

## STATUS OF THESIS

Title of thesis

OPTIMIZED ARCHITECTURE DESIGN AND  
IMPLEMENTATION OF OBJECT TRACKING ALGORITHM  
ON FPGA

I LINA NOAMAN ABDELFATAH EL-KHATIB

hereby allow my thesis to be placed at the Information Resource Center (IRC) of Universiti Teknologi PETRONAS (UTP) with the following conditions:

1. The thesis becomes the property of UTP
2. The IRC of UTP may make copies of the thesis for academic purposes only.
3. This thesis is classified as

☐ Confidential

☒ Non-confidential


If this thesis is confidential, please state the reason:

\_\_\_\_\_  
\_\_\_\_\_  
\_\_\_\_\_

The contents of the thesis will remain confidential for \_\_\_\_\_ years.

Remarks on disclosure:

\_\_\_\_\_  
\_\_\_\_\_  
\_\_\_\_\_

  
\_\_\_\_\_  
Signature of Author

Permanent address: \_\_\_\_\_  
6, JLN Bandar U20, Bandar  
\_\_\_\_\_  
University, 31750 Tronoh, Perak,  
\_\_\_\_\_  
Malaysia

Date : 25/1/2013

Endorsed by   
**DR. FAWNIZU AZMADI HUSSIN**  
Senior Lecturer  
Electrical & Electronic Engineering

\_\_\_\_\_  
Signature of Supervisor

Name of Supervisor  
Dr. Fawnizu Azmadi Hussin

Date : 25/1/2013

UNIVERSITI TEKNOLOGI PETRONAS

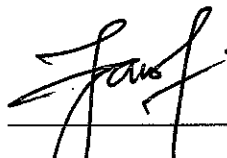
OPTIMIZED ARCHITECTURE DESIGN AND IMPLEMENTATION OF OBJECT  
TRACKING ALGORITHM ON FPGA

by

LINA NOAMAN ABDELFATAH EL-KHATIB

The undersigned certify that they have read, and recommend to the Postgraduate Studies Programme for acceptance this thesis for the fulfillment of the requirements for the degree stated.

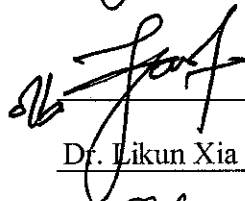
Signature:

  
**DR. FAWNIZU AZMADI HUSSIN**  
Senior Lecturer  
Electrical & Electronic Engineering

Main Supervisor:

Dr Fawnizu Azmadi Hussin

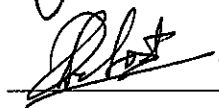
Signature:



Co-Supervisor:

Dr. Likun Xia


Signature:



Co-Supervisor:

Mr. Patrick Sebastian.

Signature:



Head of Department:

Dr. Rosdiazli Bin Ibrahim

Date:

**DR ROSDIAZLI B. IBRAHIM**  
Deputy Head/Senior Lecturer  
Electrical & Electronic Engineering Department 30/01/2013

OPTIMIZED ARCHITECTURE DESIGN AND IMPLEMENTATION OF OBJECT  
TRACKING ALGORITHM ON FPGA

by

LINA NOAMAN ABDELFATAH EL-KHATIB

A Thesis

Submitted to the Postgraduate Studies Programme

as a Requirement for the Degree of

MASTER OF SCIENCE

ELECTRICAL AND ELECTRONICS ENGINEERING

UNIVERSITI TEKNOLOGI PETRONAS

BANDAR SERI ISKANDAR,

PERAK

JANUARY 2013

## DECLARATION OF THESIS


Title of thesis

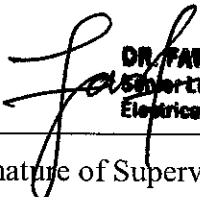
OPTIMIZED ARCHITECTURE DESIGN AND  
IMPLEMENTATION OF OBJECT TRACKING ALGORITHM  
ON FPGA

I LINA NOAMAN ABDELFATAH EL-KHATIB

hereby declare that the thesis is based on my original work except for quotations and citations which have been duly acknowledged. I also declare that it has not been previously or concurrently submitted for any other degree at UTP or other institutions.

Witnessed by

  
\_\_\_\_\_  
Signature of Author

  
\_\_\_\_\_  
Signature of Supervisor

**DR. FAWNIZU AZMADI HUSSIN**  
Senior Lecturer  
Electrical & Electronic Engineering

Permanent address: \_\_\_\_\_  
6, JLN Bandar U20, Bandar  
\_\_\_\_\_  
University, 31750 Tronoh, Perak,  
\_\_\_\_\_  
Malaysia

Name of Supervisor  
Dr. Fawnizu Azmadi Hussin

Date : 25/1/2013

Date : 25/1/2017

## DEDICATION

To almighty Allah that without his guidance and protection this work would never be achieved. To my beloved leader Prophet Mohammad peace be upon him whose teachings inspires me always to be a better person and go forward in this life.

To my beloved home Palestine, hoping that this work to contribute to its freedom in the near future.

To my first university, the Islamic University of Gaza, where I took the oath of professional ethics as a graduate engineer for the first time in my life,

To my beloved parents, Prof. Noaman El-khatib and Mrs. Mariam Abu-Hasanain, who really shared with me this work moment by moment by their care, patience and prayers.

To my beloved brothers and sisters, who always supported and encouraged me.

To my uncle the late Prof. Dr. Fawzy Abu-Hasanin, who would have been very proud to witness me reaching this goal.

To my best lasting friend, my childhood friend, Gadeer Hakeem, who was always there in pleasant and difficult moments despite the long distances separating us.

I present this work.

## ACKNOWLEDGEMENTS

First all praises and thanks to Almighty Allah for his sustainable uncounted help at all times to achieve this work. I would like to acknowledge the support of the Electrical and Electronic Engineering Department of Universiti Teknologi PETRONAS (UTP) during this study. Also I am grateful to my supervisors, Dr. Fawnizu Hussin, Dr. Likun Xia and Mr. Patrick Sebastian for their guidance and support during the execution of this research. I would like also to direct my special thanks to Prof. Noaman El-khatib, Dr. Aamir Saeed Malik and Dr. Sayed Nasir Shah, for their great help and discussions. A special thanks to Mr. M Sani B Selmat (the former EE chief technician at UTP) for his cooperation in providing access to all lab facilities I needed for my research. Great thanks to my colleagues Roshaslinie Ramli, Junaid Ahmad, and Yasir Salih who didn't hesitate to offer their time and advice. I am also grateful to my lab mates "FPG old fighters", Zahra Osman, Ngo Huy Tan and Ruzali, who spent days and nights together working in the lab, sharing good and bad times, for two and a half years.

Kind thanks to my best friends in UTP, Muna Sadig, Rere Rahardjati, Areej Babkir and Ahlam Elsafy for standing always beside me. Thanks also for my house mates, Aisha Osman, Husna Talbany, Roushanak Rahmat, Ambreen Shafeeq, Zahra Osman, Noor and Hafeezah, and for my friends Yasreen Suliman, Aicha Shalabby, Mawaheb and Sureena for their kindness and support. Also thanks to V5 block H girls in UTP hostel, who were at hard times like a shadow in a hot sunny day.

At the end I want again to thank my beloved parents, my brothers and sister for their unstopped and nonrefundable support and help. Finally special thanks to my beloved University, UTP, for giving me this opportunity to complete my academic study, where I spent precious years of my life learning research and more, that became not just a place to get a degree, but a second home and nationality to me, I will be always grateful and belonging to it.

## ABSTRACT

FPGA based Object tracking implementation is one of the most recent video surveillance applications in embedded systems. In general, FPGA implementation is more efficient than general purpose computers in attaining high throughput due to its parallelism and execution speed. The system need to be designed on a standard frame rate in such a way to achieve optimal performance in real time environment. Optimal design of a system is dependent on minimizing the cost, area (device utility) and power while achieving the required speed. Past research work that investigated object tracking systems' implementation on FPGA achieved a significantly high throughput but have shown high device utilization. This research work aims at optimizing the device utilization under real time constraints. The Adaptive Hybrid Difference algorithm (AHD), which is used to detect the moving objects, was chosen to be implemented on FPGA due to its computation ability and efficiency with regard to hardware implementation. AHD can work at various lighting conditions automatically by determining the adaptive threshold in every period of time. Compared to other tracking algorithms based on segmentation, subtractions of pixels are done between number of frames and the results are accumulated. Therefore the logic circuit required by AHD is simple but the process of accumulating the results through different time frames could potentially slow the system. To overcome this bottleneck, a system based on AHD algorithm has been proposed, designed and implemented to obtain the minimal logic utilization under real time computation requirement. Pipelining, line buffers and parallelization have been used to develop the internal design in order to increase the throughput. System frequency is set to provide data synchronization between the memory and processing units. With the 640×480 frame size, the proposed system has achieved a frame rate of 32.55 fps with device utilization of 1,821 out of 33,216 logic elements and 74,192 bits out of 483,840 bits of the device's internal memory. The resource usage of the proposed design is less than what have been achieved by the entire work in the literature.

## ABSTRAK

FPGA yang berasaskan implementasi penjejakan objek merupakan salah satu aplikasi video pemantauan yang paling terkini dalam sistem berprogram. Secara umumnya, implementasi FPGA adalah lebih cekap daripada komputer kegunaan am dalam mencapai daya pemprosesan yang tinggi kerana keselarian dan kelajuan pelaksanaannya. Sistem ini perlu direka mengikut kadar piawaian yang sedemikian untuk mencapai prestasi yang optimum dalam dunia sebenar. Reka bentuk sistem yang optimum adalah bergantung kepada minima kos, kawasan (utiliti peranti) dan kuasa ketika mencapai kelajuan yang diperlukan. Berdasarkan penyelidikan yang sebelum ini, FPGA yang berasaskan implementasi penjejakan objek mencapai daya pemprosesan yang sangat tinggi tetapi menggunakan peranti yang banyak. Penyelidikan ini bertujuan untuk mengoptimumkan penggunaan peranti di bawah kekangan masa sebenar. Kerja-kerja penyelidikan ini bertujuan untuk mengoptimumkan penggunaan peranti di bawah kekangan masa sebenar. Algoritma Perbezaan Adaptive Hibrid (AHD), yang digunakan untuk mengesan objek yang bergerak, telah dipilih untuk diimplementasikan pada FPGA berdasarkan keupayaan pengkomputeran dan kecekapan dalam mengimplementasikan peranti keras. Jika dibandingkan dengan algoritma penjejakan menggunakan segmentasi yang lain, operasi penolakan piksel antara bilangan bingkai telah dilakukan dan keputusan berjaya dikumpulkan. Oleh itu litar logik yang diperlukan oleh AHD adalah ringkas tetapi proses mengumpulkan keputusan dalam jangka masa yang berbeza menyebabkan sistem menjadi perlahan. Untuk mengatasi masalah ini, satu sistem yang menggunakan algoritma AHD telah dicadang, direka dan diimplementasikan untuk meminimumkan penggunaan logik bergantung kepada keperluan pengkomputeran dalam masa sebenar. Penggunaan teknik-teknik khas seperti saluran maklumat dan keselarian telah dipertimbangkan untuk meningkatkan daya pemprosesan. Frekuensi sistem telah ditetapkan untuk menyelaraskan data antara memori dan unit pemprosesan. Dengan bingkai saiz  $640 \times 480$ , sistem yang dicadangkan telah mencapai kadar bingkai 32.55 fps dengan penggunaan peralatan



1,821 daripada 33,216 elemen logik dan 74,192 bit daripada 48,384 bit memori dalaman peranti. Penggunaan sumber untuk sistem yang dicadangkan dalam penyelidikan ini adalah berbandingurauh daripada apa yang telah dicapai oleh penyelidikan-penyelidikan sebelum ini.

In compliance with the terms of the Copyright Act 1987 and the IP Policy of the university, the copyright of this thesis has been reassigned by the author to the legal entity of the university,

Institute of Technology PETRONAS Sdn Bhd.

Due acknowledgement shall always be made of the use of any material contained in, or derived from, this thesis.

---

© LINA EL-KHATIB, 2012

Institute of Technology PETRONAS SdnBhd

All rights reserved.

## TABLE OF CONTENT

CHAPTER 1 INTRODUCTION.....	1
1.1 Video Surveillance Systems and Image Processing .....	1
1.2 Moving Object Tracking Algorithms .....	2
1.3 Using Field Programmable Gate Array (FPGA) for Image processing.....	3
1.4 Problem Statement.....	4
1.5 Research Objectives.....	7
1.6 Research Scope.....	7
1.7 Contribution .....	8
1.8 Thesis Outline.....	8
CHAPTER 2 LITERATURE REVIEW .....	10
2.1 Chapter Overview .....	10
2.2 Object Detection and Object Tracking Algorithms .....	11
2.2.1 Point Detectors.....	11
2.2.2 Background Subtraction .....	12
2.2.3 Segmentation.....	16
2.2.4 Supervised Learning.....	18
2.3 Implementation of Object Tracking Algorithms on FPGA.....	19
2.3.1 Image Segmentation based on Color Threshold .....	19
2.3.2 Image Segmentation and Pattern Matching .....	21
2.3.3 Image Gravity Center and Matched Filter for Planer Motion Tracking.	24
2.3.4 Color Detection and Binarization Using HW/SW co-design .....	26
2.3.5 Principal Component Analysis based Planar Motion Tracking .....	28
2.3.6 Summary of Object Tracking Algorithms Implemented on FPGA .....	30
2.4 Chapter Summary .....	31
CHAPTER 3 SOFTWARE SYSTEM DESIGN AND IMPLEMENTATION	
STRATEGY .....	33
3.1 Chapter Overview .....	33
3.2 Software Design .....	33
3.2.1 Hybrid Difference Strategy.....	35

3.2.2 Adaptive Threshold Initialization.....	36
3.3 Software Selection.....	37
3.4 Software Development .....	38
3.5 Software Verification and Evaluation .....	40
3.6 Chapter Summary.....	43
CHAPTER 4 HARDWARE ARCHITECTURE AND IMPLEMENTATION .....	45
4.1 Chapter Overview .....	45
4.2 Hardware Preparation.....	45
4.3 Key Optimization Parameters.....	46
4.4 Analytical Study.....	49
4.4.1 Image Representation and Data Row.....	49
4.4.2 AHD Analysis Based Hardware Implementation.....	51
4.5 Proposed Design for the Hardware System Architecture on FPGA .....	55
4.6 Threshold Definer Unit's Frequency, Functionality and Architecture .....	58
4.7 Binary Image Builder Unit's Frequency, Functionality and Architecture .....	61
4.8 Finite State Machine and Control Signals .....	67
4.9 Pipelining and Data Flow .....	73
4.9.1 Pipelining of the Threshold Definer Unit.....	74
4.9.2 Pipelining of the Binary Image Builder Unit .....	78
4.10 Chapter Summary.....	79
CHAPTER 5 RESULTS AND DISCUSSIONS .....	81
5.1 Chapter Overview .....	81
5.2 Experimental Setup and Deployment Strategy.....	81
5.3 Baseline Approaches .....	82
5.4 Performance Analysis.....	83
5.5 Synthesis Utilization.....	86
5.6 Comparison with Other Systems.....	87
5.7 Logic Utilization Estimated Unit .....	87
5.8 Comparison with Other Systems.....	90
5.9 System Evaluation.....	93
5.10 Chapter Summary.....	95
CHAPTER 6 THESIS SUMMARY .....	96

REFERENCES..... 99

APPENDIX A FEATURES AND DESIGN CHARACTERISTICS OF THE DE2  
BOARD ..... 106

APPENDIX B K VALUE TESTING..... 109

## LIST OF FIGURES

Figure 2.1: Block diagram of object tracking system using color threshold segmentation [11].....	20
Figure 2.2: Block diagram of FPGA/ASIC implementation architecture for object tracking system using segmentation and pattern matching [22].....	22
Figure 2.3: Data flow of the pipeline processing architecture of object tracking system using segmentation and pattern matching [22].....	23
Figure 2.4: The Architecture block diagram for implementation of planar motion tracking on CMOS image detector and FPGA [18].....	25
Figure 2.5: Hardware and software integrated architecture of skin color detection system proposed [24]. .....	27
Figure 2.6: Block diagram for the internal architecture of the tracking system design on FPGA for PCA algorithm [10].....	29
Figure 3.1: Flow diagram of AHD algorithm [4]. .....	34
Figure 3.2: Original and binary images obtained by applying AHD algorithm on different frames for moving objects.....	40
Figure 3.3: Original and binary images obtained of applying AHD algorithm on different frames for moving vehicles. ....	41
Figure 3.4: Original and binary images obtained by applying AHD algorithm on [4]. .....	43
Figure 4.1: (a) Architecture of the basic elements of generic FPGA (b) A simplified architecture of logic block [6].....	48
Figure 4.2: Frame sequence technique to determine the threshold value, where $k=6$ . 52	
Figure 4.3: The system hardware architecture on FPGA .....	55
Figure 4.4: Architecture of the Threshold Definer Unit.....	58
Figure 4.5: Improved architecture of the Threshold Definer Unit.....	60
Figure 4.6: Architecture of Binary Image Builder Unit. ....	62
Figure 4.7: Improved Architecture of Binary Image Builder Unit .....	64
Figure 4.8: Processing flow between FIFOs, line buffers and BIB unit on the time line at running time for the improved Binary Builder unit. ....	65

Figure 4.9: The improved internal architecture of BIB unit .....	66
Figure 4.10: Main state diagram of AHD finite state machine.....	68
Figure 4.11: The memory map of frames and lookup tables (LUT) on the SDRAM. ....	69
Figure 4.12: Pipelining data path of Threshold Definer unit .....	75
Figure 4.13: Pipelining data path of Binary Image Builder unit.....	78
Figure 5.1: Deployment Model.....	82
Figure 5.2: Detected moving object snapshots .....	86
Figure 5.3: Altera, Cyclone II Logic element, inner architecture [45].....	88
Figure 5.4: Figure 5.3: Xilinx, Spartan II slice, inner architecture [56].....	89
Figure 0.1: k value testing.....	110

## LIST OF TABLES

Table 2.1: Summary of point detectors algorithms.....	12
Table 2.2: Summary of background subtraction algorithms.....	15
Table 2.3: Summary of segmentation algorithms.....	18
Table 2.4: Summary of supervised learning algorithm.....	19
Table 2.5: Summary of image segmentation based on color threshold on FPGA.....	21
Table 2.6: Summary of image segmentation and pattern matching algorithm on FPGA .....	24
Table 2.7: Summary of matched filter algorithm on FPGA.....	26
Table 2.8: Summary of color detection and binarization using HW/SW co-design....	28
Table 2.9: Summary of principal component analysis on FPGA.....	30
Table 2.10: Summary of object tracking algorithms implemented on FPGA.....	31
Table 3.1: Jaccard coefficient of binary images obtained in Figure3.2.....	42
Table 3.2: Jaccard coefficient for binary images obtained from video of moving vehicles.....	42
Table 4.1: Pipelining data flow table of Threshold Definer unit.....	77
Table 4.2: Pipelining data flow table of Binary Image Builder unit.....	79
Table 5.1: Summarization of performance analysis for the developed designs.....	86
Table 5.2: FPGA's Synthesis Utility of Our System.....	87
Table 5.3: Resource Usage of different tracking systems on FPGA.....	91
Table 5.4: Performance and Speed achieved by the implemented tracking systems on FPGA.....	92
Table 5.5: Frames size and Device types used with the implemented tracking systems on FPGA.....	92



## LIST OF ABBREVIATIONS

Abbreviation	Description
CCD	Charged coupled device
CMOS	Complementary metal oxide sensor
CCTV	Closed circuit television
AHD	Adaptive hybrid difference
FPGA	Field programmable gate array
CLB	Configurable logic blocks
LUT	Lookup table
VLSI	Very-large-scale integration
HDL	Hardware description language
FIFO	First in first out
SIFT	Scale invariant feature transform
SGM	Single Gaussian mixture
MOG	Mixture of Gaussian
GMM	Gaussian mixture model
ASIC	Application-specific integrated circuit
LVDS	Low-voltage differencing signaling
PCI	Peripheral component interconnect
2DFFT	Two-dimension fast Fourier transform
2D-IFFT	Two-dimension inverse fast Fourier transform
SD	Secure digital
PCA	Peripheral component interconnect
RAM	Random access memory
HD	Hybrid difference
AT	Adaptive threshold
IPL	Intel image processing library

Appreciation	Description
SDRAM	Synchronous dynamic random access memory
VGA	Video graphics array
ABS	Absolute
TD	Threshold definer
BIB	Binary image builder
LCD	Liquid crystal display
FSM	Finite state machine

---

## LIST OF NEMONCLATURE

Nomenclature	Description
$I$	Current
$V$	Voltage
$f$	Frequency
YUV	Luminance / chrominance color model
RGB	Red, green and blue color model
Fps	Frame rate (frame per second)
HSV	Cylindrical-coordinate representations of points in an RGB color model.
BMP	File name extension for the Bitmap image file format
LE	(Logic element), the smallest logic of FPGA device
M	(Mega), a prefix in the metric system denoting a factor of million ( $10^6$ or 1000000)
Hz	(hertz), the unit of frequency
$t$	Frame order among frames of video stream
$k$	The near $k$ frame, used to find difference of Frame $t$ and $(t-k)^{th}$ frame instead of $(t-1)^{th}$ frame.
$n$	Number of pixels represent the frame width
$m$	Number of pixels represent the frame high
$G$	A matrix represents a single frame of a video stream
$(i,j)$	The position of the pixel
$G^t(i,j)$	Pixel value of Frame $t$ located in position $(i,j)$
$D^t$	Pixel difference of Frame $t$ with pixel in the same position of another frame
$T$	Threshold
$\mu$	Mean

Nomenclature	Description
$\sigma$	Standard deviation
$N$	The number of frames used to set the adaptive threshold
$B^t$	The binary value (black or white) of a pixel from frame $t$
$p$	A natural number denotes the total number of frames
$J$	Jaccard Coefficient, used to evaluate the results accuracy
$TP$	(True positives), number of correctly detected moving pixels
$FP$	(False positives), number of false detected moving pixels
$FN$	(False negatives), number of missed detected moving pixels
$NJ$	(Numerical Jaccard) gives the detection accuracy for sequence of frames
Byte	Unit of digital information consists of eight bits
Bit	The basic capacity of information in computing and telecommunications that represents either 1 or 0 only
Pixel	A finite discrete quantity that represents the intensity of a point of the image
bpp	(Bit per pixel), the number of bits used to specify the color of a single pixel, also known as color depth

## CHAPTER 1

### INTRODUCTION

#### 1.1 Video Surveillance Systems and Image Processing

Video surveillance systems are systems that use video cameras for security and safety purposes such as tracking suspects and detecting dangers. The vital need of guard use started with the Second World War. However, using cameras for security systems were not significantly popular until the electronic technology of the tube-type camera has been developed by 1960s to 1970s. Years later, during 1980s and early of 1990s, solid state cameras that use a Charged Coupled Device (CCD) image sensor were introduced to the world. Gradually, CCD cameras started to be used for security systems instead of tube cameras. The drawbacks of tube sensor that affects image quality and overall performance with repeated usage has encouraged the use of solid state sensors. In contrast with the solid state sensors, metal oxide semiconductor (MOS) and complementary MOS (CMOS) sensor cameras are stable and not effected by aging. Moreover, producing these cameras with affordable prices had rapidly spread using the video for surveillance systems. Thus, as a result, the first closed-circuit television (CCTV) surveillance was introduced by 1995, which uses cameras to transmit a video to set of monitors in a specific place. Since computer technology has integrated with video surveillance technology, the golden age of surveillance systems with the use of CCD and CMOS cameras flourished between 2001 and 2005. Nowadays, by the advance of digital image transmission and digital storage production, concurrent with the improvement of Internet video transmission protocols, video surveillance systems have spread and occupied a vital role in the current security industry [1].

The use of digital surveillance systems has led to the conversion of the video stream into a sequence of digital images or frames. A digital image is represented by two dimensional function  $f(x,y)$ , where  $x$  and  $y$  are the plan coordinates of the image. The function's value at any coordination is a finite discrete quantity that represents the intensity of that point of the image, which is called a pixel [2].

Based on digital imaging, the trend nowadays is to process pixels' values to perform a specific task over videos rather than using videos for general monitoring. Many of image processing algorithms have been developed to apply specific surveillance tasks on videos through the time such as detecting and tracking moving or specific objects, shape, face and color recognition and catch over-speeding vehicles. With the use of digital image processing, the goal of surveillance has been enhanced from general observation to understanding what is happening in the scene [3]. This saved effort and cost of employing people to monitor videos. From the other side, it reduces chances of mistaking the danger and helps to detect the cause of danger immediately.

## **1.2 Moving Object Tracking Algorithms**

Tracking moving objects is a common application of video surveillance systems that are used for security purposes. First the algorithm must detect the moving object (the foreground) over still objects (the background). The outcome of detecting objects is used later for an advanced level of processing. Detection algorithms depend basically on the key features of detection; they are motion, color, specific shape or other features. For example with regard to motion, the detection algorithm based on detecting moving objects are different from the ones aiming at detecting all objects whether they are moving or still. Usually for object detection, segmentation algorithms are used, since segmentation can detect moving or still objects over a frame. However segmentation algorithms are also able to detect motion by tracking objects with the same features at different positions over the frames. Another way to detect moving objects is to use information related to movement from the first step of the algorithm such as subtracting frames from an idle background or using objects

difference through frames. Therefore, subtraction algorithms have been developed and used for motion detection purposes, since they depend on frames and background's pixels intensity differences. Subtraction algorithms procedures are simple, however through the research, background subtraction was not the preferable algorithm among object detection and tracking algorithms. This is because the classic image subtraction could be affected by noise, illumination changes and object occlusions. The background subtraction researchers have developed the classic subtraction to avoid its drawbacks by building background models that learn by light intensity changes with time over the background. An example of that is an algorithm developed in 2006 by [4] called the Adaptive Hybrid Difference (AHD) algorithm which uses 30 frames to build background model using the mean and standard deviation values of background pixels. The AHD algorithm can achieve results with better quality within faster time compared to other background modeling algorithms as will be explained in section 2.2.2.

### **1.3 Using Field Programmable Gate Array (FPGA) for Image processing**

Field programmable gate array (FPGA) is a large integrated circuit with the ability of being reconfigurable. FPGAs consist of configurable logic blocks (CLB) as the basic internal elements, programmable routing, and I/O blocks. The CLB architecture is different from one FPGA to another. However, in common the CLB of any FPGA consisting of one or more multiple input lookup table (LUT) and flip flop used to register the LUT's output. Mainly, FPGAs are signified by the number of logic blocks available to be configured which reach tens of thousands for a single chip. By the advancement of VLSI industry, chips with the same size are having larger internal hardware capabilities. With that large amount of logic blocks, an entire system can be built on FPGA device [5]. It is important to mention here that the two leader vendors of FPGA manufacture are Altera Corporation and Xilinx Inc. [6-8].

The nature architecture of FPGAs mentioned in the previous paragraph has led to advantages of using FPGA to implement hardware designs. First, being a reconfigurable device, it saves time and cost during system's design stage. Second,

with the programmable logic blocks and programmable routing, it offers high flexibility for system design. Third, the internal architecture of FPGAs gives it the advantage of parallel processing feasibility.

Large part of image processing algorithms depends on applying specific process on all or number of pixels of the current frame in parallel. In order to save running time which is important in real time applications that are also required for some image processing algorithms, the use of parallel implementation with image processing is an important issue. Thus, parallel implementation ability offered by FPGA devices was the main reason of preferring the use of FPGA rather than general purpose computers for implementing image processing algorithms. As a result, implementing image processing algorithms using FPGAs has enabled faster and more effective surveillance applications.

