Reconfigurable Shape-Adaptive Template Matching Architectures

Jörn Gause[1], Peter Y.K. Cheung[1], Wayne Luk[2]

[1]Department of Electrical and Electronic Engineering, Imperial College, London SW7 2BT, England.
[2]Department of Computing, Imperial College, London SW7 2BZ, England.

Abstract

This paper presents reconfigurable computing strategies for a Shape-Adaptive Template Matching (SA-TM) method to retrieve arbitrarily shaped objects within images or video frames. A generic systolic array architecture is proposed as the basis for comparing three designs: a static design where the configuration does not change after compilation, a partially-dynamic design where a static circuit can be reconfigured to use different on-chip data, and a dynamic design which completely adapts to a particular computation. While the logic resources required to implement the static and partially-dynamic designs are constant and depend only on the size of the search frame, the dynamic design is adapted to the size and shape of the template object, and hence requires much less area. The execution time of the matching process greatly depends on the number of frames the same object is matched at. For a small number of frames, the dynamic and partially-dynamic designs suffer from high reconfiguration overheads. This overhead is significantly reduced if the matching process is repeated on a large number of consecutive frames. We find that the dynamic SA-TM design in a 50 MHz Virtex 1000E device, including reconfiguration time, can perform more than 28,000 times faster than a 1.4 GHz Pentium 4 PC when processing a 100×100 template on 300 consecutive video frames in HDTV format.

1 Introduction

The development of multimedia technology and associated standards like MPEG-4 for coding of audio-visual objects in multimedia applications [1] and MPEG-7 for description and search of audio and visual multimedia content [2] leads to new types of video processing algorithms and therefore new challenges for their hardware implementation. Comprehensive acceptance of new multimedia services and applications depends on the availability of inexpensive, compact hardware delivering the high performance required. In addition to very high processing demands, many multimedia processing algorithms tend to be characterised by a decreasing regularity and predictability of operations. Typical examples are algorithms to process arbitrarily shaped multimedia objects: the computations to be performed need to be adapted to the size and the shape of the object. This calls for architectures with increased flexibility at run time [3].

In this paper, a shape-adaptive template matching (SA-TM) method to retrieve arbitrarily shaped objects within images or video frames is proposed. The algorithm is truly object-oriented as it uses only the object of interest as template, not the background pixels, and it does not divide the template into a number of square blocks in accordance with future generation multimedia techniques [4]. Software solutions which could provide the flexibility to adapt to different templates are too slow to allow real-time processing at video frame rate, whereas an ASIC implementation is impossible due to the infinite number of sizes of template and search frame. Hence, the use of a reconfigurable computing architecture, such as SONIC [5] is proposed to implement a fast and flexible SA-TM design, as shown in Figure 1.

The purpose of this paper is to investigate reconfigurable computing strategies regarding their suitability for implementing the SA-TM method as an example of a typical multimedia algorithm. A static design, which can match templates stored in off-chip memory of all possible shapes and sizes within a video frame using the same FPGA configuration is presented in this paper. A partially-dynamic design which uses on-chip memory, available in most recent FPGAs, to store the template is also introduced. While the circuit to compute the algorithm is static, the template can be updated by reconfiguring the memory portions of the device. In addition, this paper presents a dynamic design, where the configuration data are completely adapted to the shape and size of the template object used, resulting in significant area savings. A generic systolic array architecture provides the basis for the implementation of the three reconfigurable SA-TM designs. We compare their area usage and computation time, including reconfiguration and recompilation overheads. It is shown that the suitability of a particular design regarding total execution time of the algorithm strongly depends on the number of consecutive frames the operations are carried out on.

Section 2 presents background information and previous work on multimedia search and retrieval strategies and reconfigurable computing, including run-time reconfiguration (RTR). Section 3 describes the shape-adaptive template matching method applied in this paper. A systolic array for SA-TM is proposed in Section 4, which is used for a dynamic, a static and a partially-dynamic realisation approach. These designs are presented in Section 5. In Section 6 the FPGA implementations results for all three designs are discussed. Finally, a conclusion follows in Section 7.

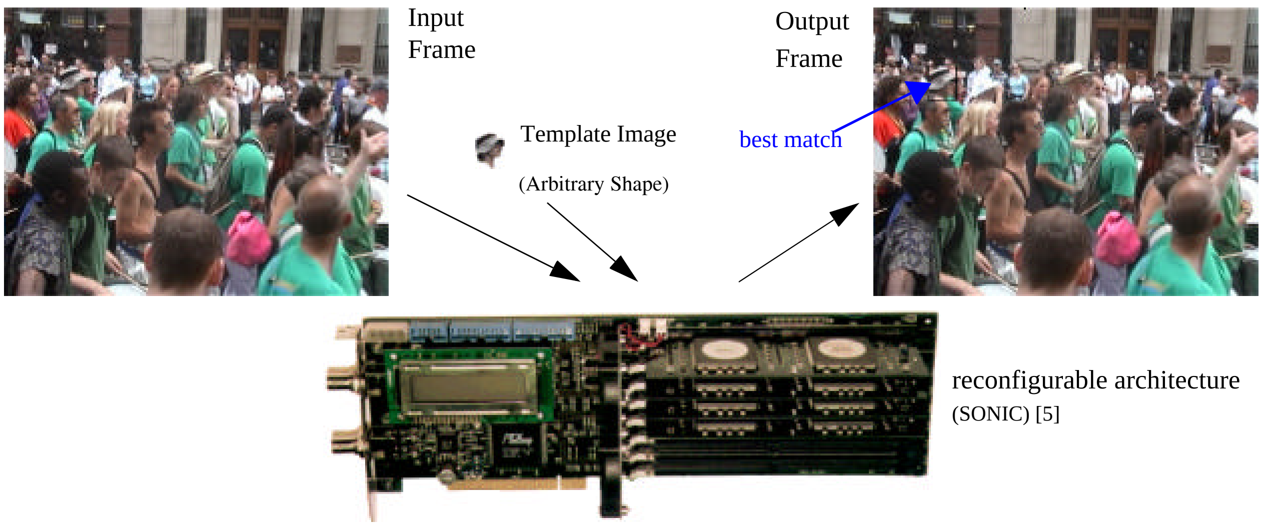

Figure 1 Reconfigurable Computing for SA-TM

2 Background

Multimedia search and retrieval have become an active research field due to the increasing demand that accompany many new practical applications. More and more video data are generated every day. Amongst this large amount of multimedia information, searching for specific video sections or objects and retrieving them is a difficult task. The traditional approach of fast-forwarding of video and looking at the screen to find interesting information is time-consuming and labour-intensive. Ideally, video will be automatically annotated as a result of machine interpretation of the semantic content of the video data. However, given the state of the art in computer vision, such sophisticated data extraction may not be feasible in practice [6].

