Design Issues in VLSI Implementation of Image Processing Hardware Accelerators

Methodology and Implementation

Hongtu Jiang

Lund 2007

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

This thesis is set in Computer Modern 10pt,
with the LaTeX Documentation System
on 100gr Colortech+™ paper.

No. 66
ISSN 1402-8662

Abstract

With the increasing capacity in today's hardware systems enabled by technology scaling, image processing algorithms with substantially higher complexity can be implemented on a single chip enabling real-time performance. Combined with the demand for low power consumption or larger resolution seen in many applications such as mobile devices and HDTV, new design methodologies and hardware architectures are constantly called for to bridge the gap between designers productivity and what the technology can offer.

This thesis tries to address several issues commonly encountered in the implementations of real-time image processing system designs. Two implementations are presented to focus on different design issues in hardware design for image processing systems.

In the first part, a real-time video surveillance system is presented by combining five papers. The segmentation unit is part of a real-time automated video surveillance system developed at the department, aiming for tracking people in an indoor environment. Alternative segmentation algorithms are elaborated, and various modifications to the selected segmentation algorithm is made aiming for potential hardware efficiency. In order to bridge the memory bandwidth issue which is identified as the bottleneck of the segmentation unit, combined memory bandwidth reduction schemes with pixel locality and wordlength reduction are utilized, resulting in an over 70% memory bandwidth reduction. Together with morphology, labeling and tracking units developed by two other Ph.D. students, the whole surveillance system is prototyped on an Xilinx VirtexII pro VP30 FPGA, with a real-time performance at a frame rate of 25 fps with a resolution of 320×240.

For the second part, two papers are extended to discuss issues of a controller design and the implementation of control intensive algorithms. To avoid tedious and error prone procedure of hand coding FSMs in VHDL, a controller synthesis tool is modified to automate a controller design flow from C-like control algorithm specification to controller implementation in VHDL. To address issues of memory bandwidth as well as power consumptions, a three level of memory hierarchy is implemented, resulting in off-chip memory bandwidth reduction from N^2 per clock cycle to only 1 per pixel operation. Furthermore, potential power consumption reduction of over 2.5 times can be obtained with the architecture. Together with a controller synthesized from the developed tool, a real-time image convolution system is implemented on an Xilinx VirtexE FPGA platform.

Contents

Preface

This thesis summarizes my academic work in the digital ASIC group at the department of Electroscience, Lund University, for the Ph.D. degree in circuit design. The main contribution to the thesis is derived from the following publications:

H. Jiang and V. Öwall, "FPGA Implementation of Real-time Image Convolutions with Three Level of Memory Hierarchy," in *IEEE conference on Field Programmable Technology (ICFPT)*, Tokyo, Japan, 2003.

H. Jiang and V. Öwall, "FPGA Implementation of Controller-Datapath Pair in Custom Image Processor Design," in *IEEE International Symposium on Circuits and Systems (ISCAS)*, Vancouver, Canada, 2004.

H. Jiang, H. Ardö and V. Öwall, "Hardware Accelerator Design for Video Segmentation with Multi-modal Background Modeling," in *IEEE International Symposium on Circuits and Systems (ISCAS)*, Kobe, Japan, 2005.

F. Kristensen, H. Hedberg, H. Jiang, P. Nilsson and V. Öwall, "Hardware Aspects of a Real-Time Surveillance System," in *European Conference on Computer Vision, Graz, (ECCV)*, Graz, Austria, 2006.

H. Jiang, H. Ardö and V. Öwall, "Real-time Video Segmentation with VGA Resolution and Memory Bandwidth Reduction," in *IEEE International Conference on Advanced Video and Signal based Surveillance (AVSS)*, Sydney, Australia, 2006.

H. Jiang, H. Ardö and V. Öwall, "VLSI Architecture for a Video Segmentation Embedded System with Algorithm Optimization and Low Memory Bandwidth," To be Submitted to *IEEE Transactions on Circuits and Systems for Video Technology*, February, 2007.

F. Kristensen, H. Hedberg, H. Jiang, P. Nilsson and V. Öwall, "Working title: Hardware Aspects of a Real-Time Surveillance System," To be submitted to *Springer Journal of VLSI Signal Processing Systems for Signal, Image, and Video Technology*, February, 2007.

Acknowledgment

First of all, I would like to thank my supervisor, Viktor Öwall, for all the help and encouragement during all these years of my graduate study. His knowledge, inspiration and efforts to explain things clearly has a deep impact on my research work in Digital IC design. I can not imagine the completion of my thesis work without his help. I would also like to thank him for his eternal attempts to get me ambitious, and his strong intentions to get me addictive to sushi and beer when we were traveling in Canada and Japan.

I would also like to thank Thomas, for all the fruitful discussions that were of great help to the project developments. I learned lots of practical design techniques during our discussions. I would also like to thank Anders for the suggestions on my project work when I started here. Many thanks to Zhan, who gave me inspirations and deep insights into Digital IC design and many other things.

I am also grateful to Fredrik for reading parts of the thesis, and Matthias, Hugo, Joachim, Erik ,Henrik, Johan, Deepak, Martin, Peter for the many interesting conversations and help. I really enjoy working in the group.

I would like to extend my gratitude to the colleagues and friends at the department. I would like to thank Jiren for introducing me here, Kittichai for all the enjoyable conversations and help, Erik for helping me with the computers and all kinds of relevant or irrelevant interesting topics, Pia, Elsbieta and Lars for their many assistance.

I would also like to thank Vinnova Competence Center for Circuit Design (CCCD) for financing the projects, Xilinx for donating FPGA boards and Axis for their network camera and expertise on image processing. Thanks to Department of Mathematics for their input on the video surveillance project, especially Håkan Ardö who gave me many precious advices.

Finally, I would like to thank my parents, my sister and my nephew, who support me all the time, and my wife Alisa and our daughter Qinqin, who bring me love and lots of joy.

List of Acronyms

ADC	Analog-to-Digital Converter
ASIC	Application-Specific Integrated Circuit
Bps	Bytes per second
CCD	Charge Coupled Device
CCTV	Closed Circuit Television
CFA	Color Filter Array
CIF	Common Intermediate Format
CMOS	Complementary Metal Oxide Semiconductor
CORDIC	Coordinate Rotation Digital Computer
DAC	Digital-to-Analog Converter
DCM	Digital Clock Manager
DCT	Discrete Cosine Transform
DWT	Discrete Wavelet Transform
DDR	Double Data Rate
DSP	Digital Signal Processor
FD	Frame Difference
FIFO	First In, First Out
FIR	Finite Impulse Response
FPGA	Field Programmable Gate Array
fps	frames per second
FSM	Finite State Machine
HDL	Hardware Description Language
IIR	Infinite Impulse Response

IP	Intellectual Property
kbps	kilo bits per second
KDE	Kernel Density Estimation
LPF	Linear Predictive Filter
LSB	Least Significant Bit
LUT	Lookup Table
Mbps	Mega bits per second
MCU	Micro-Controller Unit
MoG	Mixture of Gaussian
MSB	Most Significant Bit
PPC	Power PC
P&R	Place and Route
RAM	Random-Access Memory
RISC	Reduced Instruction Set Computer
PCB	Printed Circuit Board
ROM	Read-Only Memory
SRAM	Static Random-Access Memory
SDRAM	Synchronous Dynamic Random-Access Memory
SoC	System on Chip
VGA	Video Graphics Array
VHDL	Very High-level Design Language
VLSI	Very Large-Scale Integration

General Introduction

Chapter 1

Overview

Imaging and video applications are one of the fastest growing sectors of the market today. Typical application areas include e.g. medical imaging, HDTV, digital cameras, set-top boxes, machine vision and security surveillance. As the evolution in these applications progresses, the demands for technology innovations tend to grow rapidly over the years. Driven by the consumer electronics market, new emerging standards along with increasing requirements on system performance imposes great challenges on today's imaging and video product development. To meet with the constantly improved system performance measured in, e.g., resolution, throughput, robustness, power consumption and digital convergence (where a wide range of terminal devices must process multimedia data streams including video, audio, GPS, cellular, etc.), new design methodologies and hardware accelerator architectures are constantly called for in the hardware implementation of such systems with real-time processing power. This thesis tries to deal with several design issues normally encountered in hardware implementations of such image processing systems.

1.1 Thesis Contributions

In this thesis, implementation issues are elaborated regarding transforming image processing algorithms into hardware realizations in an efficient way. With the major concern to address memory bottlenecks which are common to most image applications, architectural considerations as well as design methodology constitute the main scope of the thesis research work. Two implementations are contributed in the thesis for the design of image processing accelerators:

In the first implementation, a real-time video segmentation unit is implemented on an Xilinx FPGA platform. The segmentation unit is part of a real-time embedded video surveillance system developed at the department, which are aimed to track people in an indoor environment. Alternative segmentation algorithms are elaborated, and an algorithm with Mixture of Gaussian approach is selected based on the trade-offs of segmentation quality and computational complexity. For hardware implementation, memory bottlenecks are addressed with combined memory bandwidth reduction schemes. Modifications to the original video segmentation are made to increase hardware efficiency.

In the second implementation, a synthesis tool is modified to automate a controller design flow from a control algorithm specification to VHDL implementation. The modified tool is utilized in the implementation of a real-time image convolution accelerator, which is prototyped on an Xilinx FPGA. An architecture of three levels of memory hierarchy is developed in the image convolution accelerator to address issues of memory bandwidth and power consumption.

1.2 Thesis Outline

The thesis is structured into three parts. The introduction part covers topics concerning a range of technologies used in the hardware implementation of a typical image processing systems, e.g. image sensors, signal processing units, memory technologies and displays. Comparisons are made in various technologies regarding performance, area and power consumption cost etc. Following the introduction are two parts covering implementations by the author with the aim of different design goals.

Part I

A design and implementation of a real-time video surveillance system is presented in this part. Details on video segmentation implementation from algorithm evaluation to the architecture and hardware design are elaborated. Novel ideas of how off-chip memory bandwidth can be reduced by utilizing pixel locality and wordlength reduction scheme are shown. Modifications to the existing Mixture of Gaussian (MoG) [1] is proposed aiming for potential hardware efficiency. The implementation of the segmentation unit is based on:

> H. Jiang, H. Ardö and V. Öwall, "Hardware Accelerator Design for Video Segmentation with Multi-modal Background Modeling," in *IEEE International Symposium on Circuits and Systems (ISCAS)*, Kobe, Japan, 2005.
>
> H. Jiang, H. Ardö and V. Öwall, "Real-time Video Segmentation with

VGA Resolution and Memory Bandwidth Reduction," in *IEEE International Conference on Advanced Video and Signal based Surveillance (AVSS)*, Sydney, Australia, 2006.

H. Jiang, H. Ardö and V. Öwall, "VLSI Architecture for a Video Segmentation Embedded System with Algorithm Optimization and Low Memory Bandwidth,"To be Submitted to *IEEE Transactions on Circuits and Systems for Video Technology*, February, 2007.

The second chapter of the part is dedicated to system integration of the segmentation into a complete tracking system, which includes segmentation, morphology, labeling and tracking. The complete system is implemented on an Xilinx VirtexII pro FPGA platform with real-time performance at a resolution of 320×240. The implementation of the complete embedded tracking system is based on:

F. Kristensen, H. Hedberg, H. Jiang, P. Nilsson and V. Öwall, "Hardware Aspects of a Real-Time Surveillance System," in *European Conference on Computer Vision, Graz, (ECCV)*, Graz, Austria, 2006.

F. Kristensen, H. Hedberg, H. Jiang, P. Nilsson and V. Öwall, "Working title: Hardware Aspects of a Real-Time Surveillance System," To be submitted to *Springer Journal of VLSI Signal Processing Systems for Signal, Image, and Video Technology*, February, 2007.

The author's contribution is on the segmentation part.

Part II

Controller design automation with a modified controller synthesis tool is discussed in the procedure of implementing control intensive image processing systems. For signal processing systems with increasing complexity, hand coding FSMs in VHDL becomes a tedious and error prone task. To bridge the difficulty of implementing and verification of complicated FSMs, a controller synthesis tool is needed. In this part, various aspects of FSM structures and implementations are explored. Details on design flows with the developed tool are presented. In the second chapter, the controller synthesizer is applied on the implementation of a real-time image convolution hardware accelerator. In addition, an architecture of three levels of memory hierarchy is developed in the image convolution hardware. It is shown how power consumption as well as memory bandwidth can be saved by utilizing memory hierarchies. Such architecture can be generalized to implementing different image processing functions like morphology, DCT or other block based sliding window filtering operations.

In the implementation, power consumption due to memory operations are reduced by over 2.5 times, and the off-chip memory access is reduced from N^2 per clock to only one pixel per operations, where N is the size of the sliding window. The whole part is based on:

> H. Jiang and V. Öwall, "FPGA Implementation of Controller-Datapath Pair in Custom Image Processor Design," in *IEEE International Symposium on Circuits and Systems (ISCAS)*, Vancouver, Canada, 2004.
>
> H. Jiang and V. Öwall, "FPGA Implementation of Real-time Image Convolutions with Three Level of Memory Hierarchy," in *IEEE Conference on Field Programmable Technology (ICFPT)*, Tokyo, Japan, 2003.

Chapter 2

Hardware Implementation Technologies

The construction of a typical real-time imaging or video embedded system is usually an integration of a range of electronic devices, e.g. image acquisition device, signal processing units, memories, and a display. Driven by the market demand to have faster, smarter, smaller and more interconnected products, designers are under greater pressure to make decisions on selecting the appropriate technologies in each one of the devices among many of the alternatives. Trade-offs are constantly made concerning e.g. cost, speed, power, and configurability. In this chapter, a brief overview of the varied alternative technologies is given along with elaborations on the plus and minus sides of each of the technologies, which motivates the decisions made on the selection of the right architecture for each of the devices used in the projects.

2.1 ASIC vs. FPGA

The core devices of an real-time embedded system are composed of one or several signal processing units implemented with different technologies such as Micro-controller units (MCUs), Application Specific Signal Processors(ASSPs), General Purpose Processors (GPPs/RISCs), Field Programmable Gate Arrays (FPGAs) and Application Specific Integrated Circuits (ASICs). A comparison is made for the areas where each of these technologies prevails [2], which is a bit biased to DSPs. This is shown in Table 2.1. No perfect technology exists that is competent in all areas. For a balanced embedded system design, a combination of some of the alternative technologies is a necessity. In general, an embedded system design is initiated with Hardware/Software partitioning, once the original specifications are settled under various system requirements.

Table 2.1: Comparisons of different types of signal processing units. Sources are from [2].

	Performance	**Price**	**Power**	**Flexibility**	**Time to market**
ASIC	Excellent	Excellent [1]	Good	Poor	Fair
FPGA	Excellent	Poor	Fair	Excellent	Good
DSP	Excellent	Excellent	Excellent	Excellent	Good
RISC	Good	Fair	Fair	Excellent	Excellent
MCU	Fair	Excellent	Fair	Excellent	Excellent

The partitioning is carried out by either a heuristic approach or by a certain kind of optimization algorithm, e.g. simulated annealing [3] or tabu search [4]. Software is executed in processors (DSPs, MCUs, ASSPs, GPPs/RISCs) for features and flexibility, while dedicated hardware are used for parts of the algorithm which are critical regarding timing constraints. With the main focus of the thesis being on the blocks that need to be accelerated and optimized by custom hardware for better performance and power, only ASIC and FPGA implementation technologies are discussed in the following sections.

With the full freedom to customize the hardware to the very last single bit of logic, both ASICs and FPGAs can achieve much better system performance compared to other technologies. However, as they differ in the inner structure of logic blocks building, they posses quite different metrics in areas such as speed, power, unit cost, logic integration, etc. In general, designs implemented with ASIC technology is optimized by utilizing a rich spectrum of logic cells with varied sizes and strengths, along with dedicated interconnection. In contrast, FPGAs with the aim of full flexibility are composed of programmable logic components and programmable interconnects. A typical structure of an FPGA is illustrated in figure 2.1. Figure 2.2 and 2.3 show the details of programable logic components and interconnects. Logic blocks can be formed on site through programming look up tables and the configuration SRAMs which control the routing resources. The programmability of FPGAs comes at the cost of speed, power, size, and cost, which is discussed in details in the following. Table 2.2 gives a comparison between ASICs and FPGAs.

Speed In terms of maximum achievable clock frequency, ASICs are typically much faster than an FPGA given the same manufacture process technology. This is mainly due to the interconnect architecture within FPGAs.

[1] Unit price for volume production

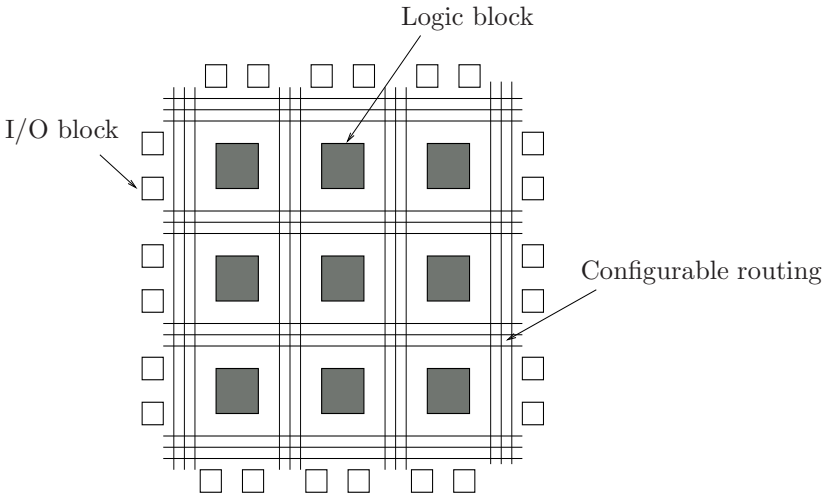


Figure 2.1: A conceptual FPGA structure with configurable logic blocks and routing.

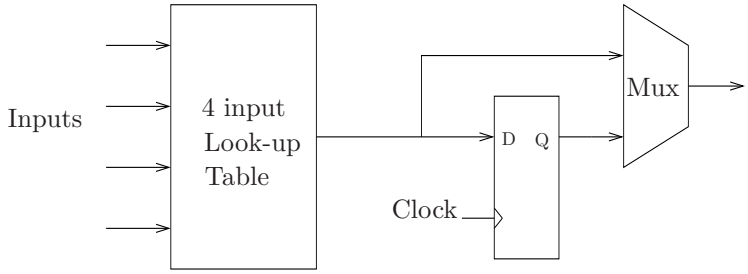


Figure 2.2: Simplified programmable logic elements in an typical FPGA architecture.

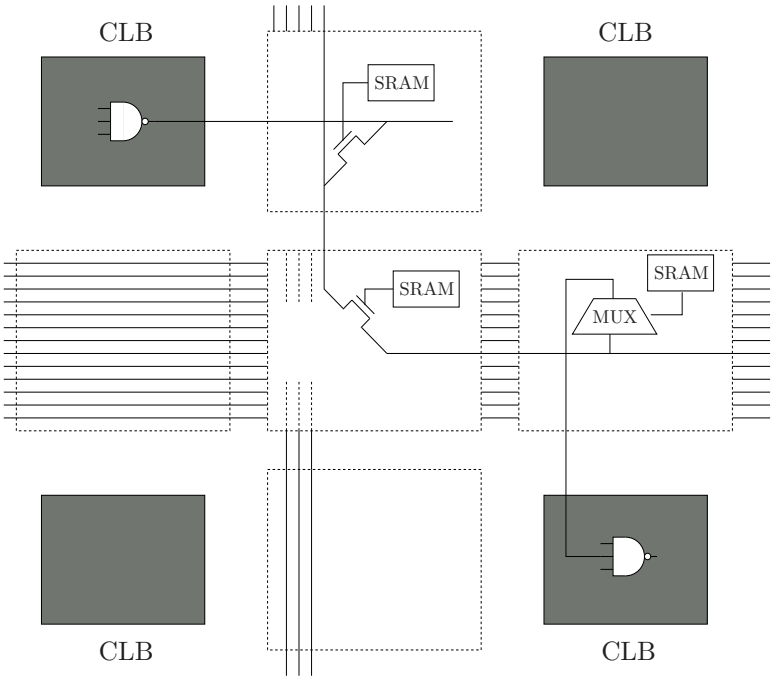


Figure 2.3: Configurable routing resources controlled by SRAMs.

Table 2.2: Comparisons between ASICs and FPGAs.

	ASICs	**FPGAs**
Clock speed	High	Low
Power	Low	High
Unit cost with volume production	Low	High
Logic Integration	High	Low
Flexibility	Low	High
Back-end Design Effort	High	Low
Integrated Features	Low	High

To ensure programmability, many FPGA devices utilize pass transistors to connect different logic cells dynamically, see figure 2.3. These active routing resources add significant delays to signal paths. Furthermore, the length of each wire is fixed to either short, medium, and long types. No further optimization can be exploited on the wire length even when two logic elements are very close to each other. The situation could get even worse if high logic utilization is encountered, in which case it is difficult to find a appropriate route within certain regions. As a result, physically adjacent logic elements do not necessarily get a short signal path. In contrast, ASICs has the facility to utilize optimally buffered wires implemented with metal in many layers, which can even route over logic cells. Another contributor to FPGAs speed degradation lies in its logic granularity. In order to achieve programmability, look-up tables are used which usually have a fixed number of inputs. Any logic function with slightly more input variables will take up additional look-up tables, which will again introduce additional routing and delay. On the contrary, ASICs, usually with a rich spectrum types of logic gates of varying functionality and drive strength (e.g. over 500 types for UMC 0.13 μm technology used at the department), logic functions can be very fine tuned during synthesis process to meet a better timing constraint.

Power The active routing in FPGA devices does not only increase signal path delays, it also introduce extra capacitance. Combined with large capacitances caused by the fixed interconnection wire length, the capacitance in FPGA signal path is in general several times larger than that of an ASIC. Substantial power consumption is dissipated during signal switching that drives such signal paths. In addition, FPGAs have pre-made dedicated clock routing resources, which are connected to all the flip flops on an FPGA in the same clock domain. The capacitance of the flip flop will contribute to the total switching power even when it is not used. Furthermore, the extra SRAMs used to program look-up tables and wires also consume static power.

Logic density The logic density on an FPGA is usually much lower compared to ASICs. Active routing device takes up substantial chip area. Look-up tables waste logic resource when they are not fully used, which is also true for flip-flops following each look-up table. Due to relatively low logic density, around 1/3 of large ASIC designs in the market usually could not fit into one single FPGA [5]. Low logic density increase the cost per unit chip area, which makes ASIC design more preferable for industry designs in mass production.

Despite of all the above drawbacks, FPGA implementation also comes with

quite a few advantages, which is served as the motivation in the thesis work.

Verification Ease Due to its flexibility, an FPGA can be re-programmed as requested when a design flaw is spotted. This is extremely useful for video projects, since algorithms for video applications usually need to be verified over a long time period to observe long term effects. Computer simulations are inherently slow. It could take a computer weeks of time to simulate a video sequences lasting for only several minutes. Besides, an FPGA platform is also highly portable compared to a computer, which makes it more feasible to use in heterogeneous environments for system robustness verification.

Design Facility Modern FPGAs comes with integrated IP blocks for design ease. Most importantly, microprocessors are shipped with certain FPGAs, e.g. (hard Power PC and soft Microblaze processor cores on Virtex II pro and later version of Xilinx FPGAs). This gives great benefit to hardware/software co-design, which is essential in the presented video surveillance project. Algorithm such as feature extraction and tracking is more suitable for software implementation. With the facilitation of various FPGA tools, interaction between software and hardware can be verified easily in an FPGA platform. Minor changes in hardware/software partitioning are easier and more viable compared to ASICs.

Minimum Effort Back-end Design The FPGA design flow eliminates the complex and time-consuming floor planning, place and route, timing analysis, and mask/re-spin stages of the project, since the design logic is already synthesized to be placed onto an already verified, characterized FPGA device. This will facilitate hardware designers more time to concentrate mainly on architecture and logic design task.

From the discussions above, FPGAs are selected as our implementation technology due to its fair performance and all the flexibilities and facilities.

2.2 Image Sensors

An image sensor is a device that converts light intensity to an electronic signal. They are widely used among digital cameras and other imaging devices. The two most commonly used sensor technologies are based on Charge Coupled Devices (CCD) or Complementary Metal Oxide Semiconductor(CMOS) sensors. Descriptions and comparisons of the two technologies are briefly discussed in the following which are based on [6–8]. A summary of the two sensor types is given in Table 2.3. Both devices are composed of a array of fundamental light sensitive elements called photodiodes, which excite electrons (charges)

Table 2.3: Image sensor technology comparisons: CCD vs. CMOS.

	CCD	**CMOS**
Dynamic Range	High	Moderate
Speed	Moderate	High
Windowing	Limited	Extensive
Cost	High	Low
Uniformity	High	Low to moderate
System Noise	Low	High

when there is light with enough photons striking on it. In theory, the transformation from photon to electron is linear so that one photon would release one electron. In general, this is not the case in the real world. Typical image sensors intended for digital cameras will release less than one electron. The photodiode measures the light intensity by accumulating light incident for a short period of time (integration time), until enough charges are gathered and ready to be read out. While CCD and CMOS sensors are quite similar in these basic photodiode structure, they mainly differs in the way how these charges are processed, e.g. readout procedure, signal amplification, and AD conversion. The inner structures of the two devices are illustrated in figure 2.4 and 2.5. CCD sensors read out charges in a row-wise manner: The charges on each row are coupled to the row above, so when the charges are moved down to the row below, new charges from the row above will fill the current position, thus the name Coupled Charged Device. The CCD shifts one row at a time to the readout registers, where the charges are shifted out serially through a charge-to-voltage converter. The signal coming out of the chip is a weak analog signal, therefore an extra off-chip amplifier and AD converter are need. In contrast, CMOS sensors integrates separate charge-to-voltage converter, amplifier, noise corrector and AD converter into each photosite, so the charges are directly transformed, amplified and digitized to digital signals on each site. Row and column decoders can also be added to select each individual pixel for readout since it is manufactured in the same standard CMOS process as main stream logic and memory devices.

With varied inner structures of the two sensor types, each technology has unique strengths but also weaknesses in one area or the other, which are described in the following:

Cost CMOS sensors in general come at a low price at system level since the auxiliary circuits such as oscillator, timing circuits, amplifier, AD converter can be integrated onto the sensor chip itself. With CCD sensors, these functionality have to be implemented on a separate Printed Circuit

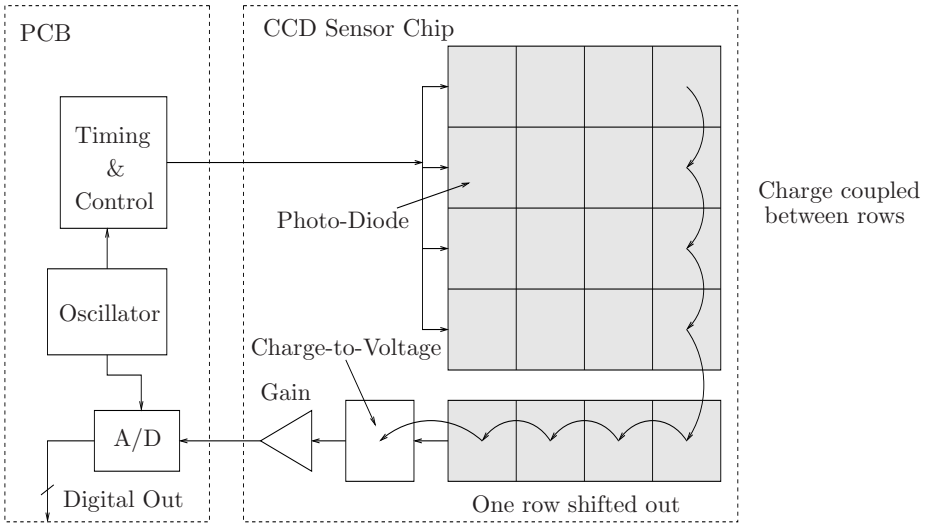


Figure 2.4: A typical CCD image sensor architecture.

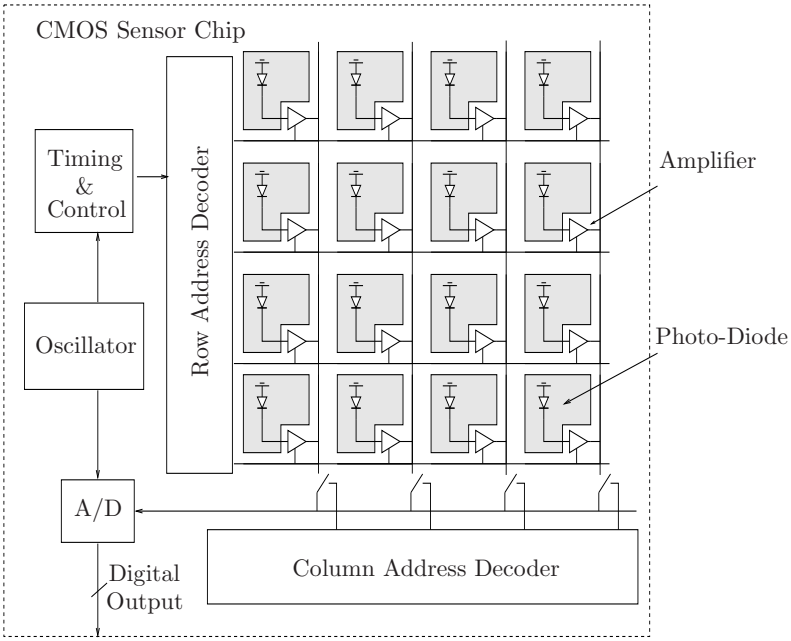


Figure 2.5: A typical CMOS image sensor architecture.

