Proc. of the IEEE Symposium on Field-Programmable Custom Computing Machines, Napa, USA, 2002

FPGA-based Template Matching using Distance Transforms

S. Hezel, A. Kugel and R. Männer
Department of Computer Science V
University of Mannheim
B6, 23-29, 68131 Mannheim, Germany
{hezel,kugel,maenner}@ti.uni-mannheim.de

D.M. Gavrila
Image Understanding Systems
DaimlerChrysler Research
Ulm 89081, Germany
dariu.gavrila@DaimlerChrysler.com

Abstract

This paper presents a high-performance FPGA solution to generic shape-based object detection in images. The underlying detection method involves representing the target object by binary templates containing positional and directional edge information. A particular scene image is preprocessed by edge segmentation, edge cleaning and distance transforms. Matching involves correlating the templates with the distance-transformed scene image and determining the locations where the mismatch is below a certain user-defined threshold. Although successful in the past, a significant drawback of these matching methods has been their large computational cost when implemented on a sequential general-purpose processor.

In this paper, we present a step by step implementation of the components of such object detection systems, taking advantage of the data and logical parallelism opportunities offered by an FPGA architecture. The realization of a pipelined calculation of the preprocessing and correlation on FPGA is presented in detail.

1 Introduction

Object detection is one of the central tasks of image understanding. Template-based matching methods using distance transforms have proven to be quite successful in this regard, because of their robustness to missing or partially incorrect data (i.e. occlusion) and their non-reliance on high-level feature extraction which is notoriously error-prone [3, 1, 2]. In these methods, target objects are represented by binary templates containing positional (and possibly directional) edge information. On-line, a particular scene image is preprocessed by edge segmentation, edge cleaning and distance transforms. Matching then involves correlating the templates with the distance-transformed scene image and determining the locations where the mismatch is below a certain user-defined threshold. These image locations are considered to contain the "detected" objects.

Such matching methods are very generic and applicable across a large variety of domains (i.e. automatic target recognition in the military domain, visual inspection in the industrial domain, object detection onboard "intelligent" vehicles). One large drawback is their large computational cost when implemented on a general-purpose sequential processor. Main bottlenecks are the computation of the distance transform and the correlation.

In this paper, we present a step by step implementation of the components of such object detection systems, taking advantage of the data and logical parallelism opportunities offered by an FPGA architecture. Integer operations are frequently used in many image processing algorithms, such as FIR-filters for edge detection, and often they need only low precision arithmetics. The employed simple computational patterns can be realized with highly parallel pipelines as described e.g. in [9]. Applied to distance transforms and other morphological operations which imply mostly the evaluation of boolean operations or comperators on local stucturing elements, high speed-up can be achieved compared to general purpose sequential machines like PCs [4]. This holds true even though the clock frequencies of typical FPGA designs (50 .. 200 MHz) are much lower than those of state of the art CPUs ($\geq$ 1 GHz). The presented work profits from the heavy use of parallel and pipelined operations, thus enabling a significant speed-up for the object-detection application.

Methods for single and multiple template matching on FPGAs are described in [6, 7]. In both cases a binary image is shifted over a binary template hardwired into FPGA, and the correlation is being calculated by adder trees. In our case the matching is done for many templates concurrently using several distance transformed images. The huge pipeline utilized for the subsequent correlation is described as are techniques for the reduction of FPGA resource requirements and optimized handling of the data of several distance transformed images. For implementation, simulation and synthesis we used CHDL (a C++ based Hardware Description Language), which is being developed at the University of Mannheim since 1995 [10]. This tool enables a very high level of integration for HS/SW co-design, which is a big advantage for this kind of distributed application. All our FPGA implementations target PCI based FPGA coprocessors. The initial tests have been carried out on the commercial boards μEnable-I and its successor μEnable-II [8]. The μEnable-I series of boards is based on the Xilinx XC4000 family of FPGAs supported by a single memory bank. The μEnable-II series uses an XCV1000 FPGA with 2 banks of memory.

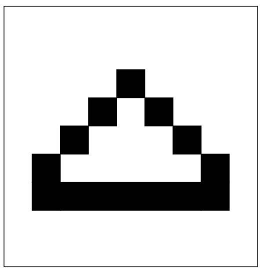
For the final implementation we are using the RACE-1 co-processor developed at the University of Mannheim [11]. Primarily, it comprises a XILINX Virtex-2 FPGA (XC2V3000) and four 36 bit wide 133 MHz SRAM banks. Moreover it supports 64bit/66MHz PCI and has multiple connectors for external interfaces, e.g. to digital cameras.

