

**ΠΟΛΥΤΕΧΝΕΙΟ ΚΡΗΤΗΣ**  
**ΤΜΗΜΑ ΗΛΕΚΤΡΟΝΙΚΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ**  
**ΜΗΧΑΝΙΚΩΝ ΗΛΕΚΤΡΟΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ**



**Μεταπτυχιακή Διατριβή**  
**«Υλοποίηση Αλγορίθμων σε Ενσωματωμένες**  
**Αρχιτεκτονικές με Σταθερούς και Αναδιατασσόμενους**  
**Πόρους»**

**ΧΡΥΣΟΣ ΕΜ. ΓΡΗΓΟΡΙΟΣ**

**ΕΞΕΤΑΣΤΙΚΗ ΕΠΙΤΡΟΠΗ**

Δόλλας Απόστολος, Καθηγητής Π.Κ.(Επιβλέπων)  
Πνευματικάτος Διονύσιος, Αναπληρωτής Καθηγητής Π.Κ.  
Παπαευσταθίου Ιωάννης, Επίκουρος Καθηγητής Π.Κ.

**Χανιά –Δεκέμβριος 2007**

# **SUMMARY**

Audiovisual data remains the preferable way of communication among all types of information and the most demanding one, concerning the communication channel and memory requirements. Despite the progress in network technologies, channel throughput increase, disk technology and memory speed and size increase, the compression of data is still essential because of the enormous amount of data that is stored or transferred, especially for video data. A separate problem for video transmission is that of encryption, which can ensure security of the transmitted information. Several algorithms have been proposed for image and video encryption. In general, the proposed algorithms are implemented mainly in software, resulting in a high quality of encryption but low throughput. The SCAN encryption algorithm belongs to the general category of iterated product cipher algorithms, which can be used to encrypt images and compressed video. SCAN probably is the first method used for combined image encryption and information hiding.

Important domain on image processing is the target recognition. Technological advances in artificial intelligence and especially in heuristic search using fractals, pattern recognition and image understanding have provided the opportunity to develop autonomous systems for Automatic Target Recognition (ATR) from still images.

The contribution of this work is to present a specific, tightly integrated architecture of an entire system for the SCAN compression, encryption and data hiding algorithm and the reverse process implemented using the Stretch technology. In addition to this, this thesis presents the implementation of the smoothing, edge detection and colour segmentation algorithms of ATR using Stretch S5000 processors and compares them with the implementation using software Matlab toolbox. This work is the first in literature, which reports a complete, tightly integrated, low-cost embeddable system for information compression, encryption and hiding and for images' Automatic Target Recognition algorithm. Finally, this thesis compares implementations with FPGAs for the SCAN compression/encryption/data hiding and software implementations with Matlab Toolbox for ATR algorithm towards software-reconfigurable designs.

## CONTENTS

<b>Chapter 1 .....</b>	<b>6</b>
<b>Introduction .....</b>	<b>6</b>
<b>Chapter 2 .....</b>	<b>10</b>
<b>Relevant Research.....</b>	<b>10</b>
<b>2.1 Previous implementations on compression, encryption and data hiding algorithms.....</b>	<b>10</b>
<b>2.1.1 SCAN methodology .....</b>	<b>10</b>
<b>2.1.2 The DES Algorithm .....</b>	<b>11</b>
<b>2.1.3 Rijndael Advanced Encryption Standard .....</b>	<b>11</b>
<b>2.1.4 Partial Encryption of Compressed Images and Videos....</b>	<b>13</b>
<b>2.2 Previous implementations on image segmentation algorithms</b>	<b>14</b>
<b>2.3 Stretch Technology .....</b>	<b>17</b>
<b>Chapter 3 .....</b>	<b>20</b>
<b>SCAN-Based Compression\Encryption\Data Hiding algorithms .....</b>	<b>20</b>
<b>3.1 SCAN methodology .....</b>	<b>20</b>
<b>3.2 SCAN Compression Algorithm.....</b>	<b>21</b>
<b>3.3 SCAN Encryption Algorithm .....</b>	<b>22</b>
<b>3.4 SCAN Encryption Algorithm .....</b>	<b>26</b>
<b>Chapter 4 .....</b>	<b>29</b>
<b>Smoothing, Edge Detection and Colour Segmentation Algorithms in Colour Images.....</b>	<b>29</b>
<b>4.1 Smoothing Algorithm.....</b>	<b>31</b>
<b>4.2 Edge Detection Algorithm .....</b>	<b>35</b>
<b>4.3 Colour Segmentation Algorithm .....</b>	<b>37</b>
<b>4.3.1 Find Big and Crisp Segments.....</b>	<b>39</b>
<b>4.3.2 Expand Segments based on the Homogeneity Criteria ...</b>	<b>40</b>
<b>4.3.3 Expand segments based on Dichromatic Reflection Model</b>	<b>42</b>
<b>4.3.4 Expand segments based on Degree of Farness.....</b>	<b>43</b>
<b>4.3.5 Iterative filtering .....</b>	<b>44</b>
<b>4.3.6 Finish segmentation process .....</b>	<b>44</b>
<b>Chapter 5 .....</b>	<b>46</b>
<b>Architecture of Compression/Encryption/Data Hiding and Decompression/Decryption/Data Unhiding Subsystems of SCAN Algorithm.....</b>	<b>46</b>
<b>5.1 Architecture of Compression/Encryption/Data hiding system</b>	<b>47</b>
<b>5.1.1 Architecture of Compression Subsystem.....</b>	<b>47</b>
<b>5.1.2 Architecture of Encryption Subsystem.....</b>	<b>48</b>
<b>5.1.3 Architecture of Data Hiding Subsystem .....</b>	<b>50</b>
<b>5.2 Architecture of Decompression/Decryption/Data Unhiding System .....</b>	<b>52</b>
<b>5.2.1 Architecture of Data Unhiding Subsystem .....</b>	<b>52</b>
<b>5.2.2 Architecture of Data Decryption Subsystem.....</b>	<b>53</b>
<b>5.2.3 Architecture of Decompression Subsystem .....</b>	<b>55</b>

## Microprocessor Hardware Laboratory

<b>Chapter 6</b> .....	<b>57</b>
<b>Architecture of Smoothing, Edge Detection and Color Segmentation</b>	
<b>Algorithm</b> .....	<b>57</b>
<b>6.1 Image Smoothing Subsystem</b> .....	<b>58</b>
<b>6.2 Edge Detection Subsystem</b> .....	<b>61</b>
<b>6.3 Color Segmentation Subsystem</b> .....	<b>63</b>
<b>Chapter 7</b> .....	<b>68</b>
<b>Performance and Comparisons</b> .....	<b>68</b>
<b>7.1 Performance of the Compression/Encryption/Data hiding</b>	
<b>system and comparison with reconfigurable designs</b> .....	<b>68</b>
<b>7.2 Performance of the Smoothing, Edge detection and Colour</b>	
<b>segmentation system and comparison with software implementation</b>	
75	
<b>Chapter 8</b> .....	<b>80</b>
<b>Conclusions and future work</b> .....	<b>80</b>
<b>8.1 Conclusions</b> .....	<b>80</b>
<b>8.2 Future work</b> .....	<b>82</b>
<b>References</b> .....	<b>83</b>
<b>Internet</b> .....	<b>83</b>
<b>Bibliography</b> .....	<b>83</b>

**Ο κόσμος του κάθε ανθρώπου  
στηρίζεται σε δύο κολώνες :  
ότι θυμόμαστε και ότι αγαπάμε...**

**Αφιερωμένη στην οικογένεια μου**

## ΕΥΧΑΡΙΣΤΙΕΣ

Πρώτα απ'όλους θα ήθελα να ευχαριστήσω τον καθηγητή Απ. Δόλλα, για την υποστήριξη και την πολύτιμη βοήθεια του κατά τη διάρκεια υλοποίησης αυτής της μεταπτυχιακής εργασίας.

Στην συνέχεια, θέλω να ευχαριστήσω :

Την επιτροπή της διπλωματικής μου εργασίας, τους καθηγητές Δ. Πνευματικό και Ι. Παπαευσταθίου για την συμβολή τους στην εργασία αυτή.

Τον κ. Κιμιωνή Μαρκο, μέλος ΕΕΔΙΠ και υπεύθυνος του εργαστηρίου Μικροεπεξεργαστών και Υλικού, για την υποστήριξη του και την αμέριστη συμπαράσταση του.

Τους κ. Ευρυπίδη Σωτηριάδη, κ. Κυπριανό Παπαδημητρίου και Δημήτρη Μειντάνη, διδακτορικοί φοιτητές, για τις συμβουλές και τις ιδέες τους σε δύσκολα σημεία της εργασίας.

Όλους τους προπτυχιακούς και μεταπτυχιακούς φοιτητές του εργαστηρίου Μικροεπεξεργαστών και Υλικού για την βοήθεια και την στήριξη τους.

Τους φίλους μου για όλες τις καλές στιγμές που περάσαμε μαζί και ελπίζω στο μέλλον να υπάρξουν ακόμα καλύτερες.

Τις αδελφές μου, την μητέρα μου και τον πατέρα μου που παρόλο τις δύσκολες καταστάσεις που αντιμετωπίσαμε παραμένουμε αγαπημένοι.

Τέλος, ένα μεγάλο ευχαριστώ στην Φωτεινή, η οποία βρίσκεται στο πλαι μου σε όλες τις καλές και κακές στιγμές της ζωής μου.

## **Chapter 1**

### **Introduction**

Audiovisual data remains the preferable way of communication among all types of information and the most demanding one, concerning the communication channel and memory requirements. Despite the progress in network technologies, channel throughput increase, disk technology and memory speed and size increases, the compression of data is still essential because of the enormous amount of data that is stored or transferred, especially for video data. Although, there are enough available products for video compression that are based on hardware and software implementations, the design of efficient coding algorithms for video compression remains a pertinent research problem. This happens because of its computing requirements and need for higher quality of picture at lower data rates. The degree of compression depends on the redundant information of data. In each frame it is very likely that the values of neighbouring pixels are close and in this case we speak about spatial redundancy. Also, most of the information in one frame can also exist in the preceding frames and this is temporal redundancy. Techniques which exploit only the spatial redundancies are categorized as intraframe techniques of coding. Examples of intraframe coding include differential pulse code modulation [1], transform coding using discrete cosine transform [2], [3], subband coding [4], pyramid coding and vector quantization [5]. It is important to mention that for better compression both intraframe and interframe techniques can be used. The SCAN algorithm for video compression is based on the differences of adjacent frames [6].

A separate problem for video transmission is that of encryption, which can ensure security of the transmitted information. Several algorithms have been proposed for image and video encryption. In general, the proposed algorithms are implemented mainly in software, resulting in a high quality of encryption but low throughput. On the other hand, very few algorithms have been proposed in order to encrypt data in hardware, and these algorithms are in general stream oriented with respect to their input source. The SCAN

encryption algorithm [6], [7] belongs to the general category of iterated product cipher algorithms, which can be used to encrypt images and compressed video. SCAN probably is the first method used for combined image encryption and information hiding [8]. This algorithm is based on permutations of the image pixels and replacement of the pixel values. The encryption power of the SCAN method is based on the very large number of private keys. In particular, for an image of 512x512 pixels the number of available private keys is  $10^{76000}$ . This means that the most powerful parallel computer today requires  $10^{75000}$  years to decrypt that image using a brute force decryption approach. The penalty of the SCAN methodology is the non-real-time performance in software due to its complexity of compressing-encrypting-hiding of an image.

The combination of video encryption and compression consists of finding the differences of each frame from the first frame of the video, the compression of this and finally the encryption of the compressed frames of differences[9]. In addition to compression and encryption it may be desirable to embed within some type of information (e.g. video) additional information which is meant to be decoded only through the use of some access key. E.g. it may be desirable to hide a patient's medical record inside the video of some medical test, but the record (as it is personal, sensitive information) should be accessible only by the appropriate doctors. The SCAN information hiding method [10] can be extended to hide information in videos by applying the image information hiding method to each video frame. Since hiding data into similar adjacent frames might make detection possible, this approach is suitable only in applications where security is not an issue. If security is important then secret data can be embedded into a few frames in each scene of the video. For a complete system the decompression/decryption/data unhiding system is essential.

The compression algorithms are divided into two categories: lossy and lossless algorithms. A lossy algorithm achieves greater compression ratio, but during the process of decompression there is a loss of information and as a result a reduction of the quality of the video frames. Generally, the decompression/decryption and data unhiding procedure is the opposite

process of the compression/encryption/data hiding algorithm. In order to have as good video quality in the reverse process, as in the initial video sequence the system needs some additional information such as the scanning keys that were used for the compression algorithm or the size of compressed frame for each video frame.

Another important domain on image processing is the target recognition. Technological advances in artificial intelligence and especially in heuristic search using fractals, pattern recognition and image understanding have provided the opportunity to develop autonomous systems for Automatic Target Recognition (ATR) from still images [11-16]. ATR in pattern recognition and image understanding based method depends on resolution for a successful classification. Fuzzy and semi-fuzzy clustering algorithms have been presented for extracting and recognizing the target's features.

In particular, the ATR algorithm consists of a combination of algorithms, such as heuristic segmentation, edge detection, thinning, region growing, fractals, etc., appropriately selected for recognizing targets under various conditions, such as moving target - still camera, still camera - moving target, moving target - moving camera.

The contribution of this work is in presenting a specific, tightly integrated architecture of an entire system for the SCAN compression, encryption and data hiding algorithm and the reverse process implemented using the Stretch technology. In addition to this, this thesis presents the implementations of the smoothing, edge detection and colour segmentation algorithms using S5000 processors and compares them with the implementations using software Matlab toolbox. This work is the first that we know of in literature which reports a complete, tightly integrated, low-cost embeddable system for information compression, encryption and hiding and for images' Automatic Target Recognition algorithm. Second, this work attempts to compare the previous implementations with FPGAs for the SCAN compression and encryption with the implementations with the new Stretch technology and software implementations with Matlab Toolbox towards software-reconfigurable designs. Third, the innovation of this work is the way of fitting complex algorithms in Stretch technology with the usage of low level design tools. Lastly, during this work one of the most difficult challenges was

the sizing problem that appeared during the hardware/software co design. This appeared because of the small size of the reconfigurable part of the S5000 processor and the restricted communication channels between the software design and the hardware platform. Preliminary results from this work were published in [18].

The rest of this thesis is organized as follows: Chapter 2 briefly describes previous implementations on compression/encryption and colour segmentation and ATR algorithm. Chapter 3 describes the algorithm of SCAN compression/encryption/data hiding and the reverse procedure. Chapter 4 outlines the smoothing, edge detection and colour segmentation algorithms of ATR. Chapters 5 and 6 have the new architectures, its major subsystems, their interconnection, and its mapping on the Stretch technology. Chapter 7 has performance results and a detailed comparison to the previous implementations. Finally, Chapter 8 has some conclusions from this work.

## Chapter 2

### Relevant Research

This section presents previous implementations of compression/encryption/data hiding algorithms and implementations of image processing algorithms. There are several algorithms that have been proposed for these two applications with various characteristics and they are described below. Finally, the Stretch technology is a new idea of processors with an embedded reconfigurable part and it is described in this part of the thesis. The architecture and the characteristics of these new processors are described in the next sections.

#### 2.1 Previous implementations on compression, encryption and data hiding algorithms

There are many algorithms about the data compression and encryption in the literature. Each of the algorithms mentioned below follow a different approach on the compression and encryption issue. Some of these algorithms have been implemented in hardware, like SCAN encryption [19, 20].

##### 2.1.1 SCAN methodology

The SCAN methodology for information hiding-compression-encryption has been studied in [6, 7, 8, 9] and specific reconfigurable architectures have been developed for the encryption [19, 20, 22] and for the compression [23] aspects of it. Prior to this work there was no implementation of the hiding aspects of SCAN, and there has been no fully integrated system in reconfigurable logic.

The FPGA implementation of the SCAN method provides real-time capabilities not only for image encryption but for video encryption as well. The FPGA implementation of the SCAN encryption is a validated design. The main

reason of using an FPGA implementation is the flexible way of creating a real-time programmable solution. In the specific architecture the image was split into blocks of 64x64 pixels each, because this gives sufficiently good encryption and simultaneously offers parallelism using RAM-based methods that are supported in hardware. The architecture for the video encryption, shown in [19, 22], consists of two independent RAMs of 4 Kbytes each, an Address Generator, an Address Counter, the Substitution Unit and the Control Unit.

In addition to the above implementation, an FPGA implementation of the SCAN Compression Scheme was fully developed and validated. The architecture, which is described in [23], consists of the SDRAM subsystem (which consists of the SDRAM memory and the corresponding Address Generator), a unit that splits each frame to 16x16 windows, four units for frame comparison, four Address Generators, four SRAMs of 32K x 16 bits each, four Dual Port RAMs of 4K x 9 bits each, four RAMS of 2500 x 32 bits each and nine FIFOs of 16 x 8 bits each.

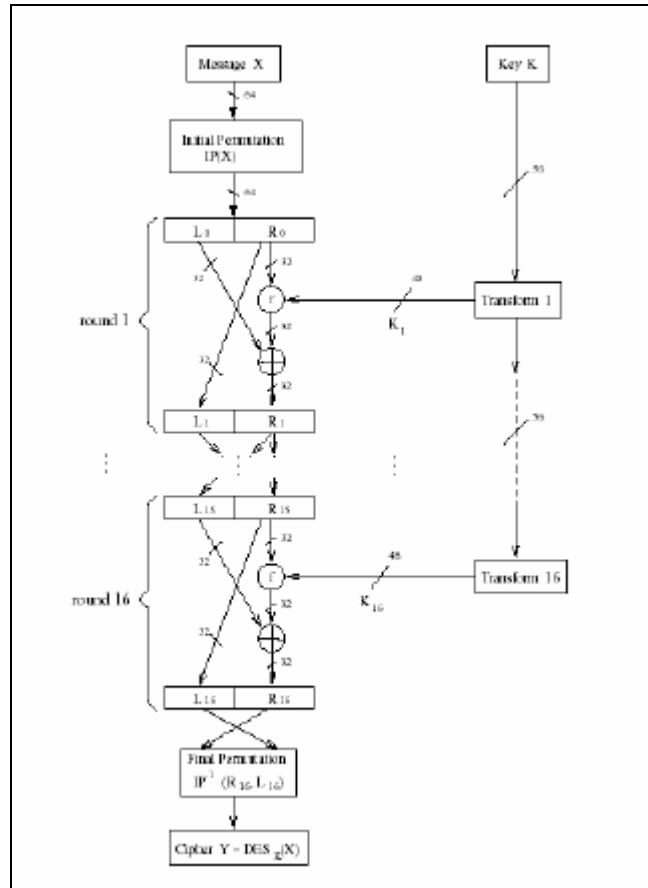
### 2.1.2 The DES Algorithm

One of the wide known block cipher using private key is the DES algorithm. It was proposed by IBM during the early 1970s and it was adopted as a federal standard on November 23, 1976. There are many DES variants, but the most possible successor is the triple DES which is much harder to break using exhaustive search:  $2^{112}$  attempts instead of  $2^{56}$  attempts. The fastest implementation of the DES algorithm in hardware is reported in [29] providing a throughput of 400Mbytes/sec. In [24] there is an application of the DES algorithm in order to encrypt images, unfortunately without providing any quality results. The block diagram of the DES algorithm is shown in **Figure 2.1**.

### 2.1.3 Rijndael Advanced Encryption Standard

Rijndael is a private-key symmetric block encryption algorithm that supports 128, 92, and 256-bit length keys and operates on 128, 192 and 256-

bit blocks. All nine combinations of key length and block size are possible. Recently, Rijndael was selected as the Advanced Encryption Standard (AES) to replace DES. Karri has presented an FPGA implementation of this algorithm. The Rijndael encryption algorithm is shown in **Figure 2.2**.

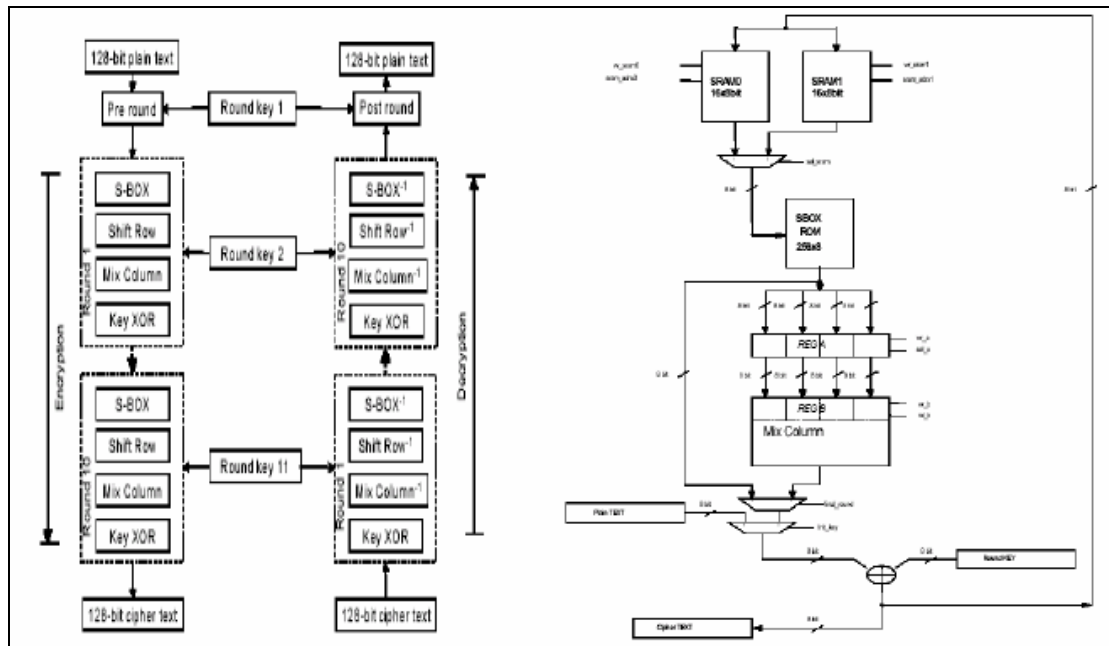


**Figure 2.1: The block diagram of the DES algorithm**

The round transformation data path shown, also, in **Figure 2.2** implements the byte substitution, shift row, mix column and key xor operations. The data path consists of two 16x8 SRAMs (SRAM 0 and SRAM 1), one 256x8 ROM (SBOX), two 32-bit registers (REG\_A and REG\_B) and three multiplexers. The total design targets the Wildforce reconfigurable computing board and its performance is 124 Mbits/s using 13.6 MHz clock frequency.

