Hardware Accelerators for Automated Digital Surveillance Systems

Hugo Hedberg

Thesis for the degree of
Licentiate in Engineering

2006
Department of Electroscience
Lund University, Sweden

Department of Electroscience
Lund University
Box 118, S-221 00 LUND
SWEDEN

This thesis is set in Computer Modern 11pt,
with the LaTeX Documentation System

April 2006

Abstract

Conventional surveillance systems are omnipresent and most are still based on analog techniques. Migrating to the digital domain grants access to the world of digital image processing enabling automation of such systems. In this thesis, automation of a surveillance system means extracting information from the image stream without human interaction. In the intended surveillance applications, the target property of the objects of interest is motion. Therefore, motion detection, i.e. segmentation, is applied on the image stream distinguishing the objects in the scene. After segmentation, additional image processing is applied, e.g. filtering, labeling, feature extraction, tracking, etc. The goal is to be able to keep track of the objects in consecutive frames. Accelerating key operations and complex calculations in hardware is an effort to address the high data rate from the sensor together with the high memory bandwidths imposed by the image processing. Furthermore, decoupling the general purpose processor from the image stream by using features can effectively reduce the amount of data needed to be processed in SW, which is crucial to sustain real–time performance. This thesis highlights and is an effort to address some issues encountered during the automation process.

In this thesis, implementation details of two hardware accelerators are presented: a low complexity morphology unit and a labeling unit based on contour tracing. The morphology architecture performs erosion or dilation on binary images and takes advantage of decomposition of rectangular structuring element supporting arbitrary sizes up to 15×15. Due to its low complexity and memory requirement, multiple instances can be connected in series enabling other fundamental morphological operations, e.g. opening and closing. In our application, the unit is used to filter noise (opening) and to reconnect split objects in the motion mask generated by prior steps in the processing chain. Furthermore, implementation details of a labeling unit with low memory requirements based on contour tracing is presented. Both accelerators have been successfully verified and integrated into a prototype of an automated digital surveillance system for which implementation aspects are also presented. The system has been implemented and verified on an FPGA development board using a CMOS sensor for image acquisition. The prototype currently has segmentation, filtering, and labeling accelerated in hardware. Additional image processing is performed by SW running on an embedded processor. The system has a resolution of 320×240 and a frame rate of 25 fps.

In loving memory of my father

Contents

Preface

This thesis summarizes the authors academic work in the digital ASIC group at the department of Electroscience for the Licentiate degree in Electrical Engineering. The main contribution to the thesis is derived from the following publications:

> F. Kristensen, H. Hedberg, H. Jiang, P. Nilsson, and V. Öwall, "Hardware Aspects of a Real-time Surveillance System," Accepted for publication in *The sixth IEEE workshop on visual surveillance*, Graz, Austria, May 13, 2006.
>
> H. Hedberg, F. Kristensen, P. Nilsson, and V. Öwall, "A Low Complexity Architecture for Binary Image Erosion and Dilation using Structuring Element Decomposition," in *Proc. of IEEE International Symposium on Circuits and Systems*, pp. 3431–3434, Kobe, Japan, May 23-26, 2005.
>
> H. Hedberg, and and V. Öwall, "Implementation of a Connected-Cluster Labeling Algorithm based on Contour Tracing," in *Proc. of the Swedish System-on-Chip Conference, SSoCC'06*, Norrköping, Sweden, May 4-5, 2006.
>
> F. Kristensen, H. Hedberg, and H. Jiang, "An Intelligent Surveillance System," in *Proc. of the Swedish System-on-Chip Conference, SSoCC'05*, Stockholm, Sweden, April 18-19, 2005.

The papers "*Hardware Aspects of a Real-time Surveillance System*" and "*An Intelligent Surveillance System*" are common efforts describing a complete automated digital surveillance system. The first paper focuses on a prototype of such a system and the latter, on the outline and goals of the research project. The author's main contributions in both these publications are mainly the parts addressing morphology and labeling. The following papers are also published, but not considered part of this thesis:

> H. Hedberg, J. N. Rodrigues, F. Kristensen, H. Svensson, M. Kamuf, and Viktor Öwall, "Teaching Digital ASIC Design to Students with Heterogeneous Previous Knowledge," in *Proc. of Microelectronic Systems Education, MSE'05*, pp. 15–16, Anaheim, California, USA, June 12-13, 2005
>
> J. N. Rodrigues, M. Kamuf, H. Hedberg, and Viktor Öwall, "A Manual on ASIC Front to Back end Design Flow," in *Proc. of Microelectronic Systems Education, MSE'05*, pp. 75–76, Anaheim, California, USA, June 12-13, 2005
>
> H. Hedberg, T. Lenart, and H. Svensson, "A Complete MP3 Decoder on a Chip," in *Proc. of Microelectronic Systems Education, MSE'05*, pp. 103–104, Anaheim, California, USA, June 12-13, 2005
>
> H. Hedberg, and P. Nilsson, "A Survey of Various Discrete Transforms used in Digital Image Compression Algorithms," in *Proc. of the Swedish System-on-Chip Conference, SSoCC'04*, Båstad, Sweden, April 13-14, 2004.

H. Hedberg, T. Lenart, H. Svensson, P. Nilsson and V. Öwall, "Teaching Digital HW-Design by Implementing a Complete MP3 Decoder," in *Proc. of Microelectronic Systems Education, MSE'03*, pp. 31–32, Anaheim, California, USA, June 1-2, 2003

Acknowledgment

First of all, I would like to thank my unofficial supervisor, colleague, nowadays tennis partner, but most of all dear friend, Fredrik Kristensen. Without his good advice and helping hand, none of this would probably have been written. Fredrik, I am proud to be on the same team as you.

The second person that I owe a great deal of gratitude to, is my official supervisor associate professor Dr. Viktor Öwall. He has spent hours of his valuable time reading through this thesis, correcting my English, removing extensive occurrences of "together with" and "i.e.". Thank you for all the constructive criticism regarding my work and for keeping your calm even though receiving messages during late nights and weekends.

My gratitude goes to all my colleagues in the whole Electroscience department; you have all contributed to my work in various ways, from taking care of administrative paper work to making it worthwhile going to work every day. I would like to thank every past and present member of the DASIC corridor, especially Thomas Lenart, Henrik Svensson, Mattias Kamuf, Jiang Hongtu, and Joachim Neves Rodrigues. Thank you all for being good friends, your profound knowledge in our field of research, and for taking care of more social related issues, making this a pleasant journey. Last but not least, my newly found friend Ulrike Richter for trying hard to cheer me up on those windless days when I seriously consider spending my future inventing a machine that gives a steady onshore wind of 15 m/s. I could probably go on forever but have to summarize in some way; I will keep you all in my heart until the day I am forced to leave this earth.

I would like to thank the other persons involved in this research project, especially associate professor Dr. Peter Nilsson for helping me during my time here at the department. I would also like to take the opportunity to thank Anders Olsson, and Daniel Elvin from AXIS Communications AB for their valuable input and professional perspective, i.e. setting the pace in this research project.

This work has been financed by the Competence Center for Circuit Design (CCCD) and sponsored by the Xilinx University Program by donating development platforms and software.

Christian Sternell, thank you for not only being a good friend who introduced me to the world of "possessivt pronomen", but also for taking your time to read through these pages (even though I am quite sure you consider it being a punishment for being born).

To my family, Christina and Björn Hedberg, thank you for helping me stay in focus when I sometimes ponder upon whether my dreams have abandoned me.

Finally, I would like to say something to you waiting patiently in the shadows of my life. Still, sometimes you motivate me, but at other times you tie my hands behind my back. For those of you who read this far and are wondering, I am not talking to a voice inside my head. Nevertheless, I guess in some unpredictable and unexplainable way...although you will never read them...these words are for you...

April, Lund, 2006

Hugo Hedberg

List of Acronyms

ASIC	Application-Specific Integrated Circuit
ADC	Analog-to-Digital Converter
B	Byte
BW	Bandwidth
Bps	Bytes per second
CCD	Charge-Coupled Device
CCTV	Closed Circuit Television
COG	Center of Gravity
CMOS	Complementary Metal Oxide Semiconductor
DAC	Digital-to-Analog Converter
DCM	Digital Clock Manager
DDR	Dual Data Rate
FIFO	First In, First Out
FIR	Finite Impulse Response
FPGA	Field Programmable Gate Array
fps	frames per second
FSM	Finite State Machine
HW	Hardware
HDL	Hardware Description Language
IP	Intellectual Property
LAN	Local Area Network
LUT	Lookup Table

PCB	Printed Circuit Board
PPC	PowerPC
P&R	Place and Route
RAM	Random-Access Memory
SE	Structuring Element
SDRAM	Synchronous Dynamic Random Access Memory
SW	Software
VGA	Video Graphics Array
VHDL	Very high-speed integrated circuit Hardware Description Language

List of definitions and mathematical operators

All definitions listed here are taken from [1] and [2].

Z^2	2-D integer space
$\lfloor A \rfloor$	Floor, rounds A to nearest lower integer towards minus infinity
$\lceil A \rceil$	Ceiling, rounds A to the nearest upper integer towards infinity
$\exists x$	There exists an x such that...
$\forall x$	For all x,...
$x \in A$	The elements x belongs to the set A
$x \notin A$	The elements x does not belong to the set A
$\ominus$	Erosion
$\oplus$	Dilation
$\circ$	Opening
$\bullet$	Closing
$\emptyset$	empty set
$A' = A^c$	Complementation or inverse
$\hat{A}$	Reflection, i.e. geometric inverse
$(A)_z$	The translation of A by z
$A \subseteq B$	A is a subset of B
$\cap$	Intersection

Chapter 1

Introduction

Ever since the introduction of television, real–time monitoring has been a growing market. Adding a video recorder opened up a new world for the security industry. Video surveillance soon made its way into the court rooms and became convicting evidence. Today, video surveillance systems are omnipresent and part of everyday life and can be found in department stores, banks, buss terminals, etc. They are not only used in crime preventing purposes but also play their role in more social and industry related applications, e.g. traffic monitoring, processing monitoring, etc. With continuously increasing fields of application and integration into our lives, a bunch of serious questions arises: *Will every move one makes be monitored*? *How will this information be used*? *What if Big Brother is evil*? Naturally, none of these questions will be answered in this thesis but rather highlight the connection between the presented technology and social interests that might be attached to this field of research. The only conclusion that can be drawn is that some of these applications are easier to accept than others, and the threshold of where they are acceptable is of course subjective. Without knowing what the future of automated surveillance will bring and without taking a stand, this statement is left with two questions from both sides of the debate: *How would you feel if someone is monitoring your path through town a dark night*? *How would you feel if somebody is constantly watching you*?

Conventional real–time surveillance systems are known as Closed Circuit Television (CCTV) systems. A typical system supports multiple cameras, event recording, auto-focusing, zooming, etc, and is traditionally controlled by an human operator. Automating such a system would not only reduce the time spent on monitoring the system itself but can also increase the number of attached cameras in the system, thus increasing surveillance efficiency. Using digital image processing enables automation of such systems which means extracting information from the image stream without human interaction. In our surveillance application, the objects of interest share at least one common property, i.e. motion. Revealing the motion in a scene is achieved by applying a motion detection algorithm on the image stream which transforms the original image stream into a motion mask. This process is called segmentation and the reliability of the automation is somewhat dependent on the result from this step since if an object is not included in the motion mask, it will not be further processed by the system. After segmentation, various image processing steps are applied to the image stream

and motion mask, e.g. filtering, labeling, feature extraction, tracking, etc. The goal of the automation is to be able to track the objects in consecutive frames. Taking the system intelligence a step further, i.e. adding a classification unit, the system should be able to detect what is being tracked, e.g. humans, cars, aeroplanes, etc. Including such a unit, the system can start to store data if there are objects of interest present in the scene. A step further would be a system that can identify anomalies or deviations in the motion pattern within a scene, e.g. running, fighting, etc. Finally, one of the most interesting but at the same time controversial additions in system features is face recognition. Possibly, a future intelligent surveillance system might be able to search for and track a person in a camera network recording the persons trajectory within the network. However, before being able to perform any kind of tracking or classification, the automation process will encounter many difficulties that needs to be addressed. The amount and nature of these issues are dependent on the application environment, e.g. varying lighting conditions, object occlusion, static objects, periodic motion (e.g. swaying trees), etc. However, they will in any case certainly require extensive calculation that poses special demands on the image processing, especially on the segmentation process. Many of these issues are related to the lighting conditions in the scene since it affects the dynamic range of the image stream. A low dynamic range in the image stream results in increased noise in the motion mask, i.e. falsely detected objects. Therefore, a distinction between indoor and outdoor applications is usually made, since indoor environments typically has a more static illumination than outdoor due to the weather, time of the day, etc. Based on these facts, the primary objective of the system presented in this thesis is to detect humans in an indoor office environment.

Any application environment, both indoors and outdoors, imposes extensive image processing, resulting in a high bandwidth not possible to handle only in software on an embedded platform to achieve real–time performance, i.e. $\geq$25 frames per second. Bandwidth (BW), measured in bytes per second (Bps), is an important issue in any image system but especially in surveillance applications due to the constrained execution time. As an example, a typical uncompressed image in the RGB color space consists of three color channels per pixel, $c_{channels}$, each represented with 8 bits, c_b, and a resolution of 640×480 (Video Graphics Array (VGA)). A system with a frame rate, f_{rate}, of 25 frames per second, has a bandwidth equal to

$$BW = c_{channels} \cdot c_b \cdot (im_{width} \cdot im_{height}) \cdot f_{rate} = 3 \cdot 8 \cdot (640 \cdot 480) \cdot 25 \approx 23 \text{ MBps}.$$

Therefore, a design challenge in any automated surveillance system is to handle or reduce the bandwidth. This can be done on different abstraction levels and various techniques can be applied. In this thesis, to be able to address the high bandwidth, the system is implemented as an embedded system, i.e. a system with one purpose. The main idea is to have key operations and complex calculations accelerated in hardware which results in that the amount of data needed to be processed in software is reduced, which is crucial to sustain real–time performance. The remainder of this thesis describes

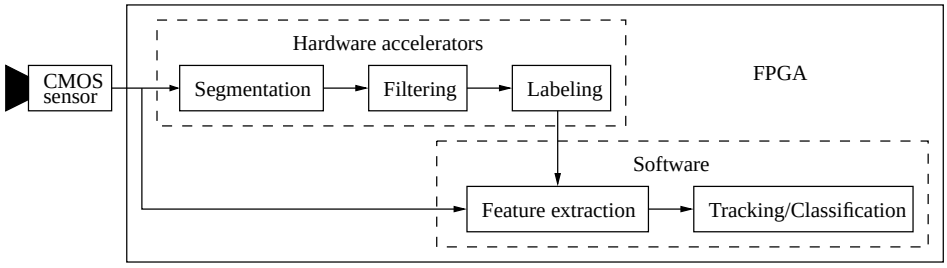


Figure 1.1: A conceptual overview of the automated digital surveillance system addressed in this thesis. The system includes key operations accelerated in HW and feature extraction, tracking and classification performed in SW.

how this can be archived in detail starting with a system overview.

1.1 System overview

A conceptual overview of the embedded automated digital surveillance system addressed in this thesis is depicted in Figure 1.1. The system is partitioned into hardware (HW) and software (SW), having key operations accelerated in HW and feature extraction, tracking and classification performed in SW running on a general purpose processor.

Starting at the input, a sensor captures an image stream that constitutes the input to the system. After image acquisition, the frame is sent to a segmentation step, which is used to separate the objects of interest from the rest of the image. The next step is to apply post segmentation processing in order to reduce noise generated in the image acquisition step and by the segmentation step, e.g. using a morphology unit. Before sending the filtered binary image to the feature extraction unit, each object is assigned a unique label. This procedure enables the system to distinguish between detected objects. Furthermore, combining the labeled result and the original video sequence, the SW now only needs to process certain parts in the original sequence from which features are extracted, e.g. location, size. Based on the extracted features, the system can track and classify objects in consecutive frames.

1.1.1 Specification

The implemented system is intended to be used in a single self contained network camera. The blocks are described in Very High Speed Integrated Circuit Hardware Description Language (VHDL) and verified on an Field Programmable Gate Array (FPGA) platform. The system timing specification is given with hierarchal constraints with frame rate and resolution as global constraints. The frame rate is the number of complete frames the system can process per second, measured in frames per second (fps). The constraints are given as parameters configuring the sensor to produce output values at a rate of f_{pixel} MHz. This timing specification propagates down in the constraint hierarchy, i.e. decides the individual timing specifications for every block in the data-path , e.g. segmentation unit, morphology unit, labeling, illustrated in Figure 1.2. The global constraints are set to

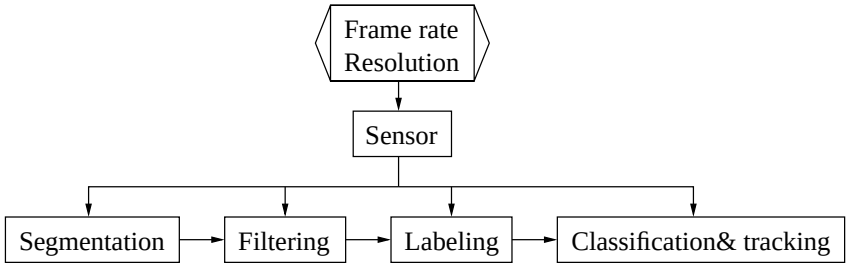


Figure 1.2: An overview of how the hierarchical constraints propagate in the system.

- **Resolution:** $\geq 320 \times 240$.
- **Frame rate:** ≥ 25 fps.

Both these constraints are set as minimum requirements for the implemented system, i.e. the prototype. The frame rate is considered real–time due to limitations in the human vision and gives the system predictions sufficient accuracy, discussed further in Section 3.5. However, compared to other modern image processing systems, the resolution should be increased to at least 640×480, especially in a commercial product.

1.1.2 Design Space Exploration

A general implementation design space includes design time, cost, speed or throughput, power and area (flexibility is excluded since it is hard to measure), illustrated in Figure 1.3 as two separate diagrams. However, neither of these constraints are orthogonal, e.g. increasing the throughput can result in increased power dissipation and area, less design time and limited budget can result in a general purpose processor implementation, thus increased power dissipation. Design time and cost are considered to be more on the project management level as opposed to throughput, power, and area, which are more on the implementation level. In reality, design time and cost are often considered equivalent and are probably the most vital design constraints since they not only involve which technology to choose but also the actual decision which blocks to implement in hardware, i.e. HW/SW partitioning. Furthermore, if the implementation is to be mass produced, implementing an Application Specific Integrated Circuit (ASIC) can be cost effective (even if maximum performance is not required) since the production cost per chip will decrease. However, if power is not constrained and the system is intended for development purposes, rapid prototyping, or will be produced in low volumes, choosing an FPGA implementation seems like the obvious choice. Finally, functional verification on an FPGA before manufacturing an ASIC is always advisory since many errors can be avoided and gives an opportunity to simulate long term effects.