Board (PCB) which results in a higher cost. On the chip level, although CMOS sensor can be manufactured using a foundry process technology that is also capable of producing other circuits in volume, the cost of the chip is not considerable lower than a CCD. This is due to the fact that special, lower volume, optically adapted mixed-signal process has to be used by the requirement of good electro-optical performance [6].

Image Quality The image quality can be measured in many ways:

Noise level CMOS sensors in general have a higher level of noises due to the extra circuits introduced. This can be compensated to some extent by extra noise correction circuits. However this could also increase the processing time between frames.

Uniformity CMOS sensors use separate amplifier for each pixel, the offset and gain of which can vary due to wafer process variations. As a result, the same light intensity will be interpreted as different value. CCD sensor with an off-chip amplifier for every pixel, excel in uniformity.

Light Sensitivity CMOS sensors are less sensitive to light due to the fact that part of each pixel site are not used for sensing light but for processing. The percentage of a pixel used for light sensing is called *fill factor*, which is shown in figure 2.2. In general, CCD sensors have a fill factor of 100% while CMOS sensor has much less, e.g. $30\% - 60\%$ [9]. Possibly, such a drawback can be partially solved by adjusting integration time of each pixel.

Speed and Power In general, a CMOS sensor is faster and consumes lower power compared to a CCD. Moving auxiliary circuits on chip, parasitic capacitance is reduced, which increase the speed at the same time consumes less power.

Windowing The extra row and column decoders in CMOS sensors enable data reading out from arbitrary positions. This could be useful if only portion of the pixel array is of interest. Reading out data with using different resolution is made easy on CMOS sensor without having to discard pixels outside the active window as compared to a CCD sensor.

2.3 Memory Technology

As a deduction from Moore's law, the performance of processors is increasing roughly 60% each year due to the technology scaling. This is never the case for memory chips. In terms of access time, memory performance have only

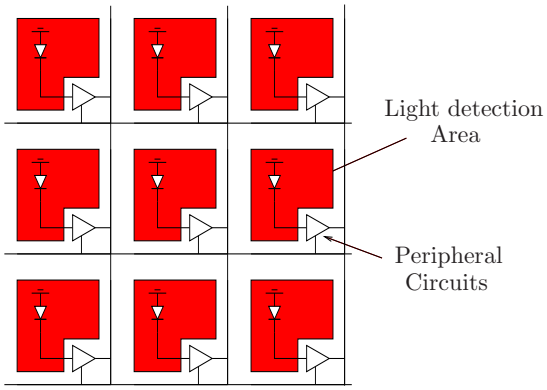


Figure 2.6: Fill factor refers to the percentage of a photosite that is sensitive to light. If circuits cover 25% of each photosite, the sensor is said to have a fill factor of 75%. The higher the fill factor, the more sensitive the sensor.

managed to increase by less than 10% per year [10, 11]. The performance gap between processors and memories has already become a bottle neck of today's hardware system design. With different increase rate, the situation will get even worse in the future until it reaches a point where further increase in processor speed yield little or no performance boost for the whole system, a phenomenon that is called "hitting the memory wall" from the most cited article [12] by W. Wulf et al. on processor memory gap. The traditional way of bridging the gap is by introducing a hierarchical level of caches, while many new approaches are under investigation e.g. [13–15]. In order for better understanding of memory issues today, topics regarding memory technology are given in the following section.

In general, memory technology can be categorized into two types, namely Read Only Memory (ROM) and Random Access Memory (RAM). Due to its read only nature, a ROM is generally made up of a hardwired architecture where a transistor is placed on a memory cell depending on intended content of the cell. The use of a ROM is limited to store fixed information, e.g. look-up table, micro-codes. Many variant technology exists to provide at least one time programmability, e.g. PROM, EPROM, EEPROM and FLASH. RAMs on the other hand with both read and write access are widely used in hardware. Basically, RAMs consists of two types: Static RAM (SRAM) and Dynamic RAM (DRAM). A typical 6 transistor SRAM cell is shown in figure 2.7, while a 1 transistor and a 3 transistor DRAM cells are shown in figure 2.8.

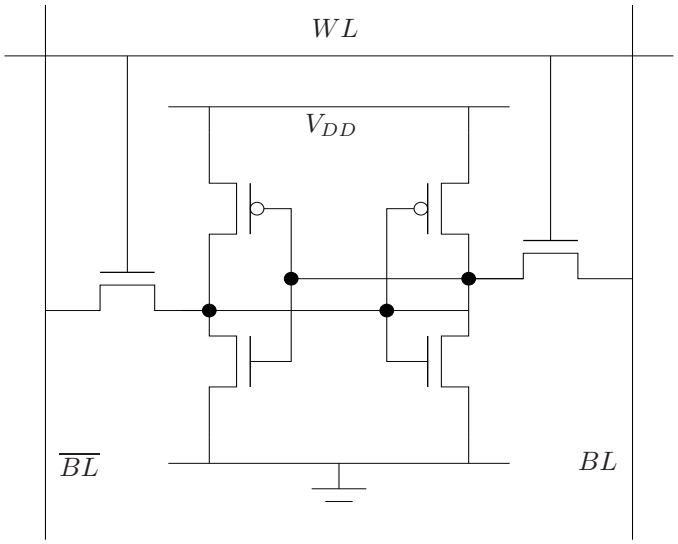
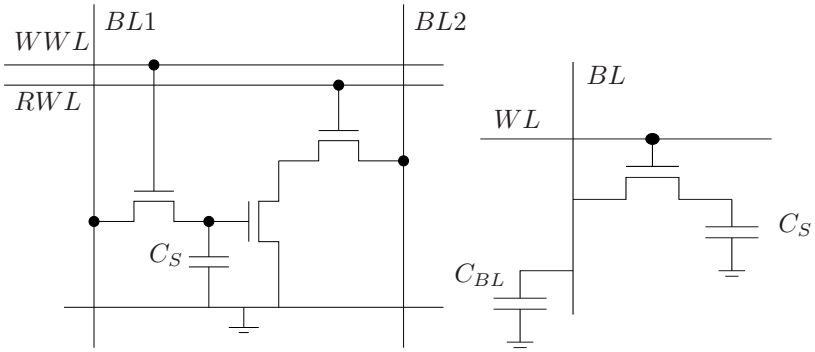


Figure 2.7: An SRAM cell architecture with 6 transistors.



(a) A 3 transistor DRAM cell structure (b) A 1 transistor DRAM cell structure

Figure 2.8: DRAM cell architectures with 1 or 3 transistors.

From the figure, static RAM holds its data in a positive feedback loop with two cascaded inverters. The value will be stored for as long as power is supplied to the circuit. This is in contrast to DRAM, which holds its value on a capacitor. Due to the leakage, the charge on the capacitor will disappear after a period of time. To be able keep the value, the capacitor has to be refreshed constantly. With their respective strengths and weaknesses incurred by their inner structures, SRAMs and DRAMs are used in quite different applications. A brief comparison is made on the two technologies in the following:

Density Each DRAM cell is made up of fewer transistors compared to a SRAM cell, which makes it possible to integrate much more memory cells given the same chip area. Due to the same reason, the cost of DRAMs is much lower.

Speed In general, DRAMs are relatively slow compared to SRAMs. One reason for this is that its high density structure leads to large cell arrays with high word and bit line capacitance. Another reason lies on its complicated read and write cycle with latencies. With its capacity, the address signals are multiplexed into row and column due to limited number of pins, potentially degrading performance. Furthermore, DRAMs needs to be refreshed constantly, during which period no read and write accesses are possible.

Special IC process Integrating denser cells requires modifications in the manufacturing process [16], which makes DRAMs difficult to integrate with standard logic circuits. In general, DRAMs are manufactured in separate chips.

From these properties, DRAMs are generally used as system memory placed off-chip due to its density and cost, while SRAMs is placed on-chip with standard logic circuits, working as L1 and L2 caches due to its speed and ease of integration.

2.3.1 Synchronous DRAM

To overcome the shortcomings existing in traditional DRAMs, new technologies have evolved over years, e.g. Fast Page Mode DRAM (FPM), Extended Data Out DRAM (EDO) and Synchronous DRAM (SDRAM). A good overview can be found from many sources, e.g. [17, 18]. SDRAM gains its popularity by several reasons:

- By introducing clock signals, memory buses are made synchronous to processors. As a result, the commands to be issued to the memories are put in pipelines, so that new operation is executed without waiting

for the completion of the previous ones. Besides, the effort of memory controller design is made easier to some extent, since timing parameters are measured in clock cycles instead of physical timing data.

- SDRAM supports burst memory access to an entire row of data. Synchronous to the bus clock, the data can be read out sequentially without stalling. No column access signals are needed for burst read, the length of the burst accessed in set by a mode register, which is a new feature in SDRAMs. Burst data access will increase memory bandwidth substantially if the data needed by the processor are stored successively in a row.

- SDRAM utilize bank interleaving to minimize extra time introduced by e.g. precharge, refresh. The memory space of a SDRAM is divided into several banks (usually two or four). When one of the bank is being accessed, other banks remains ready to be accessed. When there is a request to access another bank, this will take place immediately without having to wait for the current bank to complete. A continuous data flow can be obtained in such cases.

2.3.1.1 Double Data Rate Synchronous DRAM

To further improve the bandwidth of a SDRAM, Double Data Rate SDRAM (DDR) is developed with doubled memory bandwidth. By using $2n$ pre-fetching techniques, two bits are picked up from the memory array simultaneously to the I/O buffer in two separate pipelines, where they are to be sent on to the bus sequentially on both rising and falling edges of the clock. However, the usage is limited to the situation where the need of multiple accesses is on the same row. In addition to double data rate, the bus signaling technology is changed to a $2.5v$ Stub Series Terminated Logic_2 (SSTL_2) standard [19], which consumes less power. Data strobes signals are also introduced for better synchronization of data signals to memory controllers.

2.3.2 DDR Controller Design on Xilinx VirtexII pro FPGA

With high data bandwidth and complicated timing parameters of a DDR SDRAM, the design of a DDR interface can be challenging. DDR SDRAM works synchronously on a clock frequency at 100 MHz or above. Clock signals together with data and command signals are transferred between memory and processor chips through PCB signal traces. To make sure all data and command signals to be valid in the right timing in respective to the clock is a nontrivial task. Many factors contributes to the total signal uncertainties, e.g.

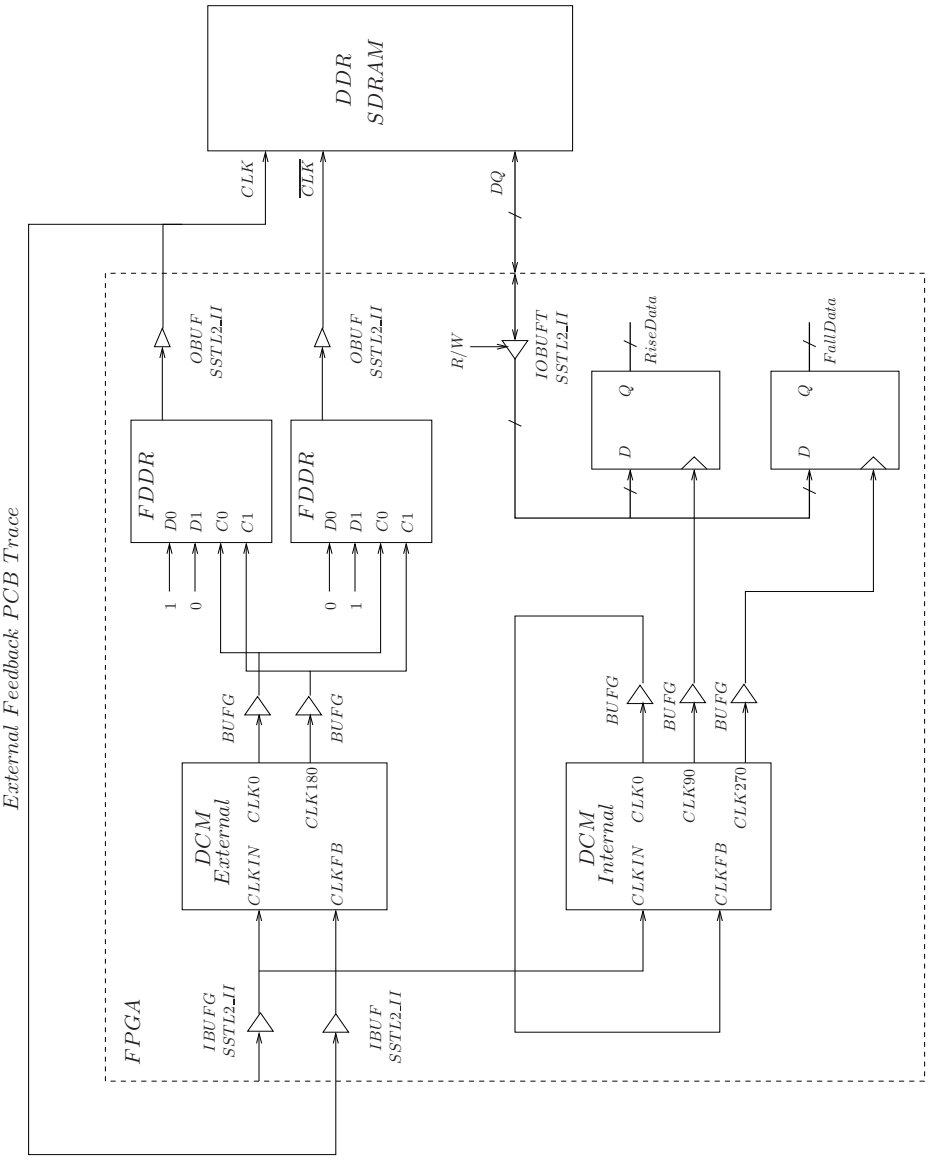


Figure 2.9: Two DCMs are used to synchronize operations on an off-chip DDR SDRAM and on-chip memory controller. DCM external sends off-chip clocks for DDR SDRAM, while DCM internal are used for sending data off-chip or capturing data from an DDR SDRAM.

PCB layout skew, package skew, clock phase offset, clock tree skew and clock duty cycle distortion.

In the following, the timing closure of a DDR controller design for the implementation of the video surveillance unit is described. The memory interface is implemented on a Xilinx Virtex II pro VP30 platform FPGA platform with a working frequency of 100 Mhz.

According to the standard, the data are transferred between a DDR and a processor (FPGA in our implementation) with a bidirectional data strobe signal (DQS). The signal is issued by the memory controller during write operation and it is center aligned with the data. During a read operation, the DDR send the signal together with the data with edge alignment in respect to each other.

To synchronize the operations between an FPGA and a DDR SDRAM, two Digital Clock Managers (DCM) are used, which is shown in figure 2.9.

DCM is a special device in many Xilinx FPGA platforms that provide many functionalities related to the clock management, e.g. delayed locked loop (DLL), digital frequency synthesizer and digital phase shifter. By using the clock signal feedback from the dedicated clock tree, the clock signal referenced internally by each flip-flop inside an FPGA are in phase with the source of the clock from off-chip. From figure 2.9, the DCM External generates the clock signals (clk0 and clk180) that go off-chip to the DDR SDRAM through double data rate flip-flops (FDDR). FDDR updates its outputs on the rising edges of both input clock signals. Thus the clock signals to a DDR can be driven by an FDDR instead of an internal clock signal directly. The DCM Internal generates the clock signals that are used internally by all flip-flops in the memory controller. To be able to align the two clock signals, they are both aligned to the original clock source (the signal driven by IBUFG). The alignment of the DCM External are implemented using a off-chip PCB trace signals that is designed to have the same length as the clock signal trace from the FPGA to the DDR SDRAM. Thus the clock signal arrives at the DDR SDRAM is assumed to be in phase with the external feedback signal that arrives at the DCM External. As the internal clock signals referenced by all flip-flops in the memory controller are also aligned to the original clock signal driven by IBUFG through an internal feedback loop, the clock signal in memory controller is aligned to the clock signal that arrives at the DDR SDRAM clock pin. During read operation, data are transferred from an off-chip DDR on both edges of the clock, in a edge alignment manner. To register the data in the memory controller, a 90° and 270° phase shifted clock signals are used to align with the read being data in the center. This is shown in figure 2.9.

In practice, the internal and external clock signals are not entirely in phase with each other due to skews from many sources. From Xilinx datasheet [20–22], the worst case skews on an Xilinx Virtex II pro devices can result in leading and trailing uncertainties of 880 ps and 540 ps respectively in a read

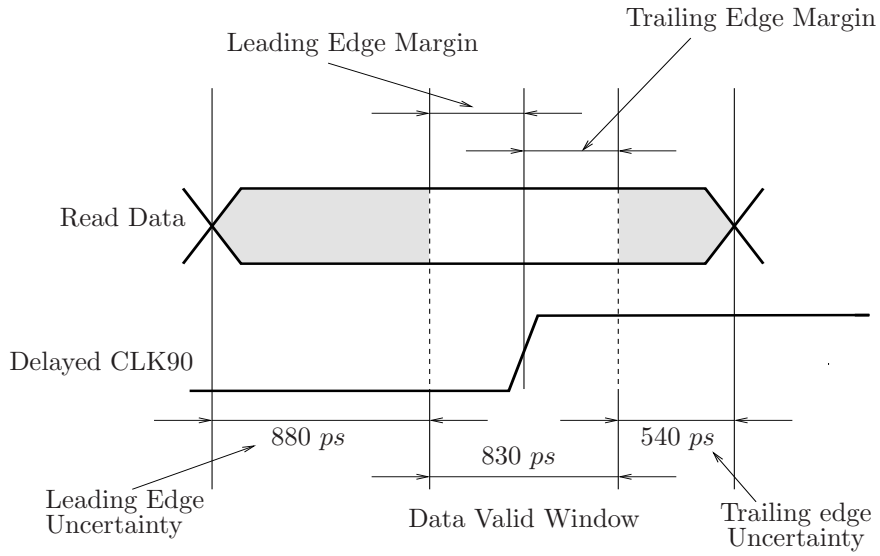


Figure 2.10: DDR read capture data valid window.

data window, which is shown in figure 2.10.

The internal DCM is phase shifted by 1 ns to take the advantage of varied leading and trailing uncertainties, thus the margin of the valid data window is improved, see figure 2.10.

On the other hand, the timing problem with data write operation is minor since clock signals and data signals generated within FPGA propagate through similar logics and trace delays.

2.4 Power Consumption in Digital CMOS technology

Minimization of power consumption has been one of the major concerns in the design of embedded systems due to one of the following two distinctive reasons:

- The increasing system complexity of portable devices leads to more power consumption by more integrated functionality and sophistication, e.g. the multimedia applications on mobile phones such as digital video broadcasting (DVB) and digital camera, higher data rate wireless communication with emerging technologies such as WiMax/802.16. This shortens battery life significantly.

- Reliability and cost issues regarding heat dissipation in the manufacturing of non-portable high end applications. High power consumption requires

expensive packaging and cooling techniques given that insufficient cooling leads to high operating temperatures, which tend to exacerbate several silicon failure mechanisms.

This is especially true for battery-driven system design. With only 30% battery capacity increase in the last 30 years and 30 to 40% over the next 5 years by using new battery technologies [23], e.g.the rechargeable lithium or polymers, the computational power of digital integrated circuits has increased by several orders of magnitude. To bridge the gap, new approaches must be developed to handle power consumption in mobile applications.

2.4.1 Sources of power dissipation

Three major sources contribute to the total power dissipation of digital CMOS circuits, which can be formulated as:

$$P_{tot} = P_{dyn} + P_{dp} + P_{stat}, \tag{2.1}$$

where P_{dyn} is the dynamic dissipation due to charging and discharging load capacitances, P_{dp} is the power consumption caused by direct path between V_{DD} and GND with finite slope of the input signal, and P_{stat} is the static power caused by leakage current. Traditionally, the power consumption by capacitive load has always been the dominant factor. This will not be the case in the design with deep sub-micron technologies, since leakage current increases exponentially with threshold scaling in each new technology generation [24]. For 130 nm technology, leakage can account for 10% to 30% of the total power when active, and dominant when standby [25]. With 90 nm and 65 nm technology, the leakage can reach more than 50%. Power dissipation due to direct path, on the other hand, is usually of minor importance, and can be minimized by certain techniques e.g. supply voltage scaling [26]. With the focus of the thesis being on architecture exploration, power consumption regarding switching power is briefly discussed in the following.

2.4.1.1 Switching Power Reduction Schemes

Power consumption due to signal switching activity can be calculated as [16]:

$$P_{switch} = P_{0\rightarrow 1} C_L V_{DD}^2 f, \tag{2.2}$$

where $P_{0\rightarrow 1}$ is the probability that a output transition of $0 \rightarrow 1$ occurs, C_L is the load capacitance of the driving cell, V_{DD} is the supply voltage, and f is the working clock frequency. From the equation, power minimization strategy can be carried out by constraining any of the factors, which is especially effective for power supply reduction since the power dissipation decreases quadratically

Table 2.4: Power Savings in Different Level of Design Abstraction.

Technique	Savings
Architectural/Logic Changes	45%
Clock Gating	8%
Low power Synthesis	15%
Voltage Reduction	32%

Table 2.5: Core power consumption contribution from different parts of a logic core [36].

Component	Percentage
PLLs/Macros	7.21%
Clocks	52.13%
Standard Cells	6.72%
Interconnect	5.97%
RAMs (including leakage)	16.94%
Logic Leakage	11.04%

with V_{DD}. Power minimization techniques can be applied in all level of design abstractions, ranging from software down to chip layout. In [27–34], comprehensive overviews of various power reduction techniques are given. Suggestions are made to minimize power consumption in all level of a circuit design. In [35], a survey is made to give an overview of amount of power savings that can be generally achieved at different design level. Their experimental results are given in Table 2.4. From the table, it is shown that the most efficient way of lowering power consumption is to work on either high architecture level or the low transistor level. In [36], the contributions to the total power consumption from different blocks of a design are given, which is shown in table2.5. From the table, it can be seen that clock net and memory access contribute over 50% of the total power consumption in the logic core. In the following section, example power reduction schemes are discussed, which only covers power consumption minimization in high level architecture design.

2.4.2 Pipelining and Parallel Architectures

Power consumption can be reduced by using pipelining or parallel architectures. According to [37], the first order estimation of the delay of a logic path can be

calculated as

$$t_d \propto \frac{V_{DD}}{(V_{DD} - V_t)^{\alpha}}. \tag{2.3}$$

With a pipelining architecture, the calculation paths of a design is inserted with pipeline registers. This effectively reduces the t_d in the critical path. Thus V_{DD} can be lowered in the equation while the same clock frequency can be maintained. As stated above, power consumption can be reduced by lowering V_{DD} since it has quadratic effects on power dissipation. The same principle applies to parallel architecture. With hardware duplicated several times, the throughput of a design increases proportionally. Alternatively, a design can achieve for lower power consumption by slowing down the clock frequency of each duplicates. The same throughput is maintained, while the supply voltage can be reduced.

Bibliography

[1] C. Stauffer and W. Grimson, "Adaptive background mixture models for real-time tracking," in *Proc. IEEE Conference on Computer Vision and Pattern Recognition*, 1999.

[2] L. Adams. (2002, November) Choosing the right architecture for real-time signal processing designs. [Online]. Available: http://focus.ti.com/lit/an/spra879/spra879.pdf

[3] P. Eles, Z. Peng, K. Kuchcinski, and A. Doboli, "System level hardware/software partitioning based on simulated annealing and tabu search," *Springer Design Automation for Embedded Systems*, vol. 2, pp. 5–32, January 1997.

[4] T. Wiangtong, P. Y. Cheung, and W. Luk, "Tabu search with intensification strategy for functional partitioning in hardware-software codesign," in *Proc. of the 10 th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM 02)*, California, USA, April 2002, pp. 297– 298.

[5] J. Gallagher. (2006, January) ASIC prototyping using off-the-shelf FPGA boards: How to save months of verification time and tens of thousands of dollars. [Online]. Available: http://www.synplicity.com/literature/whitepapers/pdf/proto_wp06.pdf

[6] D. Litwiller. (2001, January) CCD vs. CMOS: Facts and fiction. [Online]. Available: http://www.dalsa.com/shared/content/Photonics_Spectra_CCDvsCMOS_Litwiller.pdf

[7] ——. (2005, August) CMOS vs. CCD: Maturing technologies, maturing markets. [Online]. Available: http://www.dalsa.com/shared/content/pdfs/CCD_vs_CMOS_Litwiller_2005.pdf

[8] A. E. Gamal and H. Eltoukhy, "CMOS image sensors," *IEEE Circuits and Device Magzine*, vol. 21, pp. 6–20, May-June 2005.

[9] D. Scansen. CMOS challenges CCD for image-sensing lead. [Online]. Available: http://www.eetindia.com/articles/2005oct/b/2005oct17_stech_opt_ta.pdf

[10] J. L. Hennessy and D. A. Patterson, *Computer Architecture: A Quantitative Approach, Third Edition*. Morgan Kaufmann, 2002.

[11] N. R. Mahapatra and B. Venkatrao, "The processor-memory bottleneck: Problems and solutions," Tech. Rep. [Online]. Available: http://www.acm.org/crossroads/xrds5-3/pmgap.html

[12] W. A. Wulf and S. A. McKee, "Hitting the memory wall: Implications of the obvious," *Computer Architecture News*, vol. 23, pp. 20–24, March 1995.

[13] "The berkeley intelligent RAM (IRAM) project," Tech. Rep. [Online]. Available: http://iram.cs.berkeley.edu/

[14] C. C. Liu, I. Ganusov, M. Burtscher, and S. Tiwari, "Bridging the processor memory performance gap with $3D$ IC technology," *IEEE Design and Test of Computers*, vol. 22, pp. 556– 564, November 2005.

[15] "*puma*^{2}, proactively uniform memory access architecture," Tech. Rep. [Online]. Available: http://www.ece.cmu.edu/ puma2/

[16] J. M. Rabaey, A. Chandrakasan, and B. Nikolić, *Digital Integrated Circuits: A Design Perspective, Second Edition*. Prentice Hall, 2003.

[17] T.-G. Hwang, "Semiconductor memories for it era," in *Proc. of IEEE International Solid-State Circuits Conference (ISSCC)*, California, USA, February 2002, pp. 24–27.

[18] (2005) Memory technology evolution. [Online]. Available: http://h20000.www2.hp.com/bc/docs/support/SupportManual/c00266863/c00266863.pdf

[19] [Online]. Available: http://download.micron.com/pdf/misc/sstl_2spec.pdf

[20] M. George. (2006, December) Memory interface application notes overview. [Online]. Available: http://www.xilinx.com/bvdocs/appnotes/xapp802.pdf

[21] N. Gupta and M. George. (2004, May) Creating high-speed memory interfaces with Virtex-II and Virtex-II Pro FPGAs. [Online]. Available: http://www.xilinx.com/bvdocs/appnotes/xapp688.pdf

[22] N. Gupta. (2005, January) Interfacing Virtex-II devices with DDR SDRAM memories for performance to 167 mhz. [Online]. Available: http://www.xilinx.com/support/software/memory/protected/XAPP758c.pdf

[23] W. L. Goh, S. S. Rofail, and K.-S. Yeo. Low-power design: An overview. [Online]. Available: http://www.informit.com/articles/article.asp?p=27212&rl=1

[24] G. E. Moore, "No exponential is forever: But "Forever" can be delayed!" in *Proc. of IEEE International Solid-State Circuits Conference (ISSCC)*, California, USA, February 2003, pp. 20–23.

[25] B. Chatterjee, M. Sachdev, S. Hsu, R. Krishnamurthy, and S. Borkai, "Effectiveness and scaling trends of leakage control techniques for sub4 30nm CMOS technologies," in *Proc. of International Symposium on Low Power Electronics and Design (ISLPED)*, California, USA, August 2003, pp. 122–127.

[26] T. Olsson, "Distributed clocking and clock generation in digital CMOS SoC ASICs," Ph.D. dissertation, Lund University, Lund, 2004.

[27] J. M. Rabaey and M. Pedram, *Low Power Design Methodologies*. Springer, 1995.

[28] A. P. Chandrakasan, S. Sheng, and R. W. Brodersen, "Low-power CMOS digital design," *IEEE Journal of Solid-State Circuits*, vol. 27, pp. 473 – 484, April 1992.

[29] D. Garrett, M. Stan, and A. Dean, "Challenges in clockgating for a low power ASIC methodology," in *Proc. of International Symposium on Low Power Electronics and Design*, California, USA, August 1999, pp. 176 – 181.

[30] Y. J. Yeh, S. Y. Kuo, and J. Y. Jou, "Converter-free multiple-voltage scaling techniques for low-power CMOS digital design," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 20, pp. 172 – 176, January 2001.

[31] T. Kuroda and M. Hamada, "Low-power CMOS digital design with dual embedded adaptive power supplies," *IEEE Journal of Solid-State Circuits*, vol. 35, pp. 652 – 655, April 2000.

[32] A. Garcia, W. Burleson, and J. L. Danger, "Low power digital design in FPGAs: a study of pipeline architectures implemented in a FPGA using a low supply voltage to reduce power consumption," in *Proc. IEEE International Symposium on Circuits and Systems (ISCAS)*, Geneva, Switzerland, May 2000, pp. 561 – 564.

