AN FPGA BASED HIGH PERFORMANCE OPTICAL FLOW HARDWARE DESIGN FOR AUTONOMOUS MOBILE ROBOTIC PLATFORMS

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

GÖKHAN KORAY GÜLTEKİN

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
ELECTRICAL AND ELECTRONICS ENGINEERING

SEPTEMBER 2010

Approval of the thesis:

AN FPGA BASED HIGH PERFORMANCE OPTICAL FLOW HARDWARE DESIGN FOR AUTONOMOUS MOBILE ROBOTIC PLATFORMS

submitted by **GÖKHAN KORAY GÜLTEKİN** in partial fulfillment of the requirements for the degree of **Master of Science in Electrical and Electronics Engineering Department, Middle East Technical University** by,

Prof. Dr. Canan Özgen
Dean, Graduate School of **Natural and Applied Sciences** ______________

Prof. Dr. İsmet Erkmen
Head of Department, **Electrical and Electronics Engineering** ______________

Asst. Prof. Dr. Afşar Saranlı
Supervisor, **Electrical and Electronics Engineering Dept., METU** ______________

Examining Committee Members:

Prof. Dr. M. Kemal Leblebicioğlu
Electrical and Electronics Engineering Dept., METU ______________

Asst. Prof. Dr. Afşar Saranlı
Electrical and Electronics Engineering Dept., METU ______________

Prof. Dr. Gözde Bozdağı Akar
Electrical and Electronics Engineering Dept., METU ______________

Assoc. Prof. Dr. A. Aydın Alatan
Electrical and Electronics Engineering Dept., METU ______________

Asst. Prof. Dr. Melik Dölen
Mechanical Engineering Dept., METU ______________

Date: ______________

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Last Name: GÖKHAN KORAY GÜLTEKİN

Signature :

ABSTRACT

AN FPGA BASED HIGH PERFORMANCE OPTICAL FLOW HARDWARE DESIGN FOR AUTONOMOUS MOBILE ROBOTIC PLATFORMS

Gültekin, Gökhan Koray
M.S., Department of Electrical and Electronics Engineering
Supervisor : Asst. Prof. Dr. Afşar Saranlı

September 2010, 91 pages

Optical flow is used in a number of computer vision applications. However, its use in mobile robotic applications is limited because of the high computational complexity involved and the limited availability of computational resources on such platforms. The lack of a hardware that is capable of computing optical flow vector field in real time is a factor that prevents the mobile robotics community to efficiently utilize some successful techniques presented in computer vision literature. In this thesis work, we design and implement a high performance FPGA hardware with a small footprint and low power consumption that is capable of providing over-realtime optical flow data and is hence suitable for this application domain. A well known differential optical flow algorithm presented by Horn & Schunck is selected for this implementation. The complete hardware design of the proposed system is described in details. We also discuss the design alternatives and the selected approaches together with a discussion of the selection procedure. We present the performance analysis of the proposed hardware in terms of computation speed, power consumption and accuracy. The designed hardware is tested with some of the available test sequences that are frequently used

for performance evaluations of the optical flow techniques in literature. The proposed hardware is capable of computing optical flow vector field on 256x256 pixels images in 3.89ms which corresponds to a processing speed of 257 fps. The results obtained from FPGA implementation are compared with a floating-point implementation of the same algorithm realized on a PC hardware. The obtained results show that the hardware implementation achieved a superior performance in terms of speed, power consumption and compactness while there is minimal loss of accuracy due to the fixed point implementation.

Keywords: FPGA, embedded vision for mobile robotics, optical flow, real time image processing, Horn and Schunck algorithm

ÖZ

OTONOM GEZGİN ROBOTİK PLATFORMLARI İÇİN FPGA TABANLI YÜKSEK PERFORMANSLI BİR OPTİK AKIŞ DONANIM TASARIMI

Gültekin, Gökhan Koray
Yüksek Lisans, Elektrik ve Elektronik Mühendisliği Bölümü
Tez Yöneticisi : Yrd. Doç. Dr. Afşar Saranlı

Eylül 2010, 91 sayfa

Optik akış, birçok bilgisayar ile görme uygulamalarında kullanılmaktadır. Ancak, yüksek işlem karmaşıklığı ve kısıtlı işlem kaynakları sebebiyle gezgin robot uygulamalarında kullanımı limitli olmaktadır. Gerçek zamanda optik akış hesaplayan donanımların eksikliği, bilgisayar ile görme literatüründe sunulan birçok başarılı çalışmanın gezgin robot araştırmalarında kullanılmasını kısıtlayan bir etmendir. Bu tez çalışmasında, gezgin robot uygulamaları için, gerçek zaman üstü optik akış verisi sağlayabilen küçük boyutlarda ve düşük güç tüketen yüksek performanslı bir FPGA donanım tasarımı yapılmaktadır. Horn ve Schunck tarafından sunulan tanınmış bir diferansiyel optik akış algoritmasının uygulaması yapılmıştır. Önerilen donanım tasarımının tamamı detaylı bir şekilde açıklanmıştır. Ayrıca, tasarım alternatifleri ve seçilen yöntemler gerekçekçeleriyle birlikte tartışılmıştır. Önerilen donanımın performans analizleri; işlem hızı, güç tüketimi ve doğruluk bakımından sunulmuştur. Tasarlanan donanım, literatürde optik akış yöntemlerinin performans değerlendirmesinde sıklıkla kullanılan mevcut bazı test dizileri ile sınanmıştır. Önerilen donanım, 256x256 pikselden oluşan görüntüler üzerinde optik akış hesabını saniyede 257 kare işlemeye

karşılık gelen 3.89ms sürede hesap edebilmektedir. FPGA uygulamasından elde edilen sonuçlar, aynı algoritmanın PC donanımı üzerinde kayan nokta uygulamasından elde edilen sonuçlarla karşılaştırılmıştır. Elde edilen sonuçlar donanım uygulamasının, sabit nokta gösterim uygulamasından kaynaklanan doğruluktaki makul bir azalmaya karşılık, hız, güç tüketimi ve az yer kaplama bakımından üstün bir performans gösterdiğini ortaya koymuştur.

Anahtar Kelimeler: FPGA, gezgin robotik için gömülü bilgisayar görme, optik akış, gerçek zamanlı görüntü işleme, Horn ve Schunck algoritması

to my loving mother, dear father and brother

ACKNOWLEDGMENTS

I would like to express my deep appreciation and sincere gratitude to my supervisor Dr. Afşar Saranlı for his leading guidance, encouragement, and continuous support from beginning to the end of my M.S. study. His suggestions during our research meetings played a tremendous role in helping me to broaden my view and knowledge. He showed me different ways to approach a research problem and the need to be persistent to accomplish any goal. He let me take part in the SensoRHex project which was a great opportunity for me to extend my academic and technical skills. He also made the Rolab (Laboratory of Robotics and Autonomous Systems) a wonderful workplace by providing us many new equipments we need.

I wish to express my deep sense of gratitude to Dr. Uluç Saranlı for his guidance and being a source of inspiration for me in the conduct of my thesis work and our research project. I also would like to thank Dr. Kemal Leblebicioğlu for conducting the ULİSAR project in the Rolab which added much to my experience. I should also extend my sincere thanks to Dr. Hamit Erdem. I learnt a lot from him, which I am sure will be useful in different stages of my life. I thank to Engin Çiftçi from Karel and Hakan Aydın from Linera for their help on pointing out the problems occurred during the FPGA design.

I literally consider myself a lucky person to work with the amazing group of people in Rolab. I am very thankful to Mert Ankaralı, Emre Ege, Orkun Öğücü, Ferit Üzer, Emre Akgül, Ege Saygıner and all other members for sharing me their time and knowledge. Further thanks should also go to the all members of BDRL (Bilkent Dexterous Robotics and Locomotion), especially Ömür Arslan and Tolga Özaslan.

I would like to thank the Scientific and Technological Research Council of Turkey (TÜBİTAK) for awarding me their prestigious master of science studies scholarship.

I apologize from the people who I should also have included their names here for their contributions but may forgot because of the rush I am in. I send my thanks to all

individually.

Finally, yet the most importantly, nothing is adequate to express my heartfelt feelings to my beloved family forever. None of this would have been even possible without the love and patience of them. I owe a great many thanks to my loving mother(Elif Gültekin), my dear father(Mahmut Gültekin) and my sweetie brother(Korcan Emre Gültekin) for their undying love, unconditional support, encouragement and their trust in me.

TABLE OF CONTENTS

ABSTRACT iv

ÖZ vi

DEDICATION viii

ACKNOWLEDGMENTS ix

TABLE OF CONTENTS xi

LIST OF TABLES xiv

LIST OF FIGURES xvi

LIST OF ABBREVIATIONS xxi

CHAPTERS

1 INTRODUCTION 1

1.1 Motivation 1

1.2 Literature Survey 3

1.3 Contributions 6

1.4 Outline of the Thesis 7

2 BACKGROUND ON OPTICAL FLOW COMPUTATION 9

2.1 Optical Flow 9

2.2 Differential Optical Flow Computation Methods 9

2.2.1 Horn-Schunck Optical Flow Computation Algorithm 10

2.2.1.1 Pre-Assumptions 10

2.2.1.2 Optical Flow Constraint Equation 11

2.2.1.3 Horn & Schunck Smoothness Constraint . 12

2.2.1.4 The Numerical Solution 14

3 FPGA HARDWARE PLATFORM FOR VISION 17
3.1 Requirements of the FPGA Hardware Platform 17
3.2 FPGA Development Platform Specifications 19
3.3 FPGA Architecture 23
3.3.1 Logic Elements 25
3.3.2 Embedded Memory Blocks 26
3.3.3 Embedded Multipliers 26
3.4 FPGA Development Environment and Design Flow 27
3.4.1 Quartus II 27
3.5 FPGA Design Flow 27
4 PROPOSED FPGA HARDWARE DESIGN FOR OPTICAL FLOW . 30
4.1 High Level Block Diagram 30
4.2 Reset Circuitry 33
4.3 Clocking Circuitry 34
4.4 Memory 36
4.4.1 SSRAM Memory Controller 37
4.4.2 Direct Memory Access Module 38
4.4.2.1 FIFO Buffers 44
4.5 Spatiotemporal Gradient and Optical Flow Vectors Local Average Computation 45
4.6 Optical Flow Computation 50
4.7 PC Communication 52
4.7.1 UART Controller 53
4.7.2 RS232 to SSRAM Data Transfer Module 53
4.7.3 SSRAM to RS232 Data Transfer Module 56
5 HARDWARE DESIGN PERFORMANCE ANALYSIS AND TEST RESULTS 58
5.1 Tests With Standard Image Sequences 58
5.1.1 Description of Standard Data Set 58
5.1.2 Performance Measure 59
5.1.3 Results 60

5.1.3.1 Rubik's Cube Sequence 60
5.1.3.2 Hamburg Taxi Sequence 67
5.1.3.3 Translating Tree Sequence 70
5.2 Performance Analysis of Designed Hardware 73
5.2.1 Resource Usage 73
5.2.2 Power Consumption 77
5.2.3 Computation Time 81
5.3 Comments on Analysis & Results 82
6 CONCLUSION 85
6.1 Future Work 87
REFERENCES 89

LIST OF TABLES

TABLES

Table 3.1 Cyclone II FPGA family features[2]. The development board we use includes EP2C70 device which has the highest resources available in the Cyclone II family. 23

Table 4.1 Operating clock frequencies of modules. 200MHz is used for the modules that access SSRAM memory to reduce memory bottleneck. 50MHz is used for the rest. 35

Table 4.2 Properties of memories available on DE2-70 board. SSRAM is preferred because of its higher bandwidth although its capacity is lower than the SDRAM memory. 36

Table 4.3 Maximum data rates of some communication protocols. 52

Table 5.1 Total error rates of Rubik's cube sequence 65

Table 5.2 Partial error rates of Rubik's cube sequence caused by the approximation in computation of local averages of optical flow vectors. 65

Table 5.3 Error rates of Rubik's cube sequence versus number of fraction bits used to represent the optical flow vector values. 65

Table 5.4 Maximum operating frequency of a signed division operation versus the word length of its operands. 67

Table 5.5 Error rates of Hamburg Taxi sequence caused by the fixed point implementation. 68

Table 5.6 Error rates of Translating Tree sequence 71

Table 5.7 Resource usage of the overall design and available resources on the FPGA device. 73

Table 5.8 Resource usage of individual design modules 75

Table 5.9 Total power dissipation 77

Table 5.10 Current drawn from supply pins 78

Table 5.11 Power consumption of individual modules 79

LIST OF FIGURES

FIGURES

Figure 2.1 Numerical computation of E_x using first order difference of 8 pixels. 15

Figure 2.2 Numerical computation of E_y using first order difference of 8 pixels. 15

Figure 2.3 Numerical computation of E_t using first order difference of 8 pixels. 16

Figure 2.4 Weight matrix for estimating local averages of optical flow vectors . 16

Figure 3.1 The FPGA product portfolio of Altera with a comparison of their available resources, performances, power consumptions and costs. 20

Figure 3.2 Altera DE2-70 FPGA development board[1] 21

Figure 3.3 General hardware architecture of FPGA devices[21]. Functional plane is used to implement logic functions defined by user by logic cells and routing channels. Programming plane stores the LUT values in SRAMs for configuration of functional plane at start up. 24

Figure 3.4 Components of a logic element block[2]. Each logic element can be used to implement combinational, sequential logic functions and arithmetic operations. 25

Figure 3.5 FPGA Design Flow Diagram 28

Figure 4.1 Block diagram of the designed system 31

Figure 4.2 Data flow diagram of the designed system 32

Figure 4.3 Reset signal generator module. Two reset signals are generated with different release times. They are used to synchronize the start up of modules designed. 34

Figure 4.4 Phase Locked Loop (PLL) module generates the required clock signals for the operation of designed modules. There are two clocks generated which are 200MHz and 50MHz from the input clock of 50MHz. 35

Figure 4.5 SSRAM memory controller module terminals. 37

Figure 4.6 Direct Memory Access (DMA) module terminals. DMA is used to handle memory read/write operations required for the operation of optical flow computations. It helps increasing the bus utilization and reduces the memory bottleneck. 39

Figure 4.7 Layout of stored data in SSRAM for two frames of 256x256 pixels. Image frames are stored starting from the first address location and optical flow vectors are stored beginning from the 200,000th address location of SSRAM. 40

Figure 4.8 Layout of pixel FIFO buffers. Each location of FIFO buffers is 32 bits in width and stores 2 consecutive pixels from frame 1, and 2 consecutive pixels from frame 2. 41

Figure 4.9 Layout of optical flow vectors in FIFO buffers. There is a phase difference of 1 line of vectors between three FIFO buffers. 42

Figure 4.10 DMA module FSM states 43

Figure 4.11 Dual clock FIFO buffer module terminals. 44

Figure 4.12 Spatiotemporal gradient computation module terminals. This module computes both spatiotemporal gradient values and the local averages of optical flow vectors. 45

Figure 4.13 Spatiotemporal gradients data representation in 11 bits fixed point format. 2 bits fraction is enough for representing the gradient results without any accuracy lost. 46

Figure 4.14 The weight matrix used for estimating local averages of optical flow vectors on FPGA. The weights given in Fig. 2.4 are modified to simplify the division operation and increase the accuracy. 46

Figure 4.15 Optical flow vector local average data representation in 11 bits fixed point format. 3 bits fraction is enough for representing the vector local average results without any accuracy lost. 47

Figure 4.16 Spatiotemporal gradient computation module FSM states 49

Figure 4.17 Optical flow computation module 50

Figure 4.18 Optical flow computation module data flow diagram 51

Figure 4.19 RS232 controller module terminals. 53

Figure 4.20 RS232 to SSRAM data transfer module terminals. This module acts as an interface for transferring image frames read from RS232 controller and written to SSRAM memory. 54

Figure 4.21 RS232 to SSRAM data transfer module FSM states 55

Figure 4.22 SSRAM to RS232 data transfer module 56

Figure 4.23 SSRAM to RS232 data transfer module FSM states 57

Figure 5.1 Output of the E_x gradient computation on FPGA hardware. The pixel values are inverted to get better visualization. White represents the lowest value and black represents the highest value. 61

Figure 5.2 Output of the E_y gradient computation on FPGA hardware. The pixel values are inverted to get better visualization. White represents the lowest value and black represents the highest value. 61

Figure 5.3 Output of the E_t gradient computation on FPGA hardware. The pixel values are inverted to get better visualization. White represents the lowest value and black represents the highest value. 62

Figure 5.4 1st frame of Rubik's cube sequence. The turntable rotates counter-clockwise with the Rubik's cube on top of it. 62

Figure 5.5 Optical flow vectors computed on FPGA hardware for Rubik's cube sequence. 63

Figure 5.6 Error histogram of optical flow vectors for Rubik's cube sequence. Error values indicate the center points of ± 0.005 error intervals 64

Figure 5.7 Angular error rate versus number of fraction bits used to represent the optical flow vector values. 66

Figure 5.8 Endpoint error rate versus number of fraction bits used to represent the optical flow vector values. 66

Figure 5.9 13th frame of Hamburg Taxi sequence. There are 4 moving objects. The car on the left and the van on the right are driving in their way, the taxi in the middle is turning the corner and the pedestrian is walking on the pavement. 68

Figure 5.10 OF vectors computed on FPGA hardware for Hamburg Taxi sequence. 69

Figure 5.11 Error histogram of optical flow vectors for Hamburg Taxi sequence. Error values indicate the center points of ±0.005 error intervals. 69

Figure 5.12 8th frame of synthetic Translating Tree sequence. The camera is moving from right to left while looking at a constant scene including a tree in the front side. The motion field has a velocity ranging from 1.73 and 2.26 pixels/frame. 70

Figure 5.13 Error histogram of optical flow vectors for Translating Tree sequence. Error values indicate the center points of ±0.005 error intervals. 71

Figure 5.14 Optical flow vectors computed on FPGA hardware for Translating Tree sequence. Four regions are zoomed in to provide a closer view of the optical flow field computed on FPGA. 72

Figure 5.15 LE usage percentage of modules 74

Figure 5.16 Memory usage percentage of modules 75

Figure 5.17 Floorplan of the designed hardware fitted on a EP2C70 device. The schematic shows the layout of the resources on the chip that are used to implement the design. The color legend is given next to the figure. The darker color of a particular resource indicates the higher usage ratio of that resource. 76

Figure 5.18 Pie chart represents the total power consumption partitioned among the design modules according to their percentages. More than half of the total power is consumed by the DMA module because of its high operating frequency and high resource usage. It is followed by the OF function and the Gradient computation modules. 80

Figure 5.19 Pie chart represents the dynamic and static power consumption partitioned among the design modules according to their percentages. Nearly 3/4 of this power type is consumed by the DMA module because of its high operating frequency and high resource usage. It is followed by the OF function and the Gradient computation modules. 80

Figure 5.20 Pie chart represents the total routing power consumption partitioned among the design modules according to their percentages. Nearly half of the routing power is consumed by the DMA module because of its high memory usage. It is followed by the OF function and the Gradient computation modules. 81

LIST OF ABBREVIATIONS

AAE	Average Angular Error
AE	Angular Error
AEE	Average Endpoint Error
ASIC	Application Specific Integrated Circuit
DMA	Direct Memory Access
DSP	Digital Signal Processor
EE	Endpoint Error
EEPROM	Electrically Erasable Programmable Read Only Memory
FIFO	First In First Out
FIP	Fixed Point
FLP	Floating Point
FPGA	Field Programmable Gate Array
fps	Frames per second
FSM	Finite State Machine
GPU	Graphical Processing Unit
HDL	Hardware Description Language
IC	Integrated Circuit
IP	Intellectual Property
JTAG	Joint Test Action Group
LAB	Logic Array Block
LE	Logic Element
LUT	Look Up Table
OF	Optical Flow
PC	Personal Computer
PLL	Phase Locked Loop
PROM	Programmable Read Only Memory
RTL	Register Transfer Level
SDRAM	Synchronous Dynamic Random Access Memory
STD	Standard Deviation
SRAM	Static Random Access Memory
SSRAM	Synchronous Static Random Access Memory
TS	Test Sequence
UART	Universal Asynchronous Receiver Transmitter
USB	Universal Serial Bus
VHDL	Very High Speed Integrated Circuit Hardware Description Language

CHAPTER 1

INTRODUCTION

1.1 Motivation

Optical flow computation is one of the computer vision algorithms that have a high computational complexity. Although it has many application areas such as collision detection, motion segmentation, tracking, background subtraction, visual odometry, video compression and many more, these applications generally suffer from insufficient computational power. There is a huge amount of data to be processed in image sequences and optical flow computations involves complex, time consuming operations. Unlike the applications where offline computation is possible, robotic applications generally require the computation of optical flow in real time. Moreover, in most of the applications, optical flow is computed as a pre-processing step and then it is fed to other high level vision algorithms as input which requires even extra computational power for those tasks.

The architecture of microprocessors is poorly suited to the structure of image processing algorithms. The demanded computational power generally exceeds the power supplied by todays conventional general purpose desktop computers and digital signal processors[14]. In addition, because of their high clock rates, they consume a huge amount of electrical power which also causes a high heat dissipation. To cool down the hardware, they have extra cooling equipments that increase their sizes and weights. Unfortunately, in aerial and mobile robotic platforms, available amount of power, space and weight is limited. Also, high clock rates of processors prevent them to be used in some areas such as space robotics where they are more susceptible to radiation[25]. So, these high end processors are usually not feasible to operate on

these robotic platforms. It is observed that, many available optical flow computation methods in literature can not be utilized on mobile robots for real-time applications because of these weaknesses of computational hardware.

There have been many studies made in literature on developing new methods for accurate and efficient computation of optical flow. The available methods are also compared in terms of their accuracy, density and computational complexity in some studies[10, 11]. Although there are very accurate methods proposed, they still require a high computational power.

While new methods on optical flow computation continue to appear, recently there has been a noticeable interest on application specific alternative computation platforms[25]. Application specific integrated circuits(ASIC) are designed to maintain the needs of that specific application in terms of computational power, space and power consumption. They can be designed to process chunks of data at once in parallelized and pipelined structures whereas, general purpose sequential processors require multiple processor cycles. This property of ASICs make them much more efficient than conventional processors. However, ASIC design and manufacturing is a long and difficult process. Once an ASIC is manufactured it is impossible to make any changes on it any more. So ASIC design and manufacturing is a costly process and it is only advantageous in mass production. Although their numerous advantages mentioned above, ASICs can not be utilized in academic studies.

For the solution to the static structure of ASICs, field programmable gate arrays (FPGAs) are developed. Having similar performance with ASICs in terms of computational power, size and power dissipation, they can also be programmed in field. This property makes FPGAs a flexible platform which enables to make modifications to the design in a matter of hours. These advantages of FPGAs attract academic interests to them nowadays. Although FPGAs are in use since 1980s they are newly used in academic studies. There are even some recent publications made on the suitability of FPGAs in vision systems which claim that FPGAs have an important potential to be utilized in vision research[25].

At robotics laboratory of Electrical and Electronics Engineering Department in METU, we came up with the need to do some computer vision operations on our highly mo-

bile legged robot platform SensoRhex for higher level tasks. However, we faced with the shortage of computational power on our robot and searched for a feasible solution. Being motivated by our needs and encouraged by a few successful applications and suggestions presented in literature, we designed a custom hardware solution for computing optical flow which is subject to this thesis work.

We would like to clearly state here that the scope of this thesis work is related with the high performance and efficient hardware design to compute an optical flow computation algorithm that is already available in computer vision literature. We focus our efforts not on the performance of the algorithm itself but on developing the hardware that fulfills the aforementioned objectives for robotics platforms.

1.2 Literature Survey

Studies in optical flow calculation dates back to 80's and up to now, there are many methods proposed for optical flow computation. These methods can be mainly grouped as gradient based, correlation based, energy based and phase based methods [11].

Gradient-based methods depend on the evaluation of spatio-temporal derivatives. The earliest two gradient based methods are presented by Horn & Schunck [23] and Lucas & Kanade [24]. Horn and Schunck presents a method assuming that the optical flow field is smooth which introduces a global smoothness term to constrain the estimated velocity field. Lucas & Kanade's method depends on an assumption that a point's neighboring pixels move with it, meaning that the flow is constant locally. This introduces additional equations to determine optical flow vectors by utilizing a least squares estimate. Since gradient based methods are rather popular, there are many other gradient based methods presented later in [28], [12] and [31]. The Horn & Schunck's method have relatively less computational complexity over many other methods and provide high density optical flow vectors with a reasonable error rate. This method is also suitable for high performance FPGA hardware implementations. The computations can be done using fixed point representations with small word lengths. They can be implemented using parallel and pipelined structures which yields a high throughput.