The outline of the paper is as follows: In section 2 we introduce the basics of the agorithm as described in [1]. In section 3 we describe the mapping of the preprocessing onto FPGA, resulting in building two large pipelines. In section 4 a pipelined approach is presented to calculate the correlations of multiple templates in parallel. Some issues concerning optimal use of FPGA resources are discussed. Section 5 presents possible strategies how to combine the pipelines of preprocessing and template matching and how to increase the number of templates. We finish with results and conclusion.

2 The Matching Algorithm

2.1 Matching with Distance Transforms

A distance transform (DT) converts a binary image consisting of feature and non-feature pixels into an image where each pixel value denotes the distance to the nearest feature pixel [3, 1]. Figure 1 illustrates a Euclidean Distance Tranform (EDT). More often, DTs such as the *chamfer-2-3* transform are used, providing good integer approximations of true Euclidean distance at low computational cost. These DTs are computed in raster scan fashion; they approximate global distances by propagating distances locally using a mask of fixed size and shape, in a manner independent of the feature locations in the image.



4.2	3.6	2.8	2.2	2.0	2.2	2.8	3.6	4.2
3.6	2.8	2.2	1.4	1.0	1.4	2.2	2.8	3.6
2.8	2.2	1.4	1.0	0.0	1.0	1.4	2.2	2.8
2.2	1.4	1.0	0.0	1.0	0.0	1.0	1.4	2.2
1.4	1.0	0.0	1.0	1.4	1.0	0.0	1.0	1.4
1.0	0.0	1.0	1.0	1.0	1.0	1.0	0.0	1.0
1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0
1.4	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.4
2.2	2.0	2.0	2.0	2.0	2.0	2.0	2.0	2.2

Figure 1: A binary pattern and its Euclidean Distance Transform

Matching with DT is illustrated schematically in Figure 2. It involves two binary images, a segmented template T and a segmented image I, which we will call "feature template" and "feature image". The "on" pixels denote the presence of a feature and the "off" pixels the absence of a feature in these binary images. What the actual features are, does not matter for the matching method. Typically, one uses edge-points, and we will do so throughout this paper. The feature template is given off-line for a particular application, and the feature image is derived from the image of interest by feature extraction.

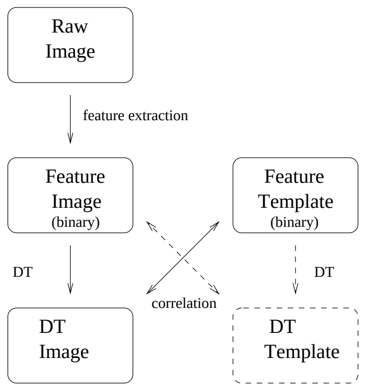


Figure 2: Matching using a DT

Matching T and I involves computing the distance transform of the feature image I. The template T is transformed (e.g. translated) and positioned over the resulting DT image of I; the matching measure $D(T, I)$ is determined by the pixel values of the DT image which lie under the "on" pixels of the template. These pixel values form a distribution of distances of the template features to the nearest features in the image. The lower these distances are, the better the match between image and template at this location.

A number of matching measures can be defined on the distance distribution. One possibility is to use the average distance to the nearest feature. This is the *chamfer* distance

$$D_{chamfer}(T,I) \equiv \frac{1}{|T|}\sum_{t\in T} d_I(t) \qquad (1)$$

where $|T|$ denotes the number of features in T and $d_I(t)$ denotes the distance between feature t in T and the closest feature in I. The chamfer distance thus consists of a correlation between T and the distance image of I, followed by a division. In applications, a template is considered as matched at locations where the distance measure $D(T,I)$ is below a user-supplied threshold θ

$$D(T,I) < \theta \qquad (2)$$

Figure 3 illustrates the matching scheme of Figure 2 for the typical case of edge features. Figure 3a-b shows a "toy" image and template. Figure 3c-d shows the edge detection and DT transformation of the edge image. The distances in the DT image are intensity-coded; lighter colors correspond to increasing distance values.

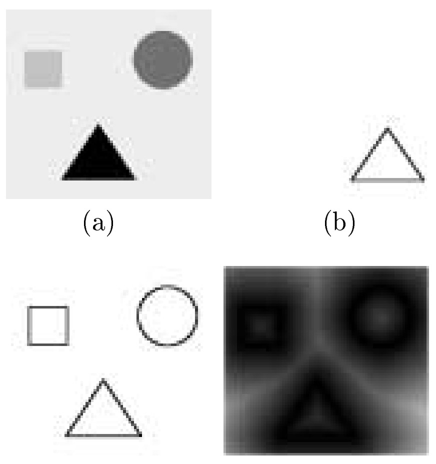


Figure 3: (a) original image (b) template (c) edge image (d) DT image

The advantage of matching a template (Figure 3b) with the DT image (Figure 3d) rather than with the edge image (Figure 3c) is that the resulting similarity measure is smoother than a function of the feature positions, allowing tolerance between a template and an object of interest in the image. Matching with the unsegmented (gradient) image on the other hand typically provides strong peak responses for "ideal" templates, but rapidly declining off-peak responses with slightly increasing template-image dissimilarity.