## 2.1.4 Partial Encryption of Compressed Images and Videos

Another approach to encrypt images is to combine the compression and the encryption in order to eliminate the demanding distinct processing. Cheng [32, 33] propose a novel approach called partial encryption in order to reduce encryption and decryption time in image and video communication and processing. In this approach, only part of the compressed data is encrypted, as shown in **Figure 2.3**. The proposed algorithm can be applied in schemes of quad tree and wavelet image compression, as well as an extension for video compression. Partial encryption allows



**Figure 2.2: The block diagram and the architecture of the Advanced Encryption Standard algorithm**

the encryption and decryption time to be significantly reduced without affecting the compression performance of the underlying compression algorithm. It is also shown that although a large portion of the compressed data is left unencrypted, it is difficult to recover the original data without decrypting the encrypted part. In the case of quad tree image compression the encrypted portion is 13%- 27% of the compressed output for typical images. For wavelet compression based on zero trees, less than 2% of the

compressed output is encrypted for 512x512 images. The results on video compression are similar.

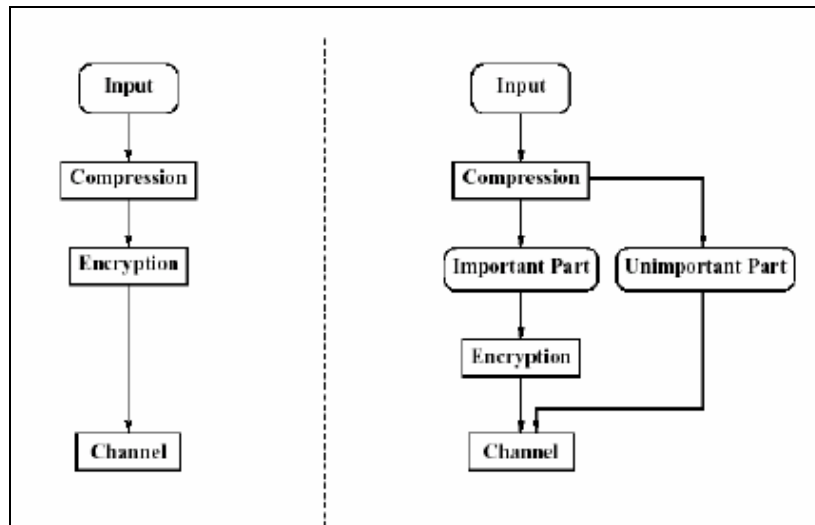


Figure 2.3: The block diagram of partial encryption

## 2.2 Previous implementations on image segmentation algorithms

The segmentation techniques are categorised into three classes [34]:

- i) characteristic feature thresholding or clustering,
- ii) edge detection
- iii) region extracting and merging.

In grey level images, only a few characteristic features are presented. The grey level value specified for each pixel in the image is the most significant. In colour images this is called clustering [35–37]. Thresholding the characteristic features or clusters is a widely used approach for segmentation [38]. Lee and Chung [38] showed that thresholding would usually produce good results in only bimodal images, where the images consisted of only one object and the background. Here, the threshold can be picked at a valley location within the image's greyscale histogram. However, when the object area is small compared to the background area, or when both the object and background assume some broad range of grey levels, selecting a good threshold is difficult. Another weakness of this technique occurs when multiple objects are present within the image. In such cases, finding sharp valleys

within the histogram is further complicated, and segmentation results may be very poor.

Edge detection is another approach to image segmentation [39]. An edge is defined as a location where a sharp change in grey level or colour is detected. However, in this method it is difficult to maintain the continuity of detected edges; a segment must always be enclosed by a continuous edge.

Region growing or merging is a third approach for image segmentation [35]. In this case, easily found, large continuous regions or segments are detected first. Afterwards, small regions may be merged by using homogeneity criteria [41, 42]. One disadvantage of region growing and merging is the inherently sequential nature of this approach. Often, the regions produced depend upon the order in which those regions grow or merge.

Klinker [37, 38] developed a creative dichromatic reflection model, which described the colour of reflected light as a linear combination of the colour of surface reflection (highlights) and body reflection (object colour). Applying this model to region growing and merging method produced impressive results. In this method, highlight areas were merged with the matte areas of an object. Conversely, using it where contrast between neighbouring objects is weak, it merged objects. However, using hard thresholds throughout degraded the performance of this technique within its intermediate stages.

Some of these image segmentation processes were fused with edge location method to produce better results [41,42,45,46]. Segmentation based on the theory of approximate reasoning or fuzzy-like reasoning produced promising results. More specifically, Huntsberger [41,42,47–51] defined colour edges as the zero crossing of differences between the membership values of each pixel. The fuzzy membership values are generated by using an iterative c-mean segmentation algorithm, but it is time consuming due to its iterative nature. Lim [64z] presents an n automated coarse-to-fine segmentation method. This approach is based on histogram thresholds for each colour and the c-means algorithm. An interesting approach, proposed by Lambert and Carron [65], combined the colour space (where hue was explicitly defined and processed according to its relevancy to chroma) and symbolic representations

and rule-based systems (using colour and luminance features to determine homogeneity among pixels).

Weeks and Hauge [66] proposed a new method for colour image segmentation. Instead of segmenting the colour image in RGB colour space, which did not closely model the psychological understanding of colour, they chose HSI (hue, saturation and intensity) space.

There are architectures of implemented segmentation algorithms in hardware. Perez and Koch proposed the use of a simplified hue description suitable for implementation in analogue VLSI. They designed and fabricated for the first time an analog CMOS VLSI circuit with on-board phototransistor input that computes normalized color and hue.

Another architectural approach on colour segmentation and pattern matching is a two level CMOS architecture on neuromorphic colour processing. They designed a 128(H) X 64(V) X RGB CMOS imager, which is integrated with analog and digital signal processing circuitry to realize focal plane region-of-interest selection, RGB-to-HSI transformation, HSI-based segmentation, 36-bin HSI histogramming, and sum-of-absolute-difference (SAD) template matching for object recognition. The organisation of chip is presented in **Figure 2.4**.

This prototype demonstrated that a real-time color segmentation and recognition system can be implemented in VLSI using a small silicon area and small power budget. They also demonstrated that the HSI representation used in this chip is robust under multiplicative and additive shift in the original RGB components.

Concluding, it is noteworthy that there are no implementations of using processors or reconfigurable technology. The hardware architectures that are presented in literature are a small number and use VLSI as they give very good throughput and real-time results.

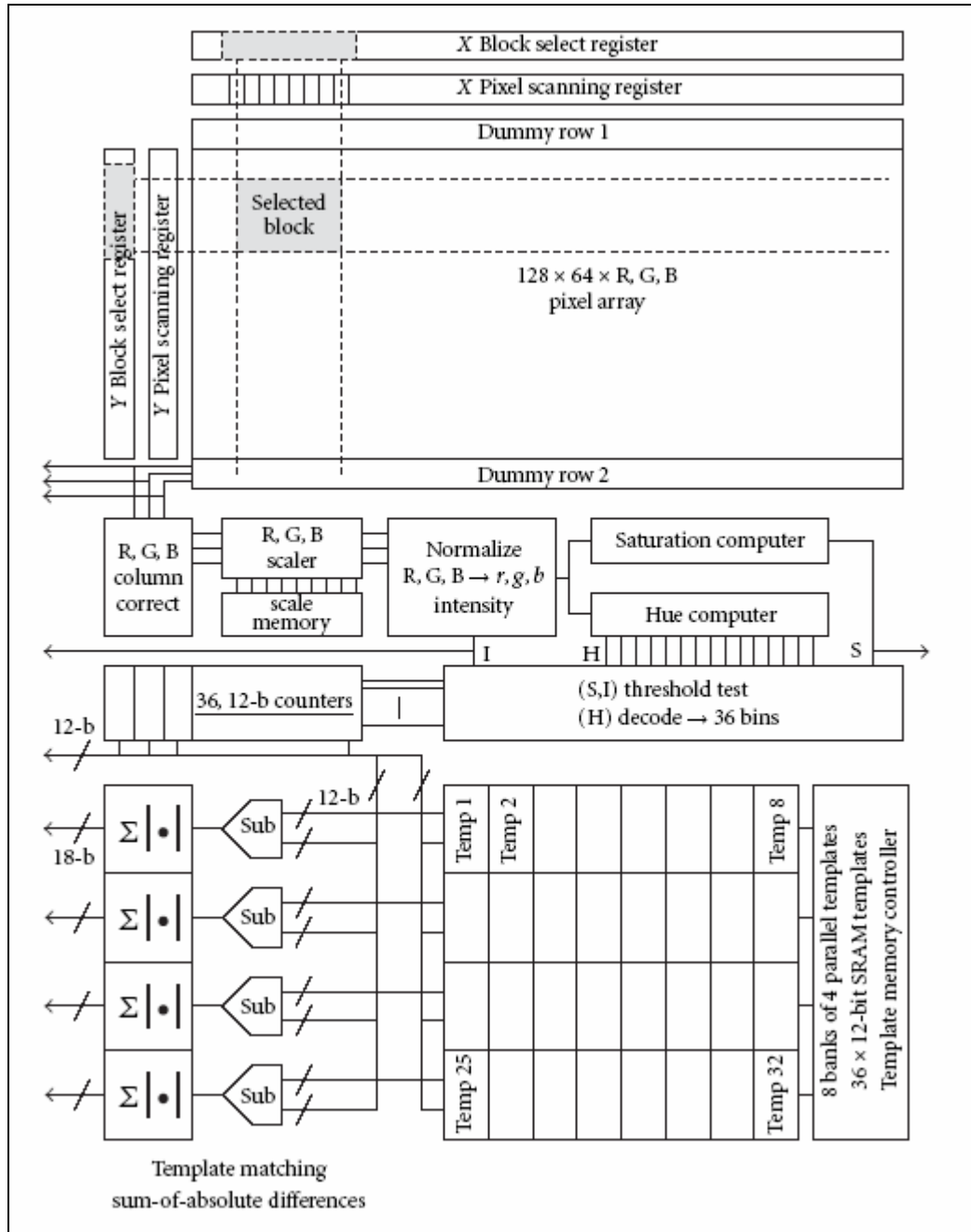


Figure 2.4 : Computational and physical architecture of the chip

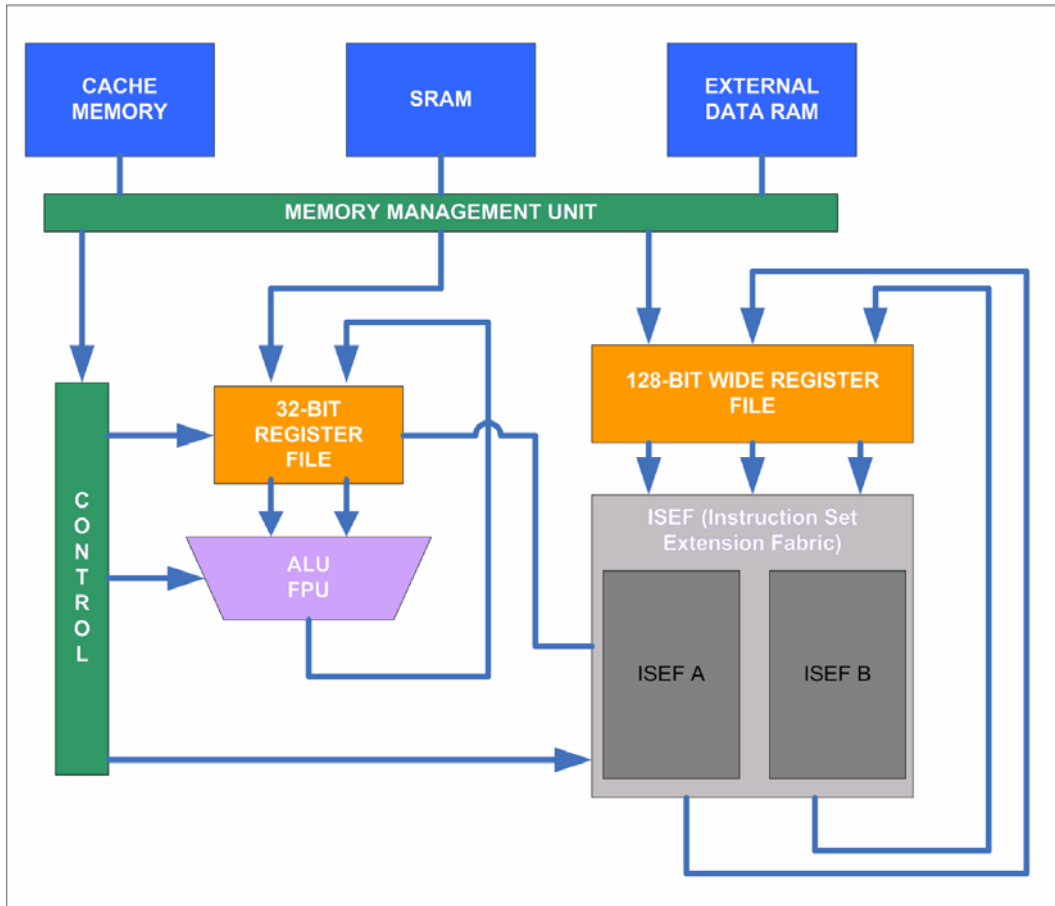
## 2.3 Stretch Technology

The Stretch technology is a new technology to design software configurable processors. The Stretch Company has constructed the series of S5000 software configurable processors, which is based on the Tensilica core RISC processor with a small embedded reconfigurable part, as shown in **Figure 2.5**. The design flow comprises of system development in C/C++,

## Microprocessor Hardware Laboratory

profiling of the code, and mapping its critical sections to the reconfigurable fabric as special, hardware-implemented instructions.

S5000 processors incorporate the Tensilica Xtensa RISC processor core and the Stretch Instruction Set Extension Fabric (ISEF). ISEF is an embedded programmable logic unit where the compute intensive parts of the implementation can



**Figure 2.5 : Architecture of S5000 processors**

be mapped. The S5 Engine family provides two independent Instruction Set Extension Fabric (ISEF) units, ISEF A and ISEF B, which can be configured and used independently. The C/C++ language is used to program the S5000 processors. Stretch C is a C-like language which includes some extensions for hardware implementation. Stretch C is the programming language which is used for mapping the critical parts of the design in reconfigurable parts of the processor.

The reconfigurable part of processor contains a built-in sum total of computation resources. There are two disjoint sets of computation resources:

## **Microprocessor Hardware Laboratory**

one for arithmetic and logic computations, and one for multiply and shift computations. The basic unit for the arithmetic and logic computation resource is one Arithmetic Unit. (AU). The basic unit for the multiply and variable shift computation resource is one Multiply Unit (MU). The number of AU and MU for each reconfigurable part is 4096 and 8192 respectively. The sum of the computation resources used by all the instructions associated with each ISEF configuration must not exceed the total available computation resources in an ISEF unit.

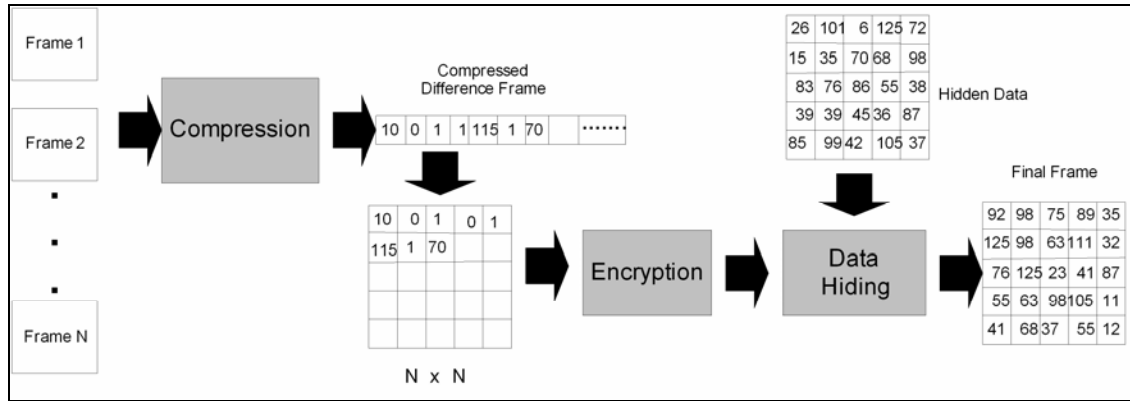
Concluding, the S5000 family of processors is based on Stretch's revolutionary S5 engine, and provides the following key benefits:

- Boosts system performance in compute-intensive applications
- Enables fast time-to-market performance
- Reduces development and system costs
- Provides high-performance I/O's at industry-leading speeds

## Chapter 3

### SCAN-Based Compression\Encryption\Data Hiding algorithms

The SCAN algorithm [6, 7, 8] is a class of formal languages, which can be applied to compression, encryption, data hiding, or combinations thereof. This section describes the SCAN language in detail and provides a presentation of the compression, encryption and data hiding algorithm. The data flow of compression/encryption /data hiding algorithm is shown in **Figure 3.1**



**Figure 3.1: Block diagram for the compression/encryption/data hiding system**

### 3.1 SCAN methodology

A scanning of a two dimensional array  $P_{m \times n} = \{p(i, j) : 1 \leq i \leq m, 1 \leq j \leq n\}$  is a bijective function from  $P_{m \times n}$  to the set  $\{1, 2, \dots, mn-1, mn\}$ . In other words, a scanning of a two dimensional array is an order in which each element of the array is accessed exactly once, or a permutation of the array elements. The terms scanning, scanning path, Scan pattern, and Scan word are used interchangeably in this paper.

The SCAN represents a family of formal languages based on two-dimensional spatial accessing methodologies, which can represent and generate a large number of scanning paths easily. The SCAN family of formal languages includes several versions such as Simple SCAN, Extended SCAN,

and Generalized SCAN, each of which can represent and generate a specific set of scanning paths. Each SCAN language is defined by a grammar and each language has a set of basic scan patterns, a set of transformations, and a set of rules to compose simple scan patterns, which in turn are used to obtain complex scan patterns. The rules for building complex scan patterns from simple scan patterns are specified by the production rules of the grammar of each specific language.

### 3.2 SCAN Compression Algorithm

The SCAN compression algorithm consists of two main steps. These steps are (1) compression of frames and (2) encoding of the information. The first step of the SCAN compression algorithm is the calculation of compare frames and difference frames. The compare frame is a two dimensional matrix with the same size as the frames of the video. The first compare frame is computed with the comparison of the pixels between the two first frames of video. The corresponding pixels of the two first frames are compared and if their values differ more than the threshold that the user has defined, then in the position of the comparison matrix the value of the first frame is placed, otherwise the value of the second frame is used. This process continues using each time the compare frame as the first frame of the comparison with the other frames. At the same time the pixels of the difference frame are calculated. The difference frame is, also, a two dimensional matrix, of the same size as compare frame, whose values are either -1 or the value of the corresponding pixel of the second frame depending on the difference of the pixels between the comparing frames. The pseudocode for the SCAN compression algorithm is shown in **Figure 3.2**.

Subsequently to the computation of compare frame and difference frame, the difference frame is broken to 4x4 windows (16 pixels) and it was specified which of these windows should be put in the compressed difference frame (the windows which contain only the value -1 in each pixel are omitted from the procedure of encoding in the compressed difference frame). Finally,

the result of the compression process is an array with only the encoded information of the compressed frames.

The *decompression method* which takes place in the receiver is exactly the opposite process vs. that of the transmitter. Apart from the data of compressed frames the receiver must have the values of the pixels of the first frame, the dimension of the

```

Compression (F1, F2, w, h, D, C, TH)
Inputs:  First frame : F1
         Second frame : F2
         w, h : dimensions of the frames
         TH : Threshold
Output:  Difference matrix D
         Compare matrix C
{
    For(i=1 to h-1,i++)
    {
        For(j=1 to w-1,j++)
        {
            If( |F1[i][j] - F2[i][j]| <= TH)
            {
                D[i][j] = -1
                C[i][j] = F1[i][j]
            }
            Else
            {
                D[i][j] = F2[i][j]
                C[i][j] = F2[i][j]
            }
        }
    }
}
    
```

**Figure 3.2: Pseudocode for the SCAN Compression Algorithm**

frames and the value of the threshold, which is sent separately and all this information is necessary for the decompression system.

### 3.3 SCAN Encryption Algorithm

The basic idea of the SCAN encryption method is to rearrange the pixels of the image and change the pixel values. The rearrangement is done by a set of scanning patterns (encryption keys) generated by an encryption-specific SCAN language, which is formally defined by the grammar  $G = (\Gamma, \Sigma, A, \Pi)$ . Grammar  $G$  comprises of non-terminal symbols  $\Gamma = \{A, S, P, U, V, T\}$ , of

## Microprocessor Hardware Laboratory

terminal symbols  $\Sigma = \{c, d, o, s, r, a, e, m, y, w, b, z, x, B, Z, X, (, ), \text{space}, 0, 1, 2, 3, 4, 5, 6, 7\}$ , its start symbol is  $A$ , and its production rules  $\Pi$  are given in **Figure 3.3**.

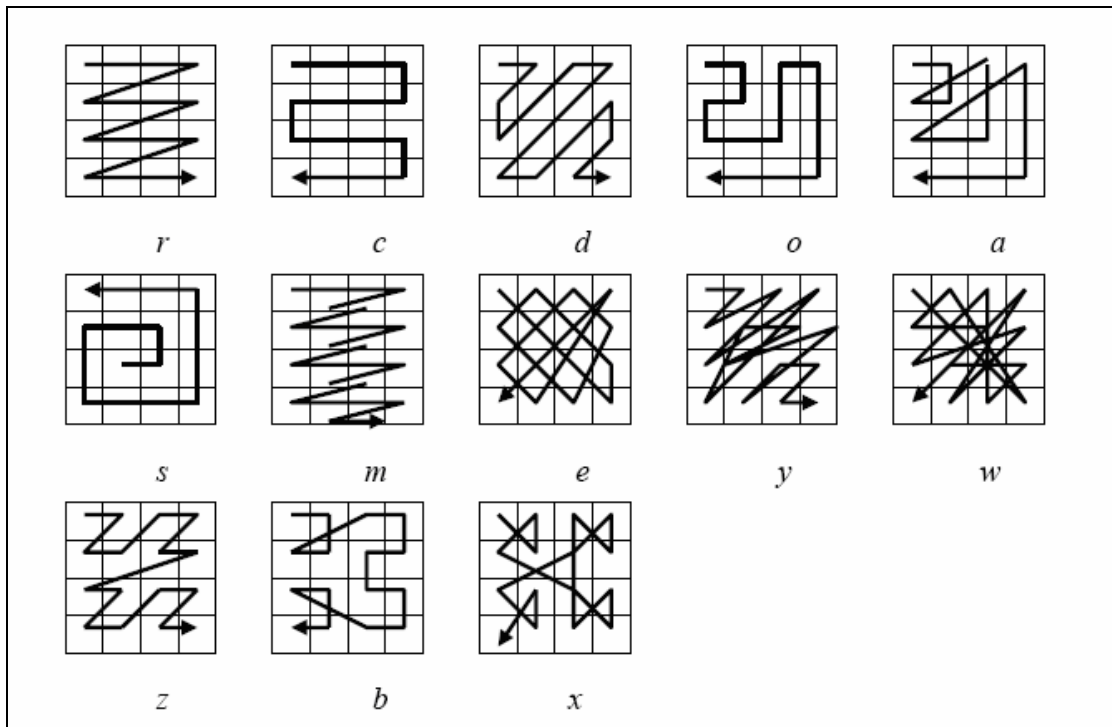
```

A → S | P
S → UT
P → VT(A A A A)
U → c | d | o | s | r | a | e | m | y | w | b | z | x
V → B | Z | X
T → 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7
r c d o a
s m e y w
z b x

```

**Figure 3.3: Grammar of SCAN language**

where the scan patterns (from which the method gets its name) for  $r, c, d, o, a, s, m, e, y, w, z, b, x$  are shown in **Figure 3.4**.