The matching based approaches depends on the determination of correspondences between consecutive frames. These correspondences can be found by correlating small patches in the images at different times to find the best fit. The first matching based method is presented by Anandan which is based on an SSD based matching technique [4]. A region including 5x5 pixels is searched around 3 pixels displacement for a suitable match.

Energy based(also called frequency based) methods utilizes the output energy of velocity tuned filters. Heeger's method [22] can be considered for one of the methods in this group. In this method, 3D Gabor filters are used to sample the power spectrum. Then a least squares estimate is utilized to minimize the difference between the predicted and the measured motion energies.

Phase based methods are similar to the energy based techniques. The method is first presented by Fleet & Jepson given in [20]. They use the outputs of velocity-tuned filters. However, they utilize the phase information instead of the amplitude component utilized in energy based methods. Their method yields a dense flow field with high accuracy in expense of high computational cost.

The authors of the presented optical flow techniques, implement and test their methods on general purpose computers. The first reason is to achieve comparable performance evaluation of the methods with each other. Although the clock rates vary from PC to PC, this measure roughly gives a benchmark. The other reason is probably the ease of implementation on a general purpose computer. Although the presented computational performances of their methods implemented on a PC are a disappointment for many realtime application, the chosen test platforms are general purpose computers for the reasons we mentioned.

On the other hand, the performance evaluation of methods on sequential general purpose computers gives no clear idea about their performances on parallelized and pipelined architectures such as implementations on ASICs, FPGAs or GPUs. It is known that in many computer vision algorithms, parallelized and pipelined implementations on FPGAs can produce better performances than in general purpose computers[25]. To be able to enhance the computation performance of optical flow methods, some FPGA implementations are presented in literature in [5, 7, 8, 9, 29, 34].

FPGA implementation of Horn & Schunck method is first peresented in [7] with a performance of processing 19 fps of 50x50 pixels images. To decrease the calculation time, they run the algorithm for 3 iterations which yields enough precision for many applications. Another implementation of the same authors uses the Camus correlation method [15] yielding an output of 25 fps at 100x100 pixels images[8]. They also implement a method presented in [32]. None of their publications gives quantitative measures on the accuracy or make comparisons with other studies. Later, they presented new studies on utilizing their implementations on real time applications such as lane departure detection[6].Another hardware implementation of Horn & Schunck method is presented in [26]. They claimed processing 256x256 pixels images at 60 fps. However, there is no information, discussion or comparison on the power consumption, error rates and accuracy of the implemented system. A recent work is presented in [17] which uses census transformation. They achieve a processing time of 22ms on images with 640x480 resolution..

Lucas & Kanade's method is implemented in hardware by Diaz et al. two times in 2004 [19] and in 2008 [18]. In [19], they claim to achieve a performance of processing 320x240 pixels images at 24 fps operating in real-time. Their implementation yield 100% density because of the absence of error thresholding. However, the resultant flow field has a high average angular error rate of 18° where software implementation can yield only 4.3° on the same image sequence. In [18], they present an improved version of their previous work which is capable of processing 800x600 images at 170 fps. They also present a detailed accuracy analysis. They achieve an angular error rate of 18.3° with 92% density and 3.5° with 36% density.

There are also alternative computation platforms other than FPGAs that can be used for high performance optical flow computation. GPUs (Graphical Processing Units) are one of these platforms. There are a few studies in literature that reports the performance of optical flow computation on GPUs. A study on optical flow computation using GPU is given in [33]. They implement a tensor based method and achieve a 2.8 times speed-up compared to a Pentium4 2.8 GHz PC implementation. However, there is no discussion on accuracy of the computed flow field. The only GPU implementation of Horn & Schunck's method is presented in [27]. They use multiresolution method with 2 levels. The computation of 316x252 images takes 443ms in multi-scale and 3ms

in single-scale. In [16], there is a comparison between an FPGA and a GPU implementation of optical flow computation. They use a tensor based optical flow method and claim a processing 320x240 images at 538 frames per second. They discuss the advantages and drawbacks of GPUs over FPGAs. The most important drawback of FPGAs is the complexity of design process. They claim that FPGAs require a 12x more development time. However, GPUs consume much more power than FPGAs and requires a host PC for operation where FPGAs can be placed on stand-alone platforms. Therefore, the high power consumption and high space occupation properties of GPUs make them unfeasible to be utilized on mobile robotic platforms. A more detailed version of this study in [16] can be found in the thesis work given in [13].

As far as we know, there is no work presenting the *accuracy*, *power consumption* and *logic resource usage* of Horn & Schunck's method implemented on FPGA. Mentioned studies on this method concentrates on the computation speed. However, it is known that, it is disadvantages to implement floating point operations on FPGAs[25]. The workaround is often to use fixed point operations which decreases the accuracy of the computations. Along with the speed, accuracy is also a key parameter to determine the feasible application areas of the presented method.

1.3 Contributions

The lack of a hardware that is capable of processing optical flow in real time is a factor that prevents the robotics research to utilize many studies presented in computer vision literature. Having the ability to implement available computer vision techniques in robotic platforms has a high potential to achieve great improvements in robotics research.

Unfortunately, it is not possible to obtain the required hardware off the shelf. Although there are some hardware mpeg coding chips that compute optical flow for video compression applications, they do not allow hardware reconfiguration to adapt to new situations. There is no such configurable commercial product for the time being as far as we know. In the literature, we came across a few universities that claim to design their own hardware but none of them provide the source HDL code for their design. Even if they provided the source code, it is still not easy to implement the

code unless having the same or similar FPGA hardware. At the end of this thesis work, we obtain a high performance hardware with low power consumption that is capable of providing over-realtime optical flow data to be used on our current robotic platform SensoRHex and other robotic platforms to be used further on.

As far as we know, none of the presented hardware implementations of Horn & Schunck optical flow in literature, report the accuracy, power consumption and resource usage of their hardware. There is an absence of this kind of knowledge in the literature. This thesis work also provides us the mentioned information. We think that the achievements of this thesis work will be appreciated by the related research society and so have a high potential for publication of these results.

1.4 Outline of the Thesis

In this chapter, we stated our motivation for this thesis work and a literature survey on the related publications. The contributions of this thesis work is presented. We finish this chapter after giving an outline of the thesis. The reminder of this thesis is organized as follows.

Chapter 2 presents a brief summary on the background of optical flow computation. We explain the mathematical derivation of the optical flow algorithm used and its numerical computation on digital hardware.

Chapter 3 explains the hardware requirements to implement a computer vision algorithm on an FPGA. We also introduce the properties of the FPGA platform used to implement the designed hardware and a short description of the FPGA structure. Then we complete this chapter with explaining the development tools utilized and the top down design methodology used to design the proposed hardware on FPGA.

Chapter 4 describes the complete hardware design of the proposed system in details. We also discuss the design alternatives and the selected approaches with providing the reasons.

Chapter 5 presents the performance analysis of the proposed design by stating the performance measures. The designed system is tested with some of the available

test sequences that are frequently used for performance evaluations of the optical flow techniques in literature. The results obtained from FPGA implementation are compared with the PC implementation of the same algorithm.

Finally, we conclude this thesis study in Chapter 6, by presenting our comments on the results and discussing the future plans and the possible improvements and modifications that can be done on the proposed design. We also summarize some important points learned from this research.

CHAPTER 2

BACKGROUND ON OPTICAL FLOW COMPUTATION

2.1 Optical Flow

Optical flow is defined as the distribution of apparent velocities of brightness patterns in an image[23]. This motion can be induced due to the relative movement of the objects in the scene or the observer(camera) itself. To determine the motion flow of a scene, it is needed an image sequence including at least two image frames taken from a camera consecutively of that scene. However, an image of a scene is just a 2D representation of a 3D world. Since the optical flow calculation is based on the 2D images, the computed optical flow field is a projection of 3D motion field of the scene. There are a number of methods presented in literature to compute the optical flow vector field on 2D images. Within this chapter, we give a brief summary on the background of optical flow computation and the Horn & Schunck's method [23] which is implemented on hardware.

2.2 Differential Optical Flow Computation Methods

There are various methods for computation of optical flow as mentioned in Section 1.2. Differential methods are rather popular among those methods. They depend on the use of spatio-temporal intensity gradients. They have relatively less computational complexity over other methods and provide high density optical flow vectors with a reasonable error rate. These methods are also suitable for high performance FPGA

hardware implementations. The computations can be done using fixed point representations with small word lengths. They can be implemented using parallel and pipelined structures which yields a high throughput. In this thesis work, we use a well known differential method for hardware implementation which is proposed by Horn & Schunck[23]. We follow the suggestions of Horn & Schunck's original paper on numerical computation of the algorithm.

2.2.1 Horn-Schunck Optical Flow Computation Algorithm

2.2.1.1 Pre-Assumptions

The differential optical flow methods have some pre-assumptions. The first main assumption is the so called "brightness constancy" which comes from the nature of the problem itself. This assumption states that the apparent brightness of the moving objects in the scene are approximately constant under motion for at least a short duration over time. If this assumption is violated, the correspondence of a pixel in the first image can not be matched correctly in the second image, since it does not have the same brightness as before anymore.

The second assumption requires the motion flow not to be large between consecutive frames. This means the changes should be gradual over time. The nature of the problem requires the brightness patterns in the first image to be found inside the second image also. However, the maximum displacement allowed, changes from algorithm to algorithm. Differential methods generally require the motion vectors to have very small velocities. For cases when the motion vectors have large velocities, multiresolution method can be used which computes flow vectors at different scales.

Generally the optical flow algorithms including the one we will utilize takes the input images to be processed to have pixel values represented with grayscale levels. Generally, the brightness data has enough information about the scene to be able to calculate the optical flow field. So the input image data is assumed to be represented by pixel brightness values.

2.2.1.2 Optical Flow Constraint Equation

Starting from the brightness constancy assumption, the constancy of image brightness pattern over time can be formulated as,

$$\frac{dE}{dt} = 0 \tag{2.1}$$

Here $E(x, y, t)$ denotes the image brightness at pixel location (x, y) at time t. If we consider a patch of brightness pattern that moves δx horizontally and δy vertically along a time period δt, the following equation holds for brightness constancy. Using chain rule, equation (2.1) can be rewritten as,

$$E(x, y, t) = E(x + \delta x, y + \delta y, t + \delta t) \tag{2.2}$$

The Taylor series expansion about the point (x, y, t) is expressed in equation (2.3).

$$E(x, y, t) = E(x, y, t) + \delta x \frac{\partial E}{\partial x} + \delta y \frac{\partial E}{\partial y} + \delta t \frac{\partial E}{\partial t} + HOT \tag{2.3}$$

For a sufficiently small δt, higher order terms can be neglected. Rearranging the equation accordingly yields,

$$\frac{\partial E}{\partial x}\frac{dx}{dt} + \frac{\partial E}{\partial y}\frac{dy}{dt} + \frac{\partial E}{\partial t} = 0 \tag{2.4}$$

If we denote optical flow vectors as,

$$u = \frac{dx}{dt} \qquad v = \frac{dy}{dt} \tag{2.5}$$

and partial derivatives of E as,

$$E_x = \frac{\partial E}{\partial x} \qquad E_y = \frac{\partial E}{\partial y} \qquad E_t = \frac{\partial E}{\partial t} \tag{2.6}$$

then equation (2.4) can be rewritten as,

$$E_x u + E_y v + E_t = 0 \tag{2.7}$$

Here we get the so called optical flow constraint equation,

$$(E_x, E_y)(u, v) = -E_t \tag{2.8}$$

It can be seen from the above equation that there are two unknown optical flow vectors u and v in one single linear equation. This is an ill-posed problem and in its current form it is impossible to find a unique solution for the optical flow vectors. To take the problem into a well defined one, Horn & Schunck introduces a second constraint which is the so called smoothness constraint equation. The following section describes the smoothness constraint definition.

2.2.1.3 Horn & Schunck Smoothness Constraint

Horn & Schunck observes that in most of the cases, the neighboring pixels make similar movements, in other words, have similar velocities which change gradually[23]. This results in a smooth vector field. Considering this observation, they introduce a smoothness constraint term which minimizes the sum of squares of the optical flow vectors' gradients,

$$\begin{array}{ccc} \left(\frac{\partial u}{\partial x}\right)^2 & + & \left(\frac{\partial u}{\partial y}\right)^2 \\ \left(\frac{\partial v}{\partial x}\right)^2 & + & \left(\frac{\partial v}{\partial y}\right)^2 \end{array} \tag{2.9}$$

Instead, the sum of the squares of the Laplacians of x and y components of the optical flow vectors can be used as another measure of the smoothness as,

$$\begin{aligned} \nabla^2 u &= \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} \\ \nabla^2 v &= \frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2} \end{aligned} \tag{2.10}$$

The Laplacians of u and v can be approximated using the expressions given in (2.11).

$$\begin{aligned} \nabla^2 u &= (\bar{u}_{i,j,k} - u_{i,j,k}) \\ \nabla^2 v &= (\bar{v}_{i,j,k} - v_{i,j,k}) \end{aligned} \tag{2.11}$$

The problem can be formulated as a minimization of a cost function given in equation (2.12).

$$\mathcal{E}^2 = \int \int (\ \mathcal{E}_b^2 + \alpha^2 \mathcal{E}_s^2\ )\ dx\ dy \tag{2.12}$$

The error terms $\mathcal{E}_b$ and $\mathcal{E}_s$ are defined as in equations (2.13), (2.14) and α is a coefficient used to adjust the weights of the two terms in the cost function.

$$\mathcal{E}_b = E_x u + E_y v + E_t \tag{2.13}$$

$$\mathcal{E}_s^2 = \left(\frac{\partial u}{\partial x}\right)^2 + \left(\frac{\partial u}{\partial y}\right)^2 + \left(\frac{\partial v}{\partial x}\right)^2 + \left(\frac{\partial v}{\partial y}\right)^2 \tag{2.14}$$

It may seem strange to include brightness constancy equation in the cost function since, a nonzero value of this equation means that the brightness constancy assumption is violated. However in real cases, the brightness values may change because of the noise or quantization errors. The coefficient α^2 is a weighting factor that adjusts the dominance of the error terms in the cost function.

The value of (u,v) pair that minimizes the cost function given in (2.12) is the optical flow vector. The equation can be reformulated as in (2.15) using calculus of variations method.

$$\begin{aligned} E_x^2 u + E_x E_y v &= \alpha^2 \nabla^2 u - E_x E_t \\ E_x E_y u + E_y^2 v &= \alpha^2 \nabla^2 v - E_y E_t \end{aligned} \tag{2.15}$$

Replacing the Laplacians of u and v with their approximation given in (2.11), the equations in (2.15) can be rewritten as in (2.16).

$$\begin{aligned} (\alpha^2 + E_x^2)u + E_xE_yv &= (\alpha^2\bar{u} - E_xE_t) \\ E_xE_yu + (\alpha^2 + E_y^2)v &= (\alpha^2\bar{v} - E_yE_t) \end{aligned} \tag{2.16}$$

Solving for u and v the equations in (2.17) are obtained.

$$\begin{aligned} (\alpha^2 + E_x^2 + E_y^2)u &= (\alpha^2 + E_y^2)\bar{u} - E_xE_y\bar{v} - E_xE_t \\ (\alpha^2 + E_x^2 + E_y^2)v &= -E_xE_y\bar{u} + (\alpha^2 + E_x^2)\bar{v} - E_yE_t \end{aligned} \tag{2.17}$$

$$\begin{aligned} u &= \frac{(\alpha^2 + E_y^2)\bar{u} - E_xE_y\bar{v} - E_xE_t}{(\alpha^2 + E_x^2 + E_y^2)} \\ v &= \frac{-E_xE_y\bar{u} + (\alpha^2 + E_x^2)\bar{v} - E_yE_t}{(\alpha^2 + E_x^2 + E_y^2)} \end{aligned} \tag{2.18}$$

2.2.1.4 The Numerical Solution

The optical flow calculation method proposed by Horn & Schunck includes the gradient and Laplacian calculations. These derivatives should be estimated from a set of discrete image brightness measurements available. So, they should be estimated numerically. It is possible to estimate these derivatives in a number of different ways. Horn & Schunck suggest to utilize the first order difference of 8 pixel values given in Fig. 2.1, Fig. 2.2 and Fig. 2.3. The i, j, k subscripts represents the row, column and frame number respectively. The corresponding difference formulas given in equations (2.19), (2.20) and (2.21) are used to compute E_x, E_y spatial and E_t temporal derivative estimations respectively[23].

$$\begin{aligned} E_x \approx \frac{1}{4} \Big(& E_{i,j+1,k} - E_{i,j,k} + E_{i+1,j+1,k} - E_{i+1,j,k} \\ + & E_{i,j+1,k+1} - E_{i,j,k+1} + E_{i+1,j+1,k+1} - E_{i+1,j,k+1} \Big) \end{aligned} \tag{2.19}$$

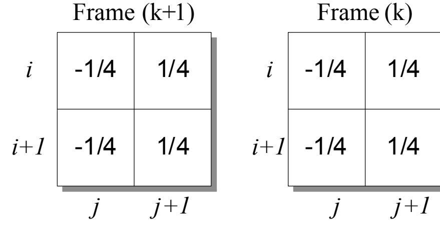

Figure 2.1: Numerical computation of E_x using first order difference of 8 pixels.

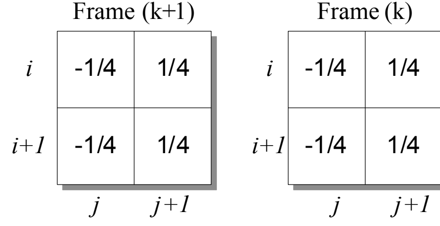

Figure 2.2: Numerical computation of E_y using first order difference of 8 pixels.

$$
\begin{aligned}
E_y \approx \frac{1}{4} \quad (& \quad E_{i+1,j,k} - E_{i,j,k} + E_{i+1,j+1,k} - E_{i,j+1,k} \\
+ & \quad E_{i+1,j,k+1} - E_{i,j,k+1} + E_{i+1,j+1,k+1} - E_{i,j+1,k+1})
\end{aligned} \tag{2.20}
$$

$$
\begin{aligned}
E_t \approx \frac{1}{4} \quad (& \quad E_{i,j,k+1} - E_{i,j,k} + E_{i+1,j,k+1} - E_{i+1,j,k} \\
+ & \quad E_{i,j+1,k+1} - E_{i,j+1,k} + E_{i+1,j+1,k+1} - E_{i+1,j+1,k})
\end{aligned} \tag{2.21}
$$

The Laplacians of u and v should also be estimated numerically. Horn & Schunck uses the following approximation to calculate the Laplacians terms:

$$
\begin{aligned}
\nabla^2 u & \approx (\bar{u}_{i,j,k} - u_{i,j,k}) \\
\nabla^2 v & \approx (\bar{v}_{i,j,k} - v_{i,j,k})
\end{aligned} \tag{2.22}
$$

$\bar{u}$ and $\bar{v}$ are called the local averages of u and v respectively which are defined as,

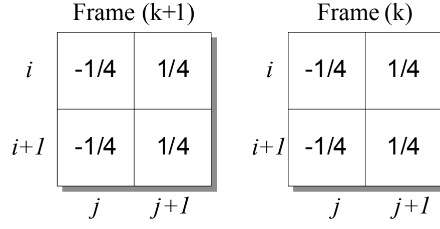

Figure 2.3: Numerical computation of E_t using first order difference of 8 pixels.

$$\begin{aligned}\bar{u}_{i,j,k} &= \frac{1}{6}\left(u_{i-1,j,k} + u_{i,j+1,k} + u_{i+1,j,k} + u_{i,j-1,k}\right) \\ &+ \frac{1}{12}\left(u_{i-1,j-1,k} + u_{i-1,j+1,k} + u_{i+1,j+1,k} + u_{i+1,j-1,k}\right) \end{aligned} \tag{2.23}$$

$$\begin{aligned}\bar{v}_{i,j,k} &= \frac{1}{6}\left(v_{i-1,j,k} + v_{i,j+1,k} + v_{i+1,j,k} + v_{i,j-1,k}\right) \\ &+ \frac{1}{12}\left(v_{i-1,j-1,k} + v_{i-1,j+1,k} + v_{i+1,j+1,k} + v_{i+1,j-1,k}\right) \end{aligned} \tag{2.24}$$

1/12	1/6	1/12
1/6	-1	1/6
1/12	1/6	1/12

Figure 2.4: Weight matrix for estimating local averages of optical flow vectors

To solve the Horn & Schunck optical flow equation numerically, they utilize iterative Gauss-Seidel method. The optical flow vector estimates calculated at each iteration is formulated as,

$$u^{n+1} = \bar{u}^n - E_x \frac{E_x \bar{u}^n + E_y \bar{v}^n + E_t}{\alpha^2 + E_x^2 + E_y^2} \tag{2.25}$$

$$v^{n+1} = \bar{v}^n - E_y \frac{E_x \bar{u}^n + E_y \bar{v}^n + E_t}{\alpha^2 + E_x^2 + E_y^2} \tag{2.26}$$

CHAPTER 3

FPGA HARDWARE PLATFORM FOR VISION

3.1 Requirements of the FPGA Hardware Platform

To be able to implement a designed digital hardware circuit, an appropriate development board should be obtained. The selected board provides the hardware infrastructure including required peripheral devices and an FPGA chip with sufficient resources on it. According to the application area of the development board, these peripheral devices and resources available on board varies. For example, a transceiver design for high speed communication applications needs different resources than an image processing hardware design. Moreover, the amount of required resources differ according to the complexity of the design.

FPGA development boards are available in a wide range of sizes with different feature sets. Different FPGA boards have a different feature set of logic, I/O interfaces, memory and other assorted hardware. As long as an FPGA board has enough logic resources and it has the required peripheral devices, a project can be implemented on any of the boards. In general, FPGAs with more logic, more I/O pins, higher speed, more memory and more peripheral devices are more expensive. While making selection, choosing the right board with peripheral devices and an FPGA chip having the proper feature set at the lowest cost is an important design consideration. The selected board should have low power consumption and moderate performance since it is designed to be used on robotic platforms. Another important point that should also be considered is the capacity of the board to meet future improvements and modifications of the design. The board and FPGA chip should have sufficient resources for potential additions to the design.

Each FPGA has a different number of logic elements (LE) that are used to implement user logic. They also contain various amounts of both internal embedded memory blocks and external memory devices such as RAM and ROM memory chips. The use of internal and external memory resources differ according to the required capacity and speed. Capacities of external memory are much larger than the internal memory, but they have a lower bandwidth and slower access times. Like many algorithms in computer vision, the implementation of optical flow algorithm processes a huge amount of data. Algorithms, that processes single image frame at a time step can be implemented easily in hardware that processes input data sequences without storing the whole data in memory. However, optical flow algorithm uses 2 image frames from consecutive time steps. This requires the whole image frames to be stored in memory before processing. The massiveness amount of data prevents it to be stored in internal memory blocks that are insufficient. So, external memories such as SSRAM or SDRAM should be utilized, although they have lower bandwidth. The bottleneck of these systems are generally caused by the memory accesses. In terms of performance, SSRAMs are faster but has lower capacity comparing SDRAMs. However, to prevent memory bottleneck, SSRAMs should be preferred for frame buffering.

Optical flow algorithm computation includes multiplication operations. The implementation of multiplication by programmable logic is not preferable both in terms of performance and its high resource usage. Instead, there are some FPGAs available which support Digital Signal Processing (DSP) applications, so they also contain hardware integer multipliers which have much higher performances and use no logic resources.

The performance of the design is proportional with the clock frequency it operates at. The limiting factor of the clock rate is the highest delay between the registers, called the critical path. To decrease the limiting effect of the critical path, the parts of the design with lower delays are clocked at higher rates and higher delayed paths are clocked at lower rates. This technique is called the multiple clock domain design. So, the design needs multiple clock frequencies. Each board contains a crystal controlled clock circuit that is normally used as the master clock for the user's digital logic circuit. Phase Locked Loop(PLL) circuits can be used to scale the crystal controlled clock to provide other clock frequencies. On some FPGAs, there are multiple PLLs

that are used to divide or multiply the clock frequencies and shift the phase of clock signals to generate different clock signals.

To send and receive data between PC and the FPGA board, there should be some communication interfaces available on board. This communication can generally be established through either RS232, USB or Ethernet protocols. Development boards which have these interfaces should be preferred.