Video object retrieval is concerned with returning similar video clips to a user given a visual object as query. The recognition and retrieval can be feature-based or template-based. Colour is most frequently used for feature-based retrieval, as the retrieval algorithms using colour are characterised by regular operations and data accesses. Colour indexing and retrieval often involves the use of colour histograms which record the number of pixels in an image for each colour [6]. However, simple histograms do not take the location of a pixel with a particular colour into account. Template matching is a classical technique for locating the position of a given small subimage inside a large image. It has been frequently used in the applications of object detection, image registration and pattern recognition. The matching process involves shifting a template image over a search area, measuring the similarity between the template and the current search area, and locating the best match position. Due to their regularity, template matching algorithms are suitable for pipelined processing in a systolic array. Various basic systolic array architectures with different degrees of parallelism are presented in [7]. However, template matching entails great computational complexity for large search areas and templates. Hence, many practical video and image retrieval systems which are generally software based, use colour histograms to search for a query image. Often, the query image has to be the same size as the search image or video frame, although the user may only be interested in a particular object within an image. The use of background pixels, not belonging to the object of interest, can therefore lead to unwanted results.

Reconfigurable computing, based on Field Programmable Gate Arrays (FPGAs) as processing devices, has been identified to be well suited to deal with the requirements of providing flexible, high-speed processing, since it combines the advantages of software and application-specific hardware. It allows user-level programmability at a low level and facilitates general purpose computing due to its reconfigurability. Thus, many applications can use the same hardware [13], [8].

Dynamic or run-time reconfiguration (RTR) of FPGAs is recognised as an advanced application area within reconfigurable computing. However, a lot of research still has to be accomplished to fully understand and evaluate RTR and quantify the trade-offs of run-time reconfigurable devices and systems [8]. Currently only a small subset of available FPGAs are capable of being reconfigured in this way, but there is a growing trend in the industry to provide dynamically reconfigurable devices with varying degrees of configuration flexibility [9]. Dynamically reconfigurable devices are characterised by their ability to continue to operate without interruption while sub-sections of the array logic are being reconfigured. The authors of [10] distinguish between two modes of configurability: *static* -- where the configuration data of the FPGA is loaded once, after which it does not change during execution of the task, and *dynamic* -- where the FPGA's configuration may change at any moment. The most cited motivations for using dynamic reconfiguration are the acceleration of algorithms that might otherwise be implemented on general-purpose computers, and the opportunity to increase effective logic capacity of programmable devices by mapping only active logic to FPGA resources at any given time. It is predicted in [11] that the importance of dynamically reconfigurable logic will increase as FPGAs become larger. This statement is based on the observation that with more and more circuits present in a single chip, there is a reduced probability of them all being required to operate at the same time. Three FPGA reconfiguration strategies (compile-time, run-time, and partial run-time reconfiguration) are evaluated in [12], based on a systolic array implementation of a scalar quantiser. It is shown that compile-time reconfiguration gives the best area-time product for the application used, whereas the suitability of run-time reconfiguration strongly depends on the number of reconfigurations. However, it is concluded that the results are application and technology dependent.

In [13], the suitability of using reconfigurable computing for implementing computer vision algorithms of different levels of regularity has been investigated. This includes a systolic correlation that can be used for template matching. However, the design presented there is not shape-adaptive, and results are given only for relatively small and square masks (3×3) and images (512×512). Video is not considered. Configurable computing solutions for automatic target recognition are presented in [14]. Here the templates are binary and have a size of 16×16 pixels, whereas the search image is 128×128. The templates are mapped onto the FPGA as simple adder trees, and the image pixels are shifted through and added at positions where the template bit is '1'. Hence, the FPGA configuration can be optimised to adapt to the template characteristics. It is shown in [15] that dynamic reconfigurability is well suited to adapt to shape-adaptive image and video processing algorithms if the reconfiguration overhead can be kept small. However, the example used consisted only of a limited number of possible configurations since only different object shapes within an 8×8 block were considered.

3 Shape-Adaptive Template Matching

The aim is to find a template object of arbitrary shape and size within a search image or video frame of any size using a reconfigurable computing architecture, as shown in Figure 1.

The search image or frame consists of $W \times H$ pixels. The template object consisting of p opaque pixels can have any shape. It is given by its bounding box of size $w \times h$, that is the smallest rectangle surrounding the object as shown in Figure 2. Within this bounding box, each pixel contains one mask bit which is '1', if the pixel belongs to the object, or '0' otherwise, as defined in MPEG-4 [16].

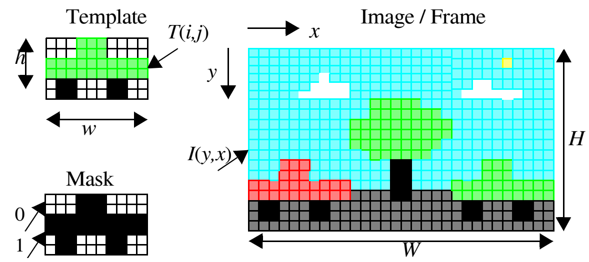

Figure 2 Template object and search image

The template is shifted through every possible location of the image that can contain the entire template and compared to the respective subimage of the same size. There are $(W-w+1)\cdot(H-h+1)$ of those locations. Only pixels of the subimage that correspond to pixels belonging to the template object are taken into account.

Simulations of various correlation and histogram based matching and image retrieval algorithms have been carried out in software to find a suitable and easily implementable similarity measure. The sum of absolute distances (SAD), carried out on luminance pixel values, was then chosen due to good matching results and because of its simple structure.