**Figure 3.4: Scan patterns for the SCAN language**

The semantics of this encryption-specific SCAN language are described as follows:

- (a)  $A \rightarrow S | P$  means process the region by scan  $S$  or partition  $P$ .
- (b)  $S \rightarrow UT$  means scan the region with scan pattern  $U$  and transformation  $T$ .

(c)  $P \rightarrow VT(A A A A)$  means partition the region with partition  $V$  and transformation  $T$ , and process each of the four subregions in partition order using  $A$ s from left to right.

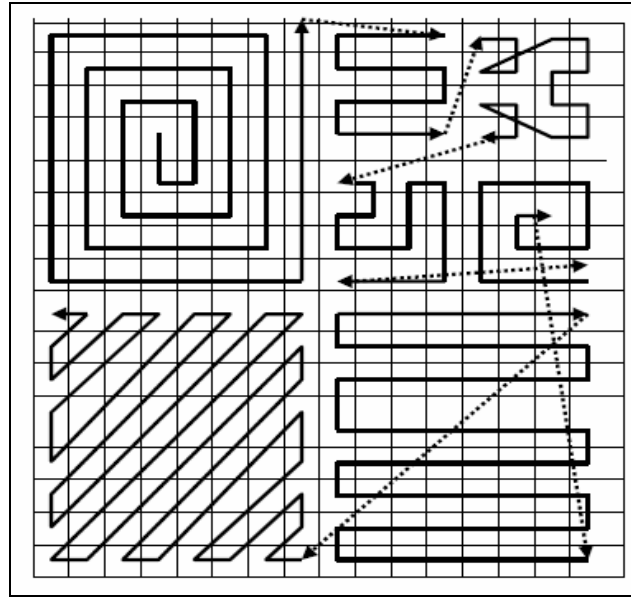
(d)  $U \rightarrow c | d | o | s | r | a | e | m | y | w | b | z | x$  means scan with continuous raster, or diagonal, or continuous orthogonal, or spiral out, or raster, or right orthogonal, or diagonal parallel, or horizontal symmetry, or diagonal symmetry, or diagonal secondary, or block, or zeta, or xi respectively. These scan patterns are shown in **Figure 3.4**.

(e)  $V \rightarrow B | Z | X$  means partition with letter  $B$  or letter  $Z$  or letter  $X$  respectively.

(f)  $T \rightarrow 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7$  means use one of the eight transformation with a scan or partition. For a partition, these transformations are shown in Fig.3-2. For all scan patterns, 0 means the identity transformation as shown in Fig.1, and 2 means 90° clockwise rotation. For scan patterns  $c, o, s, a, e, m, y, w, b,$  and  $x$ , 4 means 180° clockwise rotation and 6 means 270° clockwise rotation. For scan patterns  $r$  and  $z$ , 4 means vertical reflection and 6 means vertical reflection followed by 90° clockwise rotation. For scan pattern  $d$ , 4 means 90° clockwise rotation followed by horizontal reflection and 6 means 180° clockwise rotation followed by vertical reflection. For all scan patterns, 1, 3, 5, and 7 are reverses of scanning paths specified by 0, 2, 4, and 6 respectively.

As an example, consider the scan key  $B5(s2 Z0(c5 b0 o0 s5) c4 d1)$  for a 16×16 image. The scanning path which corresponds to this scan key is shown in **Figure 3.5**. The image is first partitioned into 4 sub regions using the  $B5$  partition order. These 4 sub regions are scanned using  $s2, Z0(c5 b0 o0 s5), c4,$  and  $d1$ . The second sub region is further partitioned into 4 sub regions using the  $Z0$  partition order and the resulting 4 sub regions are scanned using  $c5, b0, o0,$  and  $s5$  respectively.

The pixel values of the compressed frame are changed by a simple substitution mechanism, which adds the confusion and diffusion properties to the encryption method. The permutation and substitution operations are applied in an intertwined and iterative manner. Therefore, significant portions of this function are done with address generators to implement the scan patterns.



**Figure 3.5: Example of SCAN pattern**  
**B5(s2 Z0(c5 b0 o0 s5) c4 d1)**

The encryption is done by the Encrypt( ) function which is described in **Figure 3.6**. The encryption key actually consists of four components, namely, the two scan keys  $k_1$  and  $k_2$ , the random seed integer  $p$ , and the number of encryption iterations  $m$ . These four encryption key components are known to both the sender and the receiver before the communication of the encrypted image. The random numbers can be obtained by a random number generator with seed  $p$ . The keys  $k_1$  and  $k_2$  are specified by the user. The other two keys, spiral  $s_0$  and diagonal  $d_0$ , are fixed as part of the encryption algorithm and they were chosen because they have opposite directions, as shown in **Figure 3.4**.

There are two fundamental properties that every secure encryption method must satisfy. The first is the confusion property, which requires that cipher texts (encrypted data) have random appearance (uniformly distributed pixel values). The second is the diffusion property that takes under consideration the plaintexts (original data) and keys, which requires that similar plain texts produce completely different cipher texts when encrypted with the same key, and similar keys produce completely different cipher texts when encrypting the same plaintext. The proposed encryption method satisfies both the confusion and diffusion properties, as shown above with pseudocode.

```

Encrypt(I, N, k1, k2, p, m, J)
Inputs: Image I, Image size  $N \times N$  ( $N = 2n, n \geq 2$ )
        Encryption keys  $k1$  and  $k2$ 
        Random seed integer  $p$ 
        Number of encryption iterations  $m$ 
Output: Encrypted image  $J$ 
{
    Let  $A, D, G$  be two dimensional arrays of size  $N \times N$  and let  $B, C, E, F, R$  be one-dimensional
    arrays of length  $N \times N$ 
    Generate  $N \times N$  random integers between 0 and 255 using random seed  $p$  and assign to  $R$ 
    Copy  $I$  into  $A$ 
    Repeat  $m$  times
    {
        Read pixels of  $A$  using key  $k1$  and write into  $B$ 
         $C[1] = B[1]$ 
         $C[j] = (B[j] + ((C[j-1] + 1)R[j]) \bmod 256) \bmod 256$ , for  $2 \leq j \leq N \times N$ 
        Read pixels of  $C$  and write into  $D$  using spiral key  $s0$ 
        Read pixels of  $D$  using diagonal key  $d0$  and write into  $E$ 
         $F[1] = E[1]$ 
         $F[j] = (E[j] + ((F[j-1] + 1)R[j]) \bmod 256) \bmod 256$  for  $2 \leq j \leq N \times N$ 
        Read pixels of  $F$  and write into  $G$  using key  $k2$ 
    }
    Copy  $G$  into  $J$  and return  $J$ 
}

```

**Figure 3.5: The pseudocode for the encryption algorithm**

The *decryption method* is done by reversing the operations of encryption. Note that the decryption requires the encryption key which consists of  $k1, k2, p$  and  $m$ . Decryption is done as follows: Read pixels of  $G$  using key  $k2$  and write into  $F$ . Then, transform  $F$  into  $E$  by  $E[1] = F[1]$ ,  $E[j] = (F[j] - ((F[j-1] + 1)R[j]) \bmod 256) \bmod 256$  for  $2 \leq j \leq N \times N$ . Then, read pixels of  $E$  and write into  $D$  using diagonal scan  $d0$ . Then, read pixels of  $D$  using the spiral scan  $s0$  and write into  $C$ . Then, transform  $C$  into  $B$  by  $B[1] = C[1]$ ,  $B[j] = (C[j] - ((C[j-1] + 1)R[j]) \bmod 256) \bmod 256$  for  $2 \leq j \leq N \times N$ . Then, read pixels of  $B$  and write into  $A$  using key  $k1$ . Repeat this process  $m$  times to get the decrypted image. Note that the random array  $R$  is obtained with random seed  $p$ .

### 3.4 SCAN Encryption Algorithm

The data hiding algorithm consists of two main steps. The first step of the embedding data algorithm is the calculation of the complexity matrix. The complexity matrix is a two dimensional matrix whose size is equal to the frame. Its values are 0 except for specific positions where values 0, 1, 2, 3, and 4 are placed depending on values for the corresponding pixels using

thresholds. The second step of the algorithm is the bit embedding process. The second step uses the complexity matrix, which was calculated in the previous step and depending on the values of its pixels changes the bits of the main frame, embedding bits from the secret data. The pseudocode for the embedding data algorithm is shown in **Figure 3.6**.

The algorithm in general works on hierarchical decomposition of the image or video into NXN subframes, which in our implementations are of size 64x64 down to 2x2. To recover the video at the receiver, the receiver must know exactly the values of parameters N, w, h, m, n and k, which are sent to the receiver separately. These parameters effectively determine the SCAN patterns (permutations) that will be applied to the image, and the further decomposition of the image into smaller ones which recursively may be rearranged by SCAN patterns as well.

```

Complexity( I, C, k1,k2,k3,k4)
Inputs: Cover Image I
        Threshold values  $0 < k1 < k2 < k3 < k4 < 255$ 
Output: Complexity matrix C
{
    Let  $C[i][j] = 0$  for  $1 \leq \text{height}(I), 1 \leq \text{width}(I)$ 
    For( $i=2$  to  $\text{height}(I)-1, i=i+2, j=2$  to  $\text{width}(I)-1, j=j+2$ )
    {
        Let  $C[i][j] = 0, 1, 2, 3, 4$  if  $0 \leq d < k1, k1 \leq d < k2, k2 \leq d < k3, k3 \leq d < k4, k4 \leq d < 255$ ,
        respectively
    }
}
EmbedBit( I, C, M)
Inputs: Rearranged Cover Image I
        Rearranged Complexity Matrix C
        Secret data bit stream M
Output: Bits of M are embedded into I
{
    Let  $p = \text{length}(M)$ 
    For( $i=1$  to  $\text{height}(I), j=1$  to  $\text{width}(I)$ )
    {
        If p equals 0
            Stop embedding
        Else
             $k = \text{minimum}\{C[i][j], p\}$ 
            Replace k least significant bits of  $I[i][j]$  with next k bits from M
             $p = p - k$ 
    }
}

```

**Figure 3.6: The pseudocode of the embedding data algorithm**

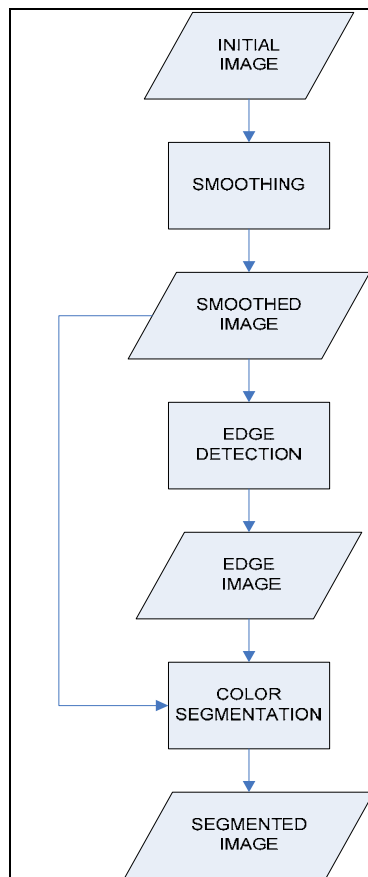
## **Microprocessor Hardware Laboratory**

The procedure that is followed in the receiver is exactly the reverse processes of those that take place in the transmitter. Also, it is important to mention that the transmitter in order to extract the hidden information from the video sequence needs the values of the complexity matrix, which is compressed and sent separately.

## Chapter 4

### Smoothing, Edge Detection and Colour Segmentation Algorithms in Colour Images

This chapter refers to the algorithms of smoothing, edge detection and colour segmentation, which were implemented in the new Stretch technology, and can be applied to colour images. These algorithms can be used in colour images to extract the objects that the image consists of, as shown in **Figure 4.1**. Each one of them implements a specific procedure and the final result is an image divided into its objects which are coloured with the same colour. This information can be used by the algorithm of Automatic Target Recognition which taking the information of the colour for each part of the image can extract the size and the species of each object.



**Figure 4.1: Smoothing/Edge Detection and Color Segmentation process**

Segmentation and edge detection in colour images have been extensively investigated in literature [31-34]. There is a variety of methods with different ways of analyzing colour images and resulting to a unique edge image. Some of these methods [40] use the histogram approach to result to the edge image. Some other methods which do not use histograms are very computationally intensive. Another way for detecting edges at material boundaries is the usage of the hue [35]. The disadvantage of the hue parameter is that the edges between two objects of the same type and colour (same hue value) are lost. Another meter for the edge detection is the colour contrast measure in the RGB colour space. The problem of the edge detection using the colour distance is that the shadows in a colour image create false edges. In this algorithm both the colour contrast in RGB and hue value are used for edge detection. The distance in the colour domain is calculated using the Euclidean distance. Finally it is noteworthy the fact that in this algorithm of edge detection a fuzzy-like thresholds are used instead of hard ones.

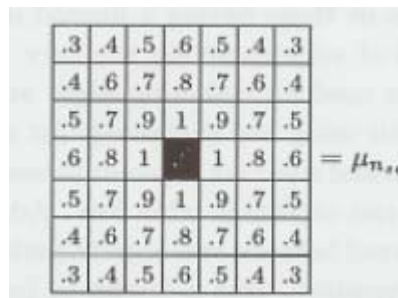
Segmentation is one of the pre-processing steps of image analysis and also one of the oldest problems in image processing. It organises areas of an image into segments that are homogeneous in respect of one or more characteristics. A segment must be composed of a continuous collection of touching pixels where the pieces are not separated from each other. When a segmentation algorithm terminates, every pixel in the image must be assigned to a particular segment. Within the field of image processing, the terms clustering and segmentation may be seen quite frequently. When analysing the colour information of an image and trying to separate regions or ranges of colour components having the same characteristics, the process is called clustering. There are different techniques for image segmentation. One of them is edge detection with disadvantage of the discontinuity of edges in the image [40]. Some others methods use the dichromatic reflection model which describes the colour of reflected as linear combination of the colour of surface [43, 44]. Another segmentation method is the segmentation of the image not in RGB domain but in HSI (Hue, Saturation and Intensity) space. Each one of the above segmentation methods have some disadvantages which are described in **Chapter 2** of this thesis.

In the next sections of this chapter the algorithms of smoothing, edge detection and colour segmentation in colour images will be described.

### 4.1 Smoothing Algorithm

The images contain noise which is introduced either by the camera or because of the transmission of the image over a noisy medium. In either case, these noises must be removed before any further image processing is applied. The most common way of noise removal is the use of filters. For example, a filter that uses a block size of 3x3 window averages the colour of pixels within the block and the centre pixel is then replaced by the average colour of the block.

Before the actual smoothing algorithm is presented, the notion of a degree of neighbourhood between two pixels is defined. This concept is about the definition of how close must be two pixels in order to be considered as neighbours. In this algorithm there is the idea of fuzzy degree of neighbourhood, where for each neighbouring pixel there is the corresponding degree of neighbourhood. The degree of neighbourhood is specified in the table of **Figure 4.2** and it is obvious that the closer two pixels, the higher the degree of the neighbourhood is.



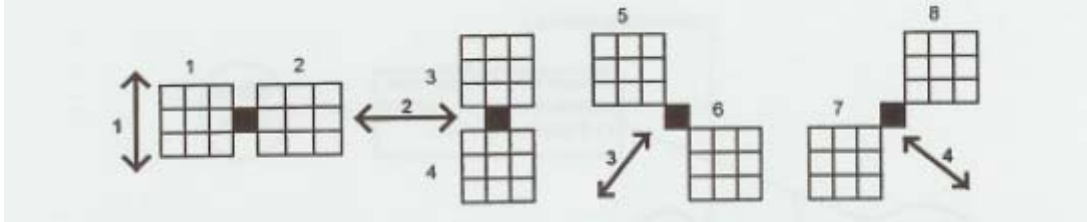
.3	.4	.5	.6	.5	.4	.3
.4	.6	.7	.8	.7	.6	.4
.5	.7	.9	1	.9	.7	.5
.6	.8	1	1	1	.8	.6
.5	.7	.9	1	.9	.7	.5
.4	.6	.7	.8	.7	.6	.4
.3	.4	.5	.6	.5	.4	.3

$= \mu_{n_{sq}}$

**Figure 4.2: Table of degree of neighbourhood**

In  $\mu_{n_{sq}}$  the subscript n indicates the neighbourhood membership function and the s, q is the relative position of a pixel with respect to the centre pixel. The specific values used in this neighbourhood matrix are static priorities based on the pixels' closeness to the centre window. The averaging approach, as described above, would destroy all weak edges and would

create “fake” edges due to edge dilation. In the smoothing algorithm described in this section, each pixel’s colour is compared with the colour of each of its neighbouring blocks, as shown in **Figure 4.3**. The size of blocks for our implementation was 3x3 which results to a strong smoothing of the image.



**Figure 4.3: Eight neighbouring blocks of size 3x3 and four edge directions. Blocks are numbered 1-8 such that they may be referred to in equation of Figure 4.4 (variable b)**

The average colour for each of the neighbouring blocks was calculated taking into account the neighbourhood membership function as shown in equation of **Figure 4.4**, where  $k$ ,  $p$  points to the low left and  $k'$ ,  $p'$  to the top right corner pixel in block  $b$  and  $C_{sq}$  represents the colour vector of the pixel at location  $sq$ . This equation evaluates the average colour vector of block  $b$  with respect to the  $i, j$  centre pixel. For smoothing, the colour contrast between the centre pixel and all of the surrounding blocks must be measured. The colour contrast between the pixel  $(i, j)$  and the block of  $b$  is the Euclidean distance in RGB domain as shown in **Eq. (1)**.

$$Contrast_{ij,b} = (R_2 - R_1)^2 + (G_2 - G_1)^2 + (B_2 - B_1)^2 \quad (1)$$

$$\bar{C}_{ij,b} = \frac{\sum_{q=p}^{p+3} \sum_{s=k}^{k+3} \mu_{sq} \cdot C_{sq}}{\sum_{q=p}^{p+3} \sum_{s=k}^{k+3} \mu_{sq}}$$

**Figure 4.4: Equation of average colour for the neighbouring blocks**

After the contrast calculation between each of the neighbouring blocks and the central pixel the maximum and the minimum contrasts are found. Considering these values, three cases are encountered:

**Case 1:** Both minimum and maximum contrasts are below a threshold  $\tau_{sm}$ . This case represents a situation in which there is little contrast around the

centre pixel. Thus, the pixel most likely is not part of an edge and is probably located in a contiguous region. By replacing this pixel's value with the average of surrounding pixels' color values, any existing color contrast is smoothed.

**Case 2:** Only the minimum contrast is below the threshold  $\tau_{sm}$ . This situation occurs when the pixel's color that is being processed is similar to one side of the edge and different from the other side. Taking the average color between the pixel's color and the color of the block of pixels with the lowest contrast results to enhance the edge contrast.

**Case 3:** The third case occurs when both the minimum and the maximum contrasts are above  $\tau_{sm}$ . This case is when the pixel is isolated and it is considered as noise and therefore it must be removed. If the noisy pixel is located in a contiguous region, there was no problem to be removed and replace it with the average color of the neighboring pixels. On the other hand, if this pixel is on an edge the averaging would increase the fuzziness of the edge. In this case a contrast measure is evaluated for the four neighboring blocks (north, south, east and west). The average color contrast between each side and the other three sides is calculated and added to the color contrast with the centre pixel. The side of the pixel that has the lowest measure is chosen and the pixel's color is replaced.

**Figure 4.5** shows the flow diagram of the smoothing algorithm applied on a pixel at location  $i, j$  with a block size  $3 \times 3$ . Overall, results produced by the algorithm are relatively insensitive to the specific threshold choice. Conversely, the algorithm is time consuming as there is a fair amount of computation for each pixel. After the algorithm is applied, noisy pixels are effectively eliminated, spikes are smoothed and edges are enhanced.