In Figure 1.3(a), as a function of design time and cost, the choice of technology stretches from a full custom ASIC to an FPGA, and at the end, a software implementation running on a general purpose processor. Furthermore, an Hardware Description Language (HDL) implementation can be targeted for either ASIC or FPGA and a software implementation can run either as an embedded processor on an FPGA or on a

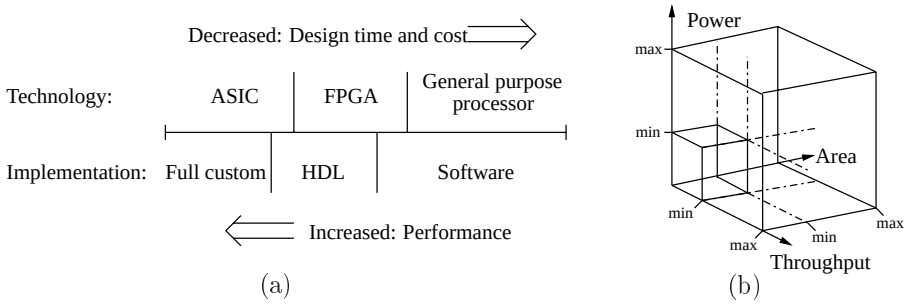


Figure 1.3: (a) Time and cost versus technology, implementation, and performance, (b) a typical hardware design space on the implementation level.

general purpose processor. Naturally, performance in terms throughput, power and area increases together with design time and cost. The decision where to place a design in Figure 1.3(a) can also be physical size dependent, e.g. software running on a workstation is not an issue for the system described in this thesis, since it intended for use in a self contained network camera.

A typical procedure when choosing a point in a hardware design space is to first partition the system into HW and SW by identifying key operations by SW modeling using a software description language, e.g. C, C++, Matlab, etc. The SW modeling includes timing and complexity analysis of the system on which the partitioning decisions are based. After SW modeling and partitioning, the execution time for each block is distinguished, resulting in a throughput constraint specific for each part of the system. When this constraint is set, focus can be spent on the implementation level, i.e. choosing architecture and exploring the throughput, power and area parameters in the design space, shown in Figure 1.3(b). The boxes in the figure illustrates that there are limitations on each axle, e.g. the maximum power it is allowed to consume, how fast it must at least run to still keep the specification, etc. Assuming constrained execution time, the design challenge is reduced to: given a throughput constraint, minimize power dissipation and area.

In our application, since the implemented system is intended to be used in a network camera, the following priority order in the design space can be distinguished

1. **Throughput** – Measured in bandwidth and is set by the system specification.

2. **Power** – Reduced at the cost of area.

3. **Area** – Reduced but not at the cost of either throughput or power dissipation.

Assuming that the global constraints are met, low power is the main design objective since the system is intended for use in a self contained network camera, i.e. in order to avoid heat problems in the camera or possible future use in a hand held device. Any time slack in the timing model is used to decrease the power consumption.

1.1.3 Low power FPGA system design

The total power dissipation P_{tot} in hardware design, i.e. both ASIC and FPGA, consists of three parts [3]: static power P_{stat}, dynamic power P_{dyn}, and direct-path power P_{pd}. The static power dissipation is mainly due to leakage currents, I_{leak}, which are technology dependent and increasing as the threshold voltages in the transistors are lowered as the technology is scaled down. The dynamic power consists of two contributions. The first part is due to the charging and discharging of capacitive loads, C_l. The third part is due to the nature of CMOS design in which a direct path between the supply voltage and ground is present during the switching phase of the gates. This direct path results in a short circuit current, I_{short}, present during the time, t_s, when both transistors are conducting.

Both power contributions, i.e. P_{stat} and P_{dyn}, are dependent on the supply voltage, V_{dd}, but only the dynamic part is proportional to the clock frequency f. The total power dissipation in a design can be written as

$$P_{tot} = P_{stat} + P_{dyn} + P_{dp} = V_{dd} \cdot (I_{leak} + f \cdot (C_l \cdot V_{dd} + I_{short} \cdot t_s)). \tag{1.1}$$

The dynamic power dissipation is design specific and most traditional low power design techniques and tools are aimed at minimizing this part of the power contribution. Typical low power design techniques are: lowering the supply voltage, the use of low power cell libraries, clock gating, etc. In FPGA design, these power contribution principles still holds. But in our application, since many design parameters are limited (e.g. supply voltage, cell library, etc), the low power techniques are basically restricted to:

- **System level** – Which includes clock or block gating, i.e. manipulate the switching activity.
- **Algorithmic level** – Reduce the number of memory accesses, arithmetic operations, and explore resource utilization.

A typical system level power saving technique is to optimize the throughput power balance. This means minimizing the operating frequency of the individual blocks but still keeping the throughput constraint. This procedure will save power since the clock tree has the high switching activity together with large capacitances, f and C_l in Equation 1.1. This can be achieved since the FPGA typically supports multiple clock domains. Clock gating can effectively reduce the switching activity in the clock tree by not letting the clock toggle inside unused blocks. Block gating is an alternative to clock gating and is applied to the input of the blocks. If a block is unused, gating the input prohibits unnecessary switching activity in the combinatorial parts of the block, thus signals to further propagate through the system, dissipating unnecessary power. But since the implemented hardware accelerators are processing the incoming data stream directly (without exception) and are used in a real time system, this technique is not applicable in the system.

On the algorithmic level, memories and memory accesses are power consuming since they typically have high switching activity and large capacitances on the bit-lines. This is especially true for off-chip memories which have additional capacitances on the I/O-ports. Thus, in any FPGA system design, much effort should be spent on minimizing and developing low memory requirement algorithms. Furthermore, there are numerous ways to implement arithmetic operands, e.g. adders, multiplier, etc. From a low power perspective, the basic rule is: less complexity and number of operands, results in less power dissipation. This can be shown by a simple example, e.g. $a(b+c)$ is preferred to $ab+ac$, since one multiplier less is used and will therefore consume less power [4]. Therefore, in our application, low complexity algorithms are preferable and traded for flexibility. Exploring the resource utilization, i.e. a time-multiplexed versus a direct mapped architecture, no clear answer can be given of which architecture will consume the least power. As an example, a time-multiplexed architecture can result in a need for higher clock frequency to keep the throughput constraint, thus an increased switching activity. Since traditional low power ASIC techniques, e.g. pipelining or parallelization, to lower the supply voltage [5], are not applicable in an FPGA, these techniques' applicability in our application is hard to define. However, a good design strategy is to weigh different trade-offs and explore time-multiplexing where possible, still keeping the throughput constraint.

1.2 Research project

The research presented in this thesis is part of the development and implementation of a complete automated surveillance system based on a single self contained network camera in hardware. The aim is to be able to detect, track and classify objects in consecutive frames. Such a system not only compete in terms of a higher frame rate and higher system resolution compared to other general processor solutions but also a reduced power dissipation due to higher hardware resource utilization. Combining a single camera system with e.g. more cameras, sound, etc, would certainly increase the accuracy of the system but is beyond the scope of this thesis.

Three PhD. students are currently involved developing different parts of the system. The author is responsible for the morphology and labeling part. J. Hongtu for the implementation of the sensor interface and the segmentation unit [6]. F. Kristensen is responsible for the feature extraction and tracking SW but also the implementation of additional HW units, e.g. the PPC interface. Furthermore, he also performed an investigation of the impact different color spaces have on shadows [7]. All three are involved in developing the system architecture and integration. The work is initiated and carried out in close collaboration with Axis Communication AB [8].

1.2.1 Main contribution and thesis statement

The main contribution of this thesis is to present implementation aspects of two hardware accelerators: a low complexity morphology unit, and a labeling unit based on contour tracing. A general overview of a complete automated system will be presented outlining and setting the goals for this research project. Additional implementation

aspects of such a system, a prototype, will also be presented but is not considered to be a part of this thesis main contribution. Furthermore, incorporating the two hardware accelerators in the prototype, the following thesis statement can be derived:

- Accelerating key operations in hardware is crucial to achieve realtime performance in an automated digital surveillance system. This should be done under strict power and complexity considerations.

1.3 Thesis outline

The thesis is organized as follows. In Chapter 2, basic digital image processing definitions and concepts are presented, needed as reference throughout the thesis. Chapter 3 presents an overview of a complete system in order to get acquainted with the included blocks. Chapter 4 presents implementation aspects of a low complexity morphology unit. Chapter 5 presents implementation aspects of a labeling unit based on contour tracing. System integration together with implementation details of the prototype are presented in Chapter 6. Finally, conclusions and future work are presented in Chapter 7 and Chapter 8 respectively.

Chapter 2

Digital Image Processing

This chapter presents a brief introduction to common sensor techniques together with basic digital image processing concepts and definitions used throughout the thesis.

2.1 Image Acquisition

A first step in any digital image processing system is to capture an input image or frame. Each frame is divided into picture elements referred to as pixels, which are aligned to a grid with R rows and C columns, i.e. spatial sampling. Image resolution is defined as the number of pixels per frame, e.g. $C \times R$ (or $R \times C$). The origin with coordinates $(0, 0)$ is defined as the upper left corner, as depicted in Figure 2.1. Furthermore, each pixel p in the grid has a certain value corresponding to the light intensity level. This value is quantized and taken from a set of discrete values $I \in \{i_{min}, i_{max}\}$. Without color processing, the number of intensity values in I corresponds to the number of gray levels in the frame, typically a power of two, e.g. $2^1 = 2$ or $2^8 = 256$. The binary representation of i_{max} determines how many bits are required to represent each pixel value in hardware. Therefore, a binary image, I_b, in which the pixel values $p \in 0, 1$, requires only one bit per pixel to represent the content in hardware.

Repeating the image capturing process producing consecutive frames results in an image stream, i.e. video sequence. In a video sequence, the number of images per second is defined as the frame rate, f_{rate}, measured in frames per second (fps). In this thesis, real–time video performance is defined as $f_{rate} \geq 25$ fps.

A color is a light source with a certain wavelength distribution, where wavelengths stretching from about 400 to 700 nm lies within the human visual spectrum [9]. A light source with a rectangular wavelength distribution within the visual spectrum is referred to as white and a light source containing only one wavelength is monochromatic (dirac distribution). The sum of the wavelength distribution is equal to the intensity. Furthermore, since a sensor only measures spatial light intensity, it becomes gray scale by nature. However, to be able to keep the wavelength distribution information, a color space is inferred splitting the intensity into different wavelength contributions: each pixel color is represented as a point in the color space. A common technique to create multiple dimensions, e.g. RGB, is to separate the wavelengths of the incoming light by inferring a color filter on top of the pixel grid. The filter, called a Color Filter Array

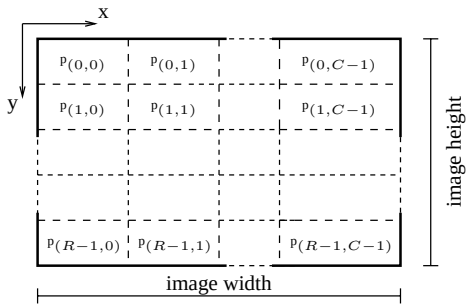


Figure 2.1: Definition of a digital image.

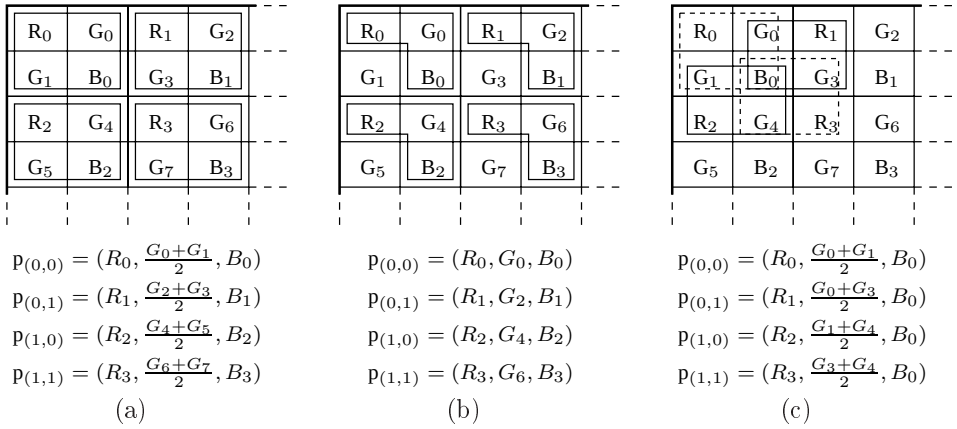


Figure 2.2: An illustration of a Bayer mask together with examples of various pixel setups, (a) is used in our system.

(CFA), is periodic and the actual colors in the filter corresponds to the desired color representation. As an example, the CFA with one red, two green, and one blue filter component will produce the *RGB* color space. This filter, depicted in Figure 2.2, is of special interest, since it commonly used by sensor manufacturers, and is named Bayer filter after its inventor [10]. After the *RGB* color intensities have been measured, the sensor output pixels are created as a set of three color components. Since there are four color components in the CFA and only three in a pixel, a decision on how create the three output values has to be made. When using the Bayer CFA, this is usually done by manipulating the green components with various techniques, e.g. calculating the mean of two green values, illustrated in Figure 2.2(a), skipping every other line of green, illustrated in Figure 2.2(b), or by using a color value to form multiple pixels, illustrated in Figure 2.2(c).

The human eye has two major types of receptors [11]: rods and cones. Rods are only sensitive to incoming light intensity and cones only to color information. Furthermore, cones are divided into three subtypes each sensitive to a specific color: blue, green, and red. The reason for choosing two green pixels in the CFA is that the human eye is more sensitive to green than red or blue. This is due to the fact that the sum of the

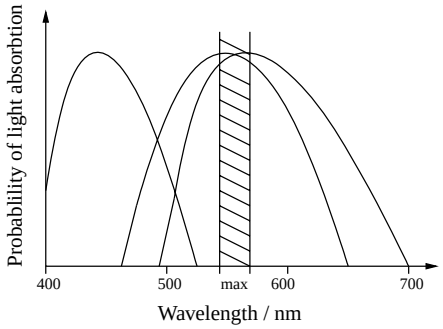


Figure 2.3: Normalized spectrum sensitivity of the human eye for each of the three cone receptor subtypes versus wavelength. The sensitivity is proportional to the probability of a receptor absorbing a light quantum with a specific wavelength.

probability of the color receptors absorbing a light quantum with a certain wavelength has its maximum around ≈ 550 nm, which corresponds to a greenish color, illustrated in Figure 2.3.

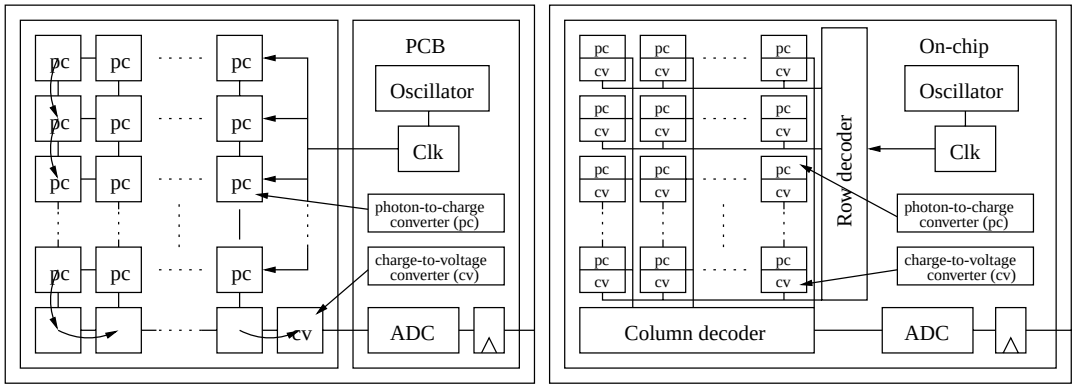
2.1.1 Sensor Techniques

There are mainly two available sensor techniques: Charge Coupled Device (CCD), and Complementary Metal Oxide Semiconductor (CMOS). Both techniques were developed in the late 60s and early 1970s. First, CCDs became dominant due to superior image quality and not until the 1990s could the fabrication methods deliver dimensions and uniformity needed when using the CMOS technology. The subsequent section presents a brief comparison between the two techniques [12] [13]. However, before going into details some important parameters are presented that can be used to compare the two techniques and that are of special interest in automated digital surveillance systems:

- **Dynamic range** – The ratio between a pixel's saturation level and its signal threshold, i.e. the ratio between max and min signal level and should be kept as high as possible.
- **Uniformity** – Consistency in pixel response for different spatial locations under identical illumination conditions, i.e. the same light intensity at different locations of the sensor should result in the same output pixel value. The uniformity can vary at different illumination intensities.
- **Speed** – The data rate of the sensor output pixel stream, measured in MHz.

2.1.2 CCD versus CMOS sensors

Both CCD and CMOS techniques are Metal Oxide Semiconductors (MOS) and function as spatial light samplers; they have a photon-to-electron charge converter at each location in the pixel grid on the image sensor. The electrical charge is converted into



(a) (b)

Figure 2.4: (a) A typical CCD sensor, where the photon-to-charge converter is placed inside pixel array and the charge-to-voltage converter is placed outside of the pixel array. (b) A typical CMOS sensor, where both the photon-to-charge and charge-to-voltage converters are placed directly in the pixels.

a voltage and finally sent to the output. However, it is the read out procedure of the charge that is the major difference between the two techniques.

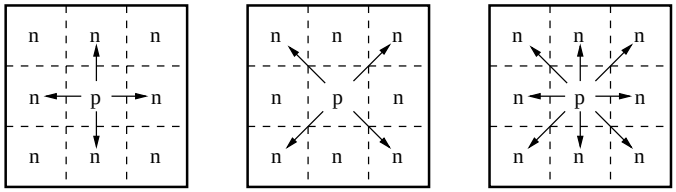
A CCD sensor transfers the charge quantum sequentially from one photon-to-charge converter to another towards the output charge-to-voltage converter. A typical CCD sensor with some required circuitry and logic blocks is depicted in Figure 2.4(a).

In a CMOS sensor, illustrated in Figure 2.4(b), the photon-to-charge and charge-to-voltage conversions takes place directly in the pixel. Therefore, signal routing to each pixel is required, reducing the pixel density to some extent.