#### **1.4 Problem Statement**

In the section 1.3, advantages of FPGAs for implementing image processing and surveillance systems have been highlighted. In contrast the problem statement of this research is represented through some of the FPGAs' disadvantages. Using the parallelism capabilities of FPGAs, many hardware systems of video surveillance algorithms with parallel nature have been designed. However, incorrect FPGA designs may cost extra unnecessary gates, area and power dissipation. In building a large system, a big challenge is to achieve the optimal design, cost, and power dissipation of the system. The power dissipation of any CMOS circuits is related to charging and discharging capacitance on gates and metal traces. The current dissipation in a capacitor is driven by the following equation

$$I = V \times C \times f$$

where  $I$  is the current,  $V$  is voltage,  $C$  is capacitance and  $f$  is frequency. In order to reduce the current dissipation, voltage, capacitance or frequency must be reduced. The voltage in FPGA design is fixed, while the frequency and capacitance can be controlled through design. For the capacitance, it is directly related to the



number of gates toggled at any time and the length of the routes between them [9]. Thus by reducing the number of gates the power could be reduced. As a result reducing the design area could help to reduce the length of routes which will reduce the capacitance. While reducing the frequency will decrease the power dissipation, it will slow the system speed. Specifying design frequency depends on the system speed that the system has to meet. In addition, the frequency usually must meet the design requirement to synchronize data between internal logics and to match timing of the system. For object tracking applications specifically, the speed of the system during running time is an important factor. The applications of moving object tracking must be able to detect immediately the suspects or dangers. Therefore, limiting system speed to save power must not go beyond the required speed for a real time system. To measure the execution speed, the frame throughput which is the number of frames processed per second, is used. The video application must work with a speed of 30 frames per second to achieve real time system's conditions. Design speed is also affected by the input data latency to get output, and by the timing between sequential elements of design paths.

In addition to achieving optimized power, device resources and design speed, there are other challenges in building object tracking system on FPGA. One challenge is to work between different clock domains through the camera, the memory and the processing units. Data synchronization for the inputs and outputs of the developed processing unit must be fulfilled.

Since the number of frames and pixels needed during processing the object tracking algorithms is large, there is a need to use external memory. Therefore the design of processing unit is governed by the memory specifications. Memories differ in the amount of data that could be read or written within a clock cycle according to the width of data bus and whether the memory has dual or single port. Implementing parallelization needs data from different frames or frame locations to be ready at the same time. For example, using a 16 bits single port memory made it necessary to look for techniques and solutions to make the data ready for parallel execution. The speed bottleneck between storage devices and processing unit is one of the main problems faced while designing the tracking system on FPGA.

All of the solutions to build the design should maintain the optimization of the device resources, power dissipation and achieve the real time speed at the same time, which is a big challenge.

The previous works that have implemented tracking moving objects on FPGAs using segmentation algorithms, had utilized the advantage of FPGAs' parallel architectures to achieve high throughputs. However, the hardware usage for implementing those algorithms is high. Another researcher [10] used the Principal Component Analysis algorithm, but the complexity of the algorithm's steps have generated a high logic utility with difficulty to develop that code with HDL. Results of background subtraction could be affected by noise, illumination changes and object occlusions. Pixels could be detected wrongly when the background information alters as a result of lighting intensity changes over time. Using segmentation algorithms that detect object based on testing pixels of the current frame rather than using background model overcomes the dynamic illumination problem. However segmentation algorithms' procedures are more complex than subtraction in terms of the operations required although it gives high ability of parallel implementation, the resulted synthesized logic is high. The use of background subtraction algorithms is not widely applied for implementing moving object tracking on FPGA since the drawbacks of the typical subtraction algorithms did not encourage its implementation on FPGA. However, the simplicity of subtraction algorithms steps, and the ability of achieving minimum cost, logic and power design are good reasons to attempt improved version of them again.

The AHD algorithm is a subtraction based algorithm that tracks moving objects with an enhanced subtraction algorithm which uses adaptive threshold values. Even the steps of the algorithm are simple, computerized, and can be run automatically without the need to assume parameters as the case with some tracking algorithms, still there are challenges to implement it on hardware. The algorithm requires a lot of operations that have to be implemented for every pixel of the frame. In addition, it needs to accumulate the results of processing pixels through 30 subsequent frames in order to find the thresholds. It also needs to keep the differences between the current frame and other six frames for each new frame at tracking time. Accessing this data

through the memory with limited data width port could affect processing speed. At the meantime, the bandwidth of the memory is very critical, since it specifies the speed at which the system should work in order to achieve the real time condition. While parallelization could be used to speed up the system, it increases the generated logic units used in parallel to process data. From another side, reducing the utilization of the resources for the design would affect the speed of execution. This research has been initiated to overcome this bottleneck and to meet real time conditions while optimizing the resource utilization of the design.

### **1.5 Research Objectives**

This research investigates appropriate techniques and methods to build an HDL design of the AHD algorithm on FPGA to achieve a tracking system with optimal logic utility, power and cost in real time environment.

The main objectives of this study are:

1. To design a real time moving object tracking system using subtraction based algorithm
2. To implement the designed moving object tracking system on FPGA.
3. To evaluate the feasibility of implementing adaptive hybrid difference algorithm on FPGA at real time with minimum logic utility.

### **1.6 Research Scope**

This research is meant to design and implement an optimal AHD processor architecture on FPGA to detect the moving objects in real time environment. Thus, non-moving objects are not subject of detection. The parameters to be optimized are logic elements, memory blocks utilization, and throughput of the system. Because it is difficult to estimate power dissipation and cost, they are not directly considered in the evaluation. However they are realized as a result of logic utilization reduction since

they are proportionally related. The minimum throughput aimed to be reached is at least 30 fps, which is the real time system throughput for video applications. System design has considered the frame size of  $640 \times 480$  pixels.

## **1.7 Contribution**

The contribution of this research is AHD processor architecture that implements tracking moving objects system on FPGA with optimized resource usage (1,821 from 33,216 of logic elements and 74,192 bits from 483,840 bits of memory on chip). The processor achieves real time system's speed with 32.55 fps frame rate and frequency of 20 MHz using frame of size  $640 \times 480$  pixels. The architecture has used pipelining technique to optimize and speed up processing time required for each pixel from 5 to 1 clock cycle. Buffering data using FIFOs and line buffers to synchronize dataflow between memory and processing units' clock domains was used with pipelining to achieve the maximum processing speed.

## **1.8 Thesis Outline**

This thesis consists of six chapters, thesis overview, literature review, software system design and implementation strategy, hardware architecture and implementation, results and discussions, and thesis summary.

Chapter 1 describes the background of study, problem statement, objectives, scope and contributions.

In Chapter 2 more detailed on theoretical background of tracking algorithms has been highlighted. Algorithms were related to the literature review of the works which developing them. The chapter then displays the literature work of implementing some of these tracking algorithms on FPGA devices by some researchers. The study concentrates on displaying designs and implementations, while results and evaluation has been left to be discussed in the chapter of results and discussions.

Thesis Methodology is distributed among Chapters 3 and 4 since each is discussing the work from a different perspective. For Chapter 3, it presents software design and development system of the Adaptive Hybrid Difference. Through the chapter, the software implementation of the AHD algorithm is verified and evaluated as a pre-step before implementing it on the Hardware.

Chapter 4 illustrates in details the proposed design and implementation of the AHD on the FPGA. An analytical study and image representation of the algorithm from hardware perspective have been discussed. Additionally, system specifications and key optimization parameters meant to be reached through the design have been identified. The chapter detailed the hardware architecture of the two core units of our design, the Threshold Definer unit and the Binary Image Builder unit. Architectures of both units are appended with frequency and functionality explanation via the data flow through the system. Also the pipelining and other techniques used to optimize the system are illustrated.

Results of the hardware design and implementation have been discussed and evaluated in Chapter 5. System performance has been analyzed and a synthesis utility of the system has been compared with other systems mentioned previously in the literature review. Finally an evaluation of this work is set at the end of chapter 5.

Chapter 6 summarizes the thesis in terms of achieved contributions and recommended work. It emphasizes on the contributions of this research.

## CHAPTER 2

### LITERATURE REVIEW

#### 2.1 Chapter Overview

Object tracking and detection are among the prime applications of video surveillance systems from implementation perspective. Some of the literature has also focused on the monitoring aspect of implementation of such systems. Recently, object tracking and detecting algorithms have been developed in the field of video surveillance systems. A number of algorithms [4, 11-15] have been introduced which are applicable in software. Nowadays, the trend has shifted to the use of standalone video surveillance systems on embedded systems instead of purely on software.

Research and development on software as well as on embedded system based object tracking have been covered by many researchers [4, 11-24]. This chapter highlights some of important research contributions from software and embedded implementations perspectives. First, object tracking algorithms using software and general purpose computers for implementation have been discussed. Then algorithms that are implemented on FPGA as embedded system are exhibited and explained. Algorithms of object tracking have been classified into four categories based on previous research. Related literature for each class is discussed thoroughly. Section 2.3 highlights research about different tracking algorithms that were implemented using different techniques on FPGA.

## **2.2 Object Detection and Object Tracking Algorithms**

Object tracking is an important application used in video surveillance and image processing systems. To track an object of interest through sequence of frames, it is needed to detect that object within each frame. Several object detection algorithms have been developed and research had contributed a lot in this domain. In common the most known algorithms are categorized under point detectors, background subtraction, segmentation and supervised learning algorithms; according to survey study produced in [23]. A number of algorithms and approaches have been designed and developed under each category. Some of them have been highlighted with their scope and objectives; as described in subsections 2.2.1–2.2.4.

### **2.2.1 Point Detectors**

Point detectors are algorithms that search for interest points through the image to use them later for motion, tracking and stereo applications. The interest points are detected according to their expressive texture in their respective localities. Some researchers have introduced the point detectors algorithm using different types of detectors, such as Moravec's detector [25], Harris detector [26], Affine Invariant Point Detector [27] and Scale Invariant Feature Transform (SIFT) [28]. The best algorithm is the one which can detect interest points that are unchanging to illumination, resolution or camera location changes. In comparison with other detectors, SIFT detector generates a greatest amount of interest points under different transformations, since it accumulates points of interest at different scales and resolutions. Among other detectors, it has achieved the highest performance of interest points, and the most robust results to image deformations [29]. This is because SIFT accumulates the interest points through different resolutions and scales of the scene. However, it is also more complex than other methods. The algorithm steps required to regenerate the image with different scales using Gaussian filters and later the histogram information is needed to allocate the obtained points from different scale images in the correct location. To use such an algorithm for real time object tracking, the consumed execution time must be considered.

Table 2.1: Summary of point detectors algorithms

Algorithm	Used in	Representative work	Comment
Point detectors	Motion, tracking and stereo applications.	<ul style="list-style-type: none"> <li>• Moravec's detector (Moravec, 1979)</li> <li>• Harris detector (Harris and Stephens, 1988)</li> <li>• Affine Invariant Point Detector (Lowe, 2004)</li> <li>• Scale Invariant Feature Transform (SIFT) (Mikolajczyk and Schmid, 2002)</li> </ul>	SIFT algorithm achieves the highest performance under different transformations and obtains the most robust results among other point detectors, however it consumes high computation time due to its steps.

### 2.2.2 Background Subtraction

Background Subtraction algorithms are simple in implementation but at the same time their results could be affected by noise, illumination changes and object occlusions. It is used to detect temporal changes on the scene through the time-information via stream of frames. Usually the background subtraction is not used in its simple form. In order to take the benefit of simplicity of the background subtraction and to avoid the affected results, other techniques have been combined with it to get more accurate results. The research started on this algorithm since late 70s [30] until now, but it only became popular when [20] introduced their work in 1997. They have proposed to model each pixel of the background with a single 3D Gaussian. Model parameters, the mean and the covariance values, are obtained from the gradual changes on the colors through consecutive frames. Next, these values are used to build the background model that is used to determine the foreground by comparing the pixels of each frame's with its background model values.

However, the model uses a Single Gaussian Mixture Model (SGM), which still has drawbacks especially for outdoor uses. Repetitive object motions, shadows and reflectance can cause multiple colors for one location. Thus, other researchers have introduced background models such as Wall flower [31], Mixture of Gaussians



(MOG) or Gaussians Mixture Model (GMM), Eigen background [32] and Dynamic texture background [33].

In [34], an adaptive background subtraction algorithm, which is an improvement of Mixture of Gaussians model, was developed. In contrast with Mixture of Gaussians, the numbers of components are not fixed. For each pixel the number of components is adapted automatically according to the current scene. Thus a training set is used to estimate the background model by estimating the density function. The training set can adapt changing of illumination that happens gradually, sudden changes or even when a new object enter or leave the scene. By discarding old samples and adding new ones to the training set every chosen period of time, the changes are adapted. The developed model by [34] can work faster than the normal GMM and achieve slightly improved results. However, the computational effort of the improved GMM is still large to be applied in real-time applications. Furthermore it is not suitable for limited resource systems such as embedded systems. Tracking quality is slightly improved than GMM but basically GMM gives low quality for tracking rigid objects [4]. The execution time of the adaptive background subtraction is increased according to the complexity of the scene as when new objects enter or leave the frame.

When the density function is highly complex, the model can't be built by the parametric model approach. Thus a non-parameter background modeling can be used instead. In [35], non-parametric background modeling using adaptive kernel density estimation for motion based modeling is introduced. To adapt changing characteristics during the running time, optical flow is utilized as a feature to detect changes. The algorithm is better than GMM in adapting changes and working under challenge conditions for online video tracking, however computational time is still not reduced.

A more simple and common background subtraction method is Image difference. The target frame is subtracted from the near previous frame, while the moving pixel is detected if the difference exceeded a given threshold. Such an algorithm is efficient and fast, however it suffers from two common drawbacks: foreground aperture and ghosting. Ghosting is caused by objects move so fast so that the foreground of the old position is caught as background for the current frame. On the other hand, if the

object moves too slowly, aperture of foreground will result [12]. In order to overcome ghosting and aperture problems, [36] have used double differences of  $t-1$  and  $t-2$  and made logical AND, while the threshold is determined at time  $t$  and  $t-1$ . Double differences methods work well with small depth field images, while producing less performance with the large field depths. However, Image difference algorithms can't be run automatically since the threshold parameters are set manually.

In [12], inter frame differencing algorithm with improvements on calculating the threshold to solve the known image difference's problems is presented. The algorithm was developed especially for detecting objects in outdoor night environment. Obtained detected objects by it are more accurate compared with those obtained by other algorithms applied for the same scene. However, the algorithm still cannot work automatically since a lot of required main parameters are set manually or by using histograms. Additionally, it is not guaranteed to work effectively under different environment such as day time or indoor conditions.

In [4], a method called Adaptive Hybrid Difference (AHD) that works in real time was developed which is an improvement over the previous methods. The GMM method is better in the quality of the image built by the tracking while the image difference method is better in terms of execution time. AHD method is based on the process of image difference method. At the same time, an adaptive threshold technique has been designed to calculate the threshold automatically rather than specifying thresholds manually. This has made the algorithm robust for real time and can be run automatically. Using hybrid difference strategy in the algorithm improves the quality of the obtained tracking image. The authors of [4] carried out a test for GMM and AHD methods on tracking object images of vehicles in the highway road. The results indicated that AHD runs faster than GMM and obtains a higher quality tracking image.

Table 2.2: Summary of background subtraction algorithms

Algorithm	Used in	Representative work	Comment
Background subtraction.	Detecting objects.	Single Gaussian Mixture Model (SGM) (Wren et al. 1997) [20].	SGM is affected by repetitive object motions, shadows and reflectance.
	Detecting objects.	Gaussians Mixture Model (GMM).	GMM computational steps are complex while it achieves low quality for rigid objects.
	Detecting objects.	Adaptive background subtraction (Zivkovic 2004) [34].	Works faster than GMM and achieves slightly improved results. However, the computational effort is large to be applied in real-time applications.
	Detecting objects	Non-parametric background modeling (Mittal et al. 2004) [35]	Adapts changes and works under challenging conditions better than GMM, however the computational time is still high
	Detecting moving objects.	Image difference.	Consumes low computational effort, efficient and fast, however it suffers from foreground aperture and ghosting.
	Detecting objects.	Double differences method (Cucchiara et al. 1999) [36].	<ul style="list-style-type: none"> <li>• Overcomes ghosting and aperture.</li> <li>• Works well with small depth field images but not with the large field depths.</li> <li>• Cannot be run automatically.</li> </ul>
	Detecting objects in outdoor night places.	Inter frame differencing (Huang et al. 2008) [12].	<ul style="list-style-type: none"> <li>• Overcomes image difference's problems. Achieves more accurate results than others.</li> <li>• Cannot be run automatically.</li> <li>• Not guaranteed to work effectively under different conditions.</li> </ul>
	Tracking vehicles in the highways	Adaptive Hybrid Difference (AHD) (Shi et al. 2007) [4].	Works automatically at real time conditions. Runs faster than GMM and obtains a higher quality than image differences algorithms.

### 2.2.3 Segmentation

The segmentation process is to partition the image into perceptually similar regions. Image segmentation algorithms that are relevant to object tracking have been widely proposed using different techniques. There is no common standard classification of object segmentation algorithms followed by segmentation researchers. One classification based on the concept of space and time [15], defined three groups, temporal, spatial and spatio-temporal.

For spatio-temporal segmentation algorithms, in 2004 [37] have worked in dividing the frame image into perceptual regions using spatial and temporal edge information. This algorithm requires complex computation for finding the required edge-motion parameters and edge detection.

Algorithms based on space or spatial segmentation algorithms are also categorized into, pixel classification, edge based, region based and model based. Pixel classification methods use threshold to group the related pixels into one segment as outlined by [38]. The work of [39] has presented edge detection methods, such as sobel filtering for segmentation, by identifying the regions surrounded by the edges. Region growing algorithm as presented in [39] is an example of region based segmentation and it is popular among segmentation algorithms. It divides the image into cells of network and starts searching for coherent pixels to combine them together, then the segmented pixels within each cell are grouped with other coherent segments from neighboring cells.

Segmentations based on region growing are working well for simple objects, but for complex objects, the output segments after the segmentation process still need to go through a grouping operation to connect segmented parts belonging to the same object.

In [14], a grouping algorithm for tracking complex objects through a video stream has been proposed. Image segmentation stage is done first by applying the region growing algorithm on each frame. Using the region growing algorithm achieves robustness among a noise that could occur when motion dependent algorithms are used instead. In the next stage, some features such as: size, position and color are

extracted for each segment. These features are used to match segments in the current frame with the corresponding segments from previous frames. In order to match the corresponding segments via frames, the Manhattan-distance method is applied whereas segments are matched with those having the minimum Manhattan distance [13, 22]. Extracting a Motion vector for each segment is an important step that the grouping algorithm depends on. The motion vector shows the pattern that has been followed by a segment during the time. It is determined by the distance between the segment in the current frame and its' matched segments in the two successive frames. The grouping algorithm searches among the frames for the segments that have the same motion vector and have an overlap of their bounding boxes. At the end, the grouped segments are judged to be belonging to the same moving object.

Later on, researchers of [14] have extended their presented algorithms that have been implemented on PC, by implementing the algorithms on FGPA in [22], as explained in section 2.3.2.

Temporal segmentation extracts objects depending on time information. Therefore, frame differences with consecutive frame or with a predefined background are considered according to [15] as temporal segmentation. However in this research the temporal segmentation will be categorized under background subtraction algorithms as defined by subsection 2.2.2, rather than segmentation since it depends basically on subtraction operations between frames. Generally, the segmentation follows parallel joining operations or scan searching for kind of similarity between neighbor pixels that are not shared the same way in background subtraction algorithms.

The high parallel computation calculations required by the segmentation has made it strongly attractive for VLSI implementations. However, the degree of complexity, and hence the hardware cost, must be taken into account for implementation. In addition to parallel operations, segmentation can detect moving as well as still objects in contrast with background subtraction which only catches the moving objects. Additional operations are required in segmentation to distinguish moving from still objects in subsequent steps of implementation.

Table 2.3: Summary of segmentation algorithms

Algorithm	Used in	Representative work	Comment
Spatio-temporal segmentation.	Detecting objects.	Spatio-temporal segmentation (Papadimitr et al. 2004) [37].	Requires complex computations.
Spatio segmentation.	Detecting objects.	Pixel classification (Jahne 2005) [38].	The threshold used in the algorithm depends strongly on image contrast and sometimes small noise-like pixel groups concedered as objects [15]
Spatio segmentation.	Detecting objects.	Edge detection using sobel filtering (Braunal et al. 2001) [39].	Edge based algorithms are very sinsitive to the noise [15].
Spatio segmentation.	Detecting and tracking objects.	Region growing (Braunal 2001 et al.) [39].	Works well for simple objects, but for complex objects, segments need to go through a grouping operation to connect the related segments.
Spatio segmentation.	Detecting and tracking moving objects.	Grouping method based on feature matching for tracking and recognition of complex objects (Nagaoka et al. 2009) [14].	Effective algorithm however segmentation algorithms consumes high computations since they need additional operations for grouping and for diffrentiation between moving and still objects.

#### 2.2.4 Supervised Learning

One of tracking algorithms was used by some researchers for supervised learning to learn different object views from a set of examples. The latter one generates a function that maps input into desired output. The learning examples consist of pairs of object features and associated object classes which are quantities information supposed to be defined manually. This technique is excluded from our interest for two reasons. First, learning examples quantities information are supposed to be defined

manually, which makes implementing an automatic real time algorithm not possible. Second, complete set of templates need to be stored in the processor that implements the supervised learning, which requires large sized storage [23].

Research covering object tracking via the supervised learning includes different supervised learning types such as, Support Vector Machines [40], Neural Networks [41] and Adaptive Boosting [42].

Table 2.4: Summary of supervised learning algorithms

Algorithm	Used in	Representative work	Comment
Supervised learning.	Detecting objects.	<ul style="list-style-type: none"> <li>• Support Vector Machines (Joachims et al. 1999) [40].</li> <li>• Neural Networks (Rowley et al. 1998) [41].</li> <li>• Adaptive Boosting (Tieu et al. 2004 ) [42].</li> </ul>	Cannot be run automatically since the learning examples information supposed to be defined manually. In addition implementing the algorithm requires a large sized storage.

### 2.3 Implementation of Object Tracking Algorithms on FPGA

Using independent specific task systems for surveillance systems instead of general purpose computers in order to save power and resources is highly demanded. Thus, many researchers in both embedded systems and surveillance applications area were motivated to design and implement object tracking and detection on embedded systems such as FPGA. A number of implementation strategies have been proposed for object tracking using FPGA; some of them are highlighted in subsections 2.3.1-2.3.5. Research works are listed under subsections according to the used tracking algorithm.

#### 2.3.1 Image Segmentation based on Color Threshold

One of the research works on object tracking implementation on FPGA is presented in [11]. This research introduced a design and implementation of real time system for remote object tracking algorithm on FPGA. Image segmentation algorithm is used to detect moving objects by applying a color threshold on each pixel. Thus, the tracking depends on the color of objects.

The block diagram for the complete system of [11] is shown in Figure 2.1.

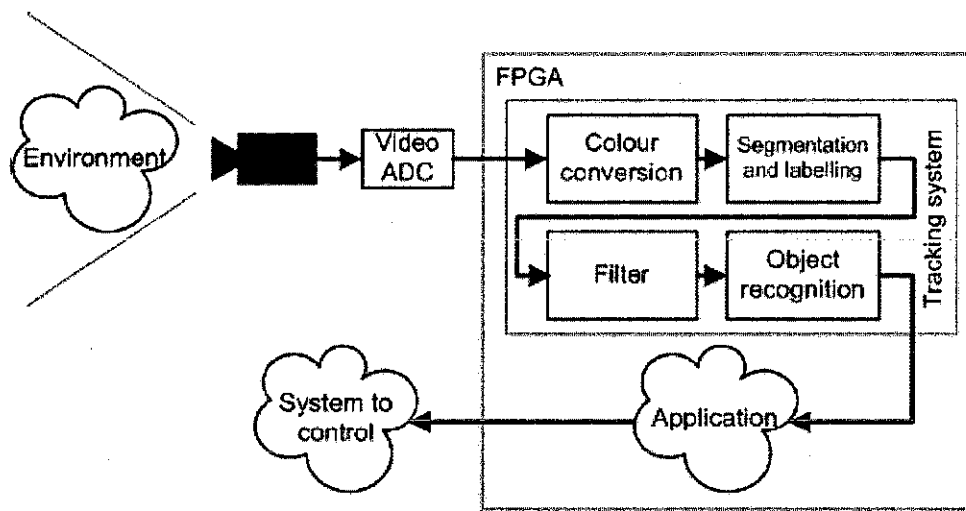


Figure 2.1: Block diagram of object tracking system using color threshold segmentation [11].

As the Figure illustrating, input data from the camera to the FPGA is done in four processes. In the first process, the pixel stream is converted to a modified YUV color space to overcome the RGB intensity interdependence that causes unreliable results in case of any diagonal movement. The second is the segmentation process using color threshold values that categorizes each pixel into a color class. In the next process, a morphological filter is applied on the categorized pixels using windowing scan to remove any mislabeled pixel that is not part of the object. The last step is to construct bounded boxes around the detected objects and extract some useful object characteristics such as position, size and area.

In addition, some optimization techniques have been used. Furthermore, solutions to overcome the use of floating point, division and multiplication operations during the color conversion and threshold comparison stages were presented.



Special application is developed to simulate the system. The application is a simple arcade game in which user arms' movements are used as tracking objects. Each user holds a fluorescent colored marker that is used as the input data for the color segmentation.

The system works with 27 MHz frequency on image size of  $768 \times 288$  pixels while using (10 %) of the logic utilities. It has achieved throughput of 25 fps using Xilinx XC2S200 device.

Table 2.5: Summary of image segmentation based on color threshold on FPGA.

Algorithm	Used in	Atchived Results	Comments
Image Segmentation based on Color Threshold (Johnston et al. 2005) [11].	Detecting and tracking moving objects according to specific input colors.	System Frequency: 27 MHz Frame Size: $768 \times 288$ pixels Logic Elements: 798 LEs Throughput: 25 fps On chip memory used: 114,688 bits	<ul style="list-style-type: none"> <li>• Does not meet the speed of real time system speed (30 fps).</li> <li>• Consumes high resources of memory on chip space.</li> </ul>

### 2.3.2 Image Segmentation and Pattern Matching

In the same area with a different technique, [21, 22, 43] have proposed a novel algorithm for object tracking based on FPGA. The algorithm that has been used is based on image segmentation and pattern matching. Also the FPGA/ASIC implementation architecture was presented. By using the segmentation, all objects in the image whether still or moving are detectable. The used segmentation algorithm has been described previously in section 2.2.3 by [14] which was developed by some of those researchers. After detecting objects of each frame, specific features -such as position, size, color and area- for objects through all frames are extracted. Then, features from the current frame are used to obtain the pattern matching with corresponding objects from the previous frame. The proposed architecture for implementing the algorithm on FPGA/ASIC is shown in Figure 2.2.

As shown in Figure 2.2, the hardware architecture consists of four main blocks. The blocks, ordered according to their functionality are: image segmentation cell-network, feature extraction, pattern matching and estimated position calculation block. The image segmentation cell-network block is responsible for extracting all objects in the frame. The feature extraction block extracts object features for each segmented object that has been detected. Next, the pattern matching block searches for the most similar objects in subsequent frames using extracted features information. The estimated position calculation block estimates positions of objects in the next frame by performing specific calculations.