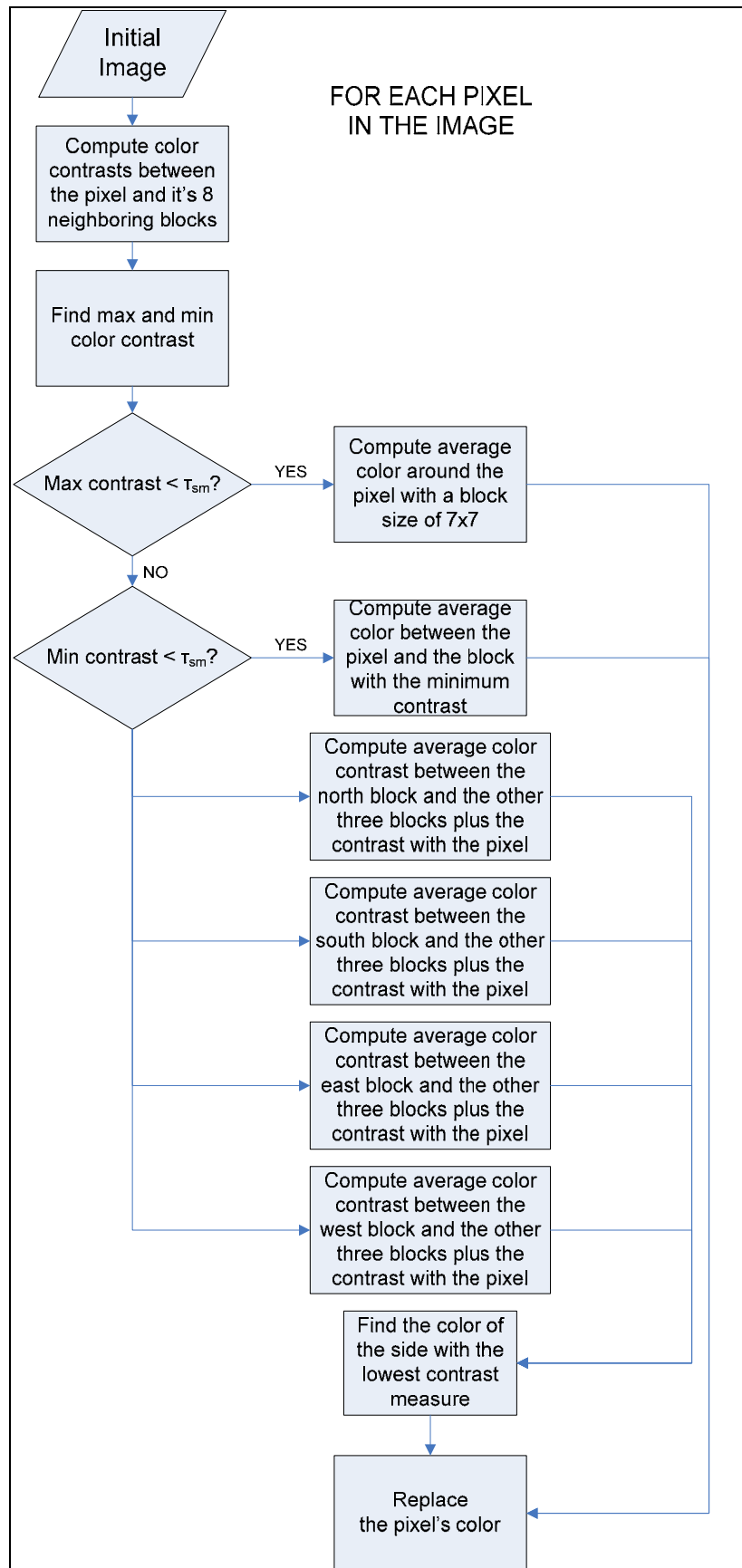


Figure 4.5: The flow chart of smoothing algorithm for pixel  $i, j$  and a block size of  $3 \times 3$

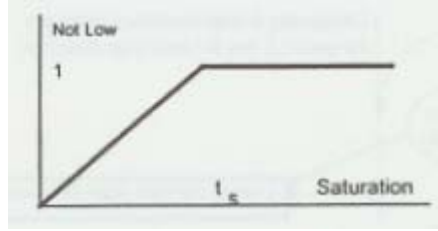
## 4.2 Edge Detection Algorithm

Hue, Intensity and Saturation are one set of parameters that are used to evaluate edge strength within images. These parameters are computed using the RGB values by the equations shown in **Figure 4.6**, where  $i$  is the intensity,  $s$  is the saturation and  $h$  is hue in the range of  $(-180, 180]$  degrees and  $X_0$ ,  $Y_0$  and  $Z_0$  are the x, y, z values of the white color.

$$\begin{aligned}
 x &= 0.49r + 0.31g + 0.2b \\
 y &= 0.177r + 0.812g + 0.011b \\
 z &= 0.01g + 0.99b \\
 l &= 116 \left( \frac{y}{Y_0} \right)^{\frac{1}{3}} \\
 a &= 500 \left[ \left( \frac{x}{X_0} \right)^{\frac{1}{3}} - \left( \frac{y}{Y_0} \right)^{\frac{1}{3}} \right] \\
 b &= 200 \left[ \left( \frac{y}{Y_0} \right)^{\frac{1}{3}} - \left( \frac{z}{Z_0} \right)^{\frac{1}{3}} \right] \\
 i &= l \\
 s &= \sqrt{a^2 + b^2} \\
 h &= \tan^{-1} \left( \frac{b}{a} \right)
 \end{aligned}$$

**Figure 4.6: Equations for the calculation of hue, intensity and saturation**

In the first steps of the algorithm, the values of the  $h$ ,  $s$  and  $i$  are computed for all eight blocks around a pixel. An object has the same hue through out, regardless of variances in shades, highlights and shadows. On the other hand, hue is unstable at low saturations and intensities therefore the needs to be normalized. Hue edges are present at locations at which hue contrast is high while saturation and intensity are both *not low*. The idea *not low* is a fuzzy one and is defined by the **Figure 4.7** for saturation.



**Figure 4.7: Not low membership function**

In our implementation the threshold  $t_s$  was chosen 20% (of the max value that saturation can take) for the saturation and 40% (of the max value that saturation can take) for the intensity, as shown in Eq. (2), (3). Hue contrast is multiplied by these *not low* membership functions to take the normalized hue contrast.

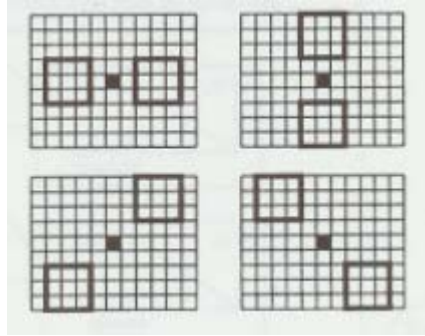
$$\mu_i = \begin{cases} 1 & , i \geq 46 \\ \frac{i}{46} & , i < 46 \end{cases} \quad (2)$$

$$\mu_s = \begin{cases} 1 & , s \geq 108 \\ \frac{s}{46} & , s < 108 \end{cases} \quad (3)$$

$$Hue_{norm} = h * \mu_i * \mu_s \quad (4)$$

The problem with the hue is in case where two objects of the same type are touching each other. To correct this problem the square of the Euclidean distance in the RGB color space is used. This distance is calculated for each edge direction (there are four edge directions as shown in **Figure 4.3**) and averaged with the normalized hue contrast of the same direction. These values are the edge candidacies for the pixel in each one of the four directions.

After edge candidacies are calculated the maximum is found. If the max candidacy value is bigger than a smoothing threshold ( $t_s = 11$ ) then the pixel is considered as an edge. If this value is lower than a low threshold ( $t_l = 7$ ) then this pixel is not considered as an edge. Finally, if this value is between the low and high threshold then the whole process is repeated with the neighboring blocks one pixel away from the processing pixel, as shown in **Figure 4.8**. This procedure is repeated once and it is calculated for the four edge directions resulting to four edge strength images. The previous results show maximum edge candidacies in the four directions.



**Figure 4.8: Blocks used to reevaluate color contrast for edges**

The presence of a local maximum depends on the ratio of the current pixel edge strength with respect to the maximum of that of the two neighbouring pixels in the edge direction. If this ratio is greater or equal to 0.6 the pixel is considered as an edge for the specified edge direction. The four edge strength images are merged keeping for each pixel the highest edge candidacy and its edge direction. In the next step, the local maximum edge candidates are found, as in the previous step, and putting the white colour for the edge pixels and black for the others results the final edge image. The flow diagram of the edge detection algorithm is shown in **Figure 4.9**.

### 4.3 Colour Segmentation Algorithm

The segmentation algorithm uses edge information and the smoothed image to find segments. The processes involved in this segmentation procedure are as follows:

1. Find big and crisp segments.
2. Expand segments based on homogeneity criteria.
3. Expand segments based on dichromatic reflection model.
4. Expand segments based on degree of farness measure.
5. Apply an iterative filter.
6. Find medium size segments.
7. Expand segments using homogeneity criteria and degree of farness.
8. Fill in blank regions.
9. Apply an iterative filter.

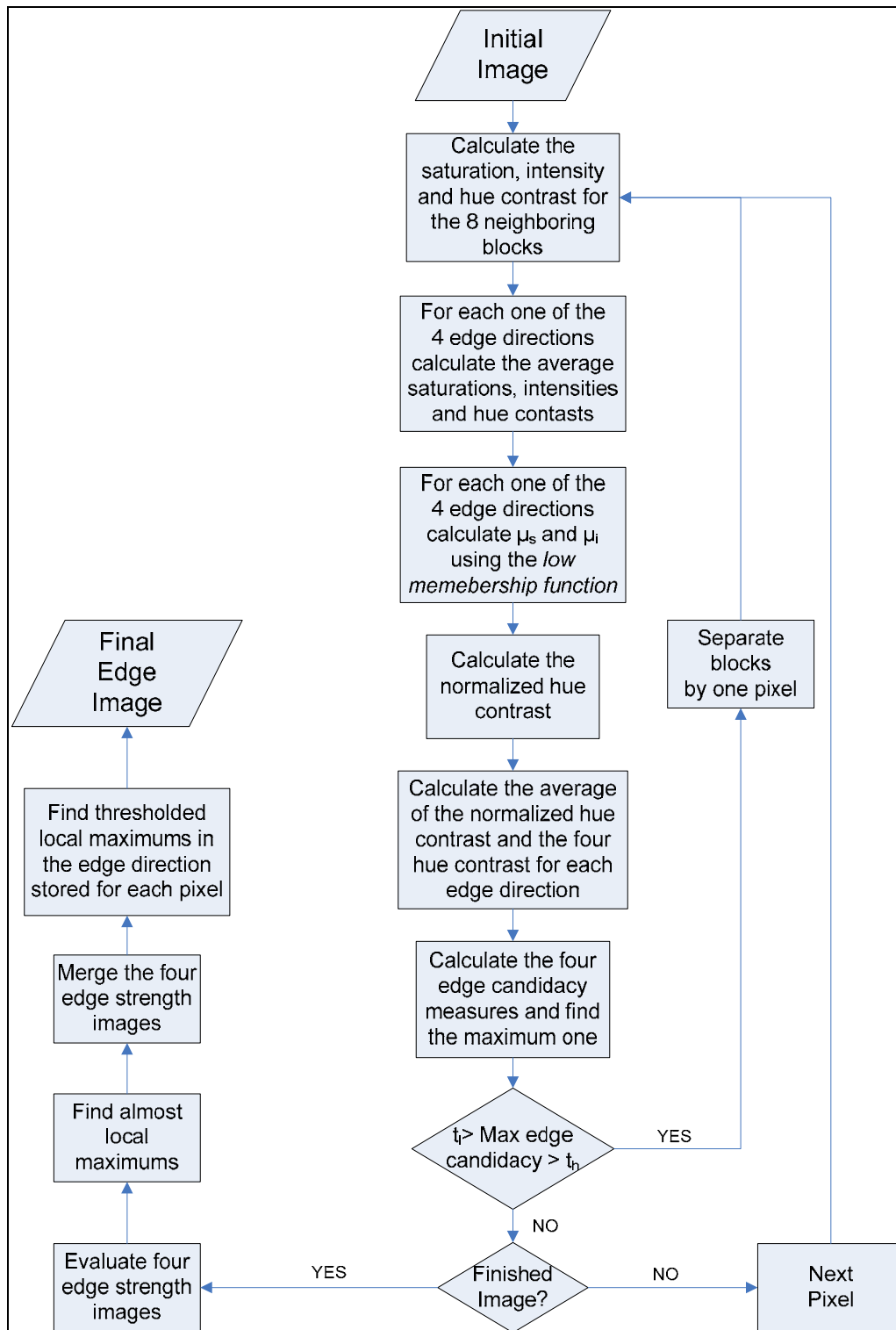


Figure 4.9: The flow chart of edge detection algorithm

It processes big regions first, and then expands them based on three criteria: homogeneity, the dichromatic reflection model and degree of farness measure. After that, it applies an iterative filter and processes the medium size regions. Then it uses homogeneity and the degree of farness criteria to

expand further. For medium size regions the same procedure is used, with the exception that the dichromatic reflection model is not applied.

### 4.3.1 Find Big and Crisp Segments

The first step of the colour segmentation algorithm is the process of finding big and crisp segments. Once edge detection has been performed on an image crisp

segments are surrounded by edge pixels or the image boundary. Specifically, a crisp segment can be defined as a set of pixels completely surrounded by edge pixels belonging to only one object. To find a crisp segment, the image is first scanned for the first non-edge pixel. This pixel is used as a growing seed. In the growing process, a pixel can grow recursively in four directions (left, right, up and down), and merged with the seed if the growing condition is met. The steps are shown below:

1. Scan line-by-line (left-to-right, top-to-bottom), find the first non-edge pixel, and use the first point as seed point.
2. Grow from  $P_n$  along four directions: left, right, up and down.
  - (a) Find  $P_n$ 's next neighbour point  $P_i$ .
  - (b) Test if  $P_i$  satisfies merge condition.
    - $P_i$  is not an edge pixel.
    - $P_i$  doesn't belong to any regions.
    - the distance (RGB) between  $P_i$  and the region  $R_i$  is less than threshold  $T_d = 30$ . The average value of all region pixels is used to be the region colour.
  - (c) If  $P_i$  satisfies the above conditions, merge it into the region  $R_i$ ; otherwise, test next neighbour points.
3. If none of  $P_n$ 's neighbour points satisfy the merge condition, go back one step to the previous point.
4. If it returns to the seed point, stop current procedure, mark the current region  $R_i$ . Go to step 1, start the procedure for a new region.
5. If it reaches the bottom-right corner, the whole procedure ends.

During the seed growing process, the growing region may leak into its neighbouring region if there is even a single undetected edge pixel between the two regions. To avoid this, a seed size of three pixels is chosen. In this way, the growing condition must be true for a block of 3x3 pixels to grow in the growing direction. Thus, a growing region does not leak into a neighbouring region unless at least a 3-pixel wide connecting non-edge area exists. Each merging pixel is marked and assigned to the growing segment.

During the growing process the average colour of each segment is computed, therefore at the end of this part of algorithm each segment is painted with the colour that was computed. The areas of the image, which are not painted with any colour, have not yet been assigned to any particular segment.

### 4.3.2 Expand Segments based on the Homogeneity Criteria

The next step of the segmentation algorithm is the expansion of the segments based on specific criteria of homogeneity. In this step, the initial image is scanned and using the information that resulted from the edge detection step expands the existing segments adding pixels. The expansion procedure is performed within three sub-phases. During these sub-phases, each segment is expanded (surrounding pixels merged with the segment) only if the resultant segment is homogeneous. The degree of homogeneity defined as a fuzzy term. The degree is high if:

1. The **absolute similarity, Eq. (5)**, (similarity between a pixel's colour and the segment's colour) is high

$$similarity\_abs = \begin{cases} 1 & , \quad abs\_contrast \leq 2 * dev\_abs \\ 0 & , \quad abs\_contrast \geq 4 * dev\_abs \\ \frac{4 * dev\_abs - abs\_contrast}{2 * dev\_abs} & , \quad \text{διαφορετικά} \end{cases} \quad (5)$$

2. The **local or relative similarity, Eq. (6)**, (similarity between the next and the previous pixel's colour in the growing direction) is high

$$similarity\_local = \begin{cases} 1 & , \quad local\_contrast \leq 2 * dev\_local \\ 0 & , \quad local\_contrast \geq 4 * dev\_local \\ \frac{4 * dev\_local - local\_contrast}{2 * dev\_local} & , \quad \text{διαφορετικά} \end{cases} \quad (6)$$

where dev\_abs is the standard deviation (computed during the first phase of segmentation, after the big crisp segments in the image are delineated, using the segments' average colour) and the dev\_local is the local standard deviation as it is computed each time.

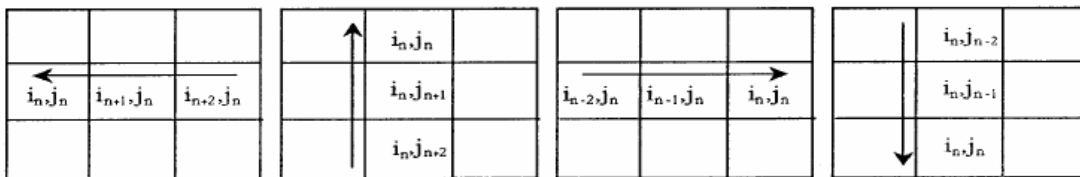
Colour contrast can be measured by computing the difference between two colour vectors and obtaining its magnitude. As shown in **Eq. (7)**, the square of the Euclidean distance is used to calculate the colour contrast between two-colour vectors v and w in this approach:

$$Contrast = (R_v - R_w)^2 + (G_v - G_w)^2 + (B_v - B_w)^2 \quad (7)$$

where R, G and B are the three-colour components (Red, Green and Blue).

If the contrast between the colour vectors, v and w, is less than threshold of colour deviation, the two pixels (or the pixel and the segment in the case of absolute contrast) are considered to have the same colour. If it is higher than another threshold value, they are considered as having different colours. If the contrast is between the two thresholds, the similarity membership function is assigned a value between 0 and 1.

The local similarity membership function is calculated between pixels at location  $(i_n, j_n)$ , and  $(i_{n \pm 2}, j_{n \pm 2})$  as shown in **Figure 4.10**. Averaging is used as an aggregate operator combining the two membership functions to obtain the final homogeneity membership function. If the homogeneity membership function is higher than the selected threshold value, the pixel is merged with the expanding segment.



**Figure 4.10: The position of local pixels with respect to each other for four growing directions, which are shown with an arrow.**

### 4.3.3 Expand segments based on Dichromatic Reflection Model

This model assumes that linear hypotheses from large image areas describe matte pixels on an object or region, or in other words, large linear clusters are matte clusters. This heuristic depends upon their distance from the camera.

The big, crisp regions of the image correspond to the large matte areas of objects. During the first step of the segmentation process, all big matte segments are found by using edge information. The weighted average colour of the segments and the standard deviation are also computed. All matte segments expand to include most nearby matching matte pixels. The shadowed and highlighted object areas are leavened out because they have a very different colour compared to the matte segment. Given the colour vector of the centroid and the illumination (usually white), the cluster plane can be found. The normal of this plane is the result of the cross-product of the two colour vectors.

Using the dichromatic reflection model, some touching pixels may be merged with the previously growing matte segment. **Eq. (8)** is used to calculate a fuzzy measure, referred to as the customised distance  $d_c$ , between the merging pixel and the cluster plane  $k$  in the colour domain.

$$d_c = d * similarity\_local * \min \left( 0.2, \frac{\left| \overline{C}_k \right|}{\left| \overline{C}_{ij} \right|} \right) * \min \left( 1, \frac{\sigma_\tau}{\sigma_\kappa} \right) \quad (8)$$

where  $d$  is the Euclidean distance in pixels between the merging pixel and the cluster plane in the colour domain,  $local\_sim$  is the local similarity function as mentioned in **Eq. (6)**,  $\left| \overline{C}_k \right|$  and  $\left| \overline{C}_{ij} \right|$  are the magnitudes of the expanding segments and the merging pixel's colour vectors,  $\sigma_\kappa$  and  $\sigma_\tau$  are the cluster's and the local standard deviation, correspondingly, as they resulted by the first and second steps of the segmentation algorithm. From the equation of the

distance is clear that the distance depends on the segments' standard deviation, the distance in color domain and the magnitude of each pixel.

The value of standard deviation is used to prevent dark pixels from merging to expanding segments and to assure highlights pixels to be put with the segment that they belong to as they may be scattered at further distances from the plane. Using this factor, if the merging pixel is a highlight one, its intensity would be higher than the cluster's, and the factor would be less than 1. The distance between the pixel's color and the segment's color is used as a normalized meter of the final distance between the pixel and the segment.

Some of the detected segments refer to the dark background or shadow, and are not a part of any object. Merging shadow areas of an object with the previously found object segments is permissible. However, expanding or growing an actual dark or shadow segment into other regions should not be permitted. Since the shadow segments have low intensities and are very close to the origin, the cross product of the illumination vector with the almost black colour would most likely yield erroneous results. Therefore, the dichromatic reflection model didn't apply to shadow segments. Finally, a hard threshold ( $t_h = 0.7$ ) is used and if the distance between the pixel's color and the segment's color is below the threshold then the pixel is put in the segment otherwise the process continues with the next unsegmented pixel.

### 4.3.4 Expand segments based on Degree of Farness

To further expand segments, the degree of farness measure is used. An unassigned pixel can be close (not far) to a neighbouring segment in two senses: close in the spatial domain (physically close); or close in the cluster domain of the colour cube (almost of the same colour). The degree of farness of a pixel to a neighbouring segment is defined as a product of these two measures. Specifically, the degree of farness for any given pixel is the absolute colour contrast multiplied by the geometric distance (in pixels) between the given pixel and the segment border. The closest segment to the pixel is the one having the lowest degree of farness. Specifically, when expanding any segment, the pixel to be merged must touch the segment, or the newly expanded area and the degree of farness to this segment are the

lowest among others computed for that pixel. The algorithm for the degree of farness is shown below:

1. Scan the pixels that lay around the processing pixel and find the segments that these pixels belongs to.
2. Repeat the above procedure for the three rows and lines away around the central pixel.
3. For each segment found, compute the distance in pixels from the processing pixel
4. Find the closest segment and put the central pixel in that segment.

### 4.3.5 Iterative filtering

After the segment expansion is complete, the resultant segments' edges are smoothed using an iterative filter. This filter is used for three block sizes of  $3 \times 3$ ,  $5 \times 5$  and  $7 \times 7$ , with the smallest one being applied first. The steps for applying the filters in the image are shown below:

1. Find the number of pixels in the filter which belong to the same segment as the central pixel
2. If this number is more than the half total number of pixels that are found in the filter then the pixel remains untouched.
3. If this number is lower than the threshold then the central changes and belongs to the segment with the maximum support in the filter.
4. This procedure is performed iteratively for the bigger filters.

At the end of this process all the edges of the image appear smooth.

### 4.3.6 Finish segmentation process

This is the final step of the segmentation process. The image is scanned for a non-assigned pixel. For each of these pixels the distance in RGB domain is computed. If this distance is below a threshold then the pixel is put in that segment. Finally, if after the above process there are no-segmented pixels then the image is scanned once again and all those pixels

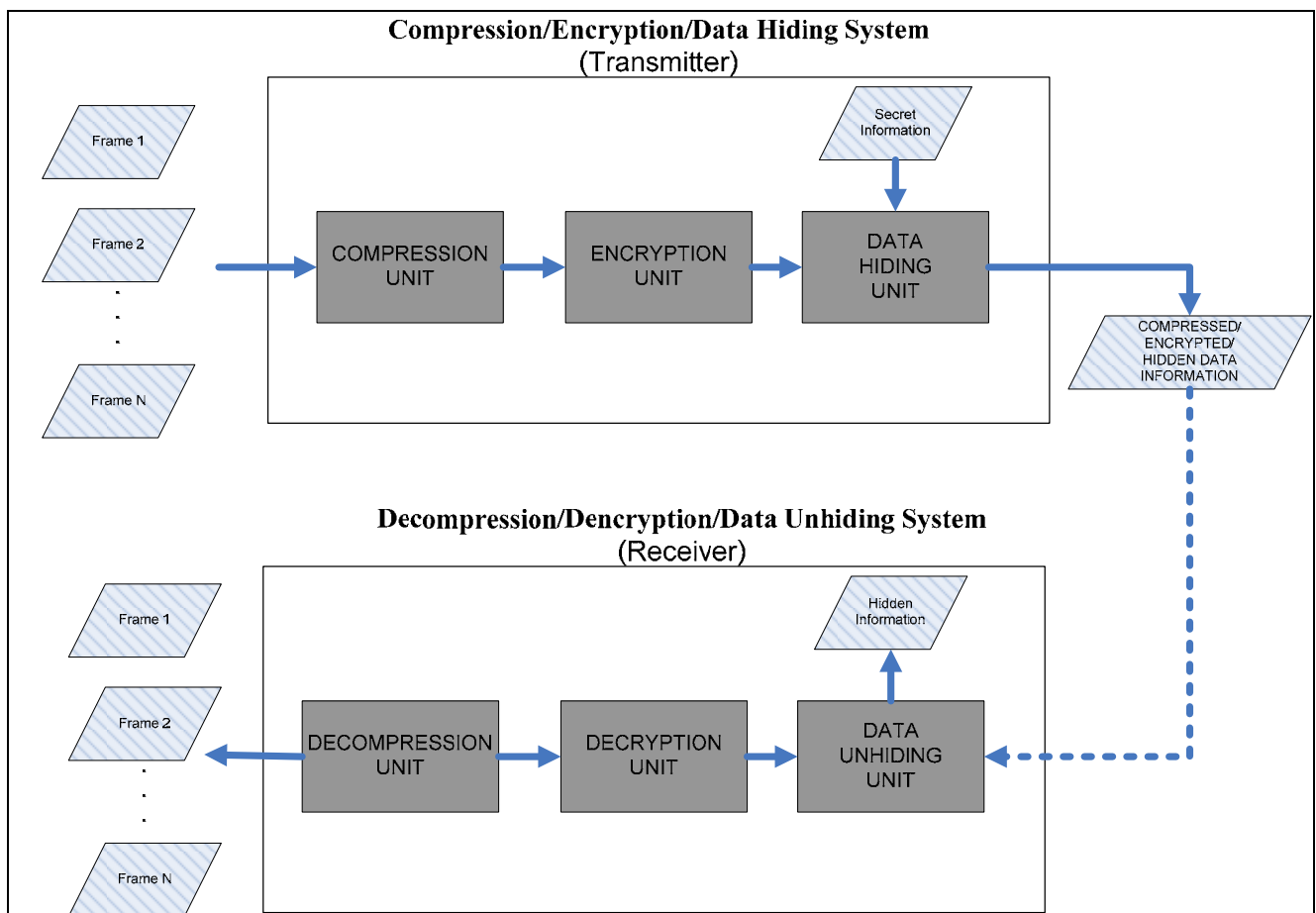
## **Microprocessor Hardware Laboratory**

are put with the segment which is the nearest one physically. Eventually, the whole image is scanned and the average color for each of the segments is computed. Finally, each segment is painted with its color resulting to the final segmented image.