For real images, edge segmentation also introduces spurious edges. In order to reduce the significant impact isolated edge points can have on subsequent distance transform computation, an additional filtering step is typically performed; it involves the removal of all connected edge segments of size below a certain user-supplied threshold.

2.2 Extension to Multiple Feature-Types: Edge Orientation

No distinction has so far been made with regard to the type of (edge) features. All features would appear in one feature image (or template) and, subsequently, in one DT image. If there are several feature types, and under consideration of the match of a template at a particular location of the DT image, it is possible that the DT image entries reflect shortest distances to features of non-matching type. The similarity measure would be too optimistic, increasing the number of false positives one can expect from matching.

A simple way to increase matching discrimination by distinguishing multiple feature types is to use separate feature images and DT images for each type. Thus having M distinct feature types results in M feature images and M DT images. Similarly, the "untyped" feature template is multiplexed in M "typed" feature templates. Matching proceeds as before, but now the match measure between image and template is the sum of the match measures between template and DT image of the same type.

Considering the case of edge points as features, we use edge orientation as feature type by partitioning the unit circle in M bins

$$\{\ [\ \frac{i}{M}2\pi, \frac{i+1}{M}2\pi\ ]\ |i = 0,...,M-1\ \} \qquad (3)$$

Thus a template edge point with edge orientation ψ is assigned to the typed template with index

$$\lfloor \frac{\psi}{2\pi} M \rfloor \qquad (4)$$

We still have to account for measurement error in the edge orientation and the tolerance we will allow between the edge orientation of template and image points during matching. Let the absolute measurement error in edge orientation of the template and image points be $\Delta\phi_T$ and $\Delta\phi_I$, respectively. Let the allowed tolerance on the edge orientation during matching be $\Delta\phi_{tol}$. In order to account properly for these quantities, a template edge point is assigned to a range of typed templates, namely those with indices

$$\{\lfloor\frac{(\psi-\Delta\phi)}{2\pi}M\rfloor,...,\lfloor\frac{(\psi+\Delta\phi)}{2\pi}M\rfloor\} \qquad (5)$$

mapped cyclically over the interval $0,...,M-1$, with

$$\Delta\phi = \Delta\phi_T + \Delta\phi_I + \Delta\phi_{tol} \qquad (6)$$

For applications where there is no sign information associated with the edge orientation, a template edge point is also assigned to the typed templates one obtains by substituting $\psi+\pi$ for ψ in Equation (5).

2.3 Matching algorithm components

In summary, our matching algorithm has the following logical components. For the preprocessing of the scene image:

1. edge detection
2. edge noise removal
3. computation of distance transform

For the actual matching:

4. correlation between template and DT image

We now proceed with the description of the FPGA implementation of the above components in the following sections.

3 Architecture of Preprocessing

The preprocessing basically consists of edge detection, morphological clean and distance transformation. Additionally, there are some data-formatting steps in order to accelerate memory access. All these operations are well suited for a straightforward pipelined implementation on an FPGA. For the calculation of the distance transformation we use a sequential approach utilising a forward and a backward step. After calculating the forward tansformation the intermediate result image must be stored. Hence the total preprocessing is composed of two parts. First the edge detection, morphological clean and forward distance transformation takes place in one pipeline as shown in Fig. 6. Backward transformation and data formatting are performed next. The data flow is displayed in Fig. 7.

In the remainder of this section we will describe the hardware implementation of all the modules specific to Sobel and distance transformation. A summary of FPGA resource utilisation and pipeline depths for all preprocessing modules is given in Tab. 1.

3.1 Edge Detection

To determine the edges we use the Sobel operators for x and y direction. They belong to the class of linear shift invariant (LSI) operations. The 3×3 convolution mask of the Sobel operator uses antisymmetric coefficients, as shown at the top of Fig. 4. These neighbourhood transformations are very often calculated by shifting the mask line by line over the image. Our implementation in hardware is done the other way round: the mask is fixed and the image is transformed under the mask line by line. For more details on implementing (LSI) filters on FPGAs see e.g. [9].

Two complete lines from the original image are copied to internal FPGA Block RAM and the currently processed 3×3 region is kept in shift register arrays (SRA) for fully parallel access. The registers and Block RAM are used for both x and y Sobel operators. The calculations are done in parallel with two pipelined arithmetic units (AUs), as shown in Fig. 4. Each AU is able to process the 3×3 pixels in one cycle. This allows to feed a new pixel into the shift register array (SRA) at every clock cycle.