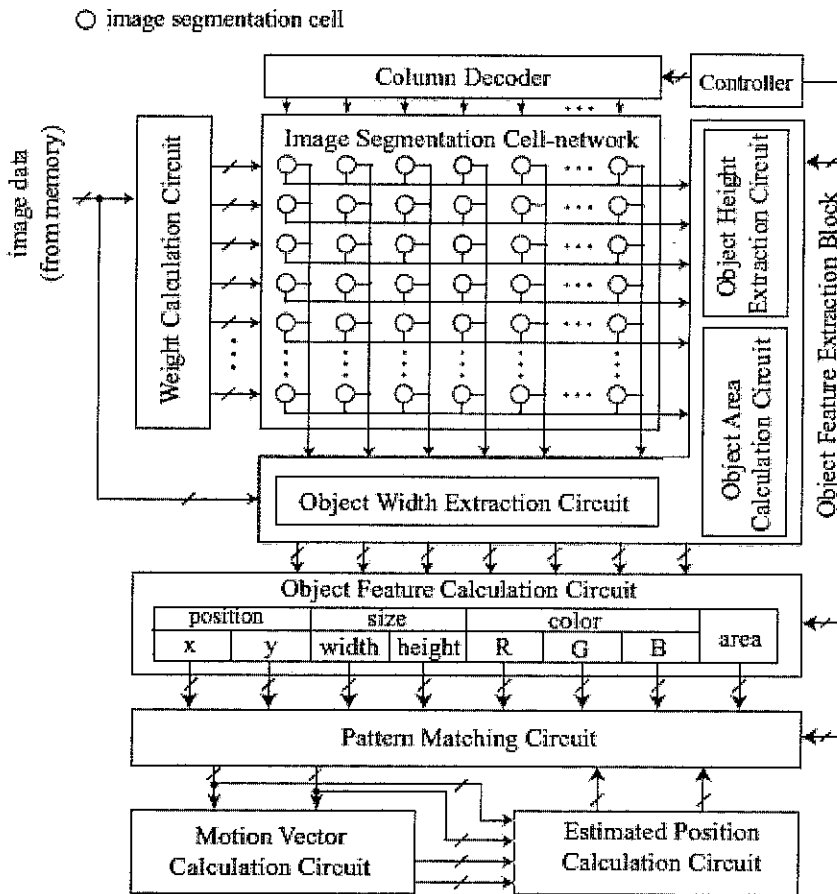


Figure 2.2: Block diagram of FPGA/ASIC implementation architecture for object tracking system using segmentation and pattern matching [22].

The image segmentation algorithm used in [22] is implemented by region growing algorithm. Since the region growing algorithm follows a sequential method, a pipeline

processing is used to improve the throughput. The data flow of the pipeline processing is illustrated in Figure 2.3. The pipelining saves the running time by executing the pattern matching process for one object while starting the segmenting for the next object at the same time.

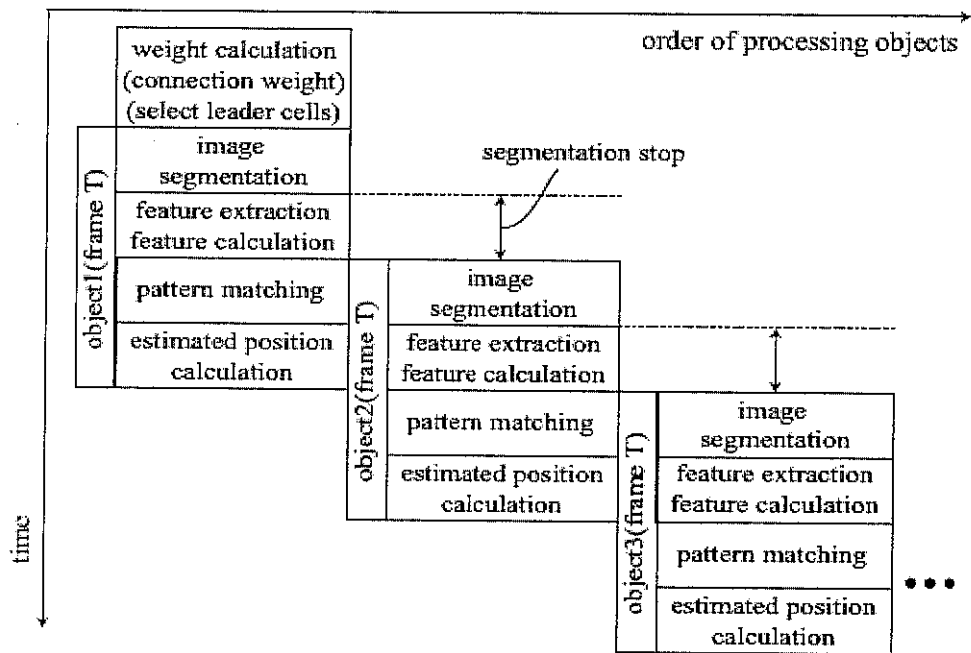


Figure 2.3: Data flow of the pipeline processing architecture of object tracking system using segmentation and pattern matching [22].

The system works with 20 MHz frequency on image size of 80×60 pixels while using 56% of the logic utility of Altera Startix EP1S60 device. For one frame, it requires 13,745 cycles + 10 cycles × N to execute, where N is the number of detected objects [22, 43].

Table 2.6: Summary of image segmentation and pattern matching algorithm on FPGA

Algorithm	Used in	Atchived Results	Comments
Image segmentation and pattern matching (Yamaoka, et al. 2006) [22].	Detecting and tracking moving objects.	System Frequency: 20 MHz Frame Size: $80 \times 60$ pixels Logic Elements: 31,987 LEs Throughput: $13,745 + 10 \times N^2$ (N is number of detected objects) On chip memory used: 144,256 bits	<ul style="list-style-type: none"> <li>• Achieves high speed, since it can detect up to 220 objects at real time.</li> <li>• Consumes high device resources.</li> </ul>

### 2.3.3 Image Gravity Center and Matched Filter for Planer Motion Tracking

A planar motion tracking is one kind of tracking applications types; authors of [18] have presented the implementation of planar motion tracking algorithm on a CMOS image detector and an FPGA. The proposed system has been developed to track the planer motion within 1MHz sampling rate. The tracking planar motion was detected using two vision algorithms; Image gravity center and matched filter.

In [17], researchers have introduced a solution for the same problem (planar motion tracking) using vision chip instead of FPGA. The proposed vision chip consists of image elements containing a photo detector and a computer with a memory. The image element is then connected with its neighboring image elements from the four sides using links. Thus, the chip works as a full parallel computer that is distributed among an array of image detectors. Even though the local operations are executed in a very fast speed using the parallel computing, implementing global operations -such as Fourier transform and Hough transform- in the parallel computing system are difficult to be performed. Another disadvantage of implementing the tracking system on a vision chip is the need to mix the analog circuit for capturing the images and the digital circuit for the computation. When any analog and digital mixing is done on LSI, interference between analog and digital circuits could occur.

In [18], the planar motion tracking implementation on FPGAs was compared with the implementation using the vision chip by [17]. By implementing the planar motion

tracking on a CMOS camera and a FPGA, the digital and analog circuits are separated. From another side, both local and global operations are realizable within the FPGA scope. The developed architecture on the FPGA of the system is shown in Figure 2.4. Images are captured by the CMOS and sent to the Channel link transceiver block on the FPGA board via low-voltage differential signaling (LVDS) which is a high speed interface connection. In the Main FPGA block, the images are processed using inner logical circuits that implement at real time and use the DDR-SDRAMs memory if necessarily. A PCI (Peripheral Component Interconnect) bridge is used to provide the communication between the FPGA and the PCI bus that is connected to a PC and to control software.

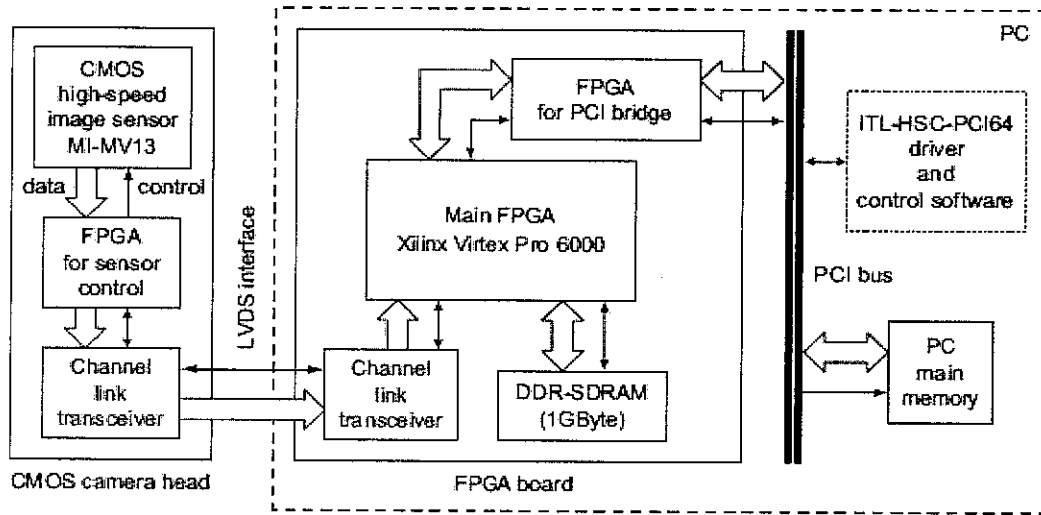


Figure 2.4: The Architecture block diagram for implementation of planar motion tracking on CMOS image detector and FPGA [18].

The main FPGA block contains logic circuits to compute the image gravity center algorithm. To achieve an implementation at the real time, numbers of logic circuits are chosen to match the amount of data raw (pixels that have been sent by the camera each clock cycle). The matched filter algorithm can detect the translation between two images robustly, even with the background and illumination changes. It consists of two global operations: 2DFFT and 2D-IFFT. The Fourier transform implementation is more complex and it needs more time to be executed. By using the benefits of parallelism and pipelining of the FPGA, the computation time has been reduced to meet the real time condition.

The implemented algorithm has achieved 1545.44 fps of throughput for frame of  $64 \times 64$  pixels, using 44% of logic utility of Xilinx Vertex Pro 6000 device, with a speed of 66 MHz.

Table 2.7: Summary of matched filter algorithm on FPGA

Algorithm	Used in	Atchived Results	Comments
Matched Filter (Kazuhiro et al. 2006) [18].	Motion Tracking.	System Frequency: 66 MHz Frame Size: $64 \times 64$ pixel Logic Elements: 30,140 LEs Throughput: 1545.44 fps	<ul style="list-style-type: none"> <li>• Achieves high speed.</li> <li>• To overcome the complex computations needs, heavy device resources have been consumed.</li> </ul>

#### 2.3.4 Color Detection and Binarization Using HW/SW co-design

Other researchers have presented the object tracking algorithms based FPGA using a mixed hardware/software solution [24]. The systems presented previously (subsections 2.3.1-2.3.3) are stand alone and provide a real time execution while accommodating complex algorithms. Instead of using only the hardware part of the FPGA, a built-in coprocessor have been built on the FPGA itself. In [24], researchers have designed a hardware/software system to implement skin color detection and tracking system. The system detects the skin color based on a specific color, and then it controls the motion of the camera to track the suspected object. Figure 2.5 shows the hardware and software architecture design and the distribution of functions among the hardware and the software sides.

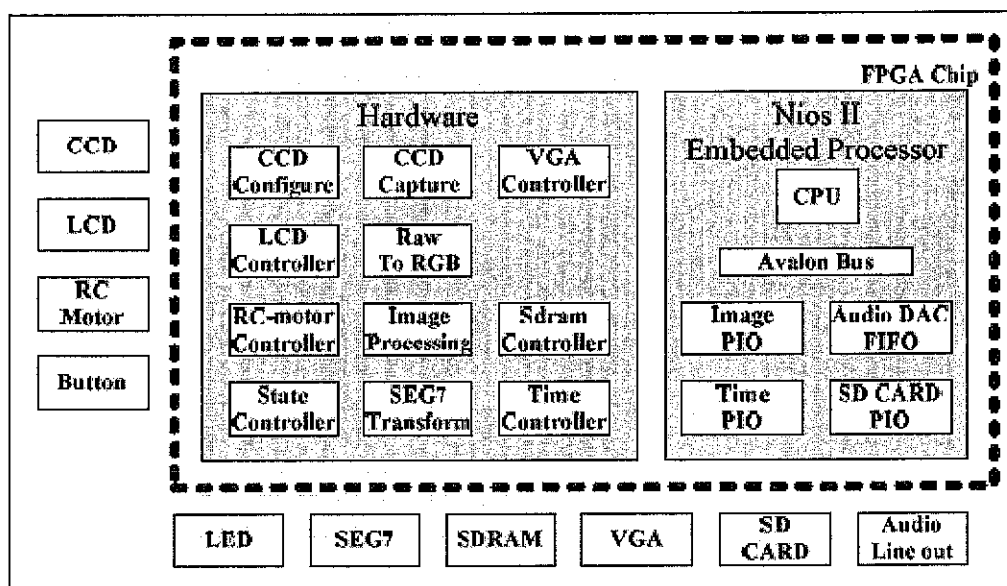


Figure 2.5: Hardware and software integrated architecture of skin color detection system proposed [24].

Image capture, Image display, memory access, Image processing and motor control blocks are all implemented on the hardware part of the FPGA. Image processing unit is responsible for transferring the image color from RGB into HSV space to enhance the image brightness. Another function is to find the histogram equalization of the HSV image that includes histogram statistics and probability distribution processes. Color detection and binarization stage compares the HSV data that fall into a range of threshold color value. To reduce the noise of the image, the image dilation technique has been used. The coordinates of the object are then specified. In the software system part, the transferring of captured images into a BMP format and storing them in the SD card are managed. Also when a suspected skin color is detected, an alarm speaker on the board is turned on and a sound file from the SD card is read. All software functions are processed through the Nios II processor that is embedded on the FPGA chip as shown in Figure 2.6.

The hardware/software co-design system achieved realistic results since it needs 16.8 ms of time to catch the suspected object of a frame till it gets its location and triggers the alarm. The implementation is done on Altera Cyclone II 2C35 device with

a frame size  $800 \times 525$  pixels and 25 MHz clock rate using 19,085 LE (54%) –for the HW part -. The system have achieved throughput of 37.3 fps.

Table 2.8: Summary of color detection and binarization using HW/SW co-design

Algorithm	Used in	Atchived Results	Comments
Color detection and binarization using HW/SW co-design (Yuan-Pao, et al. 2010) [24].	Detecting object according to a specific skin color and then tracking that object.	System Frequency 25MHz Frame Size: $800 \times 525$ pixel Logic Elements: 19,085 LEs Throughput: 37.3 fps	<ul style="list-style-type: none"> <li>• The system achieves real time speed with realistic results.</li> <li>• The software part is embedded on the FPGA chip itself while the 19,085 LE is used for the HW part only, which means additional device resources were used. However, for comparison, detection and tracking functions which are implemented on the HW part only are considered.</li> <li>• High logic resources were used.</li> </ul>

### 2.3.5 Principal Component Analysis based Planar Motion Tracking

Recently, researchers of [10] presented an architecture and implementation of moving object tracking algorithm on FPGA using Principal Component Analysis (PCA). The PCA is a method that basically works by reducing the redundant information and keeping the fundamental ones. The parallelization concept has been used to implement the deferent stages of the PCA, such as computing the correlation matrix, matrix digitalization using the Jacobi method and subspace projection of images. The main contribution of the work in [10] is to implement the complete PCA algorithm on FPGA. Other PCA based FPGA researchers have divided the PCA's implementation between FPGA and general purpose computer or microprocessor. The need to separate the PCA's implementation is because of its complex mathematical operations and the large amount of data needed during the execution time. The architecture Block diagram of the PCA implementation on FPGA is shown in Figure 2.6.



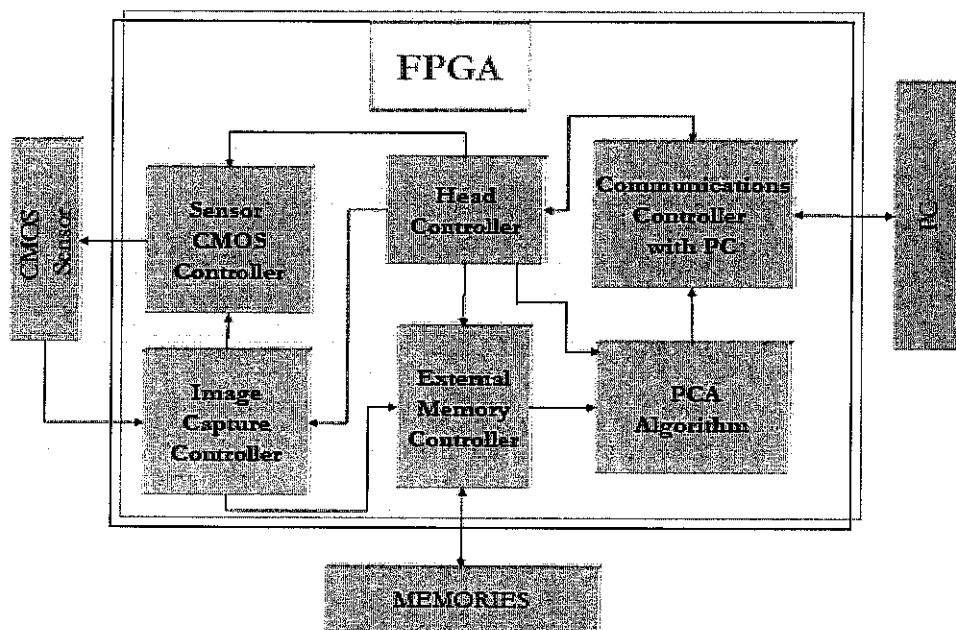


Figure 2.6: Block diagram for the internal architecture of the tracking system design on FPGA for PCA algorithm [10].

The architecture consists of many blocks according to the system functionality. The CMOS sensor controller block is used to implement the desired configuration of the CMOS sensor, such as setting the exposure time. Another Block is the Image Capture Controller that receives the frames from the camera and at the same time it can use by the user to specify the sensor's area of interest within the frame. Also the Image Capture Controller block sends the captured frames to the External Memory Controller block, where they are stored in the memory. To test and watch results, the Communications Controller with PC block controls the communication between the FPGA and the PC, where the results can be seen. Finally, the Head Controller block is playing the main block role that connects other blocks and synchronizes the entire system to make it work correctly and at the maximum speed [10].

In [10], the implemented system has achieved a throughput of 121 fps with 100 MHz frequency for  $256 \times 256$  pixels' frame size. The synthesized logic utility was 86% logic resources of the Xilinx XC2VP7 device.

Table 2.9: Summary of principal component analysis on FPGA

Algorithm	Used in	Atchived Results	Comments
Principal component analysis (Bravo et al. 2010) [10].	Motion tracking.	System Frequency: 20 MHz Frame Size: 256×256 pixel Logic Elements: 8,450 LEs Throughput: 121 fps Memory on chip used: 737,280 bits	<ul style="list-style-type: none"> <li>• Achieves real time speed.</li> <li>• Algorithm's steps are complex.</li> <li>• Memory on chip utilization is high.</li> </ul>

### 2.3.6 Summary of Object Tracking Algorithms Implemented on FPGA

Implementation work of different object tracking algorithms with different designs on FPGA have been discussed in subsections 2.3.1-2.3.5. Table 2.10 summarizes the discussed implementation work in terms of the usage and achieved results. As has been mentioned in section 1.2, object tracking is the algorithm used to detect objects through sequence of frames according to a specific feature. The feature of detection could be motion, color, shape or other. Thus, any motion, color, or shape tracking is indeed an object tracking while any object tracking is not necessarily a motion tracking. In this research, motion is the feature used for object tracking. The work listed in Table 2.10, with the exception of the one using HW/SW co-design algorithm, were applied to detect and track moving objects. For HW/SW co-design algorithm, the detection and tracking is based on a specific skin color rather than motion. So the comparison between the works of this research that investigates implementing optimal design of moving tracking algorithm on FPGA is going to be done with the works presented in Table 2.10, as shown later in section 5.8.

Some work have implemented more steps or used more complex algorithms to define motion detection and tracking. As an example, the work presented in subsection 2.3.2, segmentation and pattern matching algorithm, extracts features from each detected object like area, location and color. As was mentioned in section 2.2.3, the work in[14], and in section 2.3.2, the extracted features are used for two reasons, first to group segments that belong to the same object and second to determine the moving from the still objects. In contrast to subtraction algorithms, segmentation

algorithms detect all objects in the scene whether they are moving or still. There for, additional steps are required after detection to differentiate the still from the moving objects. For subtraction algorithms, moving objects are determined with less computational effort. After defining the background model, by subtracting frames, the moving detection results are ready. The comparison at the end of this work is done between different algorithms implemented with different designs on FPGA while all of them are aiming to achieve moving object tracking system.

Table 2.10: Summary of object tracking algorithms implemented on FPGA

Algorithm	Used in	LEs Used	Internal Memory Bits Used	Frame Size (pixel)	Frame Rate (fps)	System Speed (MHz)
Segmentation based on Color Threshold	Detecting and tracking moving objects according to specific input colors.	798	114,688	$768 \times 288$	25.00	27
Segmentation and Pattern Matching	Detecting and tracking moving objects.	31,987	144,256	$80 \times 60$	40.18 (for 220 objects)	20
Matched Filter	Motion Tracking.	30,140	124/144 (86%) Block RAMs	$64 \times 64$	1545.44	66
HW/SW co-design	Detecting and tracking object according to a specific skin color.	19,085	N/A	$640 \times 480$	37.30	25
Principal Component Analysis	Motion tracking.	8,450	737,280	$256 \times 256$	121.00	100

## 2.4 Chapter Summary

Different tracking algorithms are summarized during this chapter. Some of the algorithms are implemented on software like region growing algorithm, background subtraction using Gaussian Mixture Model, temporal average model or Non-

parameter, background image difference and Adaptive Hybrid Difference. Latest research work is mainly focused on implementation of object tracking systems on FPGA. According to specifications of this research, the interest is going to be on background subtraction and segmentation algorithms. In order to implement an object tracking algorithm on an embedded system at real time with limited resources and small power consumption, some characteristics have to be met. The algorithm has to be run automatically without manual inputs to meet the automatic system requirements. Another concern is the degree of complexity in order to be able to execute the algorithm using hardware languages. On the other hand, the execution time should meet the real time execution conditions. The complexity of the algorithm and the amount of storage needed must fit the available limited resources for the embedded systems. Most of the tracking algorithms that meet these requirements are found within background subtraction and segmentation algorithms. A number of research works have presented design and architecture of proposed tracking systems on FPGA device during sections 2.3.1-2.3.5. Each work has presented a solution using different tracking algorithm and different implementation techniques. The logic element utilization consumed by other researchers were 798 LEs, 6,320 LEs, 8,450 LEs, 19,085 LEs, 30,140 LEs and 31,987 LEs for the work presented in sections 2.3.1-2.3.5, respectively. The utilizations of logic resources in the most of the mentioned cases are high. The goal of this research is to achieve a design for tracking system on FPGA that work at real time with optimized device resources of logic elements and internal memory.

## CHAPTER 3

### SOFTWARE SYSTEM DESIGN AND IMPLEMENTATION STRATEGY

#### 3.1 Chapter Overview

This chapter highlights the design and development of the software for implementation of the Adaptive Hybrid Difference (AHD) algorithm. AHD has been selected from a number of tracking algorithms which have been discussed in Chapter 2. AHD algorithm has been used for tracking moving objects on FPGA for many reasons that will be thoroughly discussed. Thus, the aim of software design and its implementation is to verify the AHD algorithm through software implementation tools before its implementation on the hardware. An analytical study based on the software implementation will be presented. Additionally, the results and evaluations obtained by software implementation have been used in comparison of performance between AHD based software and FPGA based implementations. Moreover, the software system has been built to check the feasibility of computerizing the tracking algorithm on software and hardware.

#### 3.2 Software Design

The proposed software design is based on Adaptive Hybrid Difference (AHD) algorithm. AHD algorithm has been proposed by [4] for vehicle tracking applications using Desktop computers in a sequential manner. In this research AHD algorithm has been extended to make it executable in parallel fashion using FPGA for the purpose of detecting the moving object in such a way to optimize the system performance in real experimental environment. The selected performance parameters in this work include throughput, utilization of block RAMS, and logic elements which are effected

directly, with the systems' frequency, the power dissipation of the design, as explained in section 1.4. Optimizing the design area and cost are also realized as a result of optimizing the synthesized hardware. Therefore, the core objective is to implement the algorithm with the aim of optimizing the values of the specified performance parameters, throughput and resource utilization.

The main steps for the AHD algorithm are illustrated in Figure 3.1 which is composed mainly of two components:

1. Hybrid Difference Strategy.
2. Adaptive Threshold Initialization.

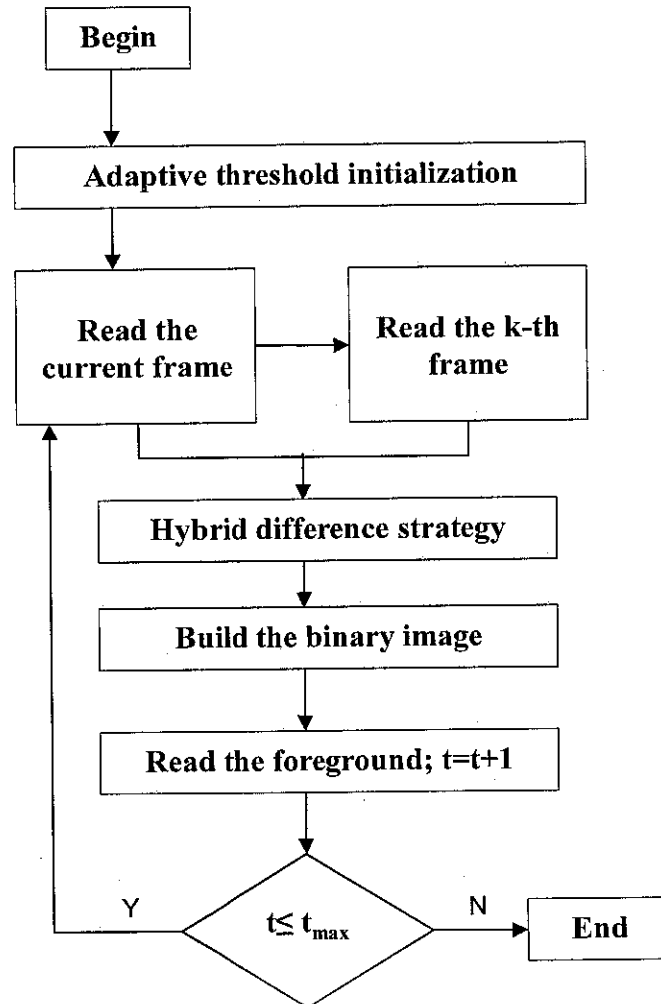


Figure 3.1: Flow diagram of AHD algorithm [4].

The parameter  $t_{\max}$  is the maximum number of the captured frames. AHD algorithm uses background subtraction to track moving objects. Instead of using the classic image difference to subtract the current frame from the previous one or from the ideal background, it uses hybrid difference (*HD*) strategy. The algorithm works on finding the adaptive threshold (*AT*) during a previous tracking time. Thus, at the real time of execution, the result of hybrid difference for a pixel from the current frame is compared with a pre-calculated adaptive threshold value. The comparison leads to decide whether this pixel is moving or still. According to that process, a binary image for the moving objects within the frame is built. The detailed discussions on above mentioned steps are thoroughly highlighted in the following paragraphs and sub sections.

### 3.2.1 Hybrid Difference Strategy

As in most of image processing algorithms, frame representation is based on matrix model that is used in the applied process operations. A single frame ( $n \times m$  pixels) of a video stream can be represented by matrix  $G$  of  $n \times m$  size.  $G^t(i, j)$  is the pixel value of position  $(i, j)$  in the  $t^{th}$  frame, where  $i=1, \dots, n$ ,  $j=1, \dots, m$  and  $t=0, 1, 2, \dots$

The pixel difference is calculated by the D4 distance equation

$$D'_k(i, j) = |G^t(i, j) - G^{t-k}(i, j)| \quad (3.1)$$

where  $D'_k(i, j)$  is the absolute value of the pixel difference between  $t^{th}$  and  $(t-k)^{th}$  frames. Where  $k$  is called the near  $k$  frame and it is an improved technique that is used instead of  $(t-1)^{th}$  frame.

The parameter  $k$  specifies the quality of tracking. If  $k$  is too small, the difference value is also reduced and the result is similar to the one of Classical image difference method. If  $k$  becomes large, the tracking sensitivity will be high even it can mistake some background as foreground. Therefore, the best setting of  $k$  can be reached by testing video and trying different values of  $k$  until the clearest and best binary image is

obtained. The  $k$  value depends on some factors relating to the nature of the application. Some of the factors are the speed of the tracked moving objects, the environment of the application whether in day or night; indoor or outdoor, the distance between the camera and the objects, and the zoom level. All of these factors govern the selected  $k$  values. In this work,  $k$  value was set to 6 for the chosen test environment, as has justified in Appendix B.