## Chapter 5

### Architecture of Compression/Encryption/Data Hiding and Decompression/Decryption/Data Unhiding Subsystems of SCAN Algorithm

In this chapter, the architecture of the compression/encryption and data hiding subsystems is described. These subsystems are combined in a total system that implements the SCAN algorithm. Additionally, the reverse process of the SCAN algorithm, decompression/decryption and data unhiding, was designed and implemented in Stretch Technology and it is described in this chapter. The general architecture of the two described systems is presented in **Figure 5.1**.



**Figure 5.1: Block diagrams of the Compression/Encryption/Data Hiding and the Decompression/Decryption/Data Unhiding Systems**

## 5.1 Architecture of Compression/Encryption/Data hiding system

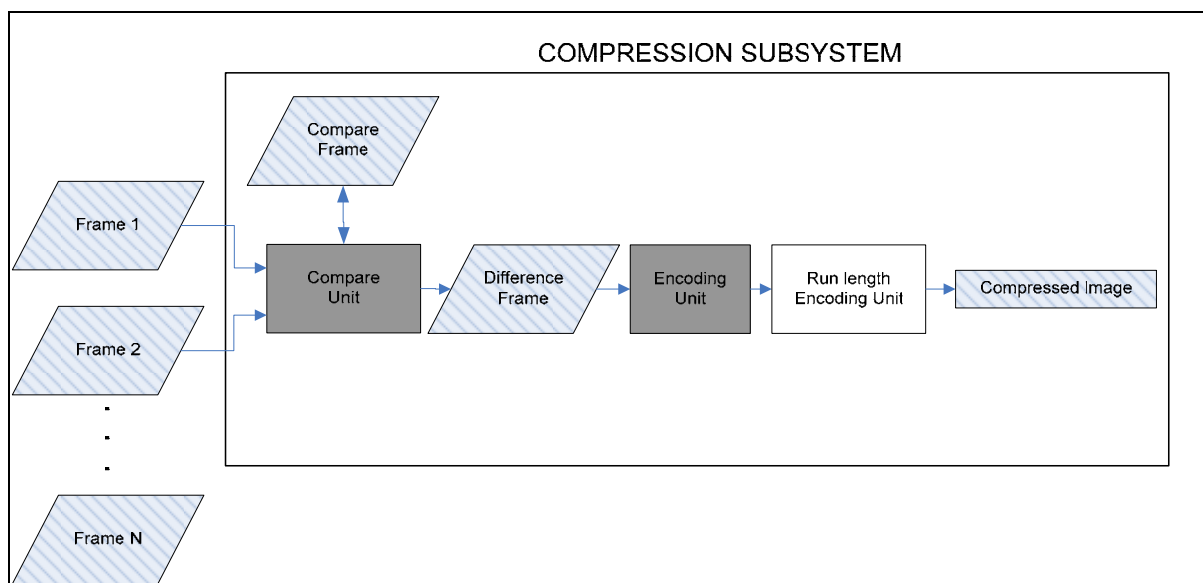
This section describes the architecture of the system in Stretch technology that was designed to implement the SCAN algorithm is described.

### 5.1.1 Architecture of Compression Subsystem

The compression subsystem implements the SCAN compression algorithm, as shown in **Figure 5.2**.

Initially, the two first two frames of the video are read and are stored in the cache of the processor S5000 and in the on board SRAM memory (SRAM 256 KB and Data cache 32 KB). After that, two frames are calculated, one of which contains the values of pixels that will be compared with the next frame (compare frame) according to the compression algorithm and the second one contains the differences between the two frames (difference frame). This process is a computationally intensive one so it was implemented using reconfigurable resources of the embedded FPGA of processor. The fact that the embedded FPGA of the processor has 128-bit wide channels for the communication with the processor lead us to process 16 pixels simultaneously ( $16 \text{ pixels} * 8 \text{ bits/pixel} = 128 \text{ bits}$ ) (compare unit). As a result of this, it was important to specify the way that frames were stored in the memory, i.e we store the value of pixels in contiguous positions of memory in row major order. Subsequently to the computation of compare frame and difference frame, the difference frame is broken to 4x4 windows (16 pixels) and it was specified which of these windows should be put in the compressed difference frame (the windows which contain only the value -1 in each pixel are omitted from the procedure of encoding in the compressed difference frame). The process of this section of the algorithm was also intensive and as a result it was implemented with reconfigurable logic. The problem with this implementation was that the pixels that should be processed were not in contiguous locations and for that reason we developed a reconfigurable logic unit (encoding unit) to process four continuous pixels at a time.

There is a difference between the encoding section of the algorithm and the implementation of the encoding unit in our architecture. The idea of our architecture was that if a window of difference frames was to be encoded each pixel of encoded window was put in 8 bits. In that way, the reverse procedure of calculating the pixels of encoded windows from a compressed difference frame was much easier. Another change in our implementation was that in case of continuous zeros in a compressed difference frame, we use a run length encoding. This procedure takes place in the software unit of run length encoding unit, as described in **Figure 5.2**.



**Figure 5.2: Block diagram of Compression Subsystem**

### 5.1.2 Architecture of Encryption Subsystem

The basic idea of the encryption subsystem is to rearrange the pixels of the image and alter the pixel values so that the histogram of the resulting image is flat. The pixel rearrangement is done by scan keys. The pixel values are changed by a simple substitution mechanism, which adds the confusion and diffusion properties to the encryption method. The permutation and substitution operations are applied in an intertwined and iterative manner. Therefore, significant portions of this function are done with address generators to implement the scan patterns.

The final form of a compressed difference frame, as shown in **Figure 5.2** was not in the right format to be processed by the next subsystem, the encoding subsystem. The encoding subsystem takes as input the compressed image in square form. For that reason, in our architecture the compression subsystem was followed by a transformation unit to convert the final compressed difference frame in a  $N \times N$  frame, where  $N$  is a power of 2. This transformation unit was embedded in encryption subsystem, as described in **Figure 5.3**.

The architecture of whole encryption unit is shown in **Figure 5.3**. The keys are given to the system before the start of the execution and each key is implemented as an algorithm. According to the algorithm the pixels are read with key1 and create an array  $1 \times N^2$ . The reading of pixels was implemented with reconfigurable logic by taking advantage of the parallel copy of 16 pixels from the two-dimensional array to the one-dimensional array. The process of reading and putting the data in the right form takes place in Reading Key1 Unit, **Figure 5.3**. After the reading process follows the substitution unit, which was also implemented in reconfigurable logic and which grouped pixels in groups of 16 that were processed each cycle. The substitution unit implements a multiplication between random numbers, with predefined seed, and the values of the pixels, as the algorithm of encryption defines.

The next two units were the reverse read of an array that creates a 2 – dimensional array according to key s0 (spiral) and after that the read with key d0 (diagonal). These two algorithms, as their implementations were control intensive, gave worse results in reconfigurable logic vs. the software solution and that was the reason why the software implementation was preferred. The problem that these two units were not implemented in reconfigurable part of the S5000 processors was that they do not use pixels in contiguous positions. As a result they had to collect the values of pixels that would be processed from different places of the memory which cycle consuming procedure. This example also shows why a fixed processor with a tightly coupled reconfigurable fabric and high throughput busses between the two units offers the designer good opportunities to partition a design. After these units, the next unit is again the substitution unit, which was described above, and after that unit follows the reverse read with key 2 which lead to the final encrypted

image. This process is repeated  $M$  times which is defined by the user. According to the algorithm, five iterations of this process produce a highly encrypted image.

Generally, the number of keys of SCAN algorithm is large but in our implementation we implemented the subset of the keys that were used in the reconfigurable implementation of algorithm.

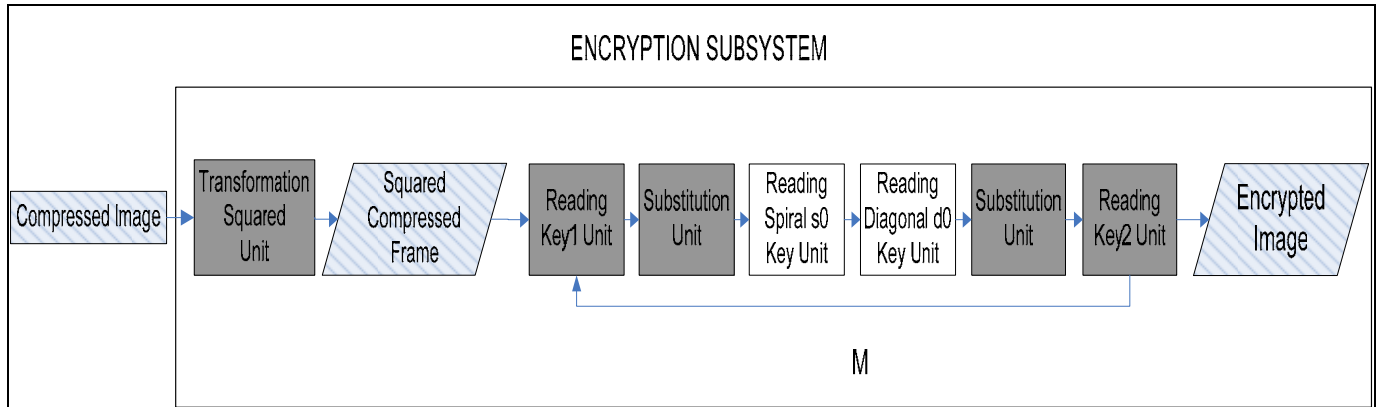


Figure 5.3: Block diagram of Compression Subsystem

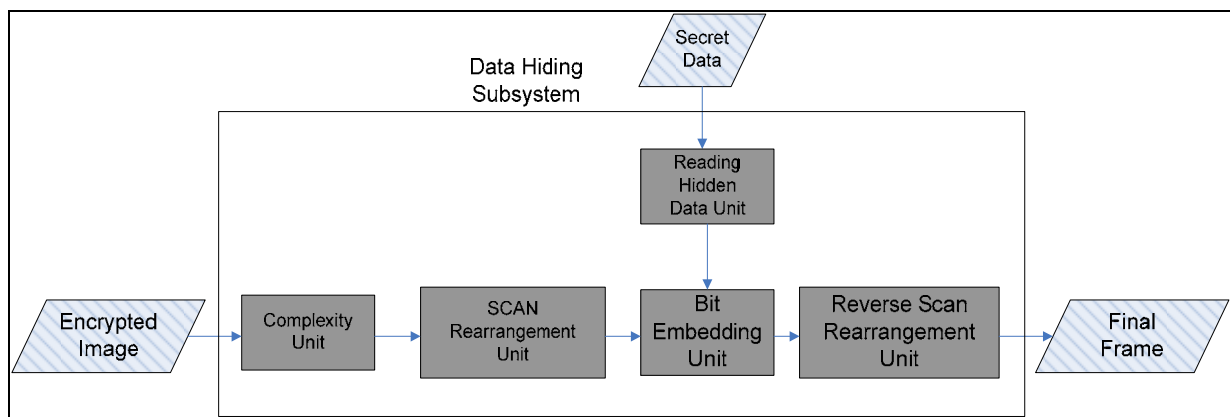
### 5.1.3 Architecture of Data Hiding Subsystem

The main idea of the information hiding algorithm is to identify the complex regions of the cover image and embed the secret data into those regions. The bits from the secret data are embedded into complex regions in random order, determined by the secret SCAN key chosen by the user. The embedding subsystem consists of five main units: (1) the unit that reads the data that are going to be hidden (Reading Hidden Data Unit), (2) the unit that identifies the complexity of cover image (Complexity Unit), (3) the unit that executes SCAN rearrangement of image and complexity matrix (SCAN Rearrangement Unit), (4) the unit that executes the bit embedding of secret data in cover image (Bit Embedding Unit) and (5) the reverse SCAN rearrangement of rearranged image (Reverse SCAN Rearrangement Unit).

In the first step, the image is broken in  $3 \times 3$  windows and is decided which pixels will embed hidden data and exactly the number of bits, which is described in the complexity matrix. This process was implemented in reconfigurable logic although there were tradeoffs: the pixels which were

processed were not in contiguous locations. As a result we constructed an array, where we put the useful pixels each time and in that way the reconfigurable part of the processor had the right data each time. The second unit, SCAN Rearrangement Unit, comprises of the same units as the encryption algorithm with the difference that this unit does not contain the substitution unit.

Following this step is the bit embedding unit where in each pixel is embedded the number of bits that are described in the complexity matrix. This unit in order to be implemented on the reconfigurable fabric had a sizing problem: the required data were wider than the communication channels and the FPGA resources were not sufficient. In order to solve this problem, we broke the Bit Embedding unit in smaller ones and that gave good enough performance. Finally, the last unit of this subsystem is the SCAN rearrangement which changes the order of the rearranged image. This unit was implemented in the same way as the encryption subsystem with the only differences that this unit does not contain the substitution unit and it follows the reverse process from that in encryption subsystem.



**Figure 5.4: Block diagram of Data Hiding Subsystem**

It is noteworthy that the keys that are used in rearranging the image are the same as the keys in the encryption algorithm. In general, due to the nature of SCAN there is good reusability of designs, which may be replicated for performance purposes or used serially if there is no overlap in their need. In that respect, we have a similar design problem as that of pipeline scheduling for CISC computers, albeit at a coarser granularity.

## 5.2 Architecture of Decompression/Decryption/Data Unhiding System

In this section, we will describe the architecture of decompression/decryption/data unhiding system of the SCAN algorithm which can be embedded as a “black box” in the receiver. Given the information of the compressed data that comes out from the compression/encryption/data hiding system, the system of decompression/decryption/data unhiding can recover the initial frames of the video. Complementary to the forward process, the architecture of this system consists of three subsystems the decompression, the decryption and the unhiding subsystem, which are shown in **Figure 5.1**. Generally, the process of each subsystem is the reverse process of the corresponding subsystem of the compression/encryption/ data hiding system. The architecture of each subsystem will be analyzed to the next sections.

### 5.2.1 Architecture of Data Unhiding Subsystem

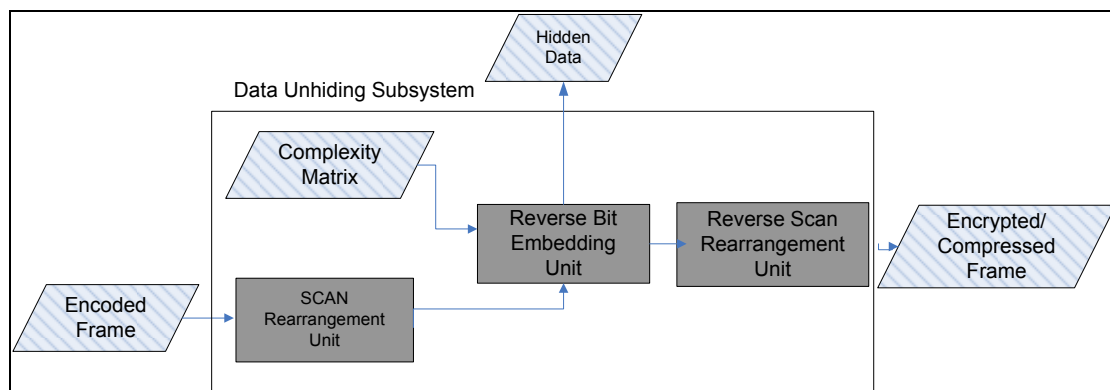
The data unhiding subsystem implements the reverse process of the SCAN data hiding algorithm, as shown in **Figure 5.5**. This subsystem takes as inputs the final frame and the complexity matrix which are produced by the data hiding unit of the transmitter. As described above, the complexity matrix describes the pixels of the frame that were embedded with hidden information in addition with the exact number of bits that were embedded to each pixel. It is noteworthy that the complexity matrix can be compressed in order to avoid the large size of data transmission.

In the first step, the pixels of the final frame are rearranged using the SCAN Rearrangement Unit of data hiding subsystem. It is important to report that the keys that are used for the pixel rearrangement are the same subunits that were used in the data hiding subsystem and that leads to the fact the unit of SCAN rearrangement is a co design of hardware and software subunits, as described in Section 5.3. In the next step, the rearranged frame is scanned and using the information of the complexity matrix the hidden bits, which are embedded in each pixel, are extracted. This process was implemented in the

reconfigurable part of the S5000 processor taking advantage of that each cycle we could process 16 pixels simultaneously. The final step of the architecture of the data unhiding subsystem is the reverse rearrangement of the pixels of the frame using the same keys as in the first step of the subsystem. The output of the subsystem is an array of hidden data that was embedded in the transmitter and the frame which will be processed by the next subsystems in order to recover the initial sequence of video frames.

### 5.2.2 Architecture of Data Decryption Subsystem

The decryption subsystem is shown in **Figure 5.6**. The architecture of this subsystem consists of the same units that were used by the Encryption subsystem in the transmitter. The only difference between the Encryption Subsystem of transmitter



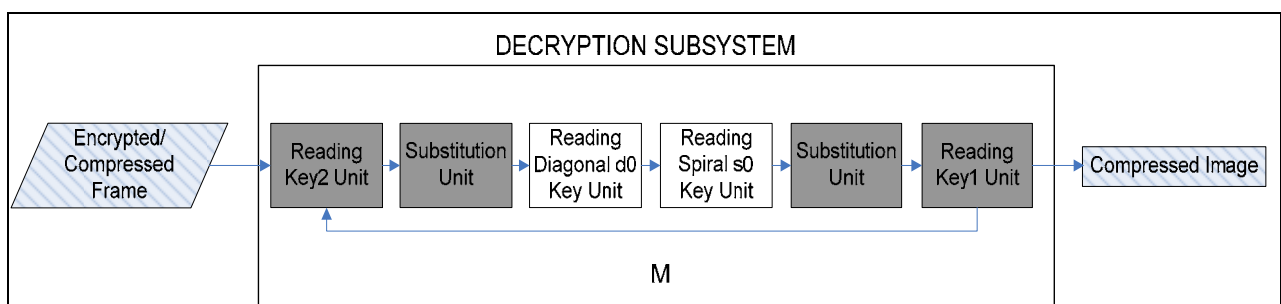
**Figure 5.5: Block diagram of Data Unhiding Subsystem**

and Decryption Subsystem of the receiver is the sequence of the flow of data in the subsystems.

During the first decryption step, the pixels of the frame which come from the data unhiding subsystem are read using the second key of the encryption algorithm. This unit was implemented in reconfigurable part and it is the same unit that was used in the encryption subsystem of the transmitter. The second step processes the substitution of the value of pixels using random numbers that are created using a random number generator. The process of substitution in this unit uses exactly the same random numbers as the Encryption subsystem of the transmitter because both generators of these two

subsystems use the same seed. The unit of substitution executes this operation:  $\text{output}[i] = \text{input}[i] - ((\text{input}[i-1] + 1) * R[i]) \% 256 + 256$ , where  $R[i]$  is the array of random numbers and  $\text{input}[i]$  is one pixel of the frame that results from the previous unit. It is obvious that the process of substitution unit is time consuming as there are multiplications and divisions for each one of the pixels. For these reasons, this unit was implemented in hardware so as 16 pixels of the image could be processed in a few cycles of the processor. The two next steps comprise the reading of pixels of the frame using the diagonal key(d0) and the spiral key(s0), consecutively. It is noteworthy that the reading with the previous control keys takes place in reverse sequence vs. the reading of pixels in the encryption subsystem and because of their complexity they were designed in software. The next step of the procedure is, again, the substitution process and the last one is the unit which scans the frame using the key1, which was also, as the reading with key2, implemented in the reconfigurable part of the S5000 processor. This process will be repeated M times, where M is determined in the Encryption unit and it is sent to the receiver with the rest of the key information. The result is the compressed frames of the initial video.

It is important to mention that the units, which were used for the Decryption Subsystem, were almost the same as the units of Encryption Subsystem, which shows that SCAN algorithm offers substantial reusability of some units. From a cost vs. performance point of view, the same hardware can be configured, with minor modifications, to do the forward or the reverse process.



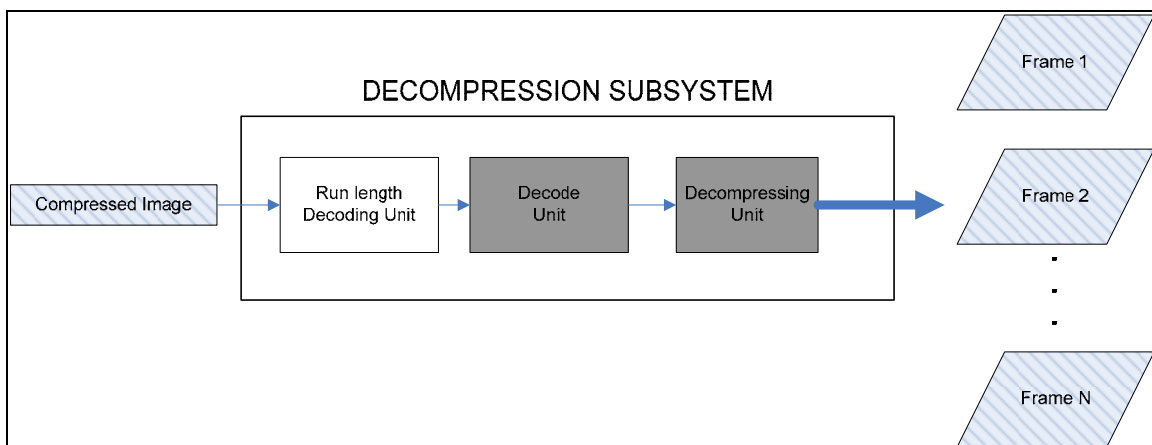
**Figure 5.6: Block diagram of Data Decryption Subsystem**

### 5.2.3 Architecture of Decompression Subsystem

In this section, we will describe the Decompression Subsystem of the Decompression/Decryption/Data Unhiding system. The architecture of the decompression subsystem is shown in **Figure 5.7**. This subsystem comprises of two units which execute the reverse procedure of the Compression Subsystem of transmitter.