FPGAs provide a wide variety of I/O features in terms of pin count, I/O voltage and pin drive strength. The number of pins should be sufficient to interface the external devices. It is generally helpful to have general purpose I/O pins for user access. Some of the peripheral devices need controllers for interface with FPGA logic. It is also advantageous if the hardware controllers are provided on board. Else, logic that provides a device interface circuit or controller will need to be constructed by the user using the FPGA's internal logic resources. Push buttons, switches, LEDs, seven segment and LCD displays, although not compulsory, are really helpful both for FPGA user interface and for debugging through the hardware design process.

3.2 FPGA Development Platform Specifications

Selection of the FPGA hardware development board is done according to the requirements explained in Section 3.1. There are a couple of suitable boards provided by companies such as Altera and Xilinx. There are two categories of FPGAs of each vendor called the "high end" and "low cost" devices. We mostly considered suitable boards that include FPGA chips classified as low cost devices(low power, moderate performance). The FPGA product portfolio of Altera is shown in Fig. 3.1 with a comparison of their available resources, performances, power consumptions and costs. Although they provide a high performance operation, high end FPGA devices such as Stratix V are not feasible for our needs because of their high power consumptions and power constraints of mobile robotic platforms. Therefore, we have to choose a device that is in the low power devices category. Altera calls its low power devices as Cyclone family. We choose a Cyclone II device because of its low power consumption, sufficient amount of resources and low cost.

Device	Process	LE	Memory	DSP Blocks	DSP (MHz)
Stratix V	28 nm	1087 K	50 Mb	3510	550
Stratix IV	40 nm	820 K	23.1 Mb	1288	550
Stratix III	65 nm	338 K	16.2 Mb	768	550
Stratix II	90 nm	180 K	9.0 Mb	384	450
Stratix	130 nm	80 K	7.3 Mb	56	350
Arria II	40 nm	350 K	16.4 Mb	1040	350
Arria	90 nm	90 K	8.5 Mb	736	350
Cyclone IV	60 nm	150 K	6.3 Mb	360	290
Cyclone III	65 nm	120 K	3.8 Mb	288	290
Cyclone II	90 nm	68 K	1.1 Mb	150	250
Cyclone	130 nm	20 K	0.2 Mb	-	-

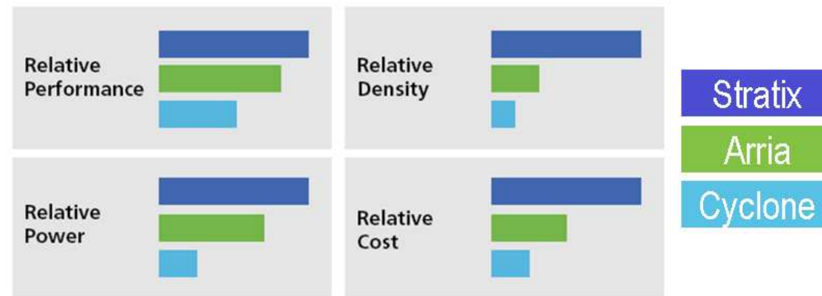

Figure 3.1: The FPGA product portfolio of Altera with a comparison of their available resources, performances, power consumptions and costs.

Peripheral devices such as memory are also important in choosing a hardware platform. To prevent memory access bottleneck, we selected SSRAM as primary external frame buffer since SDRAMs have lower bandwidths and harder to design interface. For PC communication, we preferred RS232 for its easy interface. However, for future improvements a faster communication protocol such as USB or Ethernet is also beneficial. The most suitable FPGA development board to our needs is selected as the DE2-70 board which is also used widely in many of the universities in the world for both education and research. The top view of DE2-70 board is given in Fig. 3.2.

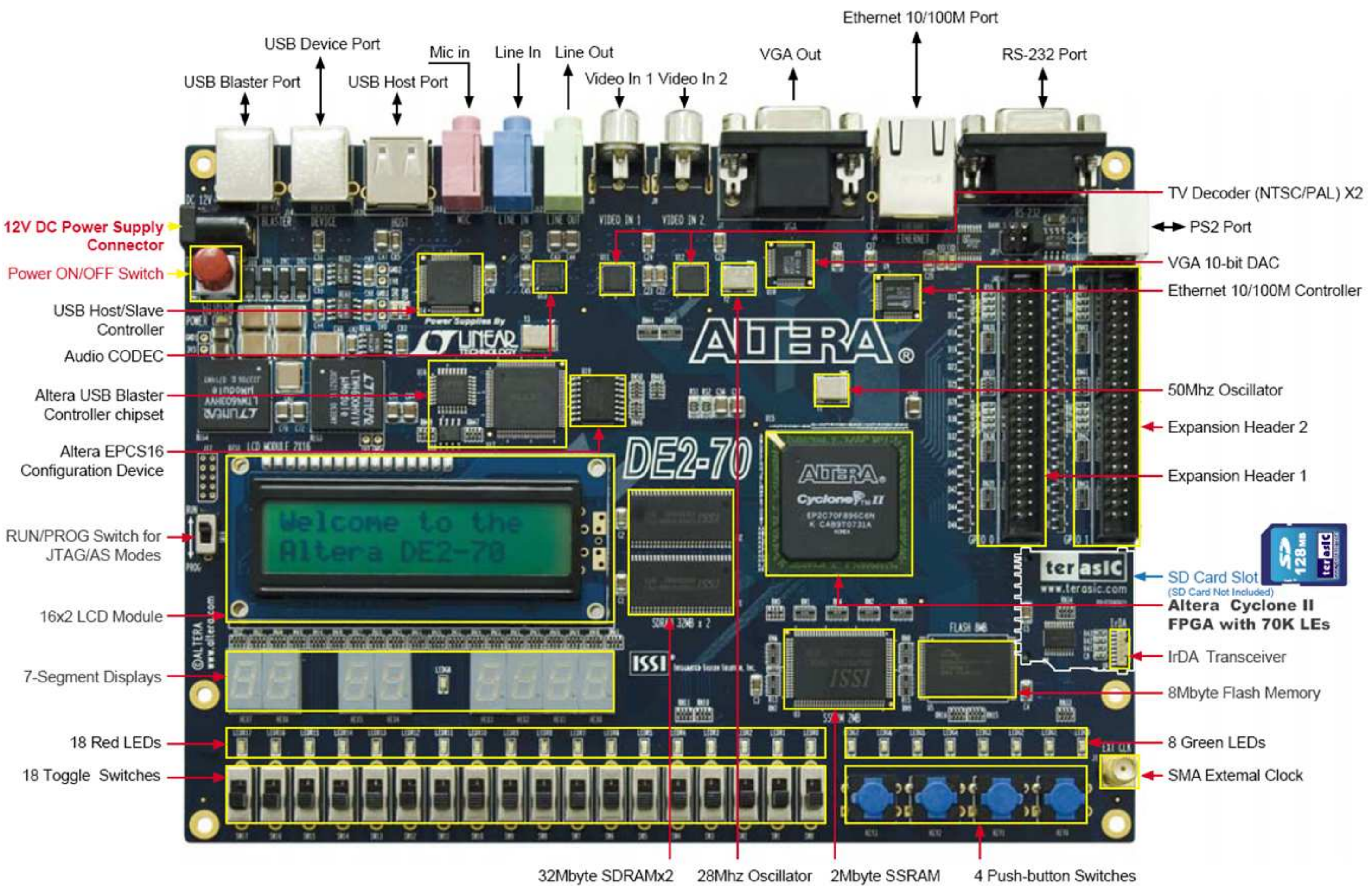

Figure 3.2: Altera DE2-70 FPGA development board[1]

The following hardware is provided on the DE2-70 board:

- Altera Cyclone II EP2C70F896 FPGA device
- Altera Serial Configuration device - EPCS16
- USB Blaster (on board) for programming; both JTAG and Active Serial
- 50-MHz oscillator and 28.63-MHz oscillator for clock sources
- 2-Mbyte SSRAM
- Two 32-Mbyte SDRAM
- 8-Mbyte Flash memory
- SD Card socket
- IrDA transceiver
- RS-232 transceiver and 9-pin connector
- 10/100 Ethernet Controller with a connector
- USB Host/Slave Controller with USB type A and type B connectors
- PS/2 mouse/keyboard connector
- VGA DAC (10-bit high-speed triple DACs) with VGA-out connector
- 2 TV Decoder (NTSC/PAL/SECAM) and TV-in connector
- 24-bit CD-quality audio CODEC with line-in, line-out, and microphone-in jacks
- Two 40-pin Expansion Headers with diode protection
- 1 SMA connector
- 4 pushbutton switches
- 18 toggle switches
- 9 green user LEDs
- 18 red user LEDs

- 2x16 character LCD display
- 8 seven segment displays

The FPGA chip included in DE2-70 board is the one that has the highest resources available in the Cyclone II family named as EP2C70. It provides sufficiently high logic elements, memory blocks, embedded multipliers, PLLs and I/O pins required for our current and future design needs. The Cyclone II family features are given in Table 3.1.

Table 3.1: Cyclone II FPGA family features[2]. The development board we use includes EP2C70 device which has the highest resources available in the Cyclone II family.

Feature	EP2C5	EP2C8	EP2C15	EP2C20	EP2C35	EP2C50	EP2C70
LEs	4,608	8,256	14,448	18,752	33,216	50,528	68,416
M4K RAM blocks (4 Kbits plus 512 parity bits	26	36	52	52	105	129	250
Total RAM bits	119,808	165,888	239,616	239,616	483,840	594,432	1,152,000
Embedded multipliers	13	18	26	26	35	86	150
PLLs	2	2	4	4	4	4	4
Maximum user I/O pins	158	182	315	315	475	450	622

3.3 FPGA Architecture

Field Programmable Gate Arrays (FPGA) are integrated circuits that have configurable structures to implement user defined logic circuits. An FPGA contains a large number of identical "logic elements"(LEs) that can be wired according to a logic function specified by a user defined code. Each elements are interconnected by a matrix of wires and programmable switches. A user's design is implemented by specifying the logic function for each logic element and selectively closing the switches in the interconnect matrix. This is done by means of a user program that defines the function of the circuit, usually written in a hardware description language such as Verilog or VHDL. The user defined function is first converted to "register transfer level" (RTL)

by a synthesizer tool. Then the configuration for implementing the RTL structure by FPGA logic elements is generated by the fitter tool according to the FPGA device used.

The general architecture of FPGA's are given in Fig. 3.3. They consist of two planes called the functional plane and the programming plane. Logic elements and interconnect routing channels are located in functional plane. This plane is configured according to the values stored in SRAM structures in programming plane. These structures define the look-up table values and interconnect switch states. They also configure the I/O blocks for pin direction whether they are input or output.

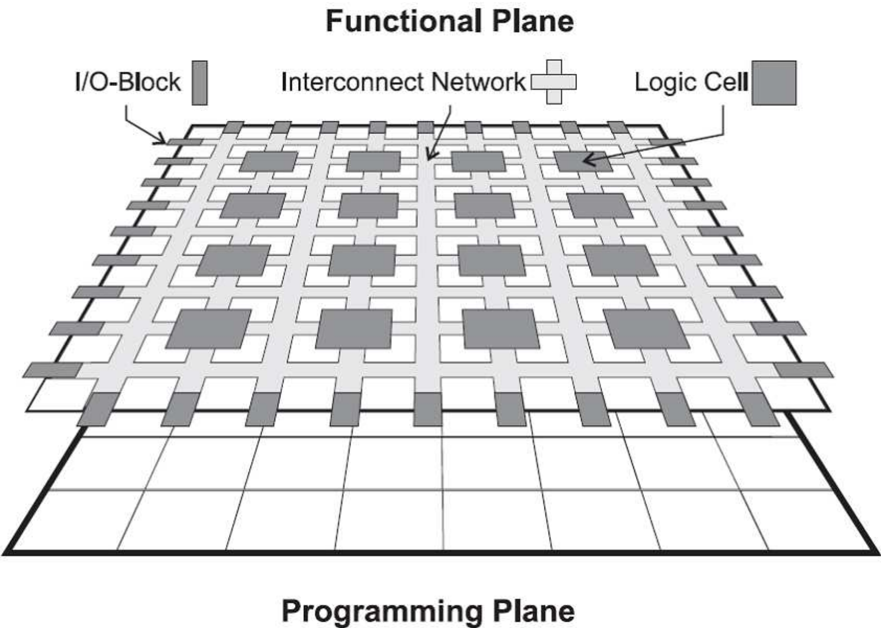

Figure 3.3: General hardware architecture of FPGA devices[21]. Functional plane is used to implement logic functions defined by user by logic cells and routing channels. Programming plane stores the LUT values in SRAMs for configuration of functional plane at start up.

FPGAs contain a two-dimensional row and column-based architecture to implement user logic. A column and row interconnection network provides signal connections between Logic Array Blocks (LABs) and embedded memory blocks. To perform more complex operations, logic elements can be automatically connected to other logic elements on the chip using a programmable interconnection network.

The Cyclone II device is configured by loading the configuration program to internal

static random access memory(SRAM). Because of the usage of SRAM is used in FPGAs, the configuration will be lost whenever power is removed. In actual systems, a small external low-cost serial flash memory or programmable read only memory (PROM) is normally used to automatically load the FPGA's programming information when the device powers up.

3.3.1 Logic Elements

An LE is a small unit of logic providing efficient implementation of user logic functions. Fig. 3.4 shows a Cyclone II device logic element. Each logic element consists of a four input look-up table(LUT), a register and a carry in/out logic for use in arithmetic mode. Logic elements can be used either for combinational or sequential functions. Each LE's programmable register can be configured for D, T, JK, or SR operation. Each register has data, clock, clock enable, and clear inputs. Logic functions are implemented using a look-up table in logic elements instead of using actual logic gates. LUTs are consisted of a high-speed 16 by 1 SRAM. Four inputs are used to address the LUT's memory. The truth table for the desired gate network is loaded into the LUT's SRAM during programming. A single LUT can therefore model any network of gates defining a combinational logic function with four inputs and one output. For combinational functions, the LUT output bypasses the register and drives directly to the LE outputs. For implementing more complex logic functions, more than one logic elements are connected to each other properly in "logic array blocks"(LABs) through local interconnect channels. Each LAB includes 16 logic elements. When a logic function needs even more logic resources, the LABs can also be connected using the interconnect routing channels[2].

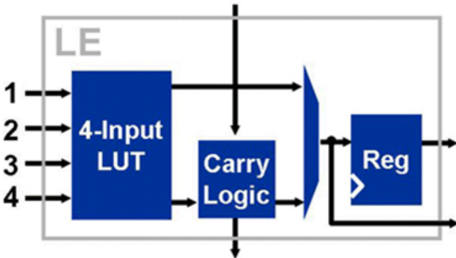

Figure 3.4: Components of a logic element block[2]. Each logic element can be used to implement combinational, sequential logic functions and arithmetic operations.

Each logic element can be configured to operate in standard logic mode(normal mode) or arithmetic mode[2]. Normal mode is used to implement combinational or sequential logic functions. For efficient implementation of arithmetic functions the logic element operates in arithmetic mode. In arithmetic mode, a logic element implements a 2-bit full adder and basic carry chain. The arithmetic mode is ideal for implementing adders, counters, accumulators, and comparators.

3.3.2 Embedded Memory Blocks

Embedded memory blocks are internal random access memories that provides a low capacity data storage with high performance data access. The embedded memory in Cyclone II devices consists of columns of M4K memory that can operate up to 250MHz. Each M4K block has a capacity to store 4608 bits including parity bits for error correction. They can be used to implement various types of memory with or without parity, including true dual-port, simple dual-port, and single-port RAM, ROM, and first-in first-out (FIFO) buffers. The word length can be adjusted between 1 and 36 bits. The memory content can be initialized by a user defined memory initialization file in programming state.

3.3.3 Embedded Multipliers

Multiplication operations are involved in many digital signal processing(DSP) applications. However, implementing a multiplication operation on an FPGA using standard logic blocks is not efficient both in terms of performance and resource usage. To allow implementation of high performance multiplication operations, some FPGA families, including Cyclone II, have embedded hardware multiplier blocks. These hardware blocks can be used to multiply both signed and unsigned operands up to 250MHz performance. Each embedded multiplier can be used in one of two operational modes which are single 18x18 bits mode or two 9x9 bits mode depending on the application needs. The multiplication on operands with any word length can also be done using multiple embedded multiplier blocks.

3.4 FPGA Development Environment and Design Flow

Increasing design complexity and higher gate densities are forcing digital designs to change the design methods used. Rapid prototyping using hardware description languages (HDLs), IP cores, and logic synthesis tools has replaced the traditional gate level design using a schematic editor. These new HDL-based logic synthesis tools can be used for both ASIC and FPGA-based designs. The two most widely used HDLs are VHDL and Verilog.

The hardware design process is known as a complicated and time consuming task. However, to simplify the design process, FPGA manufacturers are providing software packages including some useful design tools. These tools are integrated in a user friendly FPGA design environment. This environment supports behavioral description based design using an HDL.

3.4.1 Quartus II

The computer aided design software provided by Altera for FPGA designs is called the Quartus II software. This tool provides designer a number of tools that are used in FPGA design process. All of the tools are included in an integrated design environment with easy to use interfaces. The details for the usage of these tools can be found in [3]. We utilized the following tools in designing the proposed hardware:

- HDL Editor, Analysis & Synthesis, Fitter Tools

- Classic & TimeQuest Timing Analyzer

- Signaltap II Embedded Logic Analyzer

- Powerplay II Power Analyzer

3.5 FPGA Design Flow

A design in an FPGA is actually nothing but a digital logic design. The aim is to design a logic circuitry that performs the desired function. The designed circuitry can

be defined in a number of ways. The conventional method is drawing the schematic diagram of the circuitry. However, for large designs, the schematic diagram of the circuitry becomes very complicated. So, this method is not very functional for large designs. Another method which is generally preferred is using a hardware description languages (HDL) to define the logic circuitry. Verilog and VHDL languages are the two most popular HDL's that are used today. In this design, we use Verilog as our hardware description language.

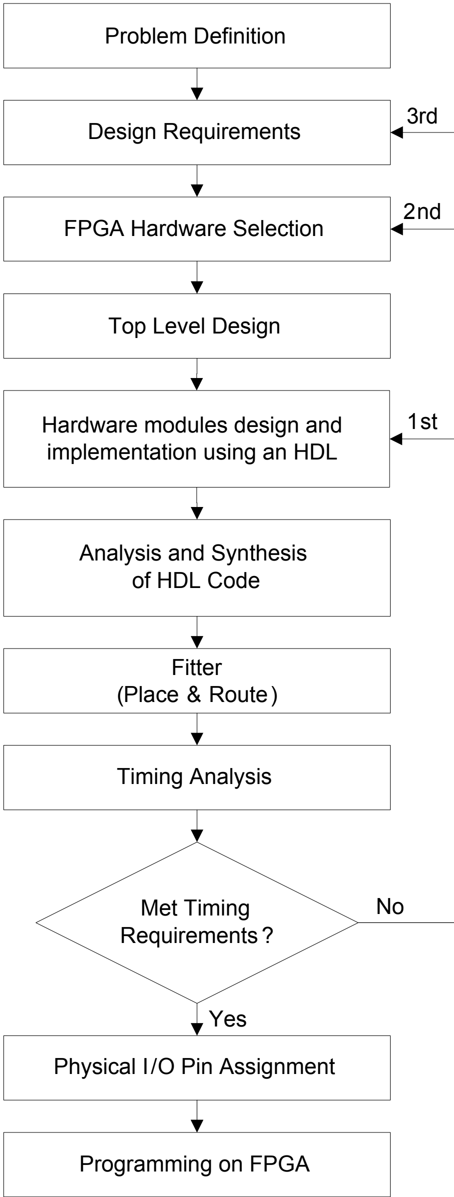

Figure 3.5: FPGA Design Flow Diagram

In designing of the hardware, we used the top-down design methodology. Top-down

design is the design method where high level functions are defined first, and the lower level implementation details are filled in later. This methodology simplifies the design task and allows the partitioning of the whole design into easily manageable subparts. The top level block represents the entire hardware structure and each lower level block represents major functions.

A typical FPGA design process using an HDL with top-down design methodology is shown in Fig. 3.5. The first step is to make the problem definition properly. In this step the main function of the design, inputs and outputs of the system should be determined. In the next step, the designer should state the design requirements clearly. There may be constraints on the power consumption, system throughput, clock rate, memory size, weight and volume of the hardware. According to the requirements, designer should determine a suitable hardware. Having defined the design requirements, the design is divided into functional sub-design blocks. Then these blocks are designed, implemented using an HDL language and tested individually. The written HDL code is analyzed and converted into low level representation which is called register transfer level(RTL) description by synthesis tools. Then the generated netlist should be reformed to place into the specified hardware. This process is carried out by the fitter tools. Fitter tools optimize the logic according to the hardware resources available. After the design is placed and routed for the device, the timing requirements such as setup&hold times, clock delays etc. should be checked. This process is called timing analysis. If the timing requirements were not met, the first action should be to revise the hardware design. If alternative design approaches do not solve the problem then the second action is moving to another device with higher speed grade. If there is no device that is capable of implementing the design, then it means that the designing with current specifications is impossible. So the design requirements should be lowered and checked if they can be satisfied. When the timing requirements are met then the next step is to assign the physical device pins to the design inputs and outputs. The last step is the programming of the device. Most of the current FPGA architectures don't include any on chip program memory. The configuration is generally stored in external EEPROM memories and loaded into the FPGA SRAMs on every power up.

CHAPTER 4

PROPOSED FPGA HARDWARE DESIGN FOR OPTICAL FLOW

This chapter includes the details of the proposed hardware design. The hardware is designed using a top down design methodology. First, the top level design blocks and the data flow between these blocks are determined. Then the sub-modules such as Direct Memory Access module, Gradient Computation module etc. are designed individually following the design, implement and test cycles. Then each functional sub-block is connected to each other using the top level design file. Finally, design is completed after the functional operation of the whole design is verified.

4.1 High Level Block Diagram

For large design projects, a logical way to design the system is by utilizing the divide and conquer methodology. The overall system is divided into sub-design parts that are performing a specific function. This design methodology is called top-down design approach in which high level functions are defined first and the lower level implementation details are filled in later. The high level block diagram of the hardware modules of our designed system is shown in Fig. 4.1.

The required data to compute optical flow vectors are read from the appropriate locations of SSRAM by the Direct Memory Access (DMA) module and stored temporarily in fifo line buffers. These buffers increases performance by reducing the number of memory accesses. The gradient and average computation module reads required data from DMA and computes spatiotemporal gradients and local averages of optical flow

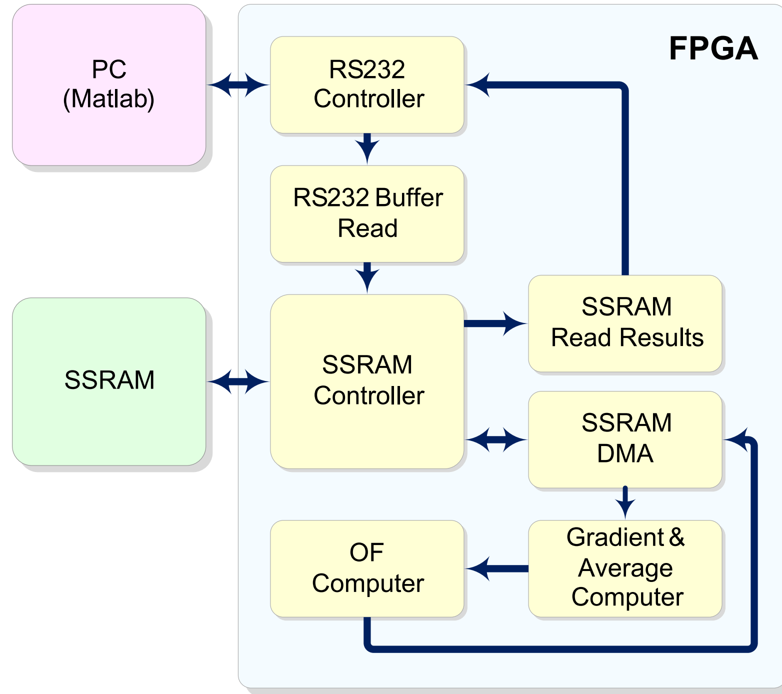

Figure 4.1: Block diagram of the designed system

vectors. Then computed values are used by optical flow computation module to produce optical flow vectors. The results are written back to the SSRAM by DMA module. Finally, after the computation is finished, the results are read from SSRAM by SSRAM read module and written to RS232 controller to be sent to the PC.