### 3.2.2 Adaptive Threshold Initialization

The tracking result depends on the threshold ( $T$ ). When  $T$  is small, the background pixels could be easily detected as foreground. If  $T$  is too large, the tracked foreground may be unclear.  $T$  is usually set manually by experience, but in AHD the method should run automatically in the application of real-time tracking video.

$AT$  is the boundary between background and foreground pixels, which is used to optimize the  $HD$  values. In the initialization of AHD algorithm, an  $AT$  is generated by tracking a video without foreground (moving objects). Supposing that the change of the pixel value follows the Gaussian distribution,  $\mu_{ij}$  and  $\sigma_{ij}$  are the mean and the standard deviation of  $D'(i, j)$ , which can be approximated by  $\mu'_{ij}$  and  $\sigma'_{ij}$ .

$$\mu'_{ij} = \frac{1}{N} \sum_{i=1}^N D'(i, j) \quad (3.2)$$

$$\sigma'_{ij} = \sqrt{\frac{1}{N-1} \sum_{i=1}^N (D'(i, j) - \mu'_{ij})^2} \quad (3.3)$$

Where  $N$  is the number of frames used to set the adaptive threshold.  $N$  should be large enough ( $N \geq 30$ ) to make the estimation of  $\mu'_{ij}$  and  $\sigma'_{ij}$  [4]. However, the larger  $N$  is, the more is the time cost of initialization.

Therefore, the  $AT$  can be calculated by

$$T^{ij} = [\mu'_{ij} - \sigma'_{ij}, \mu'_{ij} + \sigma'_{ij}]. \quad (3.4)$$



Therefore, a binary image can be built by

$$B^t(i, j) = \begin{cases} 1 & \text{if } (\bigvee_{s=1}^k D'_s(i, j) \notin T^{\theta}) \\ 0 & \text{if } (\bigvee_{s=1}^k D'_s(i, j) \in T^{\theta}) \end{cases} \quad (3.5)$$

where  $i = 1, \dots, n$ ;  $j = 1, \dots, m$ ;  $t = 1, 2, \dots, p$

$p$  is a natural number and denotes the total number of frames

The  $AT$  initialization is the first step of AHD algorithm, it uses  $N$  frames to generate  $\mu'_{ij}$ ,  $\sigma'_{ij}$  and  $T^{\theta}$ . The hybrid difference strategy and the building of the binary image are run in the next level at the real time of tracking.

### 3.3 Software Selection

Verifying the algorithm using software is required before its implementation on the hardware. Usually, MATLAB is the commonly used software tool in image processing, testing and simulation. Analyzing the algorithm has led to the suitable verification tool.

According to the AHD algorithm, the adaptive threshold needs the mean and the standard deviation to be calculated for all pixels through 30 frames. This step is performed at the initial time and the threshold values are stored in a lookup table.

Equation (3.5) illustrates the way to build the binary image, which will be applied on each pixel of the frame. Hybrid difference values for pixels in the same position through the current frame to the previous  $k$  frames are calculated to decide if the pixel is still or moving one. If  $k = 6$  and  $t$  is the current frame, then  $HD$  values for pixels in frames  $(t-1, t-2, \dots, t-6)$  are required. In the next step, a comparison operation is applied to test if all  $HD$  values are not belonging to the threshold interval specified by equation (3.4), if so the pixel is considered as a moving pixel. If at least one of the  $HD$

values was located within the threshold interval, then the pixel is judged to be a still one.

For software testing, a video with  $120 \times 160$  pixels is fit to reduce the number of processing pixels, at the same time, keeping an acceptable resolution that is needed by the algorithm. With  $120 \times 160$  frame size and with  $k=6$ , the hybrid difference need to be calculated  $120 \times 160 \times 6 = 115,200$  times. The comparison operation is also repeated in worst case for the same number of times  $115,200 \times 2 = 230,400$  times. Thus, to build a binary image for a single frame using software utilizes  $230,400$  operations. Furthermore, additional time is required to retrieve and store the required data.

The execution complexity of initiating the adaptive threshold is  $O(n^2)$ , since the mean value is needed to be used in standard deviation calculations. For building the binary image at tracking time, the consumption is  $O(n)$ . The execution time for the algorithm on MATLAB have taken long time with Intel(R) Core (TM) 2 Quad CPU, 2.66 GH, 3.46 GB of RAM and 210 GB free hard disk space. Therefore, verifications need faster programming tool like C, which will be more efficient to verify the AHD algorithm. Via C programming, performing the same number of operations for one frame has cost a minute and twenty seconds by lab experiments using Intel(R) Core (TM) 2 Quad CPU, 2.66 GH, 0.98 GB of RAM and 78.4 GB free hard disk space.

### 3.4 Software Development

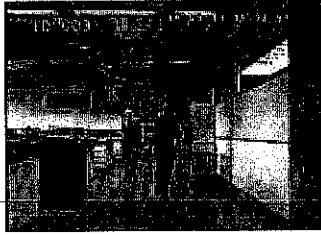
It is known for C programming language and its enhancement copy C++ to have the ability of accessing libraries using linkage specification. Open Source Computer Vision Library (OpenCV) is a library consisting of C functions and some C++ classes that mainly implement image processing and computer vision algorithms [18]. The library is supported by C++. Object tracking algorithms including background subtraction, segmentation and even optical flow are enhanced by OpenCV library. For subtraction, there is an already built-in function that identifies moving objects using accumulative background information. Change detection and threshold are already built-in functions are available to be used for tracking purposes. However, to

implement the AHD algorithm using the adaptive threshold as stated in the algorithm's steps, user-built functions have been developed to be used instead of the already built-in ones. From the other side, the code to be built is simulating the one that have to be implemented on the hardware. This will cause the built software system to be more accurate in verification and comparison. However, some built-in functions of OpenCV have been used to capture frames from the camera or from a video file while other functions are used to represent the frame image in the appropriate format and to extract the pixels. Reading the sample videos from an 'avi' file or from a camera is done via OpenCV functions. Frames are retrieved from the video stream after grabbing them. The captured frames from the video are stored in an object of class 'IplImage' which is used to represent image in OpenCV. For simplicity, since 30 frames are needed from the video in the idle condition –with no moving objects in the scene-, bitmap images which have been retrieved before from the video through MATLAB are loaded.

OpenCV uses IplImage structure to represent images in a matrix or array form. 'IplImage' structure came from Intel Image processing library (IPL) and it stores certain properties of the image and represents the image in multidimensional array according to the number of color planes of the image. Each color plane is represented by a channel, thus, RGB image has three channels where only a single channel is used for gray scale images. To reach any frame quickly, the frames are stored in an array of 'IplImage' objects. To retrieve the pixel's value within a frame, an equation of the pixel's coordinates and the target color channel is used to get the correct index of the pixel in image data attribute. The data attribute of 'IplImage' structure is a pointer for aligned image data. For example, to get the pixel value of coordinate  $(i,j)$  for channel  $k$ , the index of image data is defined by  $i \times \text{step} + j \times \text{nChannels} + k$ , where step is the size of aligned image row in bytes, nChannels is the number of color channels of the image and  $k$  is the channel value (i.e. 0,1 or 2 for Blue, Green and Red for RGB format). The implementation has used a number of FOR loops in order to move between frames and pixels during the execution of the algorithm.

### 3.5 Software Verification and Evaluation

Figure 3.2 and Figure 3.3 show some results of testing AHD in real environment videos. The video used in Figure 3.2 was taken in the lab –indoor place- at a day time under a normal lighting for humans walk in a normal speed. The exhibited frames illustrate AHD results after applying it in chosen frames. Frames were chosen from different times of the video stream that cover objects in far and near locations from the camera.



a. Original image of frame 200.



b. Binary image of frame 200 with  $k=6$ .



c. Original image of frame 223.



d. Binary image of frame 223 with  $k=6$ .



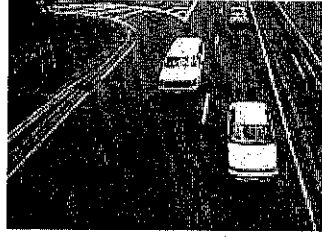
e. Original image of frame 242.



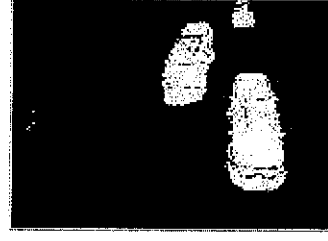
f. Binary image of frame 242 with  $k=6$ .

Figure 3.2: Original and binary images obtained by applying AHD algorithm on different frames for moving objects.

The video in Figure 3.3 is taken for fast moving vehicles in an outdoor environment, at the day time.



a. original image of frame 36.



b. binary image of frame 36 with  
k=2.



c. original image of frame 78.



d. binary images of frame 78 with  
k=1.

Figure 3.3: Original and binary images obtained of applying AHD algorithm on different frames for moving vehicles.

The evaluation of the implemented algorithm was done through calculating the correctness of obtained results. Later on, the correctness evaluation is used to compare the obtained results from the implementation with those which have been obtained by [4].

The correctness of AHD algorithm was calculated using Jaccard Coefficient ( $J$ ) [12].

$$J = TP / (TP + FP + FN) \times 100 \quad (3.6)$$

where  $TP$  (true positives) is the number of correctly detected moving pixels,  $FP$  (false positives) is the number of false detected moving pixels and  $FN$  (false negatives) is the number of missed detected moving pixels. The truth image has been determined manually using advanced image editor program by coloring the objects with white and the background with black. Pixels' colors of the sample image are compared with

the ones of the truth image to decide which counter of  $TP$ ,  $FP$  and  $FN$  has to be incremented. A code has been built in C++ using OpenCV to implement the test and determine  $J$  coefficient after implementing the AHD on a sample frame.

Table 3.1 illustrates the  $J$  coefficient for the binary images obtained in Figure 3.2.

Table 3.1: Jaccard coefficient of binary images obtained in Figure 3.2.

Frame from figure no.	Jaccard coefficient
Figure 3.2.b	88.26%
Figure 3.2.d	74.65%
Figure 3.2.f	67.46%

The Numerical Jaccard ( $NJ$ ) gives the detection accuracy for sequence of frames. From Table 3.1,

$$NJ = (88.26 + 74.26 + 67.46) / 3 = 76.66\% \quad (3.7)$$

Also  $J$  coefficients were calculated for the moving vehicles video that is matching the environment and speed of the video used by [4]. Table 3.2 shows  $J$  value for different frames of moving vehicles' video tested using the same code.

Table 3.2: Jaccard coefficient for binary images obtained from video of moving vehicles.

Frame from figure no.	$J$ Coefficient
Figure 3.3.b	68.20%
Figure 3.3.d	67.16%

The Numeral Jaccard of Table 3.2, is  $NJ = (68.20 + 67.16 / 2) = 67.68\%$ . To compare between our implementation of AHD and the work of [4], we have calculated  $J$  for the obtained binary image of [4] as shown in Figure 3.4.  $J$  value for the binary image of Figure 3.4 is ( $J=81.29\%$ ).

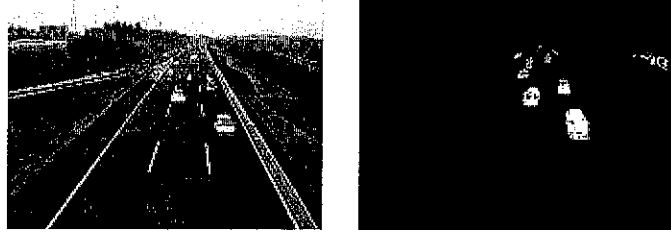


Figure 3.4: Original and binary images obtained by applying AHD algorithm on [4].

The difference in accuracy between AHD applied in [4] and this research for an indoor, not fast moving objects is (4.6%). When comparing the results in [4] and that obtained from outdoor, fast moving objects, the former is better by 13.61%.

The accepted object tracking rate of accuracy in industry is 80%, while the one achieved by verifications was 76.66%, which is close to that rate.

The (4.6%) difference in accuracy between AHD applied in [4] and this research is attributed to some factors. Different environments, outdoor, indoor, the position of the camera from the objects are all factors that influence the implementation results. The binary image quality is improved as the object being far away from the camera. The data for the binary frames published by [4] were not sufficient to calculate  $NJ$ , thus for comparison, only  $J$  factor for one frame has been used.

### 3.6 Chapter Summary

The AHD algorithm has been tested based on software implementation for the same environment that is going to be used in the FPGA implementation. By building a software system that implements the AHD algorithm, an automatic tracking system has been achieved. Choosing C++ with OpenCV as programming tool rather than MATLAB has significantly reduced the execution time from four hours into one minute and twenty seconds for calculating the threshold and building the binary image for one frame. However, the execution speed is still not fast to process at real time condition of 30 fps, since to processing of one frame without calculating the threshold takes 33 seconds. Testing the algorithm on software has achieved results with

76.66% of accuracy, using Numeral Jaccard coefficient. The test took place in indoor environment where the moving objects were people. The accuracy was calculated for the work proposed by [4] which gave 81.29%. However that was for outdoor environments with fast moving vehicles far located from the camera in their video. Thus, the comparison is not justified since the videos have different conditions. Even with less than 80% of accuracy for the AHD implementation on software, the AHD is chosen to be implemented on FPGA on the same testing conditions.

We are going to adhere to the AHD algorithm because it is computerized on HDL and can be run automatically in contrast with other algorithms of background subtraction. Moreover, the simplicity of the algorithm's steps offers implementation's possibilities to optimize the obtained synthesis of the design. With an optimized synthesis design the performance parameters which are specified by the problem statement in section 1.4 and in sections 3.2 and 4.3 are minimized. As mentioned in sections 2.3.1 and 2.3.2 of the literature review, some research work implemented segmentation algorithms on FPGA for tracking moving objects, while less number of previous works used background subtraction algorithms. The goal is to implement a simple, automatic improved background subtraction algorithm in order to achieve less resource utilization, which can be achieved with AHD.



## CHAPTER 4

### HARDWARE ARCHITECTURE AND IMPLEMENTATION

#### 4.1 Chapter Overview

This chapter illustrates the proposed hardware architecture, which has been built to implement the AHD algorithm on the FPGA device. Through an analytical study of the AHD based on the hardware implementation, the optimum solution to achieve real-time tracking system will be investigated. Therefore, the interfacing and the inner hardware architecture design on the FPGA will be demonstrated. Other related important issues that play roles in system design such as memory map, data rate, frequency and achieving a real time system are covered in this chapter. Different techniques that have been investigated to overcome encountered difficulties in meeting the system specifications are covered. Through hardware tools and specifications of the system, the design methodology that was followed for the system to implement an object tracking algorithm on an FPGA will be outlined.

#### 4.2 Hardware Preparation

Following is the detailed description of the hardware used in the implementation of the proposed system. The hardware consists of an FPGA interfaced with a camera, from other side interfaced with VGA monitor. This research has used a DEII Altera board that integrates a Cyclone II 2C35 FPGA device and 8M byte external SDRAM. A TRDB\_D5M CMOS camera from Terasic is selected to be used. The TRDB\_D5M is suitable to develop a 5 Mega Pixel Digital Camera on the Altera DEII boards. The frame rate of the camera according to adjusted resolution of  $640 \times 480$  pixels is 77.4fps[44].

The DEII board integrates many features that allow the user to design a wide range of circuits, since all connections are made through the Cyclone II FPGA device, see Appendix A. This research has used some of the board's integrated devices and features. The 7-Segment Displays have used for testing purposes in order to observe the frames sequence number, the value of the pixels, or the result of a single pixel. The VGA video port is used to connect the monitor to display captured frames or a video stream as needed. One of the expansion headers, offered also by the board, has used to connect and interfaces the TRDB\_D5M camera with the FPGA. From the integrated external memories offered on the board, the 8M byte SDRAM is used which is sufficient to handle the required frames of AHD algorithm as will be explained in section 4.4.2. DEII board's devices and features that are chosen to be part of the design of this work are presented in Appendix A in more details.

The hardware description language that is used to build the designed system was Verilog, while Quartus II Web Edition software tool from Altera has been used for Verilog code development. Quartus software offers compiling and testing for the designed modules. In addition Quartus programmer tool is used to integrate the built modules into the FPGA device.

For simulation purposes, ModelSim simulator software by Altera has been used, which allows observing the results as a wave forms.

### **4.3 Key Optimization Parameters**

This work aims to implement the AHD algorithm on FPGA while achieving the maximum throughput with minimum device utilization in real time environment. Hardware design on FPGA is influenced by some parameters. The trade-of between those parameters to get the best design is different from case to case. As an example, some cases focus on the system performance ignoring the other factors such as power consumption, cost and logic utilization. Thus during hardware designing, the desired optimization parameters have to be set as targets to be met. Scope of this thesis is to measure the performance of the system using following key optimization parameters:

1. **Throughput.** Which is the amount of data the system can process during a unit of time. In image processing applications, the frame rate which is the number of output frames that can be processed per second is used as a measure of the design throughput. In this work, the throughput is computed using the number of required clock cycles to operate one frame over the system's frequency. The performance is increased with the increment of the throughput. However from another side, by increasing throughput, a penalty of power consumption and size of design is paid. Some implemented work of tracking systems on FPGA, such those presented in sections 2.3.2-2.3.5 have set throughput as the top priority to be achieved which in the other side affected other parameters such as the amount of generated logic. For the system in this work, the priority is set for optimizing the logic utilization that at the same time achieves the throughput of real time. Thus the minimum throughput that must be achieved is at least 30 fps, which is the real time throughput. Limitation on throughput is imposed in order to optimize other parameters of logic utilization, cost and power dissipation thought this research.
2. **System frequency.** System speed increases with the increment of the clock frequency and thus the throughput. However, the power consumption increases too as the frequency increases since it is in the current dissipation equation as was explained in section 1.4. System's clock frequency of this work is set to ensure data synchronization between I/O peripherals and other internal processing blocks of the FPGA. The speed of transferring data from the memory into the processing units is also considered. The FIFOs that are used as buffers between the memory and the processing units to overcome the difference of frequencies between them are reflecting on the system speed too. The speed of filling the FIFOs and processing the data via the processing units and store the results into FIFOs then to memory again must be the same or faster than the system frequency. Therefore system frequency must be set to meet the timing of the design and to achieve the desired throughput. Sections 4.6 and 4.7 explain the selection of the system frequency according to the design architecture of the processing units.

3. Logic Utilization. While the smallest logic of FPGA device is the logic element, the amount of logic elements used to implement a design is called logic utilization. The logic element itself consists of internal components. Mainly for all FPGA devices, the logic element consists of one or more lookup tables (LUT), which works as a function generator that implements any function with the LUT inputs [6, 45], as Figure 4.1 shows.

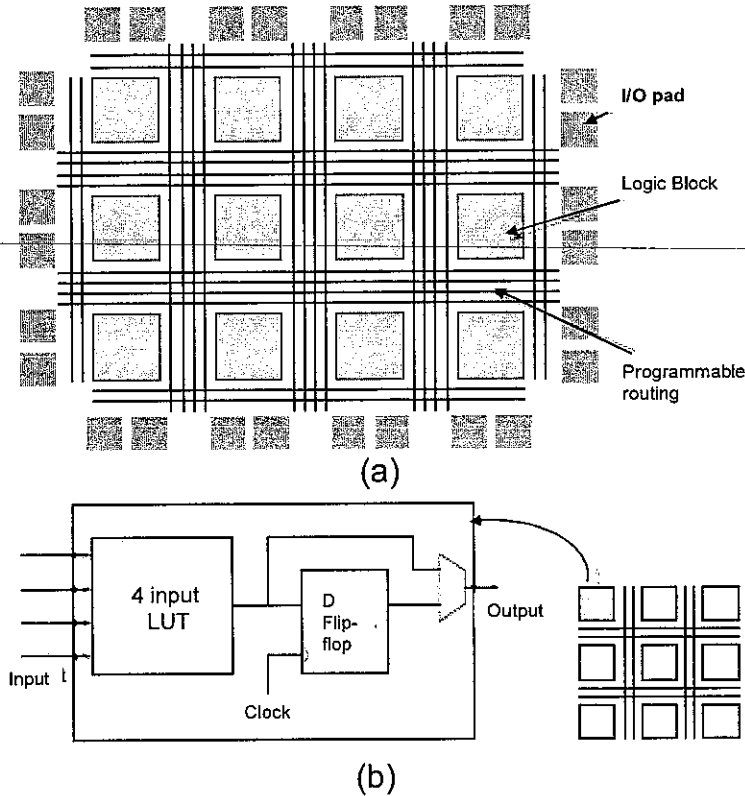


Figure 4.1: (a) Architecture of the basic elements of generic FPGA (b) A simplified architecture of logic block [6].

In addition, a programmable register is available to be connected with the output of the LUT. Also there are programmable connections to direct the output of the logic elements to other logic elements or to the I/O peripherals. By reducing the logic utilization of the FPGA, the device area, cost, power dissipation. We have considered the logic utilization's optimization as a target to be reached in implementing our design in addition to processing the frames at real time.

## 4.4 Analytical Study

We have analyzed AHD algorithm from software implementation's perspective in Software Selection of section 3.3. While implementing the algorithm on the FPGA, this research has focused on hardware capabilities and system specifications. In other considerations, how to implement AHD algorithm with specific issues on a hardware with a specific capabilities. An analytical study for algorithm steps according to the hardware is discussed in subsection 4.4.2.

### 4.4.1 Image Representation and Data Row

An important factor that influences the quality of the results, computation time and the required space is the image representation. The image can be represented in different color models with different color depths. A color model is the mathematical model which describes the way to represent the colors as tuples of numbers, typically as three or four values or color components. Examples of Famous color models are red, green, blue (RGB) model and luminance/chrominance YUV model. The color depth is the number of bits used to represent the color of a single pixel or known as bit per pixel (bpp). In this research, data row term which is a set of bits that represents one pixel is used in describing the developed system.

With a Higher color depth, a broader range of distinct colors are produced and a higher quality of image is returned. However, to set a higher quality representation of the image will option a bigger size of data row, which is a critical choice while using limited resources devices such FPGAs. The original data row width used by Terasic model that interfaces the TRDB camera with the FPGA is 30 bits. Initially, the data comes from the camera in a 12 bits Bayer pattern form. It is then converted to 12 bits represent each color, 10 bits of each are taken to represent 30 RGB bits model (the most 10 significant of the 12 data bits are taken). Memory space issues and number of bits involved in each process operation and consequently the computational time, specifies the limit of color depth that could be used. Reducing the data row width without losing the data quality is also the other border of the quality limitation.

In DEII board, the width of data bus to SDRAM is 16 bits. If data row were represented by 30 bits, then two clock cycles are needed to access the data row each time a read/write operation from the memory has occurred. In addition, two FIFOs with 16 bits are used to store each half of the pixel and then to read from both of them to display the pixel by a VGA or to perform operations on it. Since the developed system is meant to apply several process operations on the pixel value in a synchronous manner, the pixel value is needed to be ready within one clock cycle. Therefore, the data row width or the color depth is set to the same size of the SDRAM data bus. This size simplifies the system and avoids asynchronous problems that could occur. Within 16 bits, the RGB color model can be presented using High Color [46] of image representation that uses 16-bit color schemes with 5 bits to represent red, 5 bits to represent blue and 6 bits to represent 64 levels of green [47]. This format is considered sufficient to display photographic images [48]. In image processing, representing the pixel with the most significant bits while ignoring some of the least significant bits for a pixel with large number of digits does not affect the quality of the image. Since when a pixel value is represented by a large number of digits the lowest significant digits value represents a very small value compared to the most significant ones, i.e. if the red color of the pixel is 5,000,100, it will not be affected by adding or subtraction 100 from it. Thus, the 30 RGB bits can be easily converted to the 16 high color models using the most significant bits from each color channel to represent its corresponding one in the 16 bits model. Thus, the most 5 significant bits are taken for red color and so on.

Many image processing algorithms prefer to deal with gray scale images instead of the color ones. For RGB image, operations such as subtraction should be applied for each color channel, the green, the blue and the red. Thus, an operation should be applied three times. In contrast, in the gray scale representation of an image, only one value represents the pixel that carries the intensity information [49]. Using the gray scale in our hardware design provides a less computation time and less registers and logic elements that need to be used.

Many methods are used to convert from RGB to gray scale. The lightness method uses averages of the most prominent and least prominent colors:  $(\max(R, G, B) +$

$\min(R, G, B) / 2$ . Another method, the average method, simply finds averages of the RGB values,  $(R + G + B) / 3$ . The conversion to gray scale which depends on colors' luminance is the method used by MATLAB. It assigns a specific weight to each color,  $.2986 \times R + .5870 \times G + .1140 \times B$  [2].

With an FPGA implementation, there are two drawbacks of using the conversion methods mentioned above which are the use of the floating point and the multiplication or division operations. Implementing floating point with large number of arithmetic operations on FPGAs is very expensive in terms of the number of logic elements required. Also, using multiplications and, worst, using division operations is costly if it is used a lot in the design [6].

A suggested solution presented by N. H. Tan [50] is to use the average method but dividing by 4 rather than 3,  $(R + G + B) / 4$ . Division by 4 in the binary system can be performed using shift operation for two bits. This will obtain a 25% darker image than the one obtained by average conversion method. However, it does not make a big different on moving object tracking's results.

Finally, the image representation in this system is following the gray scale representation using the average method divided by 4. A 12 bits of gray scale is used during calculations, while to display results on VGA, 10 bits are used for all channels. Reducing number of bits from 12 to 10 for VGA does not affect image quality, since the binary image is represented in black and white and all bits values will be even ones or zeroes.

#### **4.4.2 AHD Analysis Based Hardware Implementation**

The initial step of AHD algorithm finds the adaptive threshold for all pixels of the window within 30 frames. Thus, it is needed to accumulate values during calculation time and to use these values later in the tracking time. Determining the threshold requires 30 frames of the video stream. As has been discussed in the analysis for software implementation, the  $k$  factor that specifies the detection sensitivity is taken as 6 according to some lab's tests. While building the binary image, the value of  $k$

determines the number of preceding frames that is going to undergo hybrid difference calculations.

To calculate the adaptive threshold, 30 frames are needed, then at tracking time each frame needs its previous 6 frames to determine the binary image. Thus,  $30 + k=36$  frames of memory space is needed. Another issue is the size of the data row which is the number of bits that represent one pixel. For data row representation, 16 bits gray scale has been used as was mentioned in section 4.4.1. The frame size used in the system is  $640 \times 480$  pixels according to the resolution offered by the Terasic camera. Therefore, the required space needed for the threshold is window size  $\times$  no. frames  $\times$  data row size  $= 640 \times 480 \times 30 \times 16 = 18.432$  M byte. The 18.432 M byte is larger than the available size of the 8 M byte SDRAM memory used in this design. To adapt the required space with the one available and to keep the system running on the real time, a frame sequence technique was developed.

	Frame t	Frame t-k		Frame t	Frame t-k		Frame t	Frame t-k
group 1	1	5	group 1	13	7	group 1	25	19
	2	4		14	8		26	20
	3	3		15	9		27	21
	4	2		16	10		28	22
	5	1		17	11		29	23
	6	0		18	12		30	24
group 1	7	1	group 1	19	13			
	8	2		20	14			
	9	3		21	15			
	10	4		22	16			
	11	5		23	17			
	12	6		24	18			

Figure 4.2: Frame sequence technique to determine the threshold value, where  $k=6$ .

Figure 4.2 shows sequences of frames from 1 to 30 categorized in groups. According to the algorithm as explained previously, the mean value (M) is calculated by equation (3.2) as given in subsection 3.2.2,

$$\mu'_{ij} = \frac{1}{N} \sum_{i=1}^N D'(i, j)$$



Thus, the calculations are passed from  $t = 1$  till  $t = 30$ . The calculations for the 30 frames are done via 30 iterations. The numbers listed in groups opposite to each other of the figure represent the numbers of two corresponding frames (Frame  $t$  and Frame  $t-k$ ) that have to be subtracted during iterations. In the first iteration, frame 1 ( $t$ ) and frame 5 ( $\text{abs}(1-6)$ ) are subtracted in order to find the hybrid difference values and so on for the rest of the iterations. Indeed the subtraction operations are done on the pixel level. As will be explained more lately, to save access and retrieve computation time, subtractions are done for the entire frame's pixels then the results are accumulated to be used with the results of the next frame. This is done instead of applying the operations on one pixel through the thirty frames before moving to the next pixel.