Comparing the dynamic range between the sensor techniques, the CCD has an advantage due to less noise in the substrate. Traditionally, the uniformity was problematic in CMOS sensors due to the output amplifier design. However, the gap is closing but CCDs still have an advantage. The image read output speed is one of the most important parameters since at a given frame rate, a faster pixel output speed results in a longer processing time per frame for the system. The speed is about the same for both techniques. A unique feature of the CMOS sensor is the ability to access a region of the pixel grid. This is important when developing an image system to maximize the resolution without changing sensor. Another important issue is the amount of required post processing. CMOS sensors typically have more post processing integrated on-chip, e.g. timing generation, Analog-to-Digital Conversation (ADC), noise reduction, etc, as opposed to the CCD which has most processing on the camera Printed Circuit Board (PCB). This together with the fact that CMOS sensors can be manufactured in standard MOS processes makes the cost per sensor less for CMOS than CCDs. Based on this brief evaluation, summarized in Table 2.1, together with lower power requirements, the CMOS sensor is more suitable in our application.

Table 2.1: A comparison of important parameters between the CCD and the CMOS sensor techniques.

Sensor property	CCD	CMOS
Dynamic range	Better	Good
Uniformity	Better	Good
Speed	Same	Same
Windowing	-	Integrated
Cost	More	Less



(a) (b) (c)

Figure 2.5: (a) A pixel p and its 4-neighbors denoted $N_4(p)$. (b) A pixel p and its D-neighbors denoted $N_D(p)$. (c) A pixel p and its 8-neighbors denoted $N_8(p)$.

2.2 Fundamental Pixel to Pixel based Relationships

Definitions and concepts presented in this section are taken from [14] [15] and are to used as background knowledge for the subsequent chapters.

2.2.1 Neighborhood

A spatial pixel neighborhood in a frame is defined as a square of size 3×3, centered around a pixel p at location (x, y). In digital image processing, it is often desirable to perform a calculation based on only the underlying pixels of this geometrical window. When moving this window from pixel to pixel, thereby shifting its underlying pixels, it is often referred to as a sliding window.

A pixel p at location (x, y) then has two horizontal and two vertical neighboring pixels at coordinates

$$(x-1, y), (x+1, y), (x, y-1), (x, y+1), \tag{2.1}$$

defined as the 4-neighbors of $p(x, y)$ and denoted $N_4(p)$, depicted in Figure 2.5(a). Furthermore, $p(x, y)$ also has four diagonal neighbors at coordinates

$$(x-1, y-1), (x+1, y-1), (x-1, y+1), (x+1, y+1), \tag{2.2}$$

denoted $N_D(p)$, depicted in Figure 2.5(b). These pixels together with $N_4(p)$ are defined as the 8-neighbors denoted $N_8(p) \in N_4(p) \cup N_D(p)$. In words, a pixel and its closest

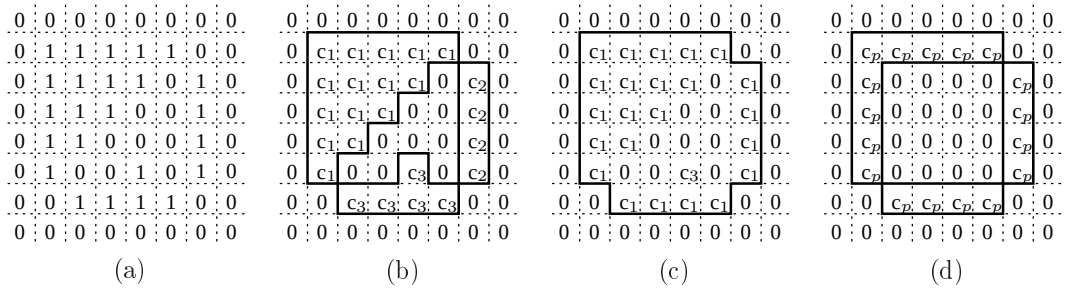


Figure 2.6: (a) An arbitrary binary image, (b) corresponding 4-connected clusters, c_1, c_2 and c_3, (c) corresponding 8-connected cluster, c_1, (d) corresponding 8-connected contour pixels, c_p.

neighboring pixels in all directions, is often referred to as nearest neighbor, illustrated in Figure 2.5(c).

2.2.2 Connectivity

Two pixels, p_1 and p_2, in a binary image are

- **4-connected** – if $p_1 \in N_4(p_2)$
- **8-connected** – if $p_1 \in N_8(p_2)$

and the *adjacent criterion* is fulfilled. The adjacent criterion or adjacency is a predefined pixel value condition, e.g. $p_1 = p_2$ or $|p_1 - p_2| < T_{threshold}$. Connectivity is a fundamental concept in digital image processing from which other important concepts are derived, e.g. contours, regions, distance, etc. An example of how 4- and 8-connectivity affects clustering is illustrated Figure 2.6(a)–(c). Notice how the 4-connectivity rule misses diagonal transitions as opposed to the 8-connectivity rule, i.e. resulting in three clusters instead of one.

2.2.3 Clusters

A cluster C is defined as a set of connected pixels that are either 4- or 8-connected. Each cluster has a contour, L, which consists of contour pixels p and is defined as

$$L = \{p \mid p \in C, \exists q \in N_4(p), q \notin C\}, \tag{2.3}$$

which means that a contour pixel p has at least one 4-connected neighboring pixel outside of C. An example of 8-connected contour pixels of an arbitrary shaped cluster is illustrated Figure 2.6(d).

2.2.4 Spatial operator

A spatial operator is a mathematical function that applies to the center pixel of the sliding window taking input data from the pixels that which are currently covered by the window. A spatial operator is defined as

$$g(x, y) = T[f(x, y)], \tag{2.4}$$

where $f(x, y)$ is the input pixel, T is the operator, and $g(x, y)$ is the output image. Simple examples of spatial operators are minimum, maximum, average, median, etc. It is also possible to form more advanced operators such as gradients.

Chapter 3

An Automated Digital Surveillance system

This chapter presents an overview of a single camera, self contained, automated digital surveillance system with the ability to track and classify objects without human interaction in real–time , i.e. ≥ 25 fps. In order to reduce the amount of data processed by SW, the main idea is to decouple the processor from the image stream by cutting out the interesting parts in a frame using the hardware accelerators and extracted features.

A conceptual overview of the implemented system is depicted in Figure 3.1 with corresponding input and output to each block. The original video is captured and sent as input to both the segmentation block and the feature extraction block, depicted in Figure 3.1(a). The segmentation block produces a binary motion mask, illustrated in Figure 3.1(b). This binary mask contains noise which is filtered using a morphological unit, which also can reconnect split objects to some extent, depicted in Figure 3.1(c). Ideally, the binary mask now only contains the objects of interest and are sent to the labeling block. The different objects are assigned a unique label in order to distinguish between the different clusters, illustrated in Figure 3.1(d) as different shades of gray. Furthermore, performing a logical AND operation on the original image stream and the labeled output, results in that the interesting pixels in the original image stream, from which features are extracted, can be cut out, shown in Figure 3.1(e). After the feature extraction stage, each cluster has features assigned to them, e.g. location, size, etc, depicted in Figure 3.1(f). These features are used to produce the final system output, shown in Figure 3.1(g) as bounding boxes around each cluster with corresponding cluster data. Using these cluster properties, objects can be tracked in consecutive frames.

3.1 Segmentation

A first step in any general computer vision processing system is to distinguish objects of interest in a frame. This procedure is called segmentation and its purpose is to create a motion mask, which is binary in our application, separating the frame into foreground (moving objects), and background which constitutes the rest of the image (static objects). For humans this is most natural and is embedded in our perception but for a computer vision system, this is a matter of extensive calculation. The needed complexity of the segmentation algorithm mainly depends on the lighting conditions in the user environment since it affects the dynamic range of the image stream. A

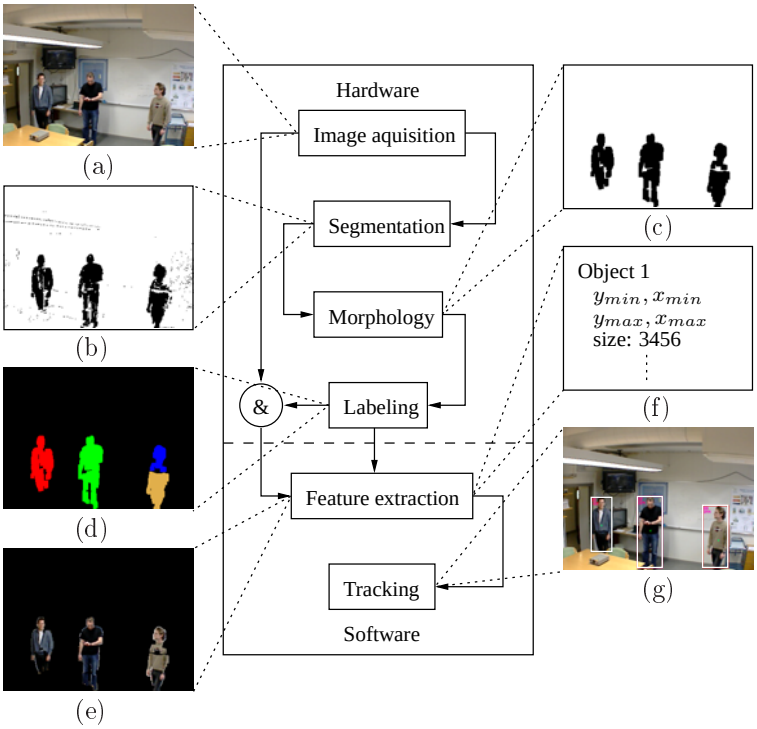


Figure 3.1: An overview of an automated surveillance system with an example of corresponding input and output to each building block, (a) original video, (b) segmented output, (c) filtered output, (d) labeled output, (e) and (f) final tracked output with corresponding logged object features.

low dynamic range in the image stream results in increased noise in the motion mask, i.e. falsely detected objects. Ideally, the scene lighting is static but this is not the case even in a indoor environment. Real-life scenes include many segmentation obstacles, e.g. varying illumination, appearance and disappearance of static objects, etc. Ideally, a robust segmentation algorithm should be independent of camera placement and adaptive to changes in illumination.

In an automatic surveillance system, the objects of interest share at least one common property, i.e. motion. The objects are moving or have at least moved at some point in consecutive frames. The simplest way to distinguish inter-frame motion is to calculate the difference frame, i.e. the absolute value of the subtraction of two consecutive frames. This calculation effectively indicate any change in pixel value but will also produce a lot of noise due to non uniformity in the image acquisition step and changes in the lighting condition. Adding a threshold for the minimum difference will result in that a foreground pixel will be able to handle slow illumination variations and suppress noise and therefore improve the segmentation result. But using a global threshold will often not be sufficient in an environment with varying illumination and can result in that objects will be lost. Furthermore, this simple type of algorithm will encounter problems with periodic background movement which is often found in outdoor environ-

ments, e.g. swaying trees, a water surface, flags, etc. Therefore, an adaptive algorithm based on pixel statistics was developed by Stauffer and Grimson [16] which is both adaptive to varying light conditions and robust to periodic background motion.

The following section will not give a thorough investigation of various segmentation algorithms but rather give a brief overview of the segmentation algorithm used in the implemented system.

3.1.1 Gaussian Multi Modal algorithm

An adaptive algorithm based on a mixture of gaussian distributions was developed by Stauffer and Grimson [16]. The basic idea is to model the variation of a pixel X over time at a specific location (x, y) in a frame with statistical distributions with a mean value μ and variance σ^2. The probability of observing a certain pixel value ,$p_{(x,y)}$, at time t can be written as

$$P(p_{(x,y,t)}) = \sum_{i=0}^{J-1} \omega_{i,t} \cdot \eta(p_{(x,y,t)}, \mu_{i,t}, \Sigma_{i,t}), \tag{3.1}$$

where J is the number of distributions per pixel and $0 < \omega_{i,t} < 1$ is a weight factor, increasing with the number of matches of the i^{th} distribution, $\Sigma_{i,t}$ is the covariance matrix of the i^{th} distribution, and η is the gaussian probability density function. The covariance matrix $\Sigma_{i,t}$ is assumed to have the form

$$\Sigma_{i,t} = \sigma_{i,t}^2 I, \tag{3.2}$$

where the pixel colors are assumed to be independent but have the same variances, $\sigma_{i,t}^2$. This approximation is important and results in that each pixel distribution can be modeled as a circle in the 2-D case and as a sphere in the 3-D color space centered around the mean value. However, in order to reduce complexity, the circles and spheres are approximated as squares in the 2-D case and cubes in the 3-D color space. Hence, the matching criterion is a cube with a side of a factor K times the standard deviation, σ, for that specific distribution ($K = 2.5$ based on empirical evaluations). Matching a pixel against a specific distribution is equal to evaluate if its value lies within the cube of that distribution. A matching criterion for the 2-D and 3-D case are depicted in Figure 3.2.

To determine wether a pixel $p_{(x,y,t)}$ is part of the foreground or the background, it is matched against each of its corresponding distributions. The pixel is regarded as foreground if it does not match any distribution or the matching criterion is fulfilled on a distribution with low weight, otherwise it is regarded as background. If a pixel does not match any distribution, the distribution with the lowest weight is replaced by a distribution with the current pixel value as mean value and with an initial high variance together with a low weight. If there is a match, the weight for the matching distribution is increased, thus a new static object will eventually become background, e.g. a moving

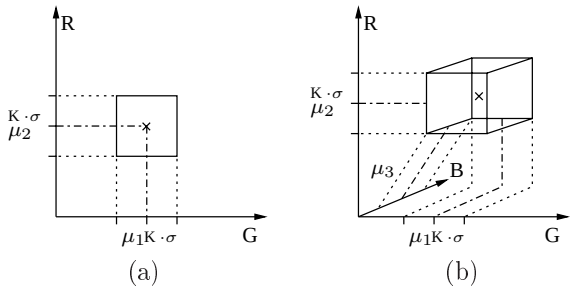


Figure 3.2: (a) $2-D$ matching criterion, (b) $3-D$ matching criterion.

car will first be detected as foreground but after it has been parked, it will eventually become background.

The implementation of the multi modal algorithm [6] reports good segmentation results but produces some over segmentation, shown as small isolated clusters in Figure 3.1(b). However, this noise can be effectively reduced by post processing as long it is small and randomly distributed over the frame. The current implementation uses three distributions per pixel and is running at $100MHz$ on a Virtex II pro FPGA.

3.2 Morphology

After segmentation, the binary mask is full of noise due to non uniformity in the image acquisition step together with noise generated in the segmentation algorithm. Therefore, noise reduction is crucial to avoid over-segmentation causing false objects to pass through the system for classification and tracking. In order to reduce this over segmentation, post processing is required, e.g. morphology.

Morphology is a branch within image processing that can perform logical operations based on the shape of the clusters, i.e. filtering. Furthermore, morphology is well suited for hardware acceleration and the result using a morphological filter on the binary mask is shown in Figure 3.1(c). Morphology in general together with details about the hardware implementation is further discussed in Chapter 4.

3.3 Labeling

To be able to distinguish between segmented clusters in the frame they have to be identified or tagged. This is done in a procedure called labeling which consists of assigning a unique label, i.e. number, to each cluster. After labeling, the system has the ability assign features to a specific cluster which is crucial to be able to track an object in consecutive frames. Implementation details of a labeling unit based on contour tracing is addressed in Chapter 5.

3.4 Features

A feature is a property extracted from an object in the image stream, e.g. location, size, etc. These features are used to describe different properties and to gather information about the clusters. A robust feature describes each object and does not change if the

object is scaled, rotated, or if it enters areas with different illumination. Some examples of features are

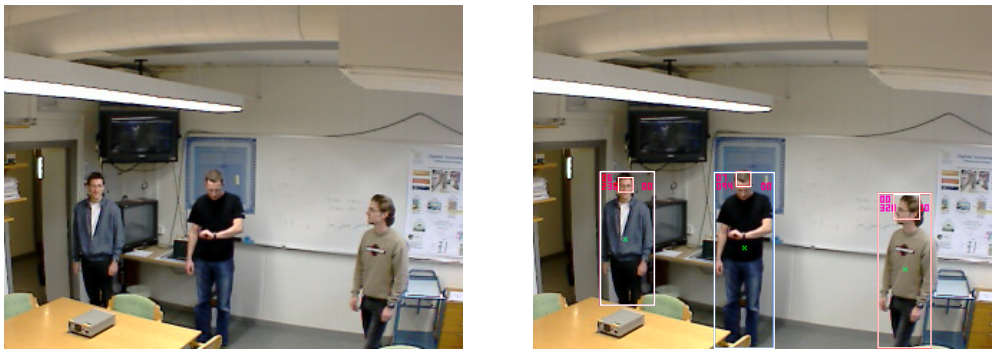
- **Coordinates:** – Maximum and minimum coordinates, i.e. shape and location information.
- **Size:** – Number of pixels in a cluster.
- **COG:** – Center of gravity.
- **Color mean value** – Color mean value for each cluster.
- **Color histogram** – Reveals the color distribution for each cluster.

Certain features are well suited for hardware acceleration since they can be extracted by image processing blocks without inferring complex hardware units. As an example, during the contour tracing phase in the labeling unit, coordinates, size, etc, can be easily extracted from the binary motion mask, discussed further in Chapter 5.

Features are calculated for each object in each video frame and if several features are used, heavy calculation is required. This means that they often are well suited to be implemented in hardware. The features are then sent to the tracking/classification part of the system. With robust features, the amount of data that the tracking/classification algorithms have to handle can be reduced to a level that is suitable for a SW implementation.

In this system, three types of features will be used. First, the ones that can be acquired from the contour tracing in the label unit, i.e. minimum and maximum x and y coordinates, the number of pixels (size), and Center Of Gravity (COG) coordinates. These features are often enough to track non-occluded objects in the video stream. Secondly, color features are calculated to solve occlusions, these include color mean, variance, and histogram of an object. These features have been chosen since they can be calculated from streaming data without any reordering of the pixels and produce a small number of data, i.e. low complexity and memory requirements. In addition, color features are size invariant and with the right color space also lighting invariant, i.e. when separating the luminescence from the color information, e.g. normalized RGB [17]. Thirdly, the tracking program calculates prediction features, such as motion and size predictions, where the size prediction corresponds to motion prediction in the direction towards or away from the camera. These features are used to make an initial guess of which object from the previous frame corresponds to a certain object in the current frame.

The described features are sufficient to perform tracking of moving objects in the video stream during the period they are present in the view of the surveillance camera. However, they are not reliable to track an object that disappears from the scene to reappear at a later time. Furthermore, if two or more persons enter the scene with similar clothing occlusion handling becomes hazardous. The obvious solution to these problems is to include more and better features, which is currently investigated.



(a) (b)

Figure 3.3: Output of an an automated surveillance system, (a) original video, (b) human and simple face detection shown as boxes around corresponding area.

3.5 Classification and Tracking

All features discussed in Section 3.4 are assumed to be quasi stationary, i.e. they have low variation in two consecutive frames since a cluster does not suddenly change color or position. By combining this assumption together with the extracted features, predictions of the features of the clusters in the next frame can be made. Tracking is performed by comparing the predictions and the extracted features, matching the current objects to the objects in the previous frame. The accuracy of the predictions are dependent on the frame rate, i.e. the time an object is able to move in consecutive frames. Thus, real-time performance is crucial to the overall system performance.