For all $(W-w+1)\cdot(H-h+1)$ possible positions (y,x) of the template within the image, calculate

$$SAD(y,x) = \sum_{i=0}^{h-1}\sum_{j=0}^{w-1}\left(|I(i+y,j+x)-T(i,j)|\cdot M(i,j)\right), \quad (1)$$

where I is the pixel value of the search frame, and T and M are pixel value and mask bit of the template object, respectively. For one frame, $(W-w+1)\cdot(H-h+1)\cdot w\cdot h$ absolute distance calculations need to be computed. A match is found at a position (y,x) where $SAD(y,x)$ is minimum and also smaller than a certain threshold determined by the user. In this paper, only the calculation of the SAD values is considered.

4 Systolic Array for SA-TM

Since in practice video data are often streamed in a horizontal raster-scan fashion (line after line), we assume that one pixel of the search frame becomes available with each clock cycle [5]. As every pixel value of the search image or video frame is read only once, it would be useful to perform in parallel all computations where this pixel value is required, so that no input data need to be stored.

A simple example of an SA-TM process is shown in Figure 3. For various matching positions of the template on the search image, the absolute difference (AD) computations to be performed for that position and in which clock cycle, and the SAD these ADs contribute to, are shown. In this example, the pixel $T(0,1)$ does not belong to the object, that is, it is transparent. Therefore, this pixel value does not contribute to the computations.

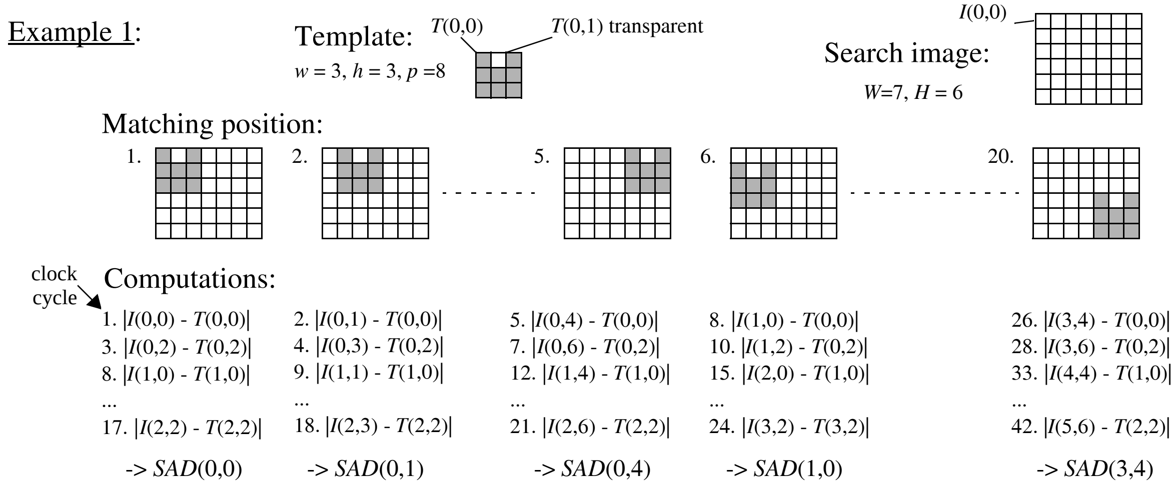

Figure 3 Example 1 of SA Template Matching

Considering that pixel $I(0,0)$ is available in the first clock cycle, pixel $I(0,1)$ in the second cycle, and so on, if a line-by-line raster-scan fashion is used, some computations can be carried out in parallel. In the first step, only $I(0,0)$ is available, and using $T(0,0)$ the first AD contributing to $SAD(0,0)$ can be calculated. In the second cycle, $I(0,1)$ becomes available and the first AD for $SAD(0,1)$ is computed using $T(0,0)$. As $T(0,1)$ is transparent, no further computation can be carried out in this cycle. In cycle 3, $I(0,2)$ is used to perform two computations in parallel: $|I(0,2) - T(0,0)|$ which contributes to $SAD(0,2)$, and $|I(0,2) - T(0,2)|$ which yields the second AD for $SAD(0,0)$. In cycle 4, when $I(0,3)$ becomes available, no contribution to $SAD(0,0)$ is calculated, as this image pixel is not necessary for the computation of $SAD(0,0)$. Not until cycle $W \cdot (h-1) + w = 17$ can all possible eight pixels of the template object be used in parallel.

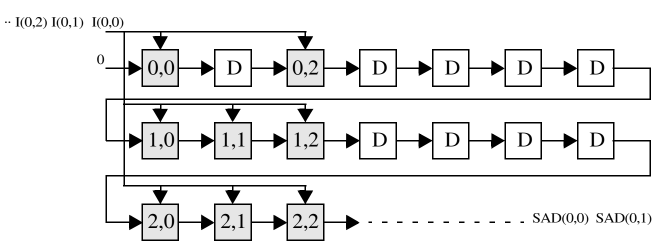

Figure 4 Signal Flow Graph (SFG) for SA-TM Example 1

Example 1 can be represented by the Signal Flow Graph (SFG) shown in Figure 4 which is proposed as a general SFG for SA-TM. The nodes marked <*i,j*> represent the absolute distance (AD) computations $|I(y,x) - T(i,j)|$. The pixel values $I(y,x)$ are broadcasted sequentially to all of those processing elements (PEs), that is, all possible AD computations for a particular image pixel are carried out in parallel. Note that some of those computations (for example, $|I(0,0) - T(0,2)|$) do not contribute to any valid result. The AD results are added to the intermediate sums coming from the left and the new sum is registered and shifted out at the right of the PE. The delay nodes <D> are used as shift registers and are required at transparent pixels within the bounding box of the template objects and at all $(W-w) \cdot (h-1)$ places where no AD contribution for a particular SAD result is produced. For example, the intermediate sum $|I(0,0) - T(0,0)| + |I(0,2) - T(0,2)|$ which is computed in the third cycle in PE <0,2> needs to be at the left input of PE <1,0> when $I(1,0)$ is broadcasted to that node, that is in cycle $W + 1 = 8$ when the first image or frame line has been completed. Note that if the pixel values of the search frame can be streamed in a vertical fashion (column after column), registers can often be saved as in reality video frames are more wide than high.