At each time there will be six frames in the memory and two lookup tables with the frame size that are used to accumulate the mean and standard deviation values. The lookup tables at the end will contain the threshold domain's borders,  $\mu'_{ij} - \delta'_{ij}$  and  $\mu'_{ij} + \delta'_{ij}$ . Six frames are used to calculate the hybrid difference then the next six frames are loaded gradually instead of old frames that are not in need any more. The first group contains frames 1, 2, 3, 4, 5 and 6. After calculating the hybrid difference for each frame in the first column with the corresponding one on the opposite column, the next frames have to be loaded. Since the next group (group 2) still need group 1 frames (frames from 1 to 6), loading the next frames after that is done gradually. When  $N=7$ (the seventh iteration) in equation 2, the seventh frame is loaded and use frame 1 to find the  $D^t$  values. After that iteration, Frame 1 is not needed any more and the next frame, Frame 8, is loaded on Frame 1 position. Therefore, each time a frame from the six frames is discarded, the new current frame is stored on its' place. In this way the number of frames needed at any moment can be reduced from 30 frames to 7 frames.

With this frame sequence technique, the memory size needed is  $640 \times 480 \times 7 \times 16 = 4,300,800 \text{ byte} = 4.301 \text{ M byte}$ . By adding two frames which are the size of lookup tables, the memory size needed to calculate the threshold and store it is  $640 \times 480 \times 16 \times (7+2) = 5,529,600 \text{ byte} = 5.530 \text{ M byte}$  which is within the available memory size of 8 M byte.

Another issue to discuss is simplifying the AHD calculations. The standard deviation is calculated by the formula

$$\sigma'_{ij} = \sqrt{\frac{1}{N-1} \sum_{t=1}^N (D'(i, j) - \mu'_{ij})^2}$$

as has been stated in section 3.2.2 by equation (3.3). To simplify what under the square root,

$$\begin{aligned} & \sum_{t=1}^N (D'(i, j) - \mu'_{ij})^2 \\ &= \sum_{t=1}^N (D'^2(i, j) - 2\mu D'(i, j) + \mu^2) \\ &= \sum_{t=1}^N D'^2(i, j) - 2\mu \sum_{t=1}^N D'(i, j) + N\mu^2 \end{aligned} \quad (4.1)$$

and since  $\mu = \frac{1}{N} \sum_{t=1}^N D'(i, j)$  and by multiply with N,

$$N\mu = \sum_{t=1}^N D'(i, j) \quad (4.2)$$

By using equation (4.2) for the middle term of (4.1), equation (4.1) become

$$\begin{aligned} & \sum_{t=1}^N D'^2(i, j) - 2N\mu^2 + N\mu^2 \\ &= \sum_{t=1}^N D'^2(i, j) - N\mu^2 \\ &= \sum_{t=1}^N D'^2(i, j) - N \left( \sum_{t=1}^N \frac{1}{N} D'(i, j) \right)^2 \end{aligned} \quad (4.3)$$

From equation (4.3), the base formulas have to be calculated to determine the adaptive threshold (the mean and standard deviation) are  $\sum_{t=1}^N D'^2(i, j)$  and  $\sum_{t=1}^N D'(i, j)$ .

If N=30, by defining  $\sum_{t=1}^N D'^2(i, j)$  and  $\sum_{t=1}^N D'(i, j)$  for all pixels frames, then some



and synchronizes the entire system to work correctly. Interface modules connect the FPGA with external devices. The TRDB\_D5M camera, the external SDRAM and VGA monitor are the main external devices used in this developed system. CCD\_Capture module receives the incoming data captured from the camera and generates data row, column and frame counters that are used to define the beginning and the ending of each coming frame. With each clock cycle, the data is captured by CCD\_Capture, which is 12 bit data row in Bayer pattern format, and sent to Row\_to\_Gray module. Row\_to\_Gray module retrieves the original value of the pixel represented in 30 bits RGB data row and then converts it to 16 bit Gray scale data row.

AHD algorithm is executed via two modules, the Threshold Definer (TD) unit and the Binary Image Builder (BIB) unit. These two units are the core of the system design since regardless of initialization processes or loading processes, most of the time one of these two units is running. Both of them take data from the FIFOs and output the results into the FIFOs again. While one of the units is working, as will be explained in section 4.6 and 4.7, the Finite State Machine is managing the control of data flow between them.

SDRAM\_Controller is the largest module of our design; it gives control to SDRAM to store data and fetch it as required by the algorithm or the output display. To match the processing speed of the SDRAM controller with the data rate of the camera, the data are buffered to a First In First Out structure (FIFO). FIFOs allow data to pass across the different clock domains of the camera (96 MHz), the memory (125 MHz), and the VGA (50 MHz). Thus, the incoming Gray data row is buffered into Write\_FIFO before the SDRAM writing operation is done, and is buffered from the SDRAM into Read\_FIFO after reading operation was done. The buffered data in Read\_FIFO is used later in AHD calculations or sent to VGA module to display it on the monitor.

The Finite State Machine is the heart module of the systems that controls the start of read or writes to FIFOs and from which FIFOs. Pixel mapping also for the proper address of the frame that supposed to be read from the memory is also performed within this module. In addition, controlling data flow between the camera, FIFOs, the

algorithm's units and between the SDRAM is performed. To decide when to read from the camera 30 frames and start calculating threshold through the Threshold Definer unit, or when to read 6 frames to start calculating the hybrid difference and build the binary image through Binary Image Builder unit, are all managed by the Finite\_State\_Machine module. The functionality of the Finite\_State\_Machine in order to control the developed system is illustrated in more details in section 4.8.

Two important blocks of SDRAM\_Controller, the 'Internal Address and Length Control' block, and the 'Auto Read/Write and Priorities Control' block are shown in the architecture. The first one responsible for assigning addresses, generated by the state machine, to the proper read or write address register that is related to one FIFO of the design. At the beginning, it sets the addresses specifying which frames are going to be processed, incrementing the addresses until the end of the frame is done automatically within this block. Important signals indicate when the frame has been finished and when six frames or thirty frames are generated and send to the Finite\_State\_Machine. Accordingly, the Finite State Machine decides which level of execution should be performed. Set and Reset of these signals are done cooperatively between this block and the Finite\_State\_Machine. The main functionality of 'Auto Read/Write and Priorities Control' block is to give the control to one FIFO at a time to access the SDRAM. It enables RD\_MASK or WR\_MASK for the specific FIFO to increment the address through the 'Internal Address and Length' block, while loading the current assigned address for that FIFO to SDRAM address buss. Distributing the accesses between FIFOs and to be sure that no more than one FIFO is accessing the memory, are assured by this block.

One of other complementary modules is the VGA\_Controller that generates the signals which drive the LCD. Also it generates 30-bit data output, a horizontal sync, a vertical sync, and data enable signals according to the data rate of the monitor. This module can be used to output the obtained results.

The I2C\_CCD\_Config module configures the camera using an I2C bus. Through it, the exposure level can be set, using the board's switches as inputs.

For debugging issues, SEG7\_LUT\_8 and SEG7\_LUT modules are used to trace some results via seven segments sets that are integrated on the DEII board.

The sdram\_pll module generates 125 MHz clock for the SDRAM, and 20 MHz clock for control logic and the algorithm units, using a phase-locked loop.

#### 4.6 Threshold Definer Unit's Frequency, Functionality and Architecture

The core part of the system is the Threshold Definer Unit and the Binary Image Builder unit. In Figure 4.4, the design architecture of the threshold unit and the interact connections with FIFOs and SDRAM are illustrated. The unit simply consists of one subtraction, two addition and a multiplication operations.

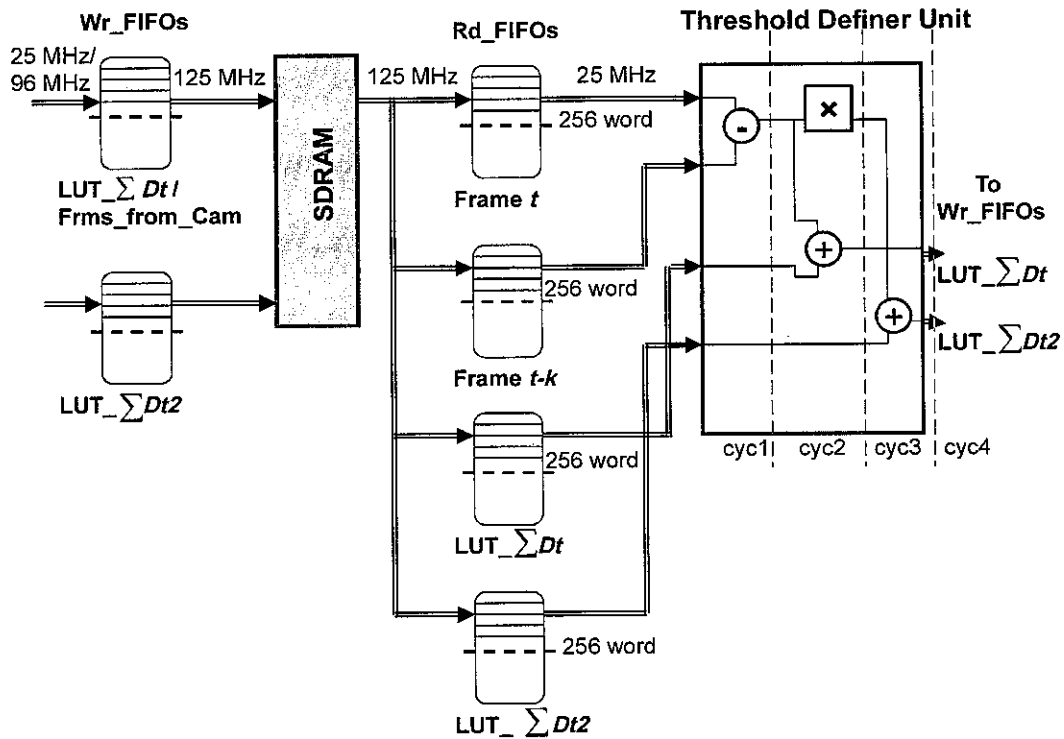


Figure 4.4: Architecture of the Threshold Definer Unit.

The SDRAM receives the read enable signal, while proper addresses of current frame  $t$ , frame  $t-k$  and the threshold LUT tables are prepared for each Read\_FIFO by the SDRAM\_Controller. Reading operation is started by the SDRAM through the 16

bits' bus to fill each FIFO by 256 words, where 256 is the memory burst. Using big memory burst saves updating the address for each memory word, instead it is updated every 256 word when the burst finished. Additionally, big memory bursts reduce the overall access time of the memory since the number of times the memory is used is significantly reduced. Since the SDRAM is a single port memory, only one read or write operation is done at a clock cycle and therefore only one FIFO at a time accesses the memory. The distribution of the memory access is controlled by the SDRAM\_Controller in such a way that each FIFO gets the access for one memory burst then reserve it again after all FIFOs had a turn, where this operation is repeated until a full frame have been buffered.

To synchronize the system, all data from four FIFOs of Figure 4.4 must reach the Threshold Definer unit at the same time. This means reading access for all FIFOs must be done for once at least before the unit start processing. To fill four FIFOs with 256 words' burst, a number of  $4 \times 256 = 1,024$  cycles are needed. It is a big amount of delay but the point is that SDRAM speed is 125 MHz, while the Threshold unit speed is set to adapt the requested loading time, which that processing speed is set to be slower than the speed of filling all FIFOs by the memory. The advantage of the FIFOs that have used that it is a dual port FIFOs and the frequency of each port is independent from that of the other port. Thus read and write operations could be done at the same time with different speeds. This ability makes the SDRAM\_Controller work as a multi-port device with multi-operation while the SDRAM itself has only one port with one operation at a time.

The Read\_FIFOs from the other side are working with 25 MHz frequency to feed the data into the Threshold Definer unit. Since the internal architecture of the unit is using the pipelining technique, only one clock cycle is required to process each pixel as explained in section 4.9. Of course, a delay of five cycles has to be paid at the beginning until the final output is ready by the pipelining, but this delay is paid only once at the beginning of each new frame.

The memory bandwidth has to be bigger than the sum of other units' bandwidths. To fill all Read\_FIFOs,  $1,024 \times (1/125M) = 8,192$  ns is required. While to process the data that have filled FIFOs, only 256 cycles are needed, since data is taken in

parallel from FIFOs at the same time. With the 25 MHz frequency, it needs  $256 \times (1/25M) = 10,240$  ns of time to process the data of FIFOs. Since, the speed of the memory is larger than the Threshold unit, data always will be ready for the unit.

A memory loading delay time, 8,192 ns, is paid once at the beginning of processing each new frame. This delay is overlapped for next bursts to the end of the processed frame, since loading from memory is done while the processing of old one is running by the unit. The selected size of FIFOs, 512 words, which is double the size of the memory burst, gives a space to write another burst to FIFOs while the first one is still there.

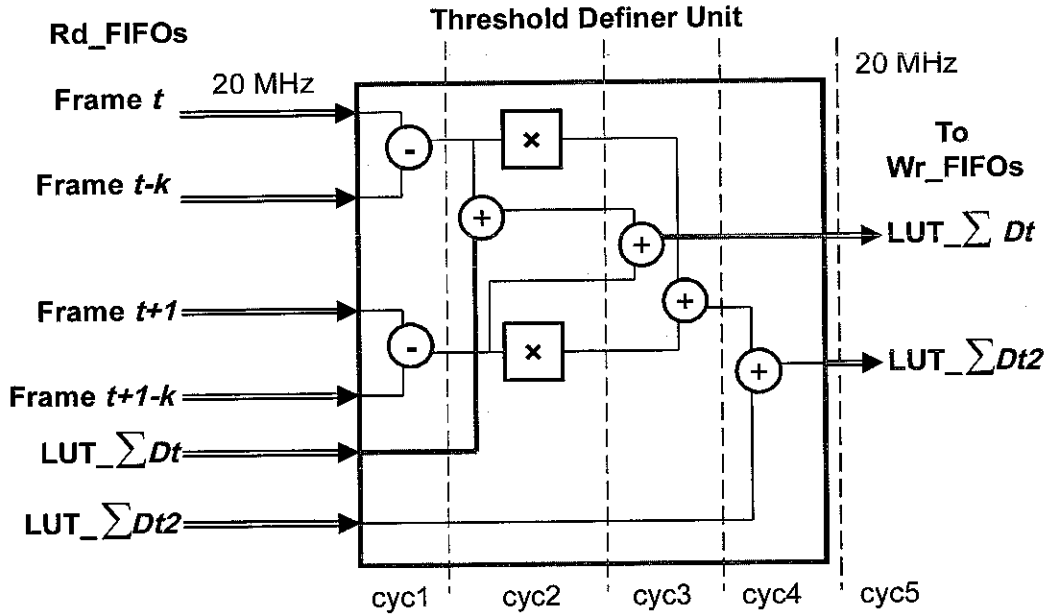


Figure 4.5: Improved architecture of the Threshold Definer Unit.

The throughput of using the unit to calculate the threshold via thirty frames is  $25M / (307,200 \text{ pixel} \times 30 \text{ iterations}) = 2.7$  frame per second (fps), where  $307,200 = 640 \times 480$ , is the number of cycles to operate one frame. In order to improve the throughput of the system, the Threshold Definer unit is modified to calculate threshold for two frames and accumulate their obtained results. The improved design of the unit as shown in Figure 4.5 uses six FIFOs while the old threshold unit is duplicated. The penalty of the improved design is implied in two issues, waiting time to fill FIFOs and the pipelining unit's depth. The former is caused by adding



additional two FIFOs, requiring to wait first time for  $256 \times 6 \times (1/125M) = 12,288$  ns. Increasing number of FIFOs does not affect number of processing cycles needed by the Threshold unit, it is always 256 cycles. However, the cost comes from the delay that the unit has to wait until all FIFOs are filled for the first time and from the other side, the number of logic elements that has to be increased. Since the FIFOs load delay has increased, the unit frequency should be reset to adapt it. Therefore the speed of loading data from FIFOs to the Threshold Definer unit is reduced to 20MHz. Using 20 MHz for the unit costs  $256 \times (1/20M) = 12,800$  ns to process data of six FIFOs containing 256 pixels in each, while the throughput has improved to  $20 M / 307,200 \times 15 = 4.34$  fps, where 15 is half the value of the thirty frames since two frames are being processed by the improved unit at a time. The adaptive threshold is calculated once at the beginning of running time with initial conditions or calculated again with system reset request. There for the throughput of TD unit is not considered as a problem while the main throughput of the system is considered by the BIB unit that will be running at tracking time.

#### 4.7 Binary Image Builder Unit's Frequency, Functionality and Architecture

Binary Image Builder (BIB) unit is run at the tracking time to build the binary image that represents the moving objects. It uses the threshold values that have been calculated at initial time by the Threshold Definer unit, where they are stored at the end in LUTs in the memory. BIB design consists of two subtractions, four comparison operations and two OR and an AND gates as shown in Figure 4.6. Building the binary image needs to find the set of hybrid differences of each pixel of the current frame with its corresponding pixels from the successive six frames. To improve the unit throughput, its design is made to calculate hybrid difference values for two frames each time. As shown in Figure 4.6, the architecture of BIB unit is built to take input values from six Read\_FIFOs, where three FIFOs of them buffer memory data of the current frame, Frame (t) and the successive two frames; Frame (t-s) and Frame (t-s+1), where s a value from 1 to k (k is set to be 6 as mentioned in 3.3.1). Subtraction operations are performed between data that come from the first FIFO, the second and the third FIFO.

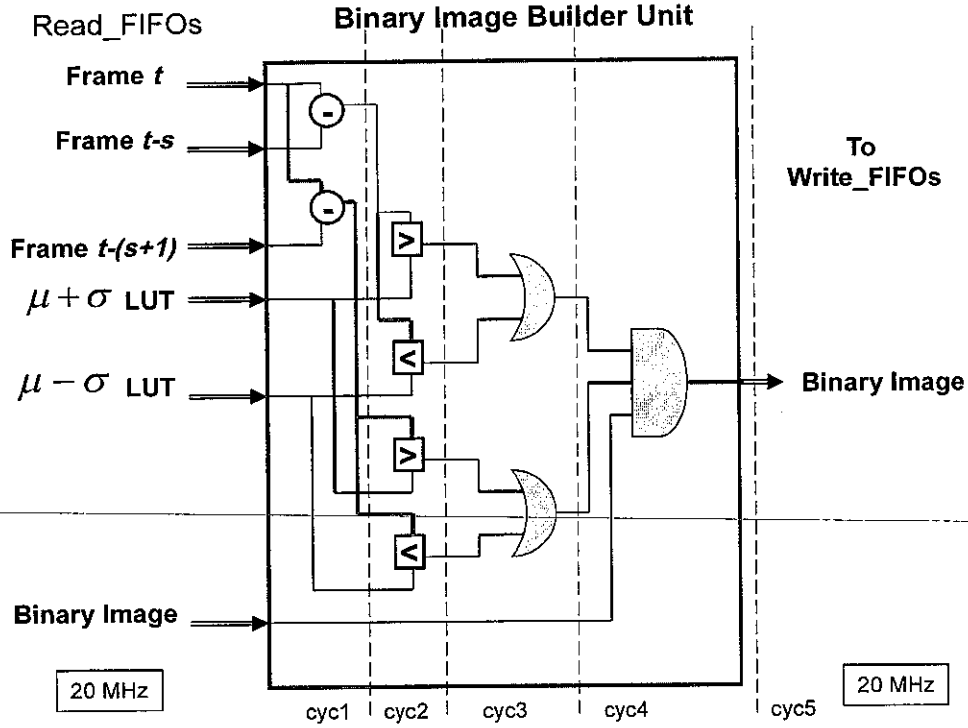


Figure 4.6: Architecture of Binary Image Builder Unit.

For the other three FIFOs of the design, two of them are used to read the stored threshold LUTs which contain the values of  $\mu + \sigma$  and  $\mu - \sigma$ , while the last FIFO is used to store the intermediate or the final obtained binary image. After subtraction, the result undergoes two comparisons with  $\mu + \sigma$  and  $\mu - \sigma$  for that pixel. The OR gate is used to determine if at least one of the comparison results is true, either  $D^t > \mu + \sigma$  or  $D^t < \mu - \sigma$ . Two OR gates are used, one for the obtained result of  $Frame\ t - Frame\ (t-s)$  and the other for  $Frame\ t - Frame\ (t+1-s)$ . For a pixel, all  $D^t$  values via six frames have to achieve the threshold condition, since only two  $D^t$  values are determined at a round, the value need to be stored and compared in the next round with the results of other  $D^t$  values. Therefore, AND gate is used to determine if the old test result with the current results are achieving the condition. An intermediate Binary Image is stored in the memory and updated each round until subtraction with the six frames is finished, and then the final binary image of the current frame will be

ready. When the binary image had been built, it can be displayed into a monitor or can be stored in some storage according to the application needs.

The FIFOs used in the Binary Image builder are the same as those used for the Threshold Definer unit. To adjust the frequency, the speed of processing should not be faster than the time the memory takes to fill the six FIFOs. The time needed by memory to fill up Read\_FIFOs is 12,288 ns, while 12,800 ns is needed by the BIB unit to process that amount of data, working with 20MHz. The frequency of the system and the control finite state machine are set at 20 MHz, since it's the frequency that achieves the highest throughput of using six FIFOs. Unifying the frequency for system units reduces skew and metastability problems that could occur between blocks with different clock domain within the same system. Only the frequency of the SDRAM is 125 MHz, which is higher than the system frequency, since the memory is and has to work faster. With the design of BIB unit, the system throughput at tracking time is  $20 \text{ MHz} / 307,200 \times 3 = 21.7 \text{ fps}$ , where  $307,200 \times 3$  is the number of cycles needed to execute the algorithm on one frame. 307,200 is the number of pixels of the frame, since with the pipelining, to process a pixel within the BIB unit cost just one clock cycle. Multiplying number of pixels of a frame with three since the hybrid difference is calculated through six frames, while the rounds reduced from six to three after processing two frames in parallel each round.

To improve the design of BIB unit in order to increase the throughput, line buffers are used. The improved design as represented in Figure 4.7, consists of two main parts, line buffers and the processing unit.

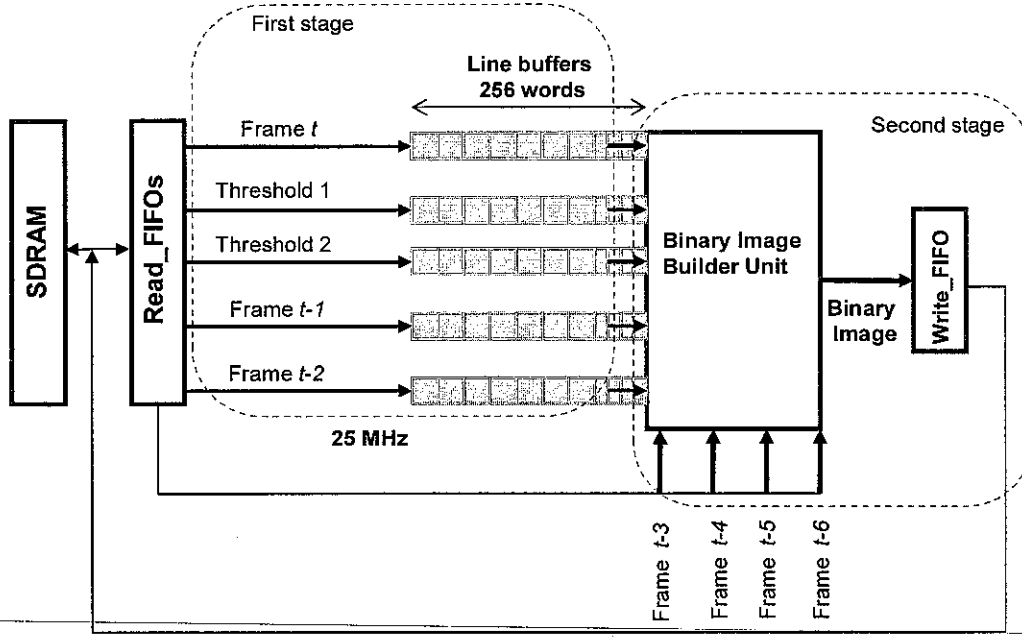


Figure 4.7: Improved Architecture of Binary Image Builder Unit

Processing data is done within two stages, where each stage processes 256 words (pixels). The stages work alternatively, when one is processing the other is idle. The first stage which is the line buffers consists of five shift line buffers for the current frame  $t$ , the two threshold values  $T1 (\mu + \sigma)$  and  $T2 (\mu - \sigma)$ , and two frames of the six frames which are to be subtracted from Frame  $t$  (Frame  $(t-1)$  and Frame  $(t-2)$ ). Each line buffer is 256 words long, for the five line buffers 1,280 words are used. The created line buffer is built using memory blocks rather than registers to optimize the logic elements utilization. By this design the number of FIFOs that access the memory at any time is six, including the FIFO used to store the obtained binary image back into the memory, the camera write FIFO for new frames and the read to VGA FIFO. By considering all FIFOs in the system while setting the frequency, the system becomes more stable. The frequency is set to 20 MHz, since  $6 \text{ FIFOs} \times 256 \times 8 \text{ ns} (1/125 \text{ MHz}) = 12,288 \text{ ns}$  and  $256 \times 50 \text{ ns} (1/20 \text{ MHz}) = 12,800 \text{ ns}$  are the required time to read bursts from the memory into all FIFOs and the time to read that burst from all FIFOs respectively. Figure 4.8 illustrates the flow of processing the two stages between FIFOs, line buffers and BIB unit on the time line at tracking time.

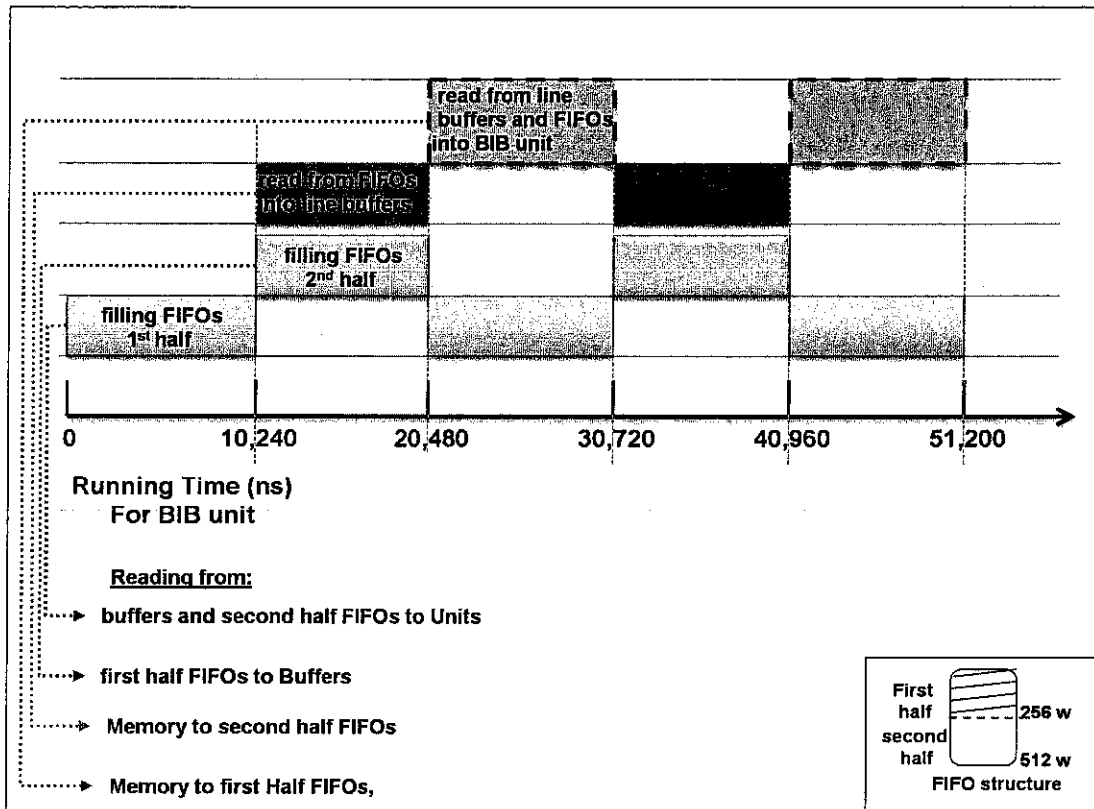


Figure 4.8: Processing flow between FIFOs, line buffers and BIB unit on the time line at running time for the improved Binary Builder unit.