[33] P. Brennan, A. Dean, S. Kenyon, and S. Ventrone, "Low power methodology and design techniques for processor design," in *Proc. International Symposium on Low Power Electronics and Design (ISLPED)*, California, USA, August 1998, pp. 268 – 273.

[34] L. Benini, G. D. Micheli, and E. Macii, "Designing low-power circuits: practical recipes," *IEEE Circuits and Systems Magazine*, vol. 1, pp. 6–25, 2001.

[35] F. G. Wolff, M. J. Knieser, D. J. Weyer, and C. A. Papachristou, "High-level low power FPGA design methodology," in *Proc. IEEE Conference on National Aerospace and Electronics Conference (NAECON)*, Ohio, USA, October 2000, pp. 554–559.

[36] S. GadelRab, D. Bond, and D. Reynolds, "Fight the power: Power reduction ideas for ASIC designers and tool providers," in *Proc. of SNUG Conference*, California, USA, 2005.

[37] K. K. Parhi, *VLSI Digital Signal Processing Systems: Design and Implementation*. John Wiley & Sons, 1999.

Hardware Accelerator Design of an Automated Video Surveillance System

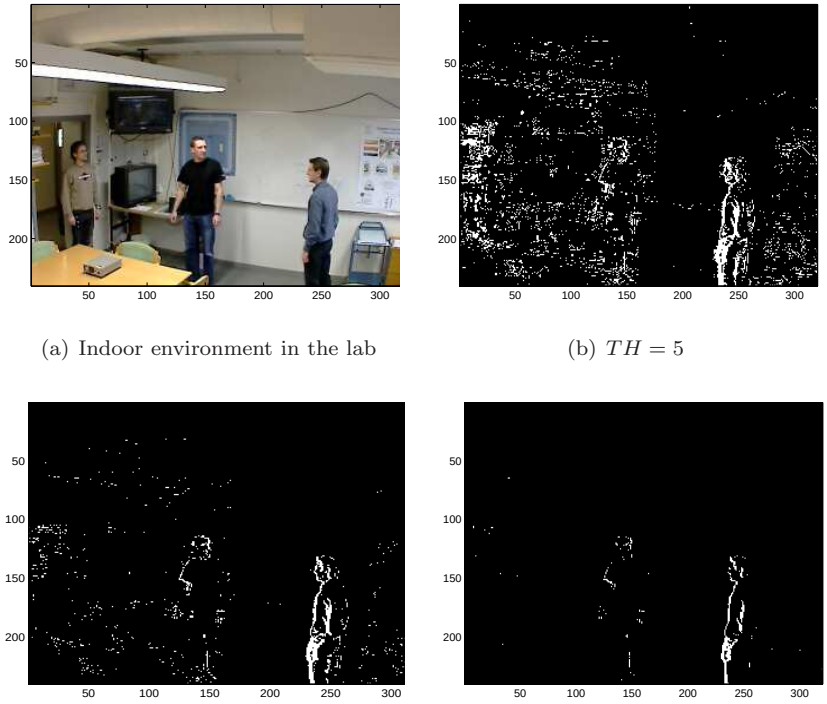
Chapter 1

Segmentation

1.1 Introduction

The use of video surveillance systems is omnipresent in the modern world in both a civilian and a military contexts, e.g. traffic control, security monitoring and antiterrorism. While traditional Closed Circuit TV (CCTV) based surveillance systems put heavy demands of human operators, there is an increasing needs for automated video surveillance system. By building a self contained video surveillance system capable of automatic information extraction and processing, various events can be detected automatically, and alarms can be triggered in presence of abnormity. Thereby, the volume of data presented to security personnel is reduced substantially. Besides, automated video surveillance better handles complex cluttered or camouflaged scenes. A video feed for surveillance personnel to monitor after the system has announced an event will support improved vigilance and increase the probability of incident detection.

Crucial to most of such automated video surveillance systems is the quality of the video segmentation, which is a process of extracting objects of interest (foreground) from an irrelevant background scene. The foreground information, often composed of moving objects, is passed on to later analysis units, where objects are tracked and their activities are analyzed. To be able to perform video segmentation, a so called *background subtraction* technique is usually applied. With a reference frame containing a pure background scene being maintained for all pixel locations, foreground objects are extracted by thresholding the difference between the current video frame and the background frame. In the



(a) Indoor environment in the lab

(b) $TH = 5$

(c) $TH = 10$

(d) $TH = 20$

Figure 1.1: Video segmentation results with the frame difference approach. Different threshold value are tested in the indoor environment in our lab.

following section, a range of background subtraction algorithms are reviewed, along with the discussions on their performances and computational complexity. Based on these discussions, trade-offs are made with a specific algorithm based on Mixture of Gaussian (MoG) being selected as the baseline algorithm for hardware implementation. The algorithm is subjected to modifications to better fit implementation on an embedded platform.

1.2 Alternative Video Segmentation Algorithms

1.2.1 Frame Difference

A Background/Foreground detection can be achieved by simply observing the difference of the pixels between two adjacent frames. By setting a threshold value, a pixel is identified as foreground if the difference is higher than the threshold value or background otherwise. The simplicity of the algorithm comes at the cost of the segmentation quality. In general, bigger regions are detected as foreground area than the actual moving part. Also it fails to detect inner pixels of a large, uniformly-colored moving object, a problem known as aperture effect [1]. In addition, setting a global threshold value is problematic since the segmentation is sensitive to light intensity. Figure 1.1 shows segmentation results with a video sequence taken in our lab, where three people are moving in front of a camera. From these figures, it can be seen that with lower threshold value, more details of the moving objects are revealed. However, this comes with substantial noise that could overwhelm the segmented objects, e.g. the left most person in figure 1.1(b). On the other hand, increasing the threshold value reduces noise level, at the cost of less details detected to a point where almost whole objects are missing, e.g. left most person in figure 1.1(d). In general, inner parts of all objects are left undetected, due to their uniformity colors that result in minor value changes over frames. In spite of the segmentation quality, the frame difference approach suits well for hardware implementation. The computational complexity as well as memory requirements are rather low. With the memory size of only one video frame and minor hardware calculation, e.g. an adder and a comparator, it is still found as part of many video surveillance systems of today [1–4].

1.2.2 Median Filter

While the frame difference approach uses the previous frame as the background reference frame, it is inherently unreliable and sensitive to noise with the moving objects contained in the reference frame and the varying illumination noise over frames. An alternative approach to obtain a background frame is by using median filters. A median filter has traditionally been used in spatial image filtering process to remove noise [5]. The basic idea of noise reduction lies in the fact that a pixel corrupted by noise makes a sharp transition in the spatial domain. By checking the surrounding pixels that centers at the pixel in question, the middle value is selected to replace the center pixel. By doing this, the pixel in question is forced to look like its neighbors, thus the extinctive pixel value corrupted by noise are replaced. Inspired by this, median filters are used to model background pixels with reduced noise deviation by filtering pixel values

in the time domain. they are used in many applications [6–8], with the median filtering process carried out over the previous n frames, e.g. $50 - 200$ frames in [6]. To avoid foreground pixel values to be mixed into the background, the number of frames has to be large so that more than half the pixel values belongs to the background. The principle is illustrated in figure 1.2, where the number of both foreground and background pixels are shown in a frame buffer. Due to various noise, a pixel value will not stay at exactly the same value over frames, thus the histograms are used to represent both the foreground and the background pixels. Consider the case when the number of background pixels is more than that of foreground pixel by only one. The median value will lie right at the right foot of background histogram. With increasing background pixel filled into the buffer, the value is moving towards the peak of the background histogram. Under the previous assumption that no foreground pixel will stay in the scene for more than half size of the buffer, the median value will move along the background histogram back and forth, representing the background pixel value for the current frame. Using buffers to store previous n frames is costly in memory usage. In certain situations, number of buffered frames could increase substantially, e.g. slowly moving objects with uniformly colored surface are present in the scene or the foreground objects stopped for a while before moving on to another location. The calculation complexity is also proportional to the number of buffers. To find the median value it is necessary to sort all the values in the frame buffer in numerical order which is hardware costly with large number of frame buffers.

1.2.3 **Selective Running Average**

One similar alternative to median filtering is to use the average instead of the median value over previous n frames. Noise distortions to a background pixel over frames can be neutralized by taking the mean value of the pixel samples collected over time. To avoid huge memory requirements similar to the median filtering approach, a running average can be utilized which takes the form of

$$B_t = (1 - \alpha)B_{t-1} + \alpha F_t, \tag{1.1}$$

where α is the learning rate, F and B are the current frame and background frame formed by the mean value of each pixel respectively. With such an approach, only a frame of mean values are needed to be stored in a memory. The average operation is carried out by incorporating a small portion of the new frames into the mean values at a time, using a learning factor α. At the same time, the same portion of the current mean value is discarded. Depending on the value of α, such a average operation can be fast or slow. For background modeling, a fast learning factor could result in foreground pixels to be quickly incorporated into background, thus limiting its usage to certain situations, e.g.

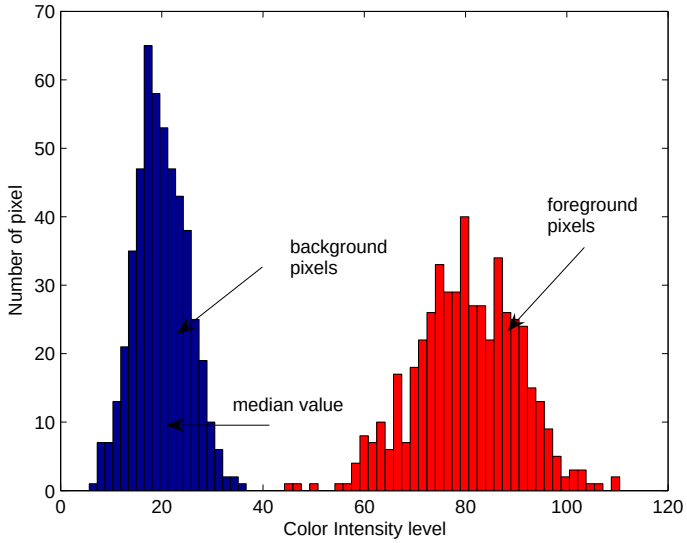


Figure 1.2: Foreground and Background pixel histograms: With more pixels in the buffer falling within Background, the median value moves towards the center of Background distribution.

initialization phase with only background scene.

To avoid a foreground pixel to be mixed into the background updating process, a selective running average can be applied. This is shown in the following equations:

$$B_t = (1-\alpha)B_{t-1} + \alpha F_t \quad if \quad F_t \subset background \tag{1.2}$$

$$B_t = B_{t-1} \quad if \quad F_t \subset foreground. \tag{1.3}$$

With the foreground/background distinction performed before background frame updating process, more recent "clean" background pixels contributes to the form of the new mean value, which makes the background modeling more accurate. The selective running average method is used in many applications, e.g. [9, 10], and forms the basics of other alternative algorithms with much higher complexity, e.g. Mixture of Gaussian (MOG) discussed in the following sections. The merit of the approach comes in its relatively low hardware complexity, e.g. simple multiplications and additions are needed to update the mean value for each pixel. Together with low memory requirements of storing

only one frame of mean values, running selective average fits well for hardware implementation. Acting virtually with the same principles as a mean filter, selective running average achieves similar segmentation results as that of the median filtering approach.

1.2.4 **Linear Predictive Filter**

To be able to estimate the current background more accurately, linear predictive filters are developed for background modeling in several literatures [11–15]. The problem with taking the median or mean of the past pixel samples lies in the fact that it does not reflect the uncertainty (variance) of how a background pixel value could drift from its mean value. Without any of this information, the foreground/background distinction has to be done in a heuristic way. An alternative approach can be utilized which predicts the current background pixel value from its recent history values. Compared to mean and median values, a prediction value can more accurately represent the true value of the current background pixel, which effectively decrease the uncertainty of the variation of a background pixel. As a result, a tighter threshold value can be selected to achieve a more precise segmentation with a better chance of avoiding camouflage problem, where foreground and background holds similar pixel values. Toyama et al. [11] uses an one-step Wiener filter to predict a background value based on its recent history of values. In their approach, a linear estimation of the current background value is calculated as:

$$\hat{B}_t = \sum_{k=1}^{N} \alpha_k I_{t-k}, \tag{1.4}$$

where $\hat{B}$ is the current background estimation, I_{t-k} is one of the history values of a pixel, and α_k is the prediction coefficient. The coefficient are calculated to minimize mean square of the estimation error, which is formulated as:

$$E[e_t^2] = E[(B_t - \hat{B}_t)^2]. \tag{1.5}$$

According to the procedure described in [16], the coefficients can be obtained by solving a set of linear equations as follows:

$$\sum_{k=1}^{p} \alpha_k \sum_t I_{t-k} I_{t-i} = -\sum_t I_t I_{t-i}, \qquad 1 \le i \le p. \tag{1.6}$$

The estimation of coefficients and pixel predictions are calculated recursively during each frame. In [11], a pixel value with a deviation of more than $4.0 \times \sqrt{E[e_t^2]}$ is considered foreground pixel. In total, 50 past values are used in [11] for each pixel to calculate 30 coefficients.

Wiener filters are also expensive in computation and memory requirement. N frame buffers are needed to store a history of frames. Background pixel prediction and coefficients updating are very costly since a set of linear functions are needed to obtain the value. p multiplication and $p-1$ additions are needed for prediction, plus the solution of a linear equation of order p.

An alternative approach for linear prediction is to use Kalman filters. Basic Kalman filter theory can be found in many literatures, e.g. [12,13,15]. Kalman filters are widely used for many background subtraction applications, e.g. [13–15]. It predicts the current background pixel value by recursive computing from the previous estimate and the new input data. A brief formulation of the theory is given in the below according to [13], while a detailed description of Kalman filters can be found in [12].

Kalman filters provide an optimal estimate of the state of the process x_t, by minimizing the difference of the average of the estimated outputs and the average of the measures, which is characterized by the variance of the estimation error. The definition of a state can vary in different applications, e.g. the estimated value of the background pixel and its derivative in [15]. Kalman filtering is performed in essentially two steps: prediction and correction: In the prediction step, the current state of the system is predicted from the previous state as

$$\hat{\mathbf{x}}_t^- = A\hat{\mathbf{x}}_{t-1}, \tag{1.7}$$

where A is the state transition matrix, $\hat{\mathbf{x}}_{t-1}$ is the previous state estimate and $\hat{\mathbf{x}}_t^-$ is the estimation of the current state before correction. In order to minimize the difference between the measure and the estimated state value $\mathbf{I}_t - \mathbf{H}\hat{\mathbf{x}}_t^-$. $\mathbf{I}_t$ is the current observation and $\mathbf{H}$ is the transition matrix that maps the state to the measurements. A variance of such difference is calculated based on

$$\mathbf{P}_t^- = A\mathbf{P}_{t-1}A^T + Q_t, \tag{1.8}$$

where Q_t represents the process noise, P_{t-1} is the previous estimation error variance and $\mathbf{P}_t^-$ is the estimation of error variance based on current prediction state value. With a filter gain factor calculated by

$$K_t = \frac{\mathbf{P}_t^- C^T}{C\mathbf{P}_t^- C^T + R_t}, \tag{1.9}$$

where R_t represents the variances of measurement noise and C is the transition matrix that maps the state to the measurement. The corrected state estimation becomes

$$\hat{\mathbf{x}}_t = \hat{\mathbf{x}}_{t-1} + K_t(I_t - H\mathbf{x}_t^-), \tag{1.10}$$

and the variance after correction is reduced to

$$\mathbf{P}_t = (1 - K_t C)\mathbf{P}_t^-. \tag{1.11}$$

Ridder et al. [15] use both background pixel intensity value and its temporal derivative $\mathbf{B}_t$ and $\mathbf{B}'_t$ as the state value:

$$\hat{\mathbf{x}}_t = \begin{bmatrix} \mathbf{B}_t \\ \mathbf{B}'_t \end{bmatrix}, \tag{1.12}$$

and the parameters are selected as follows:

$$A = \begin{bmatrix} 1 & 0.7 \\ 0 & 0.7 \end{bmatrix} \quad and \quad H = \begin{bmatrix} 1 & 0 \end{bmatrix}. \tag{1.13}$$

The gain factor $\mathbf{K}_t$ varies between a slow adaptation rate α_1 and a fast adaptation rate α_2 depending on whether the current observation is a background pixel or not:

$$\mathbf{K}_t = \begin{bmatrix} \alpha_1 \\ \alpha_1 \end{bmatrix} if\ I_{t-1}\ is\ foreground,\ and \begin{bmatrix} \alpha_2 \\ \alpha_2 \end{bmatrix}\ otherwise. \tag{1.14}$$

In summary, a recursive background prediction approach with Kalman filters can be obtained by combining equations 1.10,1.12,1.13 and 1.14, which can be formulated as follows:

$$\begin{bmatrix} \mathbf{B}_t \\ \mathbf{B}'_t \end{bmatrix} = \mathbf{A} \begin{bmatrix} \mathbf{B}_{t-1} \\ \mathbf{B}'_{t-1} \end{bmatrix} + \mathbf{K}_t \left(I_t - \mathbf{HA} \begin{bmatrix} \mathbf{B}_{t-1} \\ \mathbf{B}'_{t-1} \end{bmatrix} \right). \tag{1.15}$$

The Kalman filtering approach is efficient for hardware implementation. From equation 1.15, three matrix multiplication with size of 2 are needed. Memory requirement is low with one frame of estimated background pixel value stored. The linear predictive approach is reported to achieve better results than the many other algorithms e.g. median or mean filtering approaches, especially in dealing with camouflage problem [11], where foreground pixels holding similar color as that of the background pixels are undetected.

1.2.5 **Mixture of Gaussian**

So far, predictive methods have been discussed which model the background scene as a time series and develop a linear dynamical model to recover the current input based on past observations. By minimizing the variance between the predicted value and past observations, the estimated background pixel is adaptive to the current situation where its value could vary slowly over time. While this class of algorithms may work well with quasi-static background scenes with slow lighting changes, it fails to deal with multi-modal situations, which will be discussed in detail in the following sections. Instead of utilizing the order of incoming observations to predict the current background value, a Gaussian

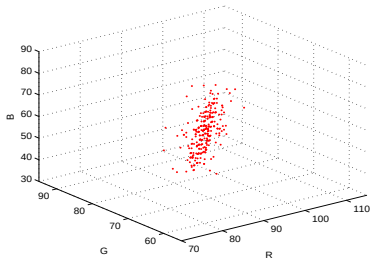
distribution can be used to model a static background value by accounting for the noise introduced by small illumination changes, camera jitter and surface texture. In [17], three Gaussian are used to model background scenes for traffic surveillance. The hypothesis is made that each pixel will contain the color of either the road, the shadows or the vehicles. Stauffer et al. [18] generalized the idea by extending the number of Gaussian for each pixel to deal with a multi-modal background environment, which are quite common in both indoor and outdoor environments. A multi-modal background is caused by repetitive background object motion, e.g. swaying trees or flickering of a monitor. As a pixel lying in the region where repetitive motion occurs will generally consists of two or more background colors, the RGB value of that specific pixel will have several pixel distributions in the RGB color space. The idea of multi-modal distribution is illustrated by figure 1.3. From the figure, a typical indoor environment in 1.3(a) consists of static background objects, which are stationary all the time. A pixel value in any location will stay within one single distribution over time. This is in contrast with the outdoor environment in figure 1.3(c), where quasi-static background objects e.g. swaying leaves of a tree, are present in the scene. Pixel value from these regions contains multiple background colors from time to time, e.g. the color of the leave, the color of house or something in between.

With multi-modal environments, the value of quasi-static background pixels tends to jump between different distributions, which will be modeled by fitting different Gaussians for each distribution. The idea of Mixture of Gaussians (MoG) is quite popular and many different variants are developed [19–24] based on it.

1.2.5.1 **Algorithm Formulation**

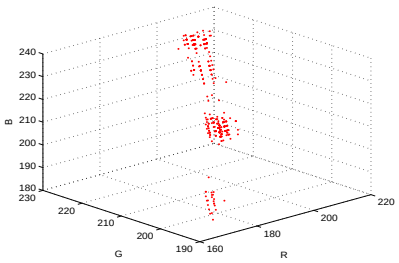
The Stauffer-Grimson algorithm is formulated as modeling a *pixel process* with a mixture of Gaussian distributions. A pixel process is defined as the recent history values of each pixel obtained from a number of consecutive frames. For a static background pixel process, the values will rather be pixel clusters than identical points when they are plotted in a RGB color space. This is due to the variations caused by many factors, e.g. surface texture, illumination fluctuations, or camera noise. To model such a background pixel process, a Gaussian distribution can be used with a mean equal to the average background color and variances accounting for the value fluctuation. More complicated background pixel processes appear when it contains more than one background object surfaces, e.g. a background pixel of a road is covered by leaves of a tree from time to time. In such cases, a mixture of Gaussian distributions are necessary to model multi-modal background distributions. Formally, the Stauffer-Grimson algorithm tries to address background modeling as in the

(a) A typical indoor environment taken in the staircase.



(b) A pixel value sampled over time in the indoor environment contains uni-modal pixel distributions.

(c) A dynamic outdoor environment containing swaying trees.



(d) A pixel value sampled over time in the region that contain leaves of a tree will generally become multi-modal distributions in the RGB color space.

Figure 1.3: Background pixel distributions taken in different environments possess different properties in the RGB color space.

following:

Each pixel is represented by a set of Gaussian distributions $k \subset \{1, 2...K\}$, where the number of distributions K is assumed to be constant (usually between 3 and 7). Some of the K distributions correspond to background objects and the rest are regarded as foreground. Each of the mixture of Gaussians is weighted with a parameter ω_k, which represents probability of current observation belonging to the distribution, thus the equation

$$\Sigma_{k=1}^{K} \omega_k = 1. \tag{1.16}$$

The probability of the current pixel value X being in distribution k is cal-

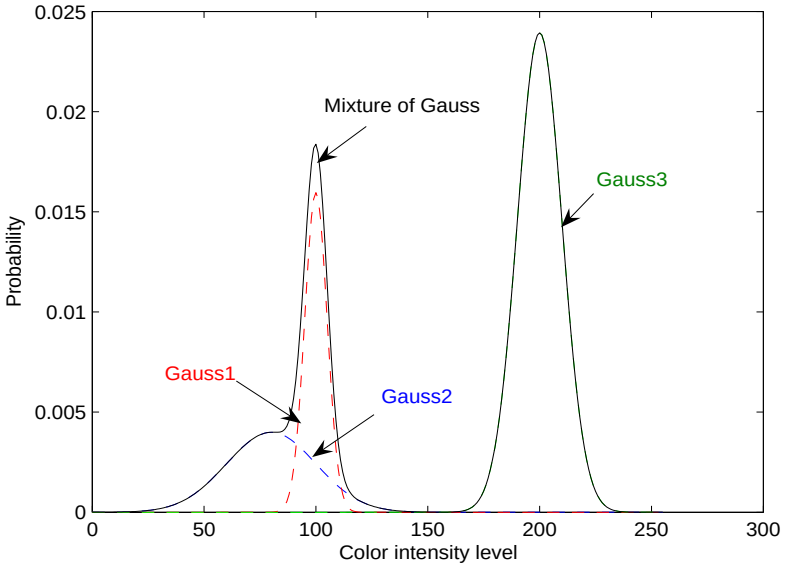


Figure 1.4: Three Gaussian distributions are plotted in the figure with their mean and variance as $\{80, 20\}$, $\{100, 5\}$, $\{200, 10\}$ respectively. Their prior weight is specified as $\{0.2, 0.2, 0.6\}$. The probability of a new pixel observation belonging to one of the distributions can be seen as a sum of three Gaussian distributions [25].

culated as:

$$f(X|k) = \frac{1}{(2\pi)^{\frac{n}{2}}|\Sigma_k|^{\frac{1}{2}}} e^{-\frac{1}{2}(X-\mu_k)^T \Sigma_k^{-1}(X-\mu_k)}, \tag{1.17}$$

where μ_k is the mean and Σ_k is the covariance matrix of the K^{th} distribution. Thus, the probability of a pixel belonging to one of the Gaussian distribution is the sum of probabilities of belonging to each of the Gaussian distribution, which is illustrated in 1.4 [25]. A further assumption is usually made that the different color component is independent of each other so that the covariance matrix is diagonal - more suitable for calculations, e.g. matrix inversion. Stauffer et.al. go even further in assuming that the variances are identical, implying for example that deviations in the red, green, and blue dimensions of a color space have the same statistics. While such simplification reduce the computational complexity, it has certain side effects which will be discussed in the following sections.

The most general solution to the foreground segmentation can be briefly formulated as: at each sample time t, the most likely distribution k from a set of observations is estimated from X, along with a procedure for demarcating the foreground states from the background states.

This is done by the following: A match is defined as the incoming pixel within J times the standard deviation off the center, where in [18] J is selected as 2.5. Mathematically, the portion of the Gaussian distributions belonging to the background is determined by

$$B = argmin_b \left(\sum_{k=1}^{b} \omega_k > H \right), \tag{1.18}$$

where H is a predefined parameter and ω_k is the weight of distribution k. If a match is found, the matched distribution is updated as:

$$\omega_{k,t} = (1 - \alpha)\omega_{k,t-1} + \alpha \tag{1.19}$$

$$\mu_t = (1 - \rho)\mu_{t-1} + \rho X_t \tag{1.20}$$

$$\sigma^2 = (1 - \rho)\sigma^2_{t-1} + \rho(X_t - \mu_t)^T(X_t - \mu_t); \tag{1.21}$$

where μ, σ are the mean and variance respectively, α, ρ are the learning factors, and X_t are the incoming RGB values. The mean, variance and weight factors are updated frame by frame. For those unmatched, the weight is updated according to

$$\omega_{k,t} = (1 - \alpha)\omega_{k,t-1}, \tag{1.22}$$

while the mean and the variance remain the same. If none of the distributions are matched, the one with the lowest weight is replaced by a distribution with the incoming pixel value as its mean, a low weight and a large variance.

1.2.6 **Kernel Density Model**

In [26], it was discovered that the histogram of a dynamic background in an outdoor environment covers a wide spectrum of gray levels (or intensity level of different color component), and all these variations occur in a short period of time, e.g. 30 seconds. Modeling such a dynamic background scene with a limited number of Gaussian distributions are not feasible.

In order to adapt fast to the very recent information about an image sequence, a kernel density function background modeling can be used which only use a recent history of past values to distinguish foreground from background pixels.

Given a history of past values x_1, x_2,...x_N, a kernel density function can be formulated as the following:

The probability of a new observation having a value of x_t can be calculated using a density function:

$$Pr(x_t) = \frac{1}{n}\sum_{i=1}^{N} K(x_t - x_i). \tag{1.23}$$

What this equation actually indicates is that a new background observation can be predicted by the combination of its recent past history samples. If K is chosen to be a Gaussian distribution, then the density estimation becomes

$$Pr(x_t) = \frac{1}{N}\sum_{i=1}^{N} \frac{1}{(2\pi)^{\frac{d}{2}}|\Sigma|^{\frac{1}{2}}} e^{-\frac{1}{2}(x_t - x_i)^T \Sigma^{-1}(x_t - x_i)}. \tag{1.24}$$

Under similar assumptions as the mixture of Gaussian approach, if different color components are independent of each other, the covariance matrix Σ becomes

$$\Sigma = \begin{bmatrix} \delta_1^2 & 0 & 0 \\ 0 & \delta_2^2 & 0 \\ 0 & 0 & \delta_3^2 \end{bmatrix} \tag{1.25}$$

and the density estimation is reduced to

$$Pr(x_t) = \frac{1}{N}\sum_{i=1}^{N}\prod_{j=1}^{d} \frac{1}{\sqrt{2\pi\delta_j^2}} e^{-\frac{1}{2}\frac{(x_{t_j} - x_{i_j})^2}{\delta_j^2}}. \tag{1.26}$$

From the definition of probability estimation, a foreground/background classification process can be carried out by checking the probability value against a threshold value, e.g. if $Pr(xt) < th$, the new observation can not be predicted by its past history, thus recognized as a foreground pixel. Kernel density estimation generalize the idea of the Gaussian mixture model, where each single sample of the N samples is considered to be a Gaussian distribution by itself. Thus it can also handle multi-modal background scenarios. The probability calculation only depends on its N past values, which makes the algorithm quickly adapt to the dynamic background scene.

Regarding hardware implementation complexity, the kernel density model needs to store N past frames, which makes it a memory intensive task. The calculation of probability in equation 1.26 is costly. In [26], a look-up table is suggested to store precalculated value for each $x_t - x_i$. This will further increase the requirement on the memory.

1.2.7 **Summary**

A wide range of segmentation algorithms have been discussed, each with related robustness to different situations and each with different computational

Table 1.1: Algorithm Comparison.

	FD	Median	LPF	MoG	KDE
Algorithm performance	fast	fast	medium	medium	slow
Memory requirement	1 frame	$50-300$ frames	1 frame of mean	1 frame of k Gaussian parameters	n frames of k Gaussian parameters
Segmentation quality	worst	low	acceptable	good	best
Hardware complexity	very low	medium	low to medium	low	high

complexity. A comparison is made in [11] on segmentation qualities of some of the algorithms. In fact, an unbiased comparison with a significant benchmark is absent. No perfect system exists to handle all kinds of issues within different background models. Furthermore, for realistic implementation of such a system, trade-offs have to be made between system robustness (quality) and system performance (frame rate, resolution, etc.). From the discussion above, a summary is made in Table 1.1 concerning Algorithm performance, memory requirements, the segmentation quality and hardware complexity.