In example 1, the first valid result, SAD(0,0), is at the output of node <2,2> after $W \cdot (h-1) + w = 17$ cycles, followed by $SAD(0,1)$ after the next cycle. There will still be invalid results at the output after each template row, as certain SAD positions (for example, $SAD(0,5)$, $SAD(0,6)$) are not defined because at that position the template cannot cover the subimage. For instance, in example 1, there are $W \times \mathrm{H} = 42$ pixels in the search frame (and therefore 42 cycles are required to produce all results), but only 20 SAD values are produced as there are only 20 positions where the template fits completely into the search frame. In the 42nd (and last) cycle, when the last image pixel of that frame $I(5,6)$ is broadcasted, the 20th result $SAD(3,4)$ is completed by adding $|I(5,6) - T(2,2)|$ to the intermediate sum in PE <2,2>.

Based on the SFG shown for example 1 we propose the following generic systolic array adapted to the shape of the template object. Each pixel belonging to the template object is represented by a PE where the AD computation using the value of that pixel is performed. The structure of a PE is shown in Figure 5.

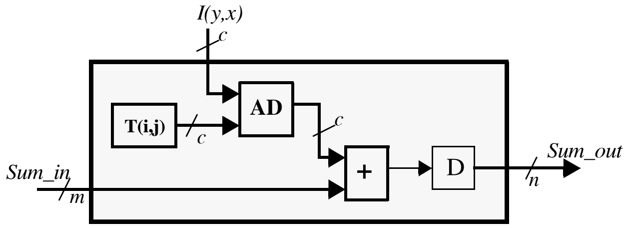

Figure 5 Structure of PE <i,j> for SA Template Matching

The word length of *I(y,x)* is *c*, and usually *c*=8. The pixel values of the template object *T(i,j)* have the same word length as *I(y,x)*. The values are constant for a particular PE and can be stored in ROM within the PE as shown, or come from outside the PE. The word width of *Sum_in* and *Sum_out* depend on the position of the PE in the SFG. For the first PE, Sum_in is 0, therefore *m* can be 0. Variable *n* needs to be at least the maximum of *m* and *c*, in case of the first PE that is *c*. Further along the SFG, *m* and *n* will increase, as more bits are required to represent the intermediate sums. The maximum absolute distance (AD) for one PE is 2^c-1, the maximum intermediate sum of *k* of those ADs is $(2^c - 1) \cdot k$ for one PE, which requires $\lceil log_2((2^c - 1) \cdot k) \rceil$ bits to be fully represented. The intermediate sum is then stored in a register.

The area of a PE consists of a constant part for the AD module and a variable part which grows with the word length *n* of the output and can therefore be calculated as:

$$A^{PE}(n) = a \cdot n + b = a \cdot \lceil log_2((2^c - 1) \cdot k) \rceil + b \ , \qquad (2)$$

where *a* and *b* are constants.

The logic resources required to implement *p* PEs can then be calculated as

$$A^{PE} = \sum_{k=1}^{p} A^{PE}(k) = \sum_{k=1}^{p} (a \lceil log_2((2^c - 1) \cdot k) \rceil + b) \ . \qquad (3)$$

To calculate A^{PE} explicitly as a function of *p* the following estimation is derived from (3), given that $2^c \gg 1$:

$$A^{PE} \approx a \sum_{k=1}^{p} (c + \lceil log_2(k) \rceil) + bp \ , \qquad (4)$$

which can be simplified further to

$$A^{PE} \approx a \sum_{k=1}^{p} (\lceil log_2(k) \rceil) + (ac + b)p \ . \qquad (5)$$

The sum term can be substituted as follows:

$$\sum_{k=1}^{p} (\lceil log_2(k) \rceil) = \sum_{k=1}^{2^q} (\lceil log_2(k) \rceil) + \sum_{k=2^q+1}^{p} (\lceil log_2(k) \rceil) \ , \qquad (6)$$

$$\text{with } q = \lfloor log_2(p) \rfloor \ , \qquad (7)$$

resulting in

$$\sum_{k=1}^{p} (\lceil log_2(k) \rceil) = \sum_{k=1}^{q} (2^{k-1} \cdot k) + \sum_{k=2^q+1}^{p} (q+1) \qquad (8)$$

$$= 2^q(q-1) + 1 + (p - 2^q)(q+1) \qquad (9)$$

$$= p \cdot (q+1) - 2^{q+1} + 1 \ . \qquad (10)$$

Hence, (5) leads to

$$A^{PE} \approx a(p \cdot (q+1) - 2^{q+1} + 1) + (ac + b)p \ . \qquad (11)$$

All pixels within the template bounding box but not belonging to the object are registers used to delay the intermediate sums. PEs and registers are arranged according to the shape of the template object. If a pixel belonging to the object is followed by a pixel not belonging to the object in the same line, the PE representing the object pixel is followed by a register, and vice versa. After each line of PEs, additional *W-w* registers are required for all but the last line of the template to store the intermediate sums until a valid input for that sum becomes available.

The register area is composed of the registers used for the *wh - p* transparent pixels within the bounding box of the template (A_D^{Reg1}) and of the $(W - w) \cdot (h - 1)$ registers used to delay the intermediate sums outside the template area (A_D^{Reg2}).

As the exact value for A_D^{Reg1} depends on the position of the transparent pixels within the template bounding box, the average word length of the PEs is used to determine the average register size which is then multiplied by the number of transparent pixels:

$$A_D^{Reg1} \approx \frac{wh - p}{p} \sum_{k=1}^{p} \lceil log_2((2^c - 1) \cdot k) \rceil \ , \qquad (12)$$

which can be estimated as

$$A_D^{Reg1} \approx \frac{wh - p}{p} (p \cdot (q+1) - 2^{q+1} + c \cdot p) \ , \qquad (13)$$

using the same value for *q* as in (7).

