

# **Object tracking on XMOS multi-threaded environment**

*A Project Report*

*submitted by*

**SHYAM KRISH K S**

**(EE08B030)**

*in partial fulfilment of the requirements  
for the award of the degree of*

**BACHELOR OF TECHNOLOGY AND MASTER OF TECHNOLOGY**

*in*

*Microelectronics and VLSI Design*



**DEPARTMENT OF ELECTRICAL ENGINEERING  
INDIAN INSTITUTE OF TECHNOLOGY MADRAS**

**May 2013**

# THESIS CERTIFICATE

This is to certify that the thesis titled **Object tracking on XMOS multi-threaded environment**, submitted by **Shyam Krish.K.S**, to the Indian Institute of Technology, Madras, for the award of the degree of **Bachelor of Technology and Master of Technology** in *Microelectronics and VLSI design*, is a bona fide record of the research work done by him under our supervision. The contents of this thesis, in full or in parts, have not been submitted to any other Institute or University for the award of any degree or diploma.

**Prof. Sridharan K**  
Guide  
Professor  
Dept. of Electrical Engineering  
IIT-Madras, 600 036

**Dr.Sudha Natarajan**  
External Guide  
XMOS Semiconductor Pvt. Ltd  
IITM  
Research park, 600 115

Place: Chennai

Date: 15<sup>th</sup> May 2013

## ACKNOWLEDGEMENTS

I am thankful to **Dr. Sudha Natarajan** for her guidance throughout the project. She helped me a lot in directing me to right materials and research papers whenever I was stuck with doubts. I am also thankful to **Dr. K. Sridharan** for his valuable discussions that helped me stay focussed in my project.

And I thank all my friends who were there all along my project. If not for them, this project might not have been complete. Even if it is not their project, they always helped me by discussing various concepts necessary for the project.

Thanks to XMOS pvt ltd for providing me required boards for my project. And thanks to IITM for providing me the best campus and lab environment. Finally thanks to my Mom for her constant care and support.

# ABSTRACT

**KEYWORDS:** Object Tracking, Parallel System, Multithreading

In this thesis a new system is developed for tracking an object in a video frame in a muticore multithreading environment. Memory storage, object tracking algorithm and displaying the output in Liquid Crystal Display(LCD) all run in different threads simultaneously. The developed system will serve as a test platform for testing any other object tracking algorithm. The implemented algorithm evolved through comparing different object tracking algorithms in Matlab.

The object tracking system uses 5 threads that run in parallel while 1 thread is used initially to load the images into SDRAM in RGB565 format. We have achieved a frame rate of  $9\text{frames/s}$  for an image of size  $480 \times 272$ . In addition, we also did experiments on memory and timing analysis of a multithreaded platform.



# TABLE OF CONTENTS

<b>ACKNOWLEDGEMENTS</b>	<b>i</b>
<b>ABSTRACT</b>	<b>ii</b>
<b>LIST OF TABLES</b>	<b>vi</b>
<b>LIST OF FIGURES</b>	<b>viii</b>
<b>ABBREVIATIONS</b>	<b>ix</b>
<b>1 INTRODUCTION</b>	<b>1</b>
1.1 Motivation . . . . .	1
1.2 Object tracking and Multithreading . . . . .	1
1.3 Literature survey . . . . .	2
1.4 Contributions of the project . . . . .	3
1.5 Organisation of the report . . . . .	3
<b>2 Object tracking and Matlab implementation</b>	<b>5</b>
2.1 Object tracking . . . . .	5
2.1.1 Problems in object tracking . . . . .	6
2.1.2 Otsu's threshold . . . . .	7
2.1.3 Connected Component Analysis (CCA) . . . . .	8
2.2 Matlab Implementation . . . . .	9
2.2.1 Background subtraction . . . . .	10
2.2.2 Frame subtraction . . . . .	11
2.2.3 Histogram of Oriented Gradients (HOG) . . . . .	15
2.3 Comparison . . . . .	19
<b>3 Object tracking system development in multithreaded environment</b>	<b>21</b>

3.1	Algorithm . . . . .	21
3.1.1	Labeling is not necessary for tracking . . . . .	21
3.1.2	CCA and Optimized CCA . . . . .	23
3.1.3	Algorithm . . . . .	23
3.1.4	Translation . . . . .	26
3.1.5	Merging . . . . .	27
3.1.6	Inner loop . . . . .	29
3.2	Implementation . . . . .	30
3.2.1	Image format . . . . .	31
3.2.2	Object tracking system . . . . .	31
<b>4</b>	<b>Experimental setup and Results</b>	<b>34</b>
4.1	Experimental setup . . . . .	34
4.1.1	Flash . . . . .	34
4.1.2	SDRAM . . . . .	38
4.1.3	LCD . . . . .	39
4.1.4	Video Display . . . . .	42
4.1.5	Slice Kit . . . . .	44
4.2	Experimental studies and results . . . . .	45
4.2.1	Experimental studies . . . . .	45
4.2.2	Resource utilization . . . . .	46
4.2.3	Results . . . . .	46
<b>5</b>	<b>Analysis of memory and timing</b>	<b>51</b>
5.1	Memory analysis . . . . .	51
5.2	Timing analysis . . . . .	52
<b>6</b>	<b>Conclusion and Future work</b>	<b>66</b>
6.1	Conclusion . . . . .	66
6.2	Future work . . . . .	66
<b>A</b>	<b>APPENDIX</b>	<b>67</b>
A.1	Introduction to XC programming . . . . .	67

A.2	Matlab codes . . . . .	69
A.2.1	Background subtraction . . . . .	69
A.2.2	Frame subtraction . . . . .	69
A.2.3	Main code for HOG tracking . . . . .	73
A.2.4	HOG - function . . . . .	75
A.2.5	Feature - function . . . . .	77
A.2.6	Detect Sample Image - function . . . . .	78
A.2.7	Predict - function . . . . .	80
A.3	XC codes . . . . .	81
A.3.1	XC-1A LED - Example code . . . . .	83
A.3.2	Video Display . . . . .	86
A.3.3	Frame subtraction - main function . . . . .	90
A.3.4	Binarization, cca call and put bounding box . . . . .	92
A.3.5	Connected Component Analysis . . . . .	100

## LIST OF TABLES

2.1	Difference image in Background subtraction and Frame subtraction	13
2.2	Variation of difference image with parameter lamda . . . . .	14
2.3	Feature selection . . . . .	18
2.4	Comparing object tracking algorithms . . . . .	20
3.1	Input and Output of CCA . . . . .	25
3.2	Merging and Translation depending on the neighbors . . . . .	29
4.1	Buffer array as stored in sdram . . . . .	39
5.1	Memory required for different datatypes . . . . .	51
5.2	Memory analysis table I . . . . .	53
5.3	Memory analysis table II . . . . .	54
5.4	Memory analysis table III . . . . .	55
5.5	Timing analysis table I . . . . .	56
5.6	Timing analysis table II . . . . .	57
5.7	Timing analysis table III . . . . .	58
5.8	Timing analysis table IV . . . . .	59
5.9	Timing analysis table V . . . . .	60
5.10	Timing analysis table VI . . . . .	61
5.11	Timing analysis table VII . . . . .	62
5.12	Timing analysis table VIII . . . . .	63

## LIST OF FIGURES

2.1	Histogram of an image . . . . .	8
2.2	Connected components . . . . .	9
2.3	Flow chart of Background subtraction . . . . .	10
2.4	Noise removal . . . . .	11
2.5	Sample processing . . . . .	12
2.6	Flow chart of Frame subtraction . . . . .	13
2.7	Extracting HOG feature . . . . .	16
2.8	Histogram binning and interpolation . . . . .	17
3.1	Flow chart of object tracking application . . . . .	22
3.2	Noise affects label's uniqueness . . . . .	22
3.3	Worst case noise . . . . .	24
3.4	Labeling . . . . .	24
3.5	Flow chart for issuing label . . . . .	25
3.6	Translation . . . . .	27
3.7	Merging . . . . .	28
3.8	Inner loop . . . . .	30
3.9	Image format . . . . .	31
3.10	Thread diagram . . . . .	32
4.1	SPI specification file for FLASH Numonyx M25P10-A . . . . .	35
4.2	Output of libflash code . . . . .	36
4.3	FLASH partition . . . . .	38
4.4	SDRAM server . . . . .	39
4.5	LCD server . . . . .	40
4.6	Processing speed versus number of parallel threads . . . . .	42
4.7	Video display . . . . .	43
4.8	Experimental setup . . . . .	44

4.9	Slice kit port map . . . . .	45
4.10	Resource utilization . . . . .	47
4.11	Bounding box has been put over the moving object . . . . .	48
4.12	Bounding box has been put over the moving object . . . . .	49
4.13	Bounding box has been put over the moving object . . . . .	50
5.1	Flash, CCA and Otsu's threshold - Parallel processing . . . . .	64
A.1	Timer operation . . . . .	68
A.2	XC-1A port map . . . . .	82
A.3	Cyclic LED control sequence . . . . .	82
A.4	par usage in cyclic led code . . . . .	83

## **ABBREVIATIONS**

<b>HOG</b>	Histogram of Oriented Gradient
<b>CCA</b>	Connected Component Analysis
<b>XTA</b>	XMOS Timing Analyzer

# CHAPTER 1

## INTRODUCTION

### 1.1 Motivation

Mobile and portable platforms increasingly require the ability to handle images and videos smoothly. Image data is large in size and mobile devices can afford to have only a chip or two to perform the processing. It is therefore important to study computing devices that support parallel processing so that applications run in real-time. Some of the contemporary computing solutions are based on Field Programmable Gate Arrays (FPGAs) and processors that support multi-threading. This project examines the power of the latter. In particular, the project examines the use of multi-threaded and multi-core processors from XMOS for performing a typical image processing task, namely object tracking, at high speed. We briefly review the basics of object tracking as well as multithreading and then proceed to state the precise contributions of this project.

### 1.2 Object tracking and Multithreading

Detecting a particular object in each and every frame of the video is known as object tracking. Almost any existing object tracking algorithm can track an object, if the image sequences are noise free but the amount of resources it consumes may be huge. In addition, real time performance is another constraint which mainly depends on the algorithm used for tracking.

Real time operation can be accomplished via parallel processing or multithreading. A thread is part of a program which can run independently. Any process or an application program can be split into independent smaller codes which can be made to run as a thread. Multithreading CPUs have hardware support to run each thread concurrently. It can be compared with multiprocessing. The main difference between multithreading



and multiprocessing is that, error in one thread can bring down all the threads in a process whereas an error in one process, can not bring down another process.

For instance, if video compression and object tracking are two different processes, failure in video compression may not affect object tracking process and vice versa. In object tracking process, we have memory interface, algorithmic unit and LCD display for the output as independent units of the code. Independent here refers to the ability to fetch next image data and store it in memory while processing the current image. However, failure in storing the next image data in memory by a memory interface thread can affect other subsequent threads in the object tracking process.

### 1.3 Literature survey

Even though implementing parallel object tracking application is the main objective of the project, the project involved thorough learning about the basic image processing concepts and sequential object tracking for moving on to parallel coding. The textbook by Gonzalez and Eddins (2004) provides all the concepts required for the basics of image processing. Otsu's algorithm is one of the main concepts that has been studied and implemented from Gonzalez and Eddins (2004). Real time object tracking implementation in Liu *et al.* (2011) involves doing a background subtraction in FPGA where the search path to locate the current position of the object has been parallelized. In Mahmoud *et al.* (2003), the authors implement a block matching algorithm in parallel. A semi-systolic array of non-programmable processor elements has been used to parallelize the whole application. A non-programmable processor is equivalent to a thread in the XMOS platform.

One of the main components of any image processing application is Connected Component Analysis (CCA). Connected component analysis helps in retrieving area, centroid and bounding box information from a binary image. Holding an image in a buffer and performing CCA is a much easier problem. It becomes really complicated when memory available for such buffers is limited. Single pass connected component analysis Johnston and Bailey (2008) gives good idea of CCA. Optimized single pass connected component analysis Ma *et al.* (2008) reduces memory consumption even further com-

promising on giving unique labels to different disconnected components. Using labels obtained from CCA output is in general not advisable for object tracking as random noise will disturb the labeling process. Hence Optimized CCA is more useful than the normal CCA. Multiple objects tracking based on frame subtraction Yang *et al.* (2005) needs memory for storing two images (current and previous frame). Histogram of Oriented Gradient (HOG) descriptor is used for human detection Dalal and Triggs (2005). Finally, Pulli *et al.* (2012) gives good perspective on what parts of the image processing can be parallelized.

## 1.4 Contributions of the project

The contributions of the project are as follows.

1. Detailed study of various object tracking methods in the literature as well as the X MOS multi-threaded computing environment.
2. Simulation of some sub-algorithms for object tracking in Matlab.
3. Design of an object tracking system that uses several threads.
4. Design of storage of image sequences carefully to facilitate streaming and processing using different threads
5. Implementation of some sub-algorithms for object tracking (especially frame subtraction) on an X MOS multi-threaded processor and testing of the sub-algorithms.
6. Analysis of memory and time management in X MOS processors.

## 1.5 Organisation of the report

The remaining of this report has been organized as follows.

**Chapter 2** explains the basics of object tracking in detail. It explains different object tracking algorithms coded in Matlab. At the end comparison between algorithms is given.

**Chapter 3** gives an introduction to experimental setup. It explains the parallel processor we used in this project and also gives an idea of how the set up has been developed.

**Chapter 4** gives the memory and timing analysis. It also has an example code built

using the concepts developed in the analysis.

**Chapter 5** gives the detailed explanation of the implemented algorithm. It also gives explanation of the code written, resources consumed and results.

**Appendix** gives an introduction to XC programming language, Matlab codes for all the implemented algorithms and also XC codes.

## CHAPTER 2

### Object tracking and Matlab implementation

In this chapter we have discussed the basics of object tracking and matlab implementation of sub algorithms of object tracking. We have explained the sub algorithms in details and have compared the algorithms at the end of this chapter.

#### 2.1 Object tracking

The main aim of any object tracking algorithm is to provide the trajectory of an object over time by locating its position every frame of the video. In a particular object tracking algorithm, there are various sub modules that needs to understood to get an over all understanding of the tracker. There are several different algorithms for object tracking. In this project report, we will take two major algorithms and explain them in detail. Depending on whether the background is static or dynamic, we have two different algorithms to deal with tracking an object. Background being static means, there should not be any illumination changes, there should not be any moving leaves due to wind or there should not be any waves or ripples in pond, and there should not be any moving clouds. So the environment is much tighter. Artificial lighting environment like inside an office may have an ideal static background.

Gaussian mixture models, Eigen model etc... are used when background remains static. In Gaussian mixture models and Eigen models, the static background is modeled using a set of video frames. After 10 to 20 frames, objects can be tracked with ease using these background models. These background modeling algorithms handles changes in the background to some extent as it always adapts itself after every 10 to 20 frames. Every frame is subtracted from the background model to get a difference image. Using this difference image, all the foreground objects can be identified. From this foreground we can detect the required object using some other algorithms and hence the object can be tracked.

In simple background subtraction, we will usually have the background of the video in advance. For example, inside an office, background can be an image (when nobody is present inside) with all lights switched on. There is another method called Frame subtraction, which is much sophisticated than Background subtraction as the variation in the background can be tolerated to some extent.

In cases where background is not static we need to follow completely different approach. An example for non-static background is tracking a car in a highway. Here we need to describe the object based on certain sharp features. These features should be such that, we can differentiate object with that of the background. It is sufficient if we are able to “Detect” the object in every frame. In order to detect the moving car, you can choose color, size, model etc... But the chosen feature can easily match with another car that is passing closer to the car which needs to be tracked. So much sharper features are necessary while tracking cars in a highway. In order to detect clouds, we can take features such as color (blue) and image derivative to get the boundary between cloud and the sky. So with this two simple features, we can detect clouds. But In order to detect a snake, we need to have even more features as snakes are of different colors and size. The features chosen for snake can easily be confused with worms if we have missed any feature that distinguishes worms and snakes. So “Object Representation” or “Feature Selection” is very important for detecting an object. A set of features or “Object Descriptor” is similar to DNA or finger print which helps differentiate one object from other.

From the discussion above, we can see that each object needs special set of features or “Object Descriptor” in order to detect them. Histogram of Oriented Gradients is one such object descriptor used mainly for human detection. With added features to HOG, it is possible for human identification but it is much different algorithm than detection. HOG based tracker will be explained later in the report. The following picture is an example for object tracking.

### **2.1.1 Problems in object tracking**

Real time speed requirements, illumination changes, object shape changes, noise in images and occlusion are few major problems when dealing with object tracking. While

we cannot remove noise which depends on the sensor used inside the camera we can deal with other issues to some extent. Handling occlusion is one of the major issue in descriptor based tracking. There are two different occlusions, partial and complete occlusion. To deal with occlusion, machine learning (or statistical) concepts can be used rather than simple algorithms. For example, if we want to track a particular book and we are able to track it using descriptor based object tracking. Certain unique features or descriptor is extracted from the book and used for detecting it in every frame. Now if some other book partially starts occluding, then the descriptor gets changed. We have two set of descriptors representing the same book. When the book comes out of occlusion, the algorithm might start tracking the other book which occluded our original book. These kinds of issues usually come up while doing experiments with different set of videos.

In frame subtraction method, we will essentially increase noise while subtracting two different frames. Hence the path of any object will be predicted with greater noise. If the object is moving in a complex track, the predicted track may not be smooth. Kalman filter can be used to reduce such noise and smooth track can be obtained in some cases.

### **2.1.2 Otsu's threshold**

Thresholding is an important step in image segmentation. In object tracking if we can segment an image into background and foreground, tracking the object becomes much simpler. For example, say we need to track a fish in a clear pond (without any leaves or bottom being visible) and also the fish is fully black in color. In such a case, there will two different peaks in image histogram as shown in figure 2.1. Thresholding with a value in the middle of the two peaks (around 125) will allow us to locate the fish in the image. In cases where there are only two peaks, threshold value can be obtained using Otsu's thresholding. In frame subtraction application, the difference image obtained in one of the step will have to be thresholded to obtain a binary image. In that case, a normal constant thresholding gives much better results than Otsu's threshold. This has been verified with lots of experiments with different videos. Otsu's threshold has been implemented and it will be explained in later section.

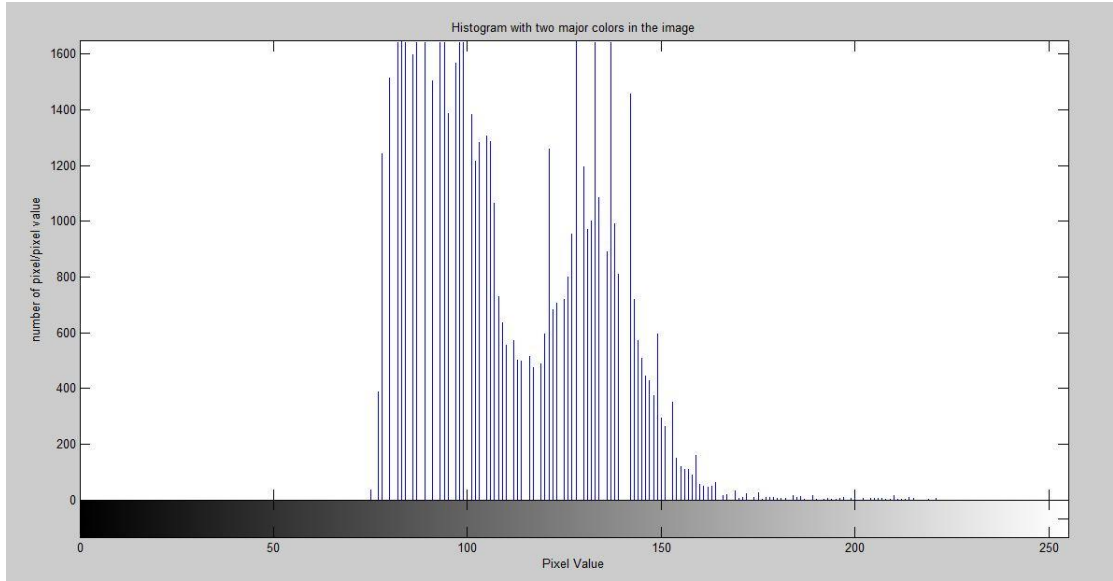


Figure 2.1: Histogram of an image

For detailed explanation of thresholding, the reader is referred to Gonzalez and Eddins (2004). Dilation and erosion, opening and closing are basic image processing steps which are discussed in detail in Gonzalez and Eddins (2004). These help in analyzing blobs while tracking an object. Selecting what type of thresholding will suit, dilation or erosion etc... is essential while dealing with frame subtraction algorithm.

### 2.1.3 Connected Component Analysis (CCA)

Connected component analysis is the main sub-module in frame subtraction algorithm. After thresholding the difference image we obtain a binary image. This binary image has blobs which have to be analyzed to obtain foreground object's location in image, area and bounding box. In the figure 2.2 there are three complex objects and they have to be labeled as shown. CCA helps us to obtain area, bounding box and centroid information from the binary image. Optimized CCA implemented will be discussed in detail later. Optimized CCA retrieves the features (area, bounding box etc) in a single pass without giving any labels.

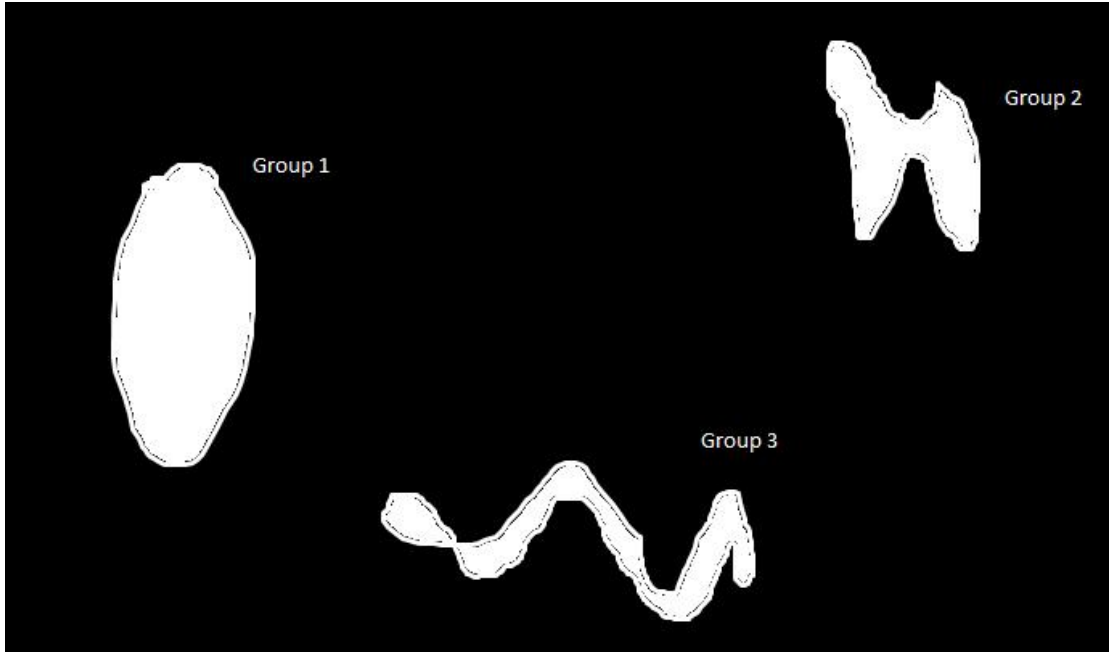


Figure 2.2: Connected components

## 2.2 Matlab Implementation

Object tracking being an important application, Matlab has lots of supporting modules necessary for developing an object tracking algorithm. All the basic steps described in introduction such as Otsu's threshold, image closing and opening, dilation and erosion and connected component analysis (CCA) are available as functions in Matlab. These basic image processing operations help in analyzing a blob obtained in frame subtraction and background subtraction algorithms. Experiments were performed to obtain a good threshold option resulting in clean binary image.

Demos available in Matlab give a lot of insight for a beginner in object tracking. Gaussian Mixture model based object tracking demo in Matlab shows how the background is modeled using GMM. After going through couple of demos in Matlab, a new code has been developed using few in built functions in Matlab. "Frame subtraction" based object tracking (Code is available in the appendix) has been implemented. This basic object tracking algorithm works only if the background is static. Background should not have illumination changes and moving objects.