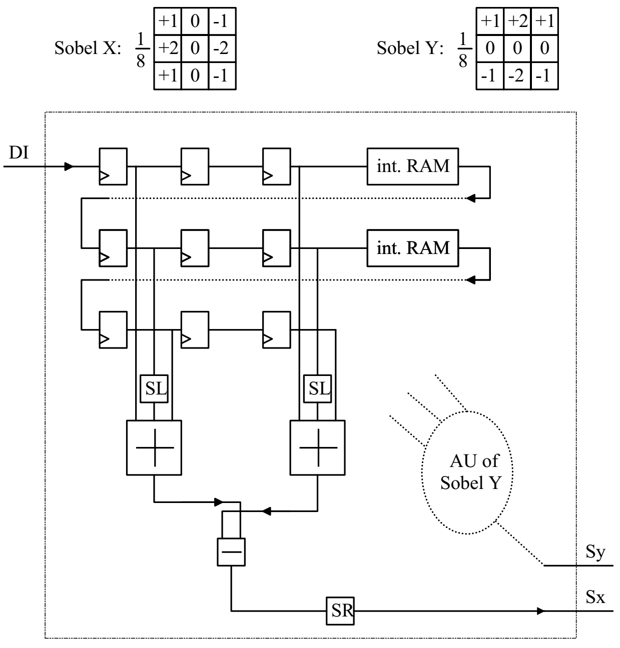

Sobel X: $\frac{1}{8}\begin{array}{|c|c|c|}\hline +1 & 0 & -1\\\hline +2 & 0 & -2\\\hline +1 & 0 & -1\\\hline\end{array}$
Sobel Y: $\frac{1}{8}\begin{array}{|c|c|c|}\hline +1 & +2 & +1\\\hline 0 & 0 & 0\\\hline -1 & -2 & -1\\\hline\end{array}$
DI
int. RAM
int. RAM
SL
SL
AU of Sobel Y
Sy
Sx
SR

Figure 4: Hardware implementation of Sobel operator.

For the calculation of the border pixels there is no need to apply techniques such as zero extension or extrapolation. Instead, the calculation is continued over the border, which can be seen as a periodic extension.

To find the features in a binary image the sum of Sx and Sy is first determined, then the threshold is checked. Only if the sum is above the threshold, the pixel is considered to be a feature pixel.

Parallel to the threshold evaluation a discrete orientation value is derived and 1 of 8 directions assigned to the pixel, according to the corresponding octant of the pixel position. The result values are clipped to 4 bit precision.

3.2 Morphological Clean

The aim of the clean operation is the elimination of noise in the binary edge image. Three or less connected pixels, the “isolated” pixels, are eliminated. In software this morphological operation is implemented in a hierarchical way which involves purely random memory access.

Since this strategy is not well suited for an FPGA implementation, the clean module is built as a pipeline with a logic unit (LU) with parallel access to all relevant pixels. Again, the pixels are stored in a SRA of size 7×5. The LU detects in parallel all possible combinations of three or less connected feature pixels.

The result of the clean operation is used to mask invalid pixels in all 8 directional Sobel images prior to initializing the DT data structure in external RAM.

3.3 Distance Transformation

To approximate the Euclidian distances we use the sequential chamfer 2-3 metric, as described in [3] and Section 2.3. A non-symmetric forward and backward mask, as illustrated in Fig. 5, is hardwired into the FPGA and the image is translated under this mask, first in forward, then in backward direction. All 8 directions are processed in parallel. For the calculation of the distance value we use 5 bit integers. The results are clipped to 4 bits, allowing to combine all directions of a pixel into a single memory word suitable for the available hardware.

The forward and backward masks are similar, so that we only need a single hardware realisation. In both cases the data at the current position has to be compared to the minimum of the three corresponding pixels in the preceeding line. This intermediate result has to be compared with the neighbouring pixel which is already stored in the register, as shown on the bottom right of Fig. 5. The result is written to the internal Block RAM and is also stored in the register at the bottom right after being incremented by 2.

Upon initialisation and in the border region a multiplexer supplies a save value to the DT output.