When classifying an object, using a single feature is not reliable and a bottom rule is; the more extracted features, the higher system accuracy. Therefore, combining and weighting several features is needed to classify an object. Furthermore, the system becomes more reliable if a priori information about the objects that is searched for is used in the tracking procedure. As example, if the system is searching for humans, the prior knowledge can be that a human walks upright, has a head above its torso, etc. The physical shape of the object is distinguished by comparing the coordinates together with examining the COG, skin can be detected by partial mean value and histogram, etc. An example of a system with human tracking and simple face recognition is depicted in Figure 3.3.

Classification and tracking are most suitable to implement in software running on a general purpose processor, since this process involves a lot of data administration. Another issue is the need to update the algorithms or make them adaptive to a certain scenario, which can be easily addressed using a software solution. However, to be able to achieve real–time performance, the amount of data processed by SW has to be decreased. This is achieved by decoupling the SW from the image stream with features. Furthermore, removing the background keeping only moving objects is achieved by using the extracted maximum and minimum coordinates for every cluster. Based on these coordinates, the important area in the original video can be cut out, letting the software

process only this reduced amount of data, illustrated in Figure 3.1(e).

How accurate the tracking and classification parts are depends on how reliable and robust the features are. The question is somewhat subjective and is left unanswered but will certainly increase with an improved segmentation result, more extracted features, higher resolution and frame rate.

Chapter 4

Morphology

The word morphology is a combination of *morphe*, Greek for "form" or "shape", and the suffix -ology, which means "the study of". Consequently, the word morphology means the study of shapes. In digital image processing, morphology is used as designation for a mathematical tool used to manipulate the shape or understand the structure of clusters of connected pixels. The technique was originally developed by Matheron and Serra [18] at the ecole des Mines in Paris in the mid sixties. Morphology is set theory based methods of image analysis providing a quantitative description of geometrical structures. It plays a key role in many digital image processing applications, e.g. computer vision, object recognition, automatic surveillance, etc. Furthermore, morphology was originally developed for binary images, i.e. the 2-D integer space Z^2, but was later extended and now applies to several image representations, i.e. 3-D gray scale and color integer space Z^3 [19] [20] [21]. However, since the input comes from a segmentation unit in our application, only the binary image representation is considered in this thesis.

In theory, the moving parts of an image should be distinguished as independent objects in the binary mask produced by the segmentation algorithm. However, in reality the mask will be distorted with noise and single objects split into several, e.g. if parts of the moving object has the same color as the background they will probably not be detected as foreground. In order to remove noise and reconnect split objects, one or more morphological operations are performed on the mask.

This chapter presents a low complexity morphological hardware accelerator for binary image erosion and dilation to be used in an automated real–time surveillance system [22]. The main focus is to reduce memory requirements and the number of memory accesses per pixel. First, a brief background on image morphology followed by implementation aspects of the actual hardware accelerator.

4.1 Basic Set Theory Definitions

Let A be a set in the binary 2-D space Z^2, representing a binary (input) image. Let B be another set in Z^2 representing the sliding window, referred to as structuring element (SE) using morphological nomenclature. The elements in the sets are the coordinates to the pixels that constitutes the clusters. Furthermore, $\hat{B}$ is the *reflection* of B, defined as

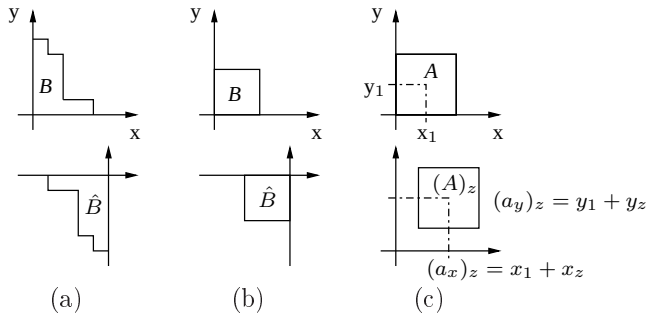


Figure 4.1: (a) A set B and its geometric inverse, (b) a set B and its geometric inverse, i.e. $\hat{B} = B$, (c) a set A translated by z, i.e. $(A)_z$.

$$\hat{B} = \{\hat{b} \mid \hat{b} = -b, \forall b \in B\},$$

which means that $\hat{B}$ constitutes of elements (coordinates) $\hat{b}$ that are equal to all elements $b \in B$, multiplied by -1, resulting in its geometric inverse, depicted in Figure 4.1(a). A special case is when B is reflection invariant which occurs when the reflection is symmetric in both the x and y direction to the original set, e.g. a square or a circle, denoted $\hat{B} = B$. An example of a reflection invariant set B is depicted in Figure 4.1(b). Furthermore, a set translation by a point $z = (z_x, z_y)$, denoted $(A)_z$, is defined as

$$(A)_z = \{(a)_z \mid (a)_z = a + z, \forall a \in A\},$$

which means that $(A)_z$ constitutes of elements $(a)_z$ that are equal to the elements $a \in A$ translated by z, depicted in Figure 4.1(c).

4.2 Morphological Operations

A morphologic operation is equal to a spatial operator, defined in Section 2.2.4 and is related to image convolution [23] but is more on the logic level rather than the numeric level. A morphological operations could be performed using a convolution architecture [24] but without taking advantage of that the structuring element (SE) consists of ones and zeros, resulting in excessive hardware. Erosion and dilation are two fundamental operations from which many other are derived [2] [18], e.g. opening (erosion followed by dilation), closing (dilation followed by erosion), hit-or-miss transformation, etc. Therefore, the need for efficient erosion and dilation algorithms and implementations is evident.

4.2.1 Erosion

Let $A, B \in Z^2$ represent a binary input image and a structuring element respectively. Erosion is defined as

$$A \ominus B = \{z \mid (B)_z \subseteq A\},$$

which means that the erosion of A by B is a set that contains elements z such that B translated by z is a subset of A. For an arbitrary SE, this means that the output pixel with the same coordinates in the pixel grid as the center pixel in the SE becomes 1 if and only if all the pixels in the input image that is covered by a 1 in the SE are equal to 1. It can be compared to a logical *AND* operation or taking the minimum of the pixels in the input image at the coordinates where the SE is equal to 1.

An example of binary erosion using a SE of 3×3 is illustrated in Figure 4.2(a) – (e). Sliding the SE over the image, from left to right, top to bottom, shifting the center pixel, the first input pixels resulting in an output equal to 1 is depicted in Figure 4.2(a) with corresponding output in (b) (pixels marked as "-" have not been processed). Continuing sliding the SE, the last pixel to produce an output equal to 1 is illustrated in Figure 4.2(c) with corresponding output in (d). The final output for this particular input is shown Figure 4.2(e).

4.2.2 Dilation

Let $A, B \in Z^2$ represent a binary input image and a structuring element respectively. Dilation is defined as

$$A \oplus B = \{z \mid (\hat{B})_z \cap A \neq \emptyset\},$$

which means that the dilation of A by B is a set that contains elements z such that the intersection B translated by z is not a subset of $\emptyset$. For arbitrary SEs, this means that the output with the same coordinates as the center pixel in the SE becomes 1 if at least one of the pixels in the input image that is covered by a 1 in the SE is 1. This can be compared to performing a logical *OR* operation, or taking the maximum of the pixels in the grid that is currently being covered by a 1 in the SE.

An example of binary dilation using a SE of 3×3 is illustrated in Figure 4.2(f) – (j). The first and last position in the input image resulting in a one is depicted in Figure 4.2(f) and (h) with corresponding output found in (g) and (i), respectively. Completing the scan, the final output for this particular input is illustrated in Figure 4.2(j).

4.2.3 Opening and closing

When combining erosion and dilation, one followed by the other, it is possible to form other important morphological operations, i.e. opening, closing. Opening is an erosion followed by a dilation and closing as a dilation followed by an erosion. An opening of A by B is defined as

$$A \circ B = (A \ominus B) \oplus B,$$

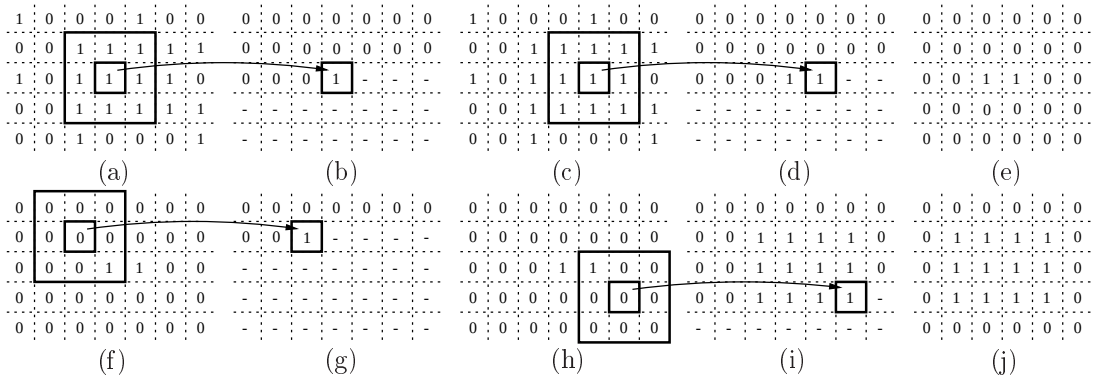


Figure 4.2: (a) – (d) An example of binary erosion using a 3×3 SE with corresponding output in (e), (f) – (i) An example of binary dilation using a 3×3 SE with corresponding output in (f). Pixels marked as "-" have not been processed.

which is of special interest in our application since it performs the actual noise filtering. The filtering is achieved by first eroding the image, resulting in that isolated clusters smaller than the SE are removed. The erosion is then followed by a dilation which restores the remaining clusters to their original size, notice how the isolated pixels in the input image are removed in Figure 4.2. An example of a typical noisy image together with the result after an opening has been performed, i.e. first an erosion using a 5×3 SE containing ones followed a dilation with 7×5 SE also containing ones, are illustrated in Figure 4.3(a) and (b) respectively.

A closing of A by B is defined as

$$A \bullet B = (A \oplus B) \ominus B,$$

which can be used to reconnect split objects. A is first dilated, expanding the clusters with the size of the SE in all directions, which results in that clusters that are closer than $SE_{height} - 1$ and $SE_{width} - 1$ pixels apart are merged. The dilation is followed by an erosion, resulting in that the clusters are restored to their original size but leaving the connections intact. The output after a closing, with the same SE settings as in (a), is illustrated in Figure 4.3(c).

Opening and closing can also be combined. An opening followed by a closing can be useful since it first filters the image and then reconnects possible split objects. Furthermore, changing the dimensions of the SE between the operations can also be useful if the system a priori knows what is searched for. As an example, a vertically outstretched SE used in the closing operation can be useful when detecting humans since we are mostly longer than wider, illustrated in Figure 4.3(d).

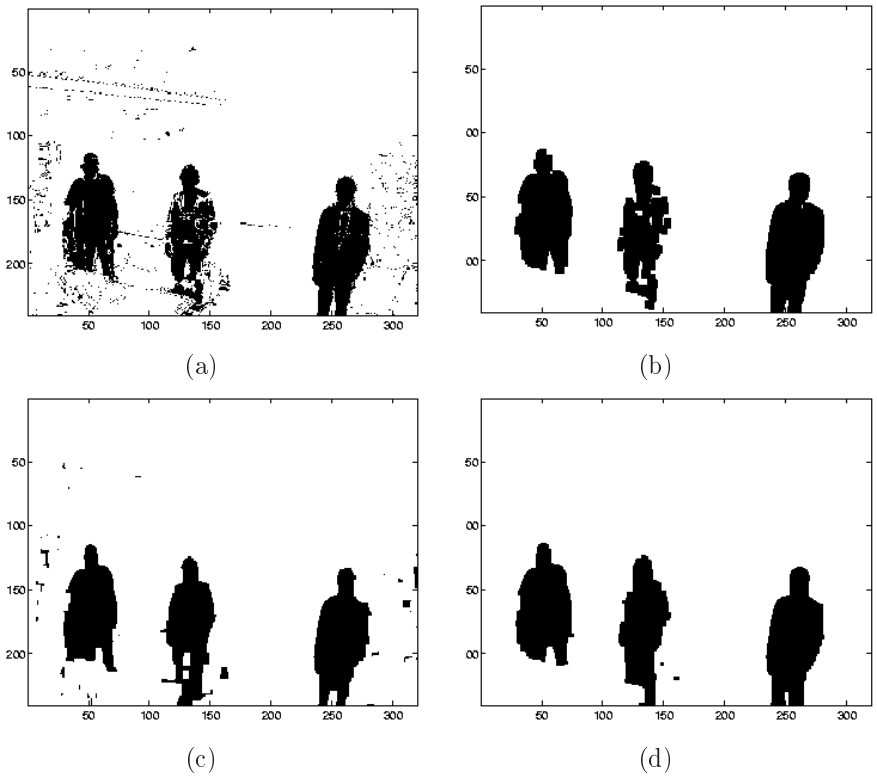


Figure 4.3: (a) A typical binary input, (b) corresponding output after an opening, (c) corresponding output after a closing, (d) corresponding output after performing an opening followed by a closing.

4.2.4 Structuring Element Decomposition

Erosion and dilation are associative which means that if the SE can be decomposed into smaller SEs according to

$$B = B_1 \oplus B_2, \tag{4.1}$$

shown in Figure 4.4, then dilating A with B, results in the same output as first dilating A with B_1 and then dilating the result with B_2 according to

$$A \oplus B = A \oplus (B_1 \oplus B_2) = (A \oplus B_1) \oplus B_2. \tag{4.2}$$

An example of an erosion using SE decomposition is illustrated in Figure 4.5. First, the input is eroded with B_1, resulting in the partial output depicted in (e). This output is then eroded using B_2 resulting in the final output in (j). With a decomposed SE, the number of comparisons per output is decreased from the number of ones in B to the number of ones in B_1 plus B_2. As an example, for a 3×5 rectangular SE consisting only

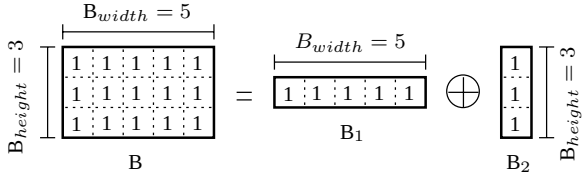


Figure 4.4: Decomposition of 5×3 structuring element B into B_1 and B_2.

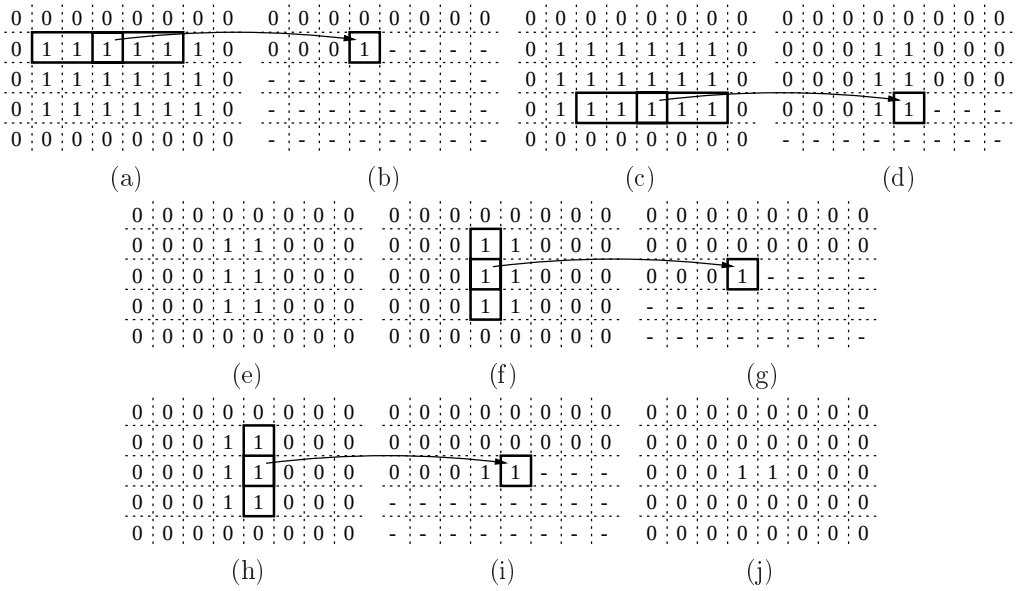


Figure 4.5: Input and output of an erosion using a SE of 5×3 decomposed into B_1 5×1 and B_2 1×3. (a) – (c) First and last pixel producing an output equal to 1 using B_1, (e) final output using B_1. (f) – (i) First and last pixel producing an output equal to 1 using B_2, (j) final result of the erosion for this particular input. Pixels marked as "-" have not been processed.

of ones, illustrated in Figure 4.4, the number of bit operations per output is decreased from 15 to 8.

Finding decompositions to a arbitrarily shaped SE is a hard problem and not always possible [25] [26]. However, one common class of SEs that is reflection invariant, thus easy to decompose, is rectangles of ones. This type of SE is well suited for the opening and closing operations needed in our application since the noise is uniformly distributed in the input image and allows reconnection of possibly split objects.

4.3 Implementation

To easily incorporate the morphology unit into the system, some requirements are set on the architecture. First and most important, input and output data must be processed sequentially from first to last pixel in the binary image to avoid unnecessary memory handling. In addition, this allows for several datapath units to be placed sequentially

after each other without any storage in between. Secondly, the architecture should be small and fast, in order to allow as much time and hardware space for the object tracking/classification part of the system as possible. To increase the overall performance of the system, it is also desirable that the size of the SE can be changed during run time. With a flexible SE size comes the ability to compensate for different types of noise and to sort out certain types of objects in the mask, e.g. high and thin objects (standing humans) or wide and low objects (side view of cars).

An important property of both erosion and dilation is that one is the dual of the other according to

$$A \oplus B = (A' \ominus \hat{B})' \tag{4.3}$$

$$A \ominus B = (A' \oplus \hat{B})', \tag{4.4}$$

where $'$ is bit inversion [27]. It is assumed that the height and width of B are odd numbers, i.e. the origin of B is equal to $O(x, y) = O(\lfloor \frac{B_{width}}{2} \rfloor + 1, \lfloor \frac{B_{height}}{2} \rfloor + 1)$. Furthermore, if the SE is both reflection invariant and decomposable, i.e. $B = \hat{B}$ and $B = B_1 \oplus B_2$, and by combining Equation 4.1, 4.2, 4.3, and 4.4, the following two equations can be derived

$$\begin{aligned} A \oplus B &= (A \oplus B_1) \oplus B_2 = (A' \ominus B_1)' \oplus B_2 \\ &= ((A' \ominus B_1)'' \ominus B_2)' = ((A' \ominus B_1) \ominus B_2)' \end{aligned} \tag{4.5}$$

$$\begin{aligned} A \ominus B &= A \ominus (B_1 \oplus B_2) = (A' \oplus (B_1 \oplus B_2))' \\ &= ((A' \oplus B_1) \oplus B_2)' = ((A'' \ominus B_1)' \oplus B_2)' \\ &= ((A \ominus B_1)'' \ominus B_2)'' = (A \ominus B_1) \ominus B_2, \end{aligned} \tag{4.6}$$

which implies that since both erosion and dilation can be expressed as an erosion; the same hardware can be used to perform both erosion and dilation using a decomposed SE, discussed further in Section 4.3.2.