The FIFOs first read a burst (256 words) from the memory for pixels and threshold values that are used for line buffers. When the unit starts to work, for 256 cycles, the values are buffered from the five FIFOs into the line buffers. Since each FIFO has place for two bursts (512 words), during buffering operation, the addresses inside SDRAM\_Controller for the FIFOs are automatically changed and read operations for a burst for the rest of frames, Frame (t-3), (t-4), (t-5) and (t-6) are started. Synchronized signals are used to organize the work between the two stages and to synchronize the data during all levels. Buffering data from FIFOs into line buffers does not start until all Read FIFOs have finished reading a burst from the memory. A synchronized signal is used to indicate when burst read operation for all FIFOs are completed. The same synchronized signal is used to start reading bursts from the memory into the FIFOs for other frames while reading from the FIFOs into line buffers is running. For the next 256 cycles, pixels are read from the line buffers that are now full, and from the FIFOs with the same speed to determine the *HD* value

for 256 pixels. At this time buffering new data to line buffers is stopped. While BIB unit is active, reading operations from the memory into the FIFOs is done for the next 256 pixels to fill the line buffers for the next 256 cycles in next stage. By this design, processing one pixel of a frame is taking two cycles, one cycle to buffer half of values into the line buffers and other cycle to read the rest of the values from the FIFOs.

The inner design of Figure 4.9 illustrates the internal architecture of the improved BIB unit. The unit integrates subtraction and comparison operations for all six frames in contrast with the old structure that integrates operations for two frames only. The old structure obtained the needs to repeat using the unit and accumulate the results via three iterations to get the final result.

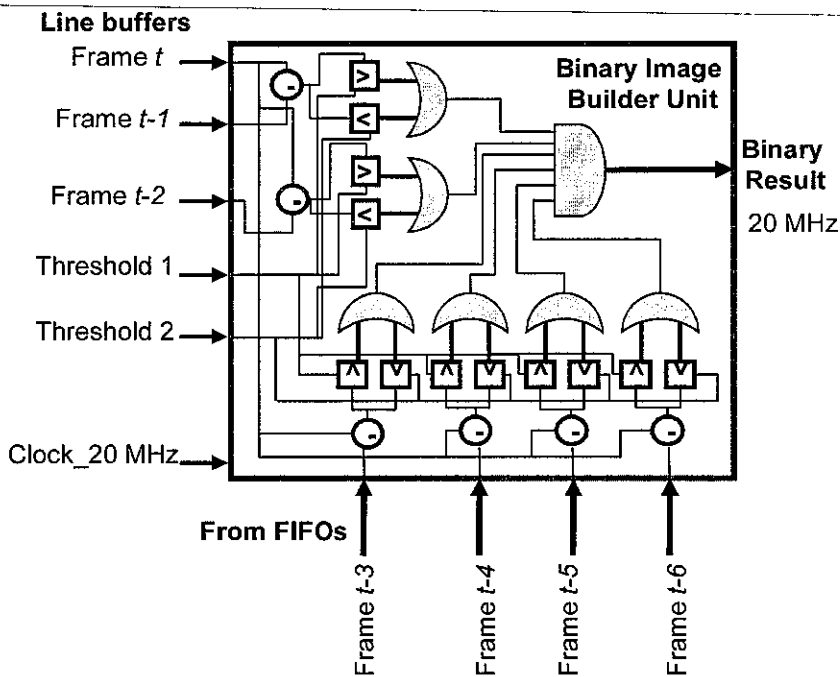


Figure 4.9: The improved internal architecture of BIB unit.

When the BIB unit is not idle (during the second stage), it takes one clock cycle to process each pixel. Since the unit is using pipelining technique, the operating time to obtain a result for a pixel takes one clock cycle, after a delay paid once at the beginning of each frame. Considering the waiting time to make all data ready (first stage), 256 cycles for 256 pixels, which means each pixel is processed during two clock cycles, one for loading data and another one for the data to be processed

through the BIB unit. At the meantime, control signals are used to organize the work between the two stages, FIFOs and reading the correct addresses at the correct time. By this design the throughput is improved from 21.7 into 32.55, since  $20 \text{ MHz} \times (307,200 \times 2) = 32.55 \text{ fps}$ .

#### 4.8 Finite State Machine and Control Signals

Finite State Machines (FSM) are the heart of any digital design [51]. It is used to model a system that transits between internal states. Regarding the sequential circuits, the transition between FSM's states does not follow a sequential order. The transition between the states is decided according to the current state value and external inputs.

The design of this research has used a FSM to control the built digital system as has showed in Figure 4.3. The FSM is used to test external commands to activate proper control signals to control operation of data path [52]. From another perspective, the FSM works on achieving the hardware timing and meeting the system specifications, specifically, to control the data flow between inputs, outputs, memory and FIFOs.

The state diagram of the FSM is illustrated in Figure 4.10, where outputs of the state machine are functions of inputs and states; thus it is a mealy state machine. The states' coding is an important design issue that reflects the system speed and the number of logics used. When the number of bits that represent the states is small, the number of logic used is reduced. The selected state's coding specifies the number of comparison operations and the number of bits needed to be tested during state transition i.e., a move from one state to another, which reflects on the speed, power and area of the design. Some detailed work have been done on optimizing the FSM by selecting the most optimized coding technique according to the number of the state and transition for each state which is out of this research area and is left for the future research work [53].

The gray coding style has been selected, to represent states, since only one bit is changed to transit from a state to its next state. Thus, the transition between states





Internal Address and length control block of Figure 4.3. With ‘end\_setup’ signal, the next state is set to ‘IDLE’ and enable write signals are disabled.

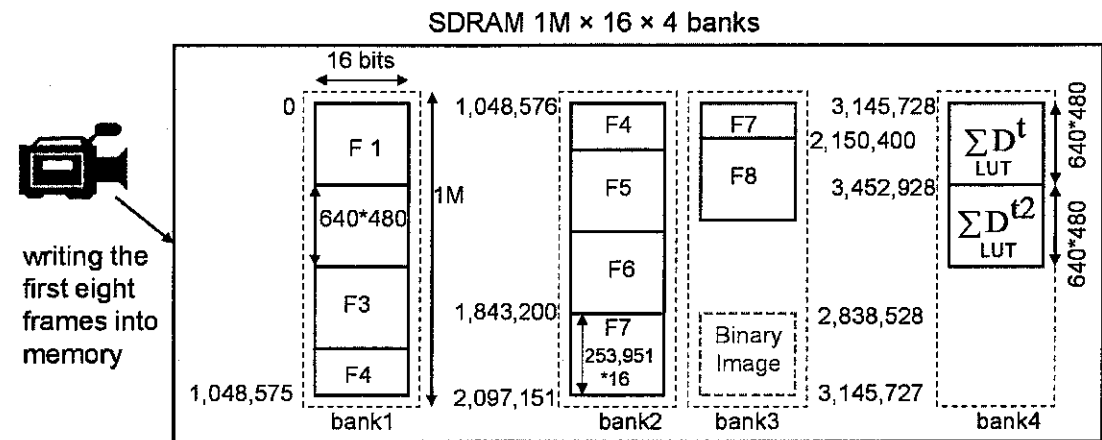


Figure 4.11: The memory map of frames and lookup tables (LUT) on the SDRAM.

Next state is ‘IDLE’ state that tests if the next state should be ‘LOAD\_FRAMES’ to apply the frame sequence technique that has been explained in section 4.4.2 or deciding to finish the state machine by stick to ‘IDLE’ state.

The job of ‘LOAD\_FRAMES’ state is to load the number of subsequent frames from the camera into the SDRAM for the first load then to load new frames gradually to replace the already finished frames. As in the final design of the *TD* unit of Figure 4.5 in section 4.6, parallel calculations are done to determine *HD* for two frames of the 30 frames at a clock cycle. The two parallel calculations require accessing four frames, Frame  $t$ ,  $t-k$ ,  $t+1$  and  $(t+1)-k$ , which are available in the memory after the first sequence of frames have been written during ‘LOAD\_FRAMES’ state. If the number of loaded frames are set to 6, then the need to load the next two frames after processing the six frames are required. However, overwriting the old six frames cannot be done since they are still required in the next rounds of calculations, i.e. when  $t=7$ ,  $t-k=1$ , means Frame 1 still under use. By processing frames 7 and 8 with frames 1 and 2, frames 1 and 2 are not going to be used in future *TD* calculations and thus they can be overwritten. The same situation is repeated for the rest of frames, therefore, the number of initial loaded frames is set to eight rather than six. After operating the 7th and the 8th frames, two new frames are loaded in the place of frames 1 and 2 since they are not in need any more. Similarly after processing frames

9 and 10 that use frames 3 and 4 ( $t-k$ ) for HD calculations, frames 11 and 12 are stored instead of frames 3 and 4 locations, and so on. Thus, after processing the first eight frames, a new two frames is stored in the memory in the oldest frames' locations. Specifying the location where the new frames have to be stored is determined using a rotating pointer starting at Frame 1's location to Frame 8's location of the memory (locations of F1 and F8 in Figure 4.11). To adapt the writing frequency from the camera (96 MHz) to the SDRAM (125 MHz), 'WRITE\_FIFO1' is used to buffer the data. WRITE\_FIFO1's write enable signal is assigned by the FSM to 'WR1\_CCD'. The 'WR1\_CCD' signal is a combined function of FRAME\_VALID and LINE\_VALID signals that are obtained by the camera to indicate the boundaries between different frames and lines within the frame, where a valid pixel is obtained when both of them are active. At the same time, data coming from the camera are directed to the 'WRITE\_FIFO1'.

---

When the eight frames or the two frames have been loaded into the SDRAM, the control signal 'stopWrFrms' is sent through Internal Address and length control to the FSM, where the last transits the system to the next state, 'TD' where the required signals and addresses for defining thresholds are set.

By having eight frames in the memory, the adaptive threshold algorithm is ready to start implementation. In 'TD' state, pixel mapping is done first to determine the addresses of the four frames ( $t$ ,  $t-k$ ,  $t+1$  and  $(t+1)-k$ ) that will be used by the current TD unit calculations. Only the first addresses of frames are required to be sent to SDRAM\_Controller module while incrementing the rest of frame's addresses is done automatically via the Internal Address and Length Control block. Pixel mapping with address pointer register are keep saving the order of the frames sequences to determine the proper sequence for the next round until all the thirty frames are processed.

Also read and write enable signals for read and write FIFOs are activated in 'TD' state as well as the coming data from read FIFOs is directed to the TD unit inputs while the output of the unit is directed to write FIFOs' input. By setting addresses, directing the data path and activating control signals, the TD unit processes frames to determine the adaptive threshold in the way stated in section 4.6. Enable writing or

reading signals are set for the FIFOs, however the SDRAM\_Controller works automatically through the Auto Read/Write and Priorities Control block to generate SDRAM's read and write signals. Auto Read/Write and Priorities Control block checks the availability or the full space of FIFOs. According to that, SDRAM signals are generated. For the case of writing from FIFO into SDRAM, the control block checks if the number of lines filled by the FIFO reached 256 or more, then it generates the proper SDRAM signals to make writing operation while blocking any other FIFOs from accessing the memory until the writing operation is finished. The same is applied for reading case, but the condition here is to have enough empty space of the FIFO for incoming 256 words from the memory. Signals for reading from the memory into the FIFO or writing from the FIFO into the memory are generated automatically through the SDRAM\_Controller, while signal enable reading from the FIFOs into the algorithm unit or writing the obtained results of the units into FIFOs are set by the FSM.

The FSM is used mainly in this design to control threshold definer implementation at the beginning of running time. After the adaptive thresholds are determined the system stays in BIB state for the rest of running time as long as system reset signal has not been activated. Building the binary image is done when the system reach 'BIB' state, which is the last one in later rounds after determining the adaptive threshold through thirty frames. In 'BIB' state, TD unit signals are deactivated and the data of read FIFOs are directed into the BIB unit instead of TD unit since the same FIFOs are used. The output of the BIB unit is directed into write FIFO and from there into the Binary Image location that is reserved in the memory to store the results as shown in Figure 4.11. First seven frames are stored in the memory in the same locations used before for TD. The seven frames are needed since the current frame (Frame  $t$ ) with its previous six frames are used in building the binary image. After processing Frame  $t$ , the next new frame is stored in the oldest frame's location since it is not used anymore and frame pointer of Frame  $t$  is updated to be Frame  $(t-1)$ . The pointers registers are used to point to the current frame and the previous six frames in the correct order. Pixel mapping and addressing are managed automatically within this state (BIB state) through SDRAM\_Controller and by using control signals without using additional states as with TD unit. The reason behind that

is loading new frames and processing the existing frames is done in parallel, which means no need to stop the processing unit for a while to load new frames and then activate it again through the states. The improved BIB design unit as was explained in section 4.8 processes data of frames burst by burst, 256 words each time, rather than finishing the whole frame at once. For the first stage, five FIFOs are used to read 256 words of data from the memory into the line buffers and one FIFO is used to write new data from the camera into the memory instead of the burst had just shifted into line buffers. As well in the second stage, four FIFOs are used to read data from the memory into the BIB unit, while an additional FIFO is used to store the results in the memory and another FIFO is used to display the results of 256 pixels into VGA monitor.

---

The state machine stays in 'TD' state until the whole frames,  $t$ ,  $t-k$ ,  $t+l$  and  $(t+l)-k$ , for the current round are processed. 'Next\_Round' state tests control signals 'frame30', 'eightFrameSignal' and 'frame\_signal' that are generated by the SDRAM\_Controller while the execution of different states is running. In 'Next\_Round' the FSM is directed to the next state supposed to be executed according to the tested signals. If the 'frame30' signal is active, then the thirty frames that define the threshold, have been covered and the next state is 'BIB' state. The 'eightFrameSignal' that indicates when the first 8th frames have processed is tested. When processing of thirty frames is not completed, if 'eightFrameSignal' is active then it means the eight frames that were written in the memory have been processed and the state machine has to load the next two frames, thus the next state is set to 'LOAD\_FRAMES'. If neither of the previous signals is active, then the signal 'frame\_signal' is tested. 'frame\_signal' indicates that the two frames under parallel processing by TD unit have been operated while unprocessed frames are still in the memory waiting to be processed. Thus the next state is set to 'TD' if the first eighth frames have not been processed yet, and is set to 'LOAD\_FRAMES' if the 'eightFrameSignal' is active, since two frames are loaded each time after processing the current frames by TD unit.

'frame30' signal indicates when all 30 frames have been processed through the TD unit and lead the FSM to 'Finalize T1 and T2' state and then to the last state,

'BIB'. In 'Finalize T1 and T2' the final threshold values,  $\mu' - \sigma'$  and  $\mu' + \sigma'$  as defined by equation (3.2) and (3.3) in subsection 3.2.2, are calculated using the accumulated  $D'$  and  $D'^2$  values for all pixels of the two LUTs that were determined through the previous states via 30 frames. The results of final threshold are stored again in the LUTs where they are used later in comparison operations in order to build the binary image.

#### 4.9 Pipelining and Data Flow

Pipelining is an implementation technique that is used to increase the throughput and parallelism [54]. The pipelined design on FPGA utilizes the advantage of the parallel processing capabilities of the FPGA to increase the efficiency of sequential code. To implement a pipeline, the code must be divided into stages separated by storage elements. The outputs of each stage deliver the results of the current stage that is lagging behind the input by the number of stages of the pipeline which specifies the pipeline depth. The last output is invalid until the pipeline is filled [55].

The pipelining judged to be synchronous or asynchronous depending on the data transition controller for the stages, [54].

Synchronous pipeline blocks are built to work in parallel during 'TD' and 'BIB' states to optimize the number of clock cycles needed to process each pixel through TD and BIB units. Since the transition between pipelining stages is controlled by the same clock cycle of the FSM, it is a synchronous one. As explained before in the FSM in section 4.8, the 'TD' state reads data from six locations in the memory to determine the adaptive thresholds and then it writes back the results into the memory. This operation should be repeated within each clock cycle while the FSM is repeating the 'TD' state. This means read/write and processing operations of TD unit supposed to be done within one clock cycle (before the next FSM's clock cycle occurs). The FSM, pipelining and the read/write operations of FIFOs from processing unit side are set with the same speed (20 MHz). To adjust the number of required clock cycles to determine the AT or to build the binary image and to speed up the system, the

pipelining is used to receive data from six reading FIFOs and send the results into two writing FIFOs within one clock cycle. In our design, we have built two pipeline architectures for the two developed units of sections 4.6 and 4.7 as represented in figures 4.12 and 4.13. The pipelining of the two units, Threshold Definer unit and Binary Image Builder unit are built to optimize the number of clock cycles needed to execute the algorithm on a pixel during the 'TD' and 'BIB' states.

#### 4.9.1 Pipelining of the Threshold Definer Unit

Figure 4.12 shows the pipelined data path that represents the Threshold Definer unit. The pipeline consists of five stages. Thus, to find  $D^t$  and  $D^{t+1}$  for the current two pixels of frame  $t$  and  $t+1$ , five clock cycles of latency are needed until the last output is ready. The latency of five clock cycles is paid once each time at the beginning of processing a new frame of the frames sequences. However, for processing the rest of the pixels in each frame, only one clock cycle is needed for each pixel. Thus, five cycles of latency against saving four cycles for  $307,200 \times 2$  pixels are almost negligible.

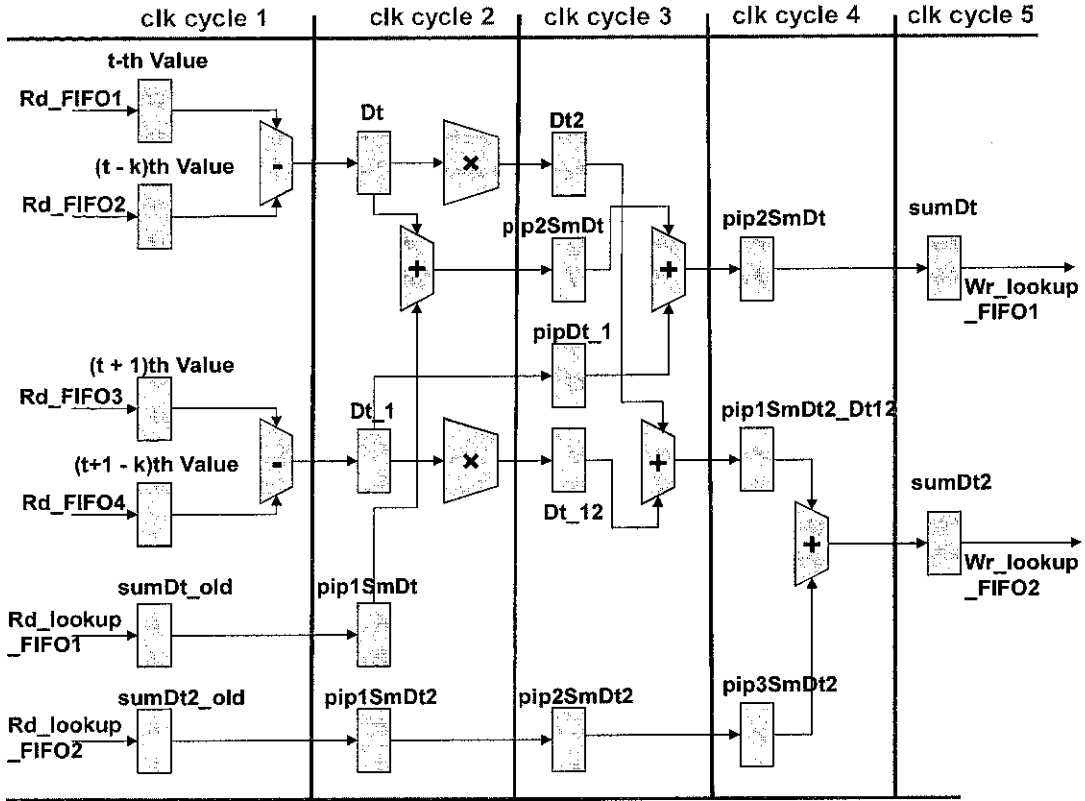


Figure 4.12: Pipelining data path of Threshold Definer unit

At the first stage of the data path, data are shifted from FIFOs into registers ' $t$ th Value', ' $(t-k)$ th Value', ' $(t+1)$ th Value' and ' $(t+1-k)$ th' which are the pixels' values of frame  $t$ , frame  $t-k$ , and the next frame of  $t$  and its corresponding frame that are processed in parallel. Data from the other two FIFOs, the accumulated  $D^t$  value from the first lookup table and the accumulated  $D^{t^2}$  value from the second lookup table respectively are shifted to registers 'sumDt\_old' and 'sumDt2\_old'. Within the same clock cycle a subtraction process to find  $D^t$  values for frames  $t$  and  $t+1$  are executed. However, subtraction results are ready with the second clock cycle where they are delivered to input registers of the second stage, ' $Dt$ ' and ' $Dt_1$ '. Pipelining stages are receiving inputs from the previous stage or RD\_FIFOs while delivering outputs to the next stage or WR\_FIFOs with each clock cycle.

When  $D^t$  of pixel( $i,j$ ) is determined in stage two, a new pixel is loaded from the FIFO into the input register in stage one. Thus 'smDt\_old' and 'smDt2\_old' values are pipelined from the first stage into the second stage registers, 'pip1SmDt' and

'pip1SmDt2', to keep the corresponding threshold values of pixel(i,j). In order to find  $D^{t2}$  of  $D^t$  for pixel's frame  $t$  and  $t+1$ , shift by 1 to the left is applied for values of 'Dt' and 'Dt\_1'. Additionally, to save the number of pipelining stages and since  $D^t$  and old  $D^t$  summation values are ready by stage two, 'Dt' + 'smDt\_old' is implemented in this stage.  $D^{t2}$  values are delivered with the third clock cycle to 'Dt2' and 'Dt\_12' registers of stage three as well as the summation result that is delivered to register 'pip2SmDt'.

In the third stage, addition operations are applied for 'Dt2' with 'Dt\_12' and 'pip2SmDt' with 'pipDt\_1', where 'pip2SmDt' carries the summation of old  $D^t$  value of LUT with  $D^t$  of frame  $t$  and 'pipDt\_1' is the  $D^t$  value of frame  $t+1$  that has shifted via stages. The old value of  $D^{t2}$  from the second LUT is shifted to 'pip2SmDt2' register since it has not been used yet. With the fourth clock cycle, the summation of  $D^t$  of the two frames,  $t$  and  $t+1$ , and the accumulated value from old rounds are determined and stored in 'pip2SmDt' register while the last addition operation is done. The accumulated  $D^{t2}$  value that was kept through three stages is added to the summation of  $D^{t2}$  of frame  $t$  and frame  $t+1$  that was determined during stage three.

Finally,  $D^t$  summation is delivered to 'sumDt' register and  $D^{t2}$  summation to 'sumDt2' register at the fifth clock cycle. The last two registers are the Threshold Definer unit's output registers that are wired to WR\_FIFOs inputs in order to store the results in LUTs in the memory.



Table 4.1: Pipelining data flow table of Threshold Definer unit.

Clock cycle	t-thvalue	t-kth value	(t+1)th value	(t+1-k)th value	Dt	Dt_1	pip1SmDt	Dt2	Dt2_1	pip2smDt	pipDt_1	pip3SmDt2	pip2SmDt	pip1SmDt2_Dt12	sumDt	sumDt2
1	v1	v1	v1	v1	x	X	x	x	x	x	x	X	x	x	x	X
2	v2	v2	v2	v2	v1	v1	v1	x	x	x	x	X	x	x	x	X
3	v3	v3	v3	v3	v2	v2	v2	v1	v1	v1	v1	X	x	x	x	X
4	v4	v4	v4	v4	v3	v3	v3	v2	v2	v2	v2	v1	v1	v1	x	X
5	v5	v5	v5	v5	v4	v4	v4	v3	v3	v3	v3	v2	v2	v2	v1	v1

Table 4.1 illustrates the data flow of the pipeline of Figure 4.9, where columns represent the pipeline registers from different stages and lines represent values of these registers during clock cycles' transition. Symbols v1, v2...etc., represent the corresponding register value of pixel number 1, 2...etc. respectively. Thus, v1 of column 't-thValue' in the first row represents the value of the first pixel of frame  $t$ , v1 in column 'Dt' is the determined  $D^t$  of the values were in 't-thValue' and 't-kth value' in the first clock cycle. Last outputs 'sumDt' and 'sumDt2' for pixel 1 are ready in the fifth clock cycle. For the first pixel of the frame, it takes five clock cycles to reach the pipeline output. Otherwise, the outputs of pixels for the rest of the frame are ready just one cycle after the previous pixel's output is ready. Therefore, only one clock cycle is needed for each pixel to implement the AHD since the pipeline works in parallel with different pixels at the same time.

Using pipelining for unit's design rather than combinational logic has been selected because the critical path of the combinational design will be the longest path among the whole unit design while in the case of pipelining the critical path will be the longest path among the stages. As apparent in the designed unit in Figure 4.12, the logic design is distributed among the stages to get minimum possible longest path. Operations have been distributed in such a way that cost one clock cycle for each single pipeline stage.

#### 4.9.2 Pipelining of the Binary Image Builder Unit

The next level of implementation is to build the binary image after the threshold has been calculated. A pipeline has been developed for the Binary Image Builder unit as illustrated by Figure 4.13.

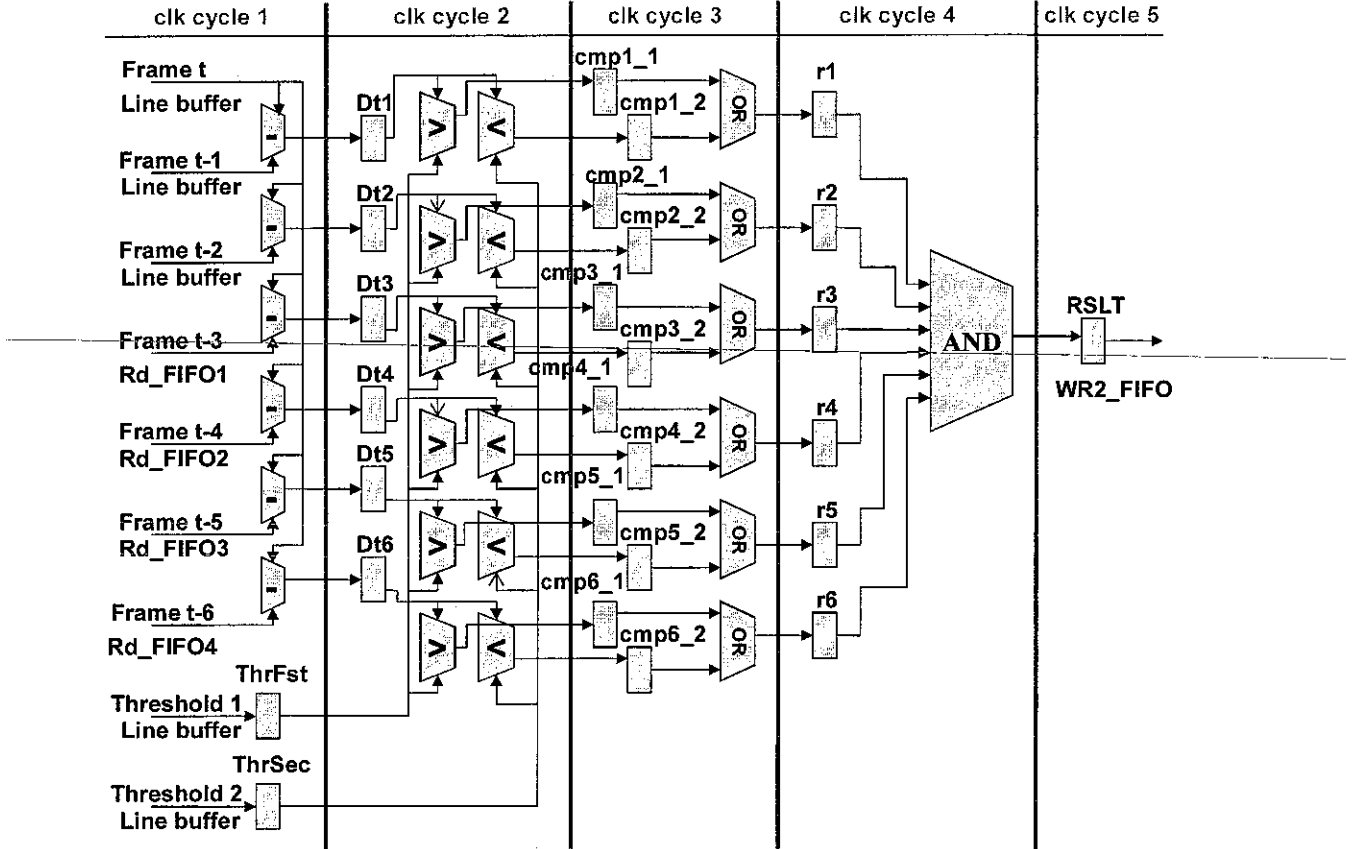


Figure 4.13: Pipelining data path of Binary Image Builder unit.