From the table, it can be seen that Stauffer-Grimson algorithm gives good segmentation quality with relatively low hardware complexity and memory requirements. In [18], a frame rate of 11-13 FPS is obtained for a frame size of 160×120 on an SGI O2 workstation. In our software implementation on an AMD 4400+ processor, a frame rate of 4-6 FPS is observed for video sequences with 352×288 resolution.

In this thesis, the hardware accelerator design is based on Stauffer-Grimson algorithm with several modifications for better segmentation quality and hardware efficiency. A variety of memory access reduction schemes are implemented, resulting in more than 70% memory bandwidth reduction.

1.3 Algorithm Modifications

The Mixture of Gaussian algorithm works efficiently only in controlled environments. Issues regarding algorithm weaknesses in different situations are addressed in many publications [19–21, 23]. In this section, instead of mainly

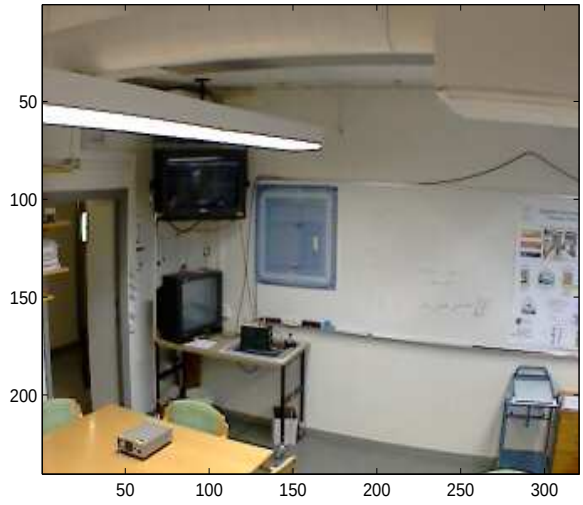


Figure 1.5: Indoor environment taken in the lab.

focusing on improving algorithm robustness, we propose several modifications to the algorithm, with a major concern on their impacts that could lead to potentially improved hardware efficiency.

1.3.1 Color Space Transformation

In theory, multi-modal situations only occur when repetitive background objects are present in the scene. However, this is not always true in practice. Consider an indoor environment where the illumination comes from a fluorescence lamp. An example video sequence of such environment is taken from our lab, which is shown in figure 1.5. In our experiments, 5 pixels are picked up evenly from the scene are measured over time. Their RGB value distributions are drawn in figure 1.6. Clearly from the figure, instead of 5 sphere like pixel distributions, the shapes of the pixel clusters are rather cylindrical. Pixel values tend to jump around more in one direction than another in the presence of illumination variations caused by the fluorescence lamp and camera jitter. This should be distinguished from the situation where one sphere distribution is moving slowly towards one direction due to slight daylight changes. Such a case is handled by updating the corresponding mean values in the original background model. Without an upper bound for the variance, the sphere de-

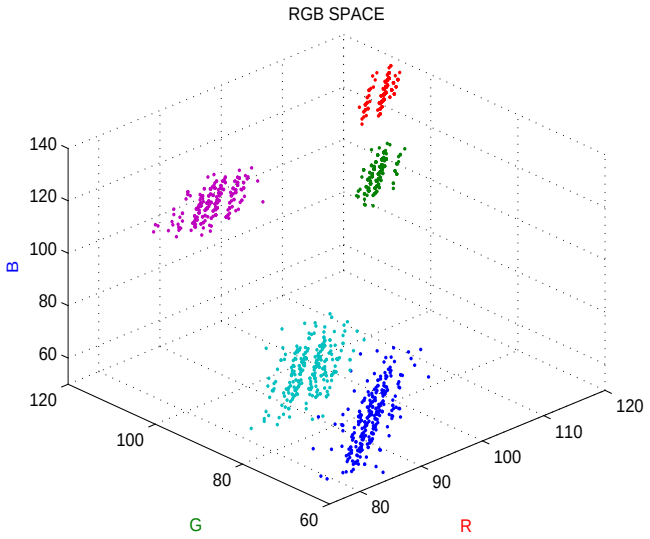


Figure 1.6: Five distributions in RGB color space.

scribing the distribution tends to grow until it covers nearly every pixel in the most distributed direction, thus taking up a large space such that most of it does not belong to the distribution (sphere A in figure 1.7). A simple solution to work around this problem is to set an upper limit for the variance, e.g. the maximum value of the variance in the least distributed direction. The result is multi-modal distributions represented as a series of smaller spheres (B-E in the same figure). Although a background pixel distribution is modeled more precisely by such a method, several Gaussian distributions are inferred which are hardware costly in terms of extra parameter updating and storage. In [27] D. Magee proposed a cylindrical model to address the issue, with primary axes of all distribution cylinders pointing towards the origin. However, more parameters are needed for each cylindrical distribution than the spherical counterpart. The coordinates of a cylindrical system comprises distance, two angles and a diameter and the height. Furthermore, it is hardware costly computation to transform RGB values to cylindrical coordinates, e.g. division and square root. In addition, not every distribution cylinder is oriented towards the origin, see the left middle distribution in figure 1.6.

To be able to model background pixels using a single distribution but without much hardware overhead, color space transformation is employed in our implementation. Both HSV and YC_bC_r space were investigated, and their

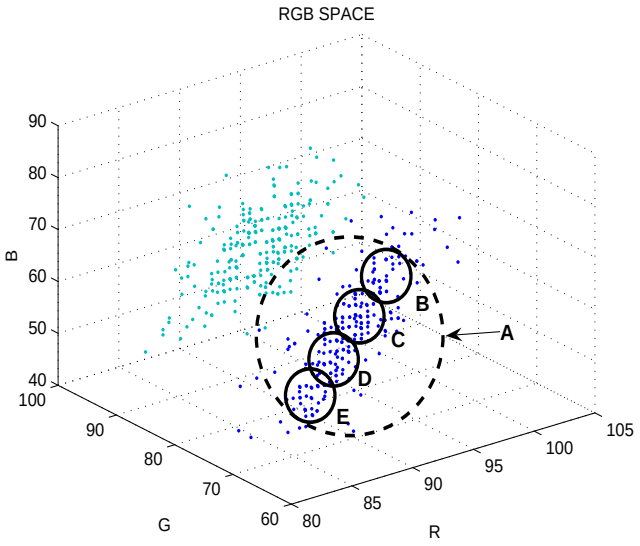


Figure 1.7: A closer look of 2 Gaussian distributions on the bottom in the left figure.

corresponding distributions are shown in figure 1.8and 1.9. By transforming RGB into YC_bC_r space, the correlation among different color coordinates are mostly removed, resulting in nearly independent color components. With varying illumination environment, only the Y component (intensity) varies accordingly, leaving C_b and C_r components (chromaticity) more or less independent. In [28], such feature is utilized for shadow reduction. Consequently, values of three independent components of a pixel in YC_bC_r color space tends to spread equally. As shown in figure 1.8, most pixel distributions are transformed from cylinders back to spheres, capable of being modeled with a single distribution. The transformation from RGB to YC_bC_r is linear, and can be calculated according to the following:

$$Y = 16 + 65.481 \times R + 128.553 \times G + 24.966 \times B \tag{1.27}$$

$$C_b = 128 - 37.797 \times R - 74.203 \times G + 112.0 \times B \tag{1.28}$$

$$C_r = 128 + 112.0 \times R - 93.786 \times G - 18.214 \times B. \tag{1.29}$$

Only minor hardware overhead with a few extra multipliers and adders are introduced, where multiplication with constant can be further utilized to reduce hardware complexity. Simplifications can be performed to further reduce the

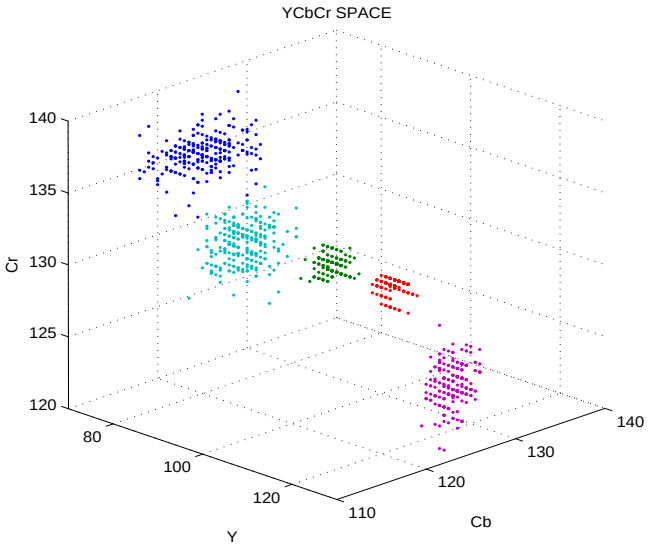


Figure 1.8: 5 Sphere distributions in YC_bC_r Space space.

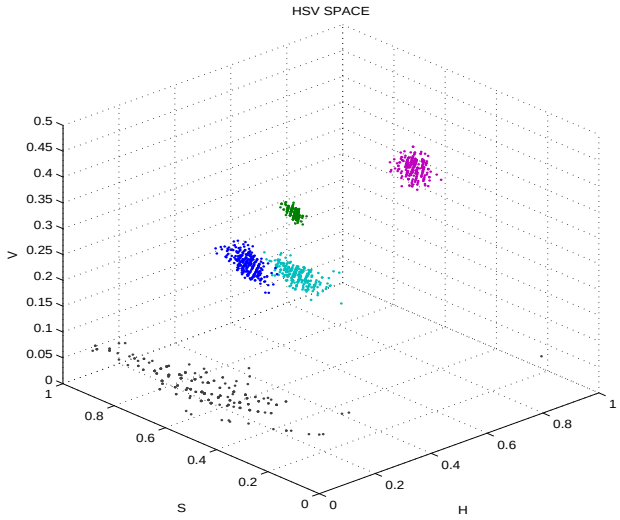


Figure 1.9: Unpredictable distributions in HSV Space.

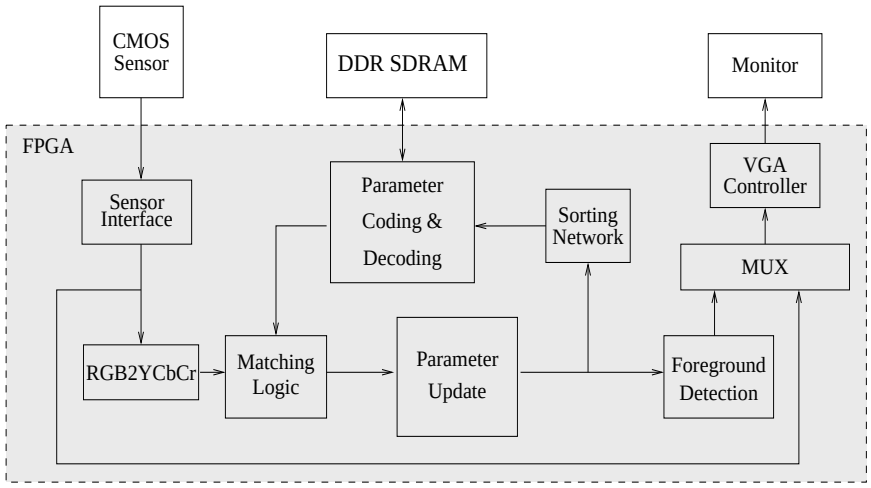


Figure 1.10: The conceptual block diagram of the segmentation unit.

number of multipliers to be 4 [29]. HSV color space, on the other hand, also with correlated coordinates, is no better than RGB color space if not worse. Unpredictable pixel clusters appeared occasionally as shown in figure 1.9, which is impossible to model using Gaussian distributions.

1.3.2 **Algorithm Simplifications**

We propose two simplifications to the algorithm. In the original algorithm specification, unbounded growing distribution will absorb more pixels. As a result, the weight of that distribution will soon dominate all others. To overcome this, in [18], all updated Gaussian distributions are sorted according to the ratio ω/σ. In this way, the distribution with dominant weight but large variance does not get to the top, identified as background distribution. In our approach, with YC_bC_r color space transformation, no upper bound is needed. All distributions can be simply sorted by their weights only, effectively eliminating division operations in the implementation.

Another simplification made in the process of foreground/background detection is instead of using equation 1, the determination can be made by checking the weight of each distribution only. This is due to the fact that one pixel cluster will not spread out in several distributions by color space transformation to YC_bC_r. The equation that determines the background distribution is changed to

$$B = argmin_k \left(\omega_k > H\right). \tag{1.30}$$

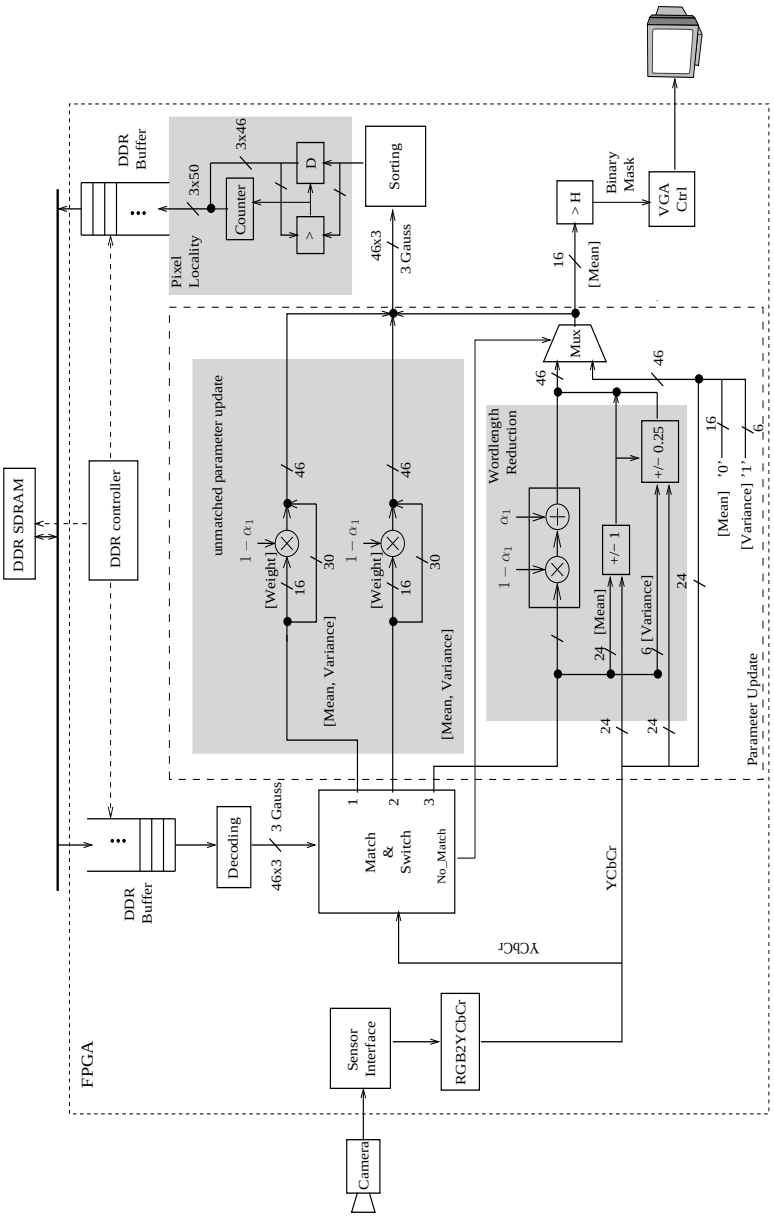


Figure 1.11: The system architecture of the segmentation unit.

This results in automatic single or multi modal background model without having to adjust the value of H.

1.4 Hardware Implementation of Segmentation Unit

To perform the algorithm with VGA resolution in real-time, a dedicated hardware architecture, with a streamlined data flow and memory bandwidth reduction schemes, is implemented to address the computation capacity and memory bandwidth bottlenecks. Algorithm modifications covered in previous sections are implemented with potential benefits on hardware efficiency and segmentation quality. This is a large improvement to the previous work [30], where only 352×288 resolution is achieved without any memory reduction schemes and algorithm modifications. In this section, a thorough description of the whole system architecture of the segmentation unit is given, followed by detailed discussions on memory reduction schemes.

To help for a better understanding of the architecture, a simplified conceptual block diagram of the whole system is given in figure 1.10 to illustrate the dataflow within the system. From the figure, the whole system starts with CMOS image sensor capturing video sequence in real-time and feeding it to the system through a sensor interface. The interface is designed to be able to control the parameters of the image sensor on run-time, e.g. analog gain and integration time, so that better image quality can be obtained within different environments. The sensor interface is also responsible for sampling the image data transferred from off-chip. In our implementation, an over sampling scheme by higher clock frequency (100Mhz) is used to ensure the accuracy of the image data. The image data is captured with one color component at a time, and three complete color components are sent to the system after serial-parallel transform. To handle different clock frequencies, input FIFOs, implemented as distributed RAMs, are used to interface to both the segmentation logic and the VGA controller where the original video data can be monitored on a screen. The image data from the sensor are RGB values, and need to be converted into YC_bC_r components before entering the segmentation logic block according to the algorithm modification above. Each pixel has a series of corresponding Gaussian distributions, where are stored on off-chip memories (DDR SDRAM) due to its size. With all the Gaussian parameters read from the DDR SDRAM and decoded by the parameter encoder/decoder, a match is calculated in the matching logic block to check if the incoming pixel matches any of the existing distributions. The match and switch block is composed of mainly comparators and multiplexer so that any the matching Gaussian parameters are multiplexed to a specific output port. The output from the matching block is reordered Gaussian distributions with the matching distribution switched to a specific port. The Gaussian distributions are updated according to the algo-

rithm modifications mentioned above, from where the foreground/background detection can start by checking the weight of the updated matched Gaussian distribution. The output is a binary stream to be multiplexed to the monitor, indicating foreground and background pixels with white and black colors. The updated Gaussian parameters have to be sorted for use in the next frame, and all distributions should be ordered according to their weight. This is implemented in a dedicated sorting network that will be covered in more details in the following section. To reduce heavy memory bandwidth incurred by accessing off-chip DDR SDRAM that stores one frame of Gaussian distributions, an encoding/decoding block is designed by utilizing pixel localities in succeeding neighboring pixels. This is covered in more detail in Section 1.4.3.

In the following sections, implementation details of the architecture shown in figure 1.11 are explained with an emphasis on the parts with algorithm modifications, which are indicated by shaded area. From the figure, with the image data captured and transformed into YC_bC_r color space as explained earlier, the match and switch block tries to match the incoming pixel with Gaussian distributions taken from the previous frame. To avoid the competition several Gaussian distributions matching the incoming pixel, only the one with highest likelihood (large weight) is selected as the matching distribution. A matched Gaussian is switched to the bottom(3 in the figure). In case no match is found, a No_match signal is asserted for use in later blocks. Depending on if there is a match for a distribution, each distribution is updated accordingly. For the matched Gaussian distribution, a proposed updating scheme is implemented with only adders/subtractors for the mean and variance values. Depending on whether the incoming YC_bC_r value is larger than the mean values, a addition or subtraction is applied for parameter updating. Similar updating schemes are utilized for variance update. The proposed parameter update results in low hardware complexity by replacing the hardware costly operations in equation1.20 and 1.21, e.g. square and multiplication with large wordlength with incrementors/decrementors. Other benefits of the proposed updating schemes are described in detail in section 1.4.2. For the case that no match is found, a MUX is used together with a No_match signal to update all parameters for the distribution (3 in the figure) with predefined values.

1.4.1 **Sorting**

The updated Gaussian parameters have to be sorted for use in the next frame. In order to reduce hardware complexity found in parallel sorting networks, such as [31] [32] [33], while still maintaining the speed, a specific feature in the algorithm is explored. By observing that only one Gaussian distribution is updated at a time and all the distributions are initially sorted, the sorting of N Gaussian distributions can be changed to rearranging an updated distribu-

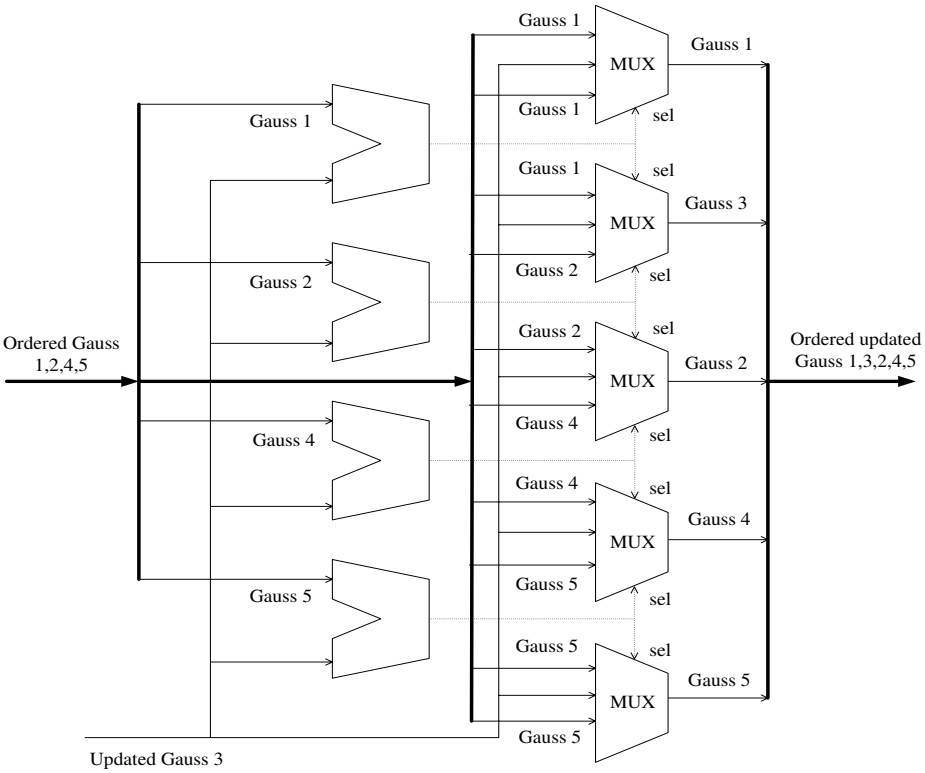


Figure 1.12: The sorting architecture. Five Gaussian are sorted according to their weight. The architecture scales well with increasing number of Gaussian due to its fixed logic level (one comparator and one MUX).

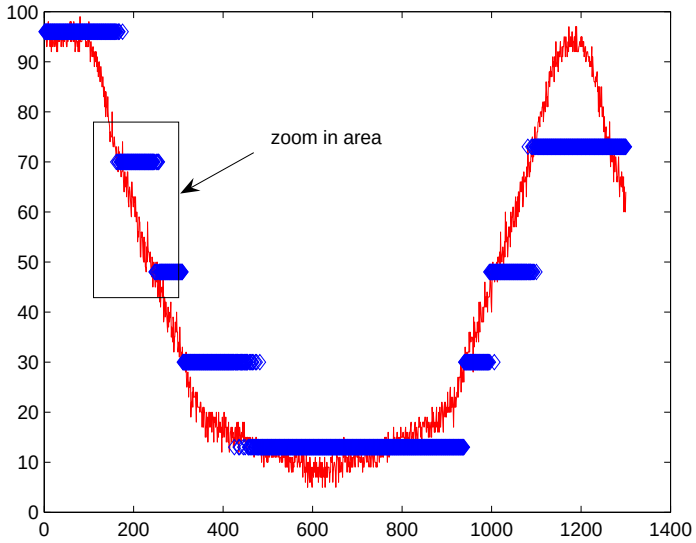
Table 1.2: Hardware complexity within different sorting architecture, where p is $\log_2 N$.

Scheme	Nr. of Comparators	Nr. of Stages
Proposed	$N-1$	2
[31]	$(p^2-p+4)2^{p-2}-1$	$(\frac{1}{4})(N)(\log_2 N)^2$
Odd-even trans. sort	$O(N)$	$O(N^2)$
Bitonic Sort	$O(N \log N)^2$	$O(\log N)^2$

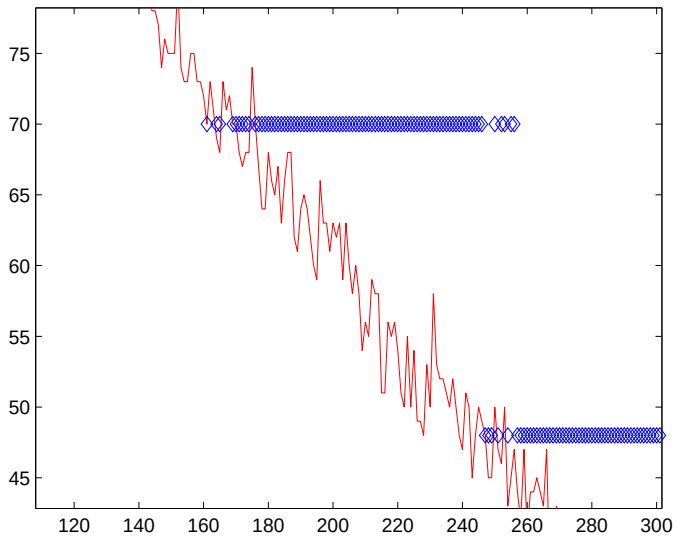
tion among N-1 ordered distributions. As a result, both the number of sorting stages and the number of comparators are reduced to only one sorting stage with N-1 comparators and N MUXes. This results in both increased speed and reduced area. The architecture for the sorting network is shown in figure 1.12. From the figure, all unmatched ordered Gaussian distributions are compared with the updated distribution (3 in the figure), and the output of each comparators signifies which distribution is to be multiplexed to the output, e.g. if the weight of any unmatched distribution is smaller than the updated one, all unmatched distributions below the current one is switched to the output at the next MUX below. Such an architecture scales very easily to support sorting more Gaussian distributions. The number of stages will not increase with the number of Gaussian distributions to be sorted. Due to memory bandwidth limitation, only 3 Gaussians needed to be sorted in our implementation which makes it trivial task. However, if in future implementations with more Gaussians per pixel is wanted, e.g. 9 Gaussians, such an architecture will be useful to reduce hardware complexity. A comparison of hardware complexity between proposed sorting architecture and other schemes mentioned above is shown in table1.2. The foreground detection is achieved by simply comparing the weight of the distribution on the bottom with a predefined parameter H according to the simplifications made in previous sections. All sorted Gaussian parameters are encoded and decoded before and after writing to and reading from a DDR SDRAM, with manipulated data flow controlled by a custom made DDR controller. The encoding and decoding scheme is explained in details in section 1.4.3. A sequence of binary data indicating background and foreground is multiplexed out to a monitor through a VGA controller.

1.4.2 **Wordlength Reduction**

Slow background updating requires large dynamic range for each parameter in the distributions. This is due to the fact that parameter values are changed slightly between frames, but could accumulate over time. In this section, parameter wordlength reduction is investigated for potential memory bandwidth

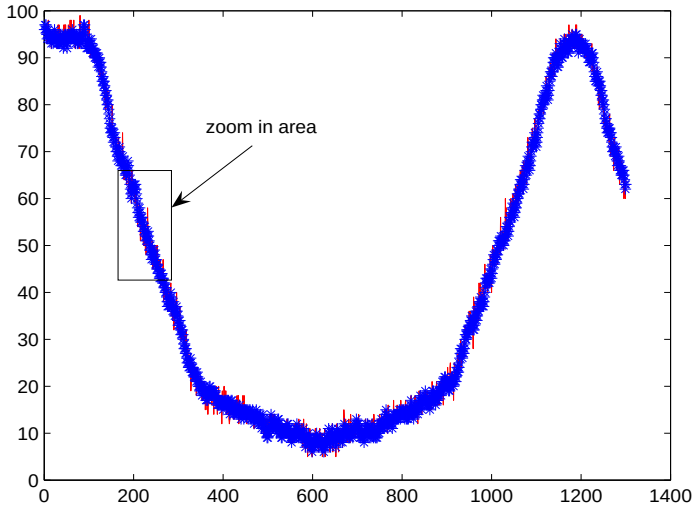


(a) The original parameter updating scheme in presence of relatively fast light changes. Red color component values (solid line) of a pixel over frames are plotted together with updated mean values for the red color channel (diamond line).

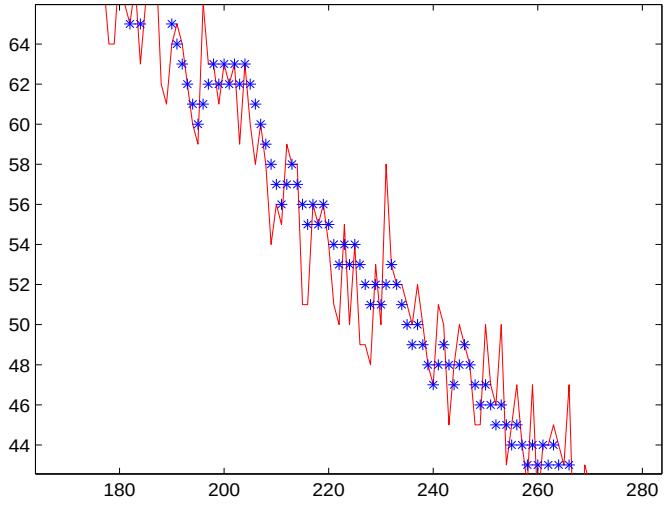


(b) A closer look at the zoom in area in the the figure above.