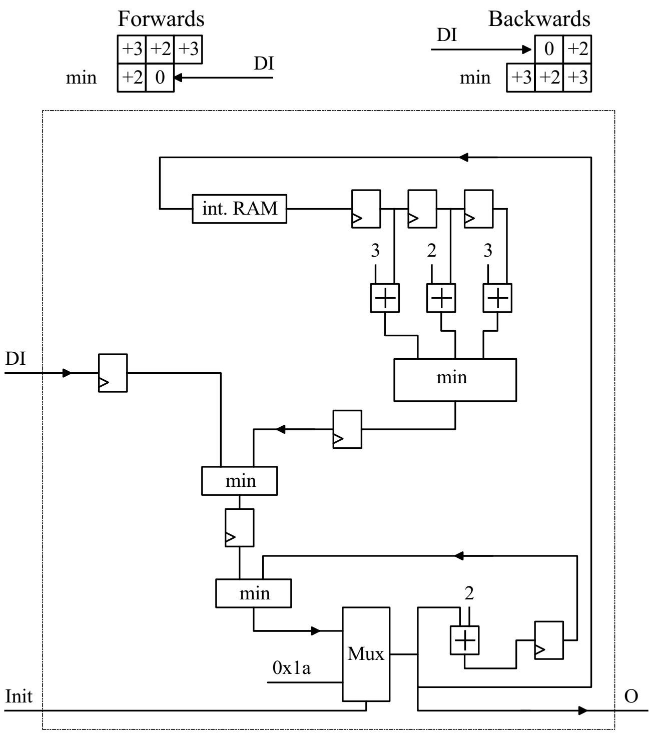


Figure 5: Pipeline of disctance transformation.

The alternative parallel approach [3] has a 16-fold resource requirement and is therefore not considered in the present work.

3.4 Control and Resources

The pipelined structure of the two preprocessing sections is depicted in Fig. 6 and Fig. 7. In the first phase, data is read on every clock cycle from the left RAM and entered into the pipeline. After the latency of the pipeline, given in Tab. 1, the result is written to the right RAM, again in every cycle. This process includes the transformation of a single input image into 8 feature images. During the second phase, data are read from the right RAM and written to the left RAM. All 8 orientations are processed in parallel.

Apart from the basic control of the preprocessing steps care has to be taken of initialisation of the various subsystems and of synchronisation with external modules like RAM and host interface.

The resource utilisation of the two pipelines for images of size 512^2 is given in Tab. 1. The size of the image only affects the use of internal Block RAM. The number of DT images and the precision are fixed and don’t have to be considered.

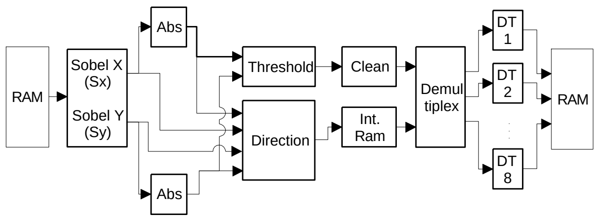


Figure 6: Pipeline 1 of preprocessing with forward distance transformation.

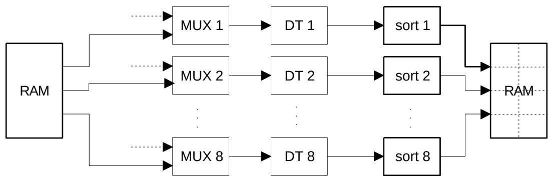


Figure 7: Pipeline 2 of preprocessing with backward distance transformation.

4 Architecture of Template Matching

In this section we describe in detail a pipelined, parallel approach for calculating the correlation for multiple templates. In some ways our pipelined approach is similar to the calculation of Sobel. The relevant data with regard to all multiple templates are stored in shift register arrays (SRAs) and all correlations of all templates are carried out simultaneously by pipelined adder trees. Depending on the number, size and shape of the templates varying amounts of FPGA resources are used. Their number could be reduced by optimization of adder trees or different calculation strategies as described in Section 4.3 and 5. With regard to the usage of FPGA resources it is important that the extension of the templates is not too large compared to the image size. It is advantageous if the templates are uniform, compact or concave. These attributes apply to templates corresponding to traffic signs. For the moment we restrict our calculations to 12 circles and 12 triangles with radii from 7 to 18 pixels. Paragraph 4.2 describes in detail the optimal handling of data of 8 DT images, so that no modules are in wait state.

Operation	Slices	Blk Ram	Delay
Sobel	$\approx$150	1	2×W+6
Abs	$\leq$ 8		1
Threshold	<8		1
Direction	<25		2
Clean	$\approx$270	1	4×W+7
Demult.	<20		2
8 DT	$\approx$8×110	2	1×W+2
Control	$\approx$340		-
$\sum$	$\approx$1700	4	7×W+21

Table 1: Resource requirements and latencies for preprocessing on XC2V3000 XILINX. W is the width of the image using 8-bit input data.

4.1 Parallel Pipelined Matching

To calculate the correlation of one template the following summations must be performed, see Equ. 1. First, the pixels of one DT image corresponding the template points have to be added up. Second, this has to be done 8 times, once for every DT image. Third, the sum of these 8 intermediate sums has to be calculated. For N templates this has to be done N times.

In our FPGA design all correlations of all templates are carried out simultaneously. This has the following effects: For each DT image we generate one shift register array (SRA) to have access to those pixels corresponding not only one template but also all templates, as seen in Fig. 8. Since the correlations are calculated on 8 DT images, we generate 8 of these SRAs as shown in Fig. 9. To each template one adder tree with access to all relevant SRAs is assigned.