In the first stage of the pipeline with the first clock cycle, pixel values from location  $(i,j)$  of frames  $t$  and  $t-1$  to  $t-6$  with the two thresholds Threshold 1 ( $\mu + \sigma$ ) and Threshold 2 ( $\mu - \sigma$ ) are arrived. The received data as was explained in section 4.7 are delivered by five line buffers and four FIFOs. The threshold values are saved in registers to be used in the next stage, while the other values are subtracted from pixel value of Frame  $t$ . By subtraction operations,  $D^t$  values are determined and sent to the second stage registers ('Dt1' - 'Dt6'). In the second stage, comparison operations are done to compare  $D^t$  values with the threshold values which represent thresholds of the current pixel. The thresholds are used to test if  $D^t$  values are greater than  $\mu + \sigma$  and

if they are less than  $\mu - \sigma$ . Results of comparison are shifted into twelve registers with single bit of the next stage. Moreover, these values went under OR operations, since one condition must be satisfied, either  $D^t$  is greater than the first threshold or it is less than the other value of threshold. However one of the conditions for each  $D^t$  values, 'Dt1' to 'Dt6', have to be met, thus in the fourth stage, the results of OR operations are delivered to 'r1' to 'r6' registers where AND operation is performed for all of them to determine if all  $D^t$  values are not belonging to the threshold interval. The pipelining of BIB unit takes five clock cycles to determine the initial binary value of one pixel, which have to be paid once at the beginning of processing the frame, while it takes only one clock cycle after that.

In Table 4.2, the data flow of the Binary Image Builder unit of Figure 4.13 is illustrated through the first 5 clock cycles of execution.

Table 4.2: Pipelining data flow table of Binary Image Builder unit.

Clock cycle	Frame t to Frame t-6	ThrFst	ThrSec	Dt1 to Dt6	cmp1_1 to cmp6_1	cmp1_2 to cmp6_2	r1 to r6	RSLT
1	v1	v1	v1	x	X	X	x	X
2	v2	v2	v2	v1	X	X	x	X
3	v3	v3	v3	v2	v1	v1	x	X
4	v4	v4	v4	v3	v2	v2	v1	X
5	v5	v5	v5	v4	v3	v3	v2	v1

#### 4.10 Chapter Summary

Hardware implementation of AHD algorithm on FPGA has been thoroughly discussed. Using Cyclone device as a processor to control the external SDRAM and the camera connected through the DEII board has been justified. Some techniques

such as parallelization as with pipelining have been covered. This research has developed a finite state machine that controls the overall modules that comprise the system. Architecture of the hardware design on FPGA was explained through previous sections. In order to justify the methodology and hardware techniques that have been used, an analytical study is made. This chapter has covered the research methods, equipment used and the techniques followed or developed in order to meet the expected results as will be discussed in the next chapter.

---

## CHAPTER 5

### RESULTS AND DISCUSSIONS

#### **5.1 Chapter Overview**

This chapter discusses the performance and feasibility of the obtained system as a real time system. The system throughput and frequency speed settings have been discussed. Previously, comparisons with other systems have been discussed in the literature review in terms of methodology. This chapter explains the results thoroughly in the coming sections and sub sections. A comparative performance analysis using a new approach has been performed with the existing ones. At the end, an evaluation of this system will be discussed in terms of achieving the optimal system specifications.

#### **5.2 Experimental Setup and Deployment Strategy**

The implementation and simulation experiments on the FPGA have been applied by tacking real time Video stream in conditions similar to those used for the software implementation as mentioned in Chapter 3. The real time recording was taken inside the lab, in door place, under neon lights during day time while moving objects were humans. Figure 5.1 shows the deployment model of the system, which illustrates the data flow of the input and output through the system.

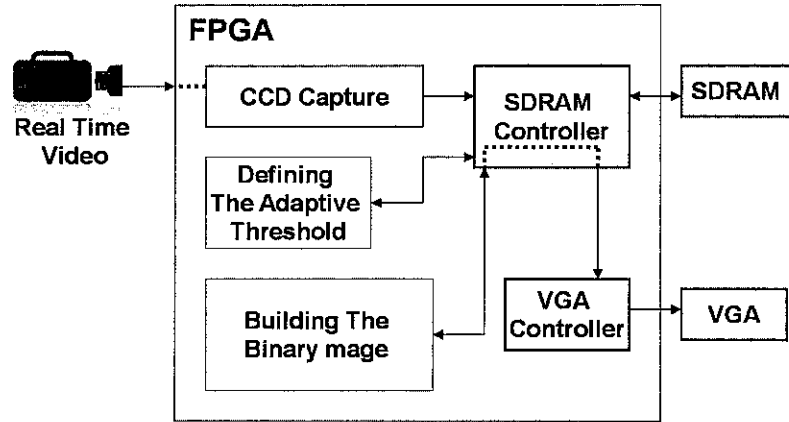


Figure 5.1: Deployment Model

Once the data come out from the camera it is captured by the FPGA via the “CCD capture” block where it is converted to the appropriate digital form for image processing. The formatted data is sent continuously to the external SDRAM memory through “SDRAM controller” block. Later on, after storing the required number of frames, the frames start to transfer to “Define Adaptive Threshold” block to calculate the threshold. When the threshold is calculated, “Building Binary Image” block starts the processing by receiving the data from the SDRAM. There the moving objects are detected and given a white color while the rest of the image is set to black. Read/write operations are interacting between the blocks and the SDRAM during processing the threshold and building the binary image. The final output is a binary image of the moving objects over a frame. To observe the output of binary image it is sent to VGA monitor through “VGA Controller” block that synchronizes the output frames.

### 5.3 Baseline Approaches

In the literature review of Chapter 2 in section 2.3, different design and implementation approaches of object tracking systems on FPGA have been described. This chapter uses those approaches for comparing performance analysis with the proposed approach. The following is a list of object tracking approaches that are going to be compared with the proposed design in section 5.6.

1. Image Segmentation and Pattern Matching [22], section 2.3.2.
2. Image Segmentation based on Color Threshold [11], section 2.3.1.
3. Principal Component Analysis [10], section 2.3.5.
4. Planer Motion Tracking using Matched Filter [18], section 2.3.3.
5. Color Detection and Binarization [24], section 2.3.4.
6. HW/SW co-design using Kernel Based [16].

#### **5.4 Performance Analysis**

The frame rate of a real time video system is the frames' speed that makes the human visual system sees it as a continuous scene. The frame rate itself is the frequency that a full frame is updated within it. Current real time video systems follow different standards of frame rates, but all rates belong to three standard rates which are 24, 25, or 30 frames per second (fps). Definite frame rate's and real time frame rate's definitions are the keys that have been used to evaluate how much close this system to the real time system and how much its speed.

The two core parts of our system are the Threshold Definer unit and the Binary Image Builder unit. The Threshold Definer unit can be run at initial time, a previous time before the real implementation. From the other side, the obtained threshold values are used at the real tracking time by the Binary Image Builder unit. However if the execution time of the Threshold Builder unit combined with the other unit could meet the real time rate, then it is feasible to run both units at the tracking time. When the system is fast enough to recalculate the threshold from time to time or with changes of lighting conditions during the tracking time, the adaptive threshold will be more accurate as detailed in the algorithm [4], since new changes in the environment are updated in the new threshold.

The system frequency is set to 20 MHz in order to achieve the best frame rate of the design. The Threshold Definer unit reads four frames and two LUTs from the

memory to calculate the frames differences and accumulates the results through thirty frames. Using pipelining has reduced the required clock cycles for each pixel from 5 to 1. However to synchronize the data coming from the FIFOs, as explained in section 4.6, a waiting penalty has been paid. The pipeline unit has to wait until all Read FIFOs are filled with 256 pixels' value, where 256 words is the selected memory burst. The system speed has to be less than the speed of reading the data from the memory into six FIFOs. Thus, the time needed to fill six FIFOs by the memory is  $256 \times 6 \times (1/125 \text{ M}) = 12,288 \text{ ns}$ , where 125 M is the memory speed. To process 256 word  $\times 6$  FIFOs in 20 MHz will cost  $256 \text{ cycles} \times (1/20\text{M}) = 12,800 \text{ ns}$ , which assures processing data later when it is ready for pipelining. For one frame, the required time is  $307,200 \times 50 \text{ n} = 15,360,000 \text{ ns}$ , while for 30 frames  $12,288,000\text{ns} \times 30 = 460,800,000 \text{ ns}$  is required. Loading next stream of data until the end of the frame is done while the pipelining is processing the current 256 pixels. Thus, the load delay time of 12,288 ns is paid only once at the beginning of implementing each frame which is negligible since it represents just 0.1 % of loading time for the whole frame. Therefore, the frame rate achieved by using the Threshold Definer unit for thirty frames is  $20 \text{ M} / 307,200 \times 15 = 4.3 \text{ fps}$ , where 15 is the number of iterations the unit have to be repeated until the threshold is determined. The iterations have been reduced from 30 to 15 with the new parallel design that processes two iterations each time. System frequency for the old developed design was set to 25 MHz that uses four read FIFOs as unit's input. However processing 30 frames are done through 30 iterations, since the old unit design performs one iteration at a time. Thus the achieved throughput was  $25 \text{ M} / 307,200 \times 30 = 2.7 \text{ fps}$ . Reducing the frequency to 20 MHz to adapt the delay loading time for six read FIFOs that provide parallel execution for two iterations at a time will achieve higher throughput.

A frame rate of 4.3 fps still doesn't meet the standard frame rate for a real time video system. However, this still does not pose a problem since the adaptive hybrid difference algorithm requires finding the threshold at the initial time of execution with an ideal background (without moving objects). At tracking time, the Binary Image is built depending on the threshold that was calculated before.



For the Binary Image Builder unit, the pipeline is used to calculate the hybrid difference for the current frame with other six frames. By using the improved unit that processes hybrid differences for pixels from six frames at the same time, the frame rate was improved from  $20\text{M} / (307,200 \times 3) = 21.7 \text{ fps}$  to  $20\text{M} / (307,200 \times 2) = 32.55 \text{ fps}$ . The frame size (307,200 pixels) was multiplied by 3 or 2 according to the number of clock cycles needed to obtain binary result for one pixel. It required three iterations per pixel to get the final result using the old design, since partial results are accumulated and used in next cycles to get the final output. From the other hand, the new design unit needs two clock cycles per pixel, one to load inputs and the other to process them. The old proposed design uses six read FIFOs while the new uses five read FIFOs and five line buffers for parallel calculations. The speed of the new design is set to 20 MHz considering in addition to two write FIFOs for the obtained binary image and for loading new frames from the camera. System frequency is set to adapt sharing the SDRAM speed between six FIFOs at any time (five for line buffers and data read, two for write and one for read to VGA). Even in the new design, eight FIFOs are sharing the SDRAM speed while only six FIFOs at any moment are accessing the memory.

To avoid clock skews, the system frequency is set to 20 MHz for processing units, read/write FIFOs' operations (with processing units) and the finite state machine. Using dual port FIFOs between SDRAM and other modules of different frequencies have prevented the clock domain crossing problems that could occur. The data is buffered into FIFOs before going to other unit or device.

This system works with 20 MHz speed and achieves 32.55 fps at tracking time, while achieving 4.3 fps for calculating the threshold through 30 frames at initial time. Table 5.1 below summarizes the performance analysis for the developed designs that has discussed in this section.

Table 5.1: Summarization of performance analysis for the developed designs

Designed unit	Unit's frequency	No. of input buffers (FIFOS)	Loading time from memory into buffers	No. of iterations to get the final result	Throughput
TD	25 MHz	4	8,192 ns	30	2.7 fps
TD (parallel design)	20 MHz	6	12,288 ns	15	4.3 fps
BIB (old design)	20 MHz	6	12,288ns	3	21.7 fps
BIB (improved design)	20 MHz	5	10,240 ns	2	32.55 fps

## 5.5 Synthesis Utilization

Figure 5.2 shows a detected moving object that has been obtained using the built system, the hand appears on the snapshots is used to move the object. The hand has been detected since it is moving too. When there is no motion, the screen stays black.

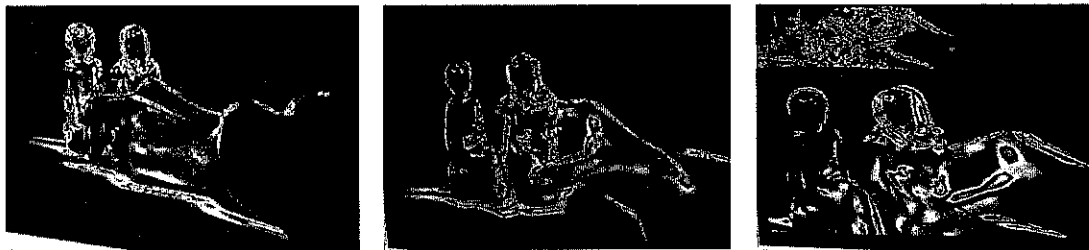


Figure 5.2: Detected moving object snapshots

Table 5.2 shows the synthesis utilization of our system as obtained from the compilation report of Quartus software. The system was achieved by using 1,821 logic elements from 33,216. While the number of used registers was 1,220, only one PLL has used to generate the system's 20 MHz clock frequency.

Table 5.2:FPGA's Synthesis Utility of Our System

<b>Total Logic Elements</b>	<b>Multipliers</b>	<b>Total Registers</b>	<b>Internal Memory Bits</b>	<b>PLL</b>
<i>1,821/33,216 -5%- (1,506 combinational functions, 1,220logic registers)</i>	<i>0/70</i>	<i>1,220</i>	<i>74,192/483,840 (15%)</i>	<i>1/4 (25%)</i>

## 5.6 Comparison with Other Systems

The implementation of Adaptive Hybrid Difference algorithm on FPGA has been compared in this section with other implemented systems presented by other researchers. Table 5.3 and 5.4 compare the results achieved by implementing tracking systems using different algorithms and techniques which have been mentioned earlier during the literature review. The differences the in achieved throughputs between this new system and other systems are discussed through these sections. To compare the synthesized utilization for configurable logic block (CLB) we have to estimate a basis of comparison between different FPGA devices. Next subsection deals with estimating a common unit for logic utilization comparison between Altera and Xilinx FPGA devices.

## 5.7 Logic Utilization Estimated Unit

As mentioned before, the main unit in FPGA is the CLB; however, its inner architecture is different for different vendors' devices. For Altera FPGAs, the smallest unit considered is the logic element which generally contains one LUT with 4 inputs [45]. On the other hand, Xilinx's FPGAs use a number of slices to evaluate logic

utilities. The slice contains two 4 input LUT with its related connections and output. Two similar slices are combined to represent a CLB [56]. Figures 5.2 and 5.3 illustrate the inner architecture of two FPGA devices from different vendors; Altera Cyclone II and Xilinx Spartan II respectively.

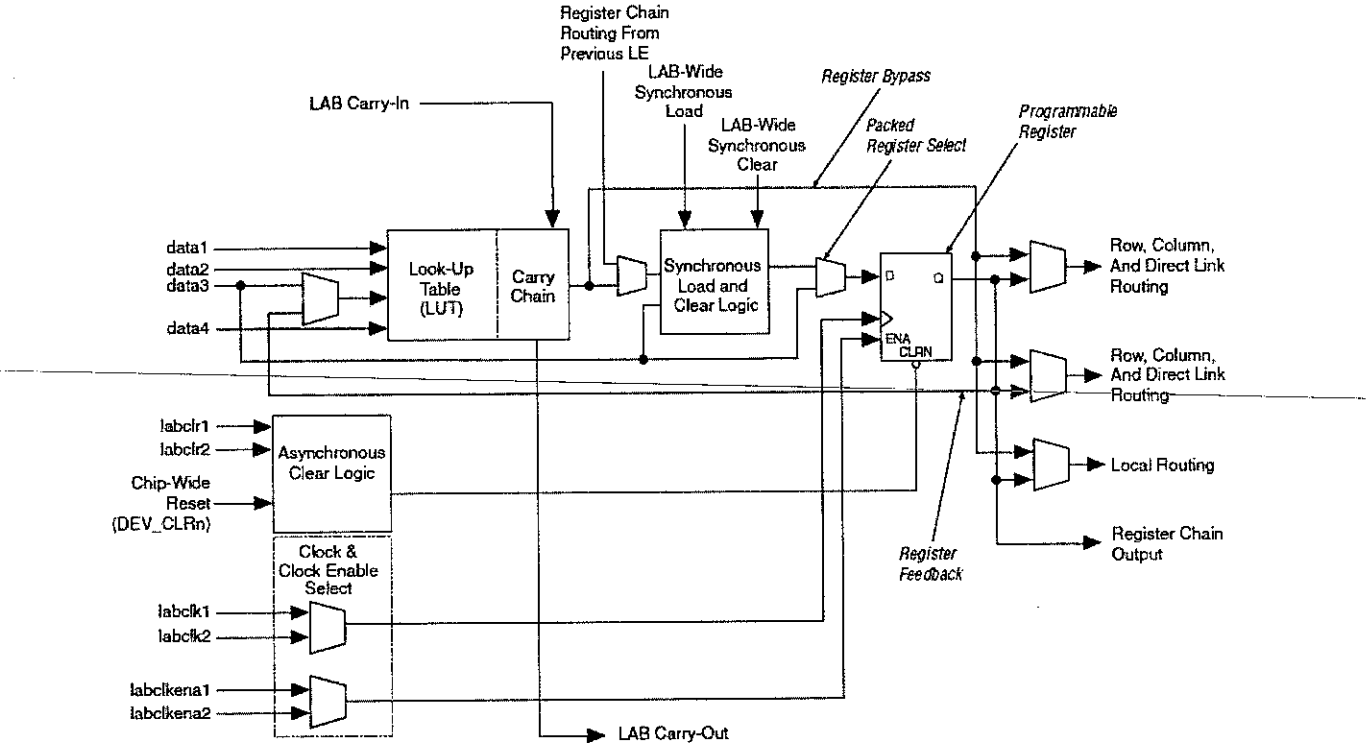


Figure 5.3: Altera, Cyclone II Logic element, inner architecture [45]

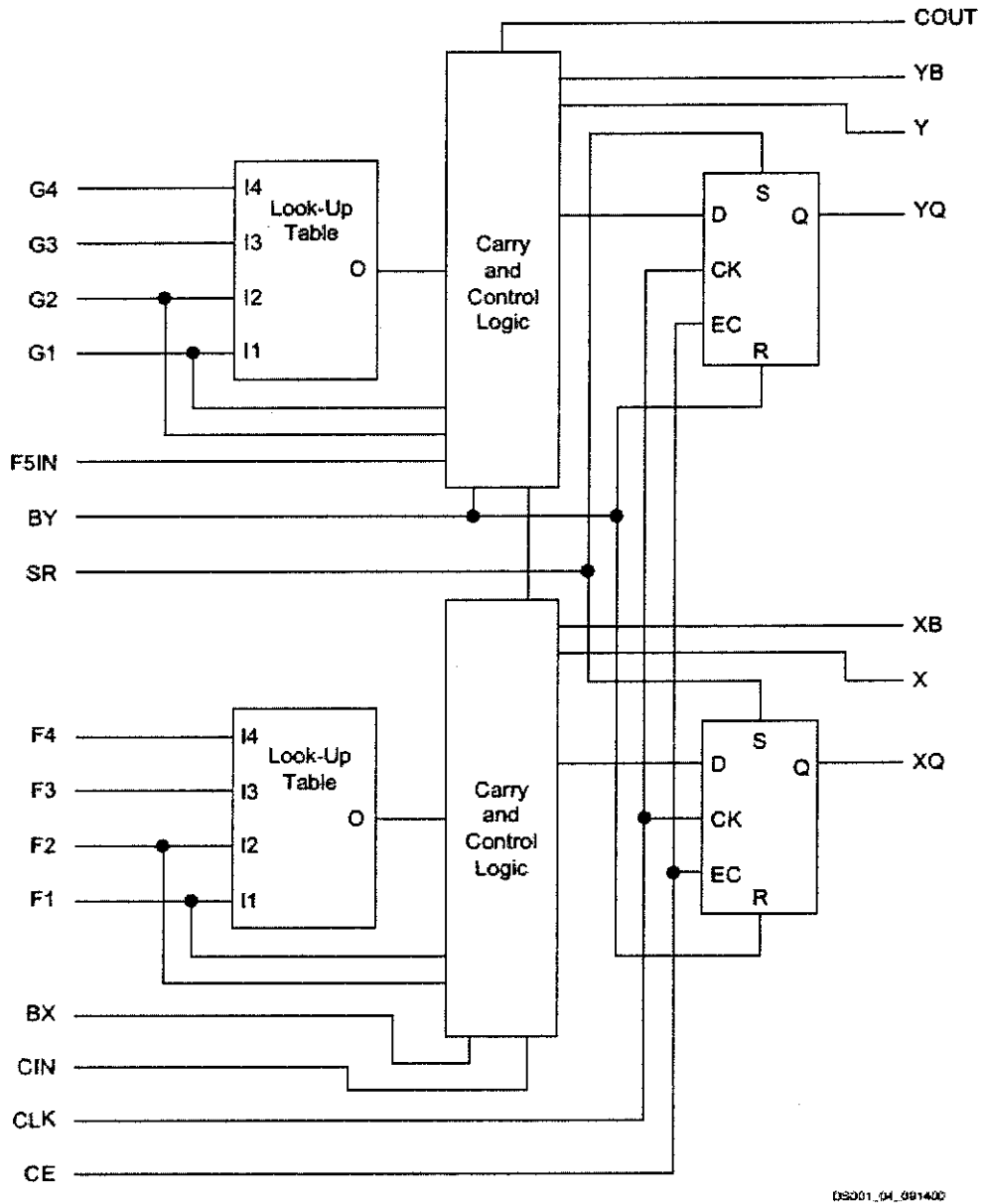


Figure 5.4: Figure 5.3: Xilinx, Spartan II slice, inner architecture [56]

From previous comparisons, in order to compare between works that used different devices, the logic elements of Altera, a device that contains one LUT is going to be used. For Xilinx, each slice is estimated by 2 logic elements of that within Altera device.

## 5.8 Comparison with Other Systems

Tables 5.3 and 5.4 list resource usage, performance and speed of different tracking systems that have been implemented on FPGA using different tracking algorithms and techniques. While making comparisons between the different designs, it is important to consider the size of the implemented frame and the FPGA device used which is illustrated in Table 5.5. Frame size specifies the number of clock cycles required to process the algorithm and as a result it increases or reduces the obtained frame rate.

In [10], I. Bravo et al. implemented the Principal Component Analysis algorithm and achieved high throughput of 121 fps for  $256 \times 256$  pixels with 100MHz speed using 8,450 LE which is 86% of the device resources.

---

The second and third rows of Table 5.3, Table 5.4 and Table 5.5 represent tracking systems that use the image segmentation algorithm. The first work is implemented by K. Yamaoka, et al. [22], which uses the region-growing algorithm to segment the objects from frames and has performed a very high frame rate but it depends also on the number of tracking objects as shown in Table 5.4. With 30 objects in a  $60 \times 80$  pixels frame, the throughput of the system is 887 fps with 20 MHz system clock frequency. Since the segmentation algorithm detects moving and still objects, in [22], the system has used extra hardware to track the moving object through frames and thus the number of logic elements and internal memory used was high, 31,987 LEs and 144,256 bits which are 56% and 2% of the device and memory resources, respectively.

The other work, in the third row of Table 5.3, Table 5.4 and Table 5.5, that uses the segmentation algorithm depending on color threshold developed by C. Johnston et al. [11], achieved video tracking system with 25 fps for  $768 \times 288$  pixels frame with 27 MHz system frequency. The work of [11] has used less logic resources of the device compared with other works, since 798 LE (10%) of logic elements were used. However 114,688 bits of internal memory were used which is not the lowest among other works, in addition the achieved frame rate is less than 30 fps which does not meet the real time frame rate requirement.

A related work of tracking a planar motion presented by K. Shimizu et al. [18] using Matched Filter, achieved a high frame rate, 1545.44 fps, with system frequency of 66 MHz for a  $64 \times 64$  pixels frame size. On the other hand, the used resources were high too, since 30,140 LE (44%), 124 (86%) BRAMs and 7 multipliers were used.

Y. Hsu et al. [24] have implemented a tracking system which depends on color detection of a specific color skin using HW/SW co-design. The system has archived frame rate of 37.3 fps for a frame with  $640 \times 460$  pixels with 25 MHz system frequency. The system has used 19,085 LE which is 54% of the device resources for the hardware part. This research work has ignored the software part in evaluation since it was built to save the detected frame on an SD card and to activate an alarm, which are out of the scope of our research.

A mixed HW/SW co-design developed by U. Ali et al. [16] using the Kernel Based algorithm to track a target object was implemented using medium amount of resources, 6,320 LE (20%), 3 multipliers (8%) and 165,888 bits of internal RAM (25%). Using 50 MHz for the system to track a target object through a frame with  $64 \times 64$  pixels, 290 fps throughput was achieved which meets the real time frame rate conditions.

Table 5.3: Resource Usage of different tracking systems on FPGA

Reference	Algorithm	LEs Used	Internal Memory Bits Used
Our design	Adaptive hybrid Difference	1,821	74,192
K. Yamaoka, et al. [22, 43]	Image Segmentation and Pattern Matching	31,987	144,256
C. Johnston, et al. [11]	Image Segmentation based on Color Threshold	798	114,688
I. Bravo et al. [10]	Principal Component Analysis	8,450	737,280
K. Shimizu et al. [18]	Planer Motion Tracking using Matched Filter	30,140	124/144 (86%) Block RAMs
Y. Hsu et al. [24]	Color Detection and Binarization	19,085 (for HW part)	N/A
U. Ali et al. [16]	HW/SW co-design using Kernal Based	6,320	165,888

Table 5.4: Performance and Speed achieved by the implemented tracking systems on FPGA.