Figure 1.13: The results from the original parameter updating scheme.

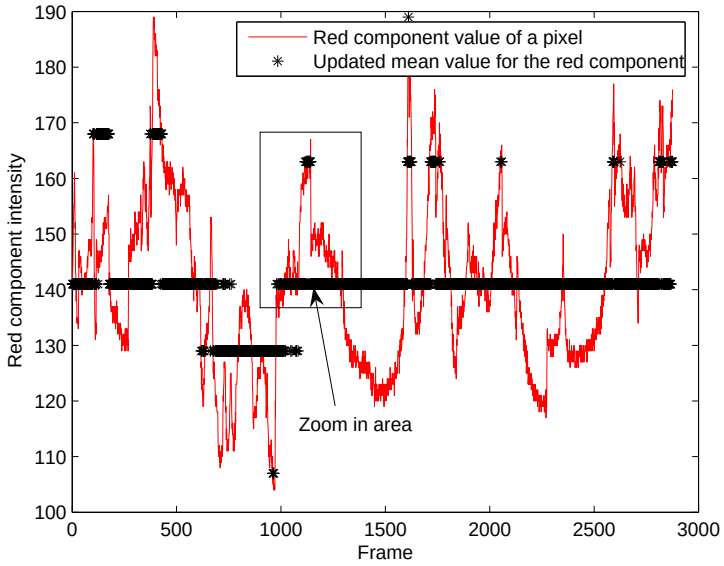


(a) The proposed parameter updating scheme in presence of relatively fast light changes. Red color component values (solid line) of a pixel over frames are plotted together with updated mean values for the red color channel (star line).

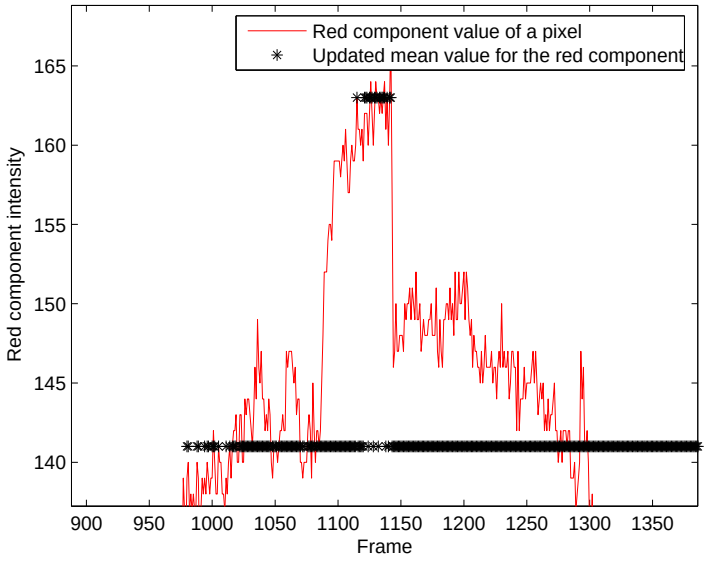


(b) A closer look at the zoom in area in the figure above

Figure 1.14: The results from the proposed parameter updating schemes.

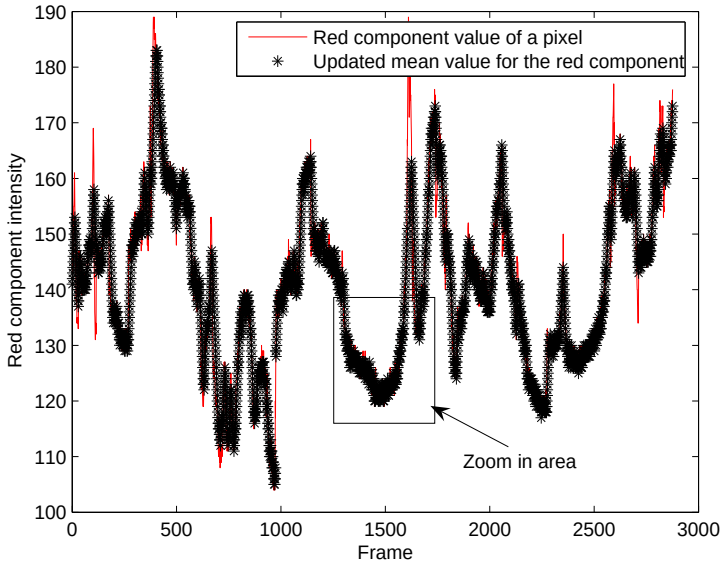


(a) The original parameter updating scheme in presence of fast light changes.

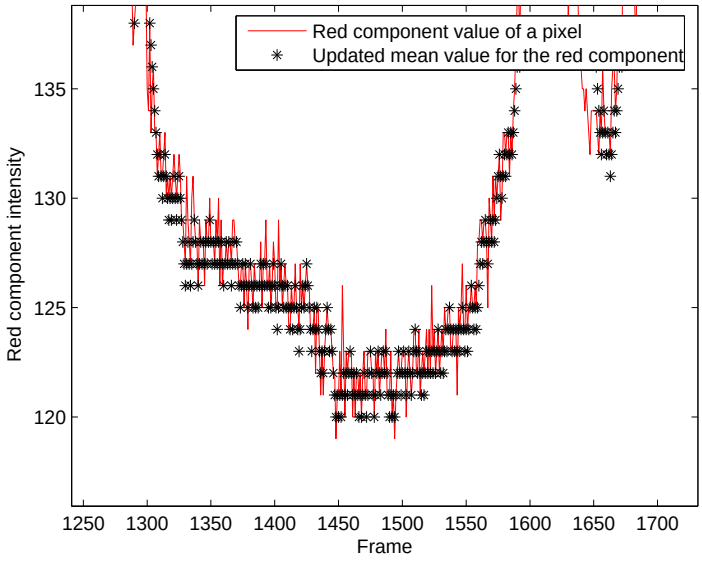


(b) A closer look at the zoom in area in the original updating scheme.

Figure 1.15: The results from the original parameter updating scheme.

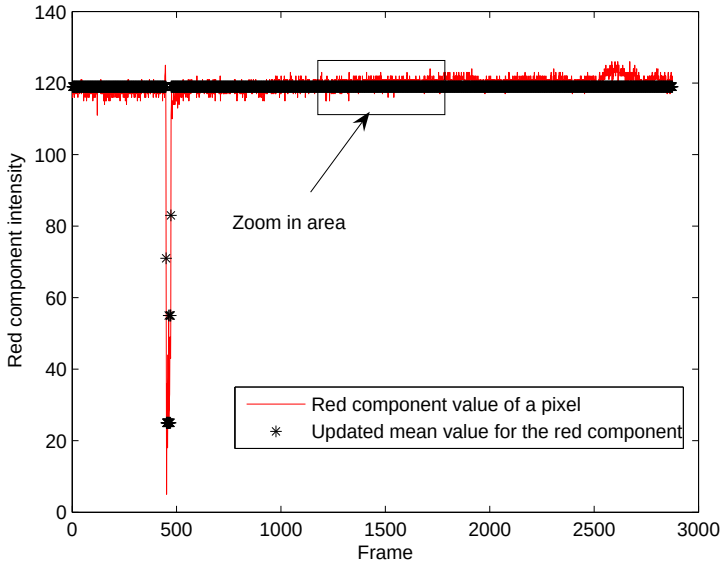


(a) The original parameter updating scheme in presence of fast light changes.

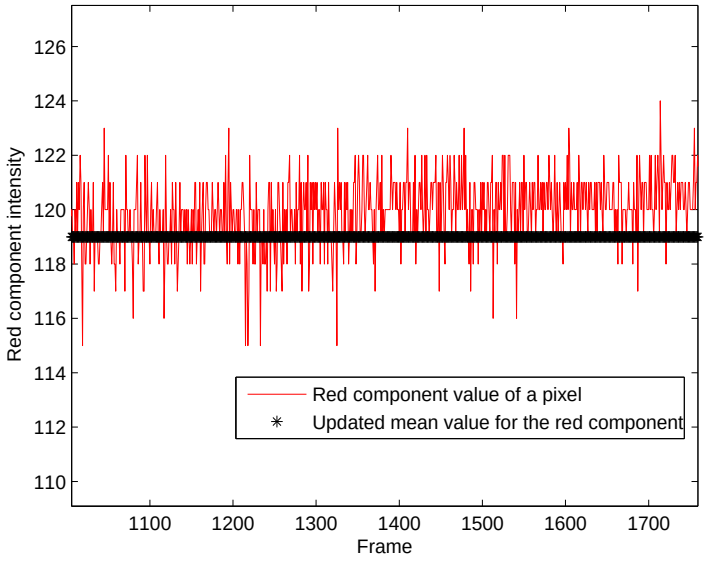


(b) A closer look at the zoom in area in the original updating scheme.

Figure 1.16: The results from the original parameter updating scheme.

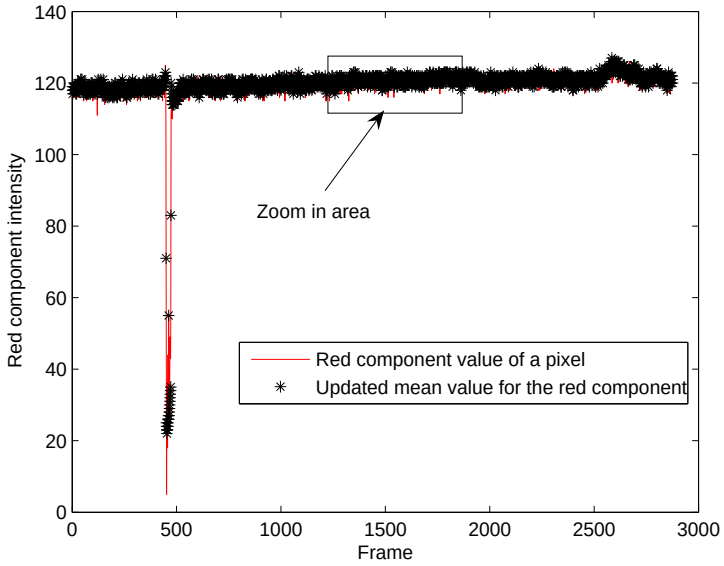


(a) The original parameter updating scheme in presence of minor light changes.

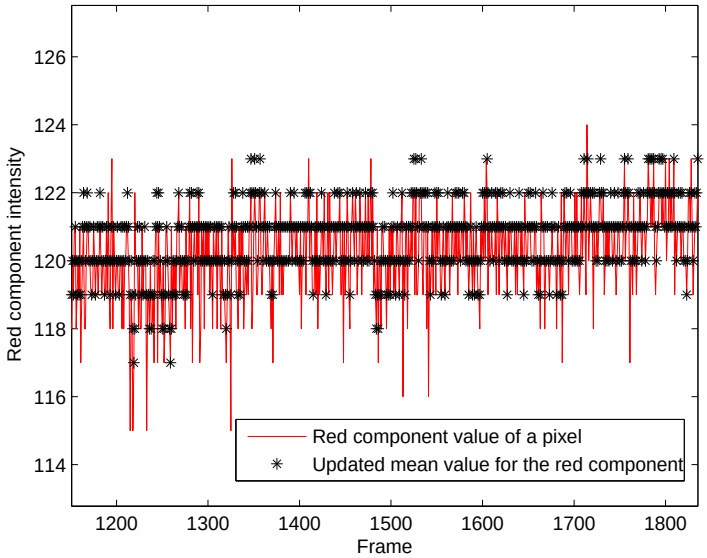


(b) A closer look at the zoom in area in the original updating scheme.

Figure 1.17: The results from the original parameter updating scheme.



(a) The original parameter updating scheme in presence of minor light changes.



(b) A closer look at the zoom in area in the original updating scheme.

Figure 1.18: The results from the original parameter updating scheme.

reduction.

According to equations 1.20 and 1.21, the mean and variance of a Gaussian distribution is updated using a learning factor ρ. The difference of mean and variance between current and previous frame is derived from the equation as

$$\Delta\mu = \mu_t - \mu_{t-1} = \rho(X_t - \mu_{t-1}) \qquad and \tag{1.31}$$

$$\Delta\sigma^2 = \sigma_t^2 - \sigma_{t-1}^2 = \rho((X_t - \mu_t)^T(X_t - \mu_t) - \sigma_{t-1}^2). \tag{1.32}$$

Given a small value for ρ, e.g. 0.0001, a unit difference between the incoming pixel and the current mean value results in a value of 0.0001 for Δ_μ. To be able to record this slight change, 22 bits have to be used for the mean value, where 14 bits accounts for the fractional part and 8 bits are used for the integer one. Less bits can be achieved by ignoring small deviations of the incoming pixel from current mean, while picking up only large ones. The extreme case is when only the largest deviation is picked, e.g. where the incoming pixel is in the range of J times standard deviation off the current mean. Larger than that, the incoming pixel will not match the current distribution. With an upper bound for the variance, e.g. 16, a maximum value of $0.0001 \times 2.5 \times \sqrt{16} = 0.001$ is derived for Δ_μ, which can be represented by 10 bits. Using a wordlength lower than that, no changes would be recorded ever. In practice, the bits for fractional parts should be somewhere in the range of 10-14 bits. With similar calculations, 7-14 bits are obtained for the fractional parts of the variance. Together with 16 bits weight and integer parts of the mean and the variance, 81-100 bits are needed for a single Gaussian distribution. To reduce this number, a wordlength reduction scheme is proposed. From equation 1.31, a small positive or negative number is derived depending on whether the incoming pixel is larger than the current mean. Instead of adding a small positive or negative fractional number to the current mean, a value of 1 or -1 is added. The overshooting caused by such coarse adjustment could be compensated by the update in the next frame, e.g. without illumination variation, the mean value will fluctuate with a magnitude of 1. This is negligible since Gaussian distribution is usually a sphere with a diameter of more than 10 from our experiment data. In a relatively fast varying illumination environment, e.g. 25 RGB value changes in a second, fast adaptation to new lighting conditions is also enabled by adding or subtracting ones in consecutive frames. Figure 1.14(a) shows the experimental results of the coarse updating in a varying lighting room, where the light is manually turned up and down. The parameter updating scheme specified in the original algorithm is also drawn in figure 1.13(a) for comparison. A closer look of the two schemes are given in figure 1.14(b) and 1.13(b). From figure 1.13(a) and 1.13(b), it is clear that parameter updating does not work well in the presence of fast light changes. As slow parameter updating (diamond line in the figure) for each Gaussian distribution will not keep track of the pixel value

changes (solid line in the figure) in fast light changing environment, that Gaussian distribution will finally not be able to match the incoming pixel values, in which case Gaussian distribution replacement takes place in stead of parameter updating. The coarse updating scheme on the other hand relieves the problem to certain extent, where consecutive ones are added or subtracted to keep track of the relatively fast changes. Similar results are obtained from two other test sequences, namely scene "trees" and "parklot" which are shown in figure 1.22(a) and 1.23(a). These two scenes are both taken in outdoor environments. Scene "trees" records many relatively fast light changes due to it is dynamic background, and scene "parklot" consists of nearly constant illumination condition. The results from both parameter updating schemes for both sequences are shown in figure 1.15,1.16, 1.17 and 1.18. From these figures, it can be seen that the proposed coarse updating scheme works fine in both situations. It keeps track of the relatively fast value changes in the dynamic scene while fluctuates around a constant value in the latter static scene. However, with the primary goal to reduce wordlength, the coarse parameter updating scheme results in limited improvements to the segmentation results. Nearly no visual difference can be observed in the segmented results from the two schemes.

With coarse updating, only integers are needed for mean specification, which effectively reduce the wordlength from $18-22$ down to 8 bits. Similar approach can be applied to the variance (a step value of 0.25 is used instead), resulting in a wordlength of 6 bits, where 2 bits account for fractional part. Together with the weight, the wordlength of a single Gaussian distribution can be reduced from $81-100$ to only 44 bits, over 43% reduction is accomplished even compared to the extreme case mentioned above in the normal updating scheme, where only 81 bits are used for each Gaussian distribution. In addition, less hardware complexity is achieved as a bonus since multiplication with the learning factor of ρ is no longer needed.

Thus, the proposed scheme enhance the algorithmic performance while at the same time reduce both memory bandwidth and computational complexity.

1.4.3 **Pixel Locality**

In addition to wordlength reduction, a data compression scheme for further bandwidth reduction is proposed by utilizing pixel locality for Gaussian distributions in adjacent areas. We classify "similar" Gaussian distributions in the following way: from the definition of a matching process, each Gaussian distribution can be simplified as a three dimensional cube instead of a sphere in the YC_bC_r color space. The center of the cube is composed of YC_bC_r mean values whereas the border to the center is specified by J times variance. One way to measure the similarity between two distributions is to check how much of the two cubes that overlap. If the overlap volume takes up certain percent-

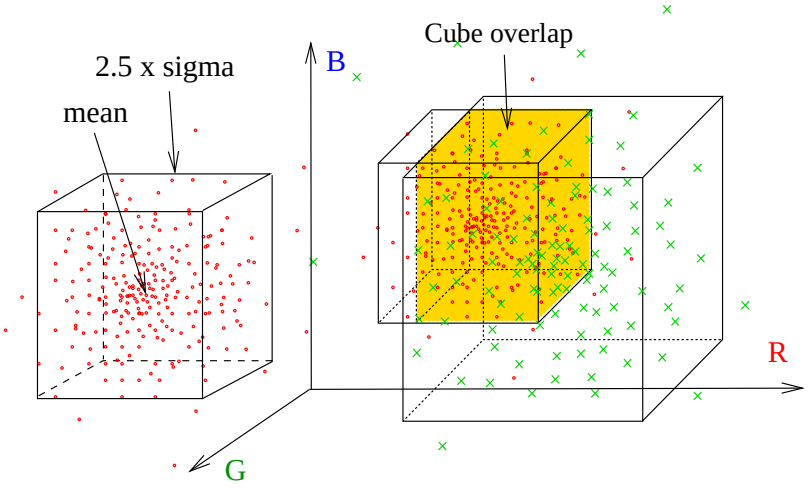


Figure 1.19: Gaussian distribution similarity as modeled by cube overlapping.

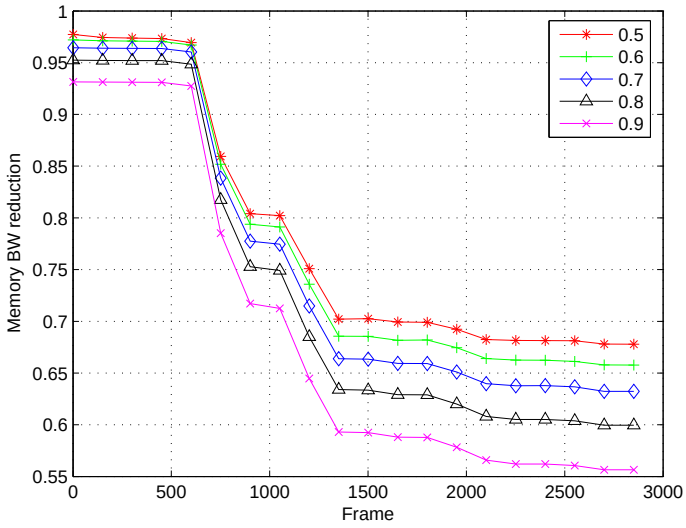
age of both Gaussian cubes, they are regarded as "similar". The whole idea is illustrated in figure 1.19. The reason for such a criteria lies in the fact that a pixel that matches one distribution will most likely match the other, if they have enough overlapping volume. The percentage is a threshold parameter that can be set to different values among different situations.

In the architecture, two similar distributions are treated as equivalent. By only saving non overlapping distributions together with the number of equivalent succeeding distributions, memory bandwidth is reduced. Various threshold values are selected to evaluate the efficiency for memory bandwidth reduction.

Table 1.3: Design Summary

FPGA Utilization			
Nr. of Slices	Nr. of Flip Flops	Nr. of DCMs	Nr. of BRAMs
6107	4273	5	84
Clock Domains			
100Mhz		16Mhz	25Mhz
Sensor interface & DDR controller		Segmentation	VGA controller
System Parameters			
Resolution	Throughput	Frame rate	Nr. of Gaussians
640×480	$170MB/s$	25	3

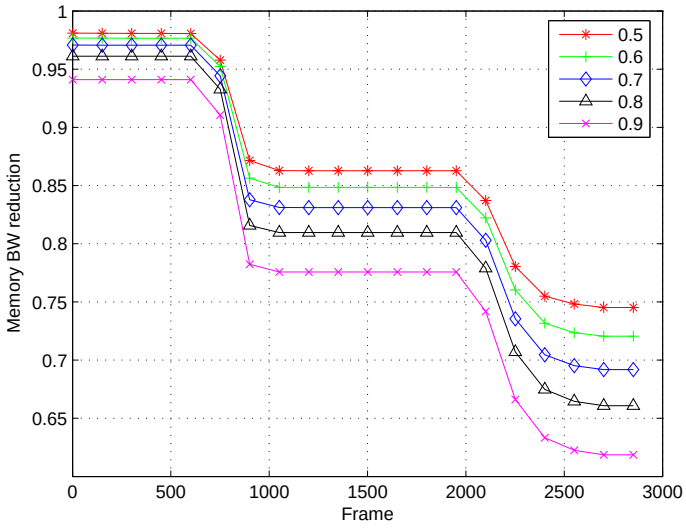
(a) Scene "stairs"



(b) Memory reduction in Scene stairs

Figure 1.20: Memory reduction results tested in Scene "stairs". Video sequences are provided by AXIS Communications [34].

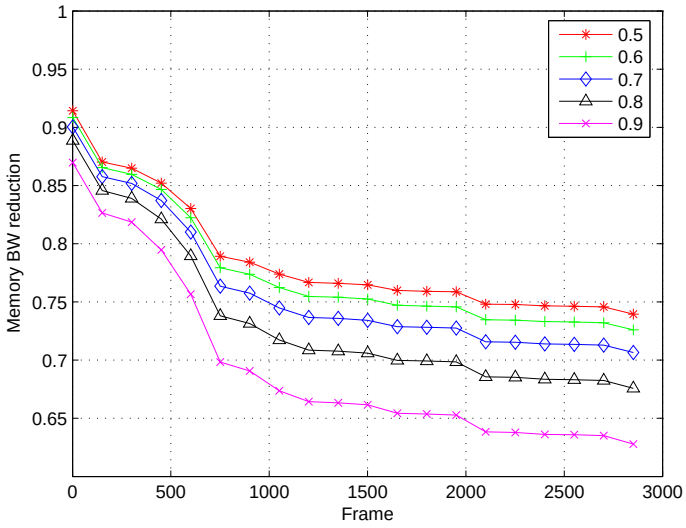
(a) Scene "hallway"



(b) Memory reduction in Scene hallway

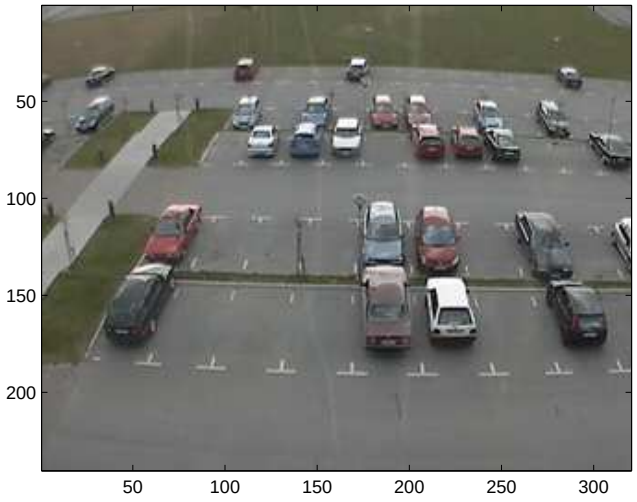
Figure 1.21: Memory reduction results tested in Scene "hallway". Video sequences are provided by AXIS Communications [34].

(a) Scene "trees"

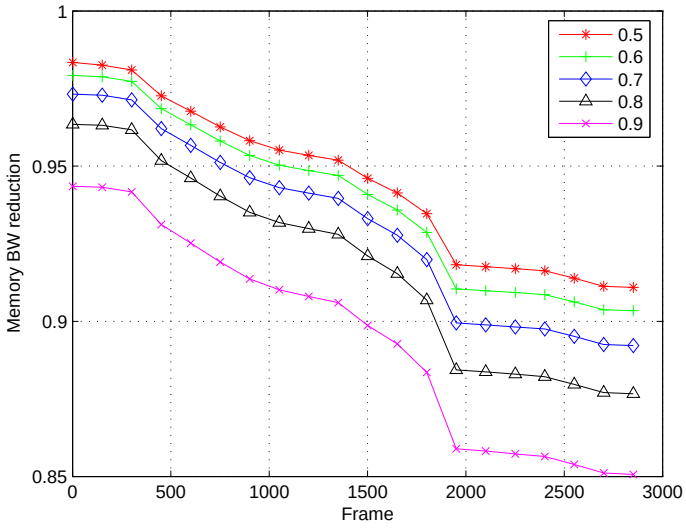


(b) Memory reduction in Scene trees

Figure 1.22: Memory reduction results tested in Scene "trees". Video sequences are provided by AXIS Communications [34].



(a) Scene "parklot"



(b) Memory reduction in Scene parklot

Figure 1.23: Memory reduction results tested in Scene "parklot". Video sequences are provided by AXIS Communications [34].

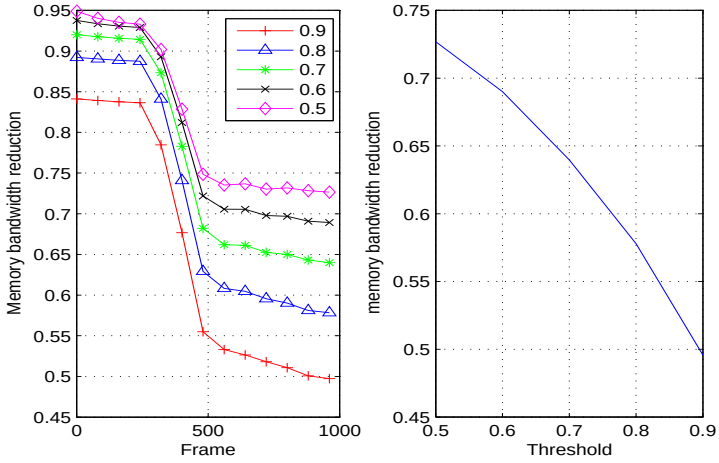


Figure 1.24: Memory bandwidth reductions achieved in the video sequences from our lab. Left: memory bandwidth reduction over frames. Right: Memory bandwidth reduction over threshold value.

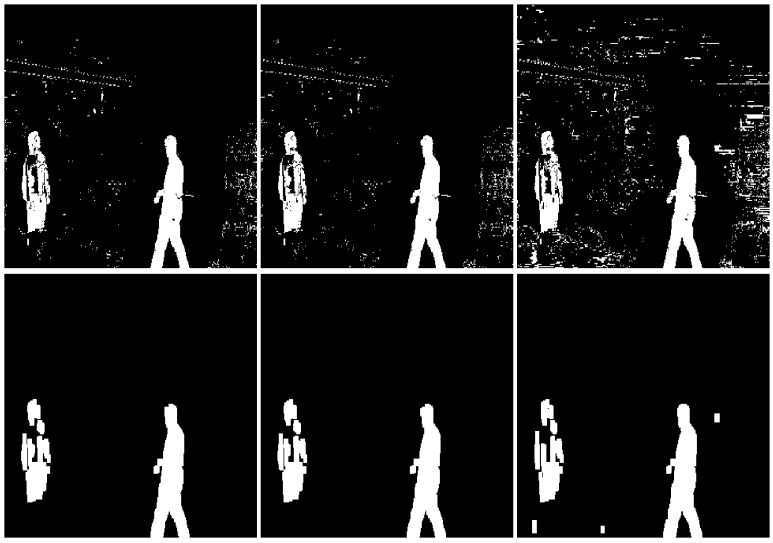


Figure 1.25: The results before and after morphological filtering for different thresholds: (left) Original result, (middle) Threshold of 0.8, (right) Threshold of 0.4.

With a low threshold value where less overlapping Gaussian distributions are regarded as the same, more savings could be achieved. However, more noise is generated due to increasing mismatches in the matching block. Fortunately, such noise is found non-accumulating and therefore can be reduced by later morphological filtering [35]. Figures 1.20,1.21,1.22,1.23 show the memory bandwidth savings over frames with various threshold values. The simulation results are obtained from matlab, with four video sequences provided by AXIS [34] are evaluated for both indoor and outdoor scenes. The sequences are selected to reflect a range of real-world background environments with possible difficulties for many segmentation and tracking algorithms. The scene "stairs" comprises people moving up and down the stairs randomly. Gradual illumination changes together with shadows are the major disturbing factors. The scene "hallway" focus on the scenarios with people moving closer or further away from the camera. The foreground object size is varying over time. The scene "trees" address the issue of quasi-static environments where a swaying tree is present as the dynamic background object. The scene "parklot" presents a environment with walking people, moving cars of different size. Gradual illumination as well as waking foreground object are also within the focus. It can be seen from the sequences, memory reductions scheme works robustly within different real-world environments with variation only in the beginning due to varied foreground actives. During initialization phase, only background pixels are present, which exhibit high similarity within neighboring pixels. With foreground objects entering the scene, part of Gaussian distributions are replaced, which results in the decrease of number of similar Gaussian distributions. The trends will continue until it reaches a certain point where most pixel locations contains a foreground distribution. The decrease will flattens out in the end since more foreground objects always replace the distribution that represent a foreground pixel. Foreground objects activities can vary in different video scenes, e.g. continuous activities in figure 1.20(a) where people going up and down the stairs all the time, and the two peak activity periods around frames $600-900$ and frames $2100-2500$ in figure 1.21(a), where people walking by in two discrete time period. In the long run, the bandwidth savings tends to stabilize (around $50\%-75\%$ depending on threshold value) after the initialization phase. Another test sequence is also experimented in our lab. Similar results are observed as shown in figure 1.24. The quality of segmentation results before and after morphology are shown in figure 1.25, where it is clear that memory reduction comes at the cost of segmentation quality. Too low threshold value results in clustered noises that would not be filtered out by morphological filtering. In this implementation, a threshold value of 0.8 is selected, combined with wordlength reduction scheme, a memory bandwidth reduction of over 70% is accomplished. To evaluate long term effects of memory bandwidth reduction scheme, FPGA platform is required to collect data in real time.