A common issue for all kinds of sliding window operations, including erosion and dilation, are boundary problems. These occur when the sliding window B, is centered on the boundary of A and thus extend outside A, as shown in Figure 4.6(a). The most common solution is to pad the input image, A, until B, centered on the original boundary, is completely covered and a well defined answer can be obtained, as shown in Figure 4.6(b). Padding is defined as ones if erosion is performed and as zeros if dilation is performed [27] to not affect the output result by the padding. With these definitions, information around the border area of A is not lost, and more complex operations, e.g. closing and opening, will perform correct.

4.3.1 Previous work

Previous hardware implementations of binary erosion and dilation units are mostly focused on small to medium sized SEs. In [28], a direct mapped 2D Finite Impulse

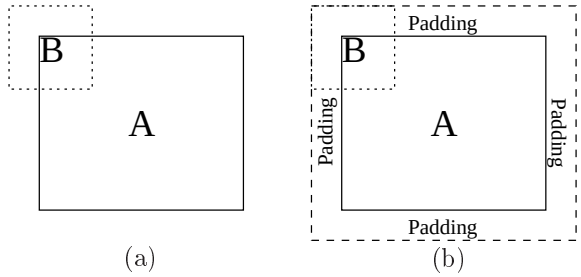


Figure 4.6: Boundary problem (a), padding (b).

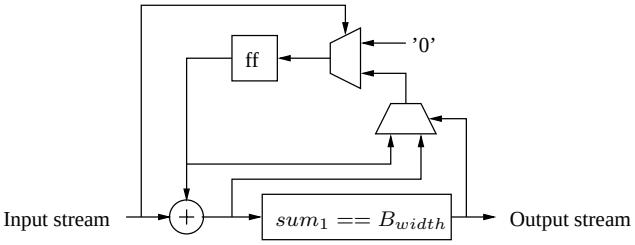


Figure 4.7: Simplified architecture of stage 1.

Response (FIR) type of architecture is implemented where the arithmetic operations are replaced by logic operands, e.g. AND or OR, depending on the SE. This architecture supports sequential pixel processing and gives the opportunity to process the pixels covered by the SE in parallel, which results in that it supports arbitrary shaped SEs. However, due to long delay lines between each logic row of the SE and that every logic operand needs a control signal, i.e. depending on the SE, making it unsuitable for larger SEs requiring additional logic. In [29], a fast implementation is presented exploring the duality of erosion and dilation together with a decomposed SE. Processing each row in the SE in parallel makes it fast but it requires one memory access for each row in the SE, resulting in high memory bandwidth. Furthermore, it does not support a flexible SE size when performing multiple operations in series, which is useful in our application as discussed in Section 4.2.3.

4.3.2 Architecture

The implemented architecture is based on Equation 4.6 taking advantage of a decomposed SE, performing erosion by default. The architecture is depicted in Figure 4.8. However, to perform a dilation, the input A and the result are inverted, according to Equation 4.5, which is done in stage-0 and 3 respectively. Hence, the same inner kernel is used for both operations.

With a rectangular SE of ones, erosion can be performed as a summation followed by a comparison. To perform binary erosion, bits in A that lies directly below the current position of B are added and compared to the size of B. If the sum is equal to the size of B the result is one, otherwise zero. When combining this with decomposition, the summation can be broken up into two stages. Stage-1, compares the number of ones

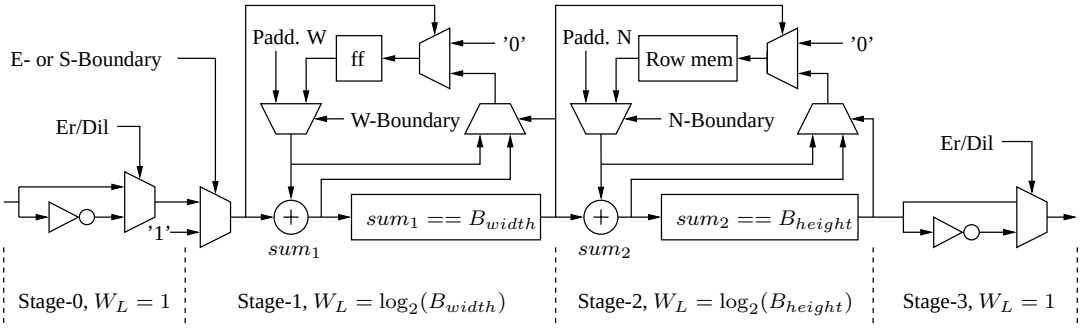


Figure 4.8: Architecture of the erosion dilation unit together with the wordlength, W_L, in each stage.

under B_1 to the width of B_1 and the second stage, stage-2, compares the number of consecutive rows with B_1 ones, i.e. the result from stage-1, to the height of B_2.

When sliding B_1 over a row in A, each pixel in A is used as many times as the width of B_1 to calculate the sum for every partial output value. However, if a running sum records the number of consecutive ones in the currently processed input row, each input is used only once. A simplified block diagram of stage-1 is shown in Figure 4.7, where ff is the flip-flop that stores the sum of consecutive ones. When the input is one, the sum is increased and if the input is zero the sum is reset to zero. Each time the sum plus the input equals the width of B_1, stage-1 outputs a one to stage-2 and the old sum, i.e. $B_{1width} - 1$, is kept to be compared to the next input. The same principle is used in stage-2 but instead of a flip-flop, a row memory is used to store the number of rows of consecutive B_{1width} ones, i.e. hits from stage-1, for each column in A. When the output from stage-1 is one, the sum in the row memory is increased and if the input is zero the sum is reset to zero. Furthermore, each time the output from stage-1 plus the input from the row memory equals the height of B_2, stage-2 outputs a one to stage-3 and the previous sum, i.e. $B_{2width} - 1$, is stored in the row memory. Which sum that should be stored in the ff and row memory, i.e. the increased or the previous sum, is controlled by the outputs from stage-2 and 3. The MUXs in stage-2 and 3 are used for the padding if the architecture is processing pixels outside the input image boundaries.

To handle the boundary problem discussed in Section 4.3, the padding is split into four parts, namely north, east, south, and west padding, corresponding to the top, right, bottom, and left side of A. The north and south padding should extend $\lfloor \frac{SE_{height}}{2} \rfloor$ rows and the east and west padding should extend $\lfloor \frac{SE_{width}}{2} \rfloor$ columns outside A. Since only an erosion datapath is needed to perform both operations, the boundaries are padded with ones to not affect the output result and is independent of which operation is performed. Furthermore, the result of all padding to the west and north can be precalculated. The precalculated result of the west padding will always be equal to $\lfloor \frac{SE_{width}}{2} \rfloor$ and will be the initial value in stage-1. Similarly, the north padding is equal to $\lfloor \frac{SE_{height}}{2} \rfloor$ and will be the initial value in stage-2. The east and south is always equal to one and have to be inserted into the data stream, i.e. the east padding at the end of every row of A and the south padding after A. Figure 4.9 shows all padding in the case that the SE is seven

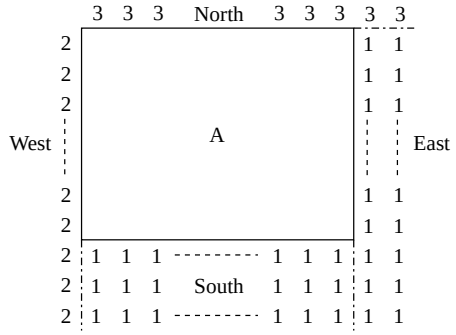


Figure 4.9: An example of the padding on each side of the frame when the SE is seven rows and five columns of ones, i.e. $west = \lfloor \frac{SE_{width}}{2} \rfloor$, $north = \lfloor \frac{SE_{height}}{2} \rfloor$, and $east = south = 1$.

rows and five columns of ones.

A FIFO is located at the input of the datapath to be able to stall the input stream as the boundaries are being processed. The longest period of time the input data have to be stalled is during the south and east padding, calculated as $\lfloor \frac{SE_{height}}{2} \rfloor \cdot (im_{width} + \lfloor \frac{SE_{width}}{2} \rfloor)$ clock cycles. The size of the FIFO is dependent on the number of stalled clock cycles together with the incoming data speed, f_{in}, and the operating speed of the datapath, f_{morph}. Therefore, the size FIFO can be calculated as

$$FIFO_{size} = \lfloor \frac{SE_{height}}{2} \rfloor \cdot (im_{width} + \lfloor \frac{SE_{width}}{2} \rfloor) \cdot \frac{f_{in}}{f_{morph}} \tag{4.7}$$

Due to the sequential pixel processing, the implemented architecture supports connecting multiple datapaths in series enabling more advanced morphological, e.g. opening, closing, etc, shown in Figure 4.10.

4.4 Results and performance

In Figure 4.8, the final architecture of the datapath is shown together with the wordlengths, W_L, in each stage. The input and output parts, stage-0 and 3, have a single bit wordlength, whereas the wordlengths in stage-1 and 2 depends on the largest supported size of B. The wordlengths are, $\log_2(B_{width})$ and $\log_2(B_{height})$ in stage-1 and 2, respectively. The required amount of memory to perform dilation or erosion is

$$mem_{datapath} = \lceil \log_2(B_{width}) \rceil + \lceil \log_2(B_{height}) \rceil A_{columns} \text{ bits}, \tag{4.8}$$

where the first part is the flip-flop in stage-1 and second part is the row memory in stage-2. For example, with a resolution of $A = 320 \times 240$ and supporting a maximum SE size of $B = 15 \times 15$, the required amount of memory for the datapath is $\lceil \log_2(15) \rceil + \lceil \log_2(15) \rceil \cdot 320 = 1284$ bits. Optimizing for power, i.e. operating at the lowest possible frequency, $f_{morph} = 8.1$ MHz (Equation 4.10), and with a incoming data rate of $f_{in} = 7.68$ MHz,

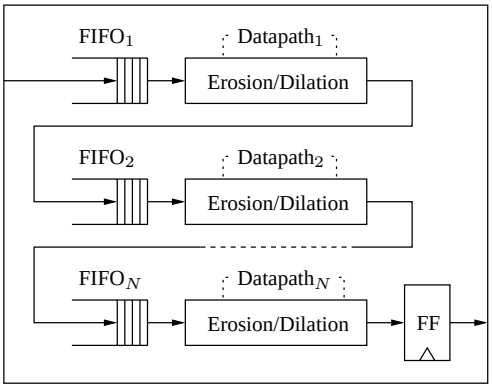


Figure 4.10: Multiple datapaths connected in series, each with individual SEs.

based on Equation 4.7 and 4.10, the FIFO requires an additional 2170 bits of memory. Combining Equation 4.7 and 4.8, the total amount of memory per erosion dilation unit can be calculated as

$$mem_{tot} = FIFO_{size} + mem_{datapath} = 2170 + 1284 = 3454 \text{ bits.} \tag{4.9}$$

The implementation in [28], with the same A and B would require $(B_{height}-1)A_{columns}+B_{width} = 4495$ bits of memory, which is 30% more than the required memory in the presented implementation. Furthermore, this architecture does not handle the boundary problem and sets the border pixels to zero, i.e. when the SE processing pixels outside the image boundaries, resulting in an image size decrease. To be able to get the correct output at the boundaries, an additional FIFO has to be inferred at the input together with inserting the padding into the pixel stream. This would make the presented implementation even more advantageous in terms of memory requirements.

The execution time, i.e. the time from first input to last output, is $t_{exe} = t_{pp} + t_{pad}$, where t_{pp} and t_{pad} is the pixel processing and padding time, respectively. The pixel processing time is equal to the size of the input image, whereas padding time depends on the size of both A and SE according to

$$t_{pad} = \lfloor \frac{B_{width}}{2} \rfloor \cdot A_{rows} + \lfloor \frac{B_{height}}{2} \rfloor \cdot (A_{columns} + \lfloor \frac{B_{width}}{2} \rfloor) \text{ clock cycles.}$$

t_{pad} includes all extra clock cycles due to padding, i.e. the east and south padding in Figure 4.9. With the same A and B as in the memory example above, the total execution time is

$$\begin{aligned} t_{exe} = t_{pp} + t_{pad} &= 240 \cdot 320 + \lfloor \frac{15}{2} \rfloor \cdot 240 + \lfloor \frac{15}{2} \rfloor \cdot (320 + \lfloor \frac{15}{2} \rfloor) \\ &= 76800 + 3969 = 80769 \text{ clock cycles.} \end{aligned}$$

Almost the same result is obtained in [28]. However, the presented implementation will have a somewhat shorter execution time, since the west and north padding never effects the padding time. In fact, padding in general is marginal compared to the processing time for larger input images. No fair comparisons of execution time and memory requirement are feasible with the implementation in [29], since all data management is omitted from that design. Instead, B_{height} different values of A are required as input each clock cycle.

Input signals to the presented morphology unit, in excess of data and the select erode/dilate control signal, are the width and height of B. A controller will set the north and west padding values and produce the control signals *W-boundary*, *N-boundary*, and *E or S-boundary*, accordingly (Figure 4.8).

The unit has been implemented and verified on a Xilinx Virtex II-pro FPGA at clock frequencies up to 100 MHz, an image resolution of 320×240, and a frame rate of 25 fps, supporting a maximum SE size of 15×15.

4.4.1 Optimizations

Based on the discussion in Section 1.1.3: any time slack in the timing model should be used to decrease the power consumption. For a system implemented on an FPGA supporting multiple clock domains, this is achieved by optimizing (decreasing) the clock frequency of the blocks, i.e. their operating speed. In this case, since the morphology block is placed in series with the segmentation unit, according to Figure 3.1, the input is a stream of consecutive ones and zeroes (left to right, top to bottom). The incoming data rate, f_{in} MHz, is dependent on the constraints given to the sensor. The incoming resolution and data rate together with the maximum supported SE size due to padding, puts a constraint on the execution time that the morphology unit can spend on processing each frame. When operating at a speed of f_{morph} MHz, the timing constraint for the unit can be written as

$$\frac{1}{f_{in}} \cdot (im_{height} \times im_{width}) \geq \frac{1}{f_{morph}} \cdot t_{exe} \Leftrightarrow f_{morph} \geq f_{in} \cdot \frac{t_{exe}}{(im_{height} \times im_{width})}. \tag{4.10}$$

With a resolution of 320×240 and a frame rate of 25 fps (each pixel requires four sensor values), resulting in a sensor output pixel speed of 7.68 MHz, the morphology could operate as low as ≈ 8.1 MHz. Running the unit at a higher operating speed will result in unnecessary power dissipation in the clock tree. With an increasing resolution, for a given maximum SE size, t_{pad} becomes even smaller compared to t_{exe}, resulting in that the operating speed of the morphology unit will move asymptotically towards the frequency of the incoming data f_{in}.

Chapter 5

Labeling

After segmentation and filtering, a binary frame is produced containing connected clusters of pixels that represent different objects. Furthermore, assuming that noise has been removed at an earlier stage, the frame now only contains objects of interest that can be traced and classified. To be able to separate and distinguish between these clusters they have to be identified, i.e. labeled. The goal is to assign a unique label to each cluster transforming the frame into a symbolic object mask and tying certain properties to a specific cluster, e.g. coordinates, size, color histogram, center of gravity, etc. These properties enable monitoring of many important properties in a surveillance system, e.g. object trajectories, appearance, disappearance, etc. Therefore, labeling is a fundamental operation in any automatic surveillance or pattern recognition system. In reality, segmentation is far from perfect resulting in shattered objects, i.e. one object consists of several clusters. After labeling has been applied on the clusters, they can be tied together and form larger objects. However, this decision is taken on a higher abstraction level, i.e. in the feature extraction / tracking stage, and the labeling block only delivers a label and certain properties of the clusters.

Another important issue in any automatic surveillance system is how to handle occlusions, or rather how the objects are handled after an occlusion. An occlusion occurs when one object is in front of another from the camera perspective, i.e. two previously separate clusters of pixels are merged and detected as one in the binary mask. An example of two objects moving towards each other is depicted in Figure 5.1(a). In (b), the objects are merged in to one, and are occluded in (c). Hence, labeling is used to sort the objects and tie the corresponding properties to each of the previously occluded objects, but how these are handled by the implemented system is not addressed in this thesis.

The subsequent sections will evaluate different labeling algorithm approaches finishing of with implementation aspects of the chosen algorithm.

5.1 Algorithms

Various labeling algorithms have been proposed over the years, all with benefits and drawbacks. Many are based on the same approach [30] but with improved data administration [31] [32] [33]. A sound survey and evaluation of various labeling approaches

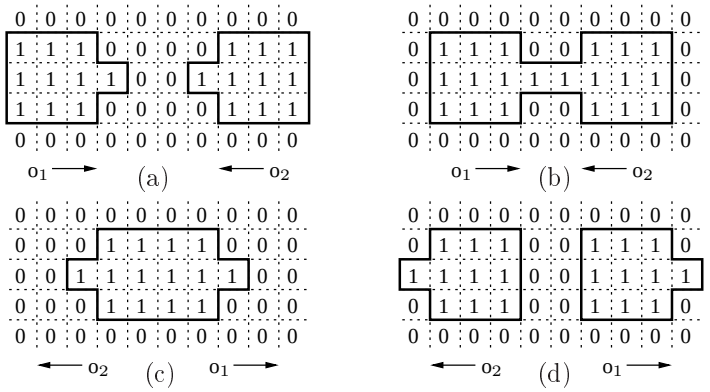


Figure 5.1: (a) Two objects, o_1 and o_2, are moving towards each other and are detected as two separate clusters in the binary mask. (b) The two previously separated objects have now been merged into one, (c) occlusion has occurred, (d) the objects have now passed each other and are again detected as two separate clusters in the binary mask.

in terms of execution time versus number of clusters per frame can be found in [34]. Labeling applies to many different number representations [35], but due to the intended use in our application, only labeling of binary and 8-connected clusters, defined in Section 2.2.2, are considered in this thesis. However, a common property for these algorithms is that they require high memory bandwidths.