Normally, each of the 8 SRAs will differ in extension depending on the shape of all templates. Due to this, the relative locations of the SRAs also change, as shown in Fig. 9. This implies that the input data of the SRAs must be read by different addresses of the DT images. The utilization of these SRAs is high if the templates have the above mentioned attributes.

The chamfer measure for each template is calculated with an arithmetic logic unit (ALU) consting of pipelined adder trees and threshold modules as indicated in Fig. 9 and Fig. 10. The ALU has parallel access to all DT pixels relevant for all templates. Because of this it could be checked in parallel if the chamfer measure is below all thresholds, see Equ. 1. No intermediate values are calculated and to be stored in the extern RAM. The adder tree we use has the special feature that registers after every x-th stage of adders can be included.

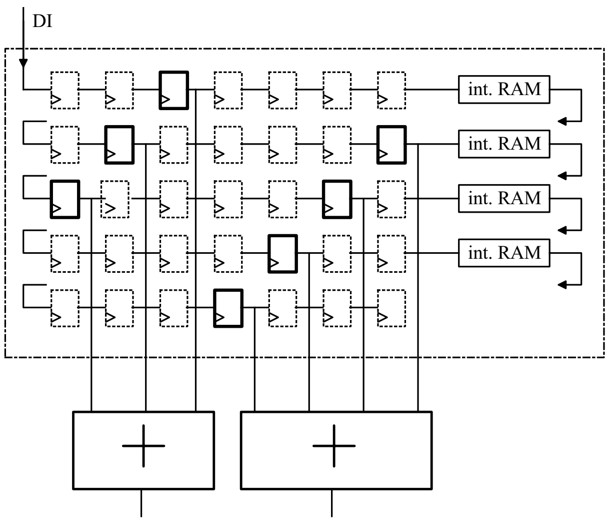


Figure 8: The SRAs are generated generically for each DT image depending on the shape of multiple templates. This is shown for two fragments of lines as suggested in the bold registers. These registers of each template are connected with the inputs of adder trees.

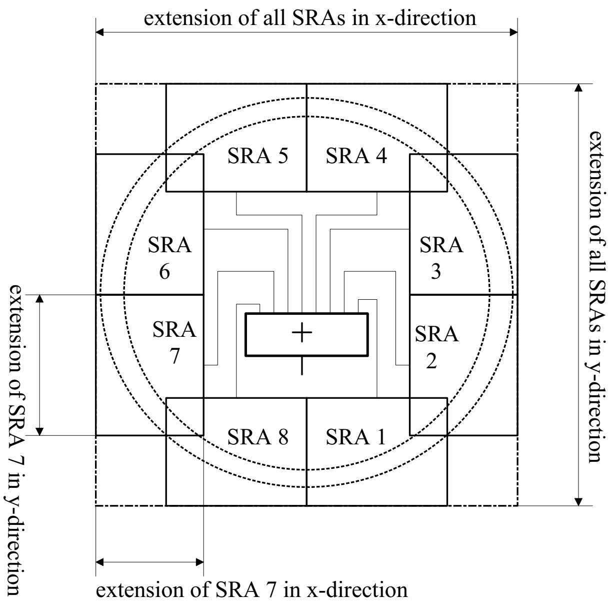

extension of all SRAs in x-direction
extension of all SRAs in y-direction
extension of SRA 7 in y-direction
extension of SRA 7 in x-direction
SRA 5
SRA 4
SRA 6
SRA 3
SRA 7
SRA 2
SRA 8
SRA 1
+

Figure 9: Generation of 8 shift register arrays (SRAs) at the example of 2 circle templates. The different extensions and locations of all 8 SRAs is shown.

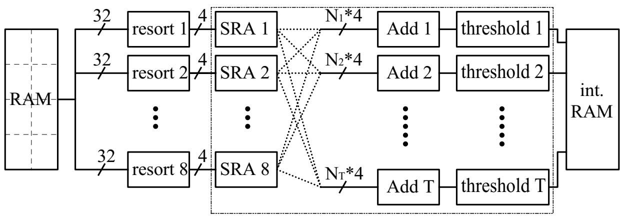


Figure 10: Data flow of template matching. The connections between SRAs and adder trees is optional.

The calculation strategy is similar to Sobel: the DT images are translated line by line under the hardwired templates. If one output of the thresholds is high, the data of the address counters and all threshold data will be stored in internal Block RAM as shown in Fig. 10.

4.2 Control

As described above we build one pipeline consisting of 8 SRAs and the ALU. The SRAs can be filled with DT pixels in a way that each SRA receives one input data per clock cycle. This is done by storing the 8 DT images in different sections in the extern RAM and 8 neighboured DT pixels in every address as described in Section 3.4. We re-sort the data after reading as we sorted the data before storing after DT. So in every 8-th clock cycle each sorting module is loaded with 8 $\times$ 4 bit DT pixels. Thus the pipeline works highly efficiently and there is no unused logic.