A_D^{Reg2} is estimated as

$$A_D^{Reg2} \approx (W - w) \sum_{k=1}^{h-1} \lceil log_2((2^c - 1) \cdot k \cdot w) \rceil \ , \qquad (14)$$

resulting in

$$A_D^{Reg2} \approx (W - w)((h - 1) \cdot (q_h + log_2(c \cdot w)) - 2^{q_h} + 1) \ , \qquad (15)$$

$$\text{with } q_h = \lfloor log_2(h - 1) \rfloor + 1 \ . \qquad (16)$$

To summarise, in an efficient systolic array solution for the presented SFG for SA-TM, the following points need to be satisfied in order to search for a w×h template with p pixels belonging to the object of interest, in a $W \times H$ search frame:

- p PEs are required to implement the AD computations and summation of intermediate results
- the PEs are arranged in the same way as the pixels belonging to the template object; the $wh - p$ gaps representing transparent pixels are filled with registers
- for horizontal raster-scan, after each, but the last, row of the PE array, W-w registers are added in the computation flow to store the intermediate sums until the next search frame pixel contributing to a particular SAD becomes available
- the kth PE in the computation flow $1 \le k \le p$ requires a $\lceil log_2((2^c - 1) \cdot k) \rceil$ bit adder
- the size of each register is the same as the output size of the previous PE in the computation flow

5 Reconfigurable design strategies for SA-TM

The following design approaches for a systolic array for SA Template Matching can be distinguished. As all PEs have the same structure, apart from different word length, their function can be changed to use a different number of pixel values and/or a different similarity measurement in the future.

5.1 Dynamic design

In this approach, the device is reconfigured for every possible template size and shape, and for every possible search frame size. This generally includes the re-compilation of the design code, as there are an infinite number of solutions. As the template can be part of the configuration data and word lengths can be optimised, an efficient systolic array solution as described above can be achieved. A PE as shown in Figure 5 can be used with the template pixel value stored in on-chip memory. As one input of the AD computation is constant, the AD module can be substituted by a look-up table (LUT) storing the AD value for each value of the incoming frame pixel $I(y,x)$. In addition to the p PEs, $w \cdot h - p + (W - w) \cdot (h - 1)$ registers are required.

The area A_D for the dynamic design consists of the area used by the PEs (A_D^{PE}) and the area required for the registers ($A_D^{Reg1} + A_D^{Reg2}$):

$$A_D = A_D^{PE} + A_D^{Reg1} + A_D^{Reg2} , \qquad (17)$$

and can be estimated using (11), (13) and (15), respectively.

The total execution time T_D for the dynamic design to find a template object in N video frames consists of the computation time $T_D^{computeN}$ to calculate the SA-TM operations, the reconfiguration time T_D^{reconf} to update the FPGA configuration for a new template, and the compilation time $T_D^{compile}$ which is required in most cases as the number of template objects and therefore the number of different device configurations is unlimited:

$$T_D = T_D^{computeN} + T_D^{reconf} + T_D^{compile} . \qquad (18)$$

The execution time for N frames can be calculated as

$$T_D^{computeN} = N\frac{WH}{f_D} , \qquad (19)$$

where f_D is the clock frequency the circuit can run at. As a systolic array is pipelined by each PE, f_D is determined by the critical path through the slowest PE, if additional FPGA routing delays are neglected.

The reconfiguration time and compilation time depend on the size of the circuit to be implemented. The reconfiguration time is generally proportional to the number of resources to be reconfigured, but also depends on the device used and the reconfiguration strategy. Compilation time depends on the hardware and software used to translate and map the code and is hard to estimate.

5.2 Static design

Due to long reconfiguration and recompilation overheads, the dynamic design approach is expected to be useful only if the a template object is searched for within a large number of video frames of the same size. As an alternative, a static design is proposed, where the configuration of the FPGA is not changed when a new template is used. As the number of different search frame sizes and template shapes and sizes is unlimited, only a subset of all solutions can be implemented.

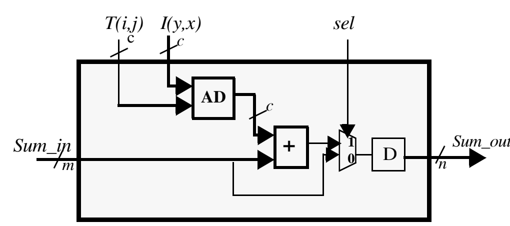

Figure 6 PE structure for static design

If the size of the search frame is fixed, the following solution is possible. The PEs have to be modified, as shown in Figure 6, so that the template pixel values $T(i,j)$ come from external memory, and a multiplexer controlled by a further input signal *sel* needs to be added to select between performing an addition, if the respective pixel belongs to the template object, or otherwise just delaying the signal. All delay elements D have to be substituted with those modified PEs. By changing the memory content, different templates can be searched for. The word length n of the kth PE output is $n = \lceil log_2((2^c - 1) \cdot k) \rceil$, with $1 \le k \le W \cdot H$, in order to cover for all possible template sizes.

The area A_S for the static design which consists of WH PEs is calculated using (2) as

$$A_S = \sum_{k=1}^{WH} A_S^{PE}(k) = \sum_{k=1}^{WH} (a_S \lceil log_2((2^c - 1) \cdot k) \rceil + b_S) \quad , \quad (20)$$

and hence estimated according to (11) as

$$A_S \approx a_S(p \cdot q_{WH} - 2^{q_{WH}} + 1) + (a_S c + b_S)p \quad (21)$$

with $q_{WH} = \lfloor log_2(WH) \rfloor + 1$.

The execution time T_S for the static design to perform the SA-TM algorithms on N frames can be calculated as

$$T_S = T_S^{computeN} = (N+1)\frac{WH}{f_S} - 1 \quad . \quad (22)$$

Advantages of this design are that no recompilation of the design code or reconfiguration of the device are necessary for a constant search frame size as the template is stored off-chip. However, the large, external RAM to store all possible *WH* template pixels and mask bits are expected to make the design slower and more complex than an optimal design. In addition, for large frame sizes the number of I/O pins required for all *WH* template pixels is extremely large. Another disadvantage is that in all cases when $p < WH$ (template smaller than search frame), area is wasted because some of the PE logic is unused.

5.3 Partially-dynamic design

To combine the advantages of the dynamic and the static design, a third design is proposed. The difference to the static design is that this design stores the template pixels and mask bits in on-chip memory available on most FPGAs. To change the template, only a reconfiguration of the memory parts is necessary, the rest of the circuit remains the same. As the template is dynamic while the circuit remains static, this design is called partially-dynamic. The structure of a PE for this design is shown in Figure 7. Both template pixel value $T(i,j)$ and mask bit $M(i,j)$ for that pixel are stored within the PE. If $M(i,j)$ is '1', that is, if the pixel belongs to the template object, the absolute distance of $I(y,x)$ and $T(i,j)$ is added to the incoming intermediate sum *sum_in*. Otherwise, *sum_in* is shifted through and registered.

