. *Fast Region Labeling on the Reconfigurable Platform ACE-V C*. Field-Programmable Logic and Applications, 2003.

[26] C. Stauffer and W. E. L. Grimson. *Adaptive background mixture models for real-time tracking*. IEEE Conference on Computer Vision and Pattern Recognition, 1999.

[27] C. Stauffer and W. E. L. Grimson. Learning patterns of activity using real-time tracking. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22:747–757, 2000.

[28] R. Williams. *Increase Image Processing System Performance with FPGAs*. Xcell Journal, Summer, 2004.

[29] C. Wren, A. Azarbayejani, T. Darrel, and A. Pentland. *Pfinder: Real-time tracking of the human body*. IEEE Transactions on Pattern Analysis and Machine Intelligence, 1997.

[30] Huiyu Zhou and Schaefer Gerald. Fuzzy c-means variants for medical image segmentation. *International Journal of Tomography & Statistics*, 13(W10):3–18, 2010.

[31] Huiyu Zhou, Schaefer Gerald, Abdul Sadka, and Celebi Emre. Anisotropic mean shift based fuzzy c-means segmentation of dermoscopy images. *Jour. of Selected Topics in Signal Processing*, 3(1):26–34, 2009.

[32] Huiyu Zhou, Schaefer Gerald, and Chunmei Shi. A mean shift based fuzzy c-means algorithm for image segmentation. *30th Annual International Conference of the IEEE Engineering in Medicine and Biology Society*, 2008.