In the first step of the process, the pixels of the frame are scanned and decoded in sequence. The decoded unit results into an array of pixels which comprises only of the windows of pixels (4x4) that were encoded. The corresponding positions in the final frame of the pixels, which were omitted by the process of encoding in Compression Subsystem, are replaced by the values of pixels of the previous frame, each time. For the first frame of the sequence, it is essential to know the pixel values. In our architecture, the intensive processes of decoding and decompressing were placed in the reconfigurable part of the processor, which led to an important reduction of the execution time.

The main problem that we faced with the implementation of this subsystem was in the non-contiguous pixels that we should process. This problem was solved by putting the proper positions of the memory to the 128-bit communication channels between reconfigurable part of processor and its core.



**Figure 5.7: Block diagram of Decompression Subsystem**

## **Microprocessor Hardware Laboratory**

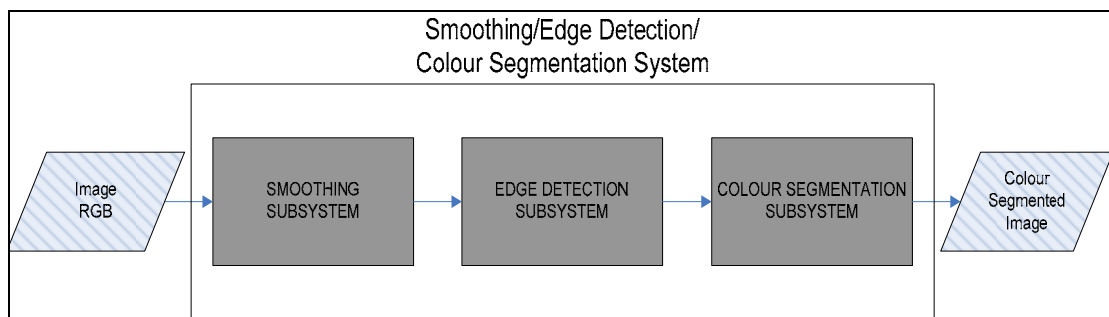
As mentioned above, the implementation of Decompression/Decryption/Data Unhiding system used many of the subunits of the Compression/Encryption/Data Hiding system, which is placed in transmitter. The main idea of the implementation of the unit is that it follows exactly the reverse process from that in Compression/Encryption/Data Hiding system. It is noteworthy that for the execution of the Decompression/Decryption and Data Unhiding process it is essential to use some information that was used by the transmitter, such as the keys that were used to scan the pixels of the frames, the number of the iterations that was used for the encryption process and the value of threshold for the encoding process. In a system that does the forward and the reverse process with the same keys, these need to be held only once in the system regardless of the operation that it performs.

## Chapter 6

### Architecture of Smoothing, Edge Detection and Color Segmentation Algorithm

This chapter describes the architecture of smoothing/edge detection/color segmentation system that was developed using Stretch technology. The system, which implements the first three steps of ATR algorithm described in Chapter 4, consists of three subsystems, the smoothing, the edge detection and the color segmentation subsystem. The system's block diagram is presented in **Figure 6.1**, and the architecture of each subsystem will be described to the next sections of this chapter.

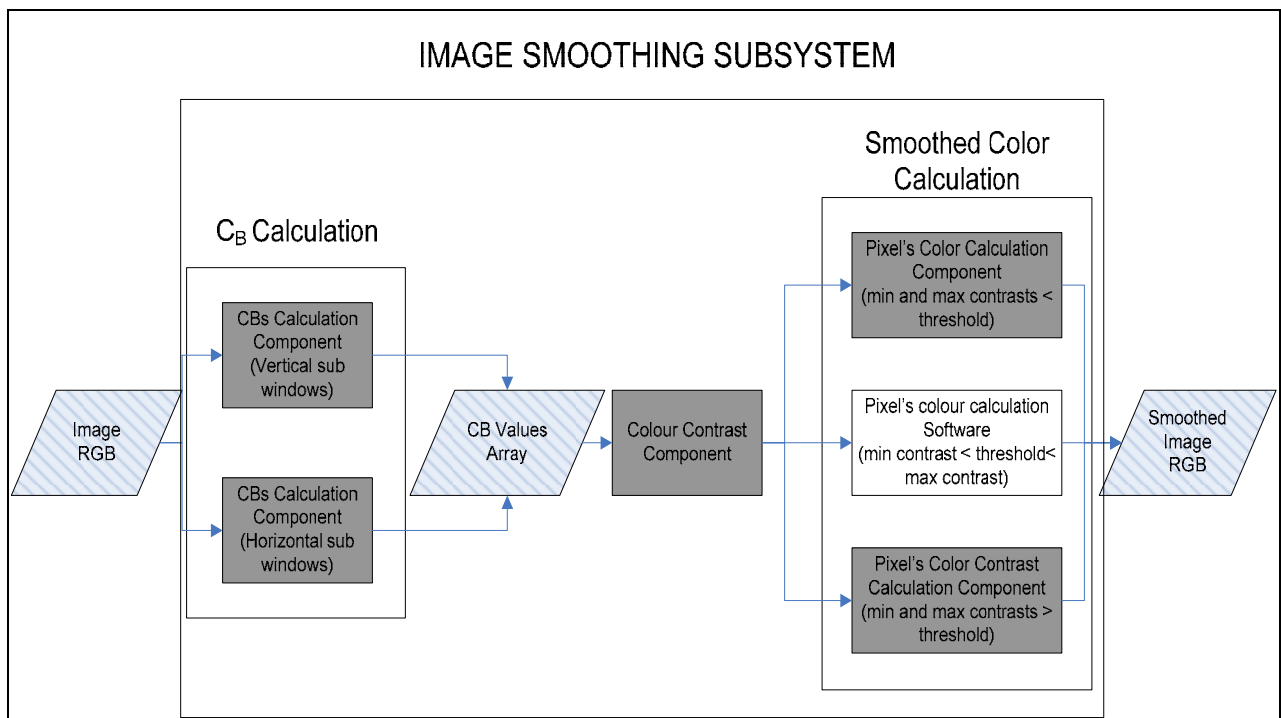
The total system consists of three main components and each of them implements the algorithm for each of the three subsystems. The components, which are implemented as C functions using Stretch technology, communicate with each other through the internal memory of the S5000 processor. Initially, the values of color of the pictures' pixels in RGB are stored in internal memory of the processor. The smoothing component reads these values which are processed and they are stored again to the memory of the system. The process continues for the other two components of the system which read the input data from the memory and stores the processed data. Finally, the output of the system is the values of the pixels colour image where the recognized segments of the picture are colored with the same colour in RGB.



**Figure 6.1: Block diagram of the Smoothing/Edge Detection/Colour segmentation System (hachured boxes are the stored data in memory of the processor)**

## 6.1 Image Smoothing Subsystem

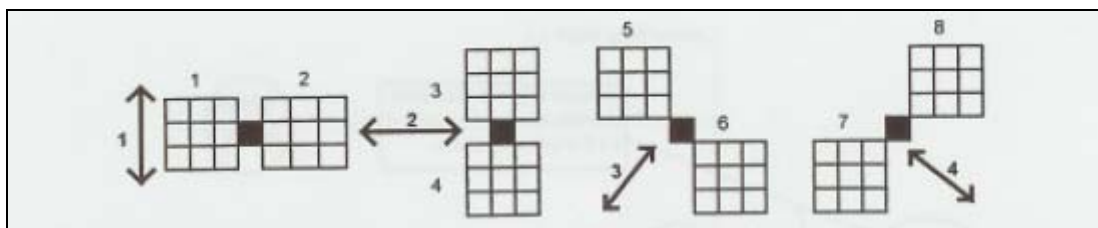
The smoothing subsystem takes as input the initial values of pixels in RGB system and outputs the values of the smoothed image according to the described algorithm of **Chapter 4**. The smoothing algorithm is divided into seven small steps each of one was designed as a separate component, as shown in block diagram of **Figure 6.2**. The dark components of block diagram have been implemented in reconfigurable part of S5000 processors in order to reduce the total execution time of the smoothing algorithm.



**Figure 6.2: Image smoothing subsystem (dark boxes show reconfigurable components, white boxes show software implementation and hachured boxes are the stored data)**

Initially, according to the smoothing algorithm there are 8 neighboring 3x3 windows of each pixel, as shown in **Figure 6.3**. The values of these windows are used to find the  $C_b$ s values between the processing pixel and the surround windows. The value of  $C_b$  shows the notion of a degree of neighborhood between the window and the processing pixel. The calculation of the  $C_b$ s was implemented in reconfigurable part of the processor, as shown in **Figure 6.2**. As it is known from **Chapter 4** one of the restrictions of the

reconfigurable part of S5000 processors is the small number of I/O registers ( $128 \text{ bits} * 3 = 384 \text{ bits}$  at most) which are used for the communication between the reconfigurable and the software part of the processor. The fact that the restriction for the input information in reconfigurable part lead to the calculation of  $C_b$ s for two neighboring sub windows each time, as 2 neighboring  $3 \times 3$  windows contain 15 pixels which means  $15 * 8 \text{ bits/pixel} = 120 \text{ bits} * 3 = 360 \text{ bits} < 384 \text{ bits}$  (as the color of the pixels is in RGB model). The pairs of the sub windows that calculated the values of  $C_b$ s, as shown in **Figure 6.3**, are 5-3, 8-2, 4-6 and 1-7.



**Figure 6.3: Eight neighboring blocks of size 3 x 3. Blocks are numbered 1 – 8**

Another observation during the calculation of the  $C_b$ s is that the direction between the pairs of the sub windows differs as a result to implement another component for the calculation of the sub windows, one with horizontal direction and another one for the vertical direction.

The choice to design the two components for the calculation of the  $C_b$  value in reconfigurable part was taken because of two facts: i) The calculation of the  $C_b$  value contains multiplications, which is a “heavy” task for the software and ii) with this implementation we managed only in **3 clock cycles** to calculate the  $C_b$ s for two  $3 \times 3$  windows. Finally, it is important to mention that as the reconfigurable part of the processor can not be loaded with floating point numbers, the values of the weighted table were, initially, multiplied by the value 1024, loaded to the reconfigurable logic and at the final stage the result was shifted right for ten bits taking the correct value.

The next step of the smoothing algorithm was the calculation of the colour contrast, **Figure 6.4**, between each of the neighbouring windows and the processing pixel. In this stage, a new component of reconfigurable logic was implemented, `smoothing_second_third_step` component. This component takes as input the  $C_b$ s values of the 8 neighboring sub windows and the color

of the processing pixel in RGB domain and calculates the contrast between them. After the calculation, this component returns to the software two values: the first one is a flag that shows which of the three cases of the next step of the algorithm is true and the second one is the id of the sub window which will be processed. This process takes **9 clock cycles** per each calculation.

$$\text{Contrast}_{i,j} = (R_j - R_i)^2 + (G_j - G_i)^2 + (B_j - B_i)^2$$

**Figure 6.4: Contrast value between two color vectors**

The previous step of the algorithm defines which of the next three conditions is true. If the max and the minimum colour contrast between the neighbouring window and the processing pixel are lower than the value of the predefined threshold then the new colour value of the pixel is the weighted average of the 7x7 window where the processing pixel is placed in the centre. The process of calculation is a heavy computational problem as a result it was put in the reconfigurable part of S5000. As there was problem with the big number of values for the pixels that needed to pass to the reconfigurable ( $7 * 7 = 49$  pixels \* 3 RGB color = 147 values \* 8 bits/value = 1176 bits) we took advantage of the fact that the values of  $C_b$ s of the four corner windows were already calculated by the first reconfigurable part of the smoothing algorithm. In that case, we used the calculated values of the four 3x3 windows which were placed in the corners and added the rest of pixels of the 7x7 window. The sum of the values was divided by the sum of the weights of the 7x7 window and the result was the new colour value of the pixel. The clock cycles for this reconfigurable unit of the implementation is about **3 clock cycles** per calculation.

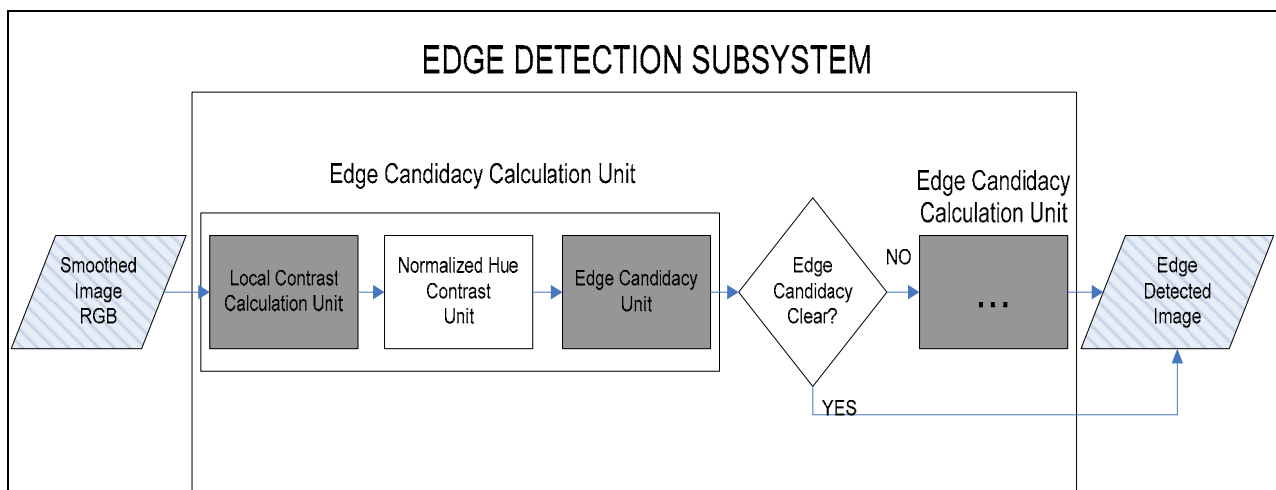
Finally, another computationally heavy part of the algorithm takes place if the minimum and maximum contrasts have larger values than the predefined threshold. In that case, a new hardware component was implemented which took as input the values of the  $C_b$ s for the four windows in four directions (North, East, South and West) ( $3 C_b$ s values of RGB \* 4 windows = 12 values \* 8 bits/values = 96 bits) which were previously calculated. The output of this component is the direction of the sub window

which has the least contrast between the processing pixel and the neighboring sub window. The time that the final reconfigurable component takes for the results is about **7 clock cycles**.

The last stage of the color smoothing component is the storage of the new smoothed color values of the pixels in the main memory of the processor. The next stage of the algorithm which is the edge detection, takes as input the smoothed values from the internal memory of the processor.

### 6.2 Edge Detection Subsystem

The second subsystem of the design takes the pixel smoothed values as they resulted from the image smoothing unit and processes them in order to find the edges of the images. The edge detection unit consists of three sub units: local contrast calculation unit, hue calculation unit and edge candidacy unit, as shown in **Figure 6.5**. In addition, as described in **Chapter 4**, in case a pixel is not clear if it is an edge or not, the calculation of the edge candidacy is repeated. In order to cover the previous situation it has been put a software controller which checks if the edge candidacy is clear or not and in case it is not clear, the sub units of the systems are repeated.

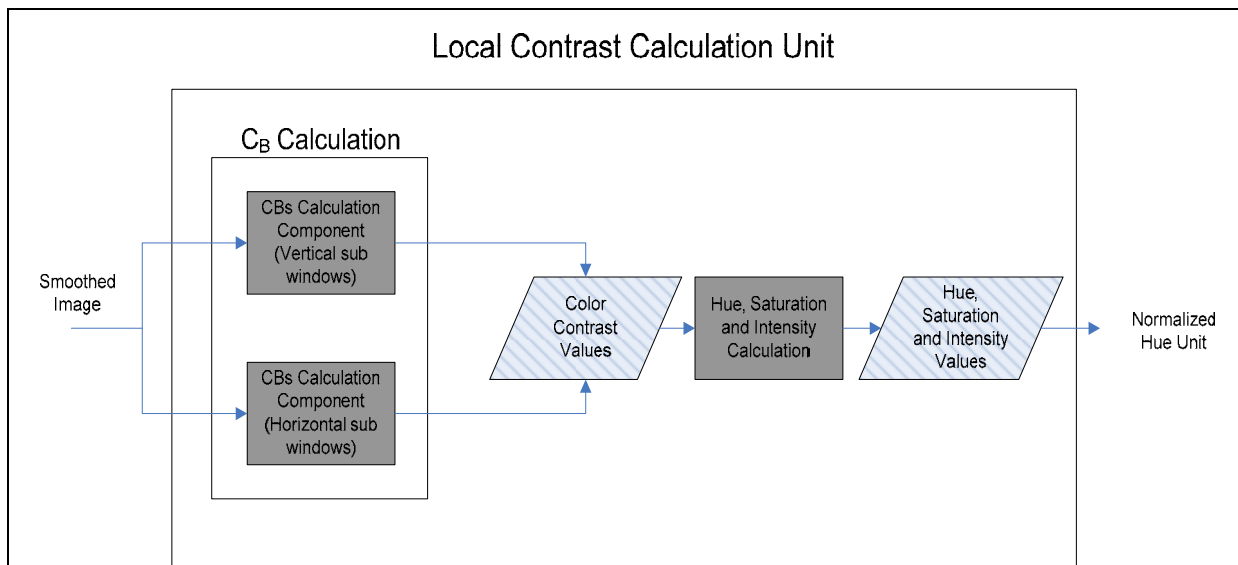


**Figure 6.5: Edge detection subsystem (white objects are implemented in software)**

The local contrast unit takes each time as input, the 7x7 pixels window values which lay around the processing pixel and using the  $C_b$ s calculation

components that were designed for the colour smoothing unit, calculates the colour contrast between the processing pixel and the surrounding 3x3 sub windows, as shown in **Figure 6.6**. The values of the colour contrast for the four directions, **Figure 6.3**, are temporarily stored and used for the calculation of the hue, saturation and intensity for each direction. The resulted values are stored in tables which pass as input to the next unit of the design which is the normalized hue contrast component.

The normalized hue contrast unit takes as input the values of hue, saturation and intensity for each of the four directions and calculates the normalized hue contrast between the processing pixel and the neighbouring sub windows. The normalized hue contrast consists of software multipliers and dividers because the values of saturation and intensity belong to float type and can not be represented in reconfigurable part of the S5000 processors.



**Figure 6.6: Block diagram for the Local Contrast Calculation Unit**

The final value of the edge candidacy is calculated in the last unit of the edge detection subunit, the edge candidacy unit. This unit was fully implemented in reconfigurable part. Hardware adders and dividers for the calculation of the average value of the hue contrast were implemented in the candidacy unit. In addition to this, in this unit two hardware comparators were implemented in order to find the max value of the edge candidacy among the

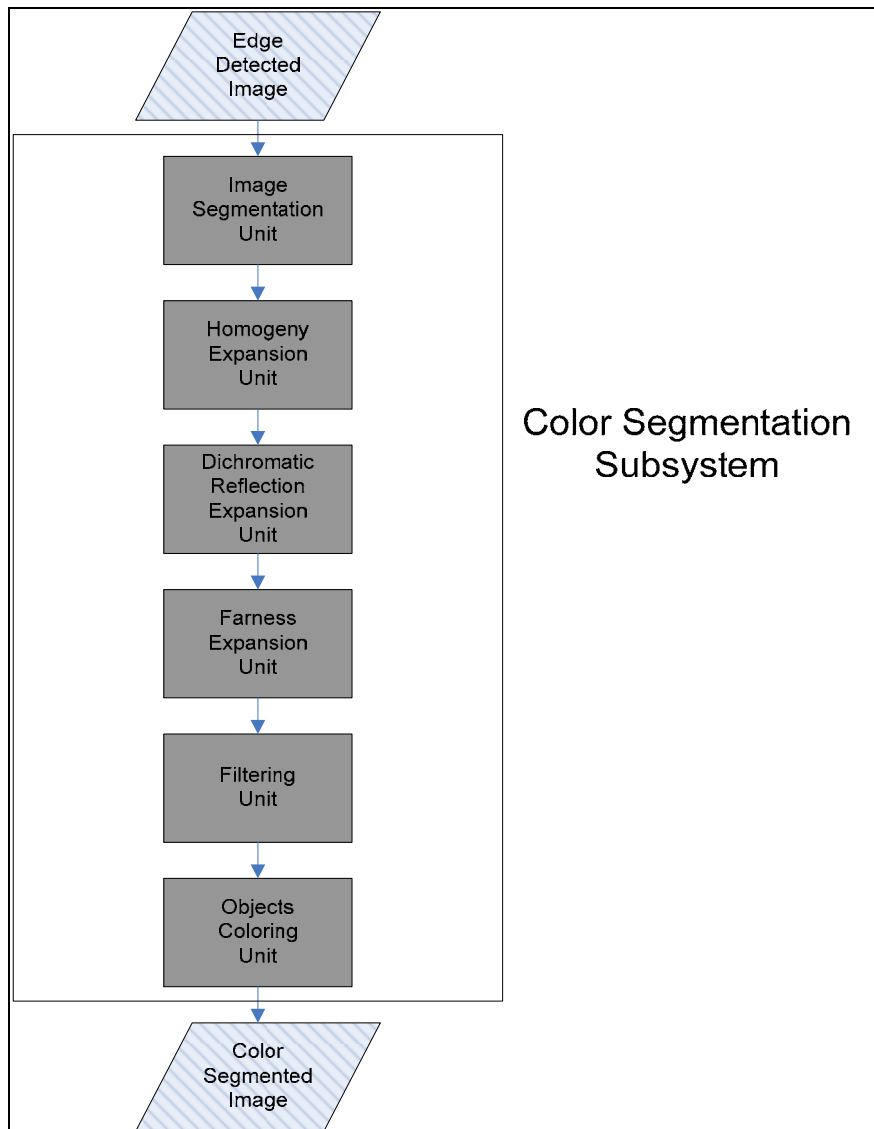
four directions around the processing pixel. This unit takes **7 clock cycles** between the input of data and the output of the results.

At the end of the edge detection subsystem, a new image has been created with the definition of the edge pixels from the initial image. This information is stored to the main memory of the S5000 processor and it is used as the input of the next subsystem which implements the color segmentation algorithm.