As the partially-dynamic design, like the static one, consists of *WH* PEs, the area can be estimated in the same way:

$$A_{PD} \approx a_{PD}(p \cdot q_{WH} - 2^{q_{WH}} + 1) + (a_{PD} c + b_{PD})p \quad , \quad (23)$$

using q_{WH} as in the static design.

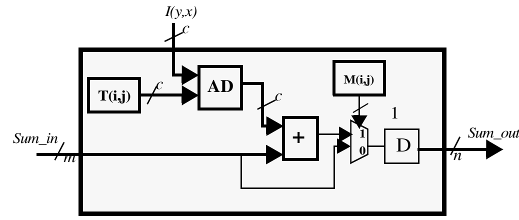

Figure 7 PE structure for partially-dynamic design

The total execution time T_{PD} for the partially-dynamic design to find a template object in N video frames consists of the time $T_{PD}^{computeN}$ to calculate the SA-TM operations and the reconfiguration time T_{PD}^{reconf} to update the FPGA memory for a new template.

The execution time for N frames is calculated as

$$T_{PD}^{computeN} = (N+1)\frac{WH}{f_{PD}} - 1 \quad , \quad (24)$$

with f_{PD} as the maximum clock rate the circuit can run at.

Considering a partially reconfigurable design is used, the reconfiguration time to load the data for a new template into memory can be calculated as

$$T_{PD}^{reconf} = (c \cdot p + W \cdot H) \cdot t_{bit} \quad , \quad (25)$$

where t_{bit} is the time to reconfigure 1 bit. Note that only the p pixel values belonging to the object need to be loaded into memory in addition to mask bits for all *WH* PEs, while any possible pixel values belonging to an old object but not belonging to the new template object do not need to be changed as their use will be disabled by the new mask bits.

6 FPGA Implementation and Results

The PEs for the three reconfigurable designs described above have been implemented for c=8 (eight bits per pixel) targeting Xilinx Virtex XCV1000E devices.

n	A_D[LCs]	f_D[MHz]	A_S[LCs]	f_S[MHz]	A_{PD}[LCs]	f_{PD}[MHz]
8	**14**	**79.9**	**30**	**42.1**	**17**	**82.8**
9	**15**	**74.0**	**32**	**41.3**	**19**	**75.4**
10	**17**	**70.5**	**33**	**40.5**	**20**	**72.0**
12	**19**	**71.3**	**36**	**37.2**	**23**	**66.5**
15	**23**	**66.4**	**41**	**34.9**	**28**	**58.8**
16	**25**	**58.8**	**42**	**34.5**	**29**	**55.0**
29	**42**	**41.6**	**62**	**22.4**	**49**	**37.5**

Table 1 Number of LCs (*A*) and clock frequency (*f*) on Xilinx Virtex 1000E, for PEs with different word length *n*, for dynamic (D), static (S) and partially-dynamic (PD) design

Table 1 shows the results for area in logic cells (LCs) and clock frequency in MHz for PEs of different word length n for the dynamic (D), static (S), and partially-dynamic (PD) designs. An LC contains a look-up table with four inputs (4-LUT) and one flip-flop (FF).

Using the results shown in Table 1, the values a and b for the area calculation of an PE, as in (2), and hence the area estimation for all PEs, can be determined for all three designs:

- dynamic design: $a_D = 1.5$, $b_D = 1$,

  hence $A_D^{PE}(n) = 1.5 \cdot n + 1$,

  $$\text{and } A_D^{PE} \approx \left\lceil 1.5(p \cdot q_p - 2^{q_p} + 1) \right\rceil + 13p \ , \qquad (26)$$

  with $q_p = \lfloor log_2(p) \rfloor + 1$.
- static design: $a_S = 1.5$, $b_S = 18$,

  hence $A_S^{PE}(n) = 1.5 \cdot n + 18$,

  $$\text{and } A_S \approx \left\lceil 1.5(p \cdot q_{WH} - 2^{q_{WH}} + 1) \right\rceil + 30p \ , \qquad (27)$$

  with $q_{WH} = \lfloor log_2(WH) \rfloor + 1$.
- partially-dynamic design: $a_{PD} = 1.5$, $b_{PD} = 5$,

  hence $A_{PD}^{PE}(n) = 1.5 \cdot n + 5$,

  $$\text{and } A_{PD} \approx \left\lceil 1.5(p \cdot q_{WH} - 2^{q_{WH}} + 1) \right\rceil + 17p \ . \qquad (28)$$

6.1 Results for small example

For Example 1, considered in Section 4 ($w = h = 3$, $p = 8$, $W = 7$, $H = 6$), the results are shown in Table 2.

Design	D	S	PD
A measured [LCs]	**223**	**1541**	**996**
A calculated [LCs]	**224**	**1544**	**998**
f [Mhz)	**73.40**	**32.32**	**58.63**
$T^{compute}$ [ns] for 1 clock cycle	**13.6**	**30.9**	**17.1**
# cycles for N frames	**42N**	**42N+41**	**42N+41**
T^{reconf} [ms]	**517.9**	**n/a**	**55.1**

Table 2 Results for Example 1, for dynamic (D), static (S) and partially-dynamic design (PD)

It can be seen from rows 1 and 2 in Table 2 that the calculated areas using the area estimations described in Section 5 are fairly accurate and are used in further examples, where compilation takes too much time or is impossible due to the size of template and/or search frame. The reconfiguration times to load the circuit for the dynamic design and to update the memory of the dynamic design were calculated for a Virtex 1000E device using a reconfiguration clock frequency of 50 MHz and 8 bits per clock cycle, as described in [17].

6.2 Results for HDTV format

For a more practical example, the following parameters are used, as in the HDTV (SMPTE 260M) video processing format: $W = 1920$, $H = 1080$, frame rate 30 Hz. Area and execution time estimations were calculated according to the equations given in Section 5. However, in most cases the designs are too complex to fit into the largest currently available FPGAs, such as Xilinx Virtex XC2V10000.

Table 3 shows the number of logic cells required to implement the dynamic design for different template sizes, separated into PE area and resources to register intermediate results, according to (11), (13), and (15), respectively. The bounding boxes are considered quadratic with 80% of the pixels belonging to the object to be mapped. It can be seen that the number of LCs required grows with the size of the template object. For relatively small templates, the largest share of resources are required for the registers outside the template bounding box necessary to delay intermediate results until valid input signals become available. While for a 10×10 template only 0.7% of the area is used for PEs, the rest for registers, this share grows to 28.7% for a 500×500 template and to 86.0% for a HDTV format size template with 80% opaque pixels.