The system is designed using multiclock domains and in a highly pipelined structure which increases the system throughput dramatically. The memory interfaces operates at higher clock rates than the computation modules. This helps overcoming the memory bottleneck in the design. The optical flow computation modules are designed in a pipelined structure to reduce computation time. At every stage in the pipe, the data is exposed to a specific operation. After a while the first input data produces the first output at the end of the pipe. Then with one clock cycle intervals, the results corresponding the input sequence are taken. Each of the individual sub-blocks are realized following the, design, implement and test cycles. After testing each of the individual design blocks for timing requirements, they are connected according to the data flow sequence given in Fig. 4.2. Overall system is tested for verification of functional operation and the design is completed.

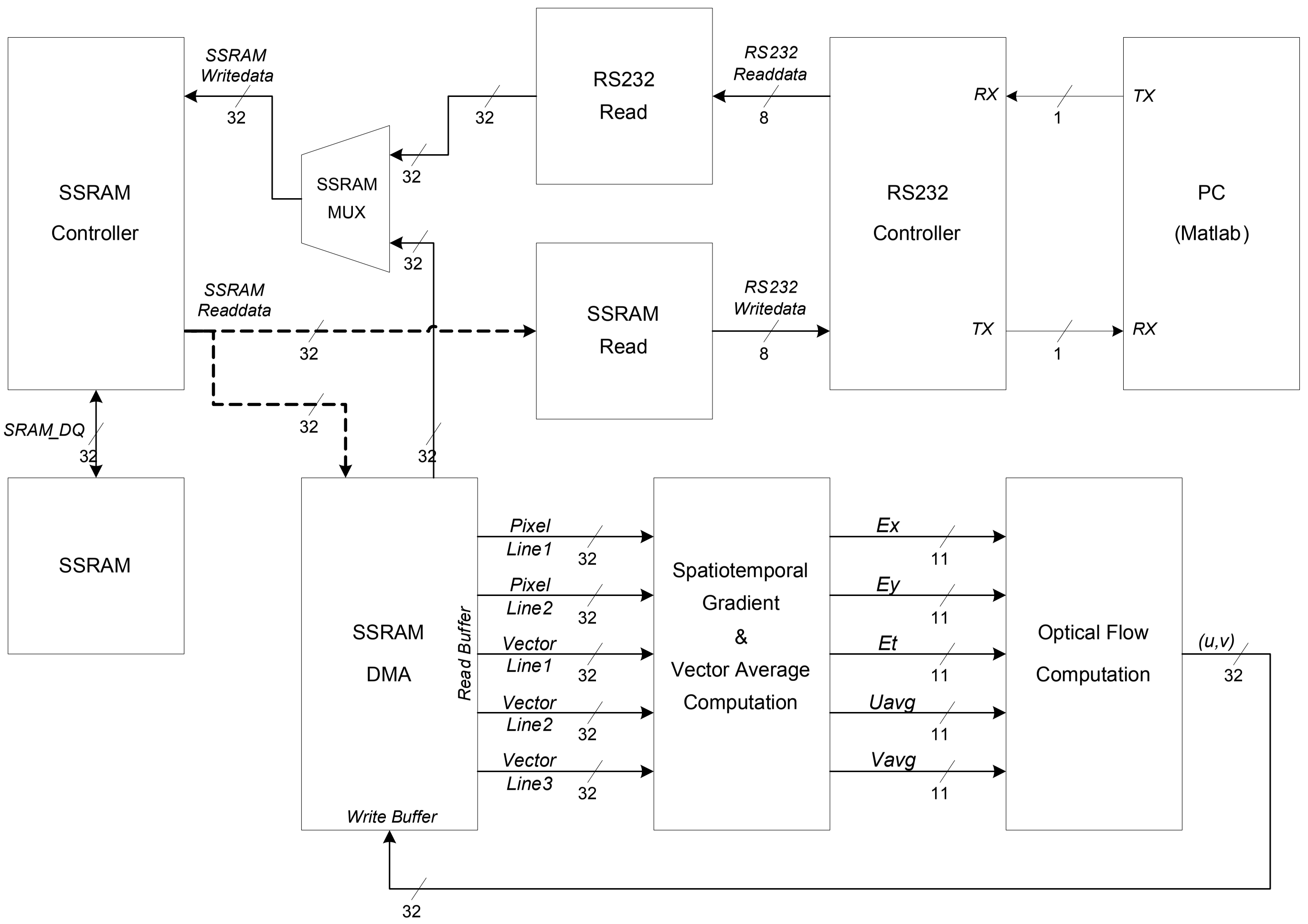

Figure 4.2: Data flow diagram of the designed system

4.2 Reset Circuitry

Reset circuitry is one of the most important parts of the digital circuits. Stable operation of a circuit depends firstly on the errorless initialization of that circuit. The main purpose of the reset circuitry is to ensure the digital circuit to start operating normally. Another usage is to manually restart the operation of the circuit because of its unexpected behavior.

When the digital circuit is powered up, the electrical lines are in an unstable state for a short time. The supply voltages fluctuate outside the operating range and crystal oscillators generate unstable waves. This is a transient state and stabilizes after a while. The circuit should not start operating until this transient period ends. The reset circuitry holds the circuit in reset state for a predetermined time period that is long enough for the electrical lines to stabilize.

Most of the digital circuits include some memory elements which are used to hold some values or the states of a finite state machine. These memories are in an undetermined state when the circuit is powered up. Most of the registers are required to be loaded with a predetermined value at the beginning of the operation. Reset circuitry loads the initial values to the registers at the beginning of the operation.

Another issue is the asynchronous reset signals in synchronous designs. At synchronous designs all signals should be ready at the inputs of the registers at a certain time interval, called setup time, in order the register to latch the signal errorless. Otherwise timing problems may occur and the circuit works unexpectedly. Reset circuitry is also responsible with synchronizing asynchronous reset signal to the system clock.

Reset circuitry consists of two parts. The first part, as given in Fig. 4.3, generates the active low reset signals for a predetermined period of time to hold the circuit in reset state. There are two reset signals generated by the circuitry. The difference between two signals is the duration of the reset time. The reset time period is generated by a 32 bit counter. After power up or when the reset button is pressed, reset signals are deasserted and the counter starts from zero and increments at every rising edge of 50MHz clock. When the counter reaches the first predetermined count the first reset signal is asserted. The second one remains low until the counter reaches to the second

predetermined value. Then the second signal is asserted and counter is disabled until the next reset or power up situation.

The second part of the reset circuitry is included in every design module. This part initializes the finite state machine states and loads the initial values to the registers when the reset signal is asserted. The operation is synchronous to the modules own clock. Initialization is done at the first rising edge of the clock that comes after the reset signal is asserted. The circuit starts its normal operation at the second rising edge of the clock and goes on.

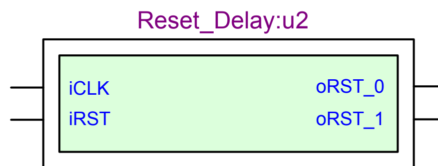

Figure 4.3: Reset signal generator module. Two reset signals are generated with different release times. They are used to synchronize the start up of modules designed.

4.3 Clocking Circuitry

Every module in the designed system needs a clock signal to operate. The most common approach is using only a single clock signal called system clock for all sequential logic in the design. This approach simplifies the design and shortens the design process. However, the maximum clock frequency that can be applied to every module in the design is different. If a single clock is used, the designer should consider the signal path with the highest delay called the critical path. The system clock rate should be lower than the clock rate of the module that has the critical path. So, single clock usage introduces a performance penalty for the design. The workaround to this problem is often to use more than one clock signal. These are called designs with multiclock domains. In this approach, modules with higher delay are clocked with lower frequency and faster modules are clocked with higher frequency. Although this technique yields a boost in the performance, the design procedure becomes more complicated. The signal transmission between different clock domains should be handled carefully, otherwise synchronization and timing problems may arise easily. Despite the complicated design process, we designed using two clock domains for the sake of

performance.

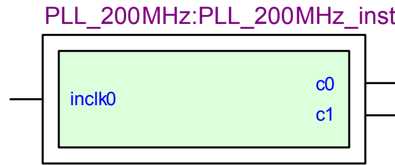

Figure 4.4: Phase Locked Loop (PLL) module generates the required clock signals for the operation of designed modules. There are two clocks generated which are 200MHz and 50MHz from the input clock of 50MHz.

The designed circuit needs two clock sources to operate which are 50MHz and 200MHz. Operating frequencies of the design modules are given in Table 4.1. The main clock source of the system is the external 50MHz crystal oscillator. The 50MHz and 200MHz clocks are generated internally by "Phase Locked Loop" (PLL) circuits. PLLs are used to generate different clock frequencies with adjustable phase shifts from a source clock. The Cyclone II FPGA on our development board has four PLL blocks. Each PLL block can generate up to three different clocks. Block diagram of PLL block in the design is given in Fig. 4.4. In our design both 50MHz and 200MHz clock sources are generated by PLLs from the input clock of 50MHz. The signal transmissions between two clock domains are handled by using dual clock FIFO buffers and some synchronization stages to prevent timing problems.

Table 4.1: Operating clock frequencies of modules. 200MHz is used for the modules that access SSRAM memory to reduce memory bottleneck. 50MHz is used for the rest.

Module Name	**Frequency**
SSRAM Controller	200MHz
SSRAM DMA	
Optical Flow Computer	50MHz
Gradient & Average Computer	
RS232 Controller	
SSRAM Read	
RS232 Buffer Read	

4.4 Memory

The nature of the optical flow algorithms require pixel data from two frames to be processed at the same time. The frames should be stored in a memory prior to the processing stage. So there is a big amount of storage needed to store frame data. When the pixel values are processed, the results should be stored in the memory also. Then the results are read back from memory and send to the PC. During processing stage the stored data is accessed from random addresses of memory and written to FIFO buffers. This requires a random access memory to be used for temporary storage.

The available random access memory devices available on DE2-70 development board are SDRAM and SSRAM. The advantage of dynamic memories(SDRAM) are their higher storage capacities. However they store the data in capacitors which need periodic refresh cycles. During refresh cycles, read or write operations could not be performed. So this reduces their data transfer bandwidth. Other option is to use SSRAMs. SSRAMs can provide more bandwidth than the SDRAMs. They store the data in flip flops. Unlike SDRAMs, they do not need to be refreshed. However, their storage capacities are lower than the SDRAMs.

Table 4.2: Properties of memories available on DE2-70 board. SSRAM is preferred because of its higher bandwidth although its capacity is lower than the SDRAM memory.

Memory Type	**Capacity(Mb)**	**Frequency(MHz)**	**Word Length(bits)**
SSRAM	18	200	32
SDRAM	512	166	16x2

In digital image processing hardware designs where the data is stored in external memory, the bottleneck of the system is generally caused by the low memory access bandwidths. To increase the system throughput, high bandwidth memories should be used. For medium resolution images, SSRAMs can provide both required capacity and speed. In this design, external 18 Mb SSRAM is used for main memory.

4.4.1 SSRAM Memory Controller

The data transfers from/to the SSRAM are regulated by an SSRAM controller. The controller we use in our design is an IP core provided within the university program of Altera to be used with the SSRAM device on our development board. The controller adjusts the data and control lines to perform error free read and write operations. The block diagram of the controller module is given in Fig. 4.5.

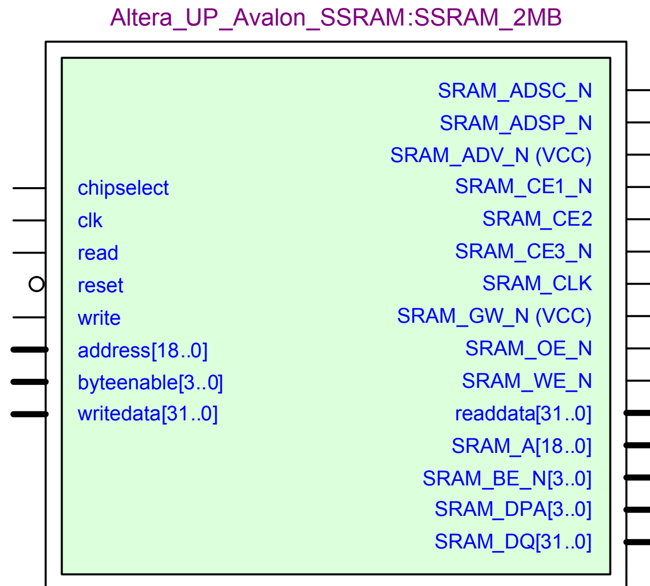

Figure 4.5: SSRAM memory controller module terminals.

The controller is designed to control an SSRAM organized as 36 bits of 512k address locations. The address bus is represented with 19 bits. At each address location, 32 bits are used for data storage and the remaining 4 bits are used internally for error correction. The data bus is 32 bits in width. There is a single data bus to be used both for read and write data operations. The controller multiplexes data bus for read and write operations. For a read operation, the pins of the FPGA becomes tristated. Write operation has 1 cycle and read operation has 2 clock cycles latencies. During read and write operations the "waitrequest" port is pulled high for the specified clock cycles long. The write operation can be done in two ways. An address location(32 bits) can be written all at once or in any individual bytes combination. Write mode can be controlled by the "byteenable" port.

The controller has one read and one write port. However, in our design two modules

should have accesses to the ram. This case is called multi-mastered bus structure where more than one masters need ram access. The bus is shared between the masters by a bus arbiter. The first master which is the "DMA module" has the highest priority. So, we simplified the arbitration task by granting the bus to the first master as long as it requests bus access. Otherwise the bus is granted to the second master which is the RS232 to SSRAM data transfer module.

4.4.2 Direct Memory Access Module

The optical flow algorithm requires 8 pixel values and 8 optical flow vector values, which we refer as one data packet, to compute one optical flow vector in the image. In our previous design, these data were read, the gradient and optical flow vectors are computed and written back to the SSRAM by the same module. The drawback of this design was its low throughput. The optical flow computation part has mathematical operations which introduces a long delay time and could operate at 50MHz, whereas SSRAM read/write operations are fast and could operate at 200MHz. When the same module does all this tasks, it could operate at 50MHz at most. Therefore, the optical flow computation had to wait for the data to be read from the SSRAM.

To increase the efficiency, the memory access bottleneck was solved by designing a Direct Memory Access Module (DMA) given in Fig. 4.6. This module operates at 200 MHz and fetches the required data in the order of process. Then the fetched data is stored in dual clock FIFO buffers to be read by the gradient and the optical flow computation modules. When the optical flow computation module computes the flow vector, it puts the result to the write FIFO buffer. The DMA module reads results from write FIFO and writes to the determined address location of SSRAM. The layout of the stored data in SSRAM is given in Fig. 4.7.

The DMA module has five read ports and one write port. For every port there is a FIFO buffer. Two read ports are used to read pixels from the consecutive two lines of the image frames. The other three read ports are used to read optical flow vectors from the consecutive three lines of the optical flow vector field. In Fig. 4.8, the pixel FIFO buffers filled with pixels to be processed can be seen. The FIFO buffers are 32 bits wide so they can store one data packet at each location for computation of one

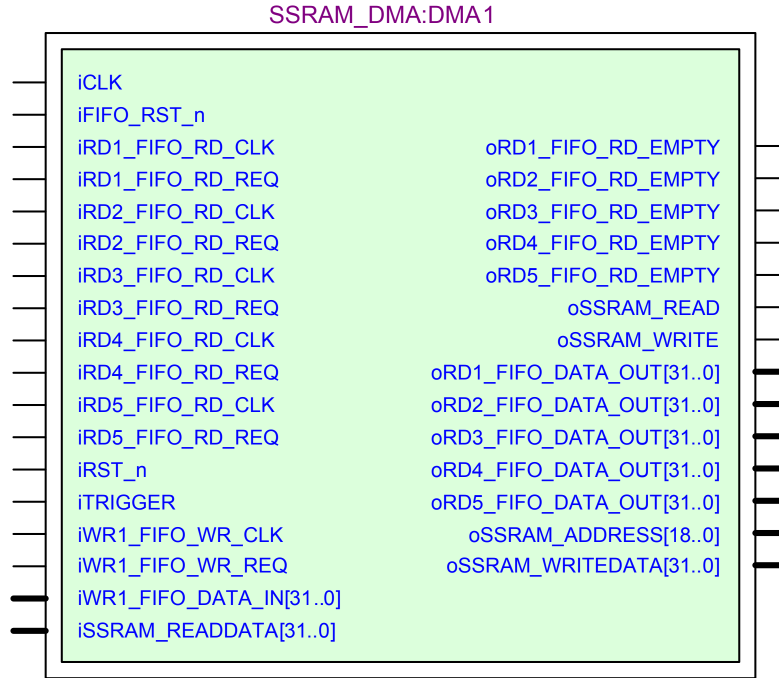

Figure 4.6: Direct Memory Access (DMA) module terminals. DMA is used to handle memory read/write operations required for the operation of optical flow computations. It helps increasing the bus utilization and reduces the memory bottleneck.

optical flow vector. The length of the buffers vary according to the read port assigned. The first pixel read buffer is filled with the pixels starting from the first line of the image. The second pixel read buffer is filled with the pixels starting from the second line. So there is a phase difference of 1 row between the first and second pixel read FIFO buffers.

To reduce the memory accesses, we only read pixels for the first read FIFO. When the pixels from the second line of the image are read from SSRAM, they are also copied to the second FIFO. However, in this method the pixels required to compute the first optical flow vector can be computed only after all the first line of the image is read from memory. So the gradient and vector average computation module can start with a latency equal to the time required to read 1 row of two frames from the memory. This phase difference also requires the length of the first read buffer to be 1 row more than the second read buffer.

The same idea for pixel read buffers is also applied to optical flow vector read buffers. However, to compute their average, 8 flow vectors from *three* consecutive rows of

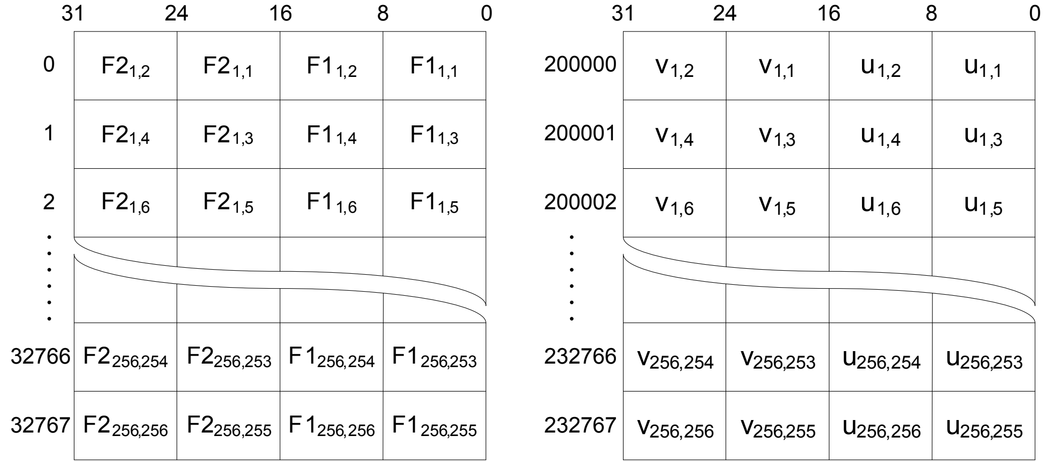

Figure 4.7: Layout of stored data in SSRAM for two frames of 256x256 pixels. Image frames are stored starting from the first address location and optical flow vectors are stored beginning from the 200,000th address location of SSRAM.

vector field is required. This time, there should be three FIFO buffers with 1 row of difference for each of them. The first FIFO buffer has length to hold three rows of optical flow vectors. The second FIFO can hold two rows and the third one can hold one row of vector data. In Fig. 4.9, the optical flow vector FIFO buffers filled with vectors to be processed can be seen.

The DMA module consists mainly of a finite state machine(FSM) operating at 200MHz and the read/write dual clock FIFO buffers. FSM states and state transition diagram is given in Fig. 4.10. The state machine serves the FIFO buffers using the round robin scheduling method. Since read operations are more than the write operations, read buffers has higher priority than the write buffer. The optical flow computation process starts with a trigger signal. The FSM is at idle state until the trigger signal comes. Then FSM puts the read address of the pixel data for the first FIFO and controls whether the FIFO is full or not. If the FIFO is not full, it initiates the read command to the SSRAM controller. After three wait cycles, the data becomes ready on the write port of the read FIFO buffer. At the same time, FSM initiates write signal for the first read buffer to latch the pixel data. If the address of the read pixel data is not from the first row of the images, the write command for the second read FIFO is also initiated. The same procedure is also applied for other read FIFOs. After serving the read FIFOs, FSM checks the write FIFO buffer if it is empty or not. If the write FIFO is not empty, FSM puts the data and the SSRAM address for the data to be

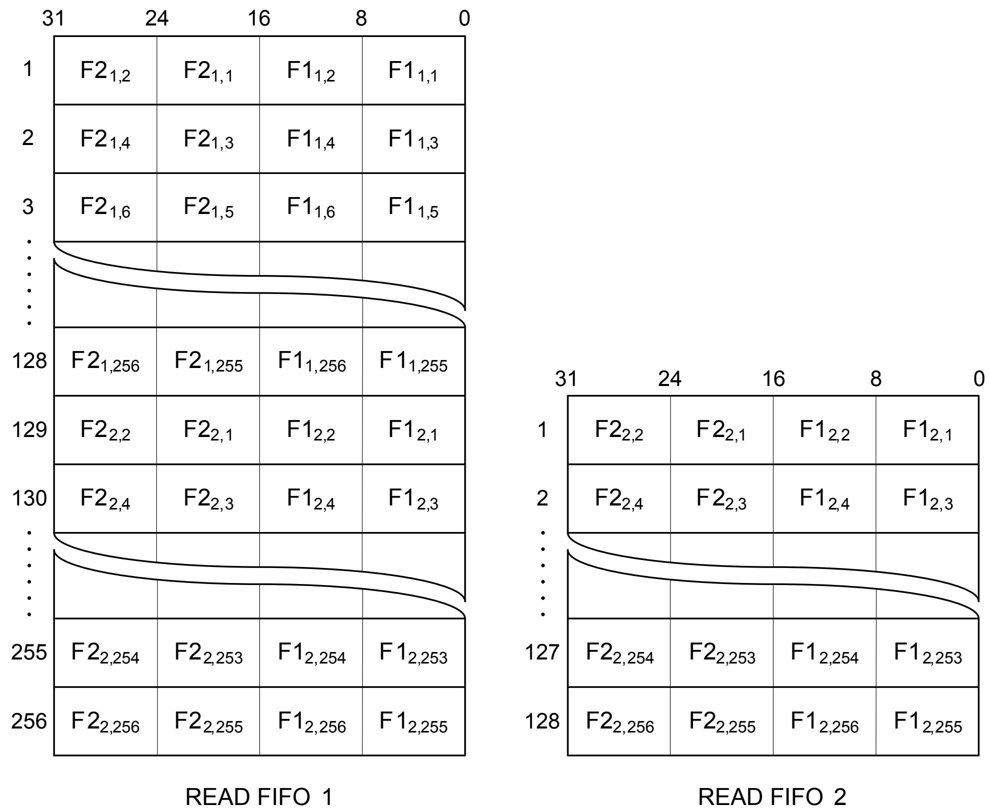

Figure 4.8: Layout of pixel FIFO buffers. Each location of FIFO buffers is 32 bits in width and stores 2 consecutive pixels from frame 1, and 2 consecutive pixels from frame 2.

written and initiates a write command to the SSRAM controller. After one cycle the write operation completes. Then FSM either continues to serve from the read buffers or switches to the idle state. If the address of the last data written is equal to the last element of the optical flow vector field, then the FSM stays in idle state, else it continues to serve the FIFO buffers.