### 2.2.1 Background subtraction

The algorithm involves subtraction of current frame from the background. First frame in the video is chosen as background for the entire video footage. The figure 2.3 of the entire algorithm.

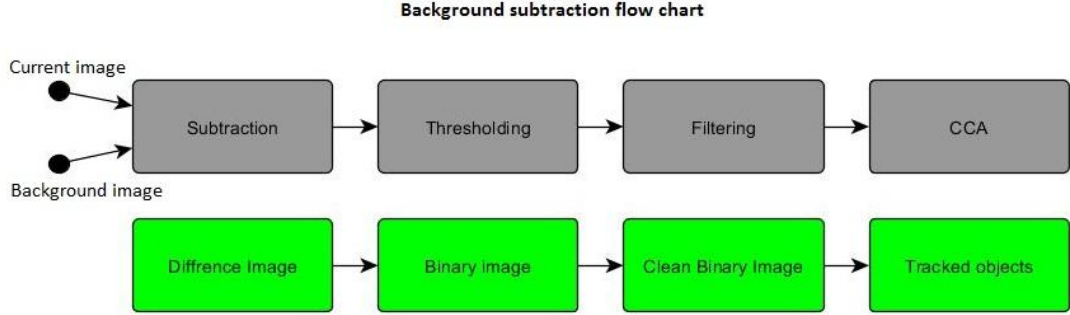


Figure 2.3: Flow chart of Background subtraction

Once we obtain the difference image, it has to go through thresholding to obtain a binary image. This binary image looks similar to the one shown in figure 2.1. Threshold is fixed for a particular video clip. It is easy to identify an ant in white background rather than a black background. It will also vary if you resize the images in the video. So choosing the right threshold is empirical.

After obtaining the binary image, REGIONPROPS function in Matlab helps us to retrieve area, bounding box and centroid information from the blobs. If there is lot of noise in the binary image sequence, the binary image is then passed through image filters like median filter followed by erosion, dilation, closing or opening function as shown in figure 2.4. While tracking a car using background subtraction, sometimes there is no connection between car's front portion and the top (viewing from top as shown in later image sequences). This disconnection may be due to the threshold value chosen and also due to the wind screen present between them. The windscreen generally has low pixel values and hence a threshold greater than these values eliminates this to be the part of the car misleading us to think it as a background. So it is better to perform closing operation.

The features obtained using regionprops function (connected component analysis), helps us to track single object. It can also track multiple objects if they are well sep-

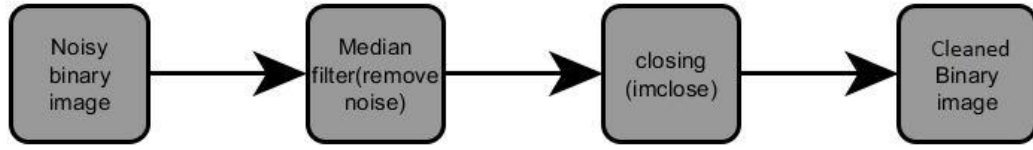


Figure 2.4: Noise removal

arated in the image sequence. Let us consider that we are tracking two cars in the available video frame. From the  $n^{th}$  binary image, we will obtain the location of both the cars and we put green bounding box on one of them and a red bounding box on the other. In the  $(n + 1)^{th}$  frame we will again get two locations (more than two will be shown due to noise, but we need to consider only those whose area exceeds certain threshold - areathresh in frame subtraction code). After eliminating the features whose area component is above area threshold or choosing only two features (since we are tracking only two cars) whose area component is highest, you can do a closest neighbor search from the previous centroid location. Hence the location which is closest to green box in previous frame will get a green bounding box and so does the red bounding box. A sample processing of background subtraction is shown in figure 2.5.

### 2.2.2 Frame subtraction

Frame subtraction is an improvement made on the background subtraction where we subtract the  $n^{th}$  frame from  $(n - k)^{th}$  frame instead of the  $1^{st}$  frame. It works better than background subtraction as small movements in the background such as moving clouds, leaves and ripples in the pond etc can be handled well. Background model has to be updated in each and every frame to handle such dynamics in the background.

The code implemented in Matlab is based on Yang *et al.* (2005). In the code that we developed based on the paper does not involve background modeling as in the paper and also does not handle occlusion. Dynamic matrix is the main concept that has been implemented from Yang *et al.* (2005). The importance of dynamic matrix will be understood after understanding the difference image obtained in frame subtraction algorithm (code for frame subtraction is available in the appendix). Flow chart for frame subtraction is given in the figure 2.6.

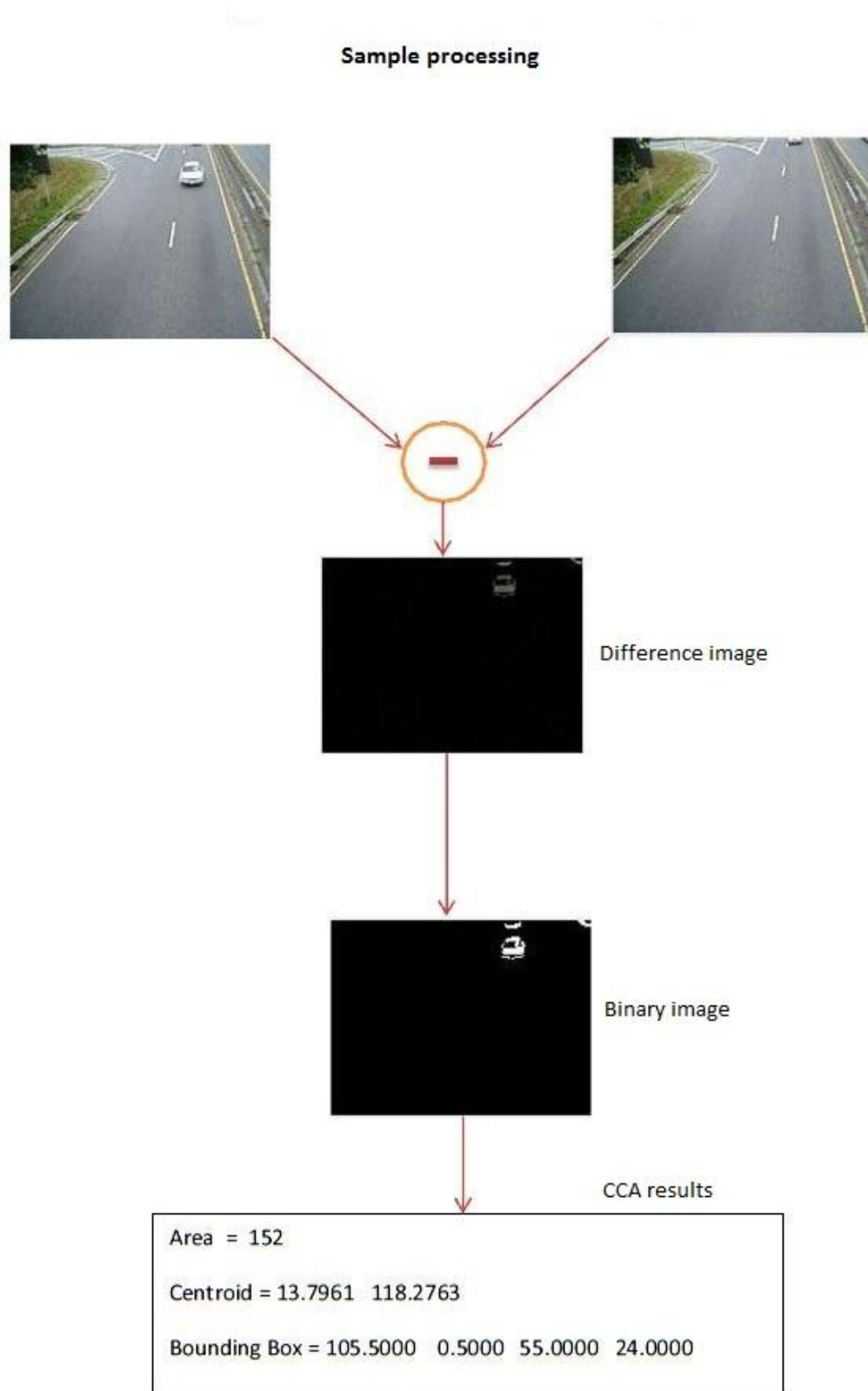


Figure 2.5: Sample processing

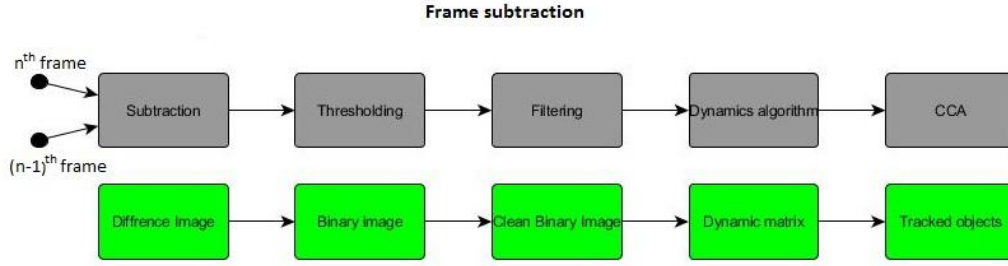


Figure 2.6: Flow chart of Frame subtraction

frame 1	frame 2	Difference image

Table 2.1: Difference image in Background subtraction and Frame subtraction

The difference image obtained in frame subtraction is quite different from that of background subtraction. The moving objects whose intensity is constant won't be visible in the binary image. If a square with constant intensity (white) moves on a black background, you will be able to see only the edges of the square in the frame subtracted difference image. Whereas in background subtraction, you will be able to get complete square in the binary image. The 2.1 will help you notice the difference in the "difference image" obtained using frame and background subtraction.

The difference image obtained in frame subtraction method has disconnection in the same object which can not be filled with closing functions (ideally). In reality such constant intensity objects are very rare and noise present in the image will definitely turn on few pixels between the two lines shown in the above difference image. So using some sort of dynamic matrix Yang *et al.* (2005) along with image closing operation will help to make a good binary image which is then sent to extract features for tracking.







lamda = 1	lamda = 10	lamda = 20
		
		

Table 2.2: Variation of difference image with parameter lamda

In the code lamda is the variable used for dynamic matrix. lamda is basically a value allocated to pixels instead of “1” in the binary image. So if lamda is equal to 1, the dynamic matrix is same as binary image. But if lamda is 2, all the pixels in the binary image whose value is 1 are replaced by 2. In the next frame, this dynamic matrix will be updated using the information from the new difference image. The pixels previously allocated as value 2 will be decremented by 1 if they are not having value 1 in current binary image (meaning they are not moving or they are part of background). If any new pixel is found with value 1 in current binary image which has value of 0 or 1 in the dynamic matrix, it is replaced with value 2. So increasing the value of lamda will result in increased size of the tail in the moving object. The effect of the dynamic matrix can be viewed in table 2.2.

In table 2.2, we see how dynamic matrix varies with lamda. Increasing the value of lamda increases the tail of the moving object in a direction opposite to its motion. Fixing this value of lamda has to be done experimentally for every video. But lamda mostly depends on the size and the speed of the moving objects and hence can be fixed for normal videos. If an object moves too fast in the scene, then lamda has to be reduced so that it doesn’t leave a huge tail. To get satisfactory results, value of lamda can be 2 for most of the input videos.

There are three more parameters in the code which has relevance to getting nice blobs in the binary image/dynamic matrix. They are area threshold (area\_thresh in code), pixel level threshold (level in code) and subtracting frame (k frames old than the current frame; k is replaced by gamma in code).

Value of gamma can usually be kept 1. That is, we are subtracting the current frame with previous frame. Pixel level threshold is a parameter which is computed using Otsu's algorithm. Pixel level threshold depends mostly on the contrast between the foreground and background object. It also depends slightly on the noise level in the difference image. Pixel level threshold value has to be higher than the noise intensity and lesser than foreground intensity value. Its value can be kept as 25 (obtained after experimenting with 5-6 videos). Area threshold depends mainly on the size of the image. It is directly proportional to the size of the image. If the value of area threshold is 30 for 120x160 image sequences, it will be 60-70 for 240x320 image sequences. All these parameters have to be set correctly for a new video for proper tracking.

### **2.2.3 Histogram of Oriented Gradients (HOG)**

Histogram of Oriented Gradient (HOG) is an object descriptor. A set of features (color, intensity gradient, area, scale etc...) which helps in describing an object is known as "descriptor". HOG is a descriptor mainly used for detecting humans in an image. This set of features is invariant to scale (to some extent). It mainly depends on two factors, one is the contrast between foreground and background and the other is the geometry of the object.

HOG captures the local features of an object much better than any of the other descriptors. A detailed description of HOG can be read from Dalal and Triggs (2005). An online lecture Shah (2012) is also available on HOG which gives very clear explanation of the same. In this chapter we will briefly describe HOG and explain the code developed for tracking (code is available in the appendix)

## Extraction HOG features

Consider the image shown in the figure 2.7. It is an image of size  $64 \times 128$ . Divide the image into  $16 \times 16$  blocks with 50 percent overlap. There will be a total of 105 blocks ( $7 \times 15$ ). Each block is further divided into  $2 \times 2$  cells. A single cell has  $8 \times 8$  pixels inside them. Take gradient of the cells and quantize them into 9 bins as shown in figure 2.8. This vector of length 9, is a feature vector describing that particular cell. Concatenating all the feature vectors gives us a super vector of size 3780 ( $105 \times 4 \times 9$ ). This super vector describes the entire image.

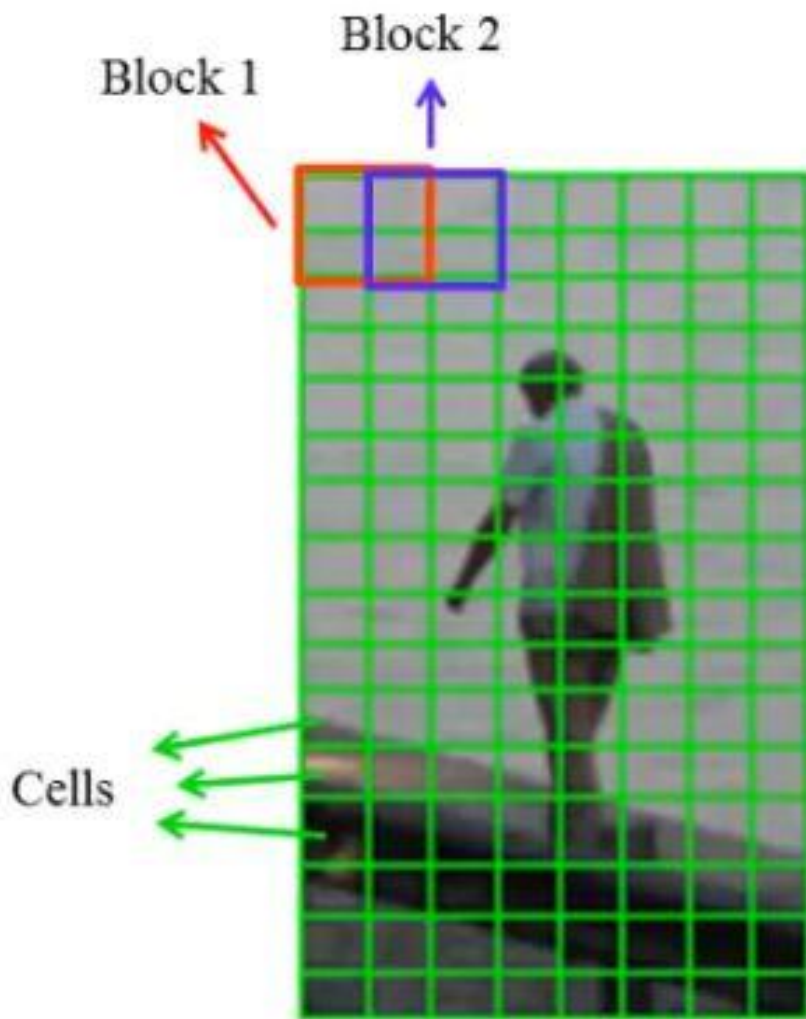


Figure 2.7: Extracting HOG feature

Authors of the paper Dalal and Triggs (2005) have given a simplified code for extracting HOG feature from an image. The code can be found in appendix and also

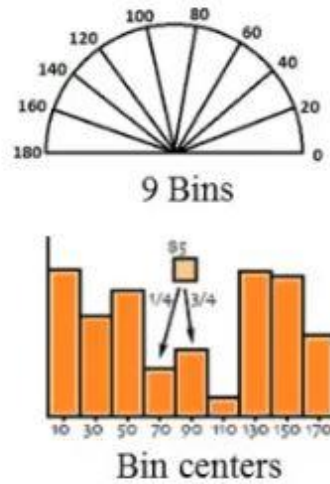


Figure 2.8: Histogram binning and interpolation

online Ludwig (2010).

### Tracking using HOG features

In the initial frame, we need to enter the object's location and its bounding box manually. From the initial bounding box, we extract HOG features and it describes the object for the next frame. We also extract HOG features from around the object in order to describe the background. In the next frame, we search for the object around its initial location. In the search radius we keep the size of the initial bounding box as constant. We keep extracting HOG features from the bounding boxes inside the search radius and compute its distance from the features extracted in the last frame. The bounding box which is closest to object feature (extracted in last frame) is assigned as the new location of the object in current frame.

During tracking, scale of the objects might vary through the image sequence and we need to predict the object's location in real time. Different objects will have different area on the image and hence we cannot use same window size for every object. Moreover, there is a problem of drift while tracking using descriptors. This drift occurs due to constant addition of noise from the background. Because of this drift, the bounding box slowly recedes from the object and gets stuck to one location in the background with slight jiggle from frame to frame. In the code, we have handled partial occlusion but it cannot detect an object which has disappeared due to complete occlusion and comes



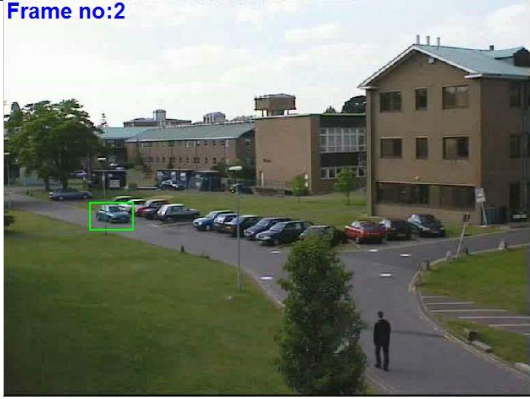
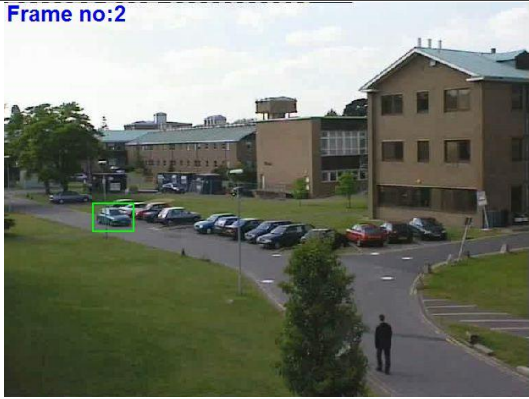
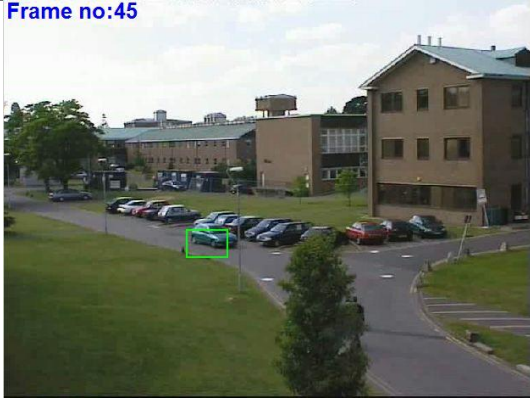

Only HOG feature	[HOG,initial location]
<p>Frame no:2</p> 	<p>Frame no:2</p> 
<p>Frame no:45</p> 	<p>Frame no:45</p> 

Table 2.3: Feature selection

back again in the scene 20-30 pixels away from the original position. For example, it cannot detect a car completely occluded by a tree once and comes back into the scene at the other end of the tree. In order to detect the car in such situations, search radius has to be increased and it might affect real time performance.

In order to improve the feature vector of the object, we included initial location, mean, variance and skewness of intensity along with HOG features. Initial location was useful when the object is moving slow otherwise drift gets increased. We also experimented with giving less weightage for location feature but still the drift was more while using initial location.

From table 2.3 it is clear that drift of the bounding box due to background noise becomes more when initial location is included as a feature vector.

## 2.3 Comparison

As descriptor based tracking suffers from serious drift problems, it seems better to track objects using background subtraction or frame subtraction method. Descriptor should be more robust than HOG and must take less computation time so that we can track the object in real time. We can come up with better descriptor than HOG (with above qualities) but implementing them in hardware is out of the scope of this project. We developed code for HOG in XC and have given in appendix. We did not implement tracker using HOG but used it to explain few concepts of XC programming language in the later chapter.

From the table 2.4 we see frame subtraction gives much better results than background subtraction. The leaves in the trees are moving due to wind. The intensity variation due to noise makes the output of background subtraction much worse. Leaves don't move much from one frame to next frame and hence frame subtraction output is better without those noises.

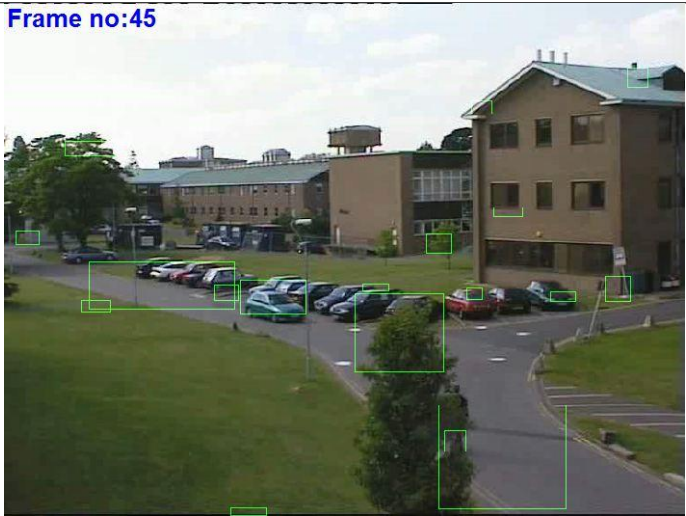


Algorithm	Output
Background subtraction	<p>Frame no:45</p> 
Frame subtraction	<p>Frame no:45</p> 
HOG tracking	<p>Frame no:45</p> 

Table 2.4: Comparing object tracking algorithms

## CHAPTER 3

### Object tracking system development in multithreaded environment

In this chapter we have discussed the optimized connected component analysis algorithm in detail. We have shown how we used optimized CCA to develop object tracking system. We have explained the object tracking system and have given the details of threads that run in parallel.

#### 3.1 Algorithm

Object tracking using frame subtraction method primarily consists of connected component analysis. It is used to analyze binary images to obtain features of the foreground objects. There are typically four stages to such algorithms. It is shown in figure 3.1. First the input (gray scale) image is obtained from FLASH or from SDRAM in RGB565 format. The image is preprocessed to extract foreground objects from the whole image. During the preprocessing step, thresholding is done to convert the image into a binary image which is easy to analyze. This binary image consists of a number of regions which are considered to be our desired object to be tracked or in other words, they are collectively known as foreground. Next, connected component analysis is used to label these foreground regions with a unique label. It also helps in extracting information such as area, bounding box and centroid of the objects available in the foreground. In the feature extraction stage, features such as area, bounding box and centroid can be obtained which helps in object tracking application.