### 6.3 Color Segmentation Subsystem

The subsystem, which is described in this section, implements the colour segmentation algorithm as described in **Chapter 4**. The colour segmentation algorithm can be divided into six smaller steps. The block diagram of this subsystem is described in **Figure 6.7**. The colour segmentation subsystem takes as input the values of the edge detected pixels from the main memory of the S5000 processor and divides the objects of the picture colouring them with the average colour of the object. The output of this subsystem is the image with the colored objects which can be used for the next processing steps of the Automatic Target Recognition algorithm.

The first subunit of the system implements the segmentation of the picture in its segments according to the information that comes from the Edge Detection Subsystem. The Image Segmentation Unit, which is described in **Figure 6.8**, consists of three smaller systems: the pixels edge check unit, the grouping segmented pixels unit and the segment colour calculation unit. The pixels edge check unit is fully implemented in reconfigurable part of the Stretch processor and takes as input the value of the processing pixel and the values of pixels which lay around on 3x3 windows and gives out the value of a flag which shows if the pixel can be used as part of a segment. The input of reconfigurable part utilizes only 264 bits of 384 bits as there 33 different pixels which means  $33 * 8 \text{ bits} = 264 \text{ bits}$  and it takes **3 cycles per run**. The



**Figure 6.7: Block diagram of the Color Segmentation Subsystem**

grouping segmented pixels unit is a software unit, which is implemented in software, and groups the pixels which can be put in the same segment, putting an id for each segment, according to the information that came from the previous subunit. Finally, the calculation of the average colour for the segments was implemented in software for one specific reason: there is no information in advance about the number of pixels for each segment; as a result the passing of the colour values of the segmented pixels should be done dynamically something which would be very 'heavy' to be implemented with Stretch technology.

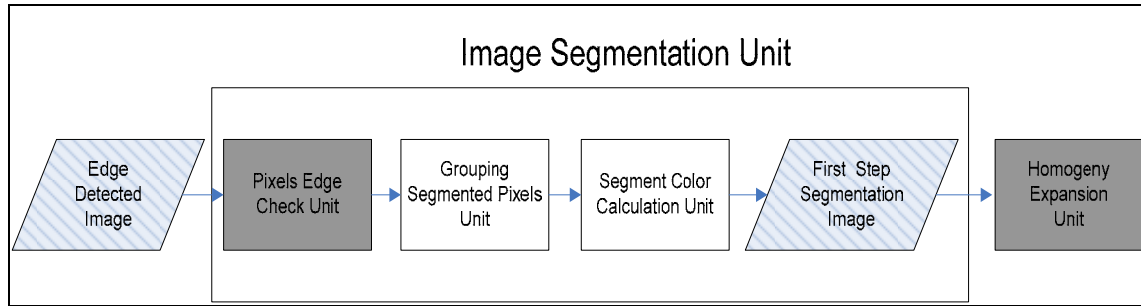


Figure 6.8: Block diagram for image Segmentation Unit

The second subunit of the colour segmentation subsystem is the Homogeny Expansion unit and its block diagram is presented in **Figure 6.9**. This unit implements the algorithm of expansion according to the homogeny criteria using the segmented image as it results from the image segmentation unit. The homogeny expansion unit is divided into two components: the contrast homogeny criteria unit and the calculation of local and absolutely similarity unit. The contrast homogeny criteria unit runs only if one of the neighbouring pixels of the processing pixel belongs to a segment. In that

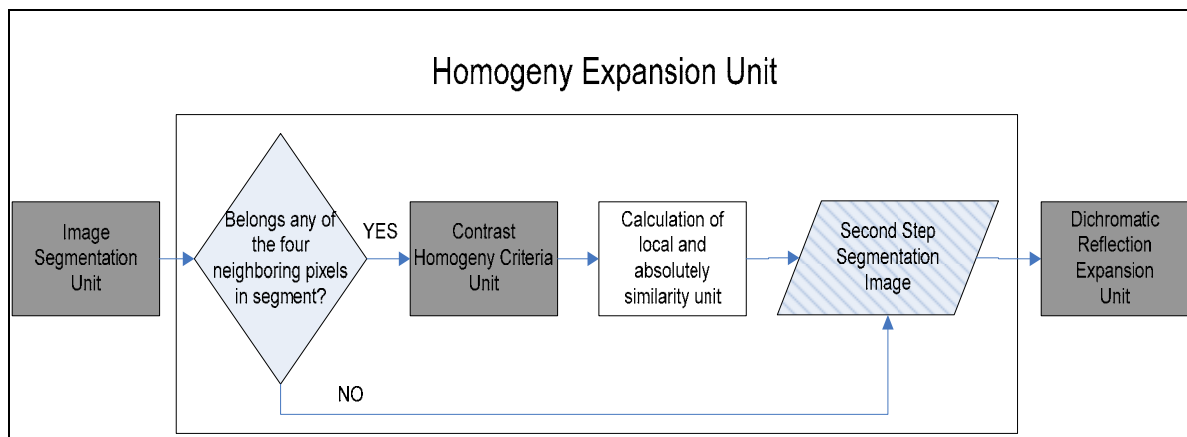
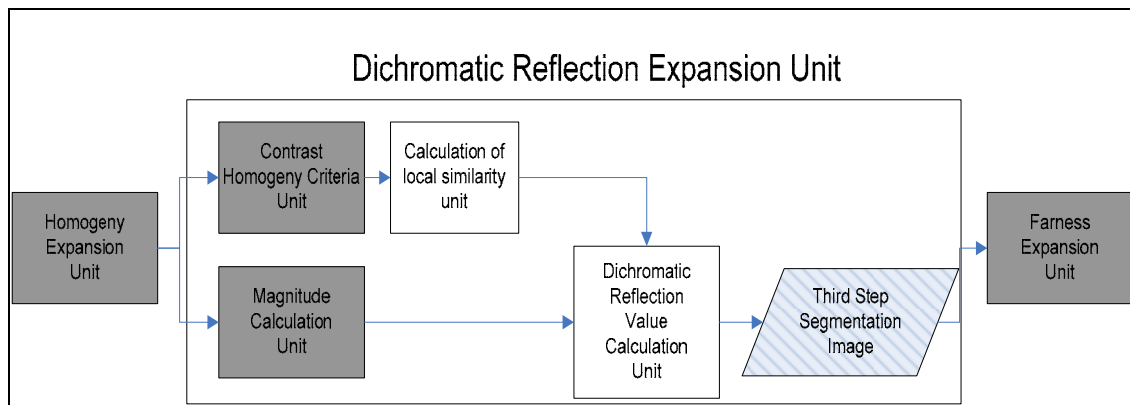


Figure 6.9: Block diagram of the Homogeny Expansion Unit

case, the contrast between the colour of the processing pixel and the neighbouring pixel is calculated and used for the calculation of the absolute and local similarity between them. So the general architecture of this unit consists of a software controller who checks if any of the four neighbouring pixels of the processing pixel belongs to a segment. If any of the pixels belongs to a segment, the control runs hardware unit for the calculation of the colour contrast. Otherwise the processing pixels stay untouched. The next

step of the contrast calculation is the calculation of the local and absolute similarity for each of the four directions and the placement of the processing pixel to the segment with the bigger values of similarities.

The third step of the colour segmentation algorithm is the segments expansion based on dichromatic reflection criteria, as referred in Chapter 4. The calculation of the dichromatic reflection for a pixel needs the calculation of the local similarity between the pixel and the neighbouring segment and the calculation of the magnitude of pixel colour. The architecture of the dichromatic reflection unit is presented in **Figure 6.10**. The part of the algorithm which was implemented in hardware consists of two components: the first hardware subunit is the Contrast Homogeny Criteria Unit, which was used in Homogeny Expansion Unit and which calculates the local contrast between the processing pixel and the neighbouring segments for the calculation of the local similarity. The second component that was implemented is the Magnitude Calculation Unit which computes the magnitude for each segment or pixel taking colour RGB as shown in **Figure 6.11**. In addition, software multipliers and adders were designed for the calculation of the dichromatic reflection value and the new colour of each expanded segment.



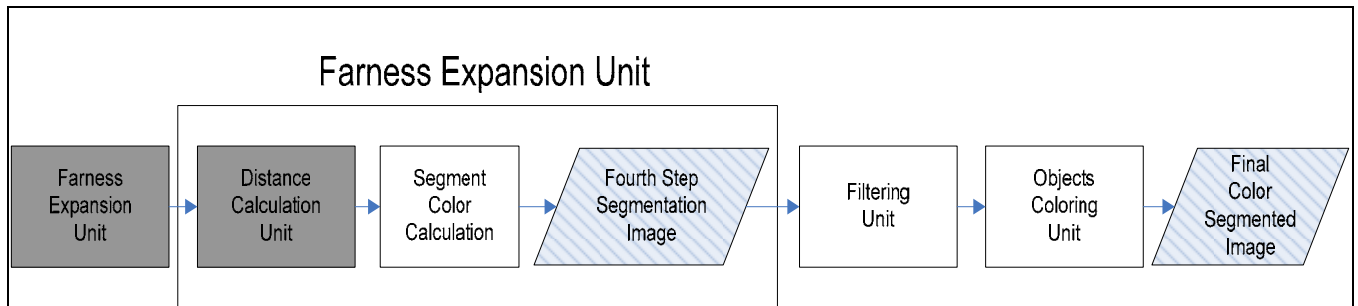
**Figure 6.10: Block diagram of the Dichromatic Reflection Expansion Unit**

$$\text{Color Magnitude} = \sqrt{R^2 + G^2 + B^2}$$

**Figure 6.11: Color magnitude for the calculation of the dichromatic reflection value**

The final expansion of the algorithm takes place according to the degree of farness of the unsegmented pixels from the neighbouring segments. The Farness Expansion Unit takes as input the values of the neighbouring pixels of the unsegmented pixels and checks which segment is the nearest one to the processing pixel, expanding the segment with that pixel. In the architecture of the Farness Expansion Unit that is presented in **Figure 6.12**, it is clear that most of the implementation has been done in software and only the calculation of the distance between the processing pixel and the neighbouring pixels is calculated in reconfigurable part of the processor. In addition to the calculation of distance, the reconfigurable part finds the nearest value and returns to the software a flag which shows the segment of the image that will be expanded with the new pixel.

Finally, the next two steps of the algorithm, filtering and object colouring, were fully implemented in software. The fact that there is no knowledge a priori about the size of each segment so that cannot be managed the parallelism in computation for the image pixels drive us to implement the two steps of the algorithm fully in software, as shown in **Figure 6.12**.



**Figure 6.12: Block diagram for the last three steps of color segmentation algorithm (degree of farness expansion, filtering and final object coloring)**

In this chapter, the architecture approach of the smoothing-edge detection and color segmentation was presented. The important point of this chapter analysis was the way of breaking of the three algorithms in smaller steps and the implementation some of these smaller steps in reconfigurable part of the S5000 processors. The whole procedure had as a result the reduction of execution time through the parallel processing of pixels and the implementation of computationally 'heavy' processes in hardware.

## Chapter 7

### Performance and Comparisons

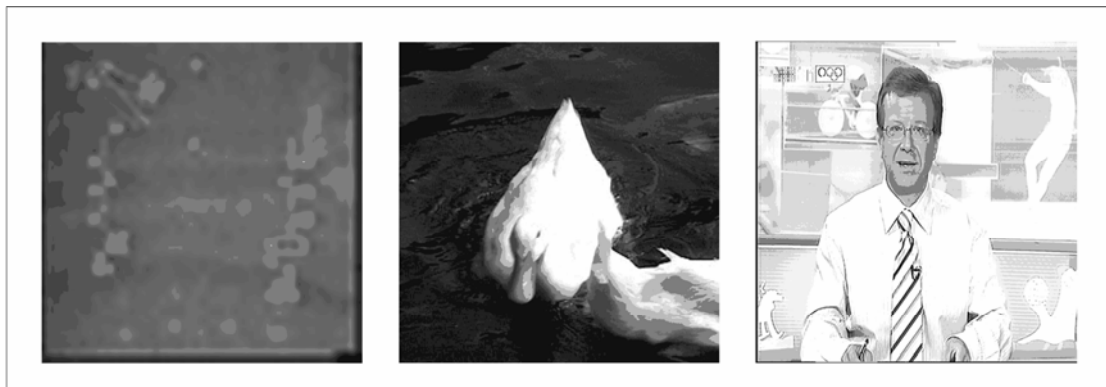
This chapter describes the performance of the systems that were implemented in the new Stretch technology. In addition to this, in this chapter of the thesis there is a comparison between the new implementations of compression/encryption/data hiding and smoothing/edge detection/colour segmentation systems and the hardware or software implementations.

As mentioned in the previous sections, the SCAN algorithm for compression and encryption was implemented in separate, reconfigurable-only designs [19, 20, 26, 28]. These implementations are compared towards to our software-reconfigurable design. On the other hand, there was no hardware implementation for the smoothing, edge detection and colour segmentation system and therefore a new software implementation was designed using the toolbox of Matlab. Despite the software implementation the comparison in processors' cycles is a logical meter to reveal the difference in throughput between our architecture with Stretch technology and our implementation with Matlab.

#### 7.1 Performance of the Compression/Encryption/Data hiding system and comparison with reconfigurable designs

The systems were tested with actual video sequences. The Compression/Encryption/Data hiding system was verified with the same videos that were used in the previous implementations. The videos that were used are 'Dogfight', 'Ducks' and 'News'. In **Figure 7.1** one frame of each video is presented. The first one, 'Dogfight', shows two F-16's engaging over the Aegean Sea. The resolution of the frames for this video sequence is 64x64. The second one, 'Ducks', shows two ducks in the lake. The resolution of the frames for this video is 128x128. This video leads to lower compression than the other ones due to the water ripples on the lake which destroy spatial locality of information. The final video shows a man in an unvarying background and its resolution is 512x512.

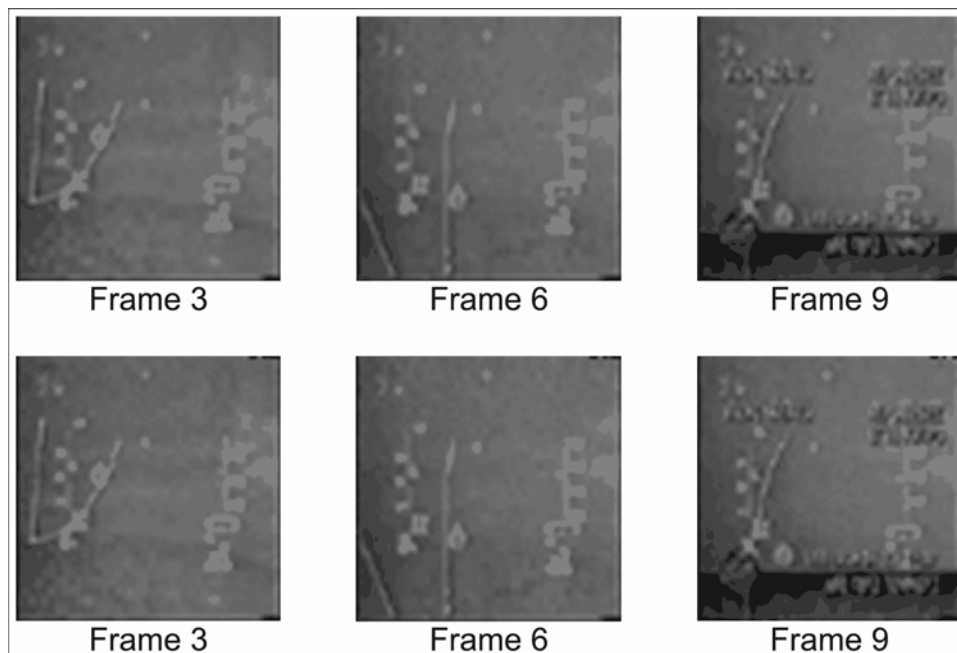
First, the above sequence of video was put as input to the compression/encryption/data hiding system. The data that came out from that system was put as input to the decompression/decryption/data unhiding system. **Figure 7.2** presents three of the original frames of the first video that were used to test our implementation and the three corresponding frames that were created by the process of the decompression/decryption and data unhiding. As one can see, the quality of the final frames is almost the same as the original frames. Some differences, which can be observed between the initial frames and the processed ones, are caused by the embedding process of the data hiding subsystem. More specifically, in that subsystem the process of bit embedding leads to change the values of some pixels and as a result when the reverse process takes place the bits of pixels that were lost because of the embedding process can not be recovered. In general, the SCAN methodology can be lossless, however, by definition, the process of hiding information within an image or video frame alters this frame. In order to make our results useful regardless of video resolution, most results are expressed in terms of throughput.



**Figure 7.1: Three frames from the videos with which we tested the system**

**Table 7.1** and **Table 7.2** show the throughput of these two systems that were designed and implemented. The conclusions that can be extracted by the above tables are that the throughput of each subsystem is not tightly connected to the resolution of the frames that are processed. In particular, there is a tight association between the movement that takes place in the frames of the video with the throughput of the compression, encryption, and data hiding subsystems (as expected, because it destroys temporal locality).

The intense movement in the video sequence provokes great changes to the values of the corresponding pixels between the frames, resulting in an



**Figure 7.2: Three frames from the video 'Dogfight' (in the first line appear the original frames of the video and in the second line the processed frames)**

increase of the number of pixels that should be processed (compressed and rearranged), which leads to increase the time of execution and to reduce the throughput of the system. E.g. the video 'Ducks' has lower resolution than the video 'News', however, the intense movement of the water in the video 'Ducks' results in both subsystems having higher throughput for the video 'News' vs. the video 'Ducks'. The throughput of the encryption depends on the number of iterations for each data stream. **Table 7.3** and **Table 7.4** show the relation between the throughputs of the systems with the number of the iterations. On the contrary, in the case of the video it has been proven that there is a uniform distribution for every

## Microprocessor Hardware Laboratory

Frame Resolution	Compression Subsystem		Encryption Subsystem		Data Hiding Subsystem	Compression Encryption Data Hiding System
	Throughput w/ Virtex II Technology (MB/sec)	Throughput w/ Stretch Technology (MB/sec)	Throughput w/ Virtex II Technology (MB/sec)	Throughput w/ Stretch Technology (MB/sec)	Throughput w/ Stretch Technology (MB/sec)	Throughput w/ Stretch Technology (MB/sec)
64 x 64 (Dogfight)	111	8.77	2.68	2.32	0.36	0.30
128 x 128 (Ducks)	111	7.10	2.68	1.74	0.27	0.22
256 x 256 (Ducks)	111	6.05	2.68	0.85	0.69	0.52
512 x 512 (News)	111	9.67	2.68	2.45	1.78	1.44

**Table 7.1: Throughput for various frame resolutions for the Compression/Encryption/Data hiding system  
(Key1 = c0, Key2 = c0 and Number of iterations = 1)**

Frame Resolution	Decompression Subsystem (MB/sec)	Decryption Subsystem (MB/sec)	Data Unhiding Subsystem (MB/sec)	Decompression Decryption Data Unhiding System (MB/sec)
64 x 64 (Dogfight)	9.34	1.48	1.71	0.73
128 x 128 (Ducks)	8.03	1.10	1.18	0.56
256 x 256 (Ducks)	7.15	0.76	0.95	0.45
512 x 512 (News)	10.24	1.25	1.52	0.75

**Table 7.2: Throughput with Stretch Technology for various frame resolutions for the Decompression/Decryption/Data Unhiding system  
(Key1 = c0, Key2 = c0 and Number of iterations = 1)**

byte value. This attribute can be exploited by reducing the number of iterations for the encryption of compressed video in order to achieve higher throughput without any compromise in the security of the encryption.

The actual number of cycles to encrypt the images is mainly dependent on the encryption key (key1 and key2). If the key is simple the number of cycles is small. In case there are a lot of iterative scan patterns, the total number of cycles increases. **Table 7.5** and **Table 7.6** contain the corresponding throughput using different keys for our systems and for their subsystems.

Number of Iterations	Compression Subsystem		Encryption Subsystem		Data Hiding Subsystem	Compression Encryption Data Hiding System
	Throughput w/ Virtex II Technology (MB/sec)	Throughput w/ Stretch Technology (MB/sec)	Throughput w/ Virtex II Technology (MB/sec)	Throughput w/ Stretch Technology (MB/sec)	Throughput w/ Stretch Technology (MB/sec)	Throughput w/ Stretch Technology (MB/sec)
<b>M = 1</b>	<b>111</b>	<b>8.77</b>	<b>5.53</b>	<b>2.32</b>	<b>0.36</b>	<b>0.30</b>
<b>M = 3</b>	<b>111</b>	<b>8.77</b>	<b>1.97</b>	<b>0.81</b>	<b>0.22</b>	<b>0.17</b>
<b>M = 5</b>	<b>111</b>	<b>8.77</b>	<b>1.82</b>	<b>0.41</b>	<b>0.16</b>	<b>0.12</b>

**Table 7.3: Throughput for the Compression, Encryption, Data Hiding system towards to the number of iterations (Video resolution 64x64)**

Number of Iterations	Decompression Subsystem (MB/sec)	Decryption Subsystem (MB/sec)	Data Unhiding Subsystem (MB/sec)	Decompression Decryption Data Unhiding System (MB/sec)
<b>M = 1</b>	<b>9.34</b>	<b>1.48</b>	<b>1.71</b>	<b>0.73</b>
<b>M = 3</b>	<b>9.34</b>	<b>0.52</b>	<b>0.59</b>	<b>0.27</b>
<b>M = 5</b>	<b>9.34</b>	<b>0.31</b>	<b>0.36</b>	<b>0.17</b>

**Table 7.4: Throughput for the Decompression, Decryption, Data Unhiding system towards to the number of iterations (Video resolution 64x64)**

**Table 7.7** shows the utilisation of the reconfigurable part of S5000 processor. The results show that each unit of the system is independent and that because of the existence of two ISEF units the total number of Arithmetic units is 8192 and Multiply units is 16384. The total utilization for the implemented architecture is about 65 % for the Arithmetic units and 48% for the Multiply units of the reconfigurable part.

Finally, for the case of the information hiding subsystem, there was no previous implementation. The results, shown in **Table 7.1** and **Table 7.2**, reveal that an important factor for the throughput of the embedding subsystem is the resolution of the main video frame.