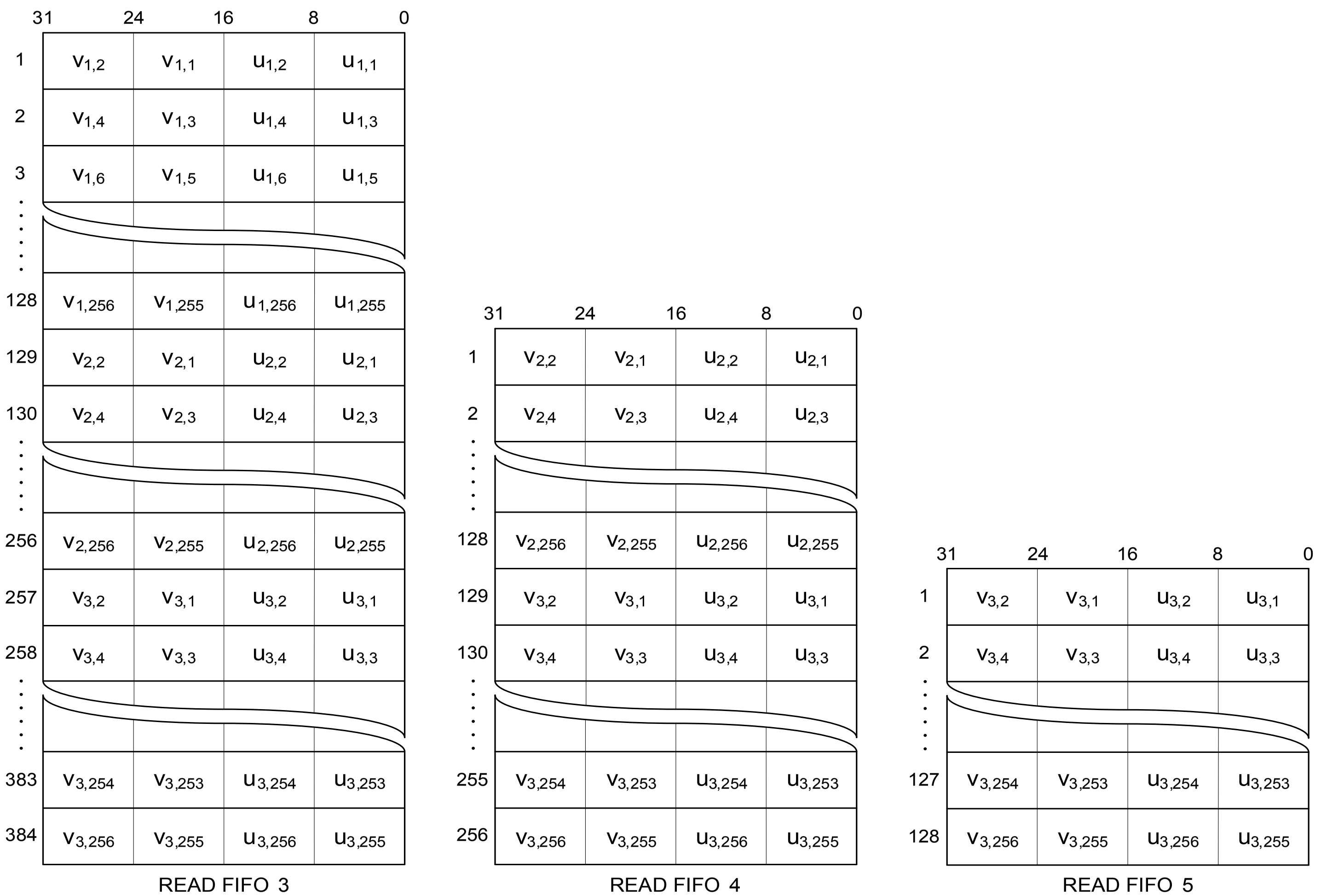

Figure 4.9: Layout of optical flow vectors in FIFO buffers. There is a phase difference of 1 line of vectors between three FIFO buffers.

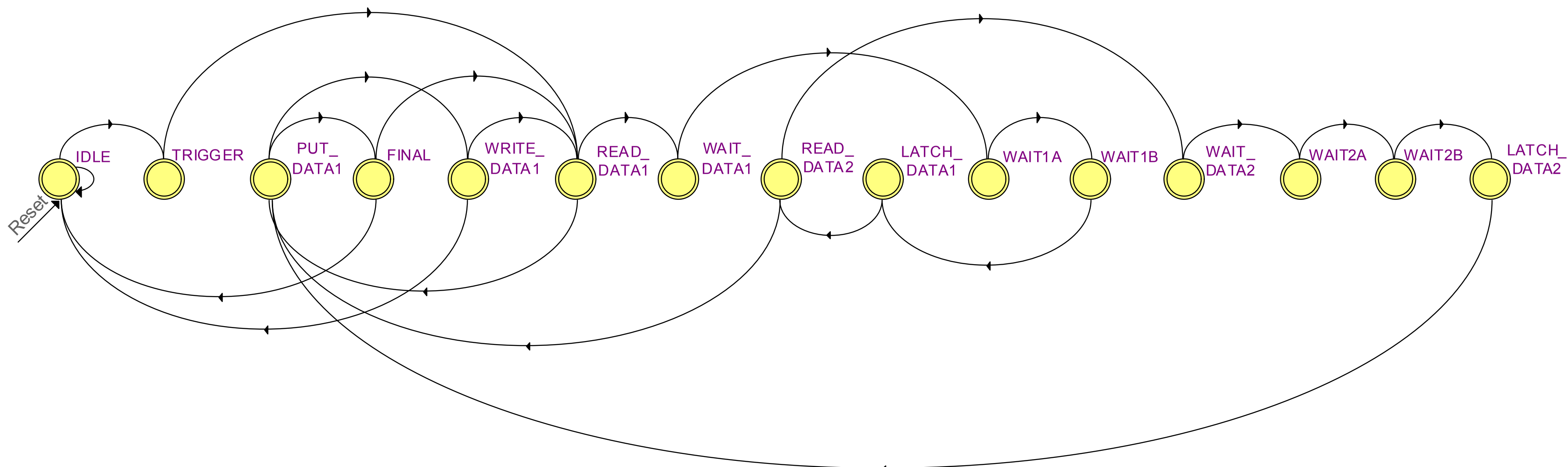

Figure 4.10: DMA module FSM states

4.4.2.1 FIFO Buffers

A first in first out(FIFO) buffer is a memory to hold the data and then output in the order of arrival. Fifo buffers are frequently used to increase the efficiency in most of the digital circuits which have parts operating at different speeds. The FIFO buffers have two types which are the ones with single or dual clocks. When the modules that are writing to or reading from the FIFO have same synchronous clocks the single clock FIFO buffers are used. But, when the writing module's clock is different than the reading module's clock, then dual clock FIFO buffers should be used. They can also have different data widths and lengths according to the needs. In this design, the FIFO buffers are utilized in the DMA module and in RS232 communication controller.

The FIFO buffer used in the DMA module is a dual clock FIFO buffer. The ports of the DMA FIFO buffer is given in Fig. 4.11. For read data FIFOs, the write operation is done by the DMA module at 200MHz and read operation is done by the gradient and average computation module at 50MHz clock rates. All the FIFOs have 32 bits width but have different lengths as explained in Section 4.4.2.

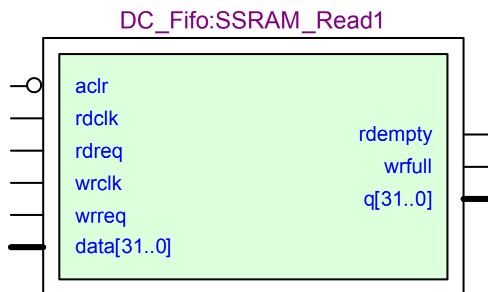

Figure 4.11: Dual clock FIFO buffer module terminals.

RS232 FIFO buffers are used both for received and transmitted data. The RS232 communication bandwidth is low with respect to the reading and writing modules. Both the reading and writing modules of FIFO operates at 50MHz, so single clock FIFO memories are used. The buffers are 8 bits in width and 256 bytes in depth.

The buffers operates in show-ahead mode. In this mode the data in the front of the buffer is continuously held in readdata port and can be read any time. After reading the data, the read command should be initiated. One clock cycle after the read command, the current data is removed and the next data in FIFO is put to the

read port. Before initiating the read command, the empty flag should be controlled to be low, else a random data appears on the readdata bus.

4.5 Spatiotemporal Gradient and Optical Flow Vectors Local Average Computation

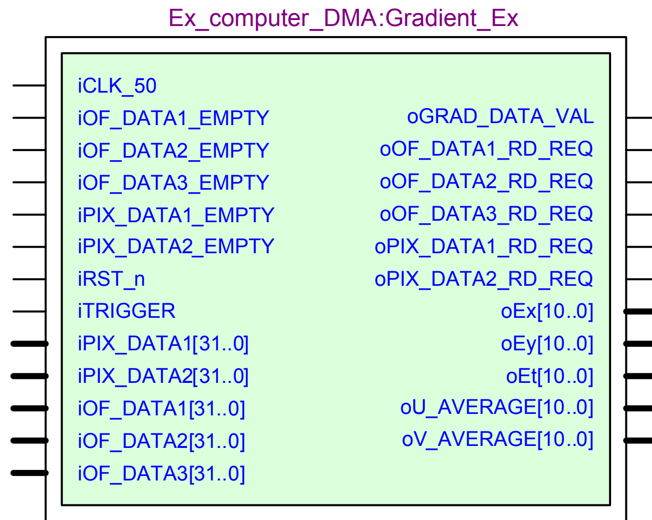

Figure 4.12: Spatiotemporal gradient computation module terminals. This module computes both spatiotemporal gradient values and the local averages of optical flow vectors.

The optical flow computation algorithm requires spatiotemporal image gradients and local averages of optical flow vectors to be computed. Horn and Schunck's algorithm approximates the numerical computation of spatiotemporal gradients using the formula given in equations (2.19), (2.20) and (2.21). This module computes the terms that are required to compute the optical flow vectors. Each pixel data used in computations are grayscale intensity values represented with unsigned 8 bits. The required pixel data are read from the FIFO buffers of the DMA module that are connected to the iPIX_DATA1 and iPIX_DATA2 ports. Spatiotemporal derivative computation includes the summation of 8 pixel values that are read from the pixel buffers. The pixels with minus sign are converted to signed two's complement format and summed. The result of this summation requires 11 bits to prevent overflow. Then the summation should be divided by 4. The division by 4 operation can be done using the arithmetic right shift by 2 bits. However, since the data is represented in integer format, this

operation causes to lose precision on the result of the division. To prevent precision lost, we decided to represent the spatiotemporal gradients in 11 bits fixed point format with 2 bits fraction. So, division by 4 is represented by putting the fraction point between the second and third bits as shown in Fig. 4.13.

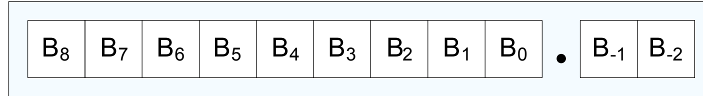

Figure 4.13: Spatiotemporal gradients data representation in 11 bits fixed point format. 2 bits fraction is enough for representing the gradient results without any accuracy lost.

Another computation made by this module is the optical flow vectors average calculation. Horn and Schunck proposes the numerical approximation of optical flow vectors local averages using equations (2.23) and (2.24). The computation of this equations on FPGA requires divide by 6 and divide by 12 operations. However, FPGA implementation of division operation has drawbacks both because of its high resource usage and low operation performance. The workaround can be to use a sequence of multiplication, shifting and addition operations to implement a division by a constant number. Another approach is often to approximate the division operation. There are many approximation methods used, such as look-up table or divide by powers of two. We preferred to use the division by powers of two method. The error introduced by this approximation is given in Table 5.2. The found results show that the approximation error is far less than the error introduced by implementation of operations in fixed point format. Therefore, we modified Horn and Schunck's numerical approximation for local averages of optical flow vectors given in equations (2.23) and (2.24) as seen in equations (4.1) and (4.2) respectively.

1/8	1/8	1/8
1/8	-1	1/8
1/8	1/8	1/8

Figure 4.14: The weight matrix used for estimating local averages of optical flow vectors on FPGA. The weights given in Fig. 2.4 are modified to simplify the division operation and increase the accuracy.

$$\bar{u}_{i,j,k} = \frac{1}{8} \left(u_{i-1,j,k} + u_{i,j+1,k} + u_{i+1,j,k} + u_{i,j-1,k}\right) + \frac{1}{8} \left(u_{i-1,j-1,k} + u_{i-1,j+1,k} + u_{i+1,j+1,k} + u_{i+1,j-1,k}\right) \quad (4.1)$$

$$\bar{v}_{i,j,k} = \frac{1}{8} \left(v_{i-1,j,k} + v_{i,j+1,k} + v_{i+1,j,k} + v_{i,j-1,k}\right) + \frac{1}{8} \left(v_{i-1,j-1,k} + v_{i-1,j+1,k} + v_{i+1,j+1,k} + v_{i+1,j-1,k}\right) \quad (4.2)$$

The implementation of divide by 8 operation on FPGA is a simple arithmetic right shift operation by three bits. The summation of the 8 vectors are computed using two's complement signed representation similar to the computation of gradients as explained above. Again, the division operations are conducted using fixed point representation because of the precision considerations as explained above. The fixed point representation of optical flow vector averages are done using 11 bits word length with 3 bits fraction as seen in figure Fig. 4.15.

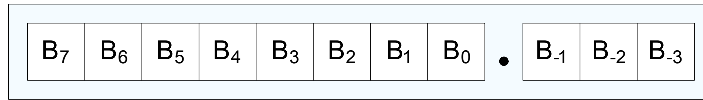

Figure 4.15: Optical flow vector local average data representation in 11 bits fixed point format. 3 bits fraction is enough for representing the vector local average results without any accuracy lost.

The computations and data flow is managed by a finite state machine. The state diagram of the designed FSM is given in Fig. 4.16. The FSM first initializes the registers such as pixel counters, read/write address counters, state variables, data valid and control registers after reset is occurred. The FSM stays in the idle state until the trigger signal comes. The trigger signal is generated by another module called the trigger delay module. This module generates trigger signals with predetermined delay times to start the modules in an order. For example, the DMA module should be triggered earlier than the gradient and vector average computation module to fill data buffers in advance for the gradient and vector average computation module to start operating.

Prior to the computation, the buffers are controlled if they are empty or not and the required data is captured from appropriate buffers. Computation of results corresponding to the first column of each row requires more data than the rest of the columns. The gradient computation of first columns need 8 pixels and the vector average computation needs 9 pixels to be read from buffers. The other columns uses a part of the previous data read from buffers. So, the gradient computation requires 4 new pixel values and vector average requires 3 new vector values to be read from buffers for the rest of the columns. This flow is regulated by the FSM using data counters. When a result is computed, it is put on the output ports with appropriate valid signals. The results are transferred to the optical flow computation module from these ports.

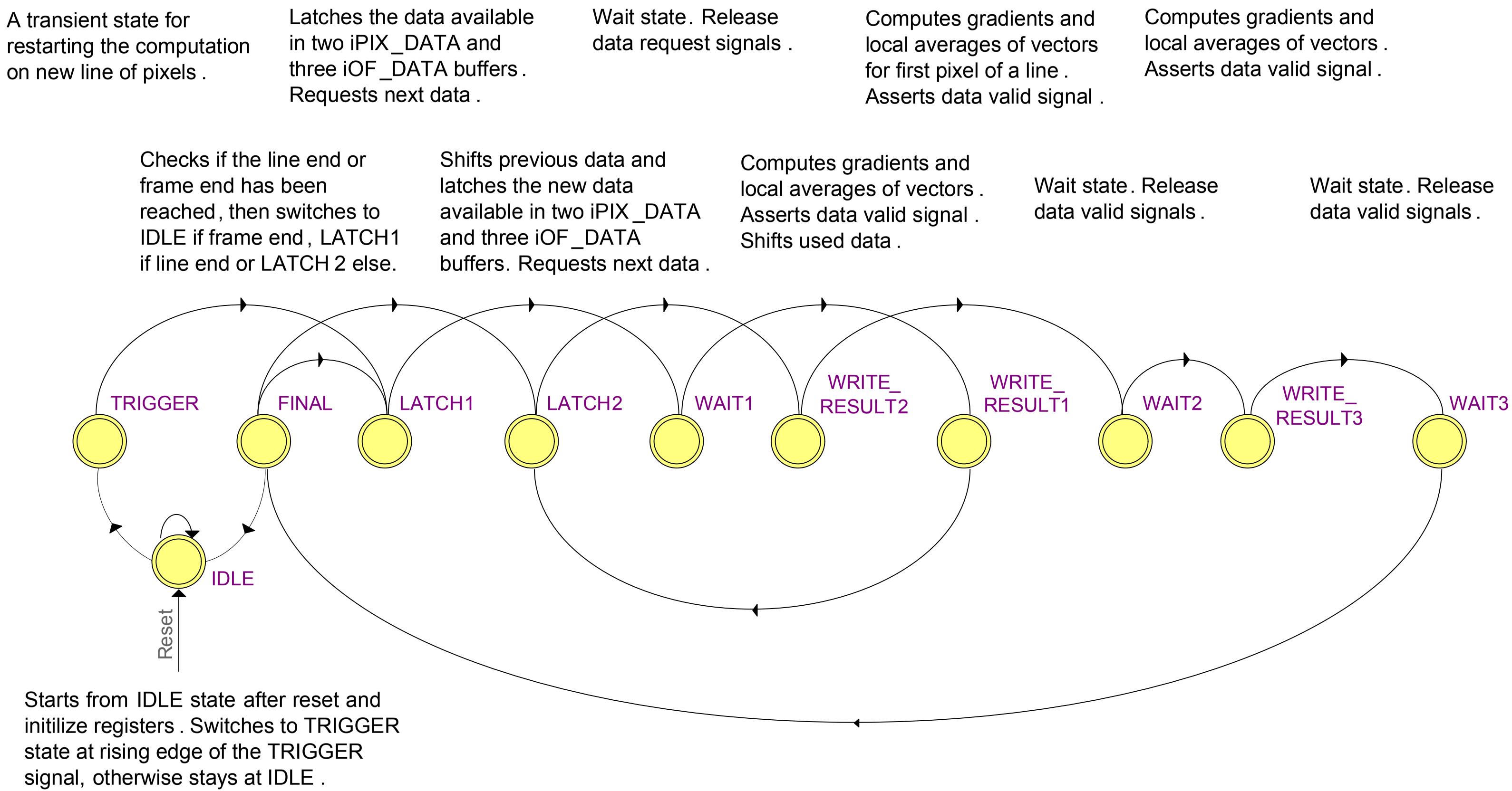

Figure 4.16: Spatiotemporal gradient computation module FSM states

4.6 Optical Flow Computation

The final stage of optical flow computation is carried out in this module. This module computes the equations of optical flow vectors given in equations (2.25) and (2.26). The I/O ports of the optical flow computation module is given in Fig. 4.17.

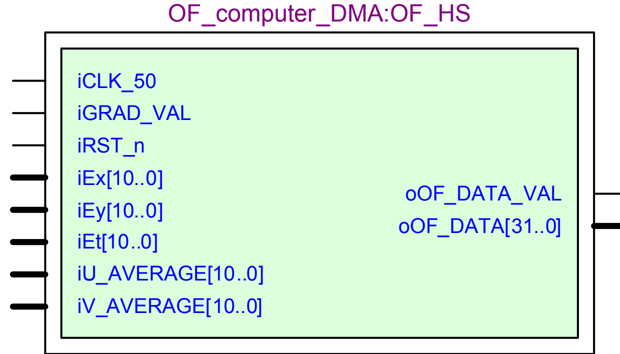

Figure 4.17: Optical flow computation module

The optical flow computation module is designed with a high performance pipelined structure that operates at 50MHz clock frequency. The terms that are required to compute these equations, spatiotemporal gradients and local optical flow vector averages, are taken as inputs to the pipeline. These signals are applied the operations that are given in Fig. 4.18 during their flow in the pipeline. The data valid signals corresponding to the input data also flow in the pipeline stages but without being exposed to any operation.

The pipeline consists of 15 stages. The first stage is the only stage that no mathematical operations are computed. At this stage the input terms of the pipeline stages are registered to prevent glitches that may appear on the input signal lines and guarantee the stable operation of the pipeline. At the next stage, the squares of spatial gradients E_x^2 and E_y^2 are computed. In parallel to this operation, $E_x\ \bar{u}$ and $E_y\ \bar{v}$ multiplications are also computed at this stage. At the third stage, both the summations of $E_x\ \bar{u} + E_y\ \bar{v} + E_t$ and $\alpha^2 + E_x^2 + E_y^2$ are computed. At fourth stage, the results of the summations are exposed to a multiplication operation with operands E_x and E_y respectively. The fifth stage consists of two division operations which compute the expressions given in (4.3) and (4.4).

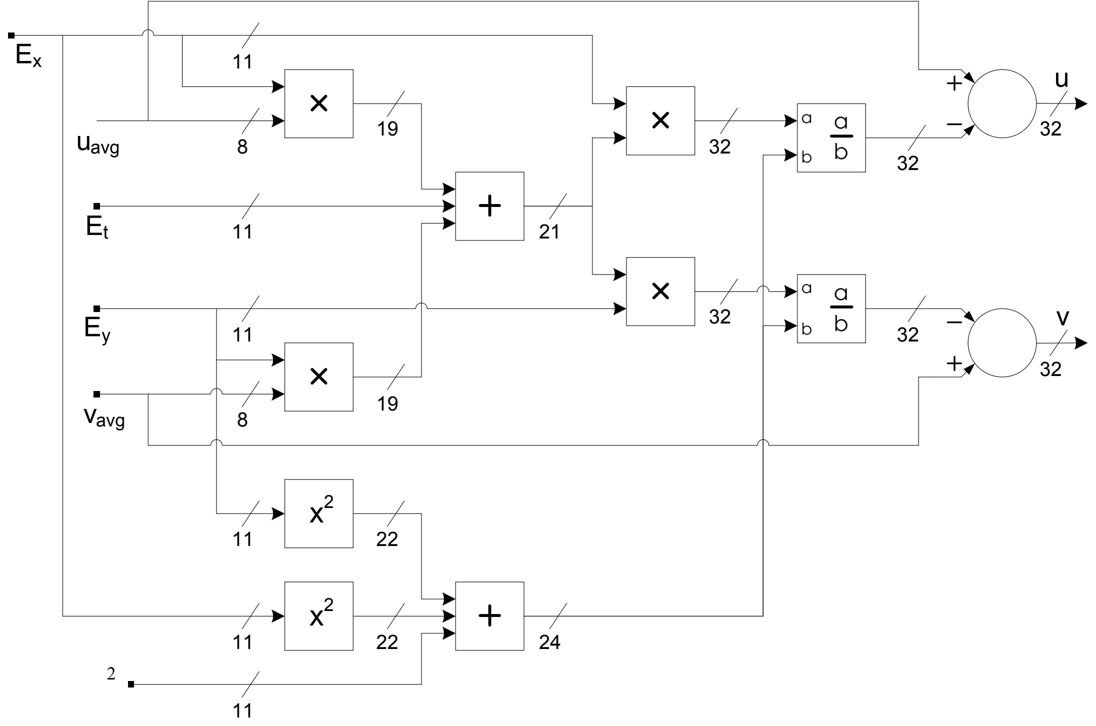

Figure 4.18: Optical flow computation module data flow diagram

$$\frac{E_x\ (E_x\bar{u} + E_y\bar{v} + E_t)}{\alpha^2 + E_x^2 + E_y^2} \tag{4.3}$$

$$\frac{E_y\ (E_x\bar{u} + E_y\bar{v} + E_t)}{\alpha^2 + E_x^2 + E_y^2} \tag{4.4}$$

As discussed in Section 4.5, the division operation is costly in terms of delay it introduce to the circuit. The 32 bits division operation can only work at a maximum frequency of 9 MHz. It is not possible to include this division operation in our pipeline that operates at 50 MHz. So each division operation is divided into 10 pipeline stages. This increases the maximum clock rate that can be applied to the division module to 55 MHz. So, the division operation with 10 pipeline stages can be included to our main pipeline design. The division operation starts in the fifth stage and the result of this operation comes out at the 14th stage of the pipeline. At 15th stage, the optical flow vectors u and v are computed by a subtraction operation. Subtracting the result of the division operation given in (4.3) from $\bar{u}$ gives u vector and similarly, subtracting the result of the division operation given in (4.4) from $\bar{v}$ gives v vector values. The

data valid signal of the corresponding result comes out from the end of the pipeline together with the resultant vector values. Finally, the computed vectors are written to the write FIFO buffer and then written back to the optical flow vector locations in the SSRAM by the DMA module explained in Section 4.4.2.

4.7 PC Communication

The FPGA implementation of the optical flow computation is tested on some sample image sequences available. To be able to compute the optical flow for these sequences on FPGA, the frames should be send to the FPGA to be stored in SSRAM. When the computation process on FPGA is finished, the results stored in SSRAM should be send back to the computer to visualize and compare with the PC implementation. So, a two way communication should be established between PC and the FPGA.

RS232, USB and Ethernet connectors are available on our FPGA development board for communication use with the PC. The maximum data rates of these protocols are listed in Table 4.3.

Table 4.3: Maximum data rates of some communication protocols.

Communication Protocol	**Maximum Data Rate**
RS232 (Practical)	920 kb/s
Ethernet (Theoretical)	100 Mb/s
USB (Theoretical)	480 Mb/s

Although the interfaces of the listed communication protocols are available on our fpga board, the hardware controllers should be designed in order to send and receive data which is a highly time consuming task. Altera provides RS232 controller free of charge with the development board. The drawback of RS232 communication over others is its low data rate. Although the amount of data to be transferred between the PC and the FPGA is large, the transfer can be done offline for testing purposes. So, despite its low data rate, we preferred to use RS232 communication to be able to put our efforts more on developing the performance of the design.