##### 3.1.1 Labeling is not necessary for tracking

Labeling the blobs in the binary image often leads to confusion. It is due the presence of noise in the image. Compare frame 30 and frame 31 shown in the figure 3.2. In the

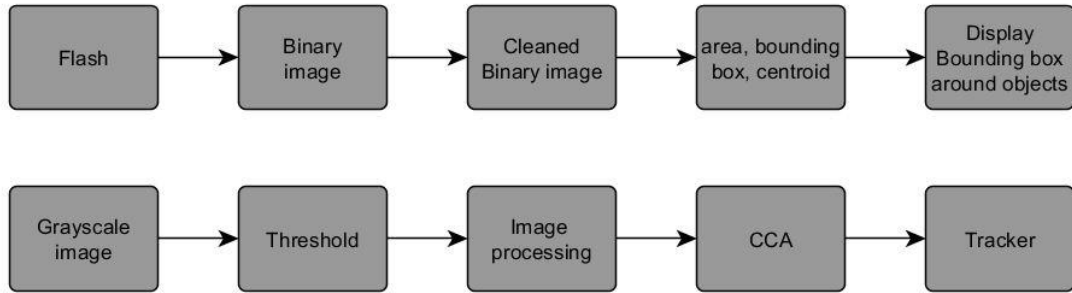


Figure 3.1: Flow chart of object tracking application

figure 3.2, we have shown 30<sup>th</sup> and 31<sup>st</sup> frame of the video, in which we are tracking an object running down a step. The ball is moving both in X-axis and Y-axis. In frame 30, we get the object labeled as two. But in frame 31, the same ball gets labeled as one. This clearly illustrates providing labels during connected component analysis is not necessary. Optimized connected component analysis does not issue labels. We can reduce the amount of memory consumption by half in optimized connected component analysis. This has been implemented in XMOS processor and the code is available in appendix.



Figure 3.2: Noise affects label's uniqueness

### 3.1.2 CCA and Optimized CCA

Traditional CCA algorithm, takes binary image as input and gives out label, area, bounding box and centroid as output. As we know labels are not necessary for tracking an object, we try to use this as an advantage to reduce the memory consumption. So optimized CCA mainly reduces the memory consumption.

The maximum number of disconnected object in a binary image is equal to  $\lceil \frac{Ht}{2} \rceil \times \lceil \frac{Wt}{2} \rceil$

### 3.1.3 Algorithm

The input for the algorithm is a cleaned binary image. The output of the algorithm is to provide area[size] and bounding box[size] information of the moving object. It is a single pass algorithm with two row buffers. Data structures used in the algorithm Ma *et al.* (2008) are given below.

size - WIDTH/2

T[size] - Translational table

PD[size] - Previous row Data table

PM[size] - Previous row Merger table

CD[size] - Current row Data table

CM[size] - Current row Merger table

Stack[size] - An array to keep track of serial merging. If label 3 merges with label 2 and while processing the current row, label 2 merges with label 1, it means label 3 is merged with label 1.

SP - Stack pointer

Size of the data structure is half the width of the image. We assume there cannot be more objects than half the width of the image. In worst case scenario, we will have to use full size of the data structure and it will be mainly due to noise.

In figure 3.3, the width of the image is 8. It shows the maximum number of objects within two rows due to noise cannot be more than 4. We further make an assumption that number of objects won't exceed "size" in the entire image. In reality we won't observe noise as shown in above figure. As described in chapter 3, noise depends on the threshold we choose. If threshold chosen is low, noise levels may exceed and program

### Worst case noise

1		2		3		4	
	1		2		3		4

Figure 3.3: Worst case noise

might terminate. It will print an error message in the console. This error message is due to illegal array access. As noise level increases, allocated data structure size won't be sufficient and there will be a label which exceeds the data structure size. When we observe an error message indicating illegal array access, increasing the threshold is one of the solutions. Let us go through the labeling process in the algorithm. The table 3.1 will show input and output of the labeling function.

### Labeling

Previous row	A/AE	B/BE	C/CE
Current row	DE	X	

Figure 3.4: Labeling

In figure 3.4, X is the current pixel which has to be given a label. It is labeled based on its four neighbors which have been already processed by labeling function. A, B and C are labels issued by labeling function while processing the previous row. AE, BE and CE are equivalent labels issued while processing the current row. A, B, C, AE, BE and CE all remain "0" at the beginning of the image pixel. Equivalent label information is available in translation data structure. The following flow chart shown in figure 3.5 is used while issuing a label for X.



Input	Output
0,0,0,0,0,1,1,1,1,1,1,1,1,1,1,1,1,1, 0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,1,0, 0,0,0,1,0,1,0,1,0,0,0,1,1,1,0,1,0,0, 0,1,1,1,0,0,1,0,1,0,0,1,1,0,1,1,0,0, 0,0,0,0,0,0,0,0,0,1,0,0,0,0,1,0,0,0, 0,0,0,0,0,0,0,0,0,0,1,1,1,1,0,0,0,0}	0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 1 0 1 0 2 0 0 0 3 3 3 0 1 0 0 0 1 1 1 0 0 1 0 1 0 0 2 2 0 1 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 0 0 0 0
1,0,0,0,0,1,1,1,1,1,1,1,1,1,1,1,1,1, 1,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,1, 1,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,1, 1,0,1,0,0,1,1,1,1,0,0,1,1,1,1,0,0,1, 1,0,1,0,1,0,0,0,1,0,1,0,0,0,1,0,1,0, 1,0,1,0,1,0,1,1,1,0,1,0,1,1,0,1,0,0}	1 0 0 0 0 2 2 2 2 2 2 2 2 2 2 2 2 2 1 0 0 0 2 0 0 0 0 0 0 0 0 0 0 0 0 2 1 0 0 2 0 0 0 0 0 0 0 0 0 0 0 0 0 2 1 0 2 0 0 3 3 3 3 0 0 4 4 4 4 0 0 2 1 0 2 0 3 0 0 0 3 0 4 0 0 0 4 0 2 0 1 0 2 0 3 0 4 3 3 0 5 0 6 5 0 2 0 0
0,1,1,0,0,0,1,1,0,1,1,1,1,1,1,1,1,1, 1,0,0,1,0,1,0,0,1,0,0,0,1,0,0,0,0,1, 1,0,0,0,1,0,0,0,1,0,0,0,1,0,0,0,0,1, 1,0,0,0,1,0,0,0,1,0,0,0,1,0,0,0,0,1, 1,0,0,0,1,0,0,0,1,0,0,0,1,0,0,0,0,1, 1,0,1,1,1,0,1,1,1,0,1,1,1,0,0,1,1,1}	0 1 1 0 0 0 2 2 0 3 3 3 3 3 3 3 3 3 1 0 0 1 0 2 0 0 2 0 0 0 2 0 0 0 0 2 1 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 0 1 1 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 0 1 1 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 0 1 1 0 2 1 1 0 3 1 1 0 4 1 1 0 0 5 1 1

Table 3.1: Input and Output of CCA

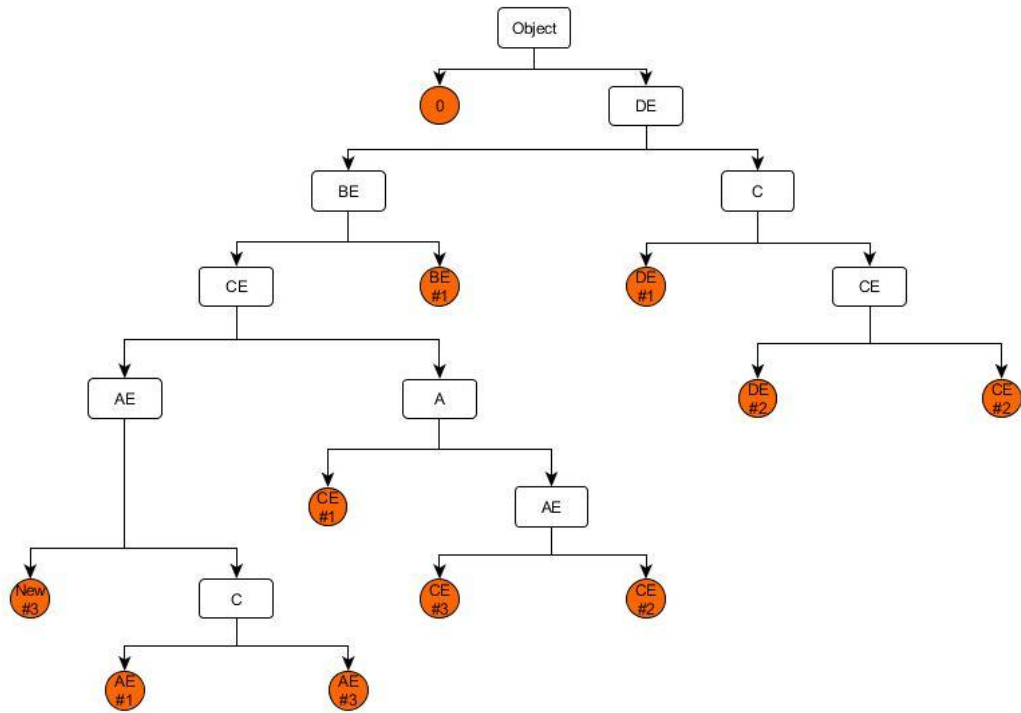


Figure 3.5: Flow chart for issuing label



In the above flow chart shown in figure 3.5, rectangular boxes indicate condition statement. The branch to the right indicates the condition is true and the left indicates the condition is false. For example, if the pixel is an object, then check if DE is greater than zero or else label the pixel as “0”. The darkened circle indicates the label issued to the pixel. #1, #2 and #3 inside each darkened circle points to specific functions.

1. It is known as update general function. When translation and merging is not necessary a simple area update function is sufficient. For area computation, the pseudo code will be

```
CD[index] = CD[index] + 1;
```

Where index is the respective label issued in the darkened circle.

2. This function is known as update stack. If DE is same as CE it means the connection between them has already been established before and hence a normal update as #1 is sufficient. If they are not same, it means we have to merge them together. Stack is used to keep in track of serial merging. Merger table is updated with the help of stack and stack pointer. For area computation, the pseudo code for update stack will be

```
if (DE==CE)
{
CD[CE] = CD[CE] + 1;
}
else
{
stack(CE,DE);
CD[CE] = CD[DE] + CD[CE] + 1;
CD[DE] = 0;
}
```

3. This function is known as update translate. The following steps are performed. Translation of previous row's label “C” to current row label “DE”. Accumulate area of label “C” in “DE”. After accumulation in current row label, previous row data table has to be cleared at index “C”. The steps in update translate are as follows.

```
T[C] = DE;
CD[DE] = CD[DE] + PD[C] + 1;
PD[C] = 0;
```

### 3.1.4 Translation

Translation table helps in maintaining connection between current row pixel and previous row pixel. We will take the image shown in figure 3.6 as example.

Translation

Previous row				1	1	1	1	
Current row	1		2					2

Figure 3.6: Translation

While processing the current row, we have started with issuing “1” as the label for first pixel. We issue “2” for second pixel as it is not connected with label “1” of current row. After issuing label “2” we notice that its neighbor C is greater than “0” which means label “2” of current row is connected to label “1” of previous row. Translation table helps us maintaining this connection. Label “1” in previous row gets transformed to label “2” of current row.

$T[1] = 2;$

Equivalent label for previous row’s label “1” is “2”. This connection has been established at the end of the current row. You can see the final pixel of the current row is labeled as “2”. Translation table helps in such connections in the current row. Translation table is cleared at the end of each row after completing all the necessary calculations for area, bounding box and centroid.

### 3.1.5 Merging

If there are two different current row labels in the neighborhood, they are connected and must have same label. This connectedness is achieved with the use of merger table. The image shown in figure 3.7 can be taken as an example for merging.

While processing 1<sup>st</sup> pixel in the 2<sup>nd</sup> row, we issue label “1” (Whenever we en-

## Merging

row number							
1			1	1	1	1	1
2		1					1
3	1					2	X

Figure 3.7: Merging

counter first object pixel in a new row we start issuing label starting from “1”. This is the main idea of label reuse). This leads to translate previous row’s label “1” to current row’s label “1”.

$$T[1] = 1$$

At the end of 2<sup>nd</sup> row after issuing label “1” to 8<sup>th</sup> column pixel, we empty translation table and make it all zero. Similarly while processing 1<sup>st</sup> pixel of 3<sup>rd</sup> row, transition table is invoked. Again T[1] becomes equal to 1. 2<sup>nd</sup> pixel of 3<sup>rd</sup> row gets a new label “2” as its not known to be connected to current row label “1” till X is seen as an object pixel. When we have to label X, we have to make sure current row label “2” is connected to current row label “1”. This connection is known with the help of translation table and merger table. First translation table makes CE of X as 1.

$$CE = T[1] = 1;$$

After going through the label selection flow chart, X is issued label “1” and merger function is invoked to connect current row label “2” as “1”.

$$M[2] = 1;$$

The table 3.2 shows when translation and merging function are invoked based on the equivalent labels.

DE	AE	BE	CE	Label	Translation	Merging
0	0	0	0	New	May be	-
0	0	0	1	CE	May be	May be
0	0	1	0	BE	-	-
0	0	1	1	CE	-	-
0	1	0	0	AE	May be	-
0	1	0	1	CE	-	May be
0	1	1	0	BE	-	-
0	1	1	1	CE	-	-
1	0	0	0	DE	May be	-
1	0	0	1	CE	-	May be
1	0	1	0	DE	-	-
1	0	1	1	DE	-	-
1	1	0	0	DE	May be	-
1	1	0	1	CE	-	May be
1	1	1	0	DE	-	-
1	1	1	1	DE	-	-

Table 3.2: Merging and Translation depending on the neighbors

### 3.1.6 Inner loop

Inner loop is a condition where there exists a loop inside a U-shaped connected object. It is the simplified version of spiral object. The image shown in figure 3.8 will help us understand the process dealing with inner loops.

When we have to label X, the algorithm goes through following steps in sequence.

1. AE=BE=DE=0, CE=1 as C is connected with current row label “1”. Its connection information is present in translation table.
2. X is issued “1” as its label.
3. CD[1] is incremented by 1 for area computation.
4. Check if A is connected or A is greater than zero.
5. In this case A=2 and hence connected to current row label 1. 2’s area has to be accumulated to current row label “1”’s area.
6.  $CD[1] = PD[2] + CD[1]$ ;
7.  $PD[2] = 0$ ;

The following ideas are important while implementing the algorithm.

1. In this case we need not translate A. We need not allocate T[2] as 1. It won’t be necessary as it is an inner loop.

## Inner loop

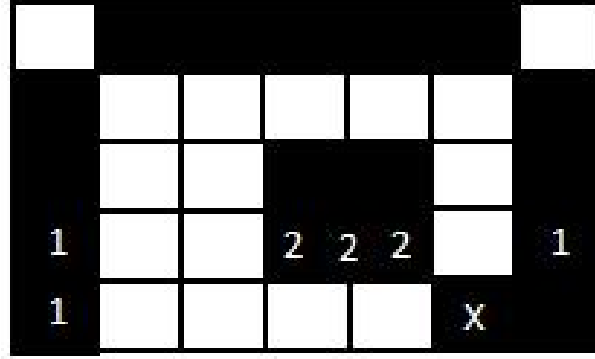


Figure 3.8: Inner loop

2. We need not check whether B is an object pixel. If B is an object pixel then BE will be same as CE. This implies CE covers BE. This can be noticed in the table for merging and translation.
3. Similarly DE covers AE and CE. BE if exists covers AE and CE. If BE is greater than 0, then even if AE or CE exists it will be equal to BE.

## 3.2 Implementation

While implementing such an algorithm in XMOS embedded platform, the image data (RGB565 format) is streamed from the SDRAM in a raster format. Classical labeling algorithm is a two pass labeling process which requires the whole image to be present in the buffer to extract features. Unfortunately, memory available in XMOS is as low as *64kb/core*. So an optimized single pass connected component analysis is implemented instead of classical connected component analysis.

### 3.2.1 Image format

Image format used is RGB565 format. We read image.tga file of size  $480 \times 272$ . In RGB888 format, we need  $382.5\text{kilobytes}$  ( $480 \times 272 \times 3 \div 1024$ ) to store image.tga. The image size is much more than  $64\text{kilobytes}$ . We need to break the image into 8 parts and load them part by part into SDRAM. Each part is  $47.8125\text{kilobytes}$ . The file image.tga has 8 bits each for red, blue and green. We need to convert it to RGB565 format by ignoring 3 least significant bits from blue and red, and 2 least significant bit from green.

Obtained RGB565 format image is loaded into SDRAM. We use only green component of the image in connected component analysis. Green is less noisy when compared to red and blue since we lose only 2 least significant bits. A RGB image when converted to grayscale, green has about 70 percent contribution[reference]. RGB565 format is used to brighten each pixel in the LCD. This is shown in figure 3.9.

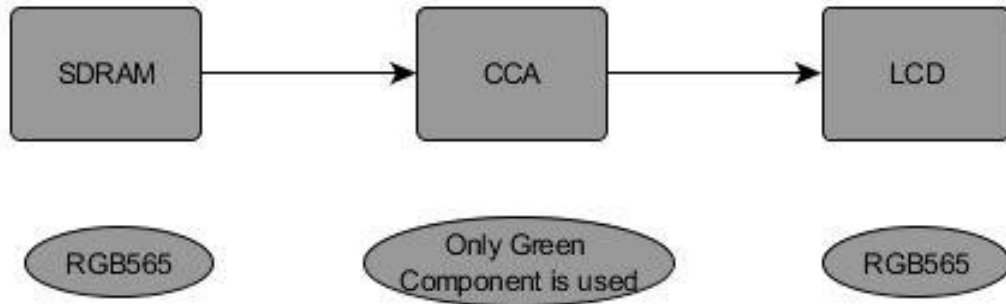


Figure 3.9: Image format

### 3.2.2 Object tracking system

The design runs 5 threads in parallel. The figure 3.10 will describe how the threads interact with each other and display tracked object in the LCD.

In figure 3.10 c\_loader, client, c\_sdrum and c\_lcd are channels used for communication between different threads. The grey rectangular boxes app, loader, display\_controller, sdrum\_server and lcd\_serer are threads. They all run in parallel. The green rectangular boxes are hardware that is connected to the slice kit. SDRAM corresponds to SDRAM

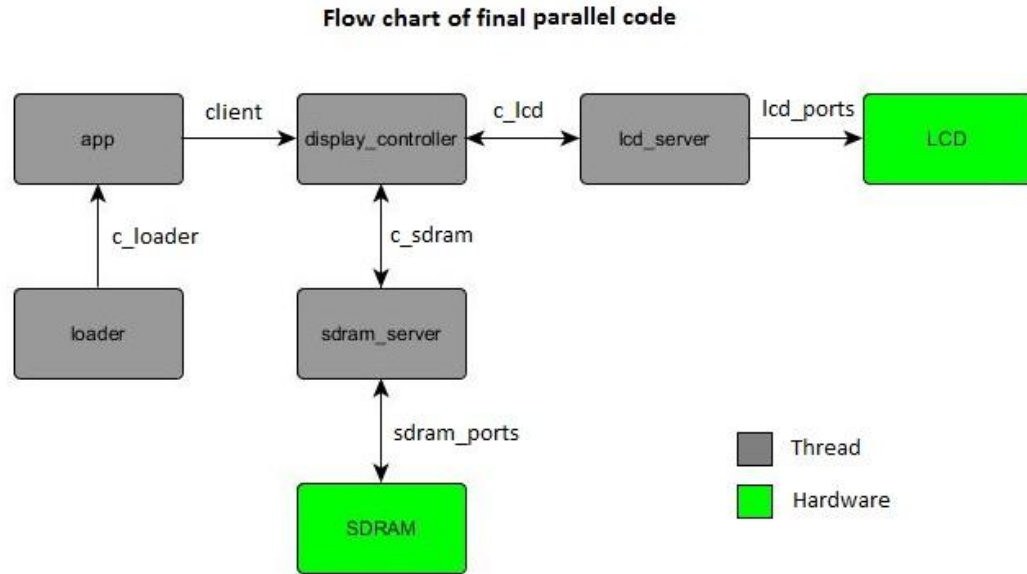


Figure 3.10: Thread diagram

slice and LCD corresponds to LCD slice. Sdram\_ports and lcd\_ports are specific ports to which sdram slice and lcd slice are connected to slice kit. Port mapping has been explained in chapter 3.

Loader thread does the following steps until all the images are loaded into the SDRAM.

1. Read image in TGA format of size  $480 \times 272$
2. Divide them into 8 parts
3. Load each part into SDRAM after converting each pixel to RGB565 format (It concatenates two pixels into one 32-bit integer)

The app thread in figure 3.10 is our object tracking application. It takes image data from SDRAM, does the processing and makes necessary changes to the image data to display it in LCD along with the bounding box around the object.

In detail, app thread reads one row of pixel data (480 pixel = integer array of size 240) from current image and same numbered row from previous frame. It subtracts these two arrays to get a difference array. Difference array needs to be processed pixel by pixel to obtain a binary image. We take 1<sup>st</sup> variable from integer difference array of size 240. An integer variable occupies 32 bits while a pixel occupies only 16 bits in RGB565 format. There are two pixels concatenated into one single integer variable.

As described earlier, 1<sup>st</sup> 16 bits of the integer corresponds to pixel 1 and later 16 bits corresponds to pixel 2 of the LCD.

We need to separate red, green and blue from 16 bit data. As per the format, 5 least significant bit corresponds to red, 5 most significant bit corresponds to blue and the remaining 6 bits in the middle corresponds to green. We consider only green component of the difference array. The green component is multiplied by 4 and passed on to thresholding. After thresholding the pixel (green difference of the pixel), it is passed on to optimized CCA. Once all the pixels ( $480 \times 272$ ) are processed, we get area, bounding box and centroid information of the object from optimized CCA. Using the extracted features, we can track the object.

Display controller thread manages SDRAM server and LCD server. It manages reading and writing of data into sdram. Next image handle is maintained in such a way that overwriting does not occur on lcd. Further information is given in comments section of the code. Display controller is available online Jason.

We have given the code for displaying a video in the appendix. It does not use display controller and loader threads. Images are loaded into sdram one by one manually. Video display has been already discussed in chapter 3. XC codes available online are not included in appendix as far as possible.

Sdram\_server and lcd\_Server gets commands from display controller. Their functions are explained in chapter 3. They are not explained under the heading of sdram\_server or lcd\_server but the functions explained are similar to it.



# CHAPTER 4

## Experimental setup and Results

We used two development board from XMOS namely XK-1A and XC-1A before moving on to slice kit. XK-1A is the basic development board whose flash size is 128 kilo bytes. We store image in raw format into the flash. With the stored image, we verified otsu's algorithm and optimized CCA algorithm. XC-1A is more sophisticated than XK-1A and has a flash memory of size 512 kilo bytes. We stored couple of images to test optimized CCA in continuous mode. We got frame rate less than  $1\text{frame/sec}$  for a sequence of  $120 \times 160$  images. After timing analysis we realized flash is taking about  $50\text{ms}$  to get one pixel while CCA computation takes place at much faster rate ( $6\text{ms}$  per pixel). Even though flash and CCA thread was running in parallel, CCA thread had to wait for flash for a long time which is very inefficient. So we either have to build a buffer to match the modules running in two different speeds or use SDRAM instead. We have given the timing analysis in results section later in this report. For our final system, we used slice kit, SDRAM for storing images and LCD for displaying the output.

### 4.1 Experimental setup

#### 4.1.1 Flash

Flash storage is permanent memory storage. Data won't get erased after we switch off the power supply to the board. In XC-1A development board, 512 Kilo byte SPI flash memory is present. Flash programming chapter in "Tools User Guide" XMOS, helps us understand the working of flash operations. A number of APIs have been present in XMOS development environment for reading and writing into flash. The syntax for using the APIs are present in the "Tools User Guide". Flash library built inside XMOS development environment (libflash) supports a wide range of flash devices available in the market. Each flash device is described using SPI specification file. The SPI

specification file describes the device characteristics such as page size, number of pages, commands for reading, writing and erasing data. If we have to use a flash device that is not supported by XMOS development tool, we need to get the SPI specification file for the device and include it in the source folder.