$w \times h$	10×10	20×20	50×50	100×100	200×200	500×500
A_D^{PE}	1,690	7,714	55,930	247,714	1,086,850	7,606,789
A_D^{Reg1}	268	1,232	8,988	39,952	175,808	1,234,464
A_D^{Reg2}	246,390	590,900	1,714,790	3,732,820	7,776,120	17,697,460
A_D	248,348	599,846	1,779,708	4,020,486	7,951,928	26,538,713

Table 3 Area results [in LCs] for dynamic design using different template sizes and *p*=80% of *wh*

However, for all possible template object sizes and shapes, the area for the dynamic design is at least 34% smaller than the area required for the static design and at least 16% smaller than the area of the partially-dynamic design, which are both constant for all templates used and equivalent to the logic resources of about 1,000 and 800 Xilinx Virtex XC2V10000 devices, respectively.

Figure 8 shows the total area A_D required to implement the dynamic design for different templates with the same number of pixels belonging to the object (p=32,000 = 80% of wh), but of different shapes, that is different proportions of w to h. It can be seen that the smaller the quotient w/h, the larger the number of logic cells required to implement the design. This is due to the fact that templates with a small width, but a large height require far more registers to store intermediate results. In fact, A_D^{PE} and A_D^{REG1} are constant for all cases. Note that this effect would be reversed if instead of a horizontal raster-scan a vertical raster-scan fashion could be used to stream in the video frame pixels.

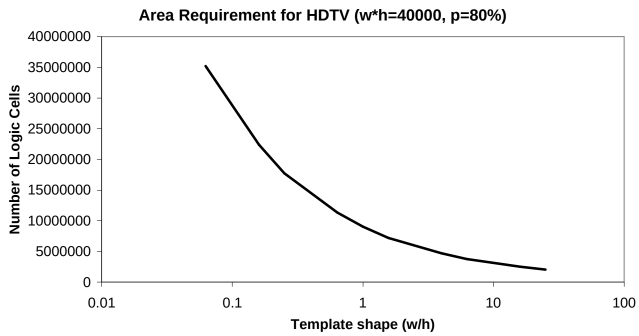

Figure 8 Area A_D [in LCs] for dynamic design using template objects with different shape for p=32,000 = 80% of wh

Figure 9 shows the total execution times T required to perform an SA-TM operation on HDTV format video for a 10×10 size template and for a 100×100 template, depending on the number of consecutive frames N, for all three reconfigurable designs. Included are the computation time and the reconfiguration time, where applicable. There is only one graph shown for the static design (dotted) as the execution time does not depend on the template object used. For the dynamic designs (full lines) and the partially-dynamic designs (dashed lines), the thin graphs show the results for the smaller template whereas the results for the larger template are shown as thicker lines.

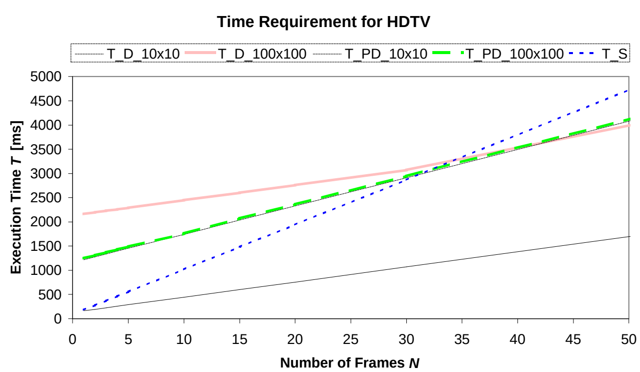

Figure 9 Execution time T in ms using a 10×10 pixel template and a 100×100 pixel template (p=80%), for dynamic (D), static (S) and partially-dynamic design (PD)

While the execution time for the static design is shortest if the algorithm is performed on only one frame as there is no reconfiguration overhead, it increases more steeply due to a lower maximum clock rate, and after about 35 frames the static design will be slower than both dynamic and partially-dynamic designs with either template. Another noticeable fact is that the execution times for the partially-dynamic design for both templates are not very different due to similar reconfiguration overheads (quotient of reconfiguration time over computation time), while the reconfiguration times for the dynamic design depend much more on the size of the template object. For the 10×10 template, the reconfiguration overhead is very small, and hence the dynamic design is the fastest for any number of N>1 and even allows real-time processing, while the time required to reconfigure the device for the 100×100 template is significantly larger, resulting in the dynamic design being the slowest if less than 33 frames are operated on. The reconfiguration overheads (in %), as a function of number of frames N are diagrammed for the dynamic (full lines) and partially-dynamic (dashed lines) designs for both 10×10 template (thin line) and 100×100 template (thick line). Not included in the results are additional overheads due to possible recompiling the dynamic design for a new template.

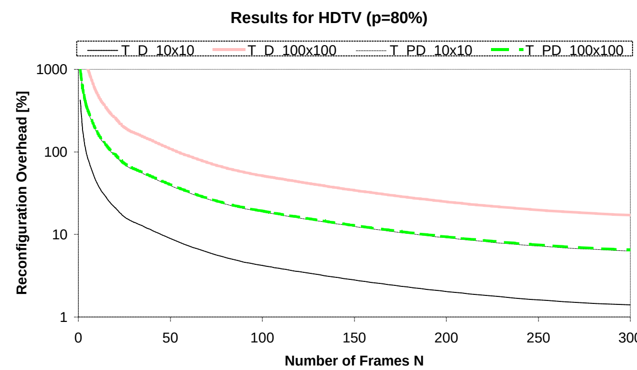

Figure 10 Reconfiguration overhead in %, using a 10×10 template and a 100×100 template (p=80%), for dynamic (D) and partially-dynamic design (PD)

For comparison, the SA-TM algorithm was also performed on a 1.4 GHz Pentium 4 PC with 384 MB RAM, for both templates on a HDTV format image (= 1 frame). We measure the following execution times T_{PC}, and estimate the resulting speed-ups S of the dynamic (D), partially-dynamic (PD), and static design (S), running at their respective maximum clock speeds (compare Table 1) and including reconfiguration time, where applicable, on a Virtex 1000E FPGA, for 300 consecutive frames (= 10 seconds), as shown in Table 4.

w×h	T_{PC}	S_D	S_{PD}	S_S
10×10	22 sec	690	350	230
100×100	1,098 sec	28,600	17,600	11,800

Table 4 Execution time T_{PC} for computing SA-TM on 1 frame on 1.4 GHz Pentium 4 PC, and Speed-ups S of dynamic (D), partially-dynamic (PD) and static (S) designs, for different templates matched on 300 consecutive frames in HDTV format

It can be shown that significant speed-ups can be achieved by using reconfigurable computing to perform SA-TM, especially for large templates, since many operations can be computed in parallel.