Algorithm	Frame Rate	System Speed
Adaptive hybrid Difference	32.55 fps	20 MHz
Image Segmentation and Pattern Matching [22, 43].	$13,745$ cycles + $10 \text{ cycles} \times N^2$ ( $N$ : no of objects)	20 MHz
Image Segmentation based on Color Threshold [11].	25 fps	27 MHz
Principal Component Analysis [10].	121 fps	100 MHz (Maximum 1,124 MHz)
Planer Motion Tracking using Matched Filter [18].	1545.44 fps	66 MHz
Color Détection and Binarization [24].	37.3 fps	25 MHz
HW/SW co-design using Kernal Based [16].	290 fps	50 MHz

Table 5.5: Frames size and Device types used with the implemented tracking systems on FPGA

Algorithm	Frame Size	FPGA Used Device
Adaptive hybrid Difference.	$640 \times 480$ pixels	Altera Cyclon II 2C35
Image Segmentation and Pattern Matching [22, 43].	$80 \times 60$ pixels	Altera Startix, EP1S60
Image Segmentation based on Color Threshold [11].	$768 \times 288$ pixels	Xilinx XC2S200
Principal Component Analysis [10].	$256 \times 256$ pixels	Xilinx XC2VP7
Planer Motion Tracking using Matched Filter [18].	$64 \times 64$ pixels	Xilinx Virtex Pro 6000
Color Detection and Binarization [24].	$640 \times 480$ pixels	Altera Cyclone II 2C35
HW/SW co-design using Kernal Based [16].	$64 \times 64$ pixels	Xilinx Spartan-3e XC3S1600E



## 5.9 System Evaluation

Our system implemented the Adaptive hybrid Difference algorithm using 1,821 logic elements which is 5% of the device resources and 74,192 bits of internal memory which is 15% of the memory on FPGA's capacity. Comparing with other systems, the logic utilization is the second smallest one after the work of C. Johnston, et al. [11] which uses 798 LEs. From another side, the work of C. Johnston uses 114,688 bits from the FPGA internal memory while our design uses 74,192 bits which is the smallest memory usage among all implementations of Table 5.3.

For the frequency, our system frequency has been adjusted to 20 MHz, which assures the synchronization between the memory, buffered FIFOs and other operations. A system with 20 MHz could be judged as a slow system; however the factor that has been considered in the AHD design was to meet the real time frame rate (30 fps), where the developed design has achieved 32.55 fps. Additionally, while developing a system on FPGA, the power dissipation is one of the considerable factors. Since the power dissipation increases with the increase of frequency, designing a system that works with low frequency is a desired one.

With the small frequency and small resources, the system has achieved 32.55 fps frame rate. Other systems, as shown in Table 5.4, have achieved higher frame rates in common while the used logic resources range between 798 LEs, 6,320 LEs, 8,450 LEs, 19,085 LEs, 30,140 LEs, and 31,987 LEs as shown in Table 5.3, which are high as compared to the results by the proposed system. Another important issue that should be considered while comparing between systems' throughput is the frame size. The frame size affects the number of pixels needed to be processed for each frame and thus the number of clock cycles that reflects on the frame rate. From tables 5.4 and 5.5, [18] and [22] have achieved very high frame rates while they used  $64 \times 64$  and  $80 \times 60$  pixels for the frame size respectively. This research work has used the smallest standard resolution listed by the TRDB\_D5 camera which is  $640 \times 480$  according to the TRDB\_D5's hardware specification manual [44]. By specifying a smaller frame size for this new system, for example, the frame rate can reach 2,441.4 fps for a frame of  $64 \times 64$  pixels and 2,083.3 fps for one of  $80 \times 60$  pixels at tracking

time. For calculating the adaptive threshold, it has generated 325.5 fps and 277.8 fps for frame size of  $64 \times 64$  and  $80 \times 60$  pixels respectively.

The basic difference between implementing the adaptive hybrid difference and the segmentation algorithms is implied by the order of processes' implementation among frames and pixels. For segmentation at running time, the processes are done on the current frame without usually needing data from previous frames. Some features such as pattern matching information or target object information may be extracted from previous frames and used while processing the next frames. This makes the use of extra hardware resources to build line buffers and parallel architectures to speed up the implementation is effective from timing perspective. Usually each pixel needs to use its' other neighbor pixels within the same frame. This makes the use of shift line buffers to keep and reuse the pixels values of the same frame very beneficial. Buffering pixels into line buffers allows the architecture to process more than one pixel at the same time, since the pixel and its neighbors are loaded from the memory. Thus algorithms with parallel implementation abilities can achieve high frame rate but at the same time consumes high resource usage. In contrast, Adaptive Hybrid Difference algorithm depends on processing pixels from previous frames while processing the current frame. The needed operations of AHD are simple, consisting of subtractions, additions and comparison operations, and thus the required logic elements are small. However, the difficulty comes with the need of accessing frames from different times to process each pixel. This had resulted in the need to wait until buffer FIFOs filled from the external RAM, since each FIFO is used to buffer frame data from the memory, which causes a delay time penalty that had to be paid.

To decide which is more important, speeding up the system or reducing the hardware resources, design area and cost, is like trying to catch three flying balls using two hands; only two balls at a moment could be caught. Our system is aimed to build a tracking system with optimal device resources, area and power achievable by a real time video surveillance system. Such optimal system has been achieved in this research.

## 5.10 Chapter Summary

This chapter has evaluated and discussed the performance of the hardware system which has been built to implement the Adaptive Hybrid Difference algorithm on an FPGA. The system has been evaluated in terms of the speed and hardware resources' cost. The system achieved throughput of 32.55 fps with a small frequency of 20MHz. An optimal resource of 1,821 logic elements and 74,192 bits of internal RAM have been used. 5% of device's area and cost are needed by this system design. By comparing with other FPGA based tracking systems done by others, with smaller frame sizes than that used by this proposed system, the throughput was higher while the resources utilization was higher too. In this chapter the research has answered the questions which were raised at the beginning of this thesis. This work can achieve an optimal resource, area and power dissipation by implementing a real time tracking system on an FPGA device and can work much faster than PCs implementing the same algorithm.

## CHAPTER 6

### THESIS SUMMARY

This research has introduced an optimal architecture design for moving object tracking system that depends on subtraction algorithm for real time environment. The design has been built to be compatible with FPGA devices' implementation and its limited resources. By using the Adaptive Hybrid Difference algorithm for tracking the moving objects, the proposed design has achieved frame rate of 32.55 fps with 1,821 LE from 33,216 of logic elements and 74,192-bits from 483,84-bits of the internal memory space. The device resource usage for the proposed design has achieved the lowest rate among other tracking systems on FPGA which have been stated by the literature review of this research. The frame rate of 32.55 fps is achieving the real time frame rate condition for video surveillance applications which is 30 fps.

The algorithm used detects the moving objects through the scene and represents them in a binary image with white color among black background. The design processes the data of the original image through two main units, Threshold Definer unit and Binary Image Builder unit. The Threshold Definer unit calculates the adaptive threshold for each pixel from 30 idle frames and stores the results in the SDRAM. At real time of tracking, the stored thresholds are used with the calculated hybrid difference values through the Binary Image Builder unit to generate the binary image. The algorithm's units and camera interfacing with the FPGA have been built using the HDL Verilog language and have been implemented on the Cyclone II FPGA device after synthesis using Quartus programmer.

The proposed system has achieved 32.55 fps using the smallest standard resolution offered by Terasic TRDB D5M camera,  $640 \times 480$  pixels frame size. To increase system speed, the frame resolution could be reduced. For example implementing frame of  $120 \times 160$  pixels on the same system can increase the

throughput to 520.8 fps. Using smaller resolution than the standard offered by the Terasic camera has been left as future work, since additional code is needed to be developed.

The single port SDRAM has restricted the prospect of using parallelization, since only 16 bits of data can be accessed within a single clock cycle. By depending on the fast speed of the SDRAM while reducing the processing units' speed, an allowable space of parallelism using buffers of FIFOs was feasible. Since the nature of AHD algorithm depends on accumulating the results among number of frames, all operations have to access the memory to take the new value or to accumulate the calculated results. Increasing the number of buffers to process more pixels in parallel, will let the memory fill all buffers with data at a slower speed than the unit would process them, and therefore the system synchronization will fail. Accordingly, system's frequency was reduced to 20 MHz to adapt dealing between the memory and six FIFOs at any moment of running time. However the new design has developed processing unit with pipelining and line buffer techniques that maximized the system frame rate with low frequency and low device resources.

In this thesis, a new design architecture of implementing AHD algorithm for object tracking on FPGA for the first time to reduce the logic element utilization under real time environment has been introduced. By optimizing the logic elements utilization, the power dissipation on the system is saved too since it depends on the number of active gates at any moment and the length of routes connecting the gates. By reducing the synthesized utilization the designed area on the chip is reduced too. The produced tracking system is optimized and can be used for more advanced surveillance applications that depend on object tracking as a basic step.

For future work we strongly recommend to continue improving the performance of this design while keeping the optimized logic utilization. We suggest to redesign the system with advanced memory device with dual port or with larger data bus width to increase system's throughput. Other suggestion is to reduce the frame resolution which is another factor that affects the performance as well. As a future work, additional techniques can be developed and added to predict the moving object location in the frame and thus applying operations only for a specific part instead of

the whole frame. This will add more logic elements but it will reduce the overall processed pixels and thus the number of operations per frame.

## REFERENCES

- [1] H. Kruegle, *CCTV surveillance: analog and digital video practices and technology*, Second ed.: Elsevier Butterworth Heinemann, 2006.
- [2] R. C. González and R. E. Woods, *Digital image processing*: Addison-Wesley, 1993.
- [3] Y. Ma and G. Qian, *Intelligent video surveillance: systems and technology*: Taylor & Francis, 2009.
- [4] S.-x. Shi, Q.-l. Zheng, and H. Huang, "A Fast Algorithm for Real-time Video Tracking," in *Intelligent Information Technology Application, Workshop on*, 2007, pp. 120-124.
- [5] S. Qureshi, *Embedded image processing on the TMS320C6000 DSP : examples in code composer studio and MATLAB*: Springer Science+Business Media, 2005.
- [6] W. J. MacLean, "An Evaluation of the Suitability of FPGAs for Embedded Vision Systems," in *Computer Vision and Pattern Recognition, 2005. CVPR. IEEE Computer Society Conference on*, 2005, pp. 131-131.
- [7] A. Corporation. *Altera Homepage* Available: <http://www.altera.com/>
- [8] X. Inc., "Xilinx Homepage."
- [9] S. Kilts, *Advanced FPGA design: architecture, implementation, and optimization*: Wiley, 2007.
- [10] I. Bravo, M. Mazo, J. L. Lázaro, A. Gardel, P. Jiménez, and D. Pizarro, "An Intelligent Architecture Based on Field Programmable Gate Arrays Designed to Detect Moving Objects by Using Principal Component Analysis," *Sensors 2010*, pp. 9232-9251, 2010.

- [11] C. Johnston, K. Gribbon, and D. Bailey, "FPGA based remote object tracking for real-time control.," *1<sup>st</sup> International Conference on Sencing Technology.*, pp. 66-71, 2005.
- [12] K. Huang, L. Wang, T. Tan, and S. Maybank, "A real-time object detecting and tracking system for outdoor night surveillance," *Pattern Recognition*, vol. 41, pp. 432-444, 2008.
- [13] T. Morimoto, O. Kiriya, Y. Harada, H. Adachi, T. Koide, and H. J. Mattausch, "Object tracking in video pictures based on image segmentation and pattern matching," in *Circuits and Systems, 2005. ISCAS 2005. IEEE International Symposium on*, 2005, pp. 3215-3218 Vol. 4.

---

- [14] N. Nagaoka, K. Okazaki, T. Sugahara, T. Koide, and H. J. Mattausch, "Grouping method based on feature matching for tracking and recognition of complex objects," in *Intelligent Signal Processing and Communications Systems, 2008. ISPACS 2008. International Symposium on*, 2009, pp. 1-4.
- [15] T. Morimoto, Y. Harada, T. Koide, and H. J. Mattausch, "Pixel-parallel digital CMOS implementation of image segmentation by region growing," *Circuits, Devices and Systems, IEE Proceedings -*, pp. 579-589, 2005.
- [16] U. Ali and M. B. Malik, "Hardware/software co-design of a real-time kernel based tracking system," *Journal of Systems Architecture*, vol. 56, pp. 317-326, 2010.
- [17] I. Ishii, Y. Nakabo, and M. Ishikawa, "Target tracking algorithm for 1 ms visual feedback system using massively parallel processing," in *Robotics and Automation, 1996. Proceedings., 1996 IEEE International Conference on*, 1996, pp. 2309-2314 vol.3.
- [18] S. Kazuhiro and H. Shinichi, "Implementing Planar Motion Tracking Algorithms on CMOS+FPGA Vision System," in *Intelligent Robots and Systems, 2006 IEEE/RSJ International Conference on*, 2006, pp. 1366-1371.



- [19] M. Meingast, O. Songhwai, and S. Sastry, "Automatic Camera Network Localization using Object Image Tracks," in *Computer Vision, 2007. ICCV 2007. IEEE 11th International Conference on*, 2007, pp. 1-8.
- [20] C. R. Wren, A. Azarbayejani, T. Darrell, and A. P. Pentland, "Pfinder: real-time tracking of the human body," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 19, pp. 780-785, 1997.
- [21] K. Yamaoka, T. Morimoto, H. Adachi, K. Awane, T. Koide, and H. J. Mattausch, "Multi-object tracking VLSI architecture using image-scan based region growing and feature matching," in *Circuits and Systems, 2006. ISCAS 2006. 2006 IEEE International Symposium on*, 2006, p. 4 pp.
- [22] K. Yamaoka, T. Morimoto, H. Adachi, T. Koide, and H. J. Mattausch, "Image segmentation and pattern matching based FPGA/ASIC implementation architecture of real-time object tracking," in *Design Automation, 2006. Asia and South Pacific Conference on*, 2006, p. 6 pp.
- [23] A. Yilmaz, O. Javed, and M. Shah, "Object tracking: A survey," *ACM Comput. Surv.*, vol. 38, 2006.
- [24] H. Yuan-Pao, M. Hsiao-Chun, and T. Ching-Chih, "FPGA implementation of a real-time image tracking system," in *SICE Annual Conference*, 2010, pp. 2878-2884.
- [25] H. MORAVEC, "Visual mapping by a robot rover," *The International Joint Conference on Artificial Intelligence (IJCAI)*, vol. 1, pp. 598–600, 1979.
- [26] C. HARRIS and M. STEPHENS, "A combined corner and edge detector," *Alvey Vision Conference*, pp. 147–151, 1988.
- [27] K. MIKOLAJCZYK and C. SCHMID, "An affine invariant interest point detector," *European Conference on Computer Vision (ECCV)*, vol. 1, pp. 128–142, 2002.

- [28] D. LOWE, "Distinctive image features from scale-invariant keypoints," *Int. J. Comput. Vision*, vol. 2, pp. 91–110, 2004.
- [29] K. Mikolajczyk and C. Schmid, "A performance evaluation of local descriptors," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 27, pp. 1615-1630, 2005.
- [30] R. Jain and H. H. Nagel, "On the Analysis of Accumulative Difference Pictures from Image Sequences of Real World Scenes," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. PAMI-1, pp. 206-214, 1979.
- [31] K. Toyama, J. Krumm, B. Brumitt, and B. Meyers, "Wallflower: principles and practice of background maintenance," in *Computer Vision, 1999. The Proceedings of the Seventh IEEE International Conference on*, 1999, pp. 255-261 vol.1.
- [32] N. M. Oliver, B. Rosario, and A. P. Pentland, "A Bayesian computer vision system for modeling human interactions," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 22, pp. 831-843, 2000.
- [33] A. Monnet, A. Mittal, N. Paragios, and R. Visvanathan, "Background modeling and subtraction of dynamic scenes," in *Computer Vision, 2003. Ninth IEEE International Conference on*, 2003, pp. 1305-1312 vol.2.
- [34] Z. Zivkovic, "Improved adaptive Gaussian mixture model for background subtraction," in *Pattern Recognition, 2004. ICPR 2004. Proceedings of the 17th International Conference on*, 2004, pp. 28-31 Vol.2.
- [35] A. Mittal and N. Paragios, "Motion-based background subtraction using adaptive kernel density estimation," in *Computer Vision and Pattern Recognition, 2004. CVPR 2004. Proceedings of the 2004 IEEE Computer Society Conference on*, 2004, pp. II-302-II-309 Vol.2.
- [36] R. Cucchiara, M. Piccardi, and R. Emilia, "Vehicle detection under day and night illumination," *Proceedings of the Third International Symposium on*

*Intelligent Industrial Automation and Soft Computing*, pp. 789-794, 1999.

- [37] T. Papadimitriou, K. I. Diamantaras, M. G. Strintzis, and M. Roumeliotis, "Video scene segmentation using spatial contours and 3-D robust motion estimation," *Circuits and Systems for Video Technology, IEEE Transactions on*, vol. 14, pp. 485 - 497, 2004.
- [38] B. Jähne, *Digital image processing*: Springer, 2005.
- [39] T. Bräunl, *Parallel image processing*: Springer, 2001.
- [40] T. Joachims, "Transductive Inference for Text Classification using Support Vector Machines," presented at the Proceedings of the Sixteenth International Conference on Machine Learning, 1999.
- [41] H. A. Rowley, S. Baluja, and T. Kanade, "Neural Network-Based Face Detection," *Pattern Analysis and Machine Intelligence, IEEE Transactions, Mach. Intell.*, vol. 20, pp. 23-38, 1998.
- [42] K. Tieu and P. Viola, "Boosting Image Retrieval," *Int. J. Comput. Vision*, vol. 56, pp. 17-36, 2004.
- [43] T. Morimoto, H. Adachi, K. Yamaoka, K. Awane, T. Koide, and H. J. Mattausch, "An FPGA-Based Region-Growing Video Segmentation System with Boundary-Scan-Only LSI Architecture," in *Circuits and Systems, 2006. APCCAS 2006. IEEE Asia Pacific Conference on*, 2006, pp. 944-947.
- [44] Terasic. (2008). *THDB D5M Terasic Hardware Specification*. Available: [http://www.cs.washington.edu/education/courses/cse467/08au/labs/Resources/THDB-D5M\\_Hardware%20specification.pdf](http://www.cs.washington.edu/education/courses/cse467/08au/labs/Resources/THDB-D5M_Hardware%20specification.pdf)
- [45] Altera, "Cyclone II Device Handbook, Volume 1.", CA 95134, 2012.
- [46] B. Waggoner, *Compression for great digital video: power tips, techniques, and common sense*: Elsevier Science & Technology, 2002.

- [47] E. M. Schwalb, *iTV handbook: technologies and standards*: Prentice Hall PTR, 2003.
- [48] D. A. Karp, *Windows 98 annoyances*: O'Reilly, 1998.
- [49] S. Johnson, *Stephen Johnson on digital photography*: O'Reilly, 2006.
- [50] N. H. Tan, "Minimize Resource Usage for a Real-Time Depth Computational Module on FPGA," M. Eng. thesis, Universiti Teknologi Petronas, Perak, Malaysia, Apr. 2011.
- [51] P. R. Wilson, *Design recipes for FPGAs*: Newnes, 2007.
- [52] P. P. Chu, *FPGA Prototyping By Verilog Examples: Xilinx Spartan-3 Version*: John Wiley & Sons, 2011.
- 
- [53] S. Golson, "State machine design techniques for Verilog and VHDL," *Synopsys Journal of High Level Design*, pp. 1–48, September 1994 1994.
- [54] O. Cadenas and G. Megson, "A clocking technique for FPGA pipelined designs," *J. Syst. Archit.*, vol. 50, pp. 687-696, 2004.
- [55] N. I. Corporation. (2009). *Pipelining to Optimize FPGA VIs (FPGA Module) (2009 ed.)*. Available: [http://zone.ni.com/reference/en-XX/help/371599E-01/lvfpgaconcepts/fpga\\_pipelining/](http://zone.ni.com/reference/en-XX/help/371599E-01/lvfpgaconcepts/fpga_pipelining/)
- [56] Xilinx, "Spartan-II FPGA Family Data Sheet," June 13, 2008 2008.
- [57] Altera, "DE2 Development and Education Board - User Manual," 2007.

## LIST OF PUBLICATIONS

- L. N. Elkhatab, F. A. Hussin, and P. Sebastian, "Camera Sensor network localization using FPGA," in *International Conference on Intelligent and Advanced Systems (ICIAS), 2010*, Kuala Lumpur, Malaysia, 2010, pp. 1-6.
- L. N. Elkhatab, F. A. Hussin, L. Xia, and P. Sebastian, "An Optimal Design of Moving Objects Tracking Algorithm on FPGA," in *International Conference on Intelligent and Advanced Systems (ICIAS), 2012*, Kuala Lumpur, Malaysia, In press.

## APPENDIX A

### FEATURES AND DESIGN CHARACTERISTICS OF THE DE2 BOARD

---

**The FPGA Device**

The FPGA device Cyclone II 2C35 FPGA contains 33,216 logic elements (LEs). For running operation and storing by Cyclone II, it has 105 M4K blocks of internal RAM. To get more options for timing that controls the design, 4 phase-locked loops (PLLs) are integrated. Additionally Cyclone contains 35 embedded multipliers and 475 user I/O pins [57].

Some external devices are integrated on Altera DEII board and connected through the Cyclone FPGA device, so interfacing and dealing with them is reliable, as shown in Figure 1 below.

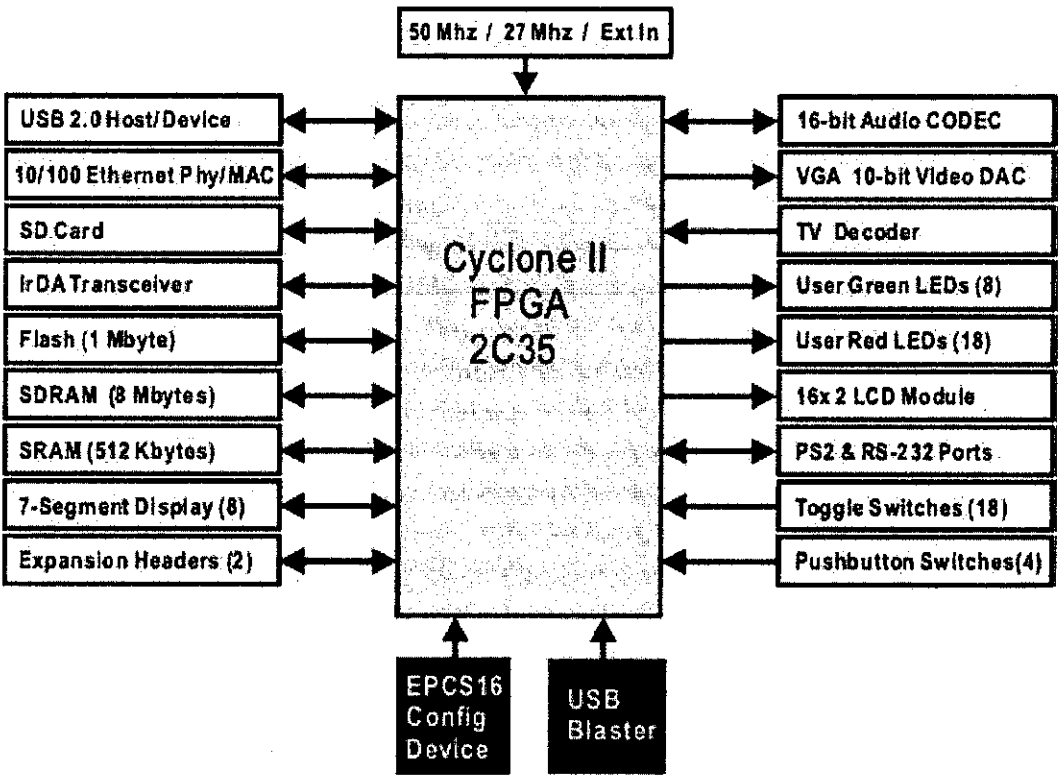


Figure 1: Block diagram of FPGA and other devices on DEII board [57].

**VGA output**

The VGA output supported by the DEII board includes a 16-pin D-SUB connector that is used to transit the synchronization signals must be provided from the Cyclone

II FPGA and by Analog Devices ADV7123 triple 10-bit high-speed video DAC. The ADV7123 device is integrated on the board and connected with the VGA output and it is used to produce the analog data signals (red, green, and blue). The VGA output with the ADV7123 can support resolutions of up to  $1600 \times 1200$  pixels at 100MHz.

### **Expansion Header**

There are two expansion headers; this research has used Expansion Header 1 (JP1). Each expansion header contains 40 pins which work as I/O pins of the Cyclone II. It is used by the system design to interface the TRDB\_5M camera, which is designed by Terasic to be compatible with the DEII board's expansion header.

---

### **SDRAM**

A 54-Pin TOSP-2(IS42S16400) SDRAM is provided by the board which is an 8-Mbyte Single Data Rate Synchronous Dynamic RAM memory chip which organized as  $1M \times 16 \text{ bits} \times 4 \text{ banks}$ .

### **Clock inputs**

The board supports these choices of frequencies:

- 50-MHz oscillator
- 27-MHz oscillator
- SMA external clock input

An extra option is offered by using the Cyclone's integrated PLLs to generate other frequencies. This research has used the standard 50 MHz to generate 25 MHz for the main system clock and 125 MHz for the SDRAM using PLLs.



APPENDIX B

K VALUE TESTING

## k Value Testing

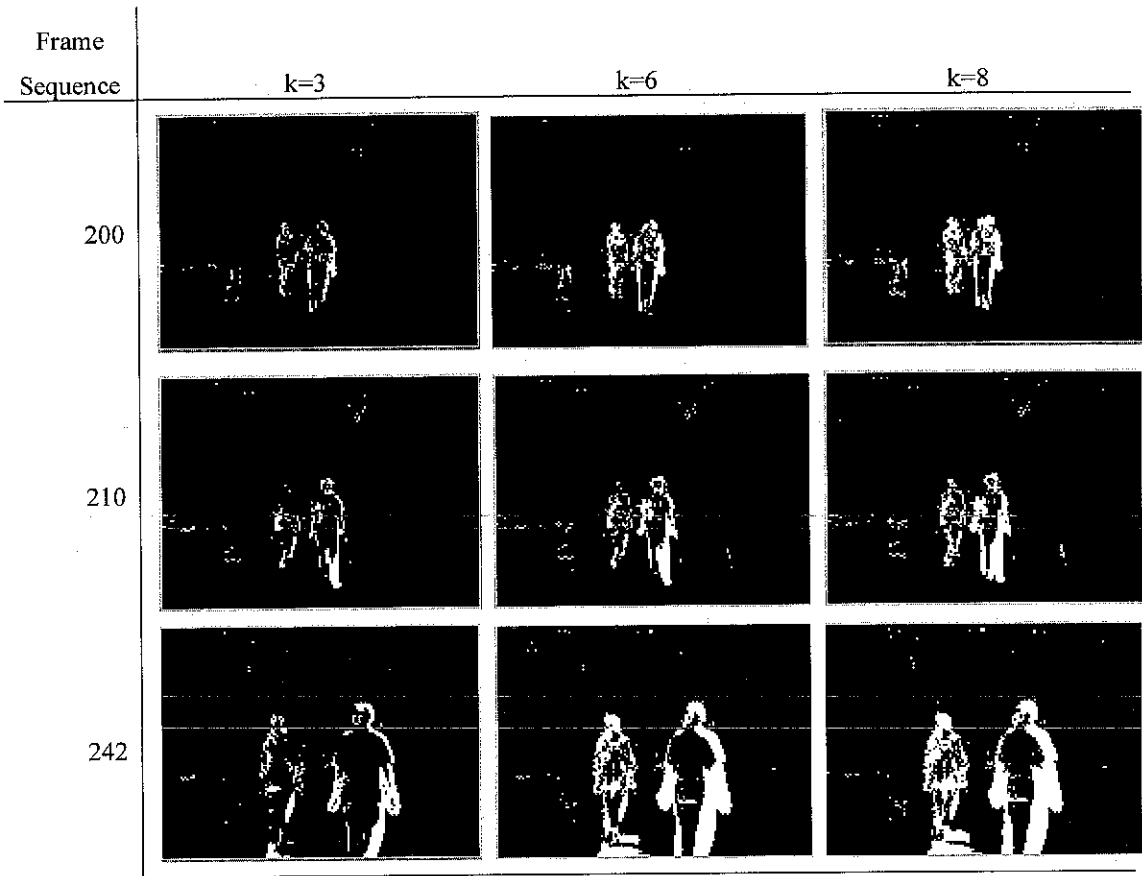


Figure 0.1: k value testing

In Figure 1, k values with different settings have been tested to determine the most accurate value. When k is set to 3, it gives low tracking quality. With k=8, the obtained image contains wrong detected pixels that make the noise around the objects. The best value of k from the experiments is 6, as it gave more accurate results than k=3 and less noise than the one obtained with k=8.