The configuration file for Numonyx M25P10-A NUMONYX (2008) is shown in figure 4.1.

```

10,          /* 1.  libflash device ID */
256,         /* 2.  Page size */
512,         /* 3.  Number of pages */
3,          /* 4.  Address size */
4,          /* 5.  Clock divider */
0x9f,        /* 6.  RDID cmd */
0,          /* 7.  RDID dummy bytes */
3,          /* 8.  RDID data size in bytes */
0x202011,    /* 9.  RDID manufacturer ID */
0xD8,        /* 10. SE cmd */
0,          /* 11. SE full sector erase */
0x06,        /* 12. WREN cmd */
0x04,        /* 13. WRDI cmd */
PROT_TYPE_SR, /* 14. Protection type */
{{0x0c,0x0},{0,0}}, /* 15. SR protect and unprotect cmds */
0x02,        /* 16. PP cmd */
0x0b,        /* 17. READ cmd */
1,          /* 18. READ dummy bytes */
SECTOR_LAYOUT_REGULAR, /* 19. Sector layout */
{32768,{0,{0}}}, /* 20. Sector sizes */
0x05,        /* 21. RDSR cmd */
0x01,        /* 22. WRSR cmd */
0x01,        /* 23. WIP bit mask */

```

Figure 4.1: SPI specification file for FLASH Numonyx M25P10-A

Flash attached in slice-kit is “NUMONYX\_M25P16”. This can be verified with a libflash code available online Jason. This code uses following functions.

1. fl\_getFlashType()
2. fl\_getFlashSize()
3. fl\_getPageSize()
4. fl\_getDataPartitionSize()
5. fl\_getNumDataPages()
6. fl\_getDataSectorsSize()
7. fl\_eraseAllDataSectors()
8. fl\_writeDataPage(0,mypage)
9. fl\_readDataPage(0,mypage)

#### 10. fl\_disconnect()

You will be able to find explanations of the above functions in “Tools User Guide”. When we run the code in slice-kit, it will print the output as shown in figure 4.2 in the console.

```
FLASH fitted : NUMONYX_M25P16.  
FLASH size: 2097152 bytes.  
FLASH page size: 256 bytes.  
FLASH data partition size: 2031616 bytes.  
FLASH number of pages in data partition: 7936  
FLASH number of sectors in data partition: 31  
FLASH data sector size: 65536 bytes.
```

Figure 4.2: Output of libflash code

Writing image data into flash and reading can be done in various ways. If the image size is small (less than 50 kilo bytes) we can store the entire image in the array “mypage” of the libflash code and directly load the image into flash. Format of the image used to store in the flash is very important for easy reading of the data. We followed the following steps to get the right image format.

1. Save the image in “pgm” format
2. Open the image in an editor and remove the headers of image.pgm. Headers contain information regarding the size of the image.
3. Load this into flash with the help of “libflash” code if the image size is small or else use the “c code” that has been developed for loading.

If the image size is small and to use “libflash” code for loading, we need to follow the following steps.

1. Read the image into Matlab
2. Open a text file and write the pixel value in comma separated format. If the image size is 120x160, then the number of rows in the text file is 120 and the number of columns is 160.
3. Copy the contents of the text file into an array “mypage” to load it into flash. Once we copied the image data into the array, running the program will load it into flash.

If the image size is large, an error message will be displayed in the console indicating memory has exceeded *64kilobytes*. We need to break the image into 2 or 3 so that we can load the image in parts.

1. Break the text file so that the memory error doesn't occur. Instead of copying the entire 120 rows in the text file, we can try loading 60 rows.
2. Copy 1<sup>st</sup> part of the image into "mypage" array and load it into the flash.
3. Copy 2<sup>nd</sup> part of the image into "mypage" array, change the writing address of flash according to size of the 1<sup>st</sup> part. If part 1 contains 60 rows and 160 columns, we have 9600 bytes of data. The size of the myspace should have been changed to 9600. While loading the second part, the writing address has to be increased by 9600.
4. If there are more parts than 2, continue 2<sup>nd</sup> and 3<sup>rd</sup> step until the whole image is loaded into flash.

As the above manual method is cumbersome, we developed a "C code" to do the same steps. The "C code" will read the raw image data (headers removed) load it into flash. It handles image of any size which is within the size of data sector in flash. This "C code" has to be built in XMOS development environment to obtain an executable binary file. Once we have the binary file and raw image file, we need to configure flash device and load the image data into it. Not all 512 kilo bytes in flash can be utilized for storing data. We need to configure amount of space for data sector. The following commands in XMOS command prompt can be used to configure the flash and load the raw image data into it.

1. `xrun -l`
2. `xflash -target-file XK_SK_L2.xn -erase-all`
3. `xflash <filename>.xe -target-file XK_SK_L2.xn -boot-partition-size 65536 -write-all image_raw.pgm`
4. `perl "write.pl" image_raw.pgm 20`

First command will list all available XMOS devices. Second command erases complete flash memory including the boot partition. Third command allocates 65536 bytes to boot partition, loads <filename>.xe into it and loads image\_raw.pgm into data partition. Slice-kit has 32 sectors as shown in the flash test output. If we want to write the

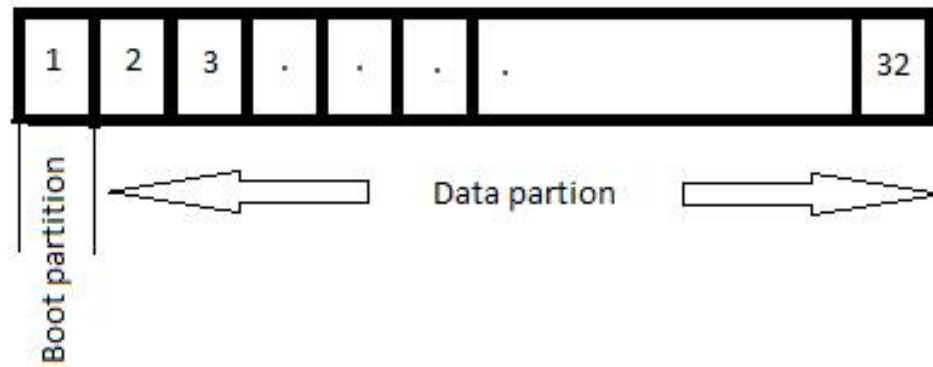


Figure 4.3: FLASH partition

image into particular sector, use  $4^{th}$  command. The image gets loaded into  $20^{th}$  sector. The figure 4.3 shows boot partition and data partition.

When we have to use slice-kit development board instead of XC-1A (or to any other board) the following changes have to be made to “C code” to load image data into flash in slice kit. In the appendix, in connect.xc file we need to change the device specification so that it matches with the flash specification in slice kit. Similarly, to use libflash code to check the flash in slice kit, we need to add the name of the flash attached in slice kit in the myflashdevices[] array.

### 4.1.2 SDRAM

SDRAM that we are using for our project has *64MegaBits*. The memory available in SDRAM is split into 4 banks. Each bank has 212 rows, 28 columns of cells where a cell is a 16-bit memory. We have used sc\_sdrum\_burst-master an open source code developed by XMOS available at Jason. The figure 4.4 gives overall picture of the SDRAM working. There are three main functions used for performing writing and reading operations in the SDRAM. They are

1. sdrum\_buffer\_write(chanend, bank, row, col, size, buffer)
2. sdrum\_buffer\_read(chanend, bank, row, col, size, buffer)
3. sdrum\_wait\_until\_idle(chanend, buffer)

‘chanend’ is one end of the channel used for communication between the application program and the sdrum server. Bank can take any value between 0 and 3 choosing one

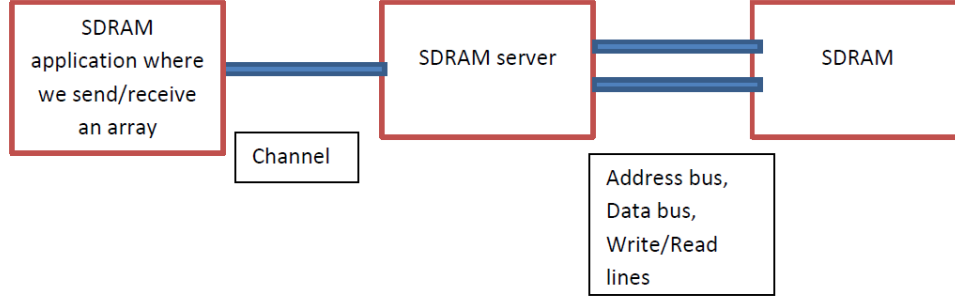


Figure 4.4: SDRAM server

among the four banks available. Similarly row and col are used to address a specific location in the SDRAM. ‘size’ is the size of the buffer that we want to be written onto or read from SDRAM.

The functions read/write two cells at a time. This can only be understood with an example. Say we need to write an integer array of 3 words.

Buffer = {0xFFFF, 0x1FFF, 0x2FFF}

Invoking the 1<sup>st</sup> function with size equals 3, `sdram_buffer_write(chanend, 0, 0, 0, 3, Buffer)`; sdram cells are written in the following fashion.

0x0000	0xFFFF	0x0000	0x1FFF	0x0000	0x2FFF
--------	--------	--------	--------	--------	--------

Table 4.1: Buffer array as stored in sdram

Now invoking the 2<sup>nd</sup> function with “col” changed as “col+1”, `sdram_buffer_read(chanend, 0, 0, 1, 3, Buffer)`; will produce a buffer with following values.

Buffer = {0xFFFF0000, 0x1FFF000, 0x2FFF\*\*\*\*}

### 4.1.3 LCD

LCD, we used in our project, has a resolution of  $480 \times 272$  with pixel format of RGB565. We have used `sc_lcd-master` which is an open source code of XMOS available at <http://www.github.com/xcore>. It displays XMOS logo on the screen. In this case, they store the entire image in RAM to display the image and use RGB565 format

for lightening each pixel. The main reason for not using RGB888 format is due to lack of availability of output pins in the slice kit.

The code writes two neighboring pixels at a time. An integer data type is of 32 bits, first 16 bits correspond to one pixel and the later 16 bits correspond to its neighboring pixel. For example, 0xFFFF0000 makes the first pixel black and second pixel white. An image has to be in above format to be displayed on the LCD screen. First an image of RGB888 format has to be converted into RGB565 format. Then we need to concatenate two neighboring pixel. The following example explains how to concatenate two pixels (We have written a Matlab code which does this for us, later a c code is also written; both are available in appendix). The figure ?? gives overall picture of the SDRAM working.

0xF0FF 0x00FF 0x01AE 0xFF0F...be the first four pixel value in RGB565 format without concatenation. 0x00FFF0FF 0xFF0F01AE will be our new concatenated format useful for displaying on LCD screen. So an image of size 120x160 in RGB24 format will become 120x80 in our new format. This can be understood better by looking at the code snippets provided in the appendix.

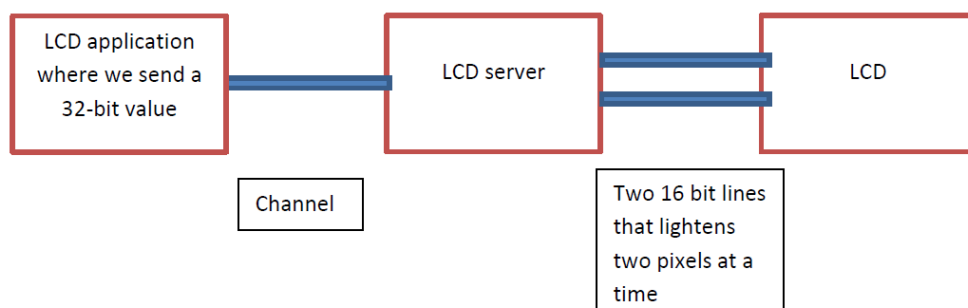


Figure 4.5: LCD server

LCD server takes care of the port configuration. It gets the data to be displayed in the correct format from the application program written by us and displays it on the screen. `lcd_init`, `lcd_req`, `lcd_update` are three main functions used in the code. `lcd_init` checks whether the LCD is ready for use. It checks for a control token of a given value. If the next byte in the channel is a control token which matches the expected value then it is input and discarded, otherwise an exception is raised. `lcd_req` is again similar to `lcd_init` where a token is sent through channel indicating the previous write operation

on the LCD is complete and hence the server can receive next data to be written on the screen. `lcd_update` receives a row of information from the application program (row here means  $240 \times 32$  bits or an integer array of size 240. Number of pixels in a row is 480. Since we write two pixels at a time in RGB565 format we only need an integer array of size 240). LCD server takes care how to display the entire row. Having finished displaying the entire row, `lcd_req` will receive a token indicating that we can send the next row's data. This happens in an infinite while loop to display an image continuously.

### **Double buffer**

Double buffer is used to prevent 'flickering' while displaying image on the LCD. We have image data stored in SDRAM. As the size of the LCD screen is  $480 \times 272$ , a single row of pixel will need a buffer of size  $240 \times 4$  bytes. An integer array of size 240 is required to display one row in LCD. It is 240 and not 480 because we use RGB565 format.

In double buffer usage, we have two integer buffers of size 240. While buffer A is being displayed on the LCD, buffer B is getting filled up with data for next row of pixels from SDRAM. Timing becomes important for real-time performance. In the code given in appendix, while displaying buffer A, we are clearing buffer B instead of loading it with next row data and vice versa. The necessity of clearing will be explained shortly. We can also triple buffer if we have memory to improve on time complexity.

There are two other functions that are fairly important. "add" and "sub" function. The server works with a dual buffer concept. There are two integer buffers of size 240. Buffer A is meant for displaying even rows while buffer B is meant for displaying odd rows on the LCD screen. The total number of rows on the LCD is 272. Let the size of an image we need to display be  $120 \times 160$  and assume we display it in the left corner of the LCD. Both A and B are initialized with the background color to start with. Between 1<sup>st</sup> row and 120<sup>th</sup> row we need to update the values of these buffers according to the image and this is done by the "add" function. Once we have finished displaying the entire image, we must display background for the entire 121<sup>st</sup> row but in the buffer B we will still have values of 119<sup>th</sup> row's data. Hence, the rest of the rows (121 - 270) will be filled with values of 119<sup>th</sup> row. To avoid this we use sub function which updates



(resets) the buffers with background color after displaying the entire image.

SDRAM and LCD are running in parallel. The following code snippet will help you understand they run in two different threads. Even though only 2 threads are required, 6 more threads are added to simulate worst case scenario. Instead of `par(int i=0;i<6;i++) while(1);`, we can write `while(1); 6 times` both means the same. The figure 4.6 shows the clock frequency versus number of threads. If more than 4 threads are active, each thread is allocated at least  $\frac{1}{n}$  cycles (for n threads).

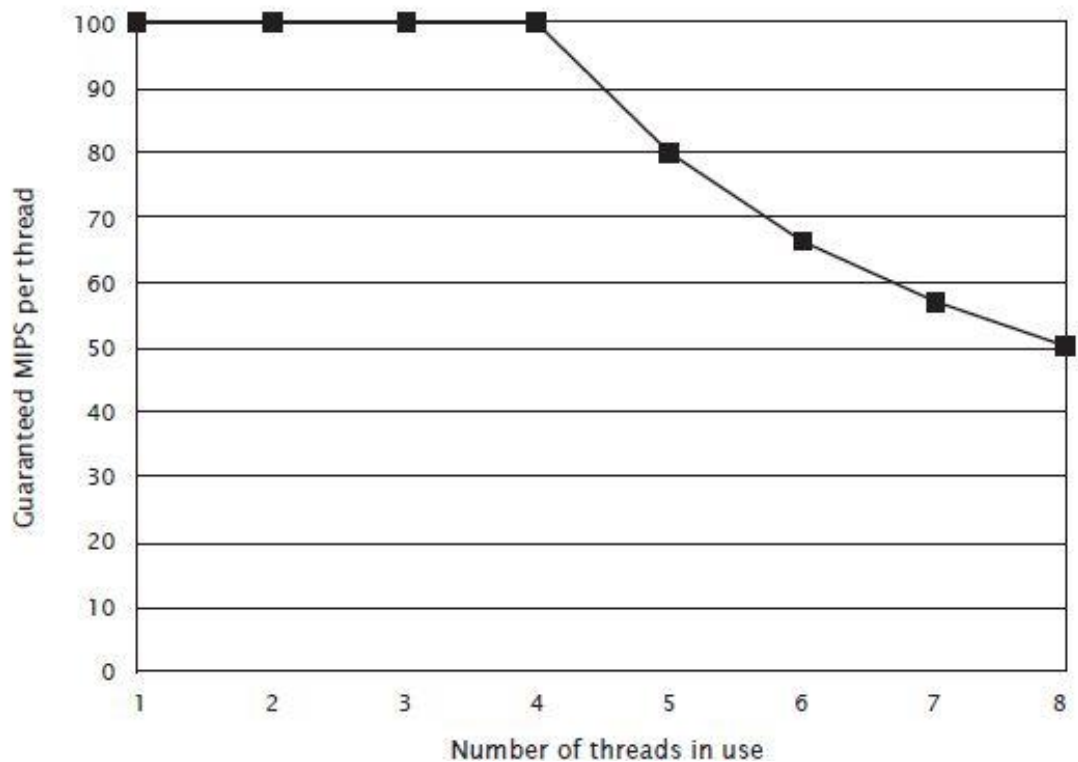


Figure 4.6: Processing speed versus number of parallel threads

#### 4.1.4 Video Display

Video is a sequence of images that has been displayed at a particular rate on the LCD screen. We store the image sequence of the video in the SDRAM and display them on the LCD screen. Every frame is displayed for 50ms which is tunable. Having understood how the SDRAM stores our images, it is easy to understand this. The following flow chart will help us understand.

The figure ?? shows the simplest way for displaying the video. If we want to im-

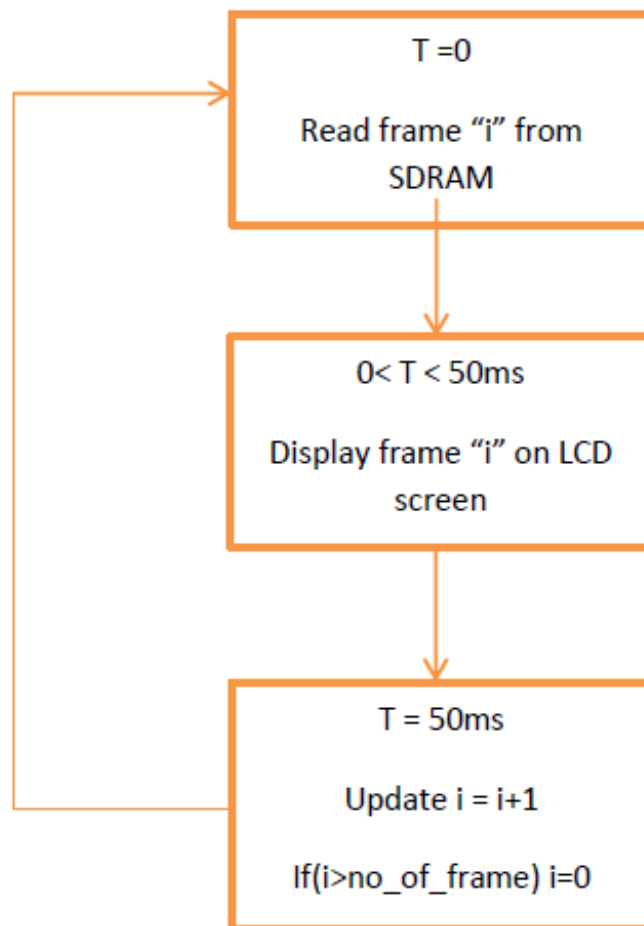


Figure 4.7: Video display

prove on the memory constraints, we can get the image line by line from the SDRAM instead of the whole frame. This might affect the real time performance but since we are allowing a delay ( $50ms$ ) this shouldn't be a problem. We can reduce the delay to account for it. When we have to display an image of size  $480 \times 272$  in RGB565 format, the size of the image is 255 kilo bytes ( $480 \times \frac{272}{2} \times 4 \div 1024$ ). We know the size of the internal RAM is 64kilobytes per processor. Even if we utilize both the processors available in slice kit, we only get 128 kilo bytes and program memory will take some part of it. This explains we cannot have an entire image in a buffer to display it on the LCD screen. So we used a display controller which helps us to display  $480 \times 272$  sized images. We will discuss about the display controller in next chapter.

#### 4.1.5 Slice Kit

Slice kit is a development board used in our project. It contains 2 XCORE processors handling 16 threads in parallel. Rapid prototyping of systems becomes possible with slice kit because it supports a lot of interfaces available as slices. We use SDRAM and LCD slices for our project. The experiment setup is shown in the figure 4.8.



Figure 4.8: Experimental setup

SDRAM used in our project has a size of 8 Mega bytes. LCD screen size is  $480 \times 272$ . In the main board you will be able to see four slots available: square, circle, start

and triangle. We can even interchange the position of LCD and SDRAM but we need to modify ports accordingly. As discussed, before developing applications for slice-kit we need to understand its hardware specifications.

CORE BOARD											
Connector	PLUG_00		SOCKET_00		SOCKET_01		SOCKET_10		SOCKET_11		
Core	0		0		0		1		1		
Type	Reverse 0		0		1		0		1		
Side	B (Top)	A (Bottom)	B (Top)	A (Bottom)	B (Top)	A (Bottom)	B (Top)	A (Bottom)	B (Top)	A (Bottom)	
1	DEBUG	MSEL	NC	NC	NC	NC	DEBUG	MSEL	NC	NC	
2	TCK	NC	NC	5V	X0D0	5V	TCK	5V	X1D0	5V	
3	GND	TMS	GND	NC	GND	X0D12	GND	TMS	GND	X1D12	
4	TDO	TDI	NC	NC	X0D11	X0D23	TDI	TDO	X1D11	X1D23	
5	PRSENT	GND	3V3	GND	3V3	GND	3V3	PRSENT N	3V3	GND	
6	X0D9	X0D3	X0D2	X0D8	X0D26	X0D32	X1D2	X1D8	X1D26	X1D32	
7	X0D8	X0D2	X0D3	X0D9	X0D27	X0D33	X1D3	X1D9	X1D27	X1D33	
8	GND	X0D10	GND	X0D1	GND	X0D25	GND	X1D1	GND	X1D25	
9	X0D7	X0D5	X0D4	X0D6	X0D28	X0D30	X1D4	X1D6	X1D28	X1D30	
10	X0D1	GND	X0D10	GND	X0D34	GND	X1D10	GND	X1D34	GND	
11	X0D6	X0D4	X0D5	X0D7	X0D29	X0D31	X1D5	X1D7	X1D29	X1D31	
SLOT											
12	X0D21	X0D15	X0D14	X0D20	X0D36	X0D42	X1D14	X1D20	X1D36	NC	
13	X0D20	X0D14	X0D15	X0D21	X0D37	X0D43	X1D15	X1D21	X1D37	NC	
14	CLK	GND	CLK	GND	CLK	GND	CLK	GND	CLK	GND	
15	X0D13	X0D22	X0D22	X0D13	X0D24	X0D35	X1D22	X1D13	X1D24	X1D35	
16	GND	RST_N	GND	RST_N	GND	RST_N	GND	RST_N	GND	RST_N	
17	X0D19	X0D17	X0D16	X0D18	X0D38	X0D40	X1D16	X1D18	X1D38	NC	
18	X0D18	X0D16	X0D17	X0D19	X0D39	X0D41	X1D17	X1D19	X1D39	NC	

Figure 4.9: Slice kit port map

In the figure 4.9 socket\_00, socket\_01, socket\_10, socket\_11 represent square, star, circle and triangle respectively. X0 indicates XCore0 and X1 indicates XCore1. We have to use correct ports in the application program.

## 4.2 Experimental studies and results

### 4.2.1 Experimental studies

Parameters of the code won't be fixed. Intensity threshold to convert difference image into binary image does not remain constant. Otsu's threshold gives bad results in obtaining clean binary image. Thus automatic threshold selection using otsu's thresholding is not possible. Otsu's threshold looks for two major peaks and gives the average value of those intensities.

In difference image we ideally need almost all pixels to fall into "0" and there will be very few pixels between 15 and 100. Object pixels would lie between 15 and 100. Hence to such histograms, otsu's threshold might issue 20 or 30 as its threshold and

binary image will be free from noise. In reality, due to presence of noise there might be a peak at 7 or 10 making the output of otsu's threshold as 7. This increases the noise present in the binary image which becomes hard to remove with any image processing. Intensity threshold has to be determined empirically for each video. This increase in noise levels due to otsu's thresholding is observed among few frames of the image sequence. It is always better to keep noise levels low by choosing the threshold empirically. A clean binary image is necessary in every frame to continuously track the object without losing it.