Algorithms based on sequential local operations [36] have to handle the same obstacle, i.e. label collisions. In a binary image, a typical label collision occur when a u-shaped object is encountered, depicted in Figure 5.2. Scanning the image from left to right, the first pixel to be labeled is p_1 Figure 5.2(a). Proceeding the scan, reaching p_2 in Figure 5.2(b), a new label will be assigned since there is no momentary information that p_1 and p_2 are part of the same cluster when reaching p_2. However, reaching pixel p_3, Figure 5.2(c), a label collision will occur since there is an ambiguity in which label to assign p_3. The number of label collisions, $label_{collisions}$, depends on the complexity of the clusters. In this thesis, the algorithms are separated depending on how they handle or avoid these collisions and are placed in two major categories:

- **Equivalence Table Based Algorithms** – Multiple scans with a corresponding equivalence table.

- **Contour Tracing Based Algorithms** – A single scan based on contour tracing.

The first category writes the label collisions into an equivalence table during an initial image scan and resolves them during a second. The second category avoids label collisions by tracing the contour pixels and assigning them a label directly.

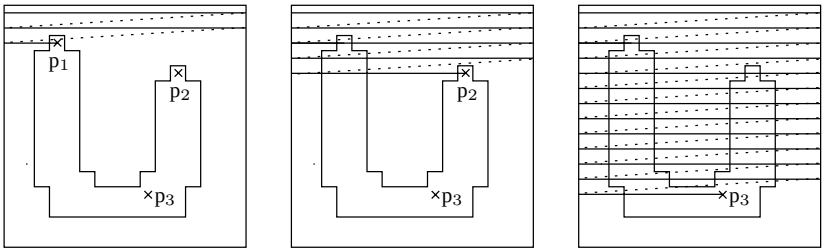


Figure 5.2: An example of a typical label collision, (a) the first upper and leftmost pixel of a cluster is reached, p_1, and labeled with l_1. (b) Continuing the scan, another part of the same cluster is reached, p_2, and is labeled with l_2. (c) Reaching p_3, a label collision will occur since pixels in the same row are labeled using l_1 and the pixel above p_3 are labeled with l_2.

5.1.1 Equivalence Table Based Algorithms

The simplest form of labeling algorithms scan through the image assigning every cluster pixel a preliminary label. The algorithms continues to scan the image memory back and forth until all label equivalences have been resolved for every pixel cluster. An advantage is that the algorithm is based on local and sequential operations. This means that in a forward scan, the algorithm only considers neighboring pixels to the left and upward. This property enables memory burst reads and writes since no random access patterns are required. However, since the number of memory scans is depending on the complexity of the connected clusters, this type of algorithm is not suited for hardware implementation.

Adding an equivalence table to administrate the label collision can effectively reduce the number of image scans, thus called equivalence table based algorithms. Furthermore, two global scans is the minimum required image scans to be able to solve label ambiguities for equivalence table based algorithms. The first image scan can be completed on the fly as the incoming frame is written into the label memory. The preliminary label together with possible label ambiguities can be written directly into the label memory and the equivalence table, respectively. This can be accomplished by storing the previous input row. A second scan resolves the label ambiguities assigning the final label to the clusters. The two scan equivalence table based algorithm is known as the classical approach [30] and will function as reference design in the sense that competing algorithms must achieve better or have major advantages compared to this algorithm.

Depending on how the succeeding processing block interprets the label result, a single scan together with the equivalence table is enough to label an image. However, this algorithm cannot extract other features than cluster size together with maximum and minimum coordinates. It would also require post processing to administrate the label collision and does not thereby complete the labeling procedure in the traditional sense. However, if throughput is the main constraint, this is an interesting solution.

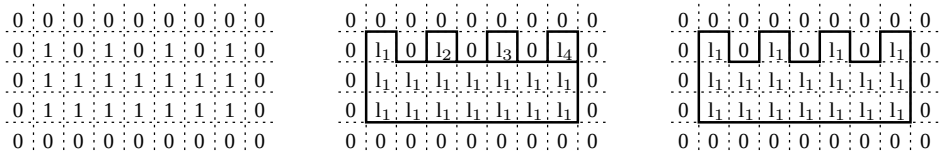


Figure 5.3: (a) An example of cluster containing multiple label collisions, (b) label result after initial scan. This particular cluster occupies four labels due to the multiple label collisions, (c) the final label result after the second image scan, where the label collisions have been resolved.

Memory requirements

To be able to store the resulting labeled image, a memory is needed. This memory is determined by the resolution together with the maximum supported number of segmented clusters that can be labeled, c_{max}, per frame. In this algorithm, c_{max} does not include the number of supported label collisions per cluster, c_{col}, which will occupy a label during the first scan. The number of label collisions per frame is dependent on the shape of the clusters and can be hard to estimate but must always be included in the memory requirement calculation. An example of a cluster occupying several labels during the initial scan is illustrated in Figure 5.3. The assumption that every cluster contains one label collision, results in additional $im_{height} \times im_{width}$ bits of memory. Therefore, the physical dimensions, i.e. wordlength and depth, of the label memory is dependent on the maximum number of clusters that can be labeled together with the estimated number of label collisions per cluster and can be written as

$$mem_{size} = \lceil log_2(c_{max} + c_{col} + 1) \rceil \cdot (im_{height} \times im_{width}), \tag{5.1}$$

where the +1 comes from the fact that 0 is a preoccupied label representing the space between or holes inside the clusters.

The required number of memory accesses for the equivalence based algorithm to complete the labeling procedure consists of several parts. The sequential input data is first written into the label memory assigning the preliminary labels. This results in $im_{height} \times im_{width}$ write operations together with additional label ambiguities written into the equivalence table, which can be neglected since they are $\ll im_{height} \times im_{width}$. This is followed by a second read scan with additional $im_{height} \times im_{width}$ read operations to resolve the label ambiguities. Let $size_i$ be the remainder of a cluster that needs to be relabeled, which results in additional $size_i$ write operations. The total number of memory accesses for a frame containing n clusters, each with $size_i$ pixels that needs to be relabeled, can be written as

$$mem_{access} = 2 \cdot (im_{height} \times im_{width}) + \sum_{i=0}^{n-1} size_i. \tag{5.2}$$

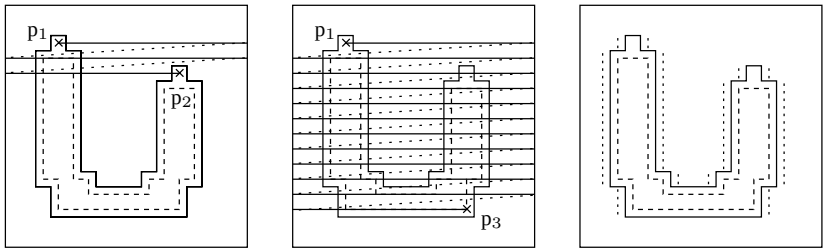


Figure 5.4: Labeling using the contour tracing technique. (a) The scan starts at the previously marked start point, i.e. the first cluster pixel equal to 1, p_1, and the contour of the cluster is traced and labeled with l_1. Continuing the scan, when p_2 is reached, since the contour has been previously labeled l_1, no new label is assigned to p_2. Due to this procedure, no label collision will occur when reaching a pixel that is part of a cluster. (b) The labeling is complete when reaching p_3, i.e. the previously marked end point. (c) The final labeled output.

The total number number of required clock cycles used by the algorithm to process a complete frame is determined by the number of memory accesses. These accesses are determined by the complexity of the input clusters, i.e. the number of pixels that needs to be relabeled. To be able to determine the final execution time, clusters with worst case label collisions have to be constructed. The worst case label collision is when a cluster is covering the complete frame with a label collision in the upper left corner. This means that almost the complete cluster needs to be relabeled, resulting in an upper limit of the latter part of Equation 5.2, i.e. $\sum_{i=0}^{n-1} size_i \leq im_{height} \cdot im_{width}$. An upper limit on the total amount of required memory accesses per frame can therefore be written as $\leq 3 \cdot (im_{height} \times im_{width})$.

5.1.2 Contour Tracing Based Algorithms

As the name implies, contour tracing based algorithms [37] [38] is a technique that traces the contour of each cluster assigning only these pixels a label. The algorithm requires only one global scan together with additional random memory accesses for the contour tracing procedure. By tracing the contour, label collisions will be avoided since when an unlabeled cluster is encountered, the contour of that cluster is labeled immediately and every pixel between two contour pixels with the same label is regarded as part of the same cluster. In Figure 5.4(a), no label collision will occur since p_2 has already been labeled during the prior contour tracing. Continuing the scan, if a cluster with a labeled contour is encountered, the scan proceeds without modification. If an unlabeled pixel is reached, the contour tracing procedure restarts. If the last pixel is reached, i.e. lower right corner illustrated in Figure 5.4(b), the labeling of the frame is completed. The final labeled output for this particular frame is illustrated in Figure 5.4(c).

In detail, the algorithm starts with writing the incoming frame into the labeling memory, marking the first and last pixel equal to 1 as start point and end point respec-

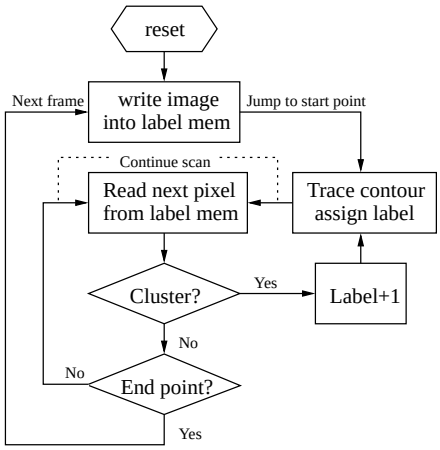


Figure 5.5: Block diagram of the contour tracing based labeling algorithm.

tively. Assuming that the input image is not empty, a global scan starts from left to right and top to bottom, starting directly at the previously marked start pixel, which is equal to the upper leftmost pixel of the first cluster, p_1 in Figure 5.4(a). The contour of this cluster is traced, writing the label into the label memory corresponding to each contour pixel. The contour tracing phase of this cluster is completed when the start pixel is reached a second time and the label is increased by 1. Continuing the global scan, one out of several possible pixel values will be encountered:

- A pixel not part of a cluster ($p_{x,y} = 0$).
- An unlabeled pixel ($p_{x,y} = 1$).
- A previously labeled pixel ($p_{x,y} > 1$).
- A border pixel ($x = im_{width}$, $y = im_{height}$).
- The previously marked end pixel.

Describing the bullets in consecutive order, if a pixel value equal to zero is encountered, the scan continues. If an unlabeled pixel is encountered, the contour tracing phase restarts. If a previously labeled pixel is reached, the global scan continues flagging that it is currently inside a cluster. If a border pixel on the right side is reached, the flag that marks that the scan is inside a cluster is reset, and the scan continues on the left side on the next row. If the marked end pixel is reached, p_3 in Figure 5.4(b), the scan is completed for this particular input and can restart. A block diagram of the algorithm and the described steps is depicted in Figure 5.5.

Furthermore, to avoid pitfalls like tracing contours of possible holes inside the clusters, a reserved label has to be written on each side of the contour, illustrated as a dotted line in Figure 5.4(c). This reserved label together with a flag is used by the algorithm to keep track of whether the scan is currently inside a cluster or not. If the flag is raised, every pixel in between the reserved label is treated as part of the cluster,

whether it is 1 or 0 (hole). Due to this procedure, holes inside clusters are filled which is useful when detecting humans. Hence, entering a previously labeled cluster l_1, i.e. a slice of $n \leq im_{width}$ pixels of this cluster on a specific row, that does not have its contour aligned with the boundaries on the left or right side of the frame, must happen in the following order:

1. p_0 – The first reserved label, i.e. $p_0 = l_{res}$. The inside cluster flag is raised.
2. p_1 – The first contour pixel, i.e. $p_0 = l_1$.
3. Intermediate pixels, i.e. cluster pixels or holes.
4. p_{n-2} – The second contour pixel on the other side of the slice, i.e. $p_{n-2} = l_1$.
5. p_{n-1} – The second encountered reserved label, i.e. $p_0 = l_{res}$. The inside cluster flag is lowered.

If the contour of the cluster is aligned with the boundaries of the frame, no reserved label will be encountered. The scan continues row by row until the last pixel is reached and the algorithm starts over.

Memory requirements

As for the equivalence table based algorithm, this algorithm needs memory to store the resulting labeled image, i.e. label memory. The physical dimension (wordlength and depth) of this memory is determined by the resolution together with the maximum supported number of segmented clusters, c_{max}, in a frame . As opposed to equivalence based algorithm, since no label collisions will occur, c_{max} corresponds directly to the maximum number of clusters that can be labeled per frame. Therefore, the physical memory requirement of label memory can be written as

$$mem_{size} = \lceil log_2(c_{max} + 3) \rceil \cdot (im_{height} \times im_{width}), \tag{5.3}$$

where the +3 comes from the fact that 0, 1, and 2 are preoccupied labels; 0 is used to represent the space between or holes inside clusters, 1 is used to represent the pixels within a cluster, and 2 is used for the reserved label.

Calculating the number of required memory accesses for the contour tracing based algorithm: first, the input stream must be written into a memory resulting in $(im_{height} \times im_{width})$ write operations. This is followed by a second global scan with additional $(im_{height} \times im_{width})$ read operations. For every cluster, trace the contour and write the reserved label on both sides of the cluster. If a frame contains n clusters, each with p_i contour pixels and the need to write l_j reserved label along each side, the total number of memory accesses can be written as

$$mem_{access} = 2 \cdot (im_{height} \times im_{width}) + \sum_{i=0}^{n-1} p_i + \sum_{j=0}^{n-1} l_j. \tag{5.4}$$

The total execution time t_{exe} (number of clock cycles) is determined by the number of memory accesses which are directly proportional to the complexity of the input clusters; a complex contour results in more memory read and write operations. Theoretically, to determine the final execution time, clusters with worst case contours have to be constructed, which can be hard to prove mathematically. Instead, the latter part of Equation 5.4 is left with an assumption that the maximum number of memory accesses needed by the algorithm cannot exceed the image size, i.e. $\sum_{i=0}^{n-1} p_i + \sum_{j=0}^{n-1} l_j \leq (im_{height} \times im_{width})$. Hence, as for the equivalence table based algorithm, the maximum number of memory access needed by the contour tracing based algorithm will not exceed $\leq 3 \cdot (im_{height} \times im_{width})$.

5.2 Algorithm evaluation

When choosing algorithm for implementation, important properties for our application was compared and evaluated, i.e. throughput, power dissipation, memory requirements, and extracted features. The subsequent section will address these parameters consecutively.

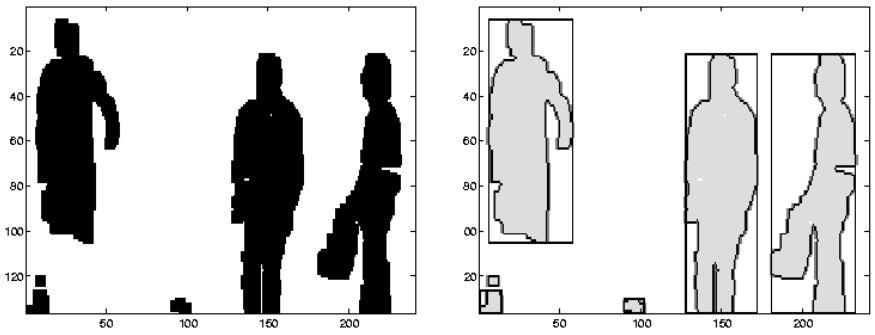
5.2.1 Throughput and power dissipation

Throughput and power dissipation are closely related since throughput is dependent on the number of memory accesses which are a major power dissipation source in any implementation. Comparing upper limits of the total amount of required memory accesses, no major advantage can be distinguished between the two algorithms in terms of throughput and memory accesses. However, the one scan algorithm mentioned in Section 5.1.1 has a major throughput advantage over any other labeling algorithm.

Based on the discussion in Section 1.1.3, a major contribution of the total power dissipation is lost in the clock tree. But since the labeling unit is continuously working on the image stream, clock and block gating can not be applied. However, a power saving technique that is useful in this application is lowering (minimizing) the operating frequency of the unit, f_{label} MHz. The incoming data rate is set to f_{data} MHz and by using the worst case number of memory access for the algorithm, i.e. $3 \cdot (im_{height} \times im_{width})$, a lower bound on the operating frequency can be written as $f_{label} = 3 \cdot f_{data}$.

5.2.2 Memory requirements

Comparing the memory requirements (label memory) between the algorithms, the contour tracing technique has an advantage over the equivalence table based algorithm. This is due to that for the same maximum number of clusters that can be labeled, c_{max}, allowing one label collision per cluster, the equivalent table based algorithm requires additional $im_{height} \times im_{width}$ bits of memory compared to the contour tracing based algorithm. In our application, the resolution is set to 320×240 and c_{max} to ≈ 64 based on simulation. Approximately is used instead of equal since some labels are preoccupied and the width of the label memory is adjusted to the closest power of two, e.g. $c_{max} = 61$ in the contour tracing based algorithm which results in a label memory width of 6 bits. Using these settings and by combining Equation 5.1 and 5.3, it can be shown that the



(a) (b)

Figure 5.6: (a) A typical input frame to the label unit, (b) corresponding labeled out. Notice that gray pixel on each side of the contour which corresponds to the reserved label, and the hole in the middle cluster.

contour tracing based algorithm requires $\approx 14\%$ less memory than the equivalence table based algorithm according to

$$\frac{mem_{contour}}{mem_{equivalence}} = 1 - \frac{\lceil log_2(c_{max}+3)\rceil}{\lceil log_2(c_{max}+c_{col}+1)\rceil)} = 1 - \frac{6}{7} \approx 0.14.$$

5.2.3 Extracted features

Extracting properties by post processing in the tracking stage or by a general purpose processor can be time consuming and every property that can be extracted without inferring additional complex hardware is an advantage and should be implemented. These properties are delivered to the tracking stage in which many decisions are made, e.g. which clusters to merge, tracing objects in previous frames, etc. In comparison, both types of labeling algorithms can extract maximum and minimum coordinates and size of each cluster. However, the contour algorithm has the ability to add Green's formula during contour tracing and thereby calculate the moments [39]. The moments are used to calculate the COG of the cluster which are useful after occlusion. In addition, this algorithm has the hole filling property which is useful when tracking humans.

5.3 Implementation

Based on the algorithm evaluation in Section 5.2, i.e. the low memory requirement and the ability calculate COG, the contour tracing algorithm was found more suitable to be used in a our application. In the implemented system, the unit should not only perform labeling on consecutive frames but should also extract features, i.e. the maximum and minimum coordinates and size of the clusters. The labeled result is stored in the label memory with corresponding features stored in the cluster memory. Since the output of the unit is the memory content and to maximize the time the SW can access this

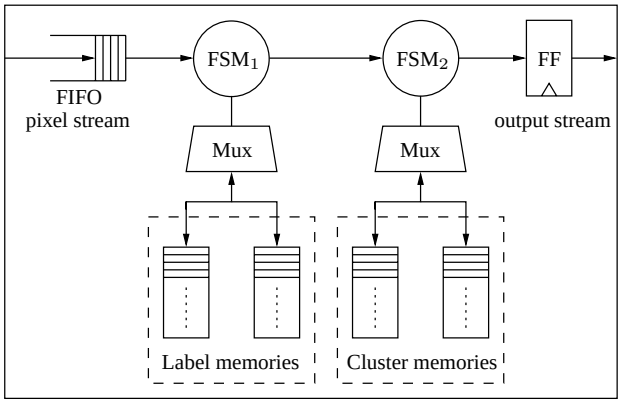


Figure 5.7: Overview of the implemented architecture.

result, a double memory architecture is inferred. Hence, as the algorithm is labeling a frame, accessing one memory pair, it gives random access to the other pair. The double architecture requirement is independent of the algorithm choice, thus not considered in the algorithm evaluation.