## Microprocessor Hardware Laboratory

Key1	Key2	Compression Subsystem		Encryption Subsystem		Data Hiding Subsystem	Compression Encryption Data Hiding System
		Virtex II Technology (MB/sec)	Stretch Technology (MB/sec)	Virtex II Technology (MB/sec)	Stretch Technology (MB/sec)	Stretch Technology (MB/sec)	Stretch Technology (MB/sec)
C0	C0	111	8.77	2.68	0.49	0.36	0.12
B0(B0(c0 c0 c0 c0 c0 c0))	z0	111	8.77	2.24	0.42	0.36	0.11
C0	B0(z0 z0 c0 c0)	111	8.77	2.68	0.47	0.36	0.12
C0	Z0	111	8.77	2.24	0.45	0.36	0.12
Z0	B0(z0 z0 c0 c0)	111	8.77	2.43	0.43	0.36	0.12
Z0	z0	111	8.77	1.82	0.41	0.36	0.11

**Table 7.5: Throughput for the Compression/Encryption/Data Hiding system using different keys (Number of iterations = 1 and video resolution 64x64)**

Key1	Key2	Decompression Subsystem (MB/sec)	Decryption Subsystem (MB/sec)	Data UnHiding Subsystem (MB/sec)	Decompression Decryption Data Unhiding System (MB/sec)
c0	C0	9.34	0.31	1.71	0.17
B0(B0(c0 c0 c0 c0 c0 c0))	Z0	9.34	0.29	1.71	0.11
c0	B0(z0 z0 c0 c0)	9.34	0.29	1.71	0.16
c0	z0	9.34	0.29	1.71	0.16
z0	B0(z0 z0 c0 c0)	9.34	0.28	1.71	0.15
z0	z0	9.34	0.28	1.71	0.15

**Table 7.6: Throughput for the Decompression/Decryption/Data Unhiding system using different keys (Number of iterations = 1 and video resolution 64x64)**

## Microprocessor Hardware Laboratory

Concluding, the results from the above tables show that even though the implementation with the Stretch Technology gives lower throughput for each subsystem than the previous implementations with only reconfigurable logic, the system-level throughput can be adequate for real applications. Apart from that, this thesis' architecture we achieved to embed the entire SCAN Algorithm in a low-cost chip, **Table 7.8**, like that of S5000 processors of Stretch Technology.

The cost of the chips is shown in **Table 7.8**. Whereas, in previous studies of ours, each of two major components required a Xilinx Virtex II, the integrated design.

Reconfigurable unit	Number of used Arithmetic Units (Total AUs: 8192)	Percentage of utilization of AUs	Number of used Multiply Units (Total MUs: 16384)	Percentage of utilization of MUs
Compare unit	929	11.4 %	0	0 %
Encoding unit	1941	23.7 %	1280	7.6 %
Transformation unit	1126	13.7 %	0	0 %
Read Pixels unit	161	2 %	0	0
Substitution unit	858	10.5 %	6528	39.8 %
Complexity unit	330	4 %	0	0 %

Table 7.7: The percentage of utilization of reconfigurable part of the S5000 Processor

Name of chip	Cost ( \$ )
FPGA Virtex II PRO Xilinx Inc.	300
S5000 processor Stretch Inc.	<100

Table 7.8: Cost of the chips that we used for our implementations

fits in a single Stretch S5000 integrated circuit. From a cost-performance point of view, the numbers can vary, depending on what we consider. E.g. a Virtex II for compression only delivers roughly ten times the performance of the

S5000 at three times the cost, however, given that there is a bottleneck in the next step of the process the compression alone is not a good metric. Taking into account that information hiding was never implemented in Virtex II technology it is sufficient to say that the contribution of this work is a different solution in the design space rather than an unequivocal preferred approach.

### **7.2 Performance of the Smoothing, Edge detection and Colour segmentation system and comparison with software implementation**

In this section there is a description of the performance of the embedded system that was designed as well as a description of the components, which are the smoothing, the edge detection and the colour segmentation algorithms of a colour image. In addition, there is a comparison of the throughput between the architecture of the system that was designed, the Stretch technology and an implementation of the algorithm with Matlab toolbox.

As mentioned above, there is no hardware implementation of these three algorithms and therefore a new implementation of the algorithms was designed using the toolbox of Matlab. First, the toolbox of Matlab was chosen for the comparison with the hardware-software implementation as this tool is used widely for image applications. Second, Matlab offers quick methods of processing groups of numbers, pixels in our case, avoiding loops which in software are time consuming. Third, although the toolbox of Matlab is implemented with Java, which is a time consuming programming language, in our comparisons, only the time that the processor (Pentium D 2.8 GHz ) processes the data and not the total execution time is taken into account.



**Figure 7.3: The initial colour image**

The system was tested with a medium resolution colour image (280 x 280), which is shown in **Figure 7.3**. This image was chosen because there are objects of the same colour touching each other and as a result it is one of the most difficult cases for the algorithm to separate the different objects in image. Each step of the algorithm (smoothing, edge detection and colour segmentation) resulted to images which are presented in this section. The results from Matlab implementation have a small number of differences with those from the hardware architecture due to the nature of the algorithms. Some of the implemented algorithms compute floating point values which can not easily be represented in hardware and as a result there was loss of definitude in the final results of Stretch implementation.

First, the image of **Figure 7.3** was put as input both to the software and to reconfigurable-software systems. These systems are separated into three independent parts, smoothing, edge detection and colour segmentation part. The results from each one of these parts are inputs to the next parts of the system. The intercalary results of the independent parts are presented in **Figures 7.4, 7.5 and 7.6**. In these figures there are also the results of the Matlab implementation, where the small differences, as explained above, are observable.

**Table 7.8** shows the comparison of the CPU time and the throughput for each subsystem independently and for the whole system, between the implementation with Matlab toolbox and the reconfigurable-software Stretch

## Microprocessor Hardware Laboratory

architecture. First, the results show that the most compute intensive part of the three algorithms is the colour segmentation. Second, it is clear that the architecture of the system implemented with reconfigurable-software technology gives better throughput and lower execution CPU time for all the subsystems and for the total system than a toolbox which is widely used for these applications, as Matlab toolbox. Third, the heaviest computationally algorithm of this system is the colour segmentation and it can be explained by the fact that in that subsystem the image is scanned many times and it is processed repeatedly until all the pixels to belong to an object.

System	MATLAB Toolbox		Stretch Technology		Comparison Implementation w/ Stretch Vs. Implementation w/ Matlab
	CPU time (sec)	Throughput (KB/sec)	CPU time (sec)	Throughput (KB/sec)	
Smoothing Subsystem	40.84	5.62	1.61	142.66	25x
Edge Detection Subsystem	47.63	4.82	11.68	19.67	4x
Color Segmentation Subsystem	109.23	2.10	19.82	11.58	5.5x
Total Architecture	197.7	1.16	33.11	6.94	6x

**Table 7.8: The CPU time (sec) and the throughput (MB/sec) for each one of the subsystems for the two different implementations (Stretch vs. Matlab)**

Reconfigurable unit	Number of used Arithmetic Units (Total AUs: 8192)	Percentage of utilization of AUs	Number of used Multiply Units (Total MUs: 16384)	Percentage of utilization of MUs
Smoothing unit	6019	73.5 %	15168	92.6 %
Edge Detection unit	5222	63.4 %	3264	19.9 %
Colour Segmentation unit	2227	27.2 %	10944	66.8 %

**Table 7.9: The percentage of utilization of reconfigurable part of the S5000 processor for Smoothing, Edge detection and Colour segmentation of a colour image**



Figure 7.4a: Smoothed colour image using Matlab



Figure 7.5b: Smoothed colour image using Stretch technology

**Table 7.9** shows the results of usage for the reconfigurable part of S5000 processor. Remarkable is the fact that the utilisation of FPGA in the S5000 processor is more than 100%, which reveals the fact that there is a swap in configurations of the reconfigurable parts. It is important to mention that this swap time is included in CPU time as described in the above tables. Finally, another important observation is the high utilisation and use of the reconfigurable part which leads to the reduction of execution time and as a result to the increase of the throughput for the system.

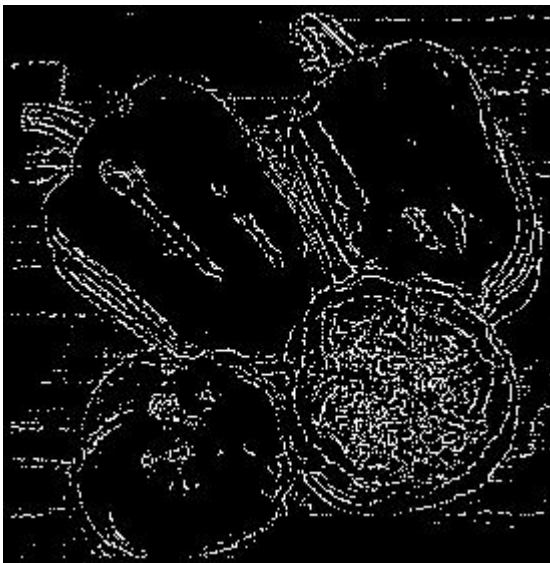


Figure 7.5a: Edge detected image using Matlab

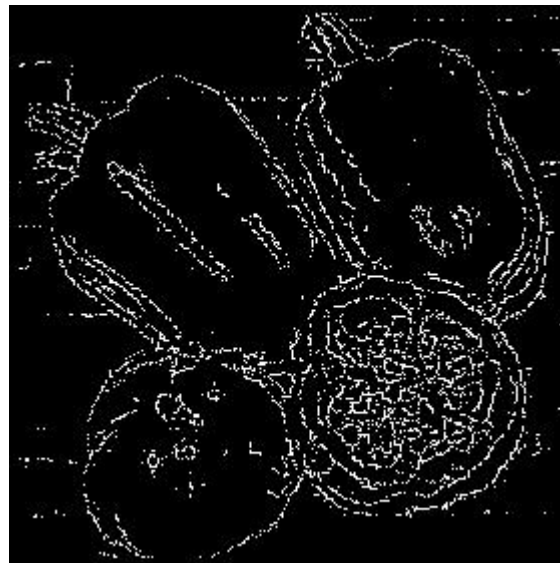


Figure 7.5b: Edge detected image using Stretch technology



**Figure 7.6a: Edge detected image using Matlab**



**Figure 7.6b: Colour segmented image using Stretch technology**

Concluding, the results from the above tables show that the implementation with the Stretch Technology gives higher throughput for the total system than the software implementation with Matlab toolbox. The remarkable issue in this implementation is the implementation of the three algorithms with a cheap S5000 processor and as a result the performance of the system can be used in real time systems.

## Chapter 8

### Conclusions and future work

This chapter describes some points of this project that could be done as future work in a possible extension of this work. Finally, in this section there is a more profound comparison between the three different architectural routes that were used or compared in this work.

#### 8.1 Conclusions

The main contribution of this work was the complete implementation of a compress/encrypt/information hide system and smoothing, edge detection and colour segmentation algorithms in a low cost core. The mapping of the entire system on the Stretch technology is quite a departure from our previous designs in terms of cost-performance, as well as, the level of complexity that designs can have. The C-based design flow proved to be easy to master and quite effective, although the lack of ability to intervene in the low-level FPGA design was at times constraining. The system-level performance met our expectations and the ability to put the entire design on the chip, was somewhat of a surprise, given the hardware complexity of the previous designs, with hand-optimized VHDL code no less.

Another important contribution of this work is the comparison of three different implementations of the same algorithms. This makes clear the differences between the Stretch solution, the FPGA solution and the Matlab solution. In addition to this, there is a comparison between the reconfigurable-only designs, the software-reconfigurable designs and the software-only designs. The differences as well as the comparison among the different technologies are shown in **Table 8.1**

As described in the table below, each of these three technologies has both advantages and disadvantages. The new technology of Stretch offers

## Microprocessor Hardware Laboratory

<b>Stretch Technology</b>	<b>FPGA Technology</b>	<b>Matlab Toolbox</b>
<b>Design with C/C++ language</b>	<b>Typical design with VHDL/Verilog language (higher flexibility at higher design time)</b>	<b>Design with Matlab language</b>
<b>Low cost ( about \$100)</b>	<b>Medium – high cost ( <math>\geq</math> \$300)</b>	<b>Expensive tool</b>
<b>No explicit control of Reconfigurable part of S5000 processors</b>	<b>Control of low level parts</b>	<b>No reconfigurable part, only software</b>
<b>Easy and quick debugging</b>	<b>Time consuming and difficult debugging</b>	<b>Easy and quick debugging</b>
<b>Easy development and quick time to market</b>	<b>Not very easy development and time to market</b>	<b>Easy development</b>
<b>Complete System Visibility</b>	<b>Limited System Visibility</b>	<b>Limited System Visibility</b>
<b>Good performance of designs</b>	<b>Excellent performance of designs</b>	<b>Good performance of designs</b>

**Table 8.1: A table-comparison between three different technologies**

easy implementation with C/C++ language, very easy and quick debugging as the designer can run his code line-by-line and very quickly and as a result the final implementation can be developed in a very small period of time. In addition to these, through Stretch technology the designer can have a complete system visibility, low cost of designs and generally a good performance of his designs. On the other hand, the FPGA technology offers the absolute control of the design, even in low levels like memories, FIFOs etc., and better performance of every other software and DSP solution. Concluding, the Matlab toolbox offers easy development and quick implementation. However, it is an expensive tool and it does not have such good performance as hardware implementations.

### 8.2 Future work

This project is almost a complete work whereas there are some points that can be extended in order to achieve better results and more comparisons. Below there are some ideas that can be proposed to extend the subject of this work:

- In the previous chapters for the SCAN compression/encryption/data hiding system there is a comparison between the architecture implemented with S5000 Stretch processors towards the previous reconfigurable implementations. A new extension on this part of the project could be the implementation of the above algorithm with a typical microprocessor or DSP and the performance comparison with the previous architectures.
- The Automatic Recognition Algorithm consists of many steps. The first three of them are the smoothing, edge detection and colour segmentation algorithm. The implementation of the rest steps of the algorithm and embedding the whole system in a S5000 processor would provide a complete chip which could be used as a black box in a bigger system.
- The ATR algorithms were implemented in software using Matlab because, as explained in the previous sections, Matlab is a toolbox which is widely used in image processing. A new implementation of the same algorithms using another programming language, as C/C++, could give us not only a comparison between these two different software implementations but also would increase the number of comparisons between the implementations.
- In the future, we need to further optimize the design of SCAN algorithm and increase the number of SCAN keys that are supported in reconfigurable-software design.

## References

### Internet

- [1] The official Stretch Inc site, <http://www.stretchinc.com/>
- [2] The official Xilinx Company site, <http://www.xilinx.com/>
- [3] The official Mathworks Company site, [www.mathworks.com](http://www.mathworks.com)

### Bibliography

- [1] Cheng Chen, Video Compression: Standards and Applications, Journal of Visual Communication and Image Representation, Vol. 4, No. 2, June, pp.103-111, 1993.
- [2] N. Ahmed, T Natarajan, K. R. Rao, Discrete cosine transform, IEEE trans Computers, 23, pp. 90-93, Jan 1974
- [3] A. H. Sadka, Compressed Video Communications, Wiley, pp. 14-72, 2002
- [4] P.H. Westerink, J. Biemond, and G. Muller, Subband coding of image sequences at low bit rates, Signal Process Image Commum 2, pp. 441-448, 1990.
- [5] J. Max, Quantizing for minimum distortion, IEEE trans. on Information Theory, 6, No. 1, pp. 7-12, Mar. 1960
- [6] S. Maniccam and N. Bourbakis, On compression and encryption for digital video, Int. ISCA Conf. on Parallel and Distributed Systems, pp. 652-657, Aug. 2000, NV, USA
- [7] N. Bourbakis, C. Alexopoulos, Picture data encryption using SCAN patterns, Pattern Recognition Journal, vol. 25, no6, 1992, pp.576-581
- [8] S. Maniccam, A Lossless compression, encryption and information hiding methodology for images and video, PhD thesis, Dept. ECE, Binghamton University, 2000
- [9] S. Maniccam, N. Bourbakis, Image and Video Encryption using SCAN patterns, Pattern Recognition Journal, Vol. 37, 2004, pp. 725-737.
- [10] S. S. Maniccam, N. Bourbakis, Lossless compression and information hiding in images, Pattern Recognition Journal , Vol. 37, 2004, pp. 475-486
- [11] S.Grossberg, H.Hawkins and A.Waxman "A special issue of ATR, Neural Network Journal, vol.8,7-8,1995
- [12] M.Brown and W.Swonger, "A prospectus for ATR", IEEE-T-AES, 25,3,1989
- [13] J.Novak and N.Bourbakis, "A survey on ATR detection and recognition", TR-1993, MV Lab BU-CIS, 21 pages.
- [14] F.Sadjadi, F.Garber, et.al., "Automatic Target Recognition, SPIE 1993
- [15] L-C.Wang, et.al. "ATR using a feature decomposition and data decomposition modular NN", IEEE T-IP, 7,8,1998,1097-1112
- [16] S.S.Young, et. al., "Foveal ATR using a multiresolution NN", IEEE T-IP, 7,8,1998, 1113-1121

- [17] J.Principe, et. al, "Target discrimination in synthetic aperture radar using ANN", IEEE T-IP, 7,8,1998, 1122-1135
- [18] G. Chrysos, A. Dollas, N.Bourbakis, S. Mertoguno, An Integrated Video Compression, Encryption and Information Hiding Architecture based on the SCAN Algorithm and the STRETCH Technology, Proceedings of the IEEE International Symposium on Field-Programmable Custom Computing Machines FCCM 2007 (Poster paper)
- [19] A. Dollas, C. Kahris, N. Bourbakis, Performance Analysis of Fixed, Reconfigurable, and Custom Architectures for the SCAN Image and Video Encryption Algorithm, Proceedings of the FCCM 2003, pp. 19-28.
- [20] C. Kahris, N. Bourbakis, A. Dollas, A Reconfigurable Logic-Based Processor for the SCAN Image and Video Encryption Algorithm, International Journal of Parallel Programming, Vol. 31, No. 6, December 2003.
- [21] N. Bourbakis, A. Dollas, SCAN-Based Compression – Encryption – Hiding for video on Demand, IEEE Multimedia Magazine, July – September 2003, pp. 79-87.
- [22] C. Kachris, S. Maniccam, A. Dollas, N. Bourbakis, A Reconfigurable Logic-Based Processor for the SCAN Image and Video Encryption Algorithm, International Workshop on Application-Specific Processors, WASP 2002, Istanbul, Turkey.
- [23] H. Sofikitis, K. Roumpou, A. Dollas, N. Bourbakis, An Architecture for Video Compression Based on the SCAN Algorithm, Proceedings of the IEEE International Symposium on Field-Programmable Custom Computing Machines FCCM 2005, pp. 295-296.
- [24] J. Kaps and C. Paar, *Fast DES Implementations for FPGAs and its Application to a Universal Key-Search Machine*, in 5th Annual Workshop on Selected Areas in Cryptography (SAC '98) (S. Tavares and H. Meijer, eds.), vol. LNCS 1556, (Queen's University, Kingston, Ontario, Canada), Springer-Verlag, August 1998.
- [25] Y. K .Lee, L. H. Chen, An Adaptive Image Steganographic Model Based on Minimum-Error LSB Replacement
- [26] X. Li, Image compression and encryption using tree structures, *Pattern Recognition Letters*, vol. 18, no11, 1997, pp 1253-1259
- [27] H. Cheng, X. Li, Partial Encryption of Compressed Images and Videos, Presently at Department of Computer Science, University of Waterloo
- [28] Fu KS, Mu JK. A survey on image segmentation. *Pattern Recognition* 1981; 13:3–16
- [29] Schettini R. Low-level segmentation of complex colour images. *Signal processing VI: Theories and Applications* 1992: 535–538
- [30] Celenk M. A colour clustering technique for image segmentation. *Computer Vision, Graphics, and Image Processing* 1990; 52:145–170
- [31] Bajcsy R, Lee SW, Leonardis A. Color image segmentation and color constancy. *SPIE, Perceiving, Measuring and Using Color* 1990; 1250
- [32] Huang C, Cheng T, Chen C. Color images' segmentation using scale space filter and Markov Random Field. *Pattern Recognition* 1992; 25(10): 1217–1229

## Microprocessor Hardware Laboratory

- [33] Lee SU, Chung SY. A comparative performance study of several global thresholding techniques for segmentation. *Computer Vision, Graphics & Image Processing* 1990; 52:171–190
- [34] Perez F, Koch C. Toward color image segmentation in analog VLSI: algorithm and hardware. *Int J Computer Vision* 1994; 12(1):17–42
- [35] Bourbakis NG. Calculating the pixels flow functions based on illumination sources that produce highlights on color images. *GMU-MVRL, TR-1987*
- [36] Moghaddamzadeh A, Bourbakis NG. A fuzzy region growing approach for segmentation of color images. *J Pattern Recognition* 1997; 30(6)
- [37] Klinker GJ, Shafer S, Kanade T. Image segmentation and reflection analysis through color. *Image Understanding Workshop, San Matteo, CA, 1988: 838–853*
- [38] Klinker GJ. *A Physical Approach to Color Image Understanding*. A. K. Peters, Wellesley, MA, 1992
- [39] Pal NR, Pal SK. A review on image segmentation techniques. *Pattern Recognition* 1993; 26(9):1277–1294
- [40] Xiaohan Y, Yla-jaaski J. Image segmentation combining region growing and edge detection. *11th International Conference on Pattern Recognition, The Netherlands, 1992; III*
- [41] Moghaddamzadeh A, Bourbakis NG. A fuzzy approach for smoothing and edge detection in color images. *IS&T/SPIE's Symposium on Electronic Imaging: Science & Technology, San Jose, CA., February 5–10 1995*
- [42] Pienkowski AEK. *Artificial color perception using fuzzy techniques in digital image processing*. TUV Rheinland, Germany, 1989
- [43] Moghaddamzadeh A, Goldman D, Bourbakis N. A fuzzy-like approach for smoothing and edge detection in color images. *JPattern Recognition and AI* 1998; 12(6)
- [44] Goldman D, Bourbakis N. A real-time tool for user's driven adaptive segmentation of sequences of color images. *AFRL-TR-1997, Proc IEEE Conf on TAI-99, Evanston, IL, 1999; 131–139*
- [45] Huntsberger T, Descalzi M. Color edge detection. *Pattern Recognition Letters* 1985; 3:205–209
- [46] Lim Y, Lee S. On the color image segmentation algorithm based on the thresholding and fuzzy c-means techniques. *Pattern Recognition Journal* 1990; 23(9):935–952
- [47] Lambert P, Carron T. Symbolic fusion of luminance-hue-chroma features for region segmentation. *Pattern Recognition Journal* 1999; 32:1857–1872
- [48] Weeks AR, Hague GE. Color segmentation in the HSI color space using the k-means algorithm. *Proceedings of the SPIE – The International Society for Optical Engineering* 1995; 3026:143–54
- [49] Etienne-Cummings R, Pouliquen P., Lewis M. A., A vision Chip for Color segmentation and Pattern Matching, *EURASIP Journal on Applied Signal Processing* 2003;7, pp. 703-712

## **Microprocessor Hardware Laboratory**

- [50] C. Alexopoulos, SCAN: An efficient data processing-accessing formal methodology, PhD thesis, Dept. of Computer Engineering and Informatics, University of Patras, 1989.
- [51] X. Yuan, D. Goldman, A. Moghaddamzadeh, N. Bourbakis, Segmentation of Colour Images with Highlights and Shadows using Fuzzy-like Reasoning, Pattern Analysis & Applications, pp. 272-282, 2001
- [52] W. E. Higgins and C. Hsu, Edge detection using two-dimensional local structure information, Pattern Recognition 27,2 (1994) 277-294
- [53] S. Ghosal and R. Mehrota, Detection of composite edges, IEEE Trans. Imag. Process. 3, 1 (1994) 14-25.