In difference pixels, we used all three components to track the object. While doing it we observed lots of noise. We observe the noise by displaying the clean binary image on the lcd. After numerous trials we observed the binary image is clean only with green component of the image. This is due to the following

1. We use 6 bits for green whereas we use only 5 bits for blue and red.
2. While converting RGB image to gray scale, green contributes about 70 percent to the gray scale image.

#### **4.2.2 Resource utilization**

The figure 4.10 shows the resources utilized for object tracking application.

In slice kit there are two Xcore processors. The above pi chart in figure 4.10 includes both the processor. Total memory is 128 kilo bytes (64 kilo bytes per processor). One timer has been used to determine the frame rate of the output. We obtained a processing speed of about 10 frames per second ( $9.67 \text{ frame/s}$ ).

#### **4.2.3 Results**

The results obtained from object tracking application are shown in figures 4.11, 4.12 and 4.13.

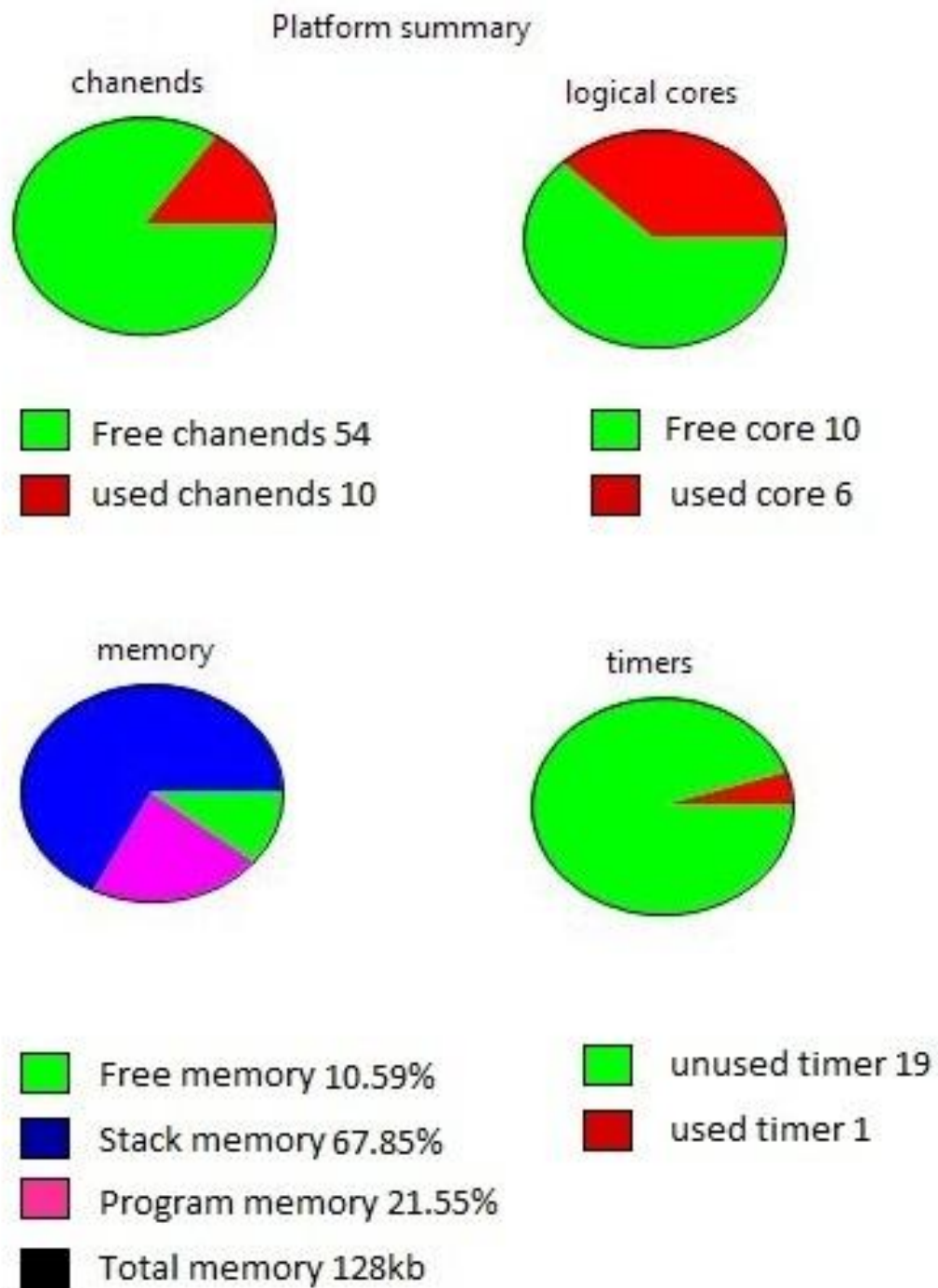


Figure 4.10: Resource utilization



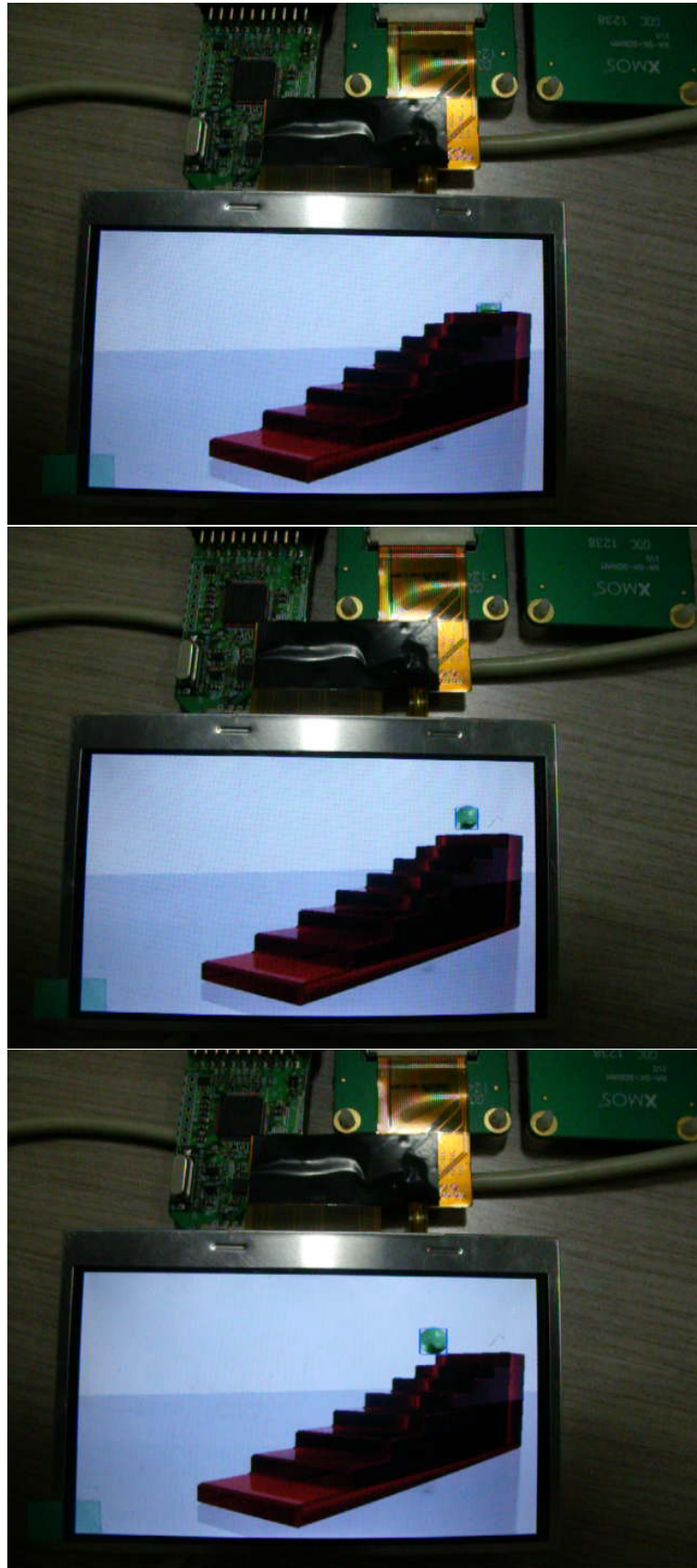


Figure 4.11: Bounding box has been put over the moving object

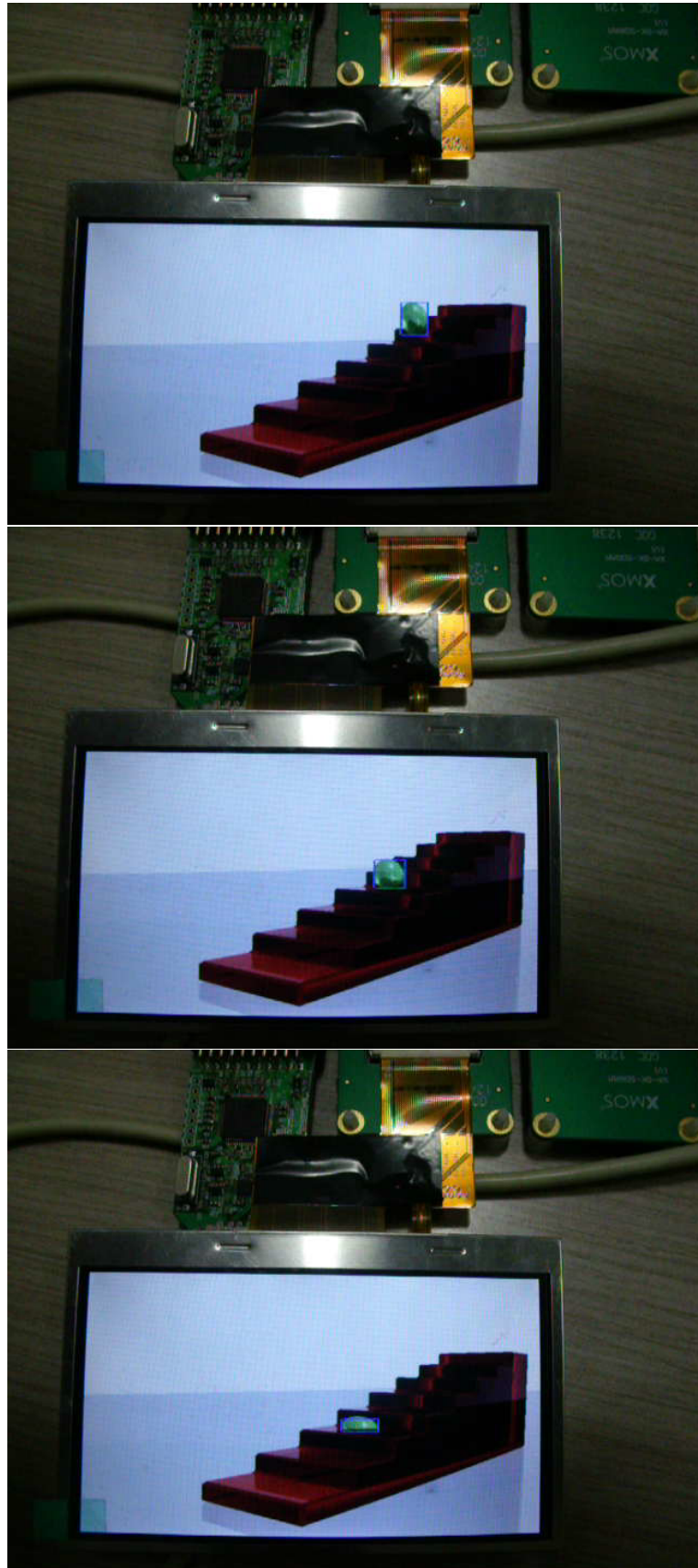


Figure 4.12: Bounding box has been put over the moving object



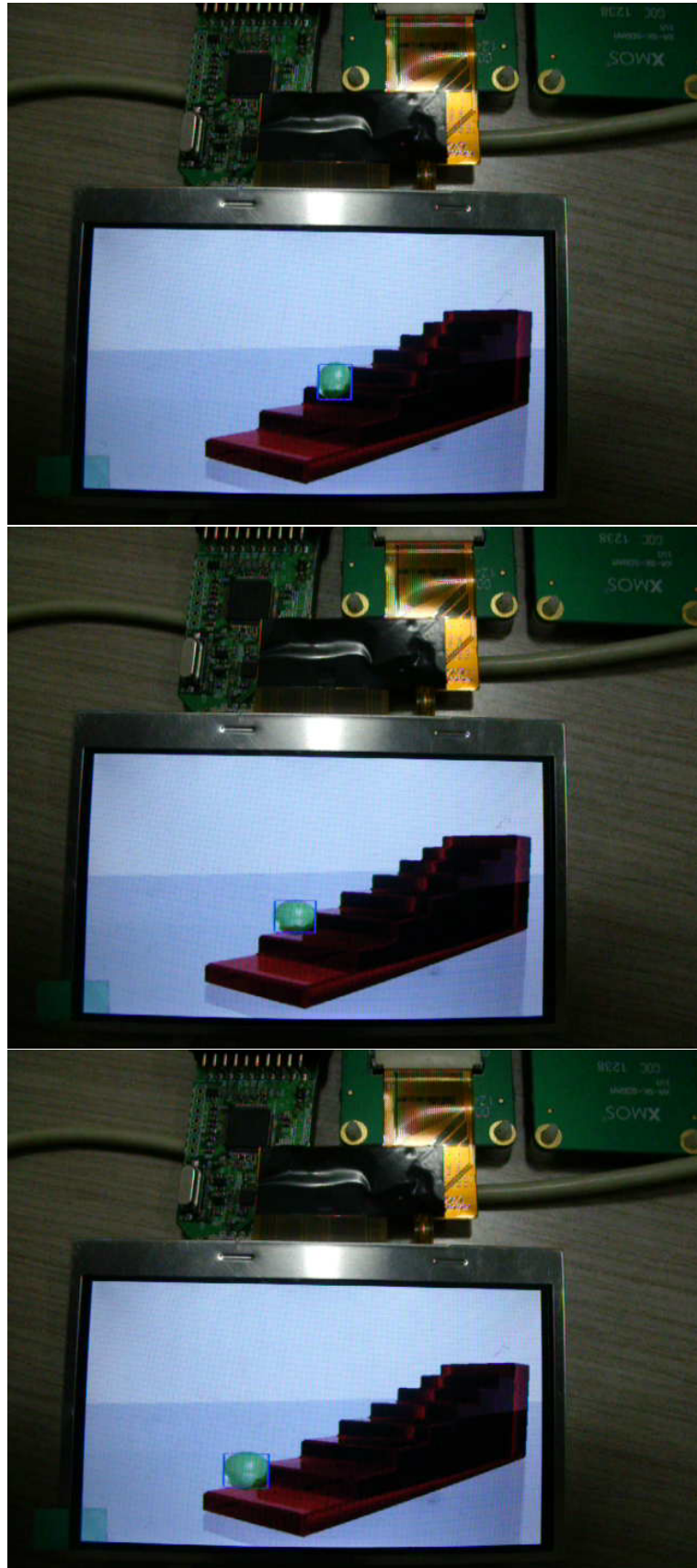


Figure 4.13: Bounding box has been put over the moving object

# CHAPTER 5

## Analysis of memory and timing

In this chapter we have given memory and timing analysis of the processor we used to develop our object tracking system. Timing analysis is done with the help of timer available in the board. Memory analysis is done with the help of XMOS development environment.

### 5.1 Memory analysis

Memory analysis is important in embedded systems in order to design an application with lowest memory consumption. Basic of C gives an overview of memory consumption due to different data types. The following tables ?? will provide few details about it.

Data type	Memory
Integer	4 bytes
Char	1 byte
Signed char	1 byte
Float	4 bytes
Double	8 bytes

Table 5.1: Memory required for different datatypes  
??

Memory in embedded systems can be categorized as program memory, stack memory and free memory. When we build a XC code, the compiler converts it to a binary file. This binary file is loaded into XMOS processor to run the application in XMOS hardware. XMOS processor has an internal RAM memory of 64 kilo bytes. The compiler checks whether the memory usage doesn't exceed 64 kilo bytes while building the XC code. Sum of program memory, stack memory and free memory will always be equal to 64 kilo bytes.

Stack memory consists of variables such as integers, characters, arrays etc...Program

memory consists of syntax and logic. This can be understood with the help of the tables 5.2 5.3 and 5.4.

In table 2, “remarks” indicate the inferences from the analysis of different codes that has been given in column “code”. Every code has very mild variations from rest of them. Comparing one among the rest gives a clear picture of the analysis.

In X MOS development environment, double click on the binary file available in the project explorer. This will open a window consisting of memory consumption, number of logical cores used and number of timers used. All the experimental data given in the table above are obtained from X MOS development environment. The above table helps programmer to know which part of the code goes to which type of memory. One can develop memory efficient code using above table.

## 5.2 Timing analysis

Timing analysis is very important to develop algorithm that can run in real time. All algorithms are developed from the basic operations such as comparison, addition, subtraction, division etc... How much time it takes to compute the value of  $\sin(x)$ , how many clock cycle it takes to compare two variables?, does it depend on the type of variables we use? All these questions can be answered after going through this section. In built timers are used for timing analysis. We have given time taken in terms of number of clock cycles rather than seconds. Time period of one clock cycle in X MOS processor is  $10ns$ .

The tables 5.5, 5.6, 5.7, 5.8, 5.9, 5.10, 5.11 and 5.12. helps in understanding timing constraints. It helps to develop better codes that can run in real time. It is better to keep memory and timing analysis concepts in mind before writing an application in XC. Once we finish writing 1000 lines of code, it becomes difficult to reduce the memory required by the code. In the same sense it is difficult to check which part of the code is responsible for exceeding the time constraint.

X MOS timing analyzer (XTA) is a tool that is available in X MOS development environment. It is used to determine whether all timing-critical sections of the code are guaranteed to execute within their deadlines. This tool can measure shortest and longest

S.no	Code	Stack memory	Program memory	Remarks
1	<pre>#include &lt;xs1.h&gt; void main(){char c;}</pre>	208	880	None
2	<pre>#include &lt;xs1.h&gt; void main(){int     c[16*1010];     char a,b;}</pre>	64652	884	Zero free memory
3	<pre>#include &lt;xs1.h&gt; void main(){timer t;}</pre>	208	896	None
4	<pre>#include &lt;xs1.h&gt; void main(){char c;     for(c=0;c&lt;1;c++);}</pre>	208	916	Compare with 1. Including “for loop” increases only program memory.
5	<pre>#include &lt;xs1.h&gt; void main(){char c;     for(c=0;c&lt;255;c++);     for(c=0;c&lt;255;c++);     for(c=0;c&lt;255;c++);     for(c=0;c&lt;255;c++);     for(c=0;c&lt;255;c++);     for(c=0;c&lt;255;c++);     for(c=0;c&lt;255;c++);     for(c=0;c&lt;255;c++);     for(c=0;c&lt;255;c++);}</pre>	208	1240	For loop increases only program memory
6	<pre>#include &lt;xs1.h&gt; void main(){char c; if(c){;} else{;}}</pre>	208	896	If else increases program memory

Table 5.2: Memory analysis table I

7	<pre>#include &lt;xs1.h&gt; #include &lt;stdio.h&gt; void main(){ printf("\n");}</pre>	1548	18808	Printf increases program memory. Stack memory increase may be due to internal variables of printf() function.
8	<pre>#include &lt;xs1.h&gt; #include &lt;stdio.h&gt; void main(){printf( "This is shyam krish...\n");}</pre>	1548	18828	Changing “ n” with “This is shyam krish...n” increases only the program memory.
9	<pre>#include &lt;xs1.h&gt; #include &lt;stdio.h&gt; void printhelp(); void main(){ printhelp();} void printhelp(){ printf("This is shyam krish...\n");}</pre>	1556	18832	Introducing a dummy function increases both stack and program memory
10	<pre>#include &lt;xs1.h&gt; #include &lt;stdio.h&gt; void printhelp(); void main(){ printhelp();} void printhelp(){ int c[1000]; printf("This is shyam krish...\n");}</pre>	5556	18836	Increase of 4000 bytes in account for an integer array of size 1000

Table 5.3: Memory analysis table II

11	<pre> #include &lt;xs1.h&gt; #include &lt;stdio.h&gt; void printhelp(); void main(){ printhelp(); printhelp(); printhelp();} void printhelp(){ int c[1000]; printf("This is shyam krish...\n");} </pre>	5556	18840	Calling the same function sequentially doesn't increase the stack memory.
12	<pre> #include &lt;xs1.h&gt; #include &lt;stdio.h&gt; void printhelp(); void main(){ par{ printhelp(); printhelp(); printhelp();}} void printhelp(){ int c[1000]; printf("This is shyam krish...\n");} </pre>	16716	18968	Parallel code triples the stack memory as compared with 10 and 11
13	<pre> #include &lt;xs1.h&gt; #include &lt;stdio.h&gt; void printhelp(); void main(){ par{ printhelp(); printhelp(); printhelp();}} void printhelp(){ int c[1000];} </pre>	12084	1024	Compare this with 8 and 10. Removing printf() statement inside printhelp() function has caused this change.
14	<pre> #include &lt;xs1.h&gt; void main(){ int c[16*1023];} </pre>	Not applicable	Not applicable	Program exceeds 64 kilo bytes and the compiler shows an error report in the console.

Table 5.4: Memory analysis table III

S.no	Code	Time taken	Remarks
1	<pre> int c[256],ts,te; char i,a[256],b[256]; timer t; for(i=0;i&lt;255;i++) {     a[i] = i;     b[i] = i;} t :&gt; ts; for(i=0;i&lt;255;i++) {     C[i] = a[i]*b[i];} t :&gt; te; printf("Time taken: %d\n",te-ts); </pre>	14297	<p>Multiplication of two char variables with some value. May be an average multiplication time of char variable can be obtained by dividing 14297 by 255 which equals 56 clock cycles.</p> <p>The variable “ts” represents starting time and “te” represents ending time.</p>
2	<pre> int c[256],ts,te; int i,a[256],b[256]; timer t; for(i=0;i&lt;255;i++) {     a[i] = i;     b[i] = i;} t :&gt; ts; for(i=0;i&lt;255;i++) {     c[i] = a[i]*b[i];} t :&gt; te; printf("Time taken: %d\n",te-ts); </pre>	13275	<p>Integer multiplication with same values as that of character variable in previous code.</p> <p>We expect integer multiplication to take more time than character multiplication. Hence this looks ambiguous and we again ran the same code.</p>
3	<pre> int c[256],ts,te; int i,a[256],b[256]; timer t; for(i=0;i&lt;255;i++) {     a[i] = i+255;     b[i] = i+255;} t :&gt; ts; for(i=0;i&lt;255;i++) {     c[i] = a[i]*b[i];} t :&gt; te; printf("Time taken: %d\n",te-ts); </pre>	13274	<p>Running the same code gave almost same answer. Even though XMOS is said to be deterministic, these ambiguous results are something to note.</p> <p>All these timings are not the worst case timing. We assumed the worst case timing will be obtained when we multiply highest values that a data type can hold.</p>

Table 5.5: Timing analysis table I

4	<pre> int c[256],ts,te; int i,a[256],b[256]; timer t; for(i=0;i&lt;255;i++) {     a[i] = 65535-i;     b[i] = 65535-i;} t :&gt; ts; for(i=0;i&lt;255;i++) {     c[i] = a[i]*b[i];} t :&gt; te; printf("Time taken: %d\n",te-ts); </pre>	13275	Changing higher values didn't change the result for multiplying two integer variables.
5	<pre> int c[256],ts,te; int i,a[256],b[256]; timer t; for(i=0;i&lt;255;i++) {     a[i] = 65535;     b[i] = 65535;} t :&gt; ts; for(i=0;i&lt;255;i++) {     c[i] = a[i]*b[i];} t :&gt; te; printf("Time taken: %d\n",te-ts); </pre>	13275	The worst case timing found out by multiplying integer variables with highest it can take (65535).
6	<pre> int c[256],ts,te; char i,a[256],b[256]; timer t; for(i=0;i&lt;255;i++) {     a[i] = 255;     b[i] = 255;} t :&gt; ts; for(i=0;i&lt;255;i++) {     c[i] = a[i]*b[i];} t :&gt; te; printf("Time taken: %d\n",te-ts); </pre>	14808	Worst case character multiplication. This worst case value is again lesser than the normal value obtained in the 1 <sup>st</sup> code.