Table 1.4: Hardware complexity for different blocks within segmentation unit.

Logic Block	Nr. of Slices (%)	Nr. Block RAMs
Match	253 (4%)	0
Switch	652 (11%)	0
Parameter Update	483 (8%)	0
Pixel Locality	1530 (25%)	0
Sensor Interface	540 (9%)	3
Sorting	355 (6%)	0
DDR controller	1599 (26%)	24
$RGB2YC_bC_r$	181 (3%)	0
VGA controller	377 (6%)	57

1.5 System Integration and FPGA Prototype

The segmentation unit is prototyped in an Xilinx VIrtexII vp30 development board, as shown in figure 1.26. The board come with one DDR memory slot with customized on board signal traces that minimizes the skew between different signals. Virtex II vp30 is equipped with two on-chip PowerPC embedded processor cores. The number of on-chip block RAMs is 136, with 2448 Kb in total. A custom-made PCB extension board is mounted on the FPGA board to support image data read in through a image sensor. A dedicated VGA controller is developed streaming output data into a monitor, where the results from different stage of the logic can be monitored. Detailed description of the system integration of the whole tracking system is covered in the next chapter. In table 1.4, a summary of the hardware complexities of different blocks in the segmentation is given. It can be seen that algorithm modifications results in a low hardware complexity in parameter updating block, which merely occupies 8% of the total utilized resources. However, memory reduction scheme with pixel locality incurs relatively costly in hardware, where many multiplications are needed to calculate the cube volumes. DDR controller contributes to a large part of the whole design resources, due to complicated memory command and data signal manipulations, clock schemes, and buffer controls. The hardware complexity of sorting and color space transformation is low by after optimizations. VGA controller consumes most of the on-chip memory resources. Dual-port block RAMs are used as video RAMS in the VGA controller, which are shared by different blocks of the whole surveillance system to display the results from different stages on a monitor . Block RAMs are also used as data buffers to support DDR burst read and write operations.

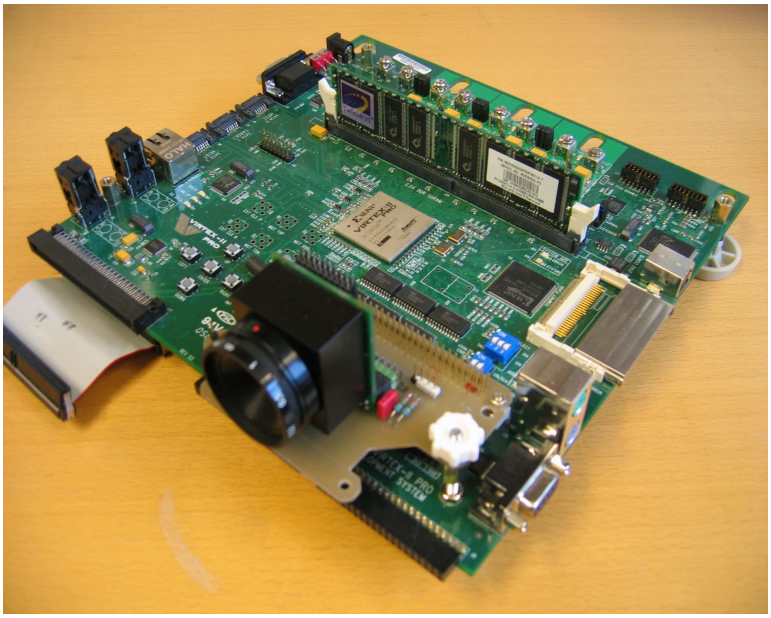


Figure 1.26: FGPA prototypes.

1.6 Results

The system is implemented on a Xilinx VirtexIIpro vp30 FPGA [36] development board as shown in figure 1.26. The whole design is partitioned into 3 clock domains for different blocks, and asynchronous FIFOs are used to interface blocks working with different clocks. A KODAK KAC-9648 CMOS sensor [37] is used to capture color images stream into the FPGA platform. Real time segmentation performance is achieved on video sequences with 3 Gaussian distributions per pixel. With the proposed memory reduction schemes, off-chip memory bandwidth is reduced by more than 70%. A summary of the whole design is given in Table 1.3.

1.7 Conclusions

A Real-time video segmentation unit is implemented on an Xilinx VirtexII FPGA platform, serving as a real-time testbench for evaluating long term effects of the corresponding algorithm. The segmentation unit is an important part of the whole tracking system developed at the department. With real time performance, tracking schemes can be evaluated in varied environments

for system robustness testing . Software/hardware partition is facilitated in an FPGA platform, where high level operations e.g. features extraction in a tracking algorithm can be implemented in software. The calculation and memory intensive tasks e.g. segmentation, morphology are left to customized hardware architecture. For the implementation of the hardware units, memory usage is identified as the main bottleneck of the whole system, which is is common in many image processing systems. This is especially true for segmentation since most algorithms operates in pixel-wise structures. To address the issue a joint memory reduction scheme is proposed by utilizing pixel locality and wordlength reduction. By measuring similarity of neighboring gaussian distributions with overlapping volume of two cubes, threshold can be set to classify Gaussian similarities. Wordlength reduction is as important for memory bandwidth reduction. By utilizing coarse parameter updating scheme, wordlength for each Gaussian parameters is reduced by more than 43%, which effectively decrease the memory bandwidth to off-chip memories. The substantial bandwidth reduction comes at the cost of segmentation quality. By setting a threshold value to a low value, noise generated can not be removed by morphology operation. Careful tradeoffs should be made based on different application environments. Algorithm modifications are of great importance for the efficiency of the hardware implementation. By utilizing YC_bC_r color space, several simplifications can be made that results in potential hardware savings.

Chapter 2

System Integration of Automated Video Surveillance System

In this chapter, an original paper is presented for system integration of a real-time video surveillance embedded system:

> Fredrik Kristensen, Hugo Hedberg, Hongtu Jiang, Peter Nilsson, and Viktor Öwall, "Hardware Aspects of a Real-Time Surveillance System" , in Proc. of The Sixth IEEE International Workshop on Visual Surveillance.

The author's contribution is on the segmentation part. Many parts of the paper have been changed since the publication. This work is currently being updated and revised. A journal paper is in preparation to be submitted in February, 2007.

Abstract

This paper presents the implementation of an automated digital video surveillance system for real-time performance on an embedded platform. To achieve real-time performance, the system includes hardware accelerators for video segmentation, morphological operations, labeling and feature extraction while tracking is handled in software. By implementing a complete system on an embedded platform, bottlenecks in computational complexity and memory requirements can be identified and addressed. A memory access reduction scheme for the video segmentation part that utilizes pixel locality is proposed which shows the potential of reducing accesses with >60%. It is also shown how a low-complexity addition to the segmentation rule can reduce the affect of shadows. Furthermore, a low complexity morphology architecture with low memory requirements is presented together with a labeling unit based on a contour tracing technique. The system is running in real-time at an FPGA development board with a resolution of 320×240 at 25 frames per second.

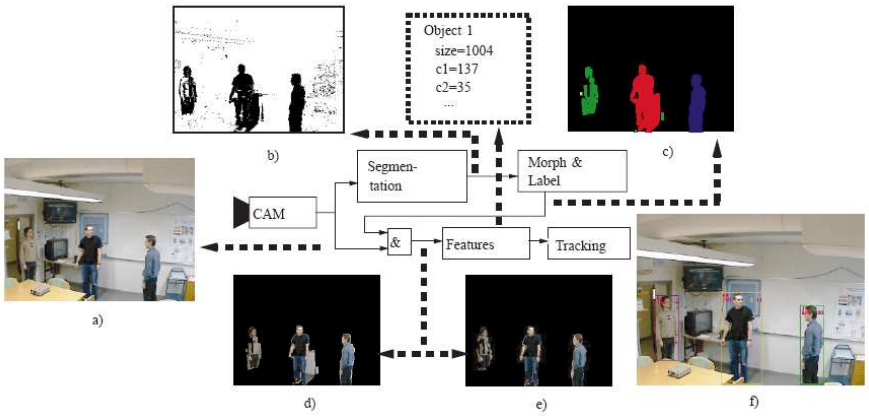


Figure 2.1: Surveillance system, (a) original image, (b) binary motion mask, (c) filtered and labeled motion mask, (d) detected objects, (e) detected objects improved result, and (f) tracking results

2.1 Introduction

In this paper we outline an attempt to automate digital surveillance running on embedded platforms in real-time. The goal is to hardware accelerate computationally complex parts of a self contained intelligent surveillance camera that can track moving objects. The demands on video surveillance systems are rapidly increasing regarding parameters such as frame rate and resolution. Furthermore, with an ever increasing data rate together with an increased number of video streams an automated process for extracting relevant information is required. Due to the large amount of input data and the computational complexity of the algorithms, software implementations are not yet sufficient to sustain real-time performance on embedded platforms. Therefore, algorithms that are well suited to be implemented as dedicated hardware accelerators with streamlined dataflow are required. The presented hardware platform has been developed with the goal of identifying computational and memory access bottlenecks. Furthermore, when proposing modifications to the original algorithms to overcome these bottlenecks extensive simulations are needed, especially if long-term effects in the video sequences can be envisioned. Utilizing a reconfigurable platform based on a Field Programmable Gate Array (FPGA) reduces the simulation and development time considerably.

A conceptual overview of the surveillance system is shown in Fig. 2.1. The

camera feeds the image processing system with a real-time image stream of 25 frames per second (fps). A segmentation algorithm, in this case Gaussian Mixture background Model (GMM), preprocesses the image stream and produces a binary mask in which zeros and ones correspond to background and foreground, respectively. In theory, only the moving parts of an image should be distinguished as independent objects in the binary mask. However, in reality the mask will be distorted with noise and single objects are shattered, e.g., if parts of the moving object has the same color as the background. In order to remove noise and merge shattered objects, one or more morphological operations are performed on the mask. These morphological operations will produce a frame of connected clusters which have to be identified, i.e. labeled, and features should be extracted to be sent as input to the tracking algorithm.

The main bottlenecks of the segmentation algorithm, shared with many other image based algorithms, are the high memory bandwidth and the overall memory requirements. A reduced memory access scheme is proposed, which introduces coding of the Gaussian parameters using pixel locality. A reduction of >60% is indicated but results in increased noise in the binary mask. However, this noise is non-accumulating and can be reduced by the morphology operation at no extra cost. Long term effects of the memory reduction scheme as well as more advanced version have to be further investigated. Furthermore, it has been investigated how different color spaces affect the segmentation result in terms of noise and shadow sensitivity. A low complexity and low memory requirement architecture for morphological operations is presented which allows for a larger number of such operations to be performed. In addition, a labeling unit based on a contour tracing technique is presented which extracts features at low extra cost.

Typical examples of intermediate results are shown in Fig. 2.1. In Fig. 2.1a the original image is shown and the segmented binary mask in Fig. 2.1b. The labeled objects are shown in Fig. 2.1c were we can notify that the noise has been removed. By modifying the segmentation algorithm to be able to identify potential shadows we see that these can be removed to a large extent, e.g. the large shadow attached to the lower right side of the mid-person as shown in Fig 2.1d and e. Finally, Fig. 2.1f shows the output from the tracking algorithm where the objects are identified by the colored bounding boxes.

2.2 Segmentation

Compared to many other algorithms for video segmentation, one based on Gaussian Mixture background Model (GMM) was proposed in [18] with the unique feature of robustness in multi-modal background scenarios. A GMM is required for modeling repetitive background object motion, for example, swaying trees, reflections on a lake surface, a flickering monitor, etc. A pixel

located in the region where repetitive motion occurs will generally consist of two or more background colors, i.e. the RGB value of that specific pixel toggles over time. This would result in false foreground object detection with most other adaptive background estimation approaches.

The advantage of the GMM is achieved by updating several Gaussian parameters for each pixel, imposing a computational complexity and high memory bandwidth that prohibit real-time performance using a general purpose computer. In our simulations, on an AMD 4400+ dual core processor, a frame rate of only 4-6 fps is achieved for video sequences with 320x240 resolution. For a real-time video surveillance system with higher resolution, hardware acceleration is required.

2.2.1 **Algorithm formulation**

The algorithm is briefly formulated as follows: Measured from consecutive video frames, the values of any pixel can be regarded as a Gaussian distribution. Characterized by mean and variance values, the distribution represents a location centered at its mean values in the RGB color space, where the pixel value is most likely to be observed over frames. A pixel containing several background object colors, eg. the leaves of a swaying tree and a road, can be modeled with a mixture of Gaussian distributions with different weights. The weight of each distribution indicates the probability of matching a new incoming pixel. A match is defined as the incoming pixel within certain deviation off the center. In this paper, J times the standard deviation of the distirbution is used as matching threshold [18], where J is 2.5. The higher the weight is, the more likely the distribution belongs to the background. Mathematically, the portion of the Gaussian distributions belonging to the background is determined by

$$B = argmin_b \left(\sum_{k=1}^{b} \omega_k > H \right),$$

where H is a predefined parameter and ω is the weight. The mean, variance and weight factors are updated frame by frame. If a match is found, the matched distribution is updated as:

$$\omega_{k,t} = (1-\alpha)\omega_{k,t} + \alpha, \qquad \mu_t = (1-\rho)\mu_{t-1} + \rho X_t$$

$$\sigma^2 = (1-\rho)\sigma^2_{t-1} + \rho(X_t - \mu_t)^T(X_t - \mu_t);$$

where μ, σ^2 are the mean and variance respectively, and α, ρ are learning factors. For those unmatched, the weight is updated according to

$$\omega_{k,t} = (1-\alpha)\omega_{k,t},$$

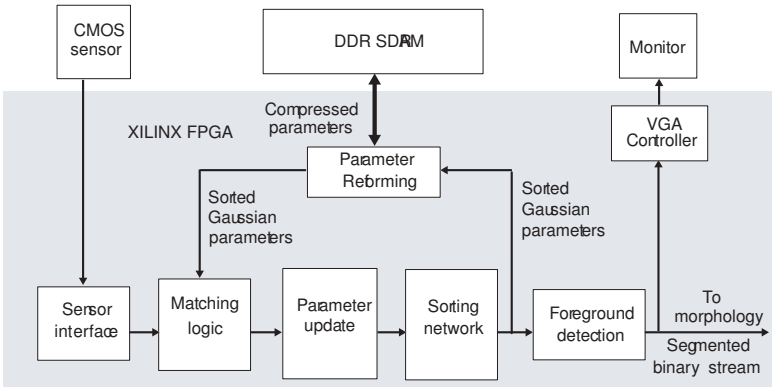


Figure 2.2: The system architecture of the segmentation unit.

while the mean and the variance remain the same. If none of the distributions are matched, the one with the lowest weight is replaced by a distribution with the incoming pixel value as its mean, a low weight and a large variance.

Maintaining a mixture of Gaussian distributions for each pixel is costly in terms of both calculation capacity and memory storage, especially with large frame size. To manage the RGB data from a video camera in real time, a dedicated hardware architecture is developed with a streamlined data flow. The hardware architecture as shown in Fig. 2.2 is presented in [30] and briefly explained as follows: A pixel value is read into the matching logic block from the sensor together with all the parameters for the mixture of Gaussian distribution and a match is calculated. In case an incoming pixel matches several Gaussian distributions, only the one with highest weight is selected as the matching distribution.

2.2.2 **Hardware implementation**

After the updated Gaussian parameters have been sorted foreground detection is achieved simply by summing up the weights of all the Gaussian distributions that have a higher likelihood than the updated one. By comparing the sum with a predefined parameter H, a sequence of binary data indicating background and foreground is streamed out to both the morphology block and the VGA controller.

2.2.3 **Memory bandwidth reduction**

Slow background updating process requires high precision for each parameter in the distributions. This is due to the fact that parameter values are changed

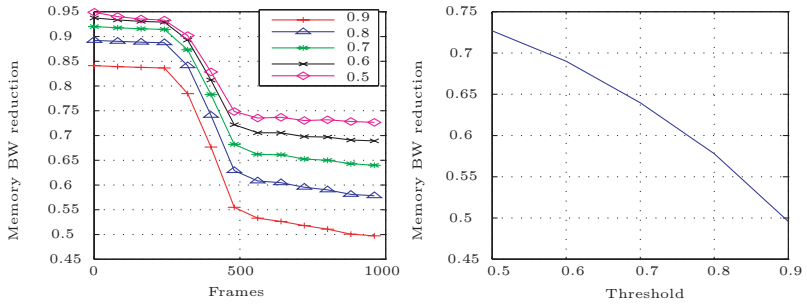
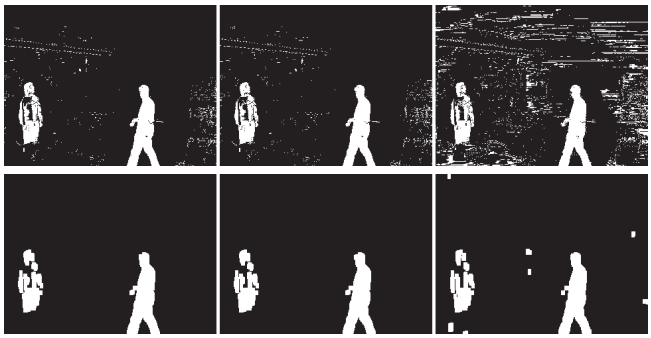


Figure 2.3: Memory bandwidth reduction over frames is shown to the left and memory bandwidth reduction versus different threshold is shown to the right.

slightly at a time, but could accumulate over time. From C++ simulation, both mean and variance for a Gaussian distribution in adjacent frames vary in the order of 10^{-4} to record slight light changes. For a fixed number representation, 28 bits are assigned to each RGB mean parameters and the variance takes 24 bits. The weight (likelihood) for each Gaussian distribution is updated using a learning factor. To avoid an over-segmentation effect with a fast learning factor, 16 bits are used for the weight with a slow learning factor (eg. 0.0002). One Gaussian distribution consists of the weight together with 3 means and 1 variance and contributes to 124 bits. For a video sequences with 320x240 resolution, a total of 27 Mbit Gaussian parameter data have to be stored and updated for each new frame, which is far beyond the on-chip memory resources.

In this paper, a data compression scheme is proposed which utilizes similarities for Gaussian distributions in adjacent areas. We classify "similar" Gaussians in the following way: from the definition of a matching process, each Gaussian distribution can be regarded as a three dimensional cube in the RGB space, where the center of the cube is composed of RGB mean values whereas the border to the center is specified by J times variance. One way to measure the similarity between two distributions is to check how much of the two cubes volume that overlap. If the overlap volume takes up a certain percentage of both Gaussian cubes, they are regarded as "similar". The reason for such criteria lies in the fact that a pixel that matches to one distribution will most likely match to the other, if they have enough overlapping volume. The percentage is a threshold parameter that can be set to different values among different simulations.

In the architecture, two similar distributions are treated as equivalent. By only saving non overlapping distributions together with the number of equivalent succeeding distributions, memory bandwidth is reduced. From simulation



a) b) c)

Figure 2.4: The result before and after morphological filtering for different thresholds, (a) original result, (b) with 0.8, and (c) with 0.4 threshold.

in C++, various threshold values are selected to evaluate the efficiency for memory bandwidth reduction. With a low threshold value where less overlapping Gaussians are regarded as the same, more savings could be achieved. However, more noise is generated in the segmented image due to increasing mismatches in the matching block. Fortunately, such noise is found non-accumulating and therefore can be reduced by later morphological filtering. Fig. 2.3 shows the memory bandwidth savings over frames with various threshold values. From the figure, large memory bandwidth savings (around 95%) are achieved from the start to approximately 380 frames, in which case no foreground object enters the scene and only one out of three Gaussian distributions are updated to contain the background. With more foreground objects moving in and occupying a bigger part of the frame, the number of similar Gaussian distributions drops dramatically, but tends to stabilize after some time when most pixels contains both foreground and background distributions. It is also shown in the figure that low threshold, by relaxing the criteria of similarity, results in more memory bandwidth savings. The quality of segmentation results before and after morphology is shown in Fig. 2.4. From these Figures, it is clear that memory reduction comes at the cost of segmentation quality. Too low threshold value results in noise clusters that would not be filtered out by morphological filtering. In this implementation, a threshold value of 0.8 is selected, with memory bandwidth savings of around 60%. To evaluate long term effects of memory bandwidth reduction scheme, FPGA platform is required to collect data in real time. In future implementations, commonly used video compression scheme such as transform coding using a DCT or DWT are under consideration for further Gaussian parameter compressions.

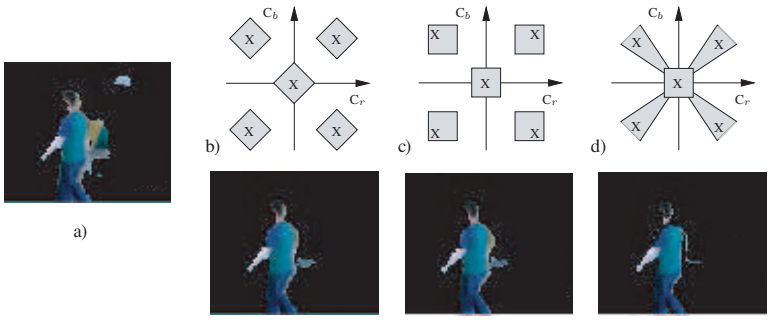


Figure 2.5: The original result (a) and the result with three different shadow detection rules (b,c,d) in the C_bC_r plane, where X is the stored mean of C_bC_r, five example points are shown for each rule. A new pixel part of a potential shadow if it falls in the gray area and there is a negative change in the Y component.

2.2.4 **Shadow reduction**

One drawback with the Gaussian mixture model segmentation algorithm is that it is sensitive to shadows and noise. Thus it has been investigated in [38] how different color spaces affect the segmentation result in terms of noise and shadow sensitivity. The YC_bC_r color space was found to be best, due to numeric stability and an independent brightness channel. The investigation was conducted with the implementation aspects in mind, i.e. simple changes that easily could be incorporated in the existing hardware.

To reduce the number of detected shadows, pixels that could be part of a potential shadow have to be recognized. Assuming white light, Y will always be smaller when shaded and that C_b and C_r will go towards the origin. With this information a simple rule for shadows can be formed; A potential shadow is found if a negative change is detected in Y and C_bC_r have moved slightly towards origin, compared to the stored mean of YC_bC_r. In addition, a shadow is not detected if a too large negative change in Y is found. Without this limit, a pixel that change color from white to black would be classified as a shadow which would result in a loss of sensitivity instead of a shadow reduction. Based on experimental results, a suitable indoor value for the maximum allowed negative change in Y is around 20-30% of the dynamic range of Y.

In the upper part of Fig. 2.5 three different rules for the C_bC_r plane to detect potential shadows are shown. All three assume that a negative change in Y has already been detected. The gray areas represent the part of the C_bC_r plane where a new pixel is ruled to be a potential shadow. The area location

is based on the stored mean of C_b and C_r, five example points are shown for each rule. With the first rule a new pixel is part of a potential shadow if the sum of absolute differences in C_bC_r is less than a threshold. The second rule compares the difference in C_b and C_r separately to thresholds that depend on the position of the stored mean of C_b, C_r. The last rule allows changes in a small sector from origin or, if the values are close to origin, in a small box around the stored mean of C_b, C_r [39]. The segmentation result with the three different shadow rules are shown in Fig. 2.5. As seen, most of the shadows are removed compared to Fig. 2.5a, with all three rules. However, the rule in Fig. 2.5d removes a part of the objects neck as well, due to acceptance of large changes in color and that color can move away from origin and still be classified as a shadow. In this particular image no significant performance difference can be seen between the two first rules. From a hardware complexity perspective, the first rule is simplest, since it only requires add and compare operations. The second rule is more complex since the thresholds depend on the location of the stored mean of C_bC_r and the third rule is most complex, since it requires a division to approximate the arctan function.

If any form of human recognition is to be included in the surveillance system, face detection becomes very important. To increase the chances of correct segmentation of faces, we try to find pixels with skin tone and use that information to improve the foreground/background decision. Skin tones differ much between human races, from black to white and with different tones of red and yellow. However, in the YC_bC_r color space, these colors are tightly distributed in the C_bC_r plane along the Y axis. This means that the Y component can be disregarded, since human skin tone has about the same color distribution in C_bC_r for most Y values [40] [41] [42].

With these methods, pixels that are likely to be part of a shadow or human skin can be found and the decision kernel of the segmentation algorithm can be altered to vary the likelihood of including these pixels as part of the foreground. Shadows should have a lower and human skin should have a higher probability to be included in the foreground. In the original paper [43], the threshold used to find a match is a constant, J, times the deviation. We propose to increase this constant if a potential shadow is detected and to decrease it if a potential skin pixel is detected. A matching distribution is found if

$$\begin{aligned} -JS_hstd < (Y_{new} &- \overline{Y}) < Jstd, \\ |Cb_{new} - \overline{Cb}| &< JS_kstd, \text{ and} \\ |Cr_{new} - \overline{Cr}| &< JS_kstd \end{aligned}$$

are true. Here *std* is the standard deviation, S_h is 1 if no shadow is found and > 1 if a shadow is found, and S_k is < 1 if the new pixel has skin color and 1 else. Fig. 2.1e shows the result of the improved segmentation with $J = 2.5$,

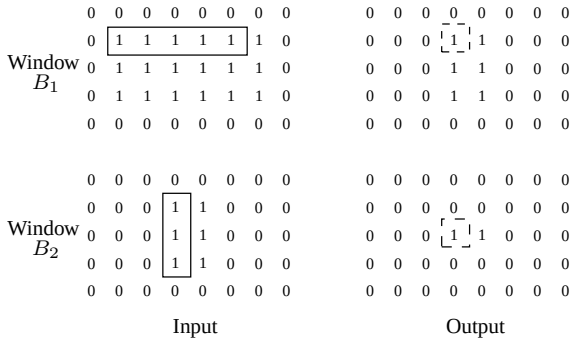


Figure 2.6: Input and output to an erosion operation were the SE is decomposed into B_1 and B_2.

$S_h = 2$, and $S_k = 0.5$.

2.3 Morphology

Erosion and Dilation (E&D) are the two foundations in mathematical morphology, since most morphologic operations can be broken down into these two basic operations [44]. For example, operations such as opening, closing, gradient, and skeletonization are performed with these two base functions. Derived from these facts, the need for efficient architectures to perform E&D becomes evident. To address this issue a low complexity architecture has been proposed in [45].

To easily incorporate the E&D unit into the system, some requirements are placed 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 burst reads from memory and that several E&D units can be placed sequentially after each other without any storage in between. Secondly, the hardware should be small, simple, and fast in order to allow as much time and hardware space for the object classification/tracking part of the system as possible. To increase the overall performance of the system, it is also desirable that the size of the Structuring Element (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). In this paper A will represent the binary input image and B the structuring element.

If the SE is both reflection invariant, i.e. $B = \hat{B}$, and decomposable, i.e

$B = B_1 \oplus B_2$. Then the following two equations can be derived

$$A \oplus B = (A \oplus B_1) \oplus B_2 = ((A' \ominus B_1) \ominus B_2)' \tag{2.1}$$

$$A \ominus B = A \ominus (B_1 \oplus B_2) = (A \ominus B_1) \ominus B_2 \tag{2.2}$$

where $'$ is bit inversion, $\oplus$ is dilation, and $\ominus$ is erosion. However to find decompositions to a general SE is a hard problem and not always possible [46] [47]. In addition, for a SE to be reflection invariant it has to be symmetric both in respect to the x and y direction, e.g., a square or a circle. However, one common class of SEs that is both decomposable and reflection invariant is rectangles of ones. This type of SE is well suited for the opening and closing operations that are needed in this system. An example of erosion with a decomposed SE is shown in Fig. 2.6, were the SE is decomposed into B_1 and B_2. The input is first eroded with B_1 and then B_2. The first position of B_1 and B_2 that produce a one is shown in the Figure, together with location in the output of this one. 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, in this case from 15 to 8.

The proposed architecture is based on Equation 2.1 and 2.2, in order to take advantage of the reduced number of comparisons that a decomposed SE require. In addition, when comparing Equation 2.1 and 2.2, it can be seen that only the erosion operation with a decomposed SE has to be implemented. To perform a dilation, the input A and the result is inverted. Hence, the same inner kernel can be 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, where the first stage compares the number of ones under B_1 to the width of B_1 and the second stage compares the number of ones under B_2 in the result from the first stage to the height of B_2.