A fragment of a typical input frame with corresponding output of the implemented labeling unit can be seen in Figure 5.6. The reserved label, discussed in Section 5.1.2, is illustrated as the gray pixels on each side of the contour.

5.3.1 Architecture

An overview of the architecture is illustrated in Figure 5.7. A FIFO is located at the input since the data stream is stalled as a frame is currently being labeled. FSM_1 in Figure 5.7, first writes the incoming frame into the labeling memory. Secondly, a global scan starts from left to right and top to bottom, searching for the upper leftmost pixel equal to one in a cluster. This pixel is marked as starting point and the contour of that cluster is traced, writing the label to each contour pixel into the label memory. When the starting point is reached a second time, the contour of that cluster is completely traced and the global scan continues until another unlabeled cluster is encountered.

Assuming that a connected cluster is encountered. To be able to trace the next contour pixel of this connected cluster, a direction d in form of a matrix is inferred, depicted in Figure 5.8(a). This matrix marks the direction from the previous contour pixel to the current contour pixel. Since the global memory scan is sequential from left to right and top to bottom, the default direction is right, 1 in the matrix. The lookup table (LUT), shown in Figure 5.8(b), contains the start direction to the next pixel to search for the consecutive contour pixel based on the direction, hence referred to as the initial search LUT. The second LUT, depicted in Figure 5.8(c), contains the actual number to add to the current address to reach each corresponding direction in the matrix, thus referred to as the address LUT. As an example, with a resolution of 320×240, to search in the downward direction, i.e. 3, $im_{width} = 320$ has to be added to the current address. Furthermore, if there is a hit, i.e. the next contour pixel is found, the direction is updated and the search starts over. If there is a miss, the

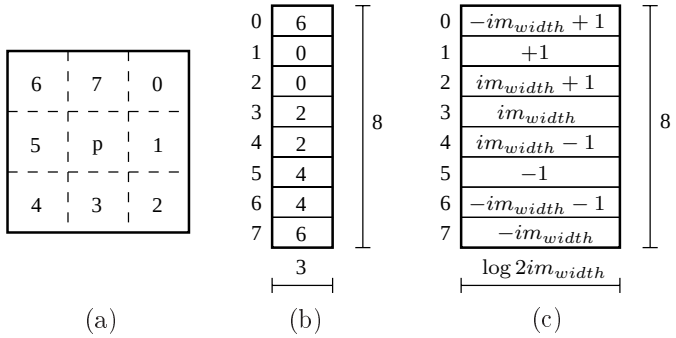


Figure 5.8: (a) Definition of the direction matrix, (b) the start search direction matrix, (c) the address LUT, i.e. the number to add to the current address to reach each corresponding direction.

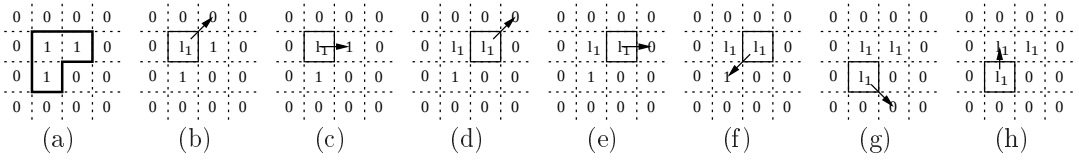


Figure 5.9: (a) A small cluster that is to be contour traced, (b) the start pixel is marked and assigned a label, l_1. The arrow indicates the initial search direction for the next contour pixel. (c) A new contour pixel is encountered in the direction indicated by the arrow. (d) The current position is updated and the arrow indicates the new initial search direction for the next contour pixel. (e) The search direction is increased, until a new contour pixel is found. (f) A new contour pixel is found in the direction indicated by the arrow. (g) The label is assigned and the search starts in the direction indicated by the arrow. (h) The search direction is increased until a new contour pixel is found, in this case equal to the start pixel, ending the contour tracing.

direction is incremented together with the number to add to the current address. This procedure reduces the number of memory accesses since the location of a consecutive contour pixel is not ad hoc. An illustration of the contour tracing of a small cluster with corresponding search directions is shown in Figure 5.9. In (b), the pixel is marked as start pixel and assigned the current label, l_1. Since the default direction is 1, a lookup is made in initial search LUT, receiving a 0. This 0 is used for a second lookup in the address LUT, resulting in that $-im_{width} + 1$ should be added to the current address to start searching for the next contour pixel. Since no contour pixel is found at this address, the index to the address LUT is increased by 1, and the next number to add to the current address is +1. On this address, the next contour pixel is found and the current position is updated together with assigning this pixel the current label. The pixel was found in a direction equal to 1, and a new lookup in the initial search LUT is made, again receiving a 0 which is used to make a new lookup in the address LUT. If a new contour pixel is not found on the current search address, the index to the address

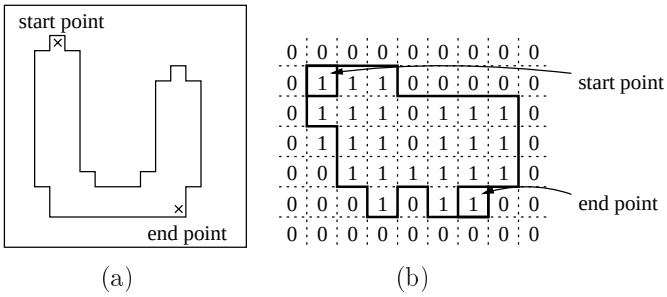


Figure 5.10: (a) Start and end point of a u-shaped cluster. (b) In detail, start and end point of another arbitrarily shaped cluster.

LUT is increased by 1. This process repeats until the start pixel is reached, indicating that all contour pixels of this particular cluster have been labeled and the image scan continues searching for the next unlabeled cluster.

During the contour tracing phase, FSM_1 outputs coordinates and control signals to FSM_2 for every contour pixel. FSM_2 is not part of the actual contour tracing but rather used to update and store the extracted features for every cluster. The double memory architecture, depicted in Figure 5.7, is a memory pipeline so when FSM_2 is updating one of the cluster memories, the tracking stage is given access to the other for an entire frame.

5.4 Results and performance

Based on the discussion in Section 5.1.2.1 and memory limitations in the implemented technology (FPGA), c_{max} is set to 61, plus three preoccupied labels, resulting in 6 bits per pixel, which is enough based on SW simulations. Hence, the labeling memory size is

$$mem_{size} = \lceil log_2(c_{max} + 3) \rceil \cdot (im_{height} \cdot im_{width}) = 6 \cdot (320 \cdot 240) \approx 450 \text{ kb}. \tag{5.5}$$

Furthermore, the resolution together with c_{max} also sets the dimensions of the cluster memory in which the properties for each cluster is stored, i.e. c_{max} determines the depth and the resolution determines the wordlength. A word in the cluster memory consists of $(x_{min}, y_{min}, cluster_{width}, cluster_{height})$ and the total size of the cluster memory is therefore equal to

$$\begin{aligned} mem_{size} = (c_{max} + 3) \cdot (&\lceil log_2(im_{width}) \rceil + \lceil log_2(im_{height}) \rceil \\ &+ \lceil log_2(im_{width}) \rceil + \lceil log_2(im_{height}) \rceil) \approx 2 \text{ kb}. \end{aligned} \tag{5.6}$$

Based on the discussion in Section 5.1.2.1 together with the specific requirements for this application, the total amount of memory in the implementation can be written as

$$mem_{tot} = FIFO_{input} + 2 \cdot label_{mem} + 2 \cdot cluster_{mem} \approx 1 \text{ Mb}. \tag{5.7}$$

The unit has been implemented and verified on a Xilinx Virtex II-pro FPGA at operating frequency up to 67 MHz, labeling up to 61 individual clusters together with extracting their coordinates with an image resolution of 320×240 and a frame rate of 25 fps.

5.4.1 Optimizations

A simple optimization that can be applied to any labeling algorithm that could, depending on the input, avoid unnecessary memory accesses is to monitor the input. As mentioned in Section 5.1.2, this can be achieved by marking the location of the first and last pixel equal to 1 as start and end pixel, as the input frame is being sequentially written into the label memory, illustrated in Figure 5.10. When the second scan starts, the first visited address is the start point and the last visited location is the end point. Depending on the input, this procedure can in many cases reduce memory accesses without additional complex hardware.

Chapter 6

System integration

This chapter presents a prototype of an automatic surveillance system implemented on a Virtex II Pro Development board [40]. The design strategy together with included peripherals are also presented. Details about the included blocks are found in the following sections: segmentation, Section 3.1; morphological filtering, Chapter 4; labeling of connected clusters, Chapter 5. The SW running on an embedded processor is intended to perform additional feature extraction and tracking, discussed in Section 3.4 and Section 3.5, respectively. As mentioned in Section 1.2 three PhD. students are involved each responsible for developing different parts of the system. The author's main contribution is the implementation of the morphological and labeling operations included in the system.

6.1 System Design Strategy

The goal of the project was to develop a prototype of an automated digital surveillance system on a development platform achieving real–time performance. The design strategy has been carried out in two phases: **top-down** and **bottom-up** modeling. First, a **top-down** approach was applied starting at the highest abstraction level (system level). A software model in C was used to distinguish required digital image processing blocks, e.g. morphology, labeling, etc, together with their timing constraints and corresponding memory requirements. The system was then partitioned into HW and SW. Software modeling is important to establish a system hierarchy especially since several people are involved in developing the same design. Using the software model together with the system hierarchy, test vectors to be used as input and reference vector to be compared with the output for each block, was produced. Furthermore, proceeding lower in abstraction level, for each block, the design space discussed in Section 1.1.2 was explored. The blocks were individually tested for functionality and verified with individual testbenches, using the software produced test and reference vectors. The individual blocks were synthesized to estimate speed and area. Synthesis is a design step that transforms the design into a netlist, i.e. transforming code into logic blocks and how these are connected. After synthesis, the **bottom-up** phase could start, i.e. design integration, forming the complete system. The interfaces between the blocks were adjusted and

the system was divided into different clock domains. Returning to system level, additional buffers, i.e. asynchronous FIFOs, to separate the various clock domains had to be inferred, e.g. between the segmentation and the morphology unit.

Synthesizing each block individually does not guarantee smooth synthesis of the complete system due to an increased FPGA resource utilization and the multiple clock domains. Furthermore, as the resource utilization increases, the physical routing also becomes more difficult, due to limited number of wires between the blocks. Therefore, much time was spent resolving Place and Route (P&R) errors. P&R is the last to step in the design flow physically mapping the netlist onto the available FPGA resources, and connecting the logic blocks.

6.2 XUP Virtex II Pro Development System

The XUP Virtex II Pro Development System is an advanced FPGA hardware platform provided by Xilinx. The system has support for various peripherals and standards, e.g. RS232, AC97, Ethernet, etc. This section will highlight and discuss the most important supported peripherals and properties for the implemented system.

The development system is based on a Xilinx Virtex II-pro XCV2VP30 FPGA which is equipped with 2448 kb of on-chip block Random Access Memory (RAM) and has eight Digital Clock Managers (DCM) enabling eight independent clock domains. It has 13.6 k slices and 136 embedded multipliers. Furthermore, the FPGA has two integrated PowerPCs (PPC) as hard on-chip IP-blocks, on which the software is running on one of them.

The platform has an onboard 64 bit, 184-pin, Double Data Rate (DDR) Synchronous Dynamic Random Access Memory (SDRAM) slot, in which the current implementation has incorporated a 256 MB memory module. Furthermore, the system has four 60-pin I/O headers used to connect the image sensor to the FPGA. The system provides a video Digital to Analog Converter (DAC) with corresponding VGA connector (15-pin D-sub). The video DAC can operate at various resolutions and refresh rates but the current implementation is running in 640×480 at 60 Hz using a smaller active screen area. A photograph of the platform including some peripherals can be seen in Figure 6.1.

6.3 Kodak KAC-9648 CMOS Image Sensor

Based on the comparison between different sensor techniques in Section 2.1.1 and the system constraints taken from Section 1.1.1, the Kodak KAC-9648 CMOS image sensor was chosen for image capturing. It is a low power and multipurpose sensor for still image and video sequence capturing. An important sensor property for the prototype is the integrated compensation circuitry, e.g. black level compensation and adjustable parameter settings for each color channel, required to optimize color reconstruction. Black level compensation is used to remove possible natural offset, i.e. black should be represented with the value 0. The sensor supports various modifications but the most important to the prototype are the integration time and separate gain for each color. Typically, the gain has to be increased for green and even more for blue to produce a more uniform color sensitivity spectrum. Changing these settings increases

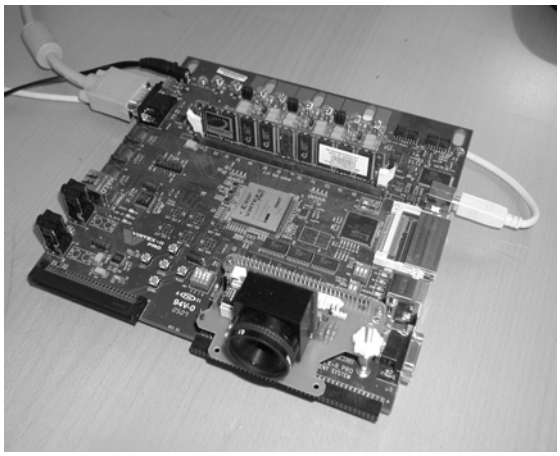


Figure 6.1: The XUP Virtex II Pro Development System, including the senor and the DDR memory module.

the color dynamic range. A high dynamic range is required by the segmentation unit to achieve a good segmentation result. Integration time is equal to the shutter time in a conventional camera and affects the image brightness. It proved necessary to decrease the integration time since the default setting saturated the sensor in a typical indoor environment, resulting in poor segmentation results.

In the current system, the sensor delivers a video sequence with a resolution of (320×240) as an 8 bit data stream (one color at the time, four colors per pixel), at a frequency f of 7.68 MHz (1.92 MHz RGB). Hence, the frame rate is calculated according to

$$\text{Frame rate} = \frac{f}{4 \cdot (im_{height} \times im_{width})} = \frac{7.68 \cdot 10^6}{4 \cdot (320 \times\ 240)} = 25 \text{ fps.} \tag{6.1}$$

6.4 Prototype – An Intelligent Surveillance System

An architectural overview of the system prototype is depicted in Figure 6.2. The sensor output consists of a 1.92 MHz continuous, 24-bit wide RGB pixel stream that is fed to both the segmentation unit and the VGA controller. The segmentation unit outputs a binary mask as a pixel stream at 100 MHz, connected to an asynchronous FIFO placed between the segmentation and the morphology unit in order to separate the different clock domains. Since the throughput of the morphology unit does not need to be higher than that of the labeling unit, the lower operating frequency of the two units is chosen as common clock domain, i.e. 67 MHz, which is the maximum frequency of the labeling unit. Furthermore, the morphology unit outputs a pixel stream at 67 MHz to the labeling unit requiring a synchronous FIFO to be placed at the input of the labeling unit. This is done to guarantee that no pixels are lost when stalling the pixel stream during the contour tracing phase in the labeling algorithm. A simple interface is placed

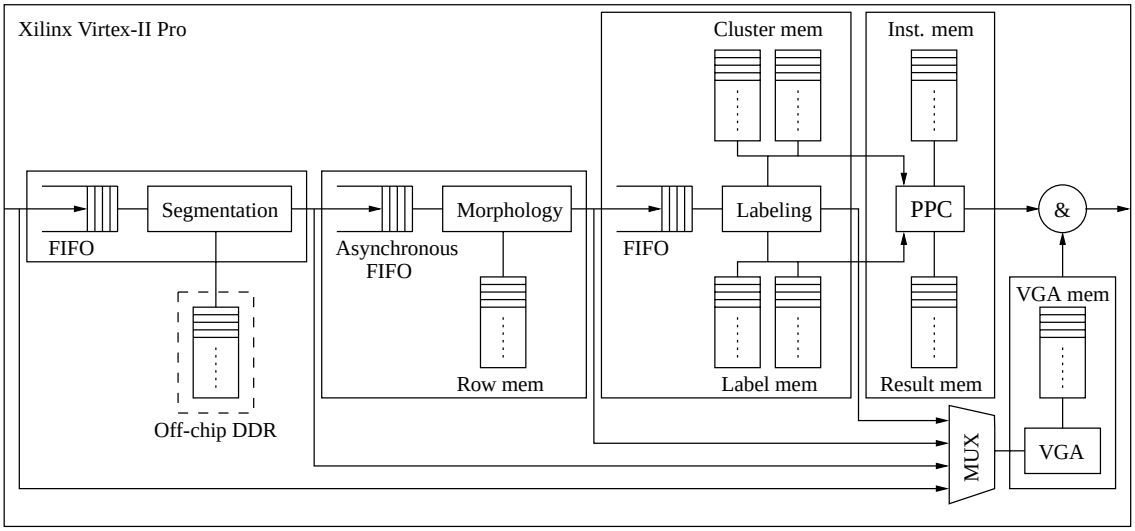


Figure 6.2: An overview of the complete implemented system.

between the labeling unit and the PPC which gives the processor access to one of the cluster memories together with the corresponding label memory. Using the extracted features in the cluster memories together with the complete labeled image stored in the label memory, the software is able to cut out the cluster areas from the original sequence. In the current system version, the software only displays a box around every cluster but future versions will support more advanced feature extraction, tracking and classification.

A VGA controller outputs synchronization signals together with image data to an external monitor. The controller requires a memory which is proportional to the resolution, i.e. number of colors N_c and number of bits per color C_b. In the prototype, this memory is implemented as dual port memory, constantly writing to one port, i.e. the system results, and constantly reading from the other, i.e. VGA output. Displaying the original sequence in full color would require

$$Dual\_mem_{size} = N_c \cdot C_b \cdot (im_{height} \times im_{width}) = 3 \cdot 8 \cdot (320 \times 240) \approx 1,84 \text{ Mb}, \quad (6.2)$$

which is about 75 % of the memory budget in the FPGA. Therefore, the current prototype only displays the original video as 8 bit gray scale.