Table 5.6: Timing analysis table II



7	<pre> int ts,te; char i,a[256],b[256],c[256]; timer t; for(i=0;i&lt;255;i++) {     if(i&lt;15){         a[i] = i;         b[i] = i;}     else{a[i] = 15;         b[i] = 15;}} t :&gt; ts; for(i=0;i&lt;255;i++) {c[i] = a[i]*b[i];} t :&gt; te; printf("Time taken: %d\n",te-ts); </pre>	14807	We tried to change the value of variables in some random fashion. Still we got the same result as worst case.
8	<pre> int ts,te,i; float a[256],b[256],c[256]; timer t; for(i=0;i&lt;255;i++) {a[i] = i*100/9; b[i] = i*111/11;} t :&gt; ts; for(i=0;i&lt;255;i++) {c[i] = a[i]*b[i];} t :&gt; te; printf("Time taken: %d\n",te-ts); </pre>	60610	Multiplication of float variables.
9	<pre> int ts,te,i; double a[256],b[256],c[256]; timer t; for(i=0;i&lt;255;i++) {a[i] = i*100/9; b[i] = i*111/11;} t :&gt; ts; for(i=0;i&lt;255;i++) {c[i] = a[i]*b[i];} t :&gt; te; printf("Time taken: %d\n",te-ts); </pre>	97638	Multiplication of double variables is takes more time than multiplication of float variable 60610.

Table 5.7: Timing analysis table III

10	<pre> int ts,te,i; double a[256],b[256],c[256]; timer t; for(i=0;i&lt;255;i++) { a[i] = i*100/9; b[i] = i*111/11;} t :&gt; ts; for(i=0;i&lt;255;i++) {c[i] = a[i]/b[i];} t :&gt; te; printf("Time taken: %d\n",te-ts); </pre>	348153	Division of double variables. This is much higher than 102608 which is the time taken for Division of float variables.
11	<pre> int ts,te,i; float a[256],b[256],c[256]; timer t; for(i=0;i&lt;255;i++) {a[i] = i*100/9; b[i] = i*111/11;} t :&gt; ts; for(i=0;i&lt;255;i++) {c[i] = a[i]/b[i];} t :&gt; te; printf("Time taken: %d\n",te-ts); </pre>	102608	Division of float variables.
12	<pre> int ts,te,i; int a[256],b[256],c[256]; timer t; for(i=0;i&lt;255;i++) {a[i] = i*100/9; b[i] = i*111/11;} t :&gt; ts; for(i=0;i&lt;255;i++) {c[i] = a[i]/b[i];} t :&gt; te; printf("Time taken: %d\n",te-ts); </pre>	Not applicable	Zero by zero division error

Table 5.8: Timing analysis table IV

13	<pre> int ts,te,i; int a[256],b[256],c[256]; timer t; for(i=0;i&lt;255;i++) {a[i] = i*10; b[i] = i*11 + 1;} t :&gt; ts; for(i=0;i&lt;255;i++) {c[i] = a[i]/b[i];} t :&gt; te; printf("Time taken: %d\n",te-ts); </pre>	15698	Division of integer variables.
14	<pre> int ts,te,i; int a[256],b[256],c[256]; timer t; for(i=0;i&lt;255;i++) {a[i] = i*10; b[i] = i*11 + 1;} t :&gt; ts; for(i=0;i&lt;255;i++) {c[i] = a[i];} t :&gt; te; printf("Time taken: %d\n",te-ts); </pre>	9449	Time taken for copying an integer variable. So while calculating time for division, multiplication this “copying time” or “register access time” is included.
15	<pre> int ts,te,i; float a[256],b[256],c[256]; timer t; for(i=0;i&lt;255;i++) {a[i] = i*10; b[i] = i*11 + 1;} t :&gt; ts; for(i=0;i&lt;255;i++) {c[i] = a[i];} t :&gt; te; printf("Time taken: %d\n",te-ts); </pre>	9449	Copying a float variable.

Table 5.9: Timing analysis table V

16	<pre> int ts,te,i; double a[256],b[256],c[256]; timer t; for(i=0;i&lt;255;i++) {a[i] = i*10; b[i] = i*11 + 1;} t :&gt; ts; for(i=0;i&lt;255;i++) {c[i] = a[i];} t :&gt; te; printf("Time taken: %d\n",te-ts); </pre>	10469	Copying a double variable with integer value.
17	<pre> int ts,te,i; double a[256],b[256],c[256]; timer t; for(i=0;i&lt;255;i++) {a[i] = i*100/9; b[i] = i*111/11;;} t :&gt; ts; for(i=0;i&lt;255;i++) {c[i] = a[i];} t :&gt; te; printf("Time taken: %d\n",te-ts); </pre>	10215	Copying a double variable with non-integer value.
18	<pre> int ts,te,i; char a[256],b[256],c[256]; timer t; for(i=0;i&lt;255;i++) {a[i] = i; b[i] = i+1;} t :&gt; ts; for(i=0;i&lt;255;i++) {c[i] = a[i];} t :&gt; te; printf("Time taken: %d\n",te-ts); </pre>	9195	Copying a character variable with integer value.

Table 5.10: Timing analysis table VI

19	<pre> int ts,te,i; char a[256],b[256],c[256]; timer t; for(i=0;i&lt;255;i++) {a[i] = i; b[i] = i+1;} t :&gt; ts; for(i=0;i&lt;255;i++) {c[i] = a[i]/b[i];} t :&gt; te; printf("Time taken: %d\n",te-ts); </pre>	15698	Division of character variables.
20	<pre> int ts,te,i; char a[256]; timer t; for(i=0;i&lt;255;i++) {a[i] = i;} t :&gt; ts; for(i=0;i&lt;255;i++) {printf("%d\n",a[i]);} t :&gt; te; printf("Time taken: %d\n",te-ts); </pre>	- 1530452759	Timer can be used to measure a maximum delay of 21 seconds
21	<pre> int ts,te,i; char a[256]; timer t; for(i=0;i&lt;255;i++) {a[i] = i;} t :&gt; ts; for(i=0;i&lt;25;i++) {printf("%d\n",a[i]);} t :&gt; te; printf("Time taken: %d\n",te-ts); </pre>	297099797	Printing 25 character values on the console takes as much as 2.97 seconds.

Table 5.11: Timing analysis table VII

22	<pre> int ts,te,i; char a[256]; timer t; for(i=0;i&lt;255;i++) {a[i] = i;} t :&gt; ts; printf("%d\n",a[5]); t :&gt; te; printf("Time taken: %d\n",te-ts); </pre>	10578311	0.1 second
23	<pre> int ts,te,i; char a[256]; timer t; for(i=0;i&lt;255;i++) {a[i] = i;} t :&gt; ts; for(i=0;i&lt;10;i++) {printf("%d\n",a[i]);} t :&gt; te; printf("Time taken: %d\n",te-ts); </pre>	126788051	1.26 seconds
24	<pre> int ts,te,i; char a[256]; timer t; for(i=0;i&lt;255;i++) {a[i] = i;} t :&gt; ts; for(i=0;i&lt;10;i++) {printf("This is shyam krish...\n");} t :&gt; te; printf("Time taken: %d\n",te-ts); </pre>	116687436	1.17 seconds. There is no memory access required to print “This is shyam krish. . . n”. So even though there are 26 characters to be displayed on console (including white spaces) it takes much lesser time than 2.97 seconds (to print 25 character variables)
25	<pre> t :&gt; ts; t :&gt; te; printf("Time taken: %d\n",te-ts); </pre>	5	“te” is assigned in the next line of assigning “ts”. It is hard to accept it takes 5 clock cycles for this. We tried to run this code multiple time and time taken didn’t change. It remained as 5 clock cycles.

Table 5.12: Timing analysis table VIII

time required to execute a section of code. This tool is limited to sequential logic. It is not advisable to determine timing reports for parallel logic.

We used XTA to analyze timing constraints in connected component analysis application. In CCA application, flash and Otsu's thresholding algorithm runs in parallel in step 1. In step 2, flash and CCA algorithm runs in parallel. CCA receives image data from flash pixel by pixel.

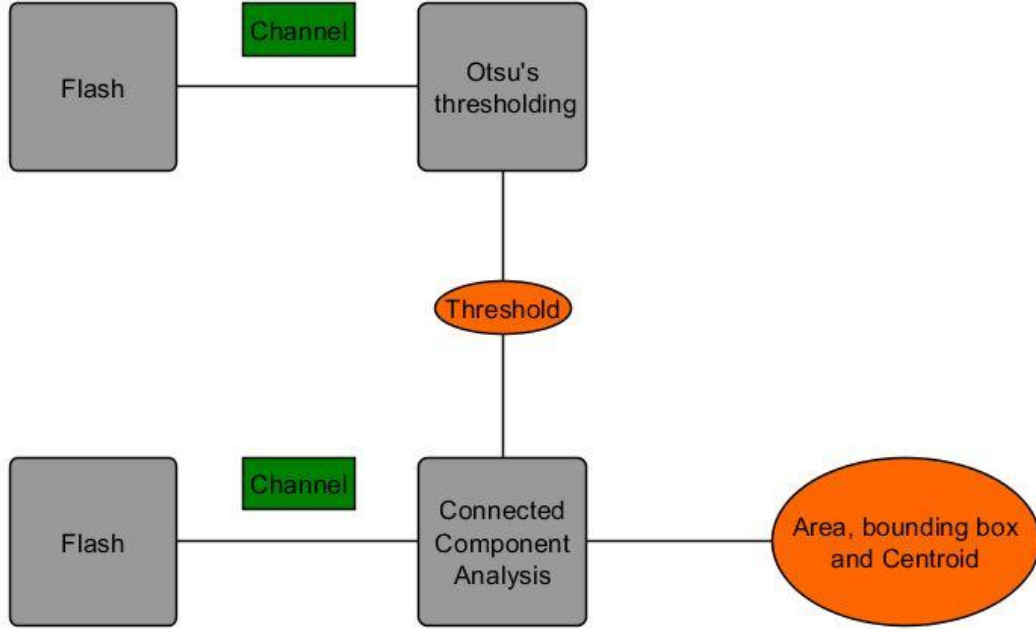


Figure 5.1: Flash, CCA and Otsu's threshold - Parallel processing

A grayscale image is stored in flash of height  $Ht$  and width  $Wt$ . In the above figure, in step 1, flash communicated to otsu's thresholding function through channel. They both run in parallel. After running for  $(Ht \times Wt)$  times (covering all the pixel of the image) we get otsu's threshold as the output. This threshold is used for converting the grayscale image to binary image. In step 2, we again read this grayscale image from flash pixel by pixel and send it to CCA function. Inside CCA function the threshold is applied to binarize the image. We are not storing any segment of the image in the processor. Every pixel goes to CCA function only once. After going through all the pixel from the image  $(Ht \times Wt)$ , CCA outputs area, bounding box and centroid information of the blobs present in the image. The whole process is shown in figure 5.1. Flashing image data and otsu's algorithm have already been discussed in detail in the

earlier chapters. CCA will be discussed in next chapter.

Using XTA we observed that, the time taken to fetch one pixel from flash is more than the time taken for that pixel to go through CCA function. It means most of the time CCA keeps waiting for the image data to come from flash. As XTA is not advisable to use for parallel logic, timers are used to analyze the timing constraint.



# CHAPTER 6

## Conclusion and Future work

In this chapter, we give the summary of the work done and discuss possible future work.

### 6.1 Conclusion

Compared several tracking algorithm in matlab and implemented frame subtraction method in XNXP multithreaded environment. We have implemented optimized CCA which reduced the memory requirements drastically. The developed object tracking system uses 5 threads simultaneously. We have achieved a frame rate of  $9\text{frames/s}$ . Currently we are detecting all the moving objects in the video and put a bounding box around the object with highest area. We extended this to show bounding box on objects with top three or more area. So the implemented code is apt for tracking a single moving object in the video perfectly as shown in the results.

### 6.2 Future work

Possible extensions are

1. It can be easily extended to track multiple moving objects by comparing the current and previous features obtained.
2. Camera slice was not available when we did the project. Once camera slice is made available, with an addition of one more thread we can make more sophisticated system.
3. Descriptor based algorithm can be applied instead of analyzing blobs in the binary image. Our conclusion is we need a better descriptor than HOG.

# APPENDIX A

## APPENDIX

### A.1 Introduction to XC programming

XC is a higher level language for hardware programmers like Verilog. Syntax for XC is similar to that of C. XC can be taken as extended version of C with port mapping, timers, parallelism etc... The following rules are different from C comparing it to XC. In XC

1. `I = I++;` // invalid statement
2. Arrays are implicitly passed by reference
3. Functions can return multiple values
4. A character array of size 10 can be reinterpreted as integer array of size 2. The remaining 2 bytes of data in the character are lost.

The following are few hardware commands used in XC programming.

1. `inp_port := register;`
2. `out_port <= register;`
3. `in port oneBit = XS1_PORT_1A;`
4. `out port counter = XS1_PORT_4A;`
5. `oneBit := register;`
6. `counter <= register;`
7. `oneBit when pinsneq(x) := x;`
8. `oneBit when pinseq(x) := x;`
9. `par{thread1;  
    thread2;} //8 parallel threads per XCore processor`
10. `timer t;`

11.  $t := \text{time}; \text{time} = \text{delay} + \text{time};$
12.  $t \text{ when timerafter}(\text{time}) := \text{void};$

1, 2 are syntax for input from a port and output to a port respectively. 3<sup>rd</sup> is syntax for using oneBit as 1 bit input port. 4<sup>th</sup> makes counter as a 4 bit output port. 5, 6 are examples for 1 and 2 respectively. Register can be any data type (int, char etc...). In general, when outputting to a n bit port, the least significant n bits of the output value are driven on the pins and the rest are ignored. 7<sup>th</sup> is a conditional waiting statement in which the data from oneBit port will be sampled and put into register x only when oneBit pin is not equal to the current value of x. It is known as conditional waiting statement because the execution will wait indefinitely at that statement until the pin becomes not equal to register x. 8<sup>th</sup> is another conditional waiting statement which checks for pin becoming equal to register x and loads it into x. 9<sup>th</sup> is a syntax for parallel logic in XC. 10<sup>th</sup> is the syntax for timer declaration. 11<sup>th</sup> is syntax where the value of the timer is loaded into the register time. The register time is an unsigned integer type. 12<sup>th</sup> is a delay statement, it is similar to conditional wait statement. It will keep on waiting till the timer variable t becomes equal to the delay plus time and load the value of t into void (means it is not used anywhere else). Timer variable t is an unsigned integer variable. The clock period is 10 ns. The maximum delay is 21 seconds (31 bit delay). The figure A.1 will help you understand 31 bit delay.

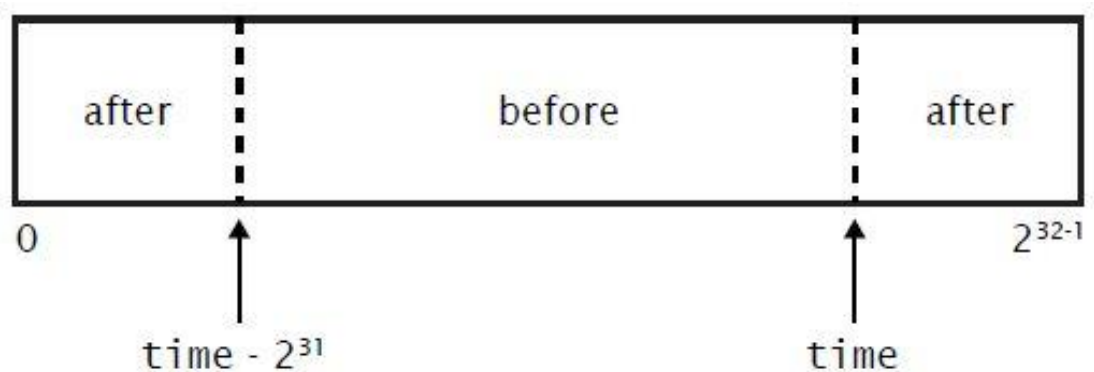


Figure A.1: Timer operation

“After” time is after the timer t value has been loaded into register time. For more information on syntax and example codes refer [reference].

## A.2 Matlab codes

### A.2.1 Background subtraction

This code is mainly derived from a demo available in Matlab. The name of the demo is “Detecting cars in a video of traffic”. We changed the video for tracking and varied different parameters available in the code. We could not get a clean binary image using this code. The details regarding the experiments are available in chapter 2.

### A.2.2 Frame subtraction

```
clear all
close all
clc

%% Multi media reader
%obj = mmreader('1.avi');
obj = mmreader('1.avi');
nframes = get(obj, 'NumberOfFrames');
nframes = 200;

% impyramid is used to reduce time complexity
I = read(obj,1);
%I = impyramid(read(obj, 1),'reduce');
%I = impyramid(I,'reduce');

% taggedCars is used to show the test video along with
    bounding boxes
% binaryvideo gives the sequence of binary frames generated
taggedCars = zeros([size(I,1) size(I,2) 3 nframes], class(I));
binaryvideo = zeros([size(I,1) size(I,2) nframes], class(I));

%% Parameters for tuning
```

```

[m n o] = size(I);
D = zeros(m,n); % Dynamic matrix
binary = zeros(m,n);
area_thresh = 80; % bwareaopen's area threshold
lamda = 1; % Dyanamic matrix's tail control
gamma = 1; % Currently previous frame
level = 25; % Thresholding, increasing it will reduce the
    noise level

%% Frame subtraction
%Frame subtraction is done using dynamic matrix. For every new
    frame
%or current frame, it's previous frame is used as it's
    background. Their
%difference will give us the difference image which is
    thresholded using
%the parameter level. After thresholding, we are assigning a
    value of lamda
%to every pixel where the difference image is greater than 1.
    This
%modification from normal binary image to dynamic matrix which
    takes lamda
%in place of "1", allows us to track constant intensity moving
    objects
%better. Elements in "Dynamic matrix" if greater than zero are
    taken as
%moving objects. Dynamic matrix is updated in the next
    iteration.

for i=2:nframes
    B = read(obj,1); %background subtraction
    % B = read(obj,i-gamma);
    %B = impyramid(read(obj,i-gamma),'reduce');
    %B = impyramid(B,'reduce'); % Background frame

```

```

CF = read(obj,i);
%CF = impyramid(read(obj,i),'reduce');
%CF = impyramid(CF,'reduce'); % Current frame
img = CF - B; % Difference image
img=img.*img; % Squaring each pixel so that all are positive
img = rgb2gray(img);
[m n] = size(img); % Better to assign it here as we
                    encountered changes

% Thresholding
for a=1:m
    for b=1:n
        if(img(a,b)>level)
            img(a,b)=255;
        else
            img(a,b)=0;
        end
    end
end

% Removes all the blobs which are having area lesser than
area_thresh
img = bwareaopen(img,area_thresh);

%Joins the blobs in the image together by filling in the gaps
between
%them and by smoothing their outer edges.
img = imclose(img,strel('rectangle',[5 5]));
img = imfill(img,'holes');

% Dynamic matrix
for j=1:m
    for k=1:n

```

```

        if(D(j,k) && (~img(j,k)))
            D(j,k) = D(j,k)-1;
        end
        if(img(j,k))
            D(j,k) = lamda;
        end
    end
end

%Binary video
for j=1:m
    for k=1:n
        if((~D(j,k)))
            binary(j,k) = 0;
        end
        if(D(j,k))
            binary(j,k) = 1;
        end
    end
end

binaryvideo(:, :, i) = 255.*binary;
taggedCars(:, :, :, i) = CF(:, :, :);
BI=binary;

% Bounding box
stats = regionprops(BI, {'Centroid', 'Area', 'BoundingBox'});
[imager1 num] = bwlabel(BI);
stats = regionprops(imager1, 'BoundingBox');
[M N] = size(stats);
for l = 1:M
    for m1 =
        floor(stats(l).BoundingBox(1)):floor(stats(l).BoundingBox(1))+stats(l).B
    for n1 =
        floor(stats(l).BoundingBox(2)):floor(stats(l).BoundingBox(2))+stats(l).B

```

```

x1 = floor(stats(l).BoundingBox(1));
y1 = floor(stats(l).BoundingBox(2));

if(m1 && n1 && x1 && y1)
    taggedCars(y1,m1,2,i) = 255;
    taggedCars(y1+stats(l).BoundingBox(4),m1,2,i) = 255;
    taggedCars(n1,x1,2,i) = 255;
    taggedCars(n1,x1+stats(l).BoundingBox(3),2,i) = 255;
end
end
end
end
i
end

%% Output videos
% implay(binaryvideo,20);
% implay(taggedCars,20);
%%
for j=1:200
    imshow(taggedCars(:,:,j));
    hold on;
    text(5, 18, strcat('Frame no: ',num2str(j)), 'Color','b',
        'FontWeight','bold', 'FontSize',20);
    pause;
    hold off;
end

```

### A.2.3 Main code for HOG tracking

HOG feature extraction has been done with the help of a code available online. We changed the parameters best suitable for tracking. Since the size of the object we are tracking on an average is  $30 \times 30$ , we are not splitting the image into two or more parts as described in HOG. So the length of the feature vector is 9. If we want to track a



large object, then we need to change number of windows in x and y directions. It will increase the length of the feature vector ( $9 \times num\_win\_x \times num\_win\_y$ ).

```
% Tracking using HOG implementation
clc;clear all;close all;
%Loading video file
addpath('./input_videos');

obj = mmreader('1.avi');
img = read(obj, 1);
%img = impyramid(img,'reduce');

% get the initial bounding box for the objects to be tracked
    from the user

[I2 rect] = imcrop(img);
init = ceil(rect);
initstate = init; %initial tracker
x = initstate(1);% x axis at the Top left corner
y = initstate(2);
initfeature = Feature(I2);

%%

nframes = get(obj, 'NumberOfFrames');
num = nframes;% number of frames
%-----
x = initstate(1);% x axis at the Top left corner
y = initstate(2);
w = initstate(3);% width of the rectangle
h = initstate(4);% height of the rectangle
%-----
for i = 2:num
    i
```

```

img = read(obj,i);
%img = impyramid(img,'reduce');

imgSr = img;% imgSr is used for showing tracking results.
%negx = NegsampleImg(img,initstate,0.8);
%posx = DetectsampleImg(img,initstate,1,1,1);
detectx = DetectsampleImg(img,initstate,8,5,2);
index = Predict1(detectx,initfeature,img);
x = detectx.sx(index);
y = detectx.sy(index);
w = detectx.sw(index);
h = detectx.sh(index);
initfeature = Feature(imcrop(img,[x y w h]));
initstate = [x y w h];
imshow(imgSr);

hold on;
rectangle('Position',initstate,'LineWidth',2,'EdgeColor','g');
text(5, 18, strcat('Frame no: ',num2str(i)), 'Color','b',
      'FontWeight','bold', 'FontSize',20);
set(gca,'position',[0 0 1 1]);
pause(0.00001);
hold off;
pause;
end

```

## A.2.4 HOG - function

```

%Image descriptor based on Histogram of Orientated Gradients
    for gray-level images. This code
%was developed for the work: O. Ludwig, D. Delgado, V.
    Goncalves, and U. Nunes, 'Trainable
%Classifier-Fusion Schemes: An Application To Pedestrian
    Detection,' In: 12th International IEEE