7 Conclusion

In this paper, a shape-adaptive template matching (SA-TM) method to retrieve arbitrarily shaped objects within images or video frames was proposed. Different strategies of reconfigurable computing regarding their suitability for implementing the SA-TM method as an example of a typical multimedia algorithm were investigated. A dynamic design, where the configuration data is completely adapted to the shape and size of the template object used, was presented in this paper. A static design which can match templates, stored in off-chip memory, of all possible shapes and sizes within a video frame using the same FPGA configuration was also presented. In addition, a partially-dynamic design which uses on-chip memory, available in most current FPGAs, to store the template was introduced. A generic systolic array architecture provided the basis for the implementation of the three reconfigurable designs. Formulae have been developed to explicitly estimate the logic resources required to implement the designs depending on the size of the search frame and the size and shape of the template object.

While the number of logic cells to implement the static design and the partially-dynamic design is constant for a certain frame size, the dynamic design is adapted to the template object leading to significant savings in area, especially if relatively small templates are used. The suitability of a particular design regarding total execution time of the algorithm strongly depends on the number of consecutive frames the operations are carried out on with the same template. Since the static design does not suffer from reconfiguration overheads, it is most suitable for an operation on one or only a few frames. However, as the partially-dynamic design and especially the dynamic design can operate at higher clock frequencies, they perform better if the matching algorithm is executed on a large number of frames. For the example of matching a 100×100 template on an HDTV format frame, the static design is fastest for an operation on up to 33 frames, at which point the partially-dynamic design becomes faster. After 38 frames the dynamic design, which has the highest reconfiguration time for that template size, will be the fastest design. Real-time SA-TM of HDTV video is possible using the dynamic design for templates with up to 128 pixels. Compared to a software implementation, all three reconfigurable SA-TM designs can achieve significant speed-ups, especially if large templates are used.

References

[1] MPEG Group, "Overview of the MPEG-4 Standard," ISO/IEC JTC1/SC29/WG11 N4030, March 2001.

[2] MPEG Group, "Overview of the MPEG-7 Standard," ISO/IEC JTC1/SC29/WG11 N4031, March 2001.

[3] P. Pirsch, C. Reuter, J.P. Wittenburg, M.B. Kulaczewski, H.-J. Stolberg, "Architecture Concepts for Multimedia Signal Processing," *Journal of VLSI Signal Processing*, vol. 29, no. 3, pp. 157-165, November 2001.

[4] L. Torres, M. Kunt, F. Pereira, "Second Generation Video Coding Schemes and Their Role in MPEG-4," *European Conf. on Multimedia Applications, Services, and Techniques*, May 1996, pp. 799-824.

[5] S. D. Haynes, J. Stone, P. Y. K. Cheung, W. Luk, "Video Image Processing with the SONIC Architecture," *IEEE Computer*, vol. 33, no. 4, pp. 50-57, April 2000.

[6] Y. A. Aslandogan, and C. T. Yu, "Techniques and Systems for Image and Video Retrieval," *IEEE Trans. Knowledge and Data Engineering*, vol. 11, no. 1, pp. 56-63, January/February 1999.

[7] T. Komarek, P. Pirsch, "Array Architectures for Block Matching Algorithms," *IEEE Trans. on Circuits and Systems*, vol. 36, no. 10, October 1989.

[8] J. Villasenor, B. Hutchings, "The Flexibility of Configurable Computing," *IEEE Signal Processing Magazine*, pp. 67-84, Sept. 1998

[9] G. McGregor, and P. Lysaght, "Self Controlling Dynamic Reconfiguration," in *Proc. 9th International Workshop on Field-Programmable Logic and Applications, FPL'99*, 1999, pp. 144-154.

[10] E. Sanchez, M. Sipper, J.-O. Haenni, J.-L. Beuchat, A. Stauffer, and A. Perez-Uribe, "Static and Dynamic Configurable Systems," *IEEE Trans. on Computers*, vol. 48, no. 6, pp. 556-564, June 1999.

[11] P. Lysaght, "Aspects of Dynamically Reconfigurable Logic," in *IEE Colloquium on Reconfigurable Systems*, Glasgow, Scotland, pp. 1-5, March 1999.

[12] J. O. Cadenas, G. M. Megson and T. P. Plaks "Quantitative Evaluation of Three Reconfiguration Strategies on FPGAs: A Case Study," in *HPC-Asia 2000. Proc. of the Fourth Int. Conf. on High-Performance Computing in the Asia-Pacific Region*, May 2000, Beijing, China, pp. 337--342.

[13] N. K. Ratha, A. K. Jain, "Computer Vision Algorithms on Reconfigurable Logic Arrays," *IEEE Trans. on Parallel and Distributed Systems*, vol. 10, no. 1, pp. 29-43, January 1999.

[14] J. Villasenor, B. Schoner, K.-N. Chia, C. Zapata, H. J. Kim, C. Jones, S. Lansing, B. Mangione-Smith, "Configurable Computing Solutions for Automatic Target Recognition," in *Proc. FCCM*, pp. 70-79, 1996.

[15] J. Gause, P. Y. K. Cheung, W. Luk, "Static and Dynamic Reconfigurable Designs of a 2D Shape-Adaptive DCT," in *Proc. FPL*, 2000, pp. 96 - 105.

[16] MPEG Group, "MPEG-4 Video Verification Model Version 13.0," ISO/IEC JTC1/SC29/WG11 N2687, March 1999.

[17] Xilinx Inc., "Virtex Series Configuration Architecture User Guide," *Application Note XAPP*151, September 2000.