The system output of the current prototype is only sent to the VGA controller and displayed as a box around each cluster. In future versions of the prototype, the output can be directed to a log-file or a mass storage unit, e.g. a hard disc. This could be done by only storing the features of each object or together with a compressed image sequence of the clusters.

6.5 Optimizations

There are two major memory bottlenecks in the current system: the bandwidth from the segmentation unit to the off-chip memory and high on-chip block ram utilization ratio (97,7%). The bandwidth requirement in the segmentation comes from the large number of parameters that need to be updated each clock cycle. The current segmentation unit implementation uses three distributions per pixel, each requiring 16 bytes, that need to be updated, i.e. read from and written to the memory, 25 times per second. This together with a system resolution of 320×240 results in a bandwidth of

$$BW = 2 \cdot 3 \cdot 16 \cdot (320 \times \ 240) \cdot 25 \approx 180 \text{ MBps}. \tag{6.3}$$

In an effort to reduce this bandwidth, a memory reduction scheme is currently being investigated which reports up to 60% memory access savings in simulation.

An obvious solution to the high on-chip block RAM utilization is to move the dual port memory required by the VGA controller off-chip, which would free 76 KB of memory. Currently, a PCB with a 32×256 asynchronous Static Random Access Memory (SRAM) is being developed. This memory will not only enable the 24-bit full color video sequence from the sensor to be displayed by the VGA controller but also give the possibility to store the correct image from which the features are extracted by the PPC, which currently has an offset of three frames.

There are several other properties that improves the system performance, e.g. increasing the system resolution, adding tracking and classification SW. Increasing the resolution is of special interest since it would further increase the advantage of a HW implementation compared to other SW implementations running on general purpose processors. Adding SW for tracking and classification is the next natural step to further develop the system. The current system is using only 10% of the dedicated program memory, which allows the SW program to increase in size, which is currently being investigated. Furthermore, freeing memory resources will also enable placing future feature extraction accelerators in HW. Another important issue is displaying the original video in full color. This would not directly increase system performance but is an appealing feature for demonstration purposes and can be important in a commercial product.

6.6 Results

An automated digital surveillance system has been developed using a XUP Virtex II Pro Development System platform. A high performance CMOS sensor, i.e. Kodak KAC-9648, is used for image acquisition. The implemented prototype has the following specification:

- **Resolution:** 320×240.
- **Frame rate:** 25 fps.
- **Segmentation:** 3 distributions per pixel.
- **Morphological operation:** opening.

Table 6.1: A summation of the system resource utilization of a Xilinx Virtex II-pro XCV2VP30-7ff896 FPGA.

Logic utilization	used	available	ratio (%)
Slices	6,710	13,696	48
PPC405s	1	2	50
Block RAM	133	136	97,7
DCM	4	8	50

- **Labeling:** Labels 61 independent clusters together with extracting their maximum and minimum coordinates.
- **Tracking:** Currently supports drawing a bounding box around every cluster, which is based on the extracted coordinates.
- **VGA output:** Supports four output modes: the original video (8 bits gray scale), the segmentation result, morphological result, and the labeling unit. The bounding box generated in SW can be shown on top of any of these output modes.

A summation of the FPGA resource utilization is found in Table 6.1. The number of used slices can still be seen as relatively low, enabling additional accelerators to be placed in HW. However, the block RAM utilization is 97,7% (maximum). In order to reduce this figure, several approaches were considered, i.e. translating block RAMs into distributed RAMs, or connecting multiple FPGAs; both rejected since neither are long term solutions. Moving blocks off-chip, e.g. the VGA-memory, thus freeing memory resources, was considered the only solid solution to this limitation and is therefore of outmost importance for future system development even though it results in increased power dissipation.

Examples of the various output modes supported by the current system implementation are depicted in Figure 6.3, excluding the original gray scale video. Each of the four modes can be drawn with or without a bounding box.

Comparing the prototype to other system implementations, an equivalent system implemented in SW running on a desktop PC (Pentium-4 1700MHz) achieves 3 fps with a resolution of 352×288 [41]. In this system, the setup is somewhat different making the comparison not directly proportional, e.g. the image stream is JPEG compressed and captured with a network camera delivered to the system via a 10Mbs Local Area Network (LAN) [42]. Increasing the computational capacity of the PC and pixel rate from the camera together with removing the JPEG decompression would certainly increase system performance. However, in our application, i.e. a self contained network camera, a desktop solution is not an issue not only due to the physical size limitations but also cost. Furthermore, as the resolution of the sensors is continuously increasing, resulting in an even higher bandwidth in future systems, accelerating key operations in hardware in an automated surveillance system is crucial to achieve real–time performance.

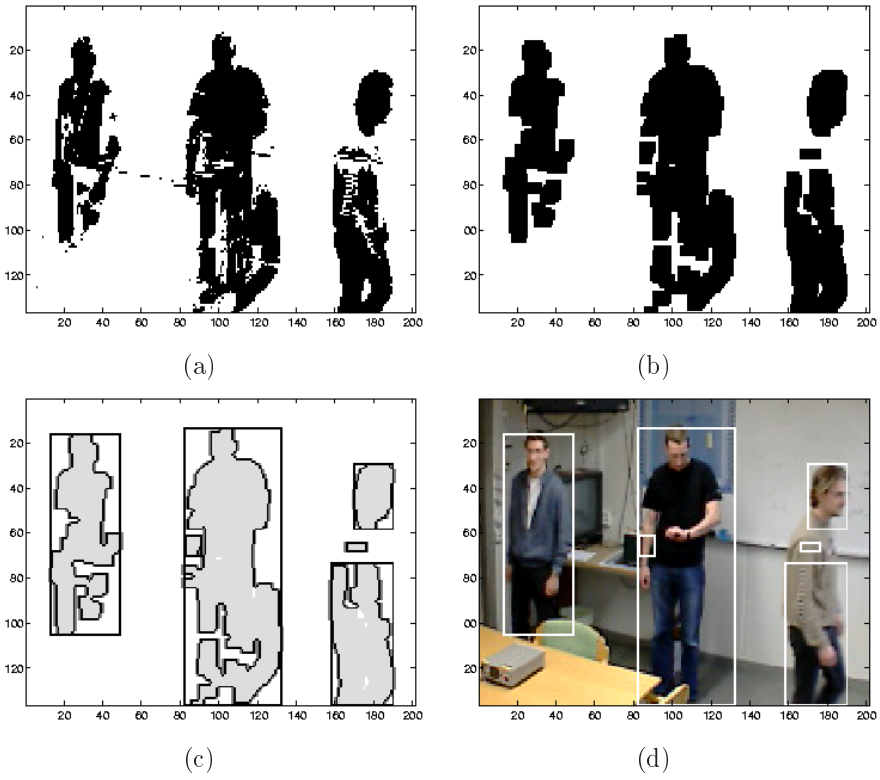


Figure 6.3: Typical results from the different units that can be displayed as VGA output, e.g. on an external monitor, (a) segmentation result, (b) the output from the morphological unit after an opening has been performed, (c) labeled output, (d) original video. The bounding boxes shown in (c) and (d) are generated from the PPC and can be applied to any of the outputs.

Chapter 7

Conclusions

In this thesis, implementation aspects of an automated digital surveillance system together with details of two hardware accelerators used in the system are presented. A software implementation of the system is used to identify computational, bandwidth and memory requirements. To be able to handle the bandwidth that a surveillance system requires, the design strategy has been to accelerate key operations in hardware and extract features at different abstraction levels of the system. This strategy has proven to efficiently reduce the amount of data processed in SW, resulting in real–time performance, i.e. 25 fps.

A low complexity hardware accelerator for binary morphological erosion and dilation is presented. The low complexity architecture uses SE decomposition, reducing memory requirements to a memory size proportional to the image width and SE height. The design allows instantiations of multiple morphological units in series enabling more advanced morphological operations based on erosion and dilation, i.e. opening, closing. The architecture supports arbitrary sized rectangular SEs that can be changed during run-time. Integrating the architecture in the surveillance system, efficiently reduce noise in the binary motion mask. The architecture has been successfully tested and verified at 100 MHz on a Xilinx Virtex-II pro FPGA.

The implementation of a hardware accelerator for cluster labeling using a contour tracing technique is presented. The contour tracing algorithm requires less memory to guarantee labeling of a specific number of clusters. This combined with the ability to add Green's formula to calculate the moments, needed to extract important features, i.e. COG and size, still makes the contour tracing algorithm advantageous. Especially when the implementation is intended for use in an automated surveillance system. The memory requirements are proportional to the image size and the maximum number of individual clusters that can be labeled in a frame. The design extracts maximum and minimum coordinates and is running at 67 MHz on a Xilinx Virtex-II pro FPGA.

Finally, a prototype of a simplified system has been implemented and verified on an XUP Xilinx Virtex-II PRO FPGA development system board. By accelerating key operations in hardware, the system achieves a resolution of 320×240 and a frame rate of 25 fps.

Chapter 8

Future work

An obvious future direction of the project is to make an effort to increase image resolution while maintaining a frame rate of $\geq$ 25 fps. The resolution is currently 320×240, which is low compared to modern digital surveillance applications and future system revisions should reach at least 640×480. The major problem when increasing the resolution is the increased memory requirement and bandwidth bottlenecks. Migrating to a larger FPGA, e.g. Xilinx Virtex-4, containing more on-chip memory and more I/O connections allowing more off-chip memories would certainly solve many current problems but might not be of scientific interest. A more scientific approach in order to manage and restrain these limitations is to investigate memory optimization on the system level, e.g. letting the accelerators share the bandwidth to one large off-chip memory. Another strategy can be to infer different computational trade-offs in the algorithms which can prove necessary since future sensor resolutions are increasing in parallel with the FPGA resources. As an example, investigation of various lossless compression schemes, e.g. Run Length Encoding, is currently in progress, especially in the segmentation and labeling blocks.

Regarding the morphology unit, several future modifications are currently being investigated. Expanding the morphology architecture to support arbitrary sized SEs containing ones and zeros and expanding the unit to apply to gray scale images would certainly increase the applicability of the architecture to other image processing applications but the enhanced effect in this particular project would probably be minor.

Future labeling unit versions will support extraction of more features, e.g. size, COG. This is achieved by while tracing the contour of a cluster, adding a discrete version of Green formula for binary images and thereby calculating the moments. Using the first and second moment of a cluster, size and COG can be easily calculated.

The search for simple, robust and stationary features is continuous process. Extracting such features increases the system performance, i.e. prediction accuracy, especially when handling occlusions and performing tracking. Accelerating these features in hardware will decrease the workload for the PPC; keeping the amount of data processed in SW at a minimum. Future high resolution sensors and robust features allows additional system features such as face detection/recognition. Adding such a system feature makes the system more attractive to other applications than automated surveillance, e.g. domestic, industrial, etc. Imagine a game of football in which every player together

with the ball is tracked; a face recognition system used as pin-lock on a mobile phone. Only imagination sets the limits of the applicability of an automated surveillance system which supports face recognition.

Bibliography

[1] L. Råde and B. Westergren, *Mathematics Handbook for Science and Engineering*, 2nd ed. Lund, Sweden: Studentlitteratur, 1998.

[2] R. Gonzalez, R. Woods, and S. L. Eddins, *Digital Image Processing using Matlab*. Upper Saddle River, NJ, USA: Prentice Hall, inc, 2004.

[3] J. M. Rabaey, A. Chandrakasan, and B. Nikolić, *Digital Integrated Circuit*, 2nd ed. Upper Saddle River, NJ, USA: Prentice Hall, inc, 2003.

[4] B. Parhami, *Computer Arithmetic, Algorithms and Hardware Designs*. New York, NY, USA: Oxford University Press, 2000.

[5] K. K. Parhi, *VLSI Digital Signal Processing Systems*. New York, NY, USA: John Wiley & Sons, 1999.

[6] H. Jiang, H. Ardö, and V. Öwall, "Hardware accelerator design for video segmentation with multi-modal background modelling," in *Proc. of IEEE International Symposium on Circuits and Systems*, Kobe, Japan, May 23-26 2005.

[7] F. Kristensen, P. Nilsson, and V. Öwall, "Background segmentation beyond RGB," in *Seventh biennial Asian Conference on Computer Vision*, Hyderabad, India, Jan. 13-16 2006.

[8] AXIS Communications AB, www.axis.com, 2005.

[9] J. Jönsson, *Våglära och Optik*, 2nd ed. Lund, Sweden: Teach Support, 1995.

[10] B. E. Bayer, "Color imaging array," U.S. Patent 3,971,065, 1976.

[11] C. L. Hardin, *Color for Philosophers*. Indianapolis, IN, USA: Hackett Publishing Company, 1988.

[12] D. Litwiller, "CMOS vs. CCD: Maturing technologies, maturing markets," *Photonics Spectra*, aug 2005.

[13] ——, "CMOS vs. CCD: Facts and fiction," *Photonics Spectra*, jan 2001.

[14] R. Gonzalez and R. Woods, *Digital Image Processing*, 2nd ed. Upper Saddle River, NJ, USA: Prentice Hall, inc, 2002.

[15] A. N. Netravali and B. G. Haskell, *Digital Pictures : Representation, Compression, and Standards*, 2nd ed. New York, NY, USA: Plenum, 1994.

[16] C. Stauffer and W. E. L. Grimson, "Adaptive background mixture models for real-time tracking," in *Proc. of IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, Ft. Collins, CO, USA, June 23-25 1999.

[17] T. Gevers and A. Smeulders, "Color-based object recognition," *Pattern recognition, the Journal of the pattern recognition society*, vol. 32, pp. 453–464, 1999.

[18] J. Serra, *Image Analysis and Mathematical Morpohology*. New York, NY, USA: Academic Press, 1982.

[19] S. Fejes and F. Vajda, "A data-driven algorithm and systolic architecture for image morphology," in *Proc. of IEEE Image Processing*, Austin, Texas, Nov. 13-16 1994, pp. 550–554.

[20] T. Q. Deng and H. J. A. M. Heijmans, "Grey-scale morphology based on fuzzy logic," *Journal of Mathematical Imaging and Vision*, vol. 16, no. 2, pp. 155–171, mar 2002.

[21] G. Louverdis and I. Andreanis, "Design and impelementation of a fuzzy hardware structure for morphological color image processing," *IEEE Transaction on Circuits and Systems for Video Technology*, vol. 13, no. 3, pp. 277–288, 2003.

[22] H. Hedberg, F. Kristensen, P. Nilsson, and V. Öwall, "A low complexity architecture for binary image erosion and dilation using structuring element decomposition," in *Proc. of IEEE International Symposium on Circuits and Systems*, Kobe, Japan, May 23-26 2005.

[23] V. Öwall, M. Torkelson, and P. Egelberg, "A custom image convolution DSP with a sustained calculation capacity of >1 GMAC/s and low I/O bandwidth," *Journal of VLSI Signal Processing*, vol. 23, no. 2-3, pp. 335–349, nov 1999.

[24] H. Jiang and V. Öwall, "FPGA implementation of real-time image convolutions with three level of memory hierarchy," in *Proc. of International Conference on Field-programmable Technology*, Tokyo, Japan, Dec. 15-17 2003.

[25] H. Park and R. Chin, "Decomposition of arbitrarily shaped morphological structuring elements," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 17, pp. 2–15, 1995.

[26] G. Anelli and A. Broggi, "Decomposition of arbitrarily shaped binary morphological structuring elements using genetic algorithms," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 20, pp. 217–224, 1998.

[27] J. Goutsias and H. J. Heijmans, "Fundamenta morphologicae mathematicae," *Fundamenta Informaticae*, vol. 41, pp. 1–31, 2000.

[28] J. Velten and A. Kummert, "FPGA-based implementation of variable sized structuring elements for 2D binary morphological operations," in *Proc. of IEEE International Symposium on Circuits and Systems*, Bangkok, Thailand, Mar. 2003, pp. 706–709.

[29] E. N. Malamas, A. G. Malamos, and T. A. Varvarigou, "Fast implementation of binary morphological operations on hardware-efficient systolic architectures," *Journal of VLSI Signal Processing*, vol. 25, pp. 79–93, 2000.

[30] A. Rosenfeld and J. Pfaltz, "Sequential operations in digital picture processing," *Journal of the Association for Computing Machinery*, vol. 13, no. 1, pp. 471–494, Oct 1966.

[31] K. Suzuki, I. Horiba, and N. Sugie, "Fast connected-component labeling based on sequential local operarations in the course of forward raster scan followed by backward raster scan," in *Proc. of 15th International Conference on Pattern Recognition*, Barcelona, Spain, Sept. 3-7 2000, pp. 434–437.

[32] L. D. Stefano and A. Bulgarelli, "A simple and efficeient connected components labeling algorithm," in *Proc. of 10th International Conference on Image Analysis and Processing*, Venice, Italy, Sept. 27-29 1999.

[33] S. D. Jean, C. M. Liu, C. C. Chang, and Z. Chen, "A new algorithm and its VLSI architecture design for connected component labeling," in *Proc. of IEEE International Symposium on Circuits and Systems*, London, England, Uk, May 30-June 2 1994.

[34] W. Kesheng, O. Ekow, and S. Arie, "Optimizing connected components labeling algorithms," in *SPIE International Symposium on Medical Imaging*, San Diego, CA, USA, Feb. 12-17 2005.

[35] M. B. Dillencourt, H. Samet, and M. Tamminen, "A general approach to connected-component labeling for aritrary image representations," *Journal of the Association for Computing Machinery*, vol. 39, no. 2, pp. 253–280, Apr 1992.

[36] K. Suzuki, H. Isao, and S. Noboru, "Linear-time connected-component labeling based on sequential local operations," *Journal of Computer Vision and Image Understanding*, vol. 89, pp. 1–23, jan 2003.

[37] F. Chang, C. J. Chen, and C. J. Lu, "A linear-time component-labeling algorithm using contour tracing technique," *Journal of Computer Vision and Image Understanding*, vol. 93, pp. 206–220, feb 2004.

[38] F. Chang and C. J. Chen, "A component-labeling algorithm using contour tracing technique," in *Proc. of 7th International Conference on Document Analysis and Recognition*, Edinburgh, Scotland, Uk, Aug. 3-7 2003.

[39] L. Yang and A. F., "Discrete Green's theorem and its application in moment computation," in *Int. Conf. Electronics and Information Technology*, Beijing, China, aug 1994, pp. 27–31.

[40] F. Kristensen, H. Hedberg, H. Jiang, P. Nilsson, and V. Öwall, "Hardware aspects of a real-time surveillance system," in *Sixth IEEE International Workshop on Visual Surveillance*, Graz, Austria, May 13 2006, accepted for publication.

[41] K. Wittenmark, *Motion Segmentation, Object Tracking, and Event Detection in Image Sequences.* Master's thesis, Lund Institute of Technology, feb 2004.

[42] G. K. Wallace, "The JPEG still picture compression standard," *IEEE Transactions on Consumer Electronics*, vol. 38, no. 1, pp. xviii–xxxiv, feb 1992.