```

%Conference On Intelligent Transportation Systems, 2009, St.  
Louis, 2009. V. 1. P. 432-437. In  
%case of publication with this code, please cite the paper  
above.

```
function H=HOG(Im,nwin_x,nwin_y)
% nwin_x=3;%set here the number of HOG windows per bound box
% nwin_y=3;
B=9;%set here the number of histogram bins
[L,C]=size(Im); % L num of lines ; C num of columns
H=zeros(nwin_x*nwin_y*B,1); % column vector with zeros
m=sqrt(L/2);
if C==1 % if num of columns==1
    Im=im_recover(Im,m,2*m);%verify the size of image, e.g.
        25x50
    L=2*m;
    C=m;
end
Im=double(Im);
step_x=floor(C/(nwin_x+1));
step_y=floor(L/(nwin_y+1));
cont=0;
hx = [-1,0,1];
hy = -hx';
grad_xr = imfilter(double(Im),hx);
grad_yu = imfilter(double(Im),hy);
angles=atan2(grad_yu,grad_xr);
magnit=((grad_yu.^2)+(grad_xr.^2)).^.5;
for n=0:nwin_y-1
    for m=0:nwin_x-1
        cont=cont+1;
        angles2=angles(n*step_y+1:(n+2)*step_y,m*step_x+1:(m+2)*step_x);
        magnit2=magnit(n*step_y+1:(n+2)*step_y,m*step_x+1:(m+2)*step_x);
        v_angles=angles2(:);
```

```

v_magnit=magnit2(:);
K=max(size(v_angles));
%assembling the histogram with 9 bins (range of 20
    degrees per bin)
bin=0;
H2=zeros(B,1);
for ang_lim=-pi+2*pi/B:2*pi/B:pi
    bin=bin+1;
    for k=1:K
        if v_angles(k)<ang_lim
            v_angles(k)=100;
            H2(bin)=H2(bin)+v_magnit(k);
        end
    end
end
end

H2=H2/(norm(H2)+0.01);
H((cont-1)*B+1:cont*B,1)=H2;
end
end

```

## A.2.5 Feature - function

```

function [f] = Feature(I)
%[m n o] = size(I);
% X = zeros(m*n,o);
%
% X(:,1) = reshape(I(:,:,1),m*n,1);
% X(:,2) = reshape(I(:,:,2),m*n,1);
% X(:,3) = reshape(I(:,:,3),m*n,1);
% for i=1:n
%     for j=1:o
%         l = (i-1)*m + 1;
%         u = i*m;

```

```

%      X(o,l:u) = I(:,i,o);
%    end
% end
% centroids = CentroidImg(I);
% avg = mean(X);
% c = cov(X);
% c1 = [c(1,1) c(2,2)];
% s = skewness(X);
I = impyramid(I,'reduce');
I = impyramid(I,'reduce');
centroids = CentroidImg(I);
I1 = rgb2gray(I);
H = HOG(I1,2,2);
%f = [centroids;% avg c1 s H']; % trial 1
f = H';
end

```

## A.2.6 Detect Sample Image - function

Search boxes around the previous object location can be obtained the following function. Alpha and beta determines the search radius. Scale determines the coarseness of the search.

```

function samples =
    DetectsampleImg(img,initstate,alpha,beta,scale)
% Generates search boxes for object detection
% Inputs
% alpha -- x-alpha .. x .. x+alpha
% beta -- y-beta .. y .. y+beta
% img -- input image
% scale -- Increments.. It's important for smooth tracking
%         results. Smaller
%         the scale smoother the tracking
% Outputs

```

```

% sx, sy, sw and sh

[m n] = size(img);
x = initstate(1,1);
y = initstate(1,2);
w = initstate(1,3);
h = initstate(1,4);

j = 0;
for i=-alpha:scale:alpha
    if((1<(x+i)) && (x+i<(n-w)))
        j = j+1;
        X(1,j) = x + i;
    end
end
if(j==0)
    X = 1;
end

j = 0;
for i=-beta:scale:beta
    if((1<(y+i)) && (y+i<(m-h)))
        j = j+1;
        Y(1,j) = y + i;
    end
end
if(j==0)
    Y=1;
end

[mx nx] = size(X);
[my ny] = size(Y);

xy = nx*ny;

```

```

Xmat = ones(ny,nx)*diag(X);
Ymat = ones(nx,ny)*diag(Y);

samples.sx = reshape(Xmat,1,xy);
samples.sy = reshape(Ymat',1,xy);
samples.sw = ones(1,xy)*w;
samples.sh = ones(1,xy)*h;

if(min(size(samples.sx))==0)
    samples.sx = 1;
    samples.sy = 1;
    samples.sw = w;
    samples.sh = h;
end
end

```

## A.2.7 Predict - function

```

function index = Predict1(detectx,initfeature,img)
for i=1:max(size(detectx.sx))
    x = detectx.sx(i);
    y = detectx.sy(i);
    w = detectx.sw(i);
    h = detectx.sh(i);
    Ht = Feature(imcrop(img,[x y w h]));
    p(i) = 1/sum((Ht-initfeature).^2); %must be closer to
        positive samples
    % q(i) = sum((Ht-Htn).^2); %must be far away from negative
        samples
end
size(detectx.sx)
Decide = p;

```

```

idx = find(max(Decide)==Decide);
index = idx(1);
end

```

### A.3 XC codes

XMOS technology provides multi-core multi-threaded environment to develop high-performance systems in a C-like language, namely XC. A comparative study between Verilog and XC programming is available online [Xmos](#).

In order to understand XMOS programming, channel communication and parallel coding, we have explained this example code which is written for XC-1A board. XC-1A has 4 processors and each processor can run a maximum of 8 threads in parallel. It has 4 push buttons, 12 LEDs around the main chip and 4 more LEDs near push buttons. The output of the code is to generate glowing LEDs in a cycle. All the four processors run in parallel and they glow their respective LEDs when the control comes to them. Before starting to code any hardware, it is better to understand the port mapping of the board. XC-1A hardware manual XMOS helps you understand the port maps with good illustrations. Our main objective in the code is to glow all LEDs in cyclic fashion. In this process we will also understand how channels work. Figure A.2 below gives the port mapping to understand how to glow the LEDs.

XCore0 has control over LED I, LED II, LED III and select which color to glow green or red. XCore1 has control over LED IV, LED V, LED VI. Similarly XCore2 and XCore3 have their respective LEDs shown above. In port column, the pin X0D40 is connected to an 8 bit port's 4<sup>th</sup> bit. In order to glow LED I, we need to send value 16 to the PORT\_CLOCKLED\_0. LED II glows when '32' is sent to PORT\_CLOCKLED\_0 and LED III glows when 64 is sent to the port. A single processor has control of only 3 LEDs. After switching its LEDs one after the other, it has to send a signal to its neighboring processor asking it to switch its LEDs one after other. In order to make a cyclic LED, we need to communicate between different processors. This communication is done with the help of channels. A data type "chan" helps us to create a channel variable. Every channel variable must have two channel ends.



Pin	Port		Processor
	1b	8b	
XCore0			
X0D40		P8D4	PORT_CLOCKLED_0 [I]
X0D41		P8D5	PORT_CLOCKLED_0 [II]
X0D42		P8D6	PORT_CLOCKLED_0 [III]
X0D12	P1E0		PORT_CLOCKLED_SELG
X0D13	P1F0		PORT_CLOCKLED_SELR
XCore1			
X1D40		P8D4	PORT_CLOCKLED_1 [IV]
X1D41		P8D5	PORT_CLOCKLED_1 [V]
X1D42		P8D6	PORT_CLOCKLED_1 [VI]
XCore2			
X2D40		P8D4	PORT_CLOCKLED_2 [VII]
X2D41		P8D5	PORT_CLOCKLED_2 [VIII]
X2D42		P8D6	PORT_CLOCKLED_2 [IX]
XCore3			
X3D40		P8D4	PORT_CLOCKLED_3 [X]
X3D41		P8D5	PORT_CLOCKLED_3 [XI]
X3D42		P8D6	PORT_CLOCKLED_3 [XII]

Figure A.2: XC-1A port map

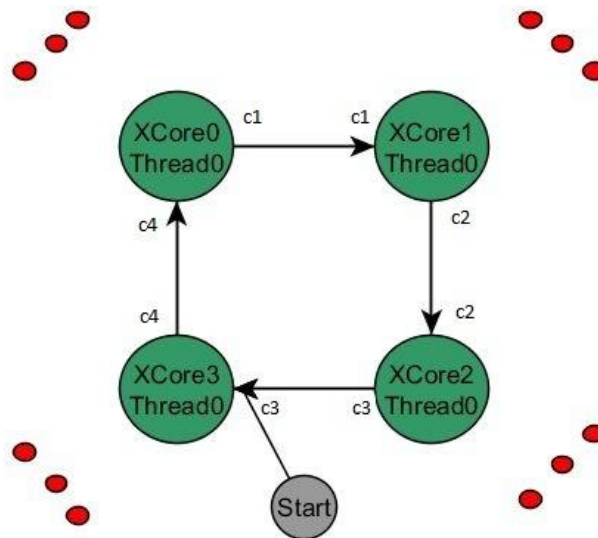


Figure A.3: Cyclic LED control sequence

In figure A.3, red circles are LEDs, arrows are channels used for communication between the processors and start is token given to any of the channel end to start the process. The following code segment will help understand “par” syntax and channel usage.

```
chan c0, c1, c2, c3;
par
{
on stdcore [0]:tokenFlash (c0, c1, cled0, PERIOD, 0);
on stdcore [1]: tokenFlash (c1, c2, cled1, PERIOD, 0);
on stdcore [2]: tokenFlash (c2, c3, cled2, PERIOD, 0);
on stdcore [3]: tokenFlash (c3, c0, cled3, PERIOD, 1);
}
```

Figure A.4: par usage in cyclic led code

The syntax for running 4 threads in parallel is shown in the figure A.4. Inside par statement, “on stdcore [0]” indicates that the thread runs in XCore0. There are totally 4 cores available in XC-1A. Single thread runs in every core available in the board. Note the usage of “chan” variable and compare it with figure 2. In XCore3 at the end of function, “1” is passed as opposed to “0” for rest of them. “1” behaves as the “start” token in figure 2. For more information go to appendix and understand the full code.

In the hardware manual for XC-1A you will be able to see lots of peripherals available on the board. LEDs, push buttons, speaker, SPI flash, I/O expansion areas and prototyping area. All these can be used once we understand how to send signal to each of them. For using any peripheral, first we need to study the hardware and then start developing application for the same.

The project folders will be given as a soft copy. Here we have given only the main codes which we have used in the project. Codes for testing flash, sdram and lcd are available in Jason.

### A.3.1 XC-1A LED - Example code

```
#include <platform.h>
#define PERIOD 20000000
out port cled0 = PORT_CLOCKLED_0;
```

```

out port cled1 = PORT_CLOCKLED_1;
out port cled2 = PORT_CLOCKLED_2;
out port cled3 = PORT_CLOCKLED_3;
out port cledG = PORT_CLOCKLED_SELG;
out port cledR = PORT_CLOCKLED_SELR;
out port ledscore = PORT_BUTTONLED;
in port button = PORT_BUTTON;

void tokenFlash (chanend left, chanend right, out port led,
    int delay, int
isMaster)
{
    timer tmr;
    unsigned t;
    int token;
    if (isMaster) /* master inserts token into ring */
        right <: 1;
    while (1)
    {
        left :> token; /* input token from left neighbor */
        led <: 16;
        tmr :> t;
        tmr when timerafter (t+ delay ) :> t;
        led <: 32;
        tmr when timerafter (t+ delay ) :> t;
        led <: 64;
        tmr :> t;
        tmr when timerafter (t+ delay ) :> t;
        led <: 0;
        right <: token; /* output token to right neighbor */
    }
}

void tokenFlash1 (chanend left, chanend right, out port led,
    int delay, int
isMaster)

```

```

{
timer tmr;
unsigned t;
int token;
int i=1;
if (isMaster) /* master inserts token into ring */
right <: 1;
while (1)
{int press;
int j=0;
button :> press;
left :> token; /* input token from left neighbor */
tmr :> t;
while(j<30){
led <: 16;
j++;
}
tmr :> t;
tmr when timerafter (t+ delay ) :> t;
while(j<50){
led <: 32;
j++;
}
/*tmr when timerafter (t+ delay ) :> t;*/
tmr when timerafter (t+ delay ) :> t;
while(j<70){
led <: 64;
j++;
}
if(press!=0xf){
ledscore <: i;
i++;
if(i>15){i=0;}
tmr :> t;

```

```

tmr when timerafter (t+ 5*delay) :> t;
}
tmr :> t;
tmr when timerafter (t+ delay) :> t;
led <: 0;
right <: token; /* output token to right neighbor */
}
}
int main(void){
chan c0, c1, c2, c3;
par
{
on stdcore [0]:{cledG <: 1;
cledR <: 1;
tokenFlash1 (c0, c1, cled0, PERIOD, 0);
}
on stdcore [1]: tokenFlash (c1, c2, cled1, PERIOD, 0);
on stdcore [2]: tokenFlash (c2, c3, cled2, PERIOD, 0);
on stdcore [3]: tokenFlash (c3, c0, cled3, PERIOD, 1);
}
return 0;
}

```

### A.3.2 Video Display

```

Video display
#include <platform.h>
#include "lcd.h"
#include <stdio.h>
#include <print.h>
#include "sprite.h"
#include "sdram.h"
#define BUF_WORDS (60/2*60)
#define MAX_FRAMES 5

```

```

unsigned frame[BUF_WORDS];
//unsigned read_buffer[BUF_WORDS];
lcd_ports ports = {
    XS1_PORT_1G, XS1_PORT_1F, XS1_PORT_16A, XS1_PORT_1B,
    XS1_PORT_1C, XS1_CLKBLK_1};

on tile[0]: sdram_ports ports1 = {
// XS1_PORT_16A, XS1_PORT_1B, XS1_PORT_1G, XS1_PORT_1C,
    XS1_PORT_1F, XS1_CLKBLK_1 };
XS1_PORT_16B, XS1_PORT_1J, XS1_PORT_1I, XS1_PORT_1K,
    XS1_PORT_1L, XS1_CLKBLK_2 };

static inline void add(unsigned x, unsigned y, unsigned line,
    unsigned buffer[]){
    if(line >= x && line < x + SPRITE_HEIGHT_PX)
        for(unsigned i=y;i<y + SPRITE_WIDTH_WORDS;i++)
            buffer[i] =
                frame[(line-x)*SPRITE_WIDTH_WORDS+(i-y)];
}

static inline void sub(unsigned x, unsigned y, unsigned line,
    unsigned buffer[]){
    if(line >= x && line < x + SPRITE_HEIGHT_PX)
        for(unsigned i=y;i<y + SPRITE_WIDTH_WORDS;i++)
            buffer[i] = BACK_COLOUR;
}

void demo(chanend c_lcd,chanend server){
    unsigned i=0,j=0,row_cnt,col_cnt;
    timer t;
    unsigned time_start,time_end;
    unsigned read_buffer[BUF_WORDS];

```

```

int x = 20,y = 20;

unsigned
    buffer[2][LCD_ROW_WORDS],black_frame[LCD_ROW_WORDS];
unsigned buffer_index = 0, update = 0;

for(unsigned i=0;i<LCD_ROW_WORDS;i++)
    buffer[0][i] = buffer[1][i] = black_frame[i]=
        BACK_COLOUR;
lcd_init(c_lcd);
i = 0;
while(1){
    j = j+1;
    j = j%MAX_FRAMES;
    //There are 256 cells in a row of a bank in SDRAM.
    row_cnt = j*BUF_WORDS*2/256; //actual address
        16-bit cell so multiplied with 2.
    col_cnt = (j*BUF_WORDS*2)%256;
    // Read the SDRAM into the read_buffer.
    sdram_buffer_read(server, 0, 0+row_cnt, 0+col_cnt,
        BUF_WORDS, read_buffer);

    //Wait until idle, i.e. the sdram had completed
        reading and hence the data is ready in the buffer.
    sdram_wait_until_idle(server, read_buffer);
    for(i=0;i<BUF_WORDS;i++){
        frame[i] = read_buffer[i];
    }
    t :> time_start;
    time_start = time_start + 100000000;
    i = 1;
    while(i)
    {
        add(x, y, 0, buffer[buffer_index]);
        lcd_req(c_lcd);

```

```

        lcd_update(c_lcd, buffer[buffer_index]);
        for(unsigned line=1;line<LCD_HEIGHT;line++){
            add(x, y, line, buffer[1 - buffer_index]);
            lcd_req(c_lcd);
            lcd_update(c_lcd, buffer[1 - buffer_index]);
            sub(x, y, line-1, buffer[buffer_index]);
            buffer_index = 1 - buffer_index;
        }
        sub(x, y, LCD_HEIGHT-1, buffer[buffer_index]);
        t :> time_end;
        if(time_end>=time_start)
        {
            i =0;
        }
    }
/*
    t :> time_start;
    time_start = time_start + 100000000;
    i = 1;
    while(i)
    {
        lcd_req(c_lcd);
        lcd_update(c_lcd, black_frame);
        for(unsigned line=1;line<LCD_HEIGHT;line++){
            lcd_req(c_lcd);
            lcd_update(c_lcd, black_frame);
        }
        t :> time_end;
        if(time_end>=time_start)
        {
            i =0;
        }
    }*/
}
}

```



```

void main() {
    chan c_lcd, c_sdram;

    par {
        sdram_server(c_sdram, ports1);
        lcd_server(c_lcd, ports);
        demo(c_lcd, c_sdram);
    }
}

```

The commented code is useful to control frame rate (decrease). It is also used to experiment the lcd working.

### A.3.3 Frame subtraction - main function

```

#include <platform.h>
#include "sdram.h"
#include "lcd.h"
#include "display_controller.h"
#include "transitions.h"
#include <stdio.h>
#include "loader.h"

on tile[0] : lcd_ports lcdports = {
    XS1_PORT_1G, XS1_PORT_1F, XS1_PORT_16A, XS1_PORT_1B,
    XS1_PORT_1C, XS1_CLKBLK_1 };
on tile[0] : sdram_ports sdramports = {
    XS1_PORT_16B, XS1_PORT_1J, XS1_PORT_1I, XS1_PORT_1K,
    XS1_PORT_1L, XS1_CLKBLK_2 };

#define IMAGE_COUNT (13)
//char images[IMAGE_COUNT][30] = { {"4001-tif.tga"},
    {"4010-tif.tga"}, {"4019-tif.tga"}, {"4037-tif.tga"}, {"4046-tif.tga"}, {"4
//          {"4064-tif.tga"},

```



```

frame_buffer_init(server, image[0]);

while(1){
    fb_index = transition_alpha_blend(server, frame_buffer,
        image[current_image], image[next_image], 3, fb_index);
    current_image = next_image;
    next_image = (current_image+1)%IMAGE_COUNT;
}

}

int main() {
    chan c_sdram, c_lcd, client, c_loader;

    par {
        on tile[0]:app(client, c_loader);
        on tile[0]:display_controller(client, c_lcd, c_sdram);
        on tile[0]:sdram_server(c_sdram, sdramports);
        on tile[0]:lcd_server(c_lcd, lcdports);
        on tile[1]:loader(c_loader, images, IMAGE_COUNT);
    }

    return 0;
}

```

### A.3.4 Binarization, cca call and put bounding box

```

// Binarization of the image, CCA and bounding box are done in
    this code

#include "transitions.h"
#include "lcd.h"
#include "display_controller.h"
#include "cca_slice.h"
#define thresh 5

```

```

//This function is used to put bounding box around the object
static void transition_alpha_blend_impl(chanend server,
    unsigned next_image_fb,
    unsigned image_from, unsigned image_to, unsigned s,
    unsigned line,int xmin[],int xmax[],int ymin[],int
    ymax[],int area[]) {
    unsigned dst[LCD_ROW_WORDS];
    unsigned Ht[SIZE],Wt[SIZE];
    unsigned max=0;//,area[SIZE];
    unsigned c;//,ht=15,wt=5;
    int xb,yb,index=0;
    char x1[SIZE];//={31,31,21,19,23,34,48,65,59,66,77,99,104};
    char y1[SIZE];//={17,17,22,27,33,41,49,60,70,81,97,113,135};
    char count=0;
    for(c=0;c<SIZE;c++)
    {
        Ht[c] = ymax[c]-ymin[c];
        Wt[c] = xmax[c]-xmin[c];
    }

    for(c=0;c<SIZE;c++){
        if(max<area[c]){
            index = c;
            max=area[c];
        }
    }
    image_read_line(server, line, image_to, dst); //Get dst[]
    from sdram. It gets the complete row of pixels
    wait_until_idle(server, dst); //Waits until the array is
    obtained from sdram
    xb = ymin[index];
    yb = xmin[index]/2; //Since two pixels are written at once
    on the lcd
    for (c = 0; c < SIZE; c++) {

```

```

if (area[c] > 30) { //area threshold. Only area
    greater than 30 can have bounding box
        count = count + 1;
        xb = ymin[c];
        yb = xmin[c]>>1; //two pixels at a time
            concept

if ((line == xb) || (line == (xb + Ht[c]))) {
    for (unsigned j = yb; j < (yb + Wt[c]
        / 2); j++) {
        if ((j>0) && (j <
            LCD_ROW_WORDS)) {
                dst[j] = 0xF800F800;
                //blue clor bounding
                box {b,g,r} =
                    {'11111', '000000', '00000'}
            }
        }
    }

if ((line < (xb + Ht[c])) && (line > xb)) {
    if ((yb>0) && (yb < LCD_ROW_WORDS)) {
        dst[yb] = 0xF800F800;
    }
    if (((yb + Wt[c] / 2)>0) && ((yb +
        Wt[c] / 2) < LCD_ROW_WORDS)) {
        dst[yb+Wt[c]/2] = 0xF800F800;
    }
}

}

}

```

```

// Display tagged video

image_write_line(server, line, next_image_fb, dst);
    //Display dst[], a row of modified pixels in the lcd. It
    is modified in order to put bbox
wait_until_idle(server, dst);
}

//above here are implimentations

//The below function is used for binarization and send pixel
values to cca module
void read_sdram_cca(unsigned dst[], unsigned src[], streaming
chanend cca, int line){

    char diff1, diff2;
    int ec, oc, eb, ob;
    unsigned c;
    for(c=0; c<LCD_ROW_WORDS; c++){
        oc = (0x000007E0 & dst[c])>>5; //extract odd
        pixel value (green component) from current
        image. {b,g,r}={'00000','11111','00000'}=
        0X07E0
        ec = (0x07E007E0 & dst[c])>>21; //extract even
        pixel value (green component) from current
        image. {b,g,r}={'00000','11111','00000'}=
        0X07E0
        ob = (0x000007E0 & src[c])>>5; //extract odd
        pixel value (green component) from background
        image. {b,g,r}={'00000','11111','00000'}=
        0X07E0
        eb = (0x07E007E0 & src[c])>>21; //extract even
        pixel value (green component) from background

```

```

    image. {b,g,r}={'00000','11111','00000'}=
    0X07E0
    if(oc>ob){
        diff1 = oc-ob; //diff1 holds absolute
            value of difference between the green
            component of the odd pixels
    }
    else
        diff1 = ob-oc;

    if(ec>eb){
        diff2 = ec-eb; //diff2 holds absolute
            value of difference between the green
            component of the even pixels
    }
    else
        diff2 = eb-ec;

    if(((diff1)>thresh) && ((diff2)>thresh)){ //Only
        if both diff1 and diff2 are greater than
        intensity threshold, it will be considered as
        object..
        dst[c] = 0xFFFFFFFF; //...pixel. It
            inherently acts as noise filter as salt
            and pepper noises won't show up in
            binary image...
    } // All other options also experimented:: for
        example diff1 can be greater than threshold
        but diff2 may not be. In such cases, it is
        logical to...
    else{ //.. consider dst[c] = 0xFFFF0000. But if
        we consider 0xFFFF0000 and 0x0000FFFF case,
        noise increases in the binary image. So the
        current..