2.3.1 Architecture

In Fig. 2.7, the architecture of the datapath is shown together with the wordlength in each stage. The architecture includes logic to insert padding and to choose between erosion and dilation operation. 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, $\lceil \log_2(B_{width}) \rceil$ and $\lceil \log_2(B_{height}) \rceil$ in stage-1 and 2, respectively. Thus, the

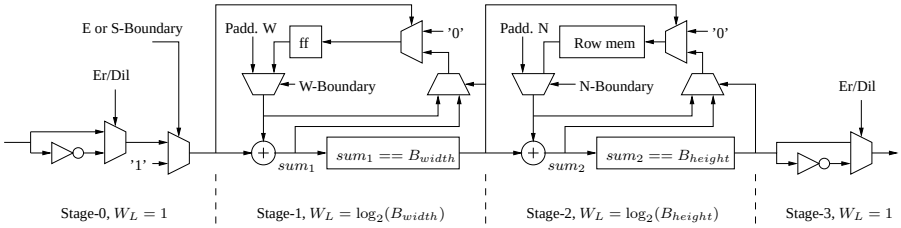


Figure 2.7: Architecture of the datapath in the erosion and dilation unit and the wordlength (W_L) in each stage.

total amount of required memory to perform dilation or erosion is

$$\text{Mem} = \lceil \log_2(B_{width}) \rceil + \lceil \log_2(B_{height}) \rceil A_{columns} \text{ bits},$$

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 $A = 240 \times 320$ bits and the size of $B = 15 \times 15$, the required amount of memory is $\lceil \log_2(15) \rceil + \lceil \log_2(15) \rceil \cdot 320 = 1.25$ kbit. The delay line implementations in [48] and [49], with the same A and B would require $(B_{height} - 1)A_{columns} + B_{width} = 4.4$ kbit of memory, which is ≈ 3.5 times more than the required memory in the presented implementation.

When sliding B_1 over a row in A, each position in A is used as many times as the width of B_1 to calculate the sum. However, if a running sum records the number of consecutive ones in the currently processed input row, each input is used only once. In Fig. 2.7, the flip-flop (ff) in stage-1 stores the sum of consecutive ones. When the input is one, the recorded 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 consecutive hits from stage-1, i.e. number of rows above the current point with B_{1width} consecutive ones, for each column in A.

The morphological operation used in this system is an opening (erosion followed by dilation), and due to the pipelined nature of the architecture, two E&D units in series will not increase the execution time and only add a few clock cycles delay. In fact, the opening operation can be performed directly on the output stream from the segmentation unit without adding any image memories and without adding anything to the overall execution time of the system.

An example of a filtered segmentation result is shown in Fig. 2.1c. An opening operation is performed with a SE size of 5x3 (height x width) pixels

in the erosion part and 7x5 in the dilation part.

2.4 Labeling

After segmentation, a binary frame is produced containing connected clusters of pixels that represent different objects. Assuming that noise has been removed by the morphology unit, the frame now only contain objects of interest that can be tracked and classified. To be able to separate and distinguish between these clusters, they have to be identified, i.e. labeled.

Various labeling algorithms have been proposed and a survey of various algorithms can be found in [50]. A common property for all these algorithms is that they are memory access intense. Furthermore, all algorithms 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. Since an image typically is scanned from top to bottom and from left to right, it is not possible to know that the two pillars in the u is part of the same object until the bottom part of the u is encountered. Two common methods to handle this problem is:

- Equivalence Table Based Algorithms – Two scans with a corresponding equivalence table.
- Contour Tracing Based Algorithms – A single scan with contour tracing.

Equivalence table based algorithms [51] scan through the memory writing every label collision into an equivalence table. The first label scan is completed on the fly as the frame is written into the memory by comparing each pixel with its neighbors to the left and above. After the first scan all pixels are assigned a label and all collisions have been detected. The second scan resolves all collisions and reduce the number of labels per cluster to one.

Contour tracing based algorithms [52] is a technique that requires one global scan together with some additional random memory accesses for the contour tracing procedure. The major advantage is that label collisions will never occur since when an unlabeled cluster is encountered, the contour of that cluster is labeled immediately and every pixel within a label is regarded as part of the same cluster. If a cluster with a labeled contour is encountered, the scan proceeds without modification continuing until an unlabeled pixel is reached, restarting the contour tracing procedure, or the last pixel is reached. If a cluster has a hole inside its contour, this hole will not be traced. Every pixel between two labels can therefore be considered a part of an object, cluster holes are filled.

Extracting properties by post processing in the tracking stage or by a general purpose processor can be time consuming, thus every property that can be extracted by an algorithm without inferring additional hardware complexity

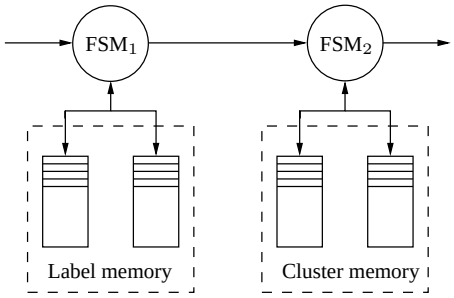


Figure 2.8: Overview of the implemented architecture.

should be seen as an advantage. In comparison, both types of labeling algorithms can extract maximum and minimum coordinates and the number of pixels in each cluster. In addition, the contour tracing algorithm can calculate moments (center of gravity) and fill holes inside the clusters. Thus, the contour tracing label algorithm is more suitable to be used in a automated surveillance system.

2.4.1 Architecture

An overview of the architecture is illustrated in Figure 2.8. The first Finite State Machine (FSM_1) in Figure 2.8, first writes the incoming frame into the labeling memory, since the contour tracing phase can not start without the complete frame. 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.

As the input frame is being sequentially written into the label memory, the location of the first and last 1 in the input is set as start point and end point respectively. When the global 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.

During the contour tracing phase, FSM_1 outputs the coordinates for every contour pixel together with control signals to FSM_2. FSM_2 updates the maximum /minimum coordinates for every label together with the variables used to calculate moments. The dual memory architecture, depicted in Figure 2.8, is a memory pipeline which gives the tracking stage access to the label information of the previous frame, i.e. when the $FSMs$ are updating one of the two memories, they give access to the other.

An example of a labeled motion mask is shown in Fig. 2.1c. Each label is represented by a color.

2.5 Feature extraction

Features are used to separate the tracking/classification part of the system from the image stream. This is necessary since the amount of data in the image stream is very large and the tracking/classification part is implemented in software running on an embedded processor. A feature is a property extracted from an object in the image, e.g. size, color, texture, or shape, that can separate different objects or recognize a certain object class.

Features are calculated for each object in each video frame and if several features are used, heavy calculations are required. This means that they often are well suited to be implemented in hardware. With good features, the amount of data that the track/classification algorithms have to handle can be reduced to a level that is suitable for an embedded software implementation

A good feature describes each object with a unique and compact code and does not change if the object is scaled, rotated, or enters an area with different lighting. This is necessary to be able to track an object through different environments. For example, to track a person standing under a lamp close to the camera that moves away from the camera to a darker corner of the room, features that are size- and lighting-invariant are required.

In this system we have three types of features. First, the once 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 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. minimum processing time and memory requirements. In addition, color features are size invariant and with the right color space also lighting invariant. 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 about which objects from previous frame corresponds to which objects in the current frame.

2.6 Tracking

Tracking are most suitable to implement in software, since this process involves a lot of exceptions and bookkeeping. Another concern is updating the algo-

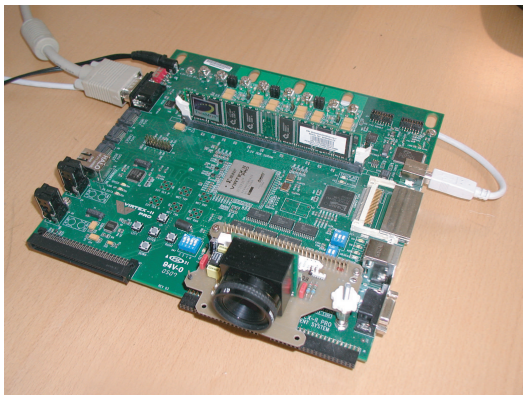


Figure 2.9: The system development board.

rithms or modify them to fit a certain scenario, which is also easily done with a software solution. Thus, our tracking is performed in software and it consists of two parts; clustering and matching. Clustering is necessary since segmentation is not perfect and larger objects, like humans, are often split into more than one object. Clustering, merge smaller objects into larger ones based on the size of the objects found in previous frame. Matching is performed between all stored objects and all new clustered objects, based on the least mean error of the features.

The features described in the previous section 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 keep track of an object that walks out from the scene and reenters at a later time. Partial and total occlusion can be handled in most cases but if two or more persons enter the scene with similar clothing occlusion handling between those persons becomes unreliable. The obvious solution to these problems is to include more and better features, which is currently investigated.

2.7 Results

The system is implemented on a Xilinx VirtexIIpro vp30 FPGA development board, with a 256MB DDR SDRAM and two FPGA embedded Power PCs. A KODAK KAC-9648 CMOS sensor is used to capture color images at 25 fps with a 320x240 resolution. The development board is shown in Fig. 2.9 with the sensor attached directly onto the board. A custom made VGA-controller provides a monitor with results from the different parts of the system.

Real time segmentation performance is achieved on video sequences with 3

Gaussian distributions per pixel at an operating frequency of 100 MHz. The off-chip SDRAM is used to fulfill the requirements to store the large amount of Gaussian parameter data. With the proposed memory reduction scheme applied to experimental test video inputs, off-chip memory bandwidth is reduced from 1.37 Gbit/s to 0.55 Gbit/s. The morphology unit supports structuring elements (SE) up to the size of 15x15 pixels with individual settings for width and height. The SE size can be changed during runtime and is individual for each erosion/dilation unit. The datapath memory and intermediate FIFOs are less than 3.2 kbit per unit and the operating speed is 100 MHz. The labeling unit extracts min and max coordinates of up to 64 objects per frame. The dual memory architecture requires 300 kbit of memory and the operating speed is 67 MHz. The color feature extraction and the tracking algorithm are currently running in software on the embedded Power PC. In the next versions of the system, moment calculation will be added to the labeling part and the color feature extraction will be moved to hardware.

The system prototype is modular, in the sense that each block can be replaced with other algorithms without changing the system architecture. Furthermore, each unit is parameterized in order to evaluate long and short time effects of various settings. More information and example videos can be found at the project homepage [53].

2.7.1 Bottlenecks

The presented system use a resolution of 320x240, which is rather low compared to modern digital video cameras. This resolution is used since there is a limited amount of resources, especially memory, on the FPGA. However, future surveillance systems will most likely require a higher resolution and therefore it is of interest to study the bottlenecks of the system and how they react to an increased resolution. For example, if the resolution increase to 640x480, i.e. four times as many pixels per image, and the frame rate remains 25 fps, how will this affect the different parts of the system and what can be done to decrease the impact of an increased resolution?

The segmentation algorithm scales linearly, i.e. the critical memory bandwidth increases to 5.5 Gbit/s with the straight forward implementation and 2.2 Gbit/s with the presented memory reduction scheme. To reduce the bandwidth further the approach used in [54] could be used, where the distributions are not updated every frame. The morphology unit is much less affected by a resolution increase, since the memory is only dependent on the width of the image. If the SE is increased to match the higher resolution, i.e. to 31x31 pixels, only 2.5 times more memory is required in the data path and the intermediate FIFOs are unaffected. In the label unit, both label memories increase with a factor of 4. One way to reduce this could be to only keep one label memory used

by the contour tracing algorithm as it is, and compress the resulting labeled image into a smaller memory using a compression scheme, e.g. run length encoding or JBIG [55]. In terms of memory, feature extraction is unaffected by the resolution increase, since it only works on streaming data and only stores the result. It will however, require 4 times as many clock cycles to execute, this is true for all previous blocks as well. The only part totally unaffected by the resolution increase is the tracking part. Neither the number of objects nor the number of features per object is affected by a resolution increase.

2.8 Conclusions

In this paper a real-time hardware platform for an automated digital surveillance system is presented. The platform has been developed in order to identify and propose solutions to computational and memory bottlenecks. Due to the real-time processing it also substantially reduces analysis of long terms effects due to changes in the algorithms and to parametric changes. A memory scheme reducing memory accesses with $> 60\%$ in video segmentation is proposed. Furthermore, a low complexity morphology architecture in combination with a labeling unit and the video segmentation part is presented. Feature extraction and tracking is currently implemented in software on an embedded Power PC on the FPGA. While the feature extraction is moving towards a hardware implementation the tracking will stay in software. A simplified system prototype is running on an FPGA development board.

Bibliography

[1] D. A. Migliore, M. Matteucci, and M. Naccari, "View-based detection and analysis of periodic motion," in *Proc. of the 4th ACM international workshop on Video surveillance and sensor networks, California, USA*, October, 2006, pp. 215–218.

[2] S. Chien, S. Ma, and L. Chen, "Efficient moving object segmentation algorithm using background registration technique," *IEEE tran. on circuits and systems for video technology*, July 2002.

[3] J. Moon, D. Kim, and R. Park, "Video matting based on background estimation," *Enformatic Transactions on Engineering, Computing and Technology*, December 2004.

[4] Q. Zhang and R. Klette, "Robust background subtraction and maintenance," in *Proc. of the 17th International Conference on Pattern Recognition (ICPR), Cambridge, UK*, August, 2004, pp. 90–93.

[5] R. C. Gonzalez and R. E.Woods, *Digital Image Processing (2nd Edition)*. Prentice Hall, 2002.

[6] R. Cutler and L. Davis, "View-based detection and analysis of periodic motion," in *14th International Conference on Pattern Recognition, Brisbane , Australia*, August, 1998.

[7] B. Lo and S. Velastin, "Automatic congestion detection system for underground platforms," in *Proc. of International Symposium on Intelligent Multimedia, Video and Speech Processing, Hongkong, China*, May, 2001, pp. 158–161.

[8] F. Porikli, "Multiplicative background-foreground estimation under uncontrolled illumination using intrinsic images," in *Proc. of IEEE Workshop on Motion and Video Computing (WACV/MOTION'05), Colorado, USA*, January, 2005, pp. 20–25.

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

[10] D. Koller, J. Weber, T. Huang, J. Malik, G. Ogasawara, B. Rao, and S. Russell, "Multiplicative background-foreground estimation under uncontrolled illumination using intrinsic images," in *Proc. of the IEEE International Conference on Pattern Recognition, Jerusalem, Isreal*, October, 1994, pp. 126–131.

[11] J. K. K. Toyama, B. Brumitt, and B. Meyers, "Wallflower:principles and practice of background maintenance," in *Proc. IEEE International Conference on Computer Vision and Pattern Recognition*, 1999.

[12] S. Haykin, *Adaptive Filter Theory (4th Edition)*. Prentice Hall, 2001.

[13] J. Zhong and S. Sclaroff, "Segmenting foreground objects from a dynamic textured background via a robust kalman filter," in *Proc. European Conference on Computer Vision (ECCV)*, pp. 44–50.

[14] S. Messelodi, C. M. Modena, N. Segata, and M. Zanin, "A kalman filter based background updating algorithm robust to sharp illumination changes," in *Proc. of the 13th International Conference on Image Analysis and Processing, Cagliari, Italy*, pp. 163–170.

[15] C. Ridder, O. Munkelt, , and H. Kirchner, "Adaptive background estimation and foreground detection using kalman-filtering," in *Proc. of International Conference on recent Advances in Mechatronics (ICRAM), UNESCO Chair on Mechatronics, Istanbul, Turkey*, pp. 193–199.

[16] J. Markhoul, "Linear prediction: A tutorial review," in *Proceedings of the IEEE*, April, 1975, pp. 561–580.

[17] N. Friedman and S. Russell, "Image segmentation invideo sequences: A probabilistic approach," in *Proc. of the Thirteenth Conference on Uncertainty in Artificial Intelligence(UAI)*, pp. 175–181.

[18] C. Stauffer and W. Grimson, "Adaptive background mixture models for real-time tracking," in *Proc. IEEE Conference on Computer Vision and Pattern Recognition*, 1999.

[19] O. Javed, K. Shafique, and M. Shah, "A hierarchical approach to robust background subtraction using color and gradient information," in *Proc. Workshop on Motion and Video Computing*, 2002.

[20] L. Li, W. Huang, I. Y, H. Gu, and Q. Tian, "Foreground object detection in changing background based on color co-occurrence statistics," in *Proc. 6th IEEE WS on Applications of Computer Vision*, 2002.

[21] M. Harville, G. Gordon, and J. Woodfill, "Adaptive video background modeling using color and depth," in *Proc. IEEE International Conference on Image Processing*, 2001.

[22] Y. Zhang, Z. Liang, Z. Hou, H. Wang, and M. Tan, "An adaptive mixture gaussian background model with online background reconstruction and adjustable foreground mergence time for motion segementation," in *Proc. IEEE International Conference on Industrial Technology*, 2005.

[23] H.Wang and D. Suter, "A re-evaluation of mixture-of-gaussian background modeling," in *roc. IEEE International Conference on Acoustics, Speech, and Signal Processing*, 2005.

[24] Y. Xu, D.Xu, Y. Liu, and A. Wu, "Illumination invariant motion estimation for video segmentation," in *Proc. IEEE International Conference on Acoustics, Speech, and Signal Processing*, 2004.

[25] P. W. Power and J. A. Schoonees, "Understanding background mixture models for foreground segmentation," in *Proc. of Image and Vision Computing*, Auckland, New Zealand, November 2002, pp. 267–271.

[26] A. Elgammal, D. Harwood, and L. Davis, "Non-parametric model for background subtraction," in *Proc. European Conference on Computer Vision (ECCV), Dublin, Ireland*, pp. 751 – 767.

[27] D. Magee, "Tracking multiple vehicles using foreground, background and motion models," *Image and Vision Computing*, no. 22, pp. 143–145, 2004.

[28] F. Kristensen, P. Nilsson, and V. Öwall, "Background segmentation beyond rgb," in *Proc. Asian Conference on Computer Vision*, 2006.

[29] Color-space converter: RGB to YC_rC_b. [Online]. Available: http://www.xilinx.com/bvdocs/appnotes/xapp930.pdf

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

[31] K. Batcher, "Sorting networks and their applications," in *Proc. of the AFIPS Spring Joint Computing Conference*, 1986.

[32] M. Ajtai, J. Komlos, and E. Szemeredi, "An o(n log n) sorting network," in *Proceedings of the 25th ACM Symposium on Theory of Computing*, 1983.

[33] G.V.Russo and M.Russo, "A novel class of sorting networks," *IEEE tran. circuits and systems I: Fundamental Theory and Applications*, July 1996.

[34] Axis communications. [Online]. Available: http://www.axis.com/

[35] H. Hedberg, F. Kristensen, P. Nilsson, and V. Öwall, "Low complexity architecture for binary image erosion and dilation using structuring element decomposition," in *Proc. IEEE International Symposium on Circuits and Systems*, 2005.

[36] Xilinx university program. [Online]. Available: http://www.xilinx.com/univ/xupv2p.html

[37] Kodak image sensor. [Online]. Available: http://www.kodak.com/US/en/dpq/site/SENSORS/name/ISSCMOSProductFamily

[38] F. Kristensen, P. Nilsson, and V. Öwall, "Background Segmentation Beyond RGB," in *Seventh biennial Asian Conference on Computer Vision*, Hyderabad, India, jan 2006.

[39] O. Schreer, I. Feldmann, U. Gölz, and P. Kauff, "Fast and Robust Shadow Detection in Videoconference Apllication," 2002.

[40] K. Wong, K. Lam, and W. Siu, "An Efficient Color Compensation Scheme for Skin Color Segmentation," in *Proc. of IEEE International Symposium on Circuits and Systems*, Bangkok, Thailand, Mar. 2003.

[41] C. Garcia and G. Tziritas, "Face Detection Using Quantized Skin Color Regions Merging and Wavelet Packet Analysis," *IEEE Transactions on Multimedia*, vol. 1, pp. 264–277, 1999.

[42] H. Wang and S. Shang, "A Highly Efficient System for Automatic Face Region Detection in MPEG Video," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 7, pp. 615–628, 1997.

[43] C. Stauffer and W. E. L. Grimson, "Adaptive background mixture models for real-time tracking," in *Proc. of IEEE Computer Society Conference on CVPR'99*, Ft. Collins, USA.

[44] J. Serra, *Image Analysis and Mathematical Morpohology, Vol 1.* Academic Press, 1982.

[45] H. Hedberg, F. Kristensen, P. Nilsson, and V. Öwall, "A low complexity architecture for binary image erosion and dilation structuring element decomposition," in *Proc. of IEEE ISCAS'05*, Kobe, Japan, 2005.

[46] 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.

[47] 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.

[48] 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.

[49] 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.

[50] W. Kesheng, O. Ekow, and S. Arie, "Optimizing connected components labeling algorithms," in *SPIE Int. Symposium on Medical Imaging*, San Diego, CA, USA, feb 2005.

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

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

[53] "Project homepage," 2005, www.es.lth.se/home/fkn/index.html.

[54] D. Magee, "Tracking multiple vehicles using foreground, background and motion models," *Image and Vision Computing*, vol. 22, pp. 143–155, 2004.

[55] "Offical JBIG homepage," 2006, www.jpeg.org/jbig/index.html.

[56] Bangkok, Thailand, Mar. 2003.

[57] *Journal of CVIU*.

Controller Synthesis in Real-time Image Convolution Hardware Accelerator Design

Chapter 1

Introduction

1.1 Motivation

As mentioned in the general introduction, signal processing algorithms can be implemented in a range of technologies with much differed efficiencies in terms of speed (throughput), power and flexibility. By moving the implementation from general purpose processors down to dedicated hardware in the design space, parallelism within a algorithm can be increasingly utilized to optimize for speed or power at the cost of flexibility. However, with more details exposed to the designer when the abstraction level is moving down, design efforts are increased substantially. To overcome these problems, new design methodologies are needed. For dedicated hardware implementation, limited flexibility can be achieved by using time-multiplexed architectures, which can be found in many applications, e.g. filter design, FFT and CNN. By sharing logic blocks with the similar functionality in a hardware mapped implementation, a time-multiplexed architecture can achieve the same functionality with reduced size, increased flexibility at the cost of the throughput. Such architecture can be crucial in many situations, where fully hardware mapped implementation is not feasible, e.g. FFT implementation with large number of points and image processing with CNN at large resolution. The trade-offs of a time-multiplexed architecture can be easily illustrated by the finite impulse response(FIR) filter design, as shown in Fig.1.1 and Fig.1.2. From the hardware implementation perspective, an FIR filter is generally conceived as the sum of a history of input data being processed by many *multiply-accumulate* (MAC) units with different coefficients. The MAC operations can be shared in a time-multiplexed architecture. Thus

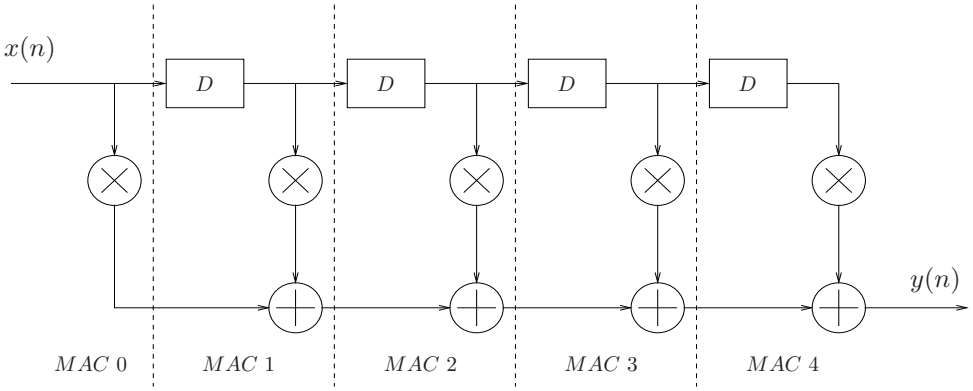


Figure 1.1: Direct hardware mapped FIR architecture.

calculations for an n tap FIR filter can be implemented in one MAC unit with n clock cycles instead of n MAC units with one clock cycle. All the coefficients are stored in a memory, feeding the shared MAC unit with one at a time. A control unit is needed to schedule all the operations, so that each of them takes place at the right time, e.g. which coefficient is feed to the MAC unit at what time, when to reset the MAC unit at the end of each filtering stage. The design is flexible to some extent since the coefficients can be changed later for different filtering function. The size of the design is also reduced to only 1 MAC unit being instead of n. To generalize the idea, the throughput of a hardware system can be sacrificed for certain flexibility and the size of a design by using time-multiplexed architecture, which is composed of three parts: datapaths, finite state machines (FSM), and memories.

As algorithms in image processing increases substantially, datapaths can get very complicated with large quantity of functional blocks. With the traditional way of hand coding FSMs in HDLs from the state transition graph, the design of a control unit capable of scheduling complex operations can be tedious and error prone. In addition, state transition graph is not an intuitive way to manipulate the dataflow within the datapath. More naturally, a sequential language with concurrency and clock cycle precision may be preferred by hardware designers for control algorithm specification. Ideally, such language can have C-like structures concentrating on functional behaviors of the datapaths while low level control signal assignments and state transition can be ignored. Synthesis of such language should also be enabled for mapping the control algorithm into a hardware controller with optimized architectures that is ready to be incorporated to state of the art hardware design flow. Motivated

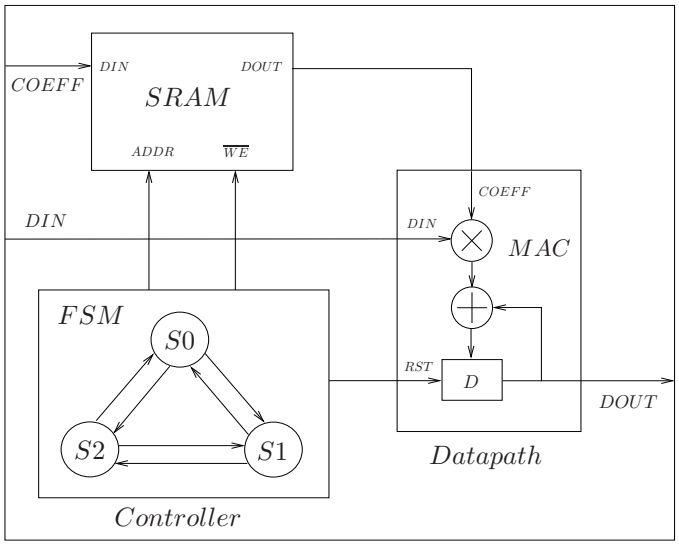


Figure 1.2: Time multiplexed FIR architecture.

by these, a design environment for controller design automation was implemented [1]. However, as the tool was designed in the 90's targeting on the old implementation environment, it can not be integrated into today's widely adopted commercial hardware design flow. As part of the thesis work, many modifications are made to the tool, which enables a design environment for a controller synthesis from a C-like sequential language transformed into a controller hardware specified in VHDL. In the following sections, various aspects of the design environment is explained in detail, together with the descriptions of possible architecture optimization techniques. An example design of image convolution accelerator implementation is also given for the demonstration of controller synthesis tool.

1.2 FSM Encoding

Control algorithm can be specified in a low level microprogram, with the syntax constructs of which being composed of statements with branch transitions. A statement is responsible for assigning values to the control signals that manipulate the dataflow of a datapath. It also decides which of the statements is to be executed next, a procedure called branch address transition. Such a transition can be sequential or conditional. Sequential branch transitions take place when non-conditional statements are present, with the address of the

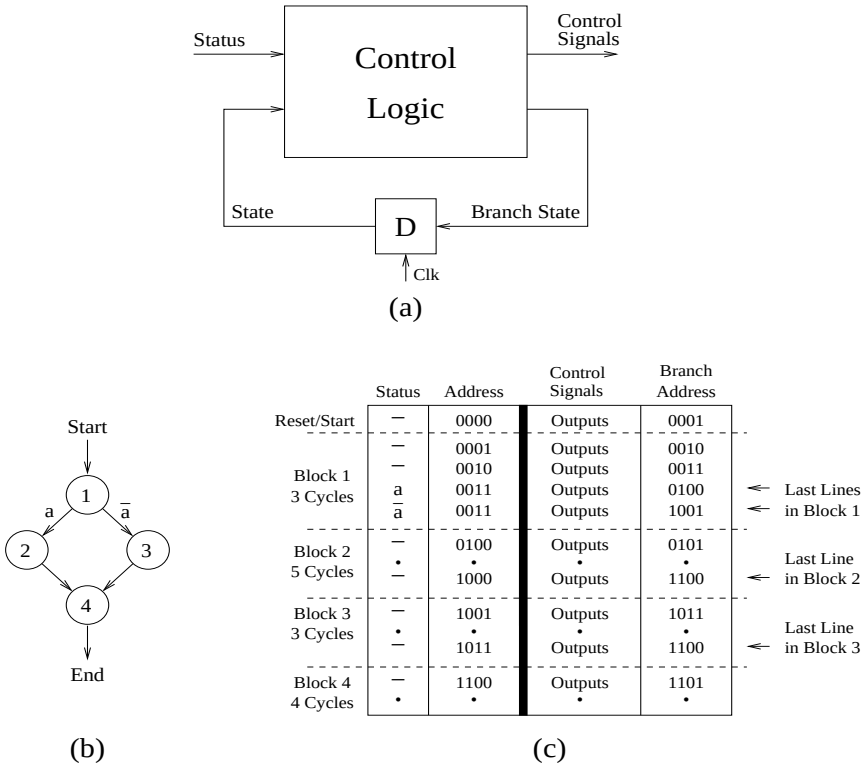


Figure 1.3: Basic idea of FSM encoding

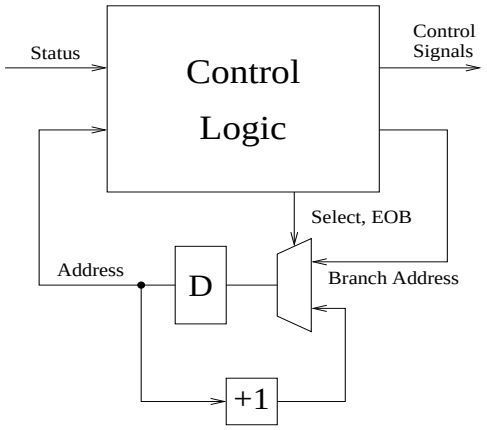


Figure 1.4: FSM architecture with incremental circuitry.

next statement being the increment of the current address by one. However, for a conditional branch transition the decision is based on the relevant conditional variable(s). For hardware implementations, such a microprogram can be realized by an FSM with the control signals as the output, the address of each statement as the states, and the conditional variables as the inputs. The mapping from a microprogram to an FSM process can be interpreted as an FSM encoding. Fig.1.3 illustrates the basic idea of FSM encoding by an example, where a simple algorithm specified as a *signal transition graph* (STG) is encoded into a truth table, which is the combinational logic part of a FSM. From the figure, each statement is assigned a unique address. During the execution of each statement, control signals are generated accordingly as part of the outputs. In addition, the address of the next statement is calculated. For sequential branch address transition, the encoding of the address for the next statement is the increment of the current address. This could get a slightly more complicated when a conditional branch transition is encountered, where two branch address entries are added to the truth table depending on the value of the conditional variable a.