4.7.1 UART Controller

The RS232 UART controller used in the design is provided by Altera. The controller signal interface is compatible with Altera specific bus structure called the Avalon bus. The sent and received data are stored in 128 byte single clock FIFO buffers. The maximum baud rate was adjustable up to 115200. To increase the data throughput we modified the module to work at 460800. Along with the baud rate increase, the 128 byte FIFO buffers are also increased to 256 bytes to prevent data losses caused by buffer underrun occurrence. At this baud rate the transmission of a 256x256 pixels image from PC(Matlab) to FPGA takes about 1.4 seconds which is reasonable for offline testing purposes. We also tested module at 921600 baud rate but saw that the error rate increases substantially. The block diagram of the controller is given in Fig. 4.19.

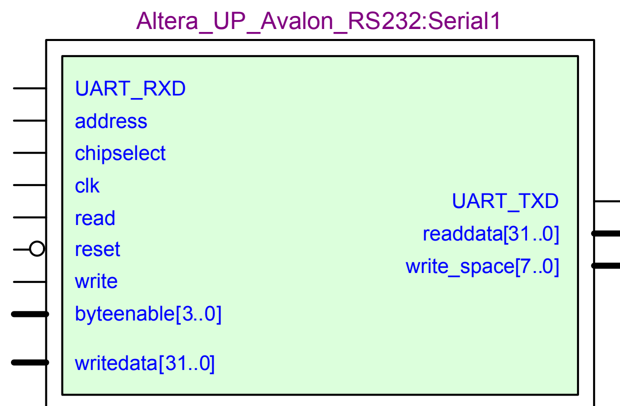

Figure 4.19: RS232 controller module terminals.

4.7.2 RS232 to SSRAM Data Transfer Module

The computation of optical flow algorithm requires two image frames to be processed. In our design the image frames are assumed to be stored in predefined address locations of SSRAM and read from there by the DMA module. This makes the design flexible to process images from different sources. The images can be written to the SSRAM that are, captured from a camera, copied from another mass storage device or transferred from a PC. In this design we implemented a module that receives the image frames from PC through RS232 communication and stores them in the SSRAM for further

processing. We also designed a module to capture images from a camera and store to SSRAM that we plan to use in our future work.

The image sequence is sent frame by frame from RS232 channel by a routine written in Matlab. The received data is written to the RX FIFO buffer by the RS232 controller module. The RS232 to SSRAM transfer module periodically polls the RS232 receive buffer counter to check if there is data available. The polling period is determined by the polling clock divider as shown in Fig. 4.20. The buffer counter signal has a latency of 2 clock cycles. So, the polling period should be more than two clock cycles, else the buffer counter may indicate wrong number of data available in the buffer.

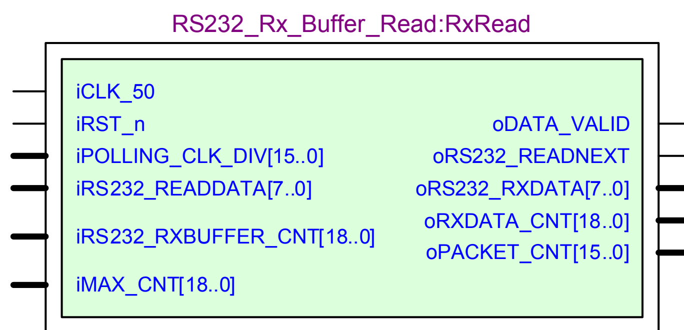

Figure 4.20: RS232 to SSRAM data transfer module terminals. This module acts as an interface for transferring image frames read from RS232 controller and written to SSRAM memory.

The module runs a FSM with a state diagram given in Fig. 4.21. FSM starts at idle state when the the module recovers from reset. At the rising edge of the polling clock, FSM switches to the polling state to check the RS232 receive buffer counter. If there is data available in the buffer, it is latched and the read command is send to the RS232 controller to delete that data from the buffer. Then the received data counter increments by one and the data is put on the output data port with data valid signal asserted. When the received data counter is equal to the defined maximum counter value, the packet count increments by one, indicating that one data packet(one frame in our case) is completely received. The packet count and the received data count are also used as the address location of the data to be written in SSRAM. The layout of the frame data stored in SSRAM is given in Fig. 4.7 of Section 4.4.2.

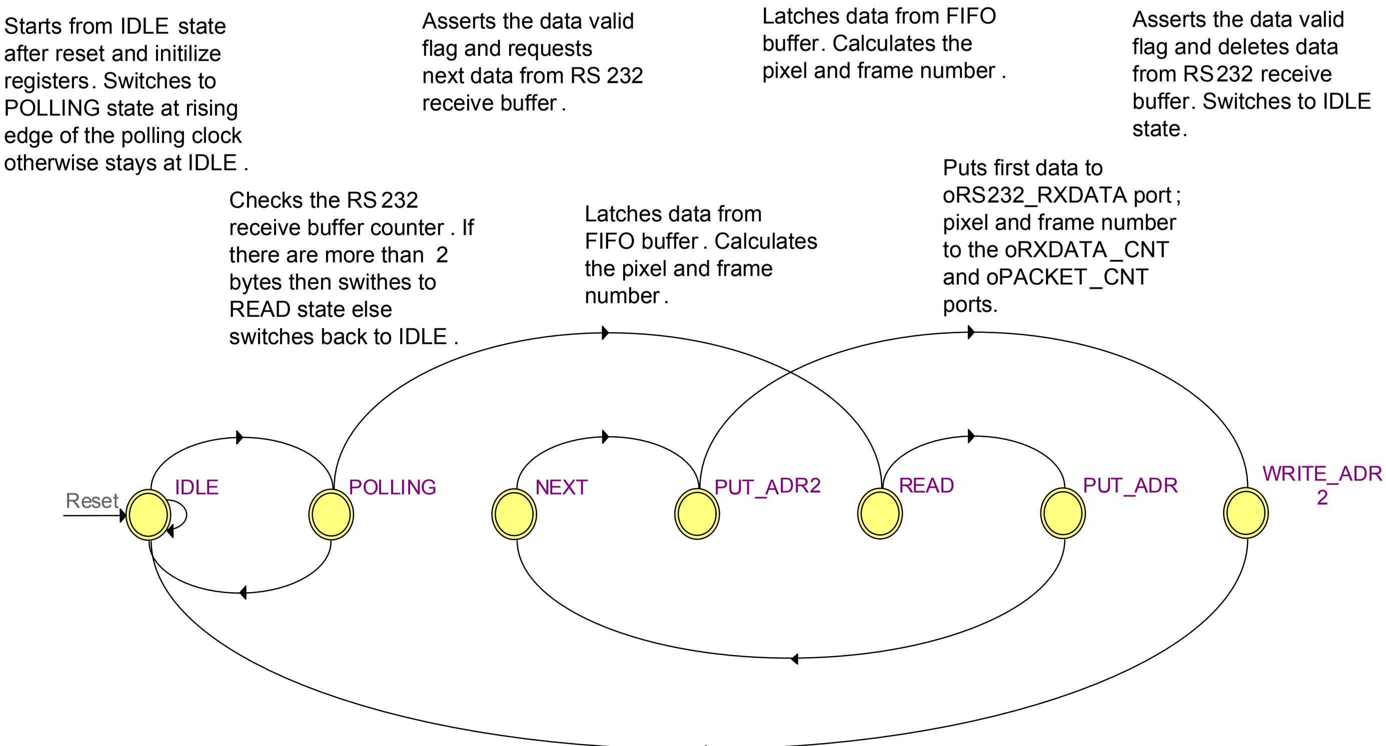

Figure 4.21: RS232 to SSRAM data transfer module FSM states

4.7.3 SSRAM to RS232 Data Transfer Module

After the optical flow computation process is terminated, the optical flow data stored in SSRAM is transfered to the PC (Matlab) through RS232 communication. The interface between the SSRAM controller and the RS232 controller is handled by this module. The I/O port diagram of the module can be seen in Fig. 4.22.

The module is designed using a FSM with 13 states. The state transition diagram of the FSM is given in Fig. 4.23. After the reset signal, the FSM starts from idle state and stays until the trigger signal comes. Then the given address interval is started to be read from SSRAM. Each address location of SSRAM stores 4 bytes of data, so the read operation fetches all of the 4 bytes at once. Then the bytes selected by the "byteenable" port of the module are written to the RS232 buffer. In our design all of the 4 bytes are used to store optical flow data. So, every read command from SSRAM is followed by 4 write command to the RS232 controller. Before initiating the write command to the RS232 buffer, the write space available in the buffer should be checked. One important point is to consider that the write space signal has a latency of 2 clock cycles. The buffer may be full when write space signal tell that there are two spaces available. So, the write space signal should be checked to be more than two spaces available in the buffer to prevent buffer overflow. The FSM continues to read and send the data until the last address location of SSRAM in the specified interval is written to the RS232 buffer. Then it swithes to the idle state until next trigger signal comes.

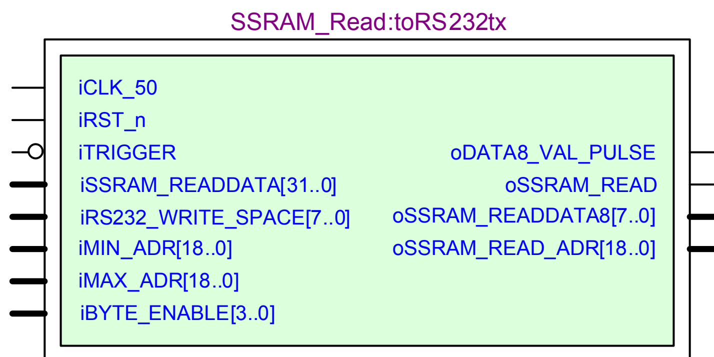

Figure 4.22: SSRAM to RS232 data transfer module

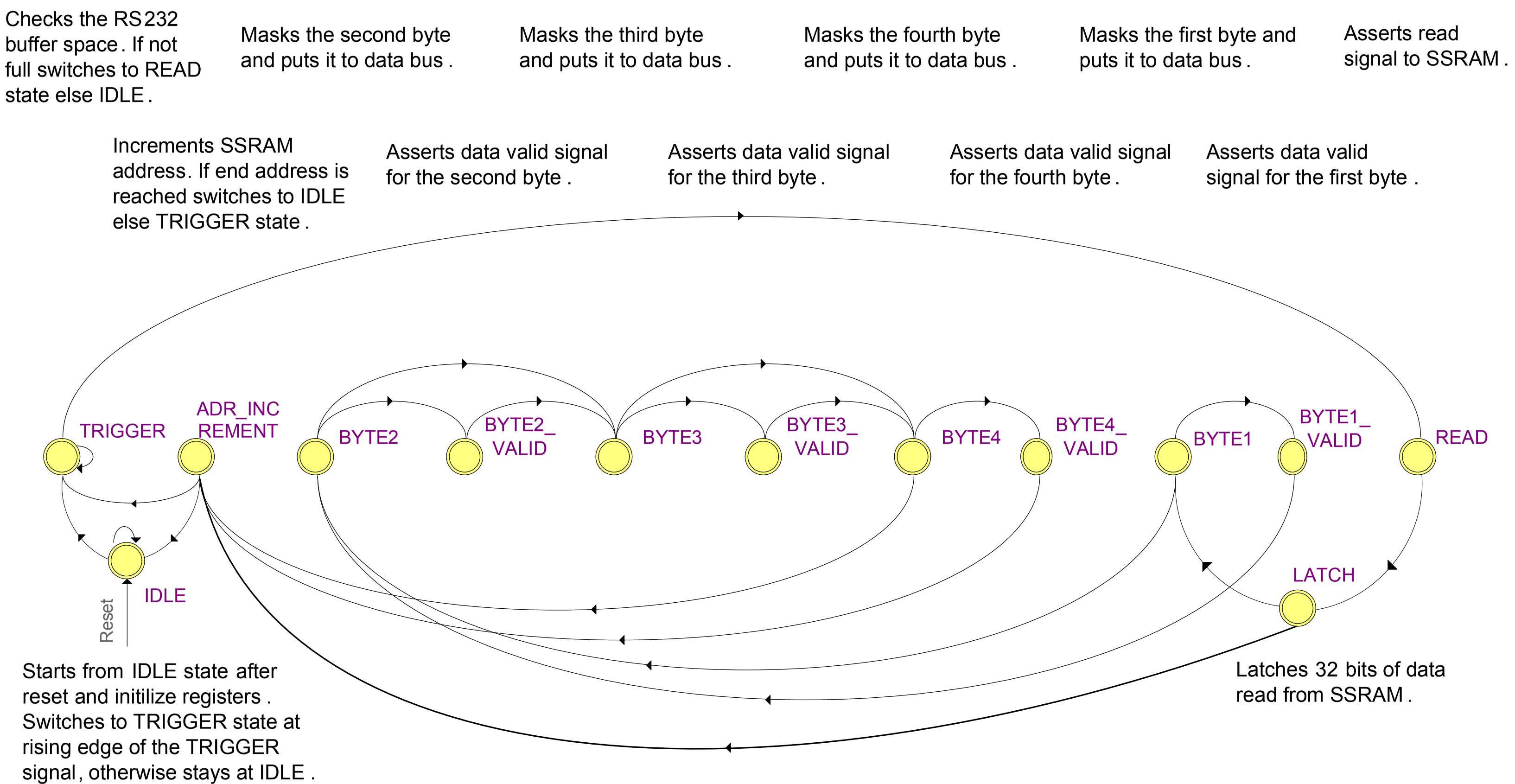

Figure 4.23: SSRAM to RS232 data transfer module FSM states

CHAPTER 5

HARDWARE DESIGN PERFORMANCE ANALYSIS AND TEST RESULTS

In this chapter we present the performance analysis of the proposed hardware design in terms of computation speed, power consumption and accuracy. The performance of the proposed hardware is analyzed both in terms of individual module and overall performance evaluations. The designed system is tested with some of the available test sequences that are frequently used for performance evaluations of the optical flow techniques in literature. Finally, we discuss the performance outcomes and make comments on the results to guide future improvements of the proposed design and similar hardware implementations.

5.1 Tests With Standard Image Sequences

5.1.1 Description of Standard Data Set

Standard test image sequences are used for evaluating the performance of the hardware implementation. We used three different real and synthetic image sequences that are frequently used in literature. The test sequences are selected from the ones that has a resolution lower than the QVGA(320x240) format. The descriptions of the selected sequences are described below according to the specifications given in [11].

The first test sequence is called "Rubik's cube" sequence given in Fig. 5.4. This real image sequence set consists of 20 image frames with 256x256 pixels resolution. In this set of images there is a motion based on the counter-clockwise rotation of a Rubik's

cube on a turntable. The motion field induced by the rotation of the cube generates pixel velocities of less than 2 pixels/frame. The velocities of the pixels on the edge of the turntable is between 1.2 to 1.4 pixels/frame and the ones on the cube are between 0.2 and 0.5 pixels/frame.

The second real image sequence is called "Hamburg Taxi" sequence whose 13th frame is given in Fig. 5.9. The images have a resolution of 256x190 pixels. The images are recorded by a fixed camera looking at a street scene with three moving cars and one walking pedestrian. A car in the left is driving to the right and a van in the right is driving to the left at a speed of 3 pixels/frame. The taxi in the middle is turning the corner at a speed of 1 pixel/frame and there is a pedestrian walking at 0.3 pixel/frame.

The last sequence is a synthetic one called "Translating Tree" sequence. It includes 40 image frames with 150x150 pixels resolution. The 8th frame of this sequence can be seen in Fig. 5.12. In this sequence the motion is based on the movement of a camera from right to left while looking at a constant scene including a tree in the front side. This movement yields a motion field between 1.73 and 2.26 pixels/frame.

5.1.2 Performance Measure

In evaluating the results obtained from the hardware computation of standard image sequences, we used a well known set of performance measures presented by Barron et al. given in [11] and a measure given in [30]. The first error measure is called "angular error" (AE) defined by Barron et al. This measure defines the angular error between a test flow vector $\mathbf{v} = (u, v)$ and the reference flow vector $\mathbf{v_r} = (u_r, v_r)$. These vectors can be defined in space-time representation as $(u, v, 1)$ and $(u_r, v_r, 1)$ in units of (pixel, pixel, frame). The angular error between the two vectors can be computed by equation given in (5.1).

$$AE = arccos\left(\frac{1 + u\ u_r + v\ v_r}{\sqrt{1 + u^2 + v^2}\ \sqrt{1 + u_r^2 + v_r^2}}\right) \tag{5.1}$$

This measure is applied to all vectors in the vector field of an image sequence. Averaging the angular errors for all vectors gives a measure for the whole vector field called "average angular error" (AAE). The standard deviation of the angular error of

vectors is also used for evaluation.

Another criteria of accuracy is the “endpoint error” (EE) defined in [30] that compares the distances between the endpoints of the flow vectors. It can be calculated using equation (5.2).

$$EE = \sqrt{(u - u_r)^2 + (v - v_r)^2} \tag{5.2}$$

We would like to clearly state here that we use this measures to compare the errors between the fixed point FPGA hardware implementation with the floating point PC implementation. The errors between the computed flow field and the ground truth of an image sequence is related with the performance of the algorithm itself and it is outside of the scope of this thesis. So, the reference vectors correspond to the vector field computed by a floating point implementation on PC and test vectors correspond to the fixed point FPGA implementation.

5.1.3 Results

We tested our hardware design using the standard test sequences explained in Section 5.1.1. The output optical flow vector field and corresponding error rates with respect to the floating point implementation on a PC is presented.

5.1.3.1 Rubik's Cube Sequence

We used the first and second frames of Rubik's cube sequence as an input. In the design, it is possible to capture the outputs of the sub-modules for debugging and evaluation purposes. In Fig. 5.1, Fig. 5.2 and Fig. 5.3, the outputs of spatiotemporal gradient computation module is given. The computation of E_x, E_y and E_t on hardware is done using a fixed point representation with 11 bits word length and 2 bits fraction. This representation is selected to compute the gradients without any errors or accuracy lost. The gradient outputs are transferred to Matlab and compared to the double precision floating point computation case. The results are confirmed to have no errors.

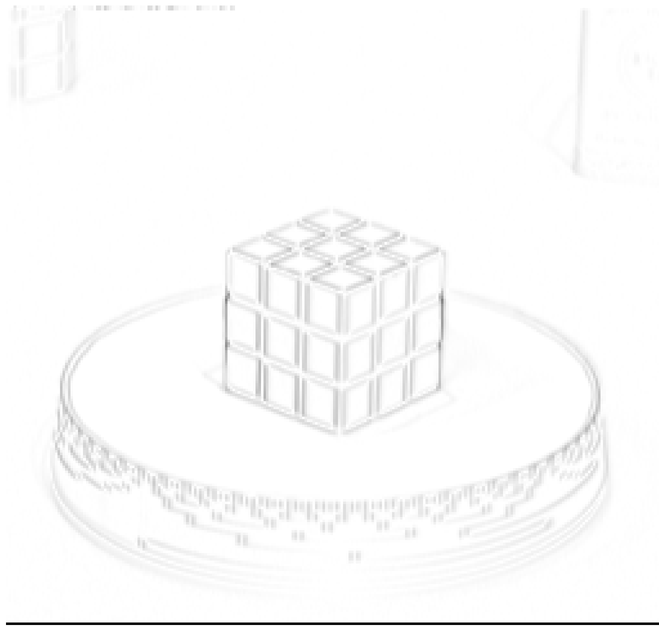

Figure 5.1: Output of the E_x gradient computation on FPGA hardware. The pixel values are inverted to get better visualization. White represents the lowest value and black represents the highest value.

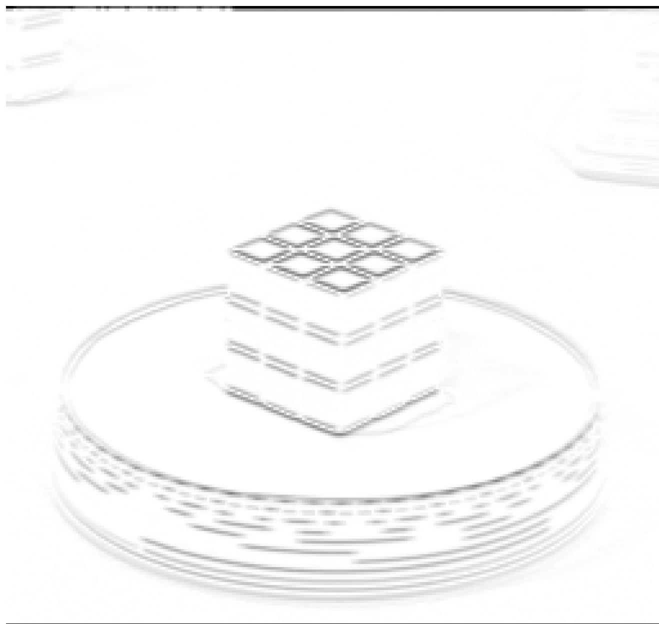

Figure 5.2: Output of the E_y gradient computation on FPGA hardware. The pixel values are inverted to get better visualization. White represents the lowest value and black represents the highest value.

Figure 5.3: Output of the E_t gradient computation on FPGA hardware. The pixel values are inverted to get better visualization. White represents the lowest value and black represents the highest value.

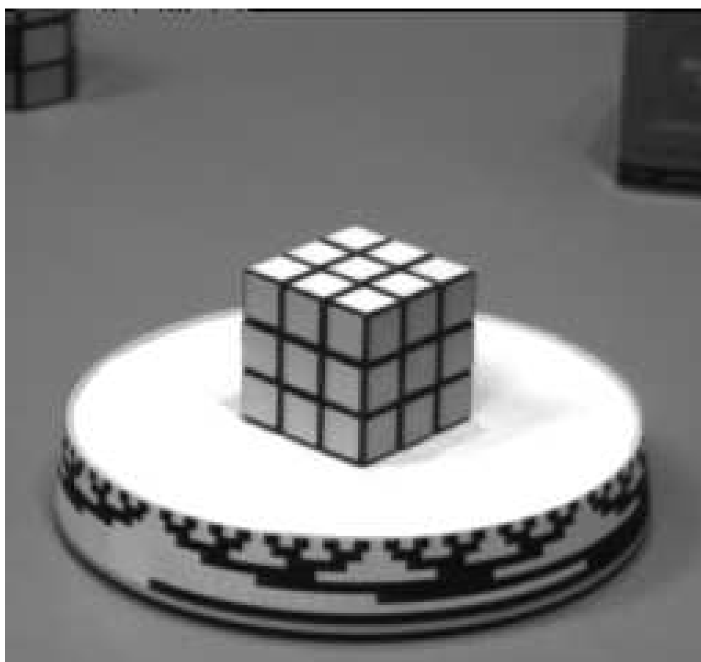

Figure 5.4: 1st frame of Rubik's cube sequence. The turntable rotates counter-clockwise with the Rubik's cube on top of it.

The computed error-free spatiotemporal gradients are input to the optical flow function computation module. The optical flow field output of this module for Rubik's cube sequence is given in Fig. 5.5. Throughout the computations in this module, the word lengths are readjusted at every operation to preserve the accuracy. The division operation at the end is computed using 32 bits numerator and 24 bits denominator. However, the result has finite fraction and it stores the real error-free result only if the remainder is zero. If the remainder is nonzero, the result is rounded to the nearest fixed point number.