To fill the pipeline is somewhat difficult. Because of the different extensions of SRAs, we need different numbers of cycles to fill them with pixels. First, we start to fill the SRA with the biggest extension. Then the next SRAs are filled simultaneously in time, each one shifted by one cycle. When all 8 SRAs are filled only correct DT pixels are in the pipeline. Since we stored 8 neighbouring DT pixels in one address it could happen that the "real" address of one pixel is not a multiple of 8. This additional offset is compensated with a few shift registers inserted between the re-sort and SRAs modules.

Using this strategy we can guarantee that all SRAs are loaded in every clock cycle with one input pixel. This means that the pipeline is never stalled and all registers can always be clock enabled (CE). Thus high fan out of CE signals can be prevented saving a noticeable amount of routing network.

While the pipeline is filled with data no results of possible matched templates are stored. At the border no special processing is done. The calculation is contiued over the border and any results that occur are discarded. The subsequent verification of these detection results is conducted on the PC. From the addresses the x and y coordinates and from the threshold data the types of templates will be determined.

4.3 Optimizations

First, it is possible to remove the unused registers from the SRA and to replace them by internal "distributed" RAM. This is only useful if they are neighboured and connected.

Second, resources of adders could be saved if the commonalities of topological similar templates are taken into account, which is described by means of examples in [7]. Calculating the correlation for one template will no longer be done by one adder tree alone. Parts of adder trees are shared.

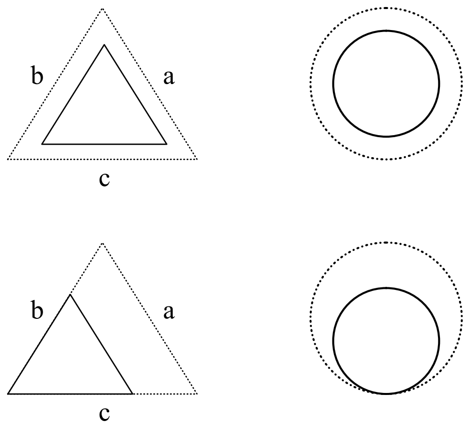


Figure 11: Above, templates concerning to central points are given. The translated templates are shown below. The topological similar triangles merged best.

Often the representation of templates is given with regard to the central points as shown at the top of Fig. 11. No overlapping will then happen within the template classes of circles and triangles. To increase the amount of templates with overlap, we propose translations as indicated at the bottom in Fig. 11. After translation the triangles merge best and the gain of FPGA resources for adders is high. But it has to be taken into account that the translation of templates also effects the extension of the SRAs. For the triangles, the extensions of the SRAs corresponding to line c will shrink, to line b, they will stay the same and to line a, they will swell. The way the circles are translated as shown in Fig. 11 only little overlap is gained. The extensions of the upper four SRAs will increase and the lower will decrease. Supposedly there is no overall gain of resources or even more resources may be needed. Gain of overlap will also be reached by putting the templates of different classes over each other. It is unclear how to do this in an exact way. We think that it is necessary to put all this into a systematic approach and to formulate an integer programming problem.

5 Further Work

The gain of FPGA resources achieved by merging similar templates and other optimizations as described in Section 4.3 have to be accomplished.

Furthermore, the design for camera readout, preprocessing and matching has to be integrated in one design as we did for other image processing applications as described in [5].

The second pipeline for preprocessing, the backward transformation of DT, the sorting of pixels (see Fig. 7), and the pipeline for template matching (see Fig. 10) could be combined in one pipeline. Storage, sorting and resorting of data become redundant. The different read-out addresses caused by the different extensions of SRAs must be compensated by internal Block RAM between the output of the DT modules and input of the SRAs. The biggest SRA needs no Block RAM, the smallest needs most.

Generally it is possible to build one big pipline of preprocessing and matching if are used parallel DTs. In this case the template matching must be calculated in forward direction which requires a turning of the templates. To be sure, we would only concentrate on this approach, if the use of FPGA resources is no sensitive issue.

Using images of a size more than 512×512 pixels, internal Block RAM could be saved by calculating the correlation not on the whole DT images, but on stripes of them. The maximum size of one stripe could be 512 plus the extension of the smallest SRA in x direction. The overlap of two neighbouring stripes has also been taken into account, since at the border of the stripes no results will be found.

To increase the number of templates of the same shape but of different scale, orientation or skew it is advantageous to put only the basic templates in our matching method and calculate the correlation of the others by transforming the images. Hardware implementations for 2 times down-sampled and rotated images are given e.g. in [12]. In this case the image must be transformed before the preprocessing, which means that the total computing time multiplies. Thus it gets clear that if the FPGA resources are the bottleneck, the trinagles up and down are taken into account only once.