Loops can be treated as a special case of a conditional branch transition, where the conditional variable is a loop counter.

1.3 Architecture Optimization

Based on the basic FSM encoding approach described above, optimizations can be done by utilizing special features within a microprogram. Usually, a big part of a microprogram is made up of a large number of sequential statements.

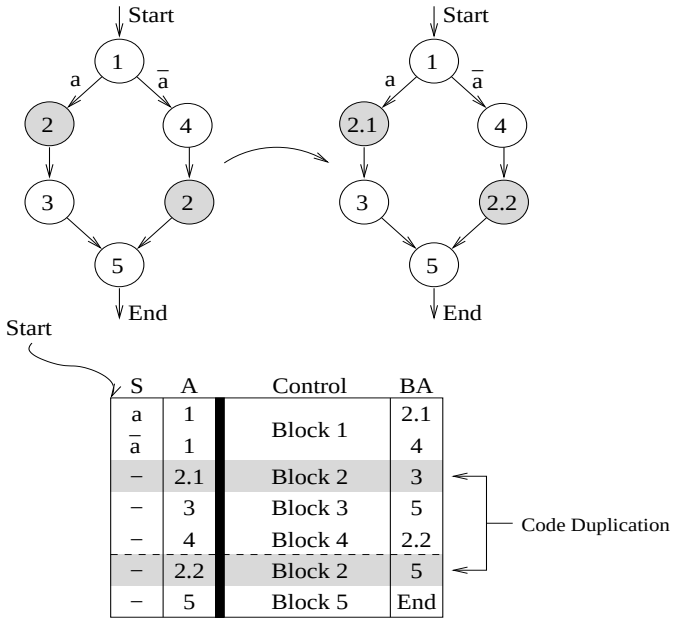


Figure 1.5: Subroutines duplicated in two locations differs only at the branch address calculation at the end of the block.

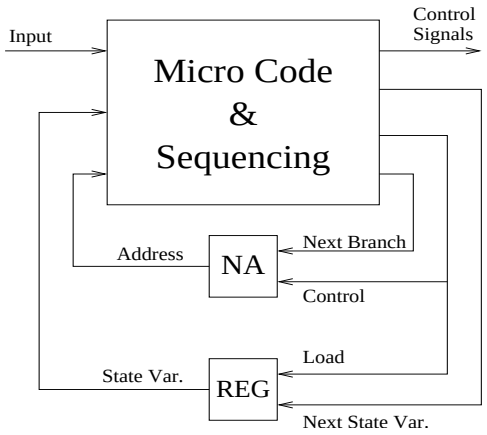


Figure 1.6: extended architecture that supports both state variables and incremental branch address calculation.

Instead of adding incremental address entry to the control logic (True table), an external incrementer can be used. As shown in Fig.1.4, the basic FSM structure can be extended with an extra incremental circuitry to calculate an incremental address or remains at the current address. With an extra select signal added to control logic to control the external incrementer, the branch address encoding of sequential blocks of a microprogram can be ignored (dont care signals). Thus logic minimization can take advantage of the situation, effectively reducing the complexity of logic gates. However, the benefit of architecture extension is limited to control algorithms with large number of sequential blocks. The overhead by introducing extra incremental circuitry and select signal can be dominant for algorithms with many conditional branch transitions.

Another optimization can be done in the area of subroutine reuse. With the basic architecture, the reuse of subroutines at different location of the microprogram is prohibited. With the basic FSM encoding approach, the same block of codes referenced in different locations in a microprogram have to be duplicated in most parts, differing only at the branch address calculation upon completion. As shown in Fig.1.5, subroutine node 2 is referenced in two different locations. To realize this with the basic FSM encoding, the same block has to be duplicated twice into in the true table, varying only in the branch address calculation at the end of the block. This is obviously waste of logic resources.

To make more efficient use of a control logic, state variables can be introduced to provide additional information of the context environment for a subroutine block. Basic FSM encoding approach can be extended, so that

branch address is calculated with the knowledge of its current location in a microprogram by referring to the state variable. As shown in Fig.**??**, a state variable is assigned with different value based on the location where the subroutine is referenced. With the extra location information, the subroutine 2 can be implemented with single blocks able to determine its branch address upon the completion of each reference. The state variables can be binary encoded, e.g. the number of bits is dependant on the number of times a subroutine is referenced in the microprogram. Fig.1.6 shows the extended architecture that supports both state variables and incremental branch address calculation.

1.4 Memories and Address Processing Unit

Memories are essential in image processing algorithms. They are used to store e.g. coefficients, parameters or intermediate calculation data. For time-multiplexed hardware implementations of image processing algorithms, complicated control flows are in a large part contributed by manipulating memory address calculations. To assist the designer for memory address calculating, Address Processing Units (APUs) synthesis is incorporated into controller synthesis. In the microprogram, the user of the tool can declare memories and their associate memory address variables. Whenever the address calculation is needed, the user can simply use C-like syntaxfor the address variables, e.g. +, -, ++, –. The rest of the task is done in the synthesis tool by mapping each of the address variables to the corresponding hardware components and sending control signals to calculate /update their values. The basic architecture of a APU is shown in figure 1.7. From the microprogram, the synthesis tool tries to locate each place where a address is calculated and map each one of the address variable into a register. The address calculation can be classified into one of the three categories: an address variable calculation with a constant, calculations between two address variables, an increment or decrement of an address variable. From figure 1.7, any of these operations can be performed in the basic hardware architecture. The constants can be implemented as a fix value, being one of the inputs to the MUX, and an increment and a decrement can be regarded as a special constant (a logic '1'). Architecture extensions can be made, e.g. using an extra MUX as described in the following chapter. An initial address value can be obtained from other logic blocks of a design. This can be useful if the APU is used as a loop counter, whose initial value depends on different requirements in a different situation.

1.5 Conclusion

The Controller synthesis tool can assist in complicated controller design by reducing design effort substantially. With a design flow from C-like syntax control

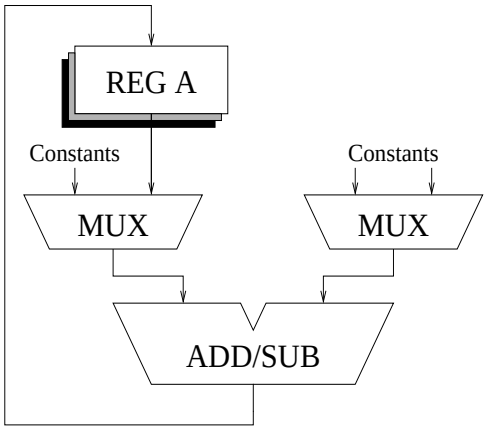


Figure 1.7: Basic architecture of an address processing unit.

algorithm specification down to VHDL implementation, a controller hardware can be generated automatically and implemented on any technologies by using state of the art digital IC design environment. Architecture extensions to basic FSM architecture can be utilized to optimize a control logic in different design tasks. Facilities like Address calculating unit are incorporated which further provides the design ease in a complex design with many address calculations. In the following section, the power of the tool is demonstrated in the design of a real-time image convolution hardware accelerator design.

Chapter 2

Controller synthesis in image convolution hardware accelerator design

2.1 Introduction

Two dimensional image convolution is one of the fundamental processing elements among many image applications and has the following mathematical form:

$$y(K_1, K_2) = \sum\sum x(K_1 - m_1, K_2 - m_2)h(m_1, m_2).$$

where x and h denotes the image to be filtered and kernel functions respectively. Due to its nature of computation intensity and a consequent high I/O data transfer, several dedicated solutions are adopted nowadays to fulfill the demanding processing task. For example, as part of image processing library for TI's highest performance TMS320C64x DSP series, such fundamental function has been implemented as software routines written in optimized assembly language to take advantage of the specific DSP architectures [2]. Although such high performance DSP can run up to 600Mhz, performing real-time image convolutions for image size of 512×512 with kernel size larger than 7×7 is still beyond its limitations [3]. Alternatively, many FPGA implementations of image convolutions has been reported [3–5]. Focusing mainly on FPGA architectures for enhancing calculation capacity, few of these implementations address the combination of data flow architecture which enables low I/O datarates with sustained calculation capability. In this paper, A streamlined dataflow composed of three level of memory hierarchy are employed, resulting in substantial I/O bandwidth reduction. Furthermore, potential power savings are also identified

by such memory hierarchy for the future ASIC solutions. It is estimated that power consumed by memory operations in three level hierarchies can be over 35 times lower than that of the one level direct memory scheme. Combined with four parallel processor with pipelined adder tree structure, a calculation capacity of larger than 2.5G MAC/s is achieved at the clock frequency of 50Mhz.

2.2 Two dimensional image convolution

Two dimensional image convolution is computationally intensive and requires a corresponding high data throughput. If such an application is demanded for large image size in a real-time manner, it would not be possible without a custom solution with a tailored architecture. In this application, two-dimensional image convolution is performed by scanning a rectangular image with a kernel function, a 15×15 two dimensional data array as shown in figure 2.1. For each kernel position, the overlapped data are multiplied and then summed together to form one output filtered image pixel. A boarder image frame is added to the raw image to prevent data missing when the position of filtering operation is close to the image boarder. The scanning pattern could be horizontal first and then vertical or the other way around. The kernel values represent the filter function that performs specific feature extractions on the image data. To facilitate flexible filtering alternatives, a RAM solution is usually demanded for the kernel changes in later use when other filtering function is applied. Kernel size is another parameter that can be changed as well. The larger the kernel size, the more the noise is reduced from the output image but more memories and calculations are needed. Therefore a trade off between performance and the hardware cost has to be made. For one example of grain quality assessment [6], it is shown that a kernel size of 15×15 is considered to be a reasonable solution in terms of the output image quality and relative implementation cost.

As a consequence of the image size, the complete image has to be stored on off-chip memory. During the convolving operations, each kernel position requires 15×15 pixel values from off-chip memory in a direct way. When this is performed in real-time, a very high data throughput is needed. Furthermore, accessing large off-chip memory is expensive in terms of power consumption and so is the signaling due to the large capacitance of package pins and PCB wires.

In [6], a tailored architecture with a streamlined dataflow was proposed to achieve the desired filtering while keeping a low I/O bandwidth. By the observation that each pixel value, except the one in the extreme corners, is used in several calculations, a two level memory hierarchy were introduced. instead of accessing off-chip memories 225 times for each pixel operation, 14 full rows of pixel values and 15 values on the 15th row are filled into 15 on-chip line memories before any convolution starts. After the first pixel operation,

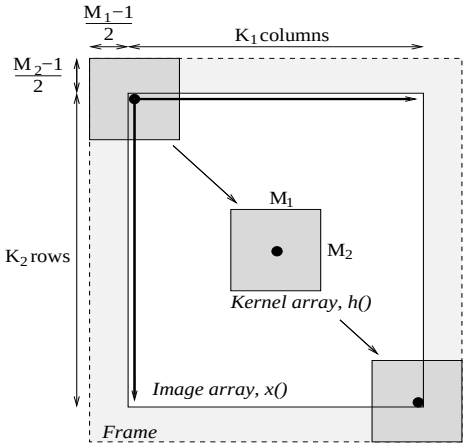


Figure 2.1: Convolution of an image by kernel function.

Table 2.1: memory hierarchy schemes and access counts for an image of 256×256 (M0=off-chip 0.18μm, C0 and C1=0.35μm)

Scheme	M0	C0	C1	energy cost
A: M0	13176900			790 mJ
B: M0→ C0	65536	13176900		56.6 mJ
C: M0→ C1	929280		13176900	68.9 mJ
D: M0→C0→C1	65536	929280	13176900	20.8 mJ

for each successive pixel position, the value in the upper left is discarded and a new value is read from the off-chip memory. As a result, pixel values from the off-chip memories are read only once for the whole convolving process from upper left to the lower right. Thus the input datarates are reduced from 15×15 accesses/pixel to only 1 read.

In light of the fact that using a two level memory structure can reduce both power consumptions and I/O bandwidth requirements, one extra cache level is introduced in this paper to further optimize for power consumptions during memory operations, figure 2.2. Since accessing large memory consumes more power, 15 small cache memories are added to the two level memory hierarchy. The small cache memories are composed of 15 separate memories to provide one column pixel values during each clock cycle. Instead of reading pixel values directly from line memories 15 times for each pixel operation, one column of pixel values are read to cache memories first from the line memories for each

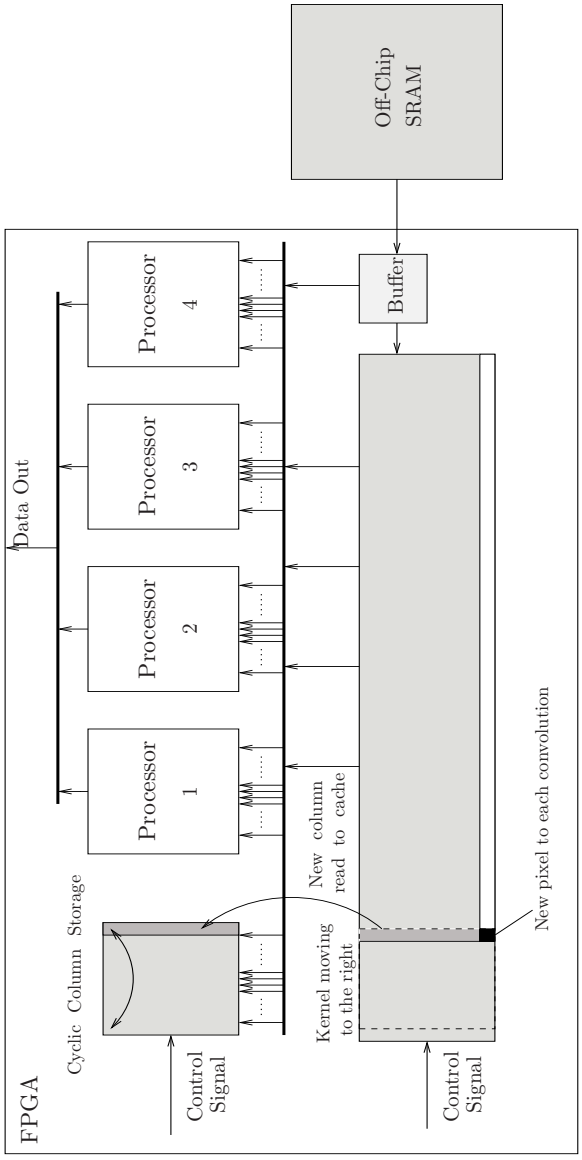


Figure 2.2: Block diagram of image processor datapath with 3 level memory hierarchies.

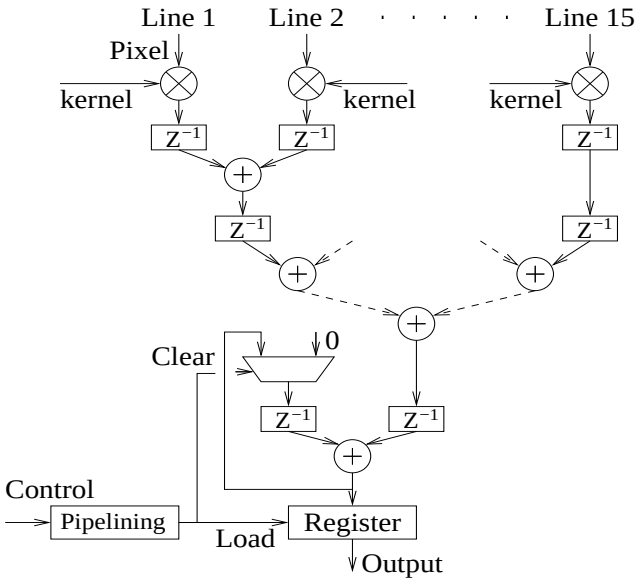


Figure 2.3: Adder tree structure of the processor core.

new pixel operation except for the first one. During each pixel operation, direct line memory access is replaced by small caches read. As a result, reading pixel values from line memories 15 times could be replaced by only once plus 15 times small cache accesses. Under the assumption that the embedded memories are implemented in a 0.35μm process CMOS, by the calculation method in [7], it is shown in Table. I that the power consumption for the total memory operations could be reduced to over 2.5 times compared to that of a two level hierarchy. In addition to the new cache memory, one extra address calculator is synthesized by proposed synthesis tool to handle the cache address calculation, the architecture of which is shown in the following section. In order to simplify address calculation, the depth for each cache is set to 16. In doing so, incrementing the largest address represented in 4 bits binaries will reset it back to the beginning. This will allow circular operations on the cache memories. During new pixel operations each new column of data are filled into the cache in a cyclic manner. However, the achieved low power solution has to be traded for extra clock cycles introduced during the initial filling of the third level cache memories. In the case of an image size of 256×256, this will contribute 61696 clock cycles in total. Compared with the total clock cycles in the magnitude of 10^7, such a side effect is negligible.

In fact, alternative schemes exists by different combinations of the three

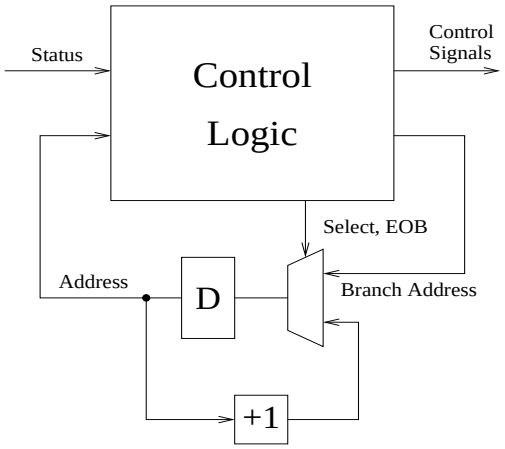


Figure 2.4: Controller Architecture with incremental circuitry.

memories in the hierarchy. All these possible solutions differ substantially in memory accesses. In Table. I, different memory scheme is shown where M0, C0, C1 denotes off-chip memory, line memories and smaller caches respectively. Although the number of data provided to the datapath remains the same for all four schemes, the access counts to the large off-chip memories varies. For the two and three level hierarchy structures, the counts to the large memory M0 are reduced by nearly 225 times compared to that of the one level solution. Between scheme B and D, the least access counts to both external memories and line memories are identified in three level memory structure, but this is achieved at the cost of extra clock cycles introduced during each pixel operation. Thus, trade off should be made when different implementation techniques are used. For the FPGAs where power consumption is considered less significant, two level hierarchies prevails due to its lower clock cycle counts. While for ASIC solutions, three level hierarchy is more preferable for it results in reasonable power reduction.

In addition to memory schemes, a pipelined adder tree is implemented on four processor cores to fulfil the sustained calculation capacity. Four cores have been chosen to be able to perform several convolutions in parallel. Each core contains 15 multipliers with adjoining RAMs filled with the kernel function. During each clock cycle, one column of data comprising 15 pixel values are read to the corresponding multipliers. The multiplication are performed in parallel and the products are summed in the subsequent adder tree. Combined with memory hierarchy, a very high sustained calculation capacity is obtained.

2.3 Controller synthesis

The processor cores require a very simple controller with only 2 control signals. However, the line memories and caches, plus RAMs accommodating kernel function values require extensive address calculations and loop control. In order to reduce the effort of the controller design, a synthesis tool is modified based on [1] to support synthesis process from C-like control algorithm specification to RTL controller module in VHDL format. The modified tool takes in a behavioral description of the datapath architecture defining the available set of micro-operations, and a microprogram written in C-like input syntax that contains the algorithm with additional declarations such as memories. As an output a complete controller with module descriptions and interconnection specifications is generated. Since architecture extensions to basic FSM could result in optimized controllers in specific application, a range of controller architectures are supported and can be accessed through architecture option specification before the synthesis starts. For the current image processor, a controller architecture with incremental circuitry is implemented, as shown in figure 2.4. In this architecture, the branch address calculation within the same block of code, composed of only sequential statements, is performed by the hardware incrementer. At the end of a block a non-incremental branch address is calculated by the control logic and a select branch signal is set. This architecture is particularly suitable for algorithms that have long stretches of sequential statements, i.e. the next state generation logic in the basic FSM architecture can be replaced by an incrementer and MUX. With the tool, more complicated architectures can be implemented freely based on user requirements and algorithm structures since it is a fully automated synthesis process.

In addition to the control structure, address processing units (APU) are also synthesized by the tool to perform address calculation. Due to the considerable data flow in calculating the memory addresses, APUs are usually implemented separately from controllers to reduce control logic. Declared in the microprogram, like variables in C, APUs are implemented in a common structure as shown in figure 2.5. To map the address calculations in the microprogram into hardware, constant value assignments to the APUs are implemented as one of the MUX constants. Additions and subtractions are performed in the adder and the results are stored in one of the register banks on either sides. Parameters such as image size is provided through the external input. To facilitate simple calculation descriptions in a microprogram, several sets of micro operations are predefined by the synthesis tool so that the user of the microprogram could use calculation functions like "++" and "--" with no knowledge of the implementation details. This will simplify the microprogram substantially in an address calculation intensive tasks.

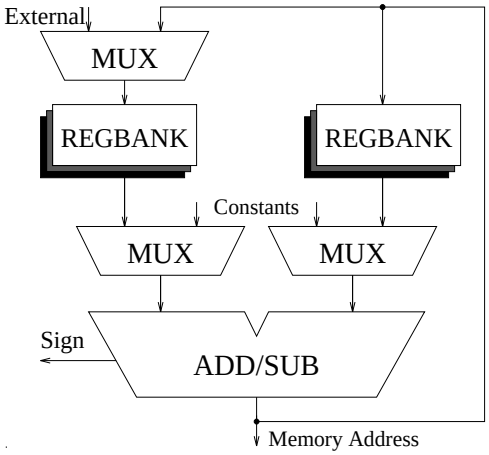


Figure 2.5: The address processing unit used in image convolution unit.

In the future, system level modelling language like SystemC is within special interest for its ability to specify both algorithm and hardware. With this support, future designs could go up further to the system level to enable more complicated system design and early verification.

2.4 Results

The control logic is generated in the format of espresso [8]–a PLA design format supported by Synopsys design compiler. Other formats are also likely in the future since the information stored in the database are truth tables and could be easily modified for other formats. The associated controller architecture as well as the datapath are implemented in VHDL. The whole system has been prototyped on Xilinx VirtexE 1156G FPGAs. Extra parts dealing with communication with PCs is developed to enable image transfer between PC and FPGAs. The whole design works under 20Mhz with a total equivalent gate count of nearly 800000. Matlab is used as an interface for both handling the communication on the PC side and results comparison with built-in filter functions. The image data are 8 bits unsigned values while the kernel functions are signed. The results are 24 bits signed values in the form of double precision matrix. The word length is only implemented for comparison with Matlab results. In real applications, scaling or rounding scheme can be used to reduce the output bits. To demonstrate the functionality, an edge detection operator Mexican hat is chosen to detect image edges in figure 2.6. Convolution results from different kernel size are given together with their zero crossings. Its dis-

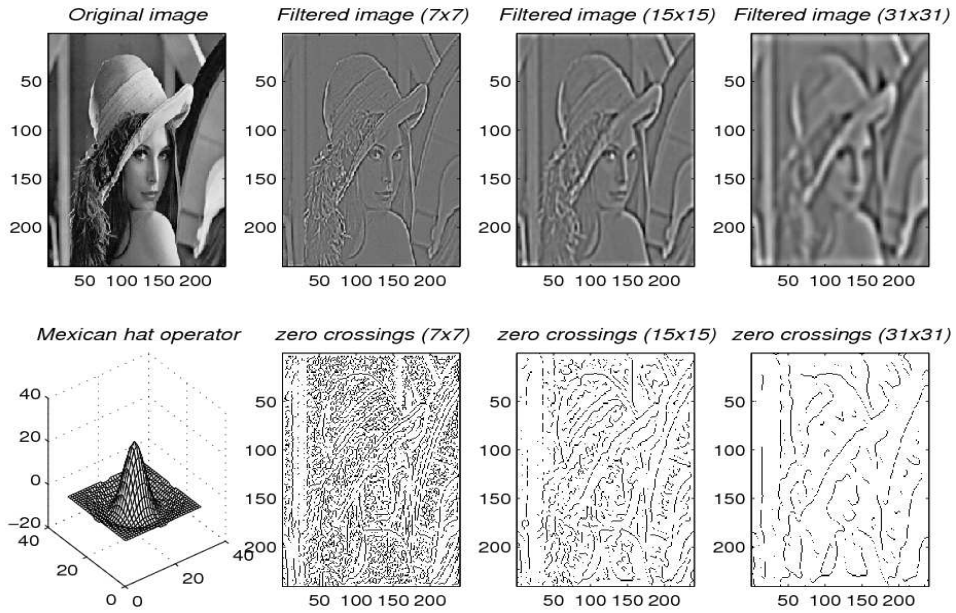


Figure 2.6: Edge detections using Mexican hat operator.

tribution in two dimensions may be expressed in terms of the radial distance r from the origin,

$$z(r) = \nabla^2 G(r) = K(1 - \frac{r^2}{2\sigma^2})e^{-\frac{r^2}{2\sigma^2}}.$$

The filtered image by Matlab functions is also given in the picture, by subtracting these two filtered images in Matlab, a zero plane is acquired meaning they are exactly the same.

2.5 Conclusions

The controller synthesis tool is powerful for the design of controller/datapath systems. By supporting higher level C-like control algorithm specification, the effort of controller design is reduced substantially. Memory hierarchy schemes are important factors for optimizing datapath designs. In this image application, both I/O requirements and power consumptions are reduced considerably. In future controller designs, reprogrammability is considered to be one of the major concerns. By enabling user reprogrammability, the user can often provide functionality equal to that of a programmable processor without any change

of the circuits. In recent years, this has already become a hot topic among networking and wireless communication system designs [9]. But it should be realized that level of programmability be taken as a trade off between performance and flexibility in the design space [10].

Bibliography

[1] V. Öwall, "Synthesis of controllers from a range of controller architectures," Ph.D. dissertation, Lund University, Lund, 1994.

[2] K. karadayi and Y. Kim, "Imaging with texas instruments TMS320C64x," Texas Instruments White paper, Tech. Rep., December 2001.

[3] E. Jamro and K. Wiatr, "FPGA implementation of addition as a part of the convolution," in *Proc. IEEE Euromicro Symposium on Digital Systems Design*, Warszawa, Poland, September 2001.

[4] ——, "Implementation of image data convolutions operations in FPGA reconfigurable structures for real-time vision systems," in *Proc. IEEE International Conference on Information Technology: Coding and Computing*, Nevada, USA, 2000, pp. 152–157.

[5] D. Bilsby, R. Walke, and R. Smith, "Comparison of a programmable DSP and an FPGA for real-time multiscale convolution," *IEE Colloquium on High Performance Architectures for Real-Time Image Processing*, pp. 4/1–4/6, February 1998.

[6] 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," *Springer Journal of VLSI Signal Processing Systems for Signal Image and Video Technology*, vol. 23, pp. 335–350, 1999.

[7] F. Catthoor, S. Wuytack, and E. D. Greef, *Custom memory management methodology*. Kluwer Academic Publishers, 1998.

[8] R. Rudell, "ESPRESSO - boolean minimization program," University of California at Berkeley, Tech. Rep., December 1985.

[9] K. Keutzer, "Bright future for programmable processors," *IEEE Design and test of Computers*, vol. 18, pp. 7–8, November-December 2001.

[10] N. Bang and R. W. Brodersen, "Architectural evaluation of flexible digital signal processing for wireless receivers," in *Proc. Thirty-Fourth Asilomar Conference on Signals, Systems and Computers*, California, USA, October 2000, pp. 78–83.