```

```

        dst[c] = 0x00000000; //..implementation is
            good. Need not experiment
    }
}

for(c=0; c<LCD_ROW_WORDS; c++)
{
    for(char j=0; j<=1; j++)
    {
        if((line>5) && (line<265) && (c>5) &&
            (c<(LCD_ROW_WORDS-5))){//Near the edges
            of the image, I am avoiding any pixel
            being an object pixel..
        if(dst[c]==0xFFFFFFFF) //... It is done to
            avoid any run time error while issuing
            bounding box. You can change it and see
            o/p.
            cca <: 200; //Something greater than
                threshold for cca. CCA must
                recognise this as object pixel. Go
                to CCA code..
        else //... and see the threshold while
            labeling. 200 is greater than the
            intensity threshold in cca. You can
            give 255 instead of 200.
            cca <: 0;}
        else
            cca <: 0;
    }
}
}

```



```

unsigned transition_alpha_blend(chanend server, unsigned
    frame_buffer[2],
    unsigned image_from, unsigned image_to, unsigned frames,
    unsigned cur_fb_index) {
    unsigned next_fb_index;
    streaming chan ch_cca;
        unsigned src[LCD_ROW_WORDS];
        unsigned dst[LCD_ROW_WORDS];
        int
            xmin[SIZE], ymin[SIZE], xmax[SIZE], ymax[SIZE], area[SIZE];
    for (unsigned frame = 0; frame < frames; frame++) {
        unsigned fade = frame * MAX_ALPHA_BLEND / frames;
        unsigned line1, line2;
        next_fb_index = (cur_fb_index + 1) & 1;
        //CCA has to be included here to make it online
        for(line1=0;line1<SIZE;line1++){
            xmin[line1]=0;
            xmax[line1]=0;
            ymin[line1]=0;
            ymax[line1]=0;
            area[line1]=0;
        }
    par{ //CCA and sdram reads are happening in parallel. When a
        row of pixels are evaluated by CCA module, next row of
        pixels are obtained from sdram.
        {
            for(line1=0;line1<LCD_HEIGHT;line1++)
            {
                image_read_line(server, line1, image_to,
                    dst);
                wait_until_idle(server, dst);
                image_read_line(server, line1, image_from,
                    src); // Frame subtraction
                //image_read_line(server, line1, 0, src); //

```

```

        Background subtraction. Comment Frame
        subtraction and uncomment Background
        subtraction...
        wait_until_idle(server, src); //... to do
        background subtraction
        read_sdram_cca(dst,src,ch_cca,line1);

//Binary video can be seen if you uncomment the following two
//lines and comment the lines meant for tagged video in
function meant for bbox
        //image_write_line(server, line1,
        frame_buffer[next_fb_index], dst);
        //wait_until_idle(server, dst);
    }
}
conn_comp_anal(ch_cca,10,xmin,xmax,ymin,ymax,area); //CCA
}

for (unsigned line = 0; line < LCD_HEIGHT; line++) {
    transition_alpha_blend_impl(server,
        frame_buffer[next_fb_index],
        image_from, image_to, fade,
        line,xmin,xmax,ymin,ymax,area); //Outputs from CCA
        are sent to bounding box function
    }

    frame_buffer_commit(server, frame_buffer[next_fb_index]);
    cur_fb_index = next_fb_index;
}
next_fb_index = (cur_fb_index + 1) & 1;
frame_buffer_commit(server, frame_buffer[next_fb_index]);
return next_fb_index;
}

```

### A.3.5 Connected Component Analysis

```
//-----  
// Program: Single Pass Connected Component Analysis with  
//          Label Reuse  
// Input:   Grey image (read from the flash of XMOS board,  
//          XK-1A for example)  
// Output:  Curret and Previous Data table displaying features  
//          of connected components  
// References:  
// 1. Ni Ma, D.G.Bailey and C.T.Johnston, "Optimized Single  
//    pass connected component analysis",  
//    Image and Vision Computing Conference, Dec 2007.  
// 2. R.C.Gonzalez and R.E. Woods, 10.3.3-Optimum global  
//    thresholding using Otsu's method,  
//    Digital Image Processing, Pearson  
//    Prentice Hall, 2009.  
//-----  
  
//<-----HEADER  
//    FILES----->  
  
#include<xs1.h>  
#include<flashlib.h>  
#include<platform.h>  
#include<print.h>  
#include<stdio.h>  
#include<math.h>  
#include<stdlib.h>  
  
//<-----MACROS-----  
  
#define HEIGHT 272  
#define WIDTH 480
```

```

#define SIZE WIDTH/2+1+10

//<-----FUNCTION
    DECLARATION----->
//CCA label reuse
void update_translation(unsigned int, unsigned int, unsigned
    int, unsigned int,
        int[], int[], int[], int[], int[], int[], int[],
        int[], int[], int[],
        int[], int[], int[], int[], int[]); //Whenever new
        label is assigned
void update_stack(int, int, unsigned int, unsigned int,
    unsigned int, int[],
        int[], int[], int[], int[], int[], int[], int[]);
        //Whenever merging condition
void update_general(int, unsigned int, unsigned int, int[],
    int[], int[],
        int[], int[], int[], int[]); //If old label is
        assigned
void conn_comp_anal(streaming chanend, int, unsigned[],
    unsigned[], unsigned[],
        unsigned[], unsigned[]); //Connected component
        analysis using label reuse
void update_DE(int);

//<-----MAIN
    FUNCTION----->

//<-----UPDATE
    TRANSLATION----->
//Keeps track of connection between new label and the
    previously assigned labels

void update_translation(unsigned int x, unsigned int y,

```

```

unsigned int i,
        unsigned int j, int CD[], int CD_cogx[], int
        CD_cogy[], int CD_xmin[],
int CD_xmax[], int CD_ymin[], int CD_ymax[], int
        T[], int PD[],
int PD_cogx[], int PD_cogy[], int PD_xmin[], int
        PD_xmax[],
int PD_ymin[], int PD_ymax[]) {
if (y) {
    T[y] = x; //x = DE, y = C or x = new + 1, y = A|B|C
    //Area computation
    CD[x] = PD[y] + CD[x] + 1;
    PD[y] = 0;
    //COG computation
    CD_cogx[x] = PD_cogx[y] + CD_cogx[x] + j;
    PD_cogx[y] = 0;

    CD_cogy[x] = PD_cogy[y] + CD_cogy[x] + i;
    PD_cogy[y] = 0;
    //Bounding box computation
    if (CD_xmin[x] > PD_xmin[y])
        CD_xmin[x] = PD_xmin[y];
    if (CD_xmax[x] < PD_xmax[y])
        CD_xmax[x] = PD_xmax[y];
    if (CD_ymin[x] > PD_ymin[y])
        CD_ymin[x] = PD_ymin[y];
    if (CD_ymax[x] < PD_ymax[y])
        CD_ymax[x] = PD_ymax[y];
}
}

//<-----UPDATE
    STACK----->
//Whenever a merging condition occurs, Current data table is

```

```

    updated along with
//the labels are stored separately so that it can update
    Current merger at the end
//of every row

void update_stack(int x, int y, unsigned int i, unsigned int j,
    unsigned int SP, int CD[], int CD_cogx[], int
    CD_cogy[], int CD_xmin[],
    int CD_xmax[], int CD_ymin[], int CD_ymax[], int
    stack[]) {
    if (x == y) {
        CD[y] = CD[y] + 1;
        CD_cogx[y] = CD_cogx[y] + j;
        CD_cogy[y] = CD_cogy[y] + i;
    } else {
        stack[SP] = x; //x = DE, y = CE
        stack[SP + 1] = y;
        SP = SP + 2;
        CD[y] = CD[y] + CD[x] + 1;
        CD[x] = 0;
        //COG computation
        CD_cogx[y] = CD_cogx[y] + CD_cogx[x] + j;
        CD_cogx[x] = 0;
        CD_cogy[y] = CD_cogy[y] + CD_cogy[x] + i;
        CD_cogy[x] = 0;
        //Bounding box
        if (CD_xmin[y] > CD_xmin[x])
            CD_xmin[y] = CD_xmin[x];
        if (CD_xmax[y] < CD_xmax[x])
            CD_xmax[y] = CD_xmax[x];
        if (CD_ymin[y] > CD_ymin[x])
            CD_ymin[y] = CD_ymin[x];
        if (CD_ymax[y] < CD_ymax[x])
            CD_ymax[y] = CD_ymax[x];
    }
}

```

```

    }
} // stack(3,2), stack(2,1)---> [1 2 2 3]'

//<-----UPDATE
    GENERAL----->
//When old label is assigned and no merger condition is
    encountered

void update_general(int index, unsigned int i, unsigned int j,
    int CD[],
        int CD_cogx[], int CD_cogy[], int CD_xmin[], int
            CD_xmax[],
        int CD_ymin[], int CD_ymax[]) {
    CD[index] = CD[index] + 1;
    CD_cogx[index] = CD_cogx[index] + j;
    CD_cogy[index] = CD_cogy[index] + i;
    //Bounding box
    if (CD_xmin[index] > j)
        CD_xmin[index] = j;
    if (CD_xmax[index] < j)
        CD_xmax[index] = j;
    if (CD_ymin[index] > i)
        CD_ymin[index] = i;
    if (CD_ymax[index] < j)
        CD_ymax[index] = i;
}

//<-----LABEL
    SELECTION----->
//Selection tree is from the paper. At each end of the branch
    set of decisions are taken.
//If a new label is assigned, then Translation table update is
    required.
//If a merging condition occurs, then stack is updated...

```

```
//then at the end of the row Current Merger table is updated.
//If only a old label is assinged then the label's
    corresponding data must be updated.
```

```
int labell1(int data, int th) {
}

```

```
//<-----CCA----->
```

```
void conn_comp_anal(streaming chanend c, int threshold,
    unsigned xmin[],
        unsigned xmax[], unsigned ymin[], unsigned ymax[],
            unsigned area[]) {

```

```
//<-----VARIABLE
    DECLARATION----->
```

```
int T[SIZE] = { 0 }, PD[SIZE] = { 0 }, PM[SIZE] = { 0 },
    CD[SIZE] = { 0 },
        CM[SIZE] = { 0 }, stack[SIZE] = { 0
            };//,area[SIZE]={0}; //size of SIZE
//T - TRANSLATION TABLE, PD - PREVIOUS DATA TABLE, PM -
    PREVIOUS MERGER TABLE
//CD - CURRENT DATA TABLE, CM - CURRENT MERGER TABLE,
    STACK - STACK TABLE FOR UPDATING CM[]
int PD_cogx[SIZE] = { 0 }, CD_cogx[SIZE] = { 0 },
    PD_cogy[SIZE] = { 0 },
        CD_cogy[SIZE] = { 0 };
int cogx[SIZE] = { 0 }, cogy[SIZE] = { 0 };
int PD_xmin[SIZE] = { 0 }, CD_xmin[SIZE] = { 0 },
    PD_ymin[SIZE] = { 0 },
        CD_ymin[SIZE] = { 0 };
int PD_xmax[SIZE] = { 0 }, CD_xmax[SIZE] = { 0 },
    PD_ymax[SIZE] = { 0 },
```



```

        CD_ymax[SIZE] = { 0 };

//int
    xmin[SIZE]={0},xmax[SIZE]={0},ymin[SIZE]={0},ymax[SIZE]={0};
int RB[WIDTH + 2] = { 0 }; //size of WIDTH+1 with last
    element always be zero and also  $0^{th}$  element
    remains zero
//RB[0] = RB[WIDTH+1] = 0 ALWAYS
int A, B, C, AE, BE, CE, DE;
//A B C are derived from PM[] and RB[]
//AE BE DE are equivalent labels derived from A B C AND
    T[]
int data, i, j, newl, SP = 0, temp; //SP := Stack Pointer
//newl is label assigning variable

unsigned int count = 0, size;
//printf("I came in :)\n");
//Algorithm
for (i = 0; i < HEIGHT; i++) {
    SP = SP - 1;
    while (SP > 0) {
        CM[stack[SP - 1]] = CM[stack[SP]];
        SP = SP - 2;
    }

    SP = 0; //Stack pointer goes to zero for newl row
    for (int k = 0; k < SIZE; k++) {
        T[k] = 0;
        if (PD[k]) {
            area[count] = PD[k];
            cogx[count] = PD_cogx[k] / (PD[k]);
            cogy[count] = PD_cogy[k] / (PD[k]);
            xmin[count] = PD_xmin[k];
            xmax[count] = PD_xmax[k];

```

```

        ymin[count] = PD_ymin[k];
        ymax[count] = PD_ymax[k];
        count = count + 1;
    }
    PD[k] = CD[k];
    PD_cogx[k] = CD_cogx[k];
    PD_cogy[k] = CD_cogy[k];
    PD_xmin[k] = CD_xmin[k];
    PD_ymin[k] = CD_ymin[k];
    PD_xmax[k] = CD_xmax[k];
    PD_ymax[k] = CD_ymax[k];
    CD[k] = 0;
    CD_cogx[k] = 0;
    CD_cogy[k] = 0;
    CD_ymax[k] = 0;
    CD_ymin[k] = 0;
    CD_xmax[k] = 0;
    CD_xmin[k] = 0;
    PM[k] = CM[k];
    CM[k] = 0;
}
DE = 0;
newl = 0;
RB[0] = 0;
RB[WIDTH + 1] = 0;
for (j = 1; j < (WIDTH + 2); j++) {
    if (j < (WIDTH + 1)) {
        c := data; //get the data from the
                    channel or from an array
        if (data)
            //printf("data:%d ",data);
            //Get all the value of
            neighbourhood context through
            A = PM[RB[j - 1]]; //temp =

```

```

        RB[j-1];
B = PM[RB[j]];
C = PM[RB[j + 1]]; //Labels assigned
        based on the last row information
AE = T[A];
BE = T[B];
CE = T[C]; //Labels re-assigned simply
        to preserve connectivity alone
RB[j - 1] = DE; //Previous DE value is
        updated in Row Buffer
//new1 label is assigned to DE &&
        Label selection exactly as given in
        table
if (data > threshold) {
    if (DE && (!C)) {
        update_general(DE, i, j,
            CD, CD_cogx, CD_cogy,
            CD_xmin,
                CD_xmax,
                    CD_ymin,
                        CD_ymax);
        DE = DE;
    }

    if (DE && C && (!CE)) {
        update_translation(DE, C,
            i, j, CD, CD_cogx,
            CD_cogy,
                CD_xmin,
                    CD_xmax,
                        CD_ymin,
                            CD_ymax, T,
                                PD,
                                    PD_cogx,

```

```

PD_cogy,
PD_xmin,
PD_xmax,
PD_ymin,
PD_ymax);

DE = DE;
}

if (DE && C && CE) {
    update_stack(DE, CE, i, j,
        SP, CD, CD_cogx,
        CD_cogy,
        CD_xmin,
        CD_xmax,
        CD_ymin,
        CD_ymax,
        stack);

    DE = CE;
}

if ((!DE) && BE) {
    update_general(BE, i, j,
        CD, CD_cogx, CD_cogy,
        CD_xmin,
        CD_xmax,
        CD_ymin,
        CD_ymax);

    DE = BE;
}

if ((!DE) && (!BE) && CE &&
    (!A)) {
    update_general(CE, i, j,
        CD, CD_cogx, CD_cogy,

```

```

        CD_xmin,
            CD_xmax,
            CD_ymin,
            CD_ymax);

    DE = CE;
}

if ((!DE) && (!BE) && CE && A &&
    AE) {
    update_stack(AE, CE, i, j,
        SP, CD, CD_cogx,
        CD_cogy,
            CD_xmin,
            CD_xmax,
            CD_ymin,
            CD_ymax,
            stack);

    DE = CE;
}

if ((!DE) && (!BE) && CE && A &&
    (!AE)) {
    //update_translation(CE,A);
    //Translation is not
    required as it is a
    inner loop
    CD[CE] = CD[CE] + PD[A];
    PD[A] = 0;
    //COG
    CD_cogx[CE] = CD_cogx[CE]
        + PD_cogx[A];
    PD_cogx[A] = 0;
    CD_cogy[CE] = CD_cogy[CE]
        + PD_cogy[A];

```

```

PD_cogy[A] = 0;
//Bounding box
if (CD_xmin[CE] >
    PD_xmin[A])
    CD_xmin[CE] =
        PD_xmin[A];
if (CD_xmax[CE] <
    PD_xmax[A])
    CD_xmax[CE] =
        PD_xmax[A];
if (CD_ymin[CE] >
    PD_ymin[A])
    CD_ymin[CE] =
        PD_ymin[A];
if (CD_ymax[CE] <
    PD_ymax[A])
    CD_ymax[CE] =
        PD_ymax[A];
DE = CE;
}

if ((!DE) && (!BE) && (!CE) &&
    AE) {
    if (C)
        update_translation(AE,
            C, i, j, CD,
            CD_cogx,
            CD_cogy,
            CD_xmin,
            CD_xmax,
            CD_ymin,
            CD_ymax,
            T,
            PD,

```

```

PD_cogx,
PD_cogy,
PD_xmin,
PD_xmax,
PD_ymin,
PD_ymax);

else
    update_general(AE,
        i, j, CD,
        CD_cogx, CD_cogy,
        CD_xmin,
        CD_xmax,
        CD_ymin,
        CD_ymax);

DE = AE;
}

if ((!DE) && (!BE) && (!CE) &&
    (!AE)) {
    //Begining of new object
    new1 = new1 + 1;
    CM[new1] = new1;
    //Area
    CD[new1] = 1;
    //COG
    CD_cogx[new1] = j;
    CD_cogy[new1] = i;
    //Bounding box
    CD_ymin[new1] = i;
    CD_xmin[new1] = j;
    CD_ymax[new1] = 0;
    CD_xmax[new1] = 0;

    if (A) {

```

```

T[A] = new1;
CD[new1] = CD[new1]
    + PD[A];
PD[A] = 0;

CD_cogx[new1] =
    CD_cogx[new1] +
    PD_cogx[A];
PD_cogx[A] = 0;

CD_cogy[new1] =
    CD_cogy[new1] +
    PD_cogy[A];
PD_cogy[A] = 0;
//Bounding box
if (CD_xmin[new1] >
    PD_xmin[A])
    CD_xmin[new1]
        =
        PD_xmin[A];
if (CD_xmax[new1] <
    PD_xmax[A])
    CD_xmax[new1]
        =
        PD_xmax[A];
if (CD_ymin[new1] >
    PD_ymin[A])
    CD_ymin[new1]
        =
        PD_ymin[A];
if (CD_ymax[new1] <
    PD_ymax[A])
    CD_ymax[new1]
        =

```



```

PD_ymax[A];

}

if (B) {
    T[B] = new1;
    CD[new1] = CD[new1]
        + PD[B];
    PD[B] = 0;
    CD_cogx[new1] =
        CD_cogx[new1] +
        PD_cogx[B];
    PD_cogx[B] = 0;
    CD_cogy[new1] =
        CD_cogy[new1] +
        PD_cogy[B];
    PD_cogy[B] = 0;
    //Bounding box
    if (CD_xmin[new1] >
        PD_xmin[B])
        CD_xmin[new1]
            =
            PD_xmin[B];
    if (CD_xmax[new1] <
        PD_xmax[B])
        CD_xmax[new1]
            =
            PD_xmax[B];
    if (CD_ymin[new1] >
        PD_ymin[B])
        CD_ymin[new1]
            =
            PD_ymin[B];
    if (CD_ymax[new1] <
        PD_ymax[B])

```

```

        CD_ymax[new1]
        =
        PD_ymax[B];

    }

    if (C) {
        T[C] = new1;
        CD[new1] = CD[new1]
            + PD[C];
        PD[C] = 0;
        CD_cogx[new1] =
            CD_cogx[new1] +
            PD_cogx[C];
        PD_cogx[C] = 0;
        CD_cogy[new1] =
            CD_cogy[new1] +
            PD_cogy[C];
        PD_cogy[C] = 0;
        //Bounding box
        if (CD_xmin[new1] >
            PD_xmin[C])
            CD_xmin[new1]
            =
            PD_xmin[C];
        if (CD_xmax[new1] <
            PD_xmax[C])
            CD_xmax[new1]
            =
            PD_xmax[C];
        if (CD_ymin[new1] >
            PD_ymin[C])
            CD_ymin[new1]
            =
            PD_ymin[C];
    }

```

```

        if (CD_ymax[new1] <
            PD_ymax[C])
            CD_ymax[new1]
                =
                PD_ymax[C];
    }
    DE = new1;
}

else {
    DE = 0;
}

//Printing the array after label
    re-use algorithm
/*if(j==WIDTH)
    printf(" %d\n",DE);
else
    printf(" %d",DE);*/
} else {
    RB[j - 1] = DE;
}
}

}

size = SIZE - count;
for (j = 1; j < size; j++) {

    if (CD[j] > 0) {
        area[j + count - 1] = CD[j];
        cogx[j + count - 1] = CD_cogx[j] / CD[j];
        cogy[j + count - 1] = CD_cogy[j] / CD[j];
        xmin[j + count - 1] = CD_xmin[j];
        xmax[j + count - 1] = CD_xmax[j];
    }
}

```

```

        ymin[j + count - 1] = CD_ymin[j];
        ymax[j + count - 1] = CD_ymax[j];
    }
}
count = 1;

/*printf(" Obj.No \t Area \t COG \t Bound_x \t Bound_y
    \n");

for(j=1;j<SIZE;j++){
if(area[j]){
printf(" %d \t\t %d \t (%d,%d) \t (%d,%d) \t (%d,%d)
    \n",count,area[j],cogx[j],cogy[j],xmin[j],xmax[j],ymin[j],ymax[j]);
count = count+1;}
if(PD[j]){

printf(" %d \t\t %d \t (%d,%d) \t (%d,%d) \t (%d,%d)
    \n",count,PD[j],cogx[j],cogy[j],xmin[j],xmax[j],ymin[j],ymax[j]);

count = count+1;}
}

printf("\nTotal number of objects = %d\n",count-1);*/
}

```

## REFERENCES

1. **Dalal, N.** and **B. Triggs**, Histograms of oriented gradients for human detection. *In Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, volume 1. 2005. ISSN 1063-6919.
2. **Gonzalez, R. E. W., Rafael C.** and **S. L. Eddins**, *Digital image processing using MATLAB*. Pearson Education India, Reading, MA, 2004.
3. **Jason, A. S. ()**. Controlling external sdram. URL [https://github.com/xcore/sc\\_sdram\\_burst](https://github.com/xcore/sc_sdram_burst).
4. **Johnston, C.** and **D. Bailey**, Fpga implementation of a single pass connected components algorithm. *In Electronic Design, Test and Applications, 2008. DELTA 2008. 4th IEEE International Symposium on*. 2008.
5. **Liu, S., A. Papakonstantinou, H. Wang,** and **D. Chen**, Real-time object tracking system on fpgas. *In Application Accelerators in High-Performance Computing (SAAHPC), 2011 Symposium on*. 2011.
6. **Ludwig, O.** (2010). Hog descriptor for matlab. URL <http://www.mathworks.in/matlabcentral/fileexchange/28689-hog-descriptor-for-matlab>.
7. **Ma, N., D. Bailey,** and **C. Johnston**, Optimised single pass connected components analysis. *In ICECE Technology, 2008. FPT 2008. International Conference on*. 2008.
8. **Mahmoud, I., H. Abd El-Halym,** and **S.-D. Habib**, Hardware development and implementation of an object tracking algorithm. *In Microelectronics, 2003. ICM 2003. Proceedings of the 15th International Conference on*. 2003.
9. **MathWorks** (1994). Detecting cars in a video of traffic. URL <http://www.mathworks.in/products/image/examples.html?file=/products/demos/shipping/images/ipextraffic.html>.
10. **NUMONYX** (2008). Datasheet of flash m25p10-a. URL <http://www.xmos.com/references/m25p10a>.
11. **Pulli, K., A. Baksheev, K. Korniyakov,** and **V. Eruhimov** (2012). Realtime computer vision with opencv. *Queue*, **10**(4), 40:40–40:56. ISSN 1542-7730. URL <http://doi.acm.org/10.1145/2181796.2206309>.
12. **Shah, M.** (2012). Histogram of oriented gradients(hog). URL <http://www.youtube.com/watch?v=0Zib1YEE4LU>.
13. **XMOS ()**. Xc-1a hardware manual. URL <https://www.xmos.com/download/final/XC-1A-Hardware-Manual%281.2%29.pdf?time=1366795753>.

14. **Xmos** (). Xc for verilog designers. URL <https://www.xmos.com/download/final/XC-for-Verilog-Designers-Whitepaper%281.0%29.pdf?time=1366795703>.
15. **XMOS** (). xtimecomposer user guide. URL <https://www.xmos.com/download/final/xTIMEcomposer-User-Guide%28X3766B%29.pdf?time=1366795935>.
16. **Yang, T., Q. Pan, J. Li, and S. Li**, Real-time multiple objects tracking with occlusion handling in dynamic scenes. *In Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, volume 1. 2005. ISSN 1063-6919.