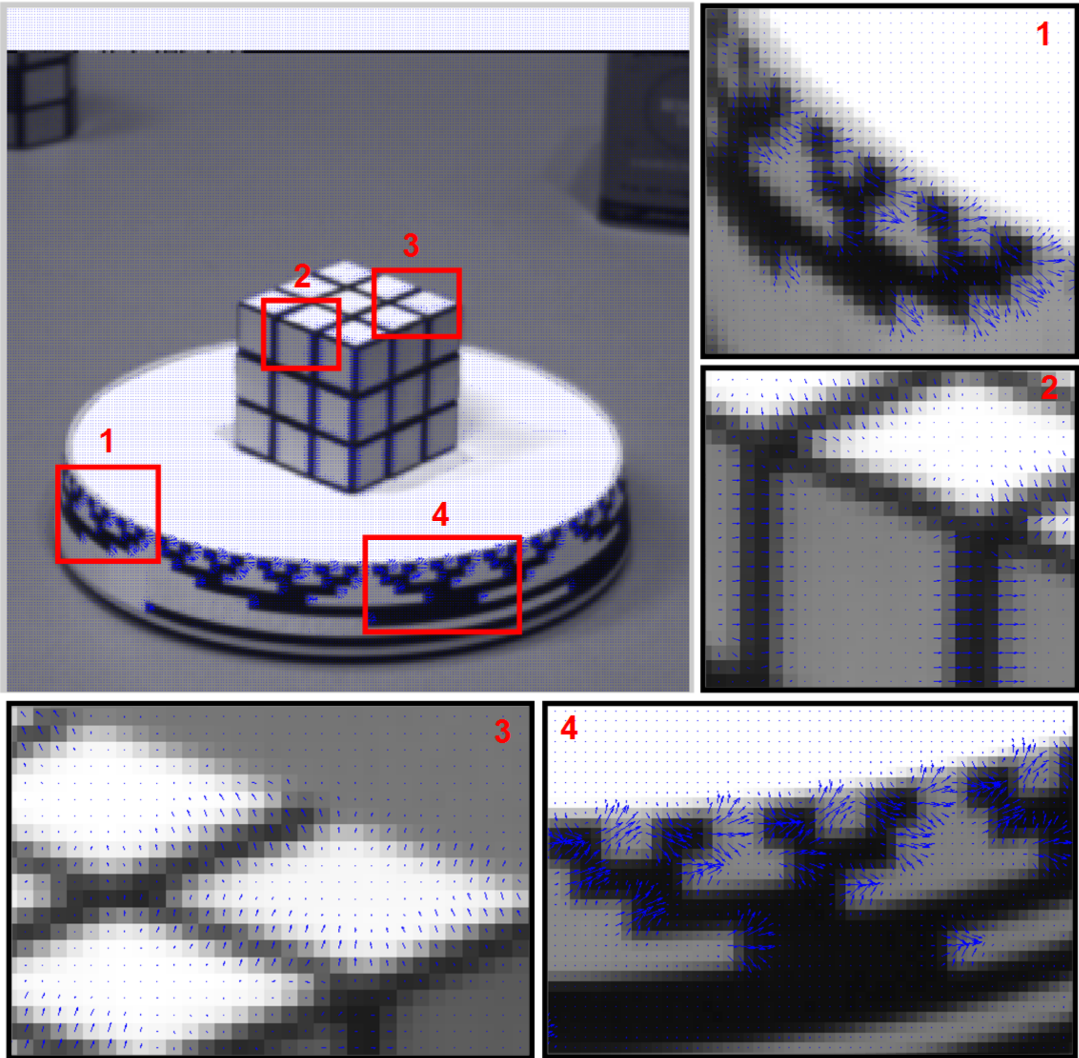

Figure 5.5: Optical flow vectors computed on FPGA hardware for Rubik's cube sequence.

The FPGA computation of Rubik's cube optical flow field is subtracted from the reference flow field computed in Matlab. The histogram of the erroneous flow vectors

are shown in Fig. 5.6. The bar graphs indicate the number of flow vectors that are within the error intervals specified at the underside of it. The shown error values correspond to the center points of error intervals that are within ±0.005 of the specified values. As shown from the graph, the histogram is piled mostly around the zero error. The maximum error is ±0.03125 which corresponds to the fixed point resolution of 4 bits fraction and the result rounding.

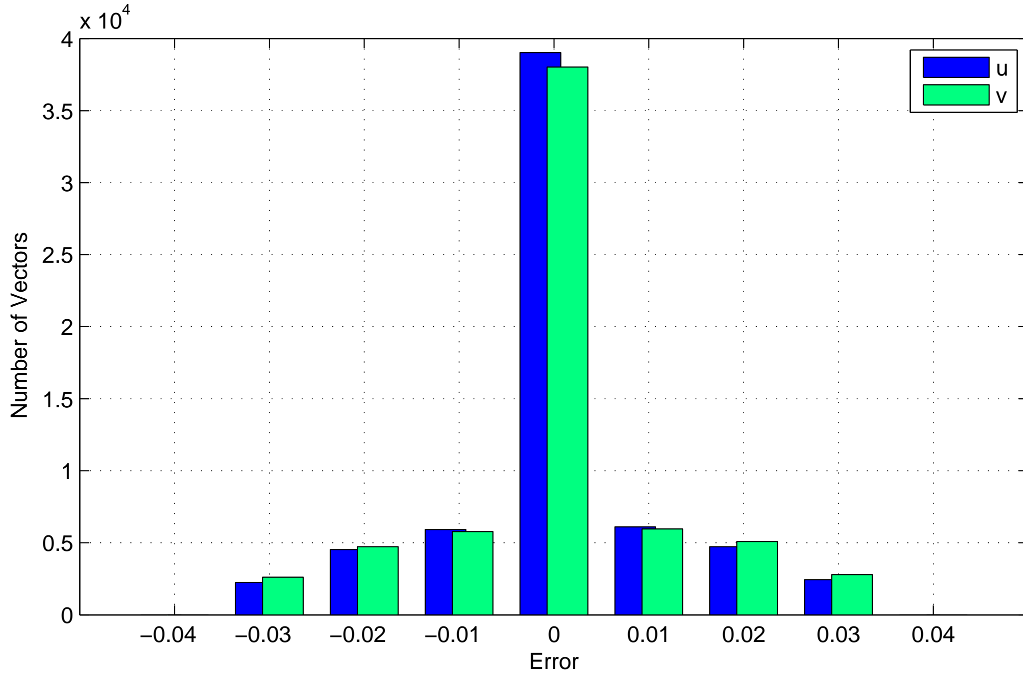

Figure 5.6: Error histogram of optical flow vectors for Rubik's cube sequence. Error values indicate the center points of ±0.005 error intervals

The accuracy measures given in Section 5.1.2 is calculated for the Rubik's cube sequence. The total error results are listed in Table 5.1. This table presents the angular error and end point error rates compared to the reference vector field. This results should be read as the deficiency caused by the approximations and precision lost between the fixed point hardware computation and the floating point PC computation. The errors in FPGA implementation over the PC implementation are caused by the approximation made in computation of local averages of optical flow vectors as explained in Section 4.5. The errors introduced by this approximation alone is given in Table 5.2. It can be easily seen that the contribution of this approximation is negligible with respect to the total error. Therefore, the main source of error in the computations originate from the fixed point implementation of division operation

explained in Section 4.6 in detail.

Table 5.1: Total error rates of Rubik's cube sequence

Error Measure	**Mean**	**STD**
Angular Error	1.002°	0.546°
Endpoint Error	0.018	0.010

Table 5.2: Partial error rates of Rubik's cube sequence caused by the approximation in computation of local averages of optical flow vectors.

Error Measure	**Mean**	**STD**
Angular Error	0.079°	0.234°
Endpoint Error	0.002	0.007

The optical flow vectors are represented by a fixed point number with 4 bits fraction and with rounding. We analyzed the error rates when the vectors are represented by less number of fraction bits or in integer. Table 5.3 shows the error rates of Rubik's cube sequence versus number of fraction bits used for fixed point representation. Referencing the error rates listed in Table 5.3, one can decide on the number of fraction bits to be used in the design according to the application and the required accuracy. Although some sensitive applications may require more accurate results, we think that the achieved accuracy is enough for most of the applications.

Table 5.3: Error rates of Rubik's cube sequence versus number of fraction bits used to represent the optical flow vector values.

Fraction	**0 bits (Int.)**	**1 bit**	**2 bits**	**3 bits**	**4 bits**	**4 bits + rounding**
Angular Error(Mean)	32.211°	19.290°	10.223°	5.149°	2.563°	1.002°
Angular Error(STD)	20.170°	12.058°	6.169°	2.857°	1.246°	0.546°
Endpoint Error(Mean)	0.765	0.381	0.189	0.094	0.047	0.018
Endpoint Error(STD)	0.495	0.236	0.110	0.050	0.034	0.010

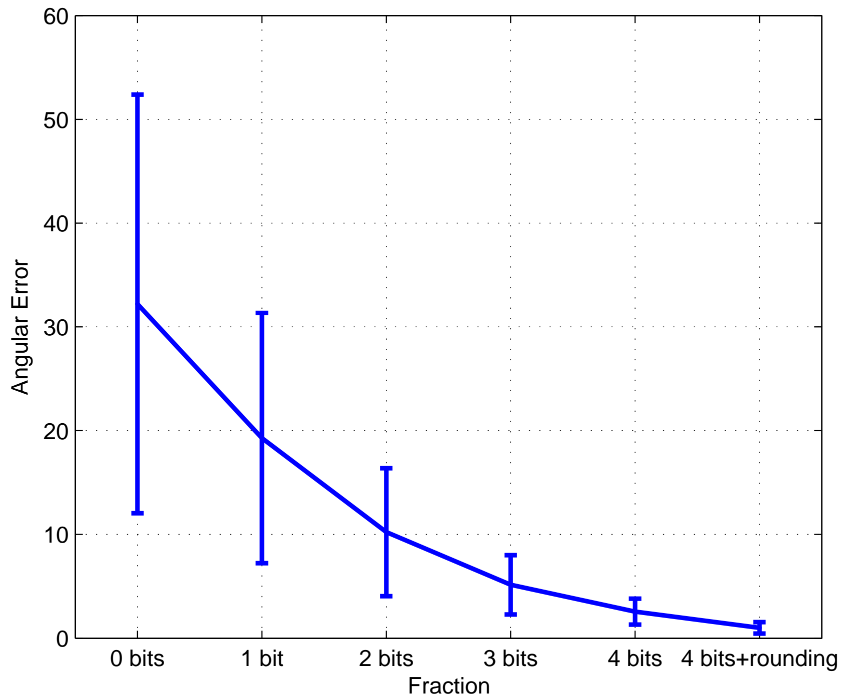

Figure 5.7: Angular error rate versus number of fraction bits used to represent the optical flow vector values.

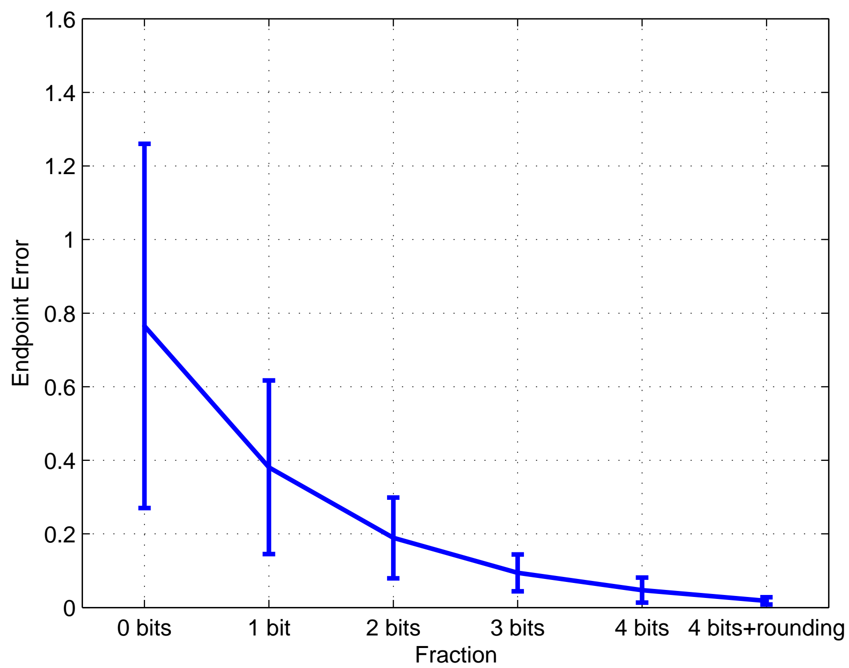

Figure 5.8: Endpoint error rate versus number of fraction bits used to represent the optical flow vector values.

The graph of error rates versus fraction bits are shown in Fig. 5.7 and Fig. 5.8. As seen from the graphs, integer representation yields high error rates on the results because, most of the optical flow vectors are generally smaller than 1 pixel/frame. As expected, the increasing number of fraction bits decreases the error rates. It can be seen that error increases exponentially for every bits of fraction. However, the number of fraction bits has an upper limit. The larger word lengths with more fraction bits requires more memory to be stored in. In our case, we are constrained by the storage of data in SSRAM. On the other hand, operands with increasing number of word lengths also increases the required time for an arithmetic operation to produce the result. This will decreases the computation performance of the hardware. An example table of maximum operating speed versus operands' word lengths of a signed division operation is given in Table 5.4. Given values correspond to operating frequencies without pipelined operation.

Table 5.4: Maximum operating frequency of a signed division operation versus the word length of its operands.

Word Length	**Maximum Frequency**
8 bits	56.09 MHz
12 bits	46.56 MHz
14 bits	34.80 MHz
16 bits	28.18 MHz
18 bits	23.71 MHz
20 bits	17.43 MHz
26 bits	13.21 MHz
32 bits	9.05 MHz

5.1.3.2 Hamburg Taxi Sequence

This is a real test sequence. The 13th frame of the sequence is given in Fig. 5.9.

The optical flow field results obtained from the FPGA implementation is given in Fig. 5.10. and the error histogram of the optical flow vectors can be seen in Fig. 5.11. The error values are larger than the previous test sequence results in this case. This

Table 5.5: Error rates of Hamburg Taxi sequence caused by the fixed point implementation.

Error Measure	**Mean**	**STD**
Angular Error	1.319°	0.509°
Endpoint Error	0.024	0.009

sequence includes comparatively smaller and larger motion vectors with respect to the previous Rubik's cube sequence. So, flow vectors with small magnitude yields more error in the computation because of the limited resolution of the fixed point representation. The corresponding angular error and endpoint error results are listed in Table 5.5.

Figure 5.9: 13th frame of Hamburg Taxi sequence. There are 4 moving objects. The car on the left and the van on the right are driving in their way, the taxi in the middle is turning the corner and the pedestrian is walking on the pavement.

Figure 5.10: OF vectors computed on FPGA hardware for Hamburg Taxi sequence.

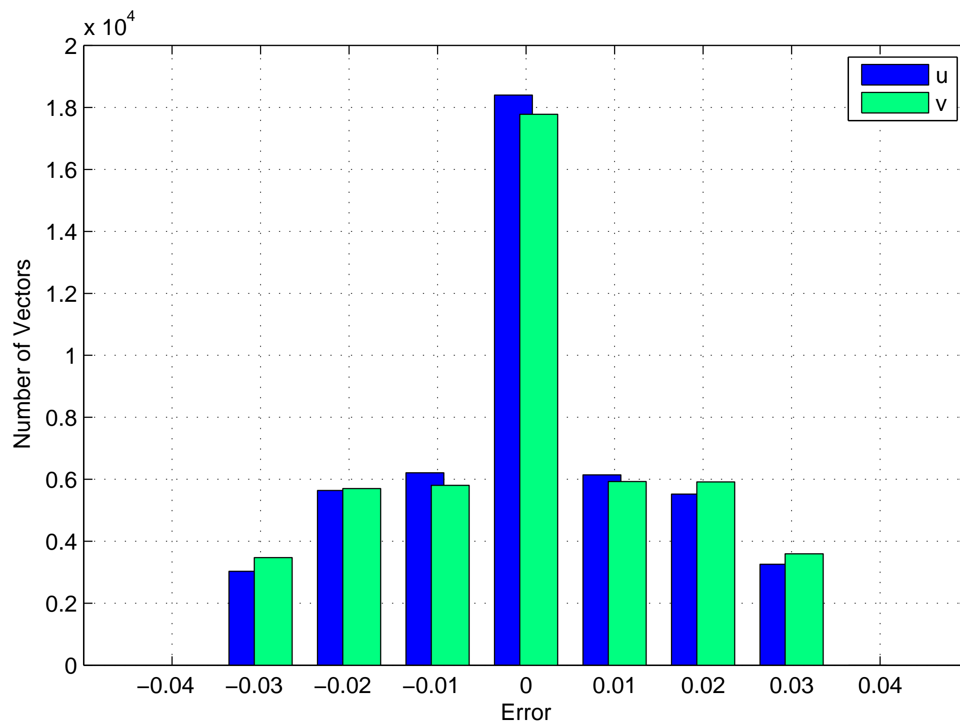

Figure 5.11: Error histogram of optical flow vectors for Hamburg Taxi sequence. Error values indicate the center points of ± 0.005 error intervals.

5.1.3.3 Translating Tree Sequence

The last test sequence we use is a synthetic one called Translating Tree sequence. The 8th frame of this sequence is given in Fig. 5.12.

In this sequence the camera translates at a constant distance and speed with respect to the scene. This yields a uniform motion field with small velocity variations. The flow vectors of translating tree sequence are close to integer displacements. This reduces the error caused by fixed point representation of vectors. Therefore, the resultant angular error and end point error results for this sequence are lower than the previous Hamburg Taxi sequence. The angular and endpoint error rates are shown in Table 5.6. The error histogram graph of the flow vectors are given in Fig. 5.13.

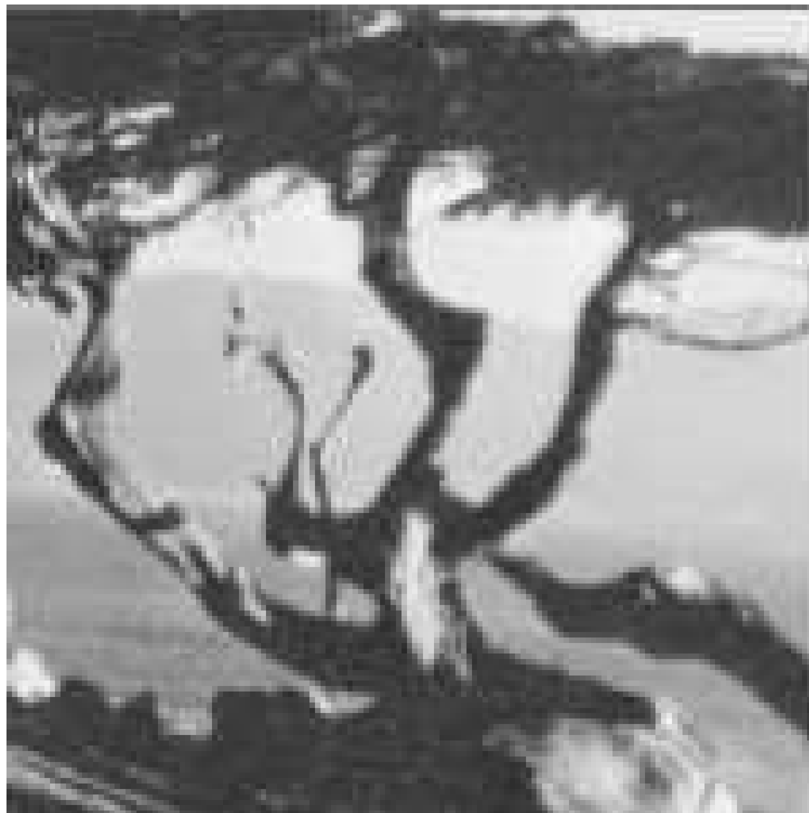

Figure 5.12: 8th frame of synthetic Translating Tree sequence. The camera is moving from right to left while looking at a constant scene including a tree in the front side. The motion field has a velocity ranging from 1.73 and 2.26 pixels/frame.

Because of the purely translational motion field at x direction, the apparent motion at y direction is zero. Therefore, the error rates of v vectors are lower than the u vectors. Ideally, if the algorithm could estimate the ground truth motion, then all the computed v vectors should be zero. Since, the representation of zero in our fixed point format has no error, the error rates corresponding to v vectors are expected to be zero also. However, the computed v vectors have representation errors since the algorithm yields nonzero values for some v vectors.

Table 5.6: Error rates of Translating Tree sequence

Error Measure	**Mean**	**STD**
Angular Error	1.045°	0.550°
Endpoint Error	0.023	0.009

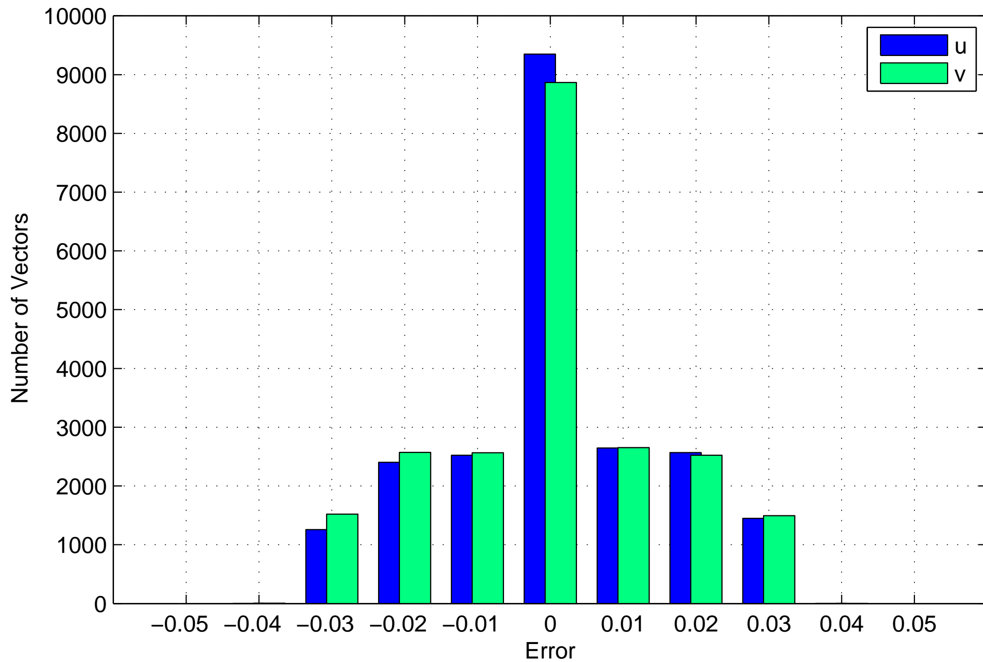

Figure 5.13: Error histogram of optical flow vectors for Translating Tree sequence. Error values indicate the center points of ±0.005 error intervals.

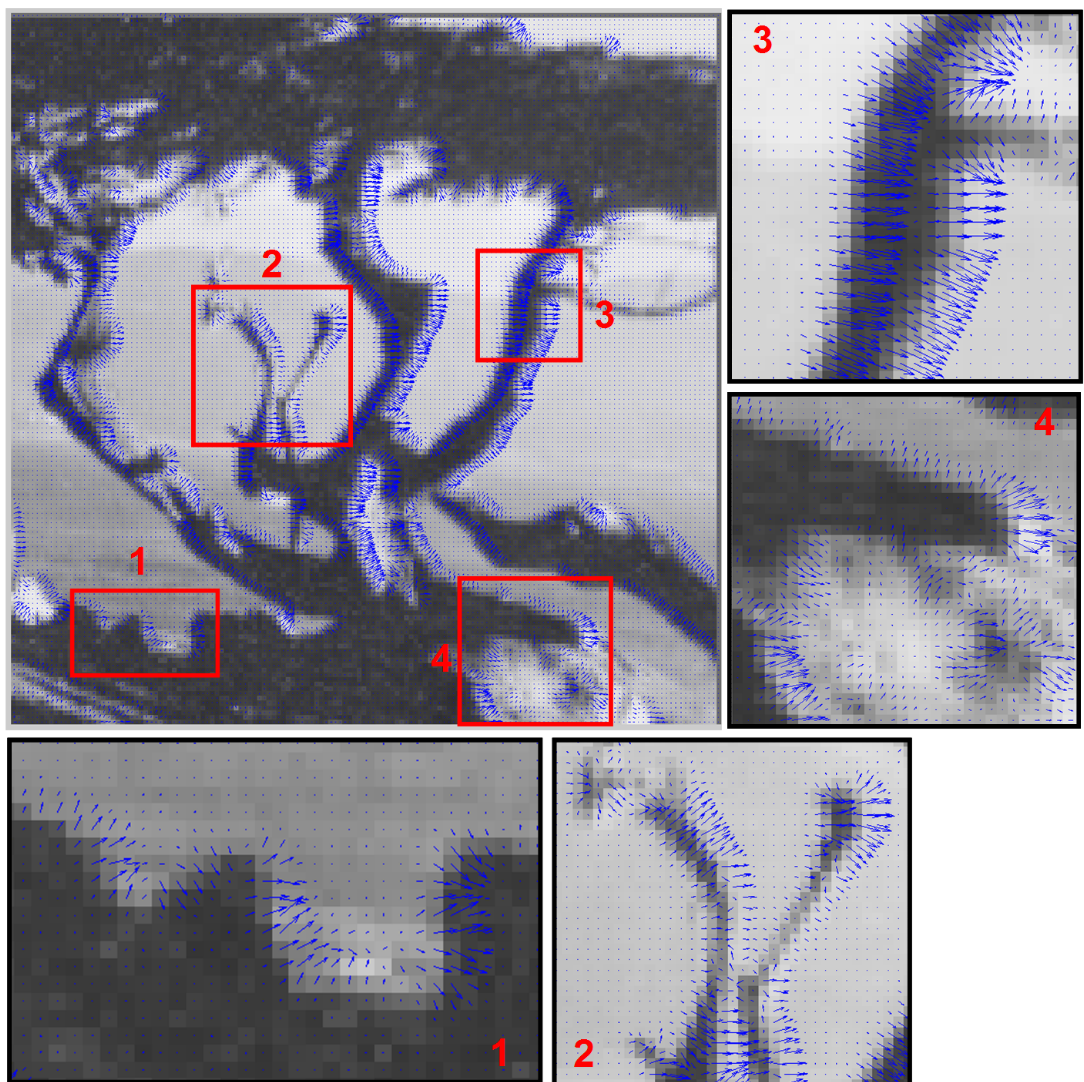

Figure 5.14: Optical flow vectors computed on FPGA hardware for Translating Tree sequence. Four regions are zoomed in to provide a closer view of the optical flow field computed on FPGA.