Another method to increase the number of templates is to use FPGAs with submillisecond reconfiguration time. The preprocessing will then be done only once and the matching will be retried. The possibility of partial reconfiguration reduces the reconfiguration time significantly. In this case the best strategy could be to fix the SRAs and the adder trees and to replace the connections between them only.

6 Results

The simulation of preprocessing, template matching and combined preprocessing and template matching to top 6 circles has been done successfully using CHDL [10].

The Place and Route (P&R) of the designs is performed with P&R tools (version 3.1) from XILINX. For XC2V3000 FPGA results for preprocessing (PP) only and combined preprocessing and template matching (TM) are given for 12 circles and 12 triangles in Tab. 2. Also, the expected calculation times for 512^2 images are shown.

	Slices	Blk RAM	Freq. [MHz]	Time [ms]
PP	1712 (11%)	4	96	5.5
PP+TM 12	10443 (72%)	20	82	9.6
PP+TM 24	14334 (99%)	26	57	14.8

Table 2: Results of P&R for preprocessing and template matching and the expected calculation times for 512^2 images on XC2V3000 FPGA.

The designs have so far been tested on two different demonstrator systems. First, grabbing, preprocessing and matching of 6 circles runs on three μEnable-I boards [8], each on one board. Second, the combined design of preprocessing and matching of 12 circles runs on one μEnable-II board.

7 Conclusions

We presented a novel FPGA-based implementation for (edge-based) object detection in images. The various logical entities of preprocessing (edge detection, noise edge removal and distance transform), were tightly integrated into two large pipelines. Template matching with distance images was multiplexed to account for multiple edge orientations, and implemented in highly parallel fashion. As demonstrated in simulations and actual tests on various FPGA boards, the followed approach resulted in high processing rates with efficient use of resources.

References

[1] D.M. Gavrila, “Multi-feature Hierarchical Template Matching Using Distance Tranforms”, in *International Conference on pattern Recognition*, pp. 439-444, Brisbane, 1998.

[2] D.M. Gavrila, V. Philomin, “Real-Time Object Detection for ”Smart” Vehicles”, in *Proc. Int. Conf. on Computer Vision*, pp. 87-93, 1999.

[3] G. Borgefors, “Distance Transformations in Digital Images” *Computer Vision, Graphics, and Image Processing 34*, pp. 344-371, 1986

[4] T. Ikenaga, T. Ogura, “Real-Time Morphology Processing Using Highly parallel 2-D Cellular Automata CAM ^{2}”, *IEEE Transactions on Image Processing*, Vol. 9, No. 12, Dec. 2000.

[5] P. Dillinger, S. Hezel, H. Lauer: “FPGAs zur Echtzeit-Bildverarbeitung mit 1D/2D-FIR-Filteroperationen”. *Image Processing and Machine Vision, VDI Berichte 1572, Düsseldorf* , pp. 213-218, 2000.

[6] T. Kean, A. Duncan, “A 800 Mpixel/sec Reconfigurable Image Correlator on XC6216”, in *Proceedings of FPL '97*, pp. 382-391, 1997.

[7] J. Villasenor, B. Schoner, K. Chia, and C. Zapta, et.al., “Configurable Computing Solutions for Automatic Target Recognition”, *Proceedings of the 1996 Symposium on FPGAs for Custom Computing Machines*, pp. 70-79 , April 1996.

[8] O. Brosch, P. Dillinger, K. Kornmesser, A. Kugel, R. Männer, M. Sessler, H. Simmler, H. Singpiel, S. Rühl, R. Lay and K.-H. Noffz, L. Levinson, “MicroEnable - A Reconfigurable FPGA Coprocessor”, *4th Worksh. on Electronics for LHC Experiments”*, pp. 402-406, Rome, Italy, 1998.

[9] S. Hezel, R. Männer, “Schnelle Berechnung von 2-D FIR-Filteroperationen mittels FPGA-Coprozessor mEnable”, in W.Förstner e.a. (Hrsg.), *Mustererkennung 1999, 21. DAGM Symposium*, Springer, pp. 250-257, Sept. 1999.

[10] K. Kornmesser, A. Kugel, R. Männer, "The FPGA Development System CHDL", *Proceedings of the 2001 Symposium on FPGAs for Custom Computing Machines*, April 2001.

[11] A. Kugel, "RACE-1 - A PCI-64 based High Performance FPGA Co-Processor", http://www-li5.ti.uni-mannheim.de/fpga/race/, Jan. 2002.

[12] R.D. Turney, C.H. Dick, "Real Time Image Rotation and Resizing, Algorithmns and Implementations", www.xilinx.com, 1999.