5.2 Performance Analysis of Designed Hardware

5.2.1 Resource Usage

The resource usage of a design is one of the performance measures of the hardware. The proposed design can only be realized if the technology of the day is able to supply the required resources of the design. On the other hand, even if there are hardware platforms that has more resources than the required resources, the price of the hardware platform is directly related with the resource it provides. Since optical flow computation is generally a pre-processing step of a computer vision system, it is even better if there is enough resources left for the implementation of further high level tasks.

The *resource usage* of an FPGA hardware design implies the use of *programmable logic elements* for implementing logic functions, the *embedded memory blocks* for storage of data to be processed, the *embedded DSP blocks* such as hardware multipliers for the implementation of arithmetic operations, *PLL blocks* to generate the required clock signals for the operation of synchronous design modules and the I/O pins used for interfacing with devices outside the FPGA chip. The floorplan of the designed hardware fitted on a EP2C70 device is given in Fig. 5.17.

In Table 5.7, the resource usage of the proposed hardware is listed with the corresponding resources available on the FPGA device included in the development board we used. The device resource utilization percentages are also given at the last column of the table.

Table 5.7: Resource usage of the overall design and available resources on the FPGA device.

	Design usage	**EP2C70 Resources**	**Utilization**
Logic Elements	8,086	68,416	11.9 %
Embedded Memory bits	151,772	1,152,000	13.2 %
Hardware Multipliers	6	150	4.0 %
PLL Blocks	1	4	25.0 %
I/O Pin Count	262	622	42.1 %

As seen from Table 5.7, our proposed design utilize far less than the half of the resources available on EP2C70. Low resource usage is beneficial to implement high level tasks on the same device. Also, another option may be to select a cheaper device with less resources and lower power consumption.

The overall resource usage, is partitioned according to the amount of resources utilized by individual design modules. The resource usage of modules is analyzed in terms of the number of Logic Elements (LE), registers, look up tables (LUT), embedded memory bits and I/O pins used and their percentages to the the whole design. In Table 5.8, resource utilization of individual design modules are listed. In Fig. 5.15 and Fig. 5.16 the distribution of utilization percentages are shown on a pie chart.

The highest number of logic elements are used by the optical flow function computation module. It consumes nearly half of the LEs used by the whole design. As we explained in Section 4.6, this module includes many arithmetic operations. Since the word lengths of operands are kept large to preserve accuracy, the mathematical operations need a high amount of logic resources to be implemented.

Embedded memory bits are mostly used by the fifo buffers of DMA module for storing data that is read from and written to the SSRAM. The hardware multipliers are used by arithmetic operations in computation of the optical flow equation. Most of the I/O pins of the device are used by the SSRAM chip. The rest is used by RS232 communication, clock inputs, LEDs, switches and push-buttons.

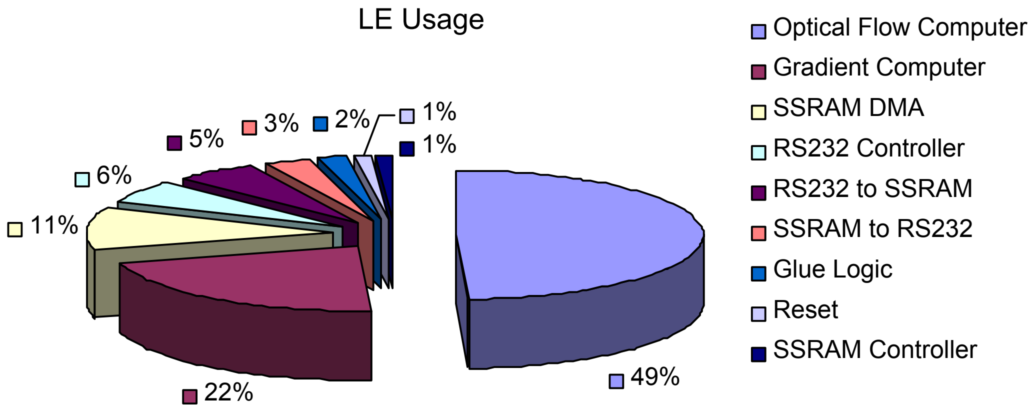

Figure 5.15: LE usage percentage of modules

Table 5.8: Resource usage of individual design modules

	Module Name	LE Usage	Register Usage	LUT Usage	Memory Usage	Pins Usage
1	Optical Flow Comp.	3,976 (49.2%)	1,282 (40.2%)	2,694 (55.0%)	220 (0.1%)	0
2	Gradient Computation	1,754 (21.7%)	424 (13.3%)	1,330 (27.2%)	0 (0.0%)	0
3	SSRAM DMA	874 (10.8%)	662 (20.8%)	212 (4.3%)	147,456 (97.2%)	0
4	RS232 Controller	468 (5.8%)	332 (10.4%)	136 (2.8%)	4,096 (2.7%)	2
5	RS232 to SSRAM	424 (5.2%)	202 (6.3%)	222 (4.5%)	0 (0.0%)	0
6	SSRAM to RS232	250 (3.1%)	156 (4.9%)	94 (1.9%)	0 (0.0%)	0
7	Glue Logic	162 (2.0%)	58 (1.8%)	104 (2.1%)	0 (0.0%)	160
8	Reset	96 (1.2%)	66 (2.1%)	30 (0.6%)	0 (0.0%)	1
9	SSRAM Controller	82 (1.0%)	6 (0.2%)	76 (1.6%)	0 (0.0%)	69
TOTAL		**8,086**	**3,188**	**4,898**	**151,772**	**232**

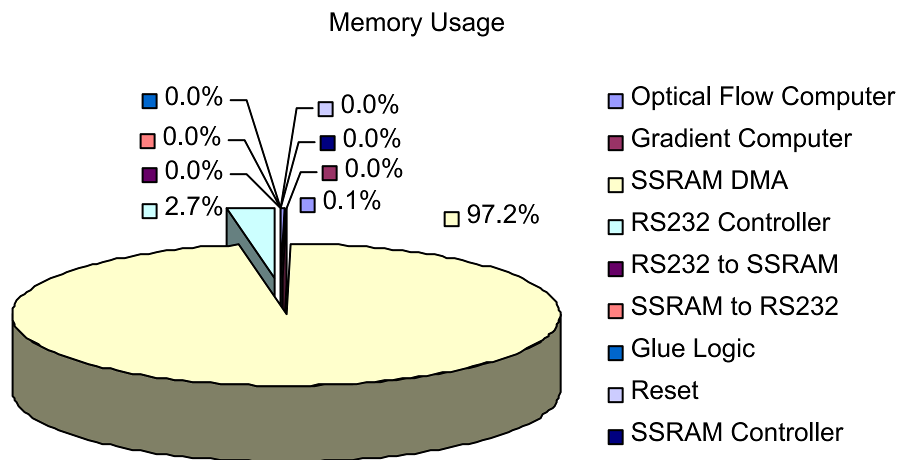

Figure 5.16: Memory usage percentage of modules

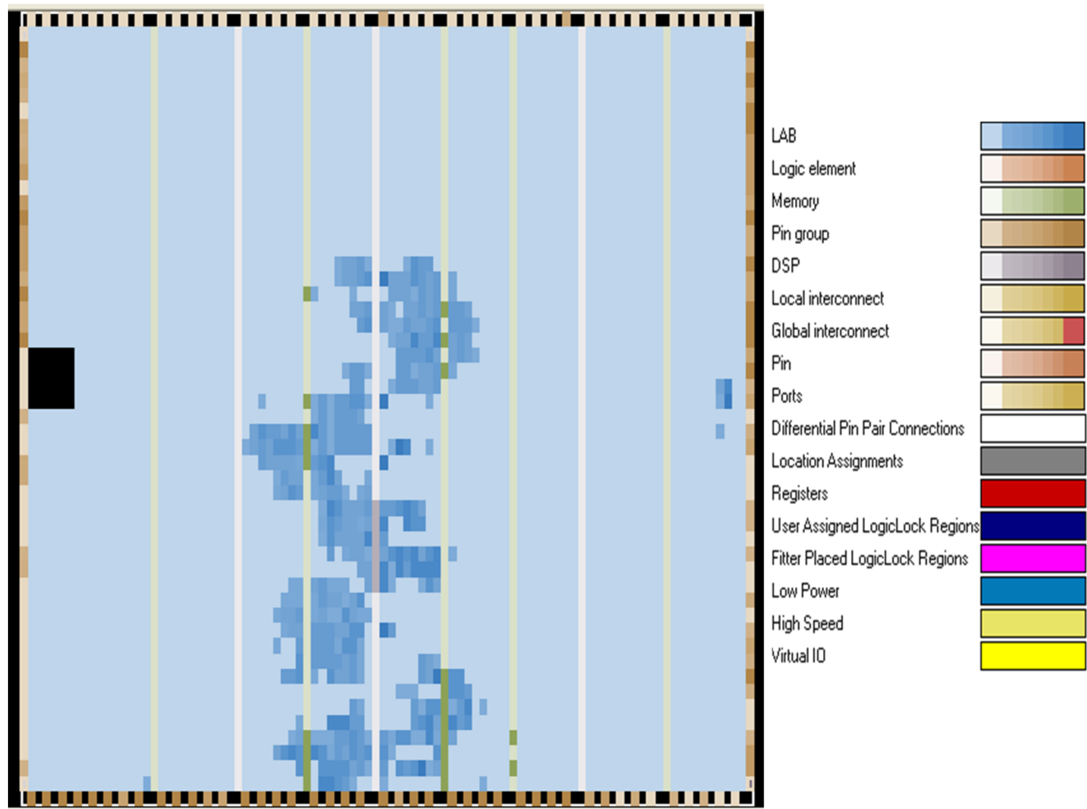

Figure 5.17: Floorplan of the designed hardware fitted on a EP2C70 device. The schematic shows the layout of the resources on the chip that are used to implement the design. The color legend is given next to the figure. The darker color of a particular resource indicates the higher usage ratio of that resource.

5.2.2 Power Consumption

Power consumption is an important performance measure of an hardware for utilization in mobile robotic platforms. In this section, we present the power consumption of the proposed design. The power usage of hardware is analyzed in terms of dynamic, static and I/O blocks power dissipation.

Dynamic power consumption is caused by the transition of signals. To change a logic state of a wire, the parasitic capacitance should be charged to the new voltage level of the logic state by drawing the required current from the supply. So, dynamic power consumption is higher on nodes with high switching rates. On the contrary, static power consumption is independent from the switching behavior of the circuit. It is mainly caused by the leakage currents and remains constant even when the circuit is not switching(idle). It is only affected by the number of logic used in the circuit. More resource usage yields an increase in the static power consumption too.

Table 5.9: Total power dissipation

Core Dynamic Thermal Power Dissipation	216.55 mW
Core Static Thermal Power Dissipation	165.79 mW
I/O Thermal Power Dissipation	462.04 mW
Total Thermal Power Dissipation	**844.38 mW**

The power consumption is related with the voltage level of the power supplies and the currents drawn from them. There are two different supply voltages used by the hardware. To decrease the power consumption, lower voltage levels are used for internal logic and to interface other devices to the FPGA higher voltage levels are used for I/O pins. The voltage levels and the currents drawn by the designed hardware is listed in Table 5.10.

As seen from Table 5.9 and Table 5.10, more than half of the power consumption is caused by the I/O terminals of the FPGA. This is mainly caused by the high supply voltage level of the I/O pins and the number of I/O channels utilized for connection interface of external devices such as SSRAM. To reduce the power consumption, the

Table 5.10: Current drawn from supply pins

	VCC Core (1.2V)	**VCC I/O (3.3V)**
Dynamic Current Drawn	189.63 mA	118.95 mA
Static Current Drawn	148.43 mA	13.59 mA
TOTAL Current Drawn	338.06 mA	132.54 mA

number of I/O pins used should be decreased. The use of an SSRAM memory with 16 bits interface instead of 32 bits can decrease the power consumption. However, there is again a trade off between the power consumption and the computation performance. This will decrease the memory bandwidth and eventually the computation time will increase too.

The overall power consumption is partitioned between the sub-design modules in Table 5.11 and the percentages are visualized on pie charts given in Fig. 5.18, Fig. 5.19 and Fig. 5.20. Table 5.11 indicates the dominant modules effecting the whole power consumption the most. As seen from the table, DMA module has the highest power consumption both in terms of dynamic, static and routing power consumptions. Although it does not use the highest resources amongst the other modules, the main reason of its high dynamic power consumption is its high operating frequency of 200 MHz. The high static power consumption is caused by the high resource usage of this module. Although its LE usage is not as much as some other modules, it utilizes lots of memory resources. The main reason of static power consumption of this module is the high leakage currents of embedded memory blocks. Using that much memory resource also causes many routing connections to be made between the module and the memory blocks. This results in a high amount of routing power consumption together with the high operating frequency.

Another module that operates at 200 MHz is the SSRAM controller. However, it can be seen from the table that it has the lowest dynamic and static power consumption. This is because of the small amount of combinational and sequential structures used. This module mostly include routing interfaces between SSRAM ports and the modules accessing the SSRAM. So this results in a high routing power consumption when

Table 5.11: Power consumption of individual modules

	Module Name	**Total Power Consumption**	**Dynamic&Static Consumption**	**Routing Pow. Consumption**
1	SSRAM DMA	236.88 mW (61.9%)	179.49 mW (72.0%)	57.39 mW (43.1%)
2	Optical Flow Comp.	55.90 mW (14.6%)	29.44 mW (11.8%)	26.46 mW (19.9%)
3	Gradient Comp.	37.73 mW (9.9%)	20.28 mW (8.1%)	17.46 mW (13.1%)
4	Glue Logic	17.50 mW (4.6%)	5.73 mW (2.3%)	11.77 mW (8.6%)
5	Reset	13.68 mW (3.6%)	0.69 mW (0.3%)	12.99 mW (9.8%)
6	RS232 Controller	9.87 mW (2.6%)	7.61 mW (3.1%)	2.26 mW (1.7%)
7	RS232 to SSRAM	4.55 mW (1.2%)	2.87 mW (1.1%)	1.68 mW (1.2%)
8	SSRAM to RS232	4.12 mW (1.1%)	2.91 mW (1.2%)	1.21 mW (0.9%)
9	SSRAM Controller	2.54 mW (0.7%)	0.32 mW (0.1%)	2.22 mW (1.7%)
TOTAL		382.77 mW	249.33 mW	133.44 mW

compared to its dynamic and static power consumption.

The rest of the modules in the design operates at 50 MHz. The gradient and optical flow function computation modules are the two most power consuming modules amongst them. Although they operate at lower frequency, the high resource usage in both combinational and sequential structures is an effect to increase the static and dynamic consumption. The main difference between other modules is their considerable amount of arithmetic resources used. This dramatically increases the consumption of these modules in all three power types.

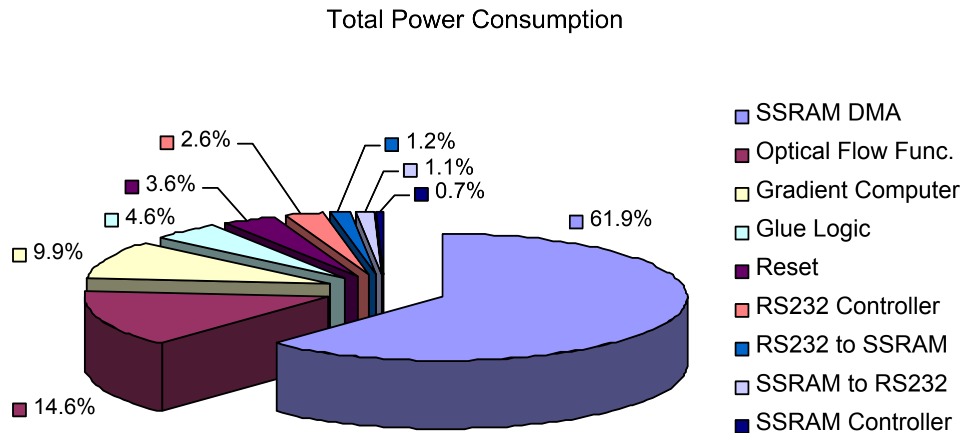

Figure 5.18: Pie chart represents the total power consumption partitioned among the design modules according to their percentages. More than half of the total power is consumed by the DMA module because of its high operating frequency and high resource usage. It is followed by the OF function and the Gradient computation modules.

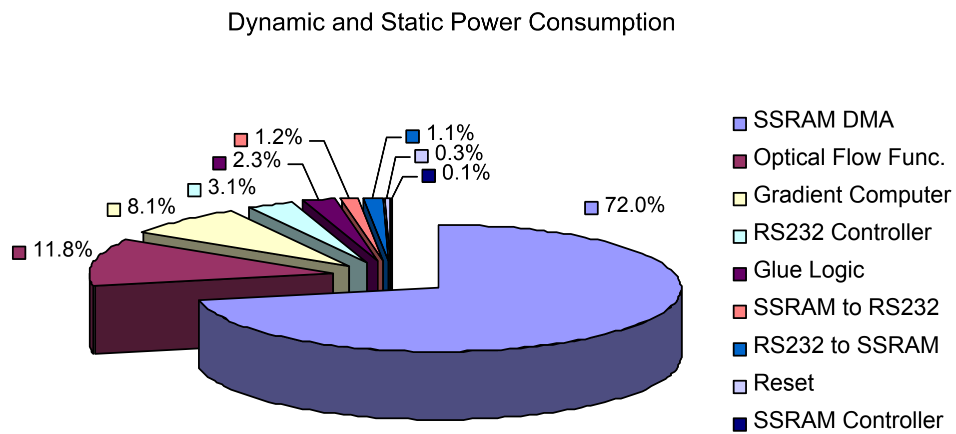

Figure 5.19: Pie chart represents the dynamic and static power consumption partitioned among the design modules according to their percentages. Nearly 3/4 of this power type is consumed by the DMA module because of its high operating frequency and high resource usage. It is followed by the OF function and the Gradient computation modules.

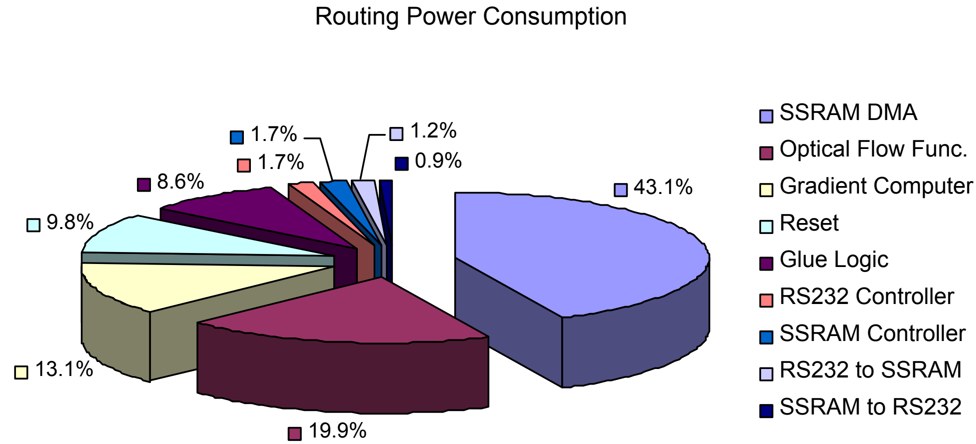

Figure 5.20: Pie chart represents the total routing power consumption partitioned among the design modules according to their percentages. Nearly half of the routing power is consumed by the DMA module because of its high memory usage. It is followed by the OF function and the Gradient computation modules.

5.2.3 Computation Time

In the beginning of this thesis work our aim was to design a hardware that can compute optical flow *fast enough* to be utilized in robotics applications. In mobile robotic vision applications, low resolution cameras that has resolutions of QVGA or QCIF format are frequently used because of computational considerations. In some applications, image resolutions down to 100x100 pixels are still enough for gathering the required information from the image sequence. However, even these low resolutions are a problem to process in realtime on the mobile robotic platforms. In many cases realtime operation requires 30 frames to be processed in one second which corresponds to approximately 33ms of processing time. However, optical flow should be computed much faster in applications where optical flow data is used in computations of other high level tasks. The required computation time of optical flow data depends on the application. We determined our success measure in computation time as achieving a minimum frame rate of twice the realtime operation which corresponds to processing an image sequence with a resolution lower than QVGA format at a minimum of 60 fps.

For testing the computation time of the designed hardware, we utilized a standard image sequence given in Fig. 5.4 called “Rubik’s Cube” which is frequently used in literature. The first and second frames with a resolution of 256x256 pixels are used to measure the computation time. After transferring of the test image frames from Matlab to the FPGA is completed, the optical flow computation process is started with a button pressed. While the computation is in progress a register flag is pulled high. The time elapsed during the computation process is measured by an oscilloscope that is connected to the corresponding I/O pin of the flag register. The width of the pulse captured by the oscilloscope gives the total computation time. For the mentioned test image frames the computation is finished in **3.89ms**. This corresponds to processing of 257 fps which is much higher than the aimed frame rate of 60fps.

5.3 Comments on Analysis & Results

The accuracy of the designed hardware is tested using three different test sequences including real and synthetic data. The implementation method used in arithmetic operations affects the accuracy dramatically. We tested different hardware implementations using integer representation to fixed point implementations using 1 to 4 bits fraction and different word lengths. The word length adjustments are done with or without result rounding. Each method has many consequences that effect accuracy, computation time, power consumption, memory and other resource usage. However, there is no optimal set of configuration to be used. The best design configuration, depends on the requirements of the application that the hardware is planned to be used for. The presented results in this chapter can be taken as a design guide for determining the most suitable approaches.

The floating point representation of the mathematical operations improves the accuracy but increases both the memory and logic elements usage at the same time. More memory usage also increases the computation time and more logic element resource usage increases the power consumption.

The multi clock design method we used boosts the performance dramatically. In the design, the higher clock frequency(200MHz) that is used for memory accesses is one of the main factors that cuts down the memory read/write times and so as

the computation time. However, increase in clock rate directly increases the power consumption.

The DMA usage introduces a trade off between the memory usage and the computation time. An alternative and simpler method is performing memory access operations by the gradient and optical flow equation computation modules without using large fifo buffers which saves a lot of internal memory resource. However, these modules operates at 50MHz and each read operation from memory takes 3 clock cycles. Each optical flow computation needs 8 pixel values and 8 vector values to compute the resultant optical flow vectors. The operations can only be carried out after all required values are read from the memory. This method reduces the SSRAM bus utilization and increases the computation time dramatically. So, we suggest utilizing a DMA operation whenever there is no strict memory constraints.

It can be seen from the results that there is a trade off between resource usage, power consumption, computation time and accuracy. There are also other constraints on these parameters. These constraints are both introduced by the specifications of the hardware platform we used to implement the proposed design and the minimum performance objectives we set at the beginning of this thesis work.

Besides these trade offs, the main advantage of the hardware design comes from the parallelized and pipelined implementations of the functions. The hardware can compute a number of operations at once. Moreover, the consecutive operations can be done in a pipelined structure. When these operations are applied to a large amount of data, the pipelined structure yields a high data throughput.

The floating point PC implementation of the algorithm shows a low performance as expected. The PC platform has an Intel Core2 processor operating at 1.66 GHz and 1GB memory at 667MHz clock rate. The processor has a power rating of 34 watts as stated in the datasheet. The operating system is Windows XP and the implementation of the algorithm is done in Matlab R2008b. The computation of optical flow field on a 256x256 pixels images on this platform takes 0.571 seconds which can only achieve approximately 2 fps.

As stated in Section 5.2.3, the proposed hardware needs only 3.89ms to compute the

flow field on the same image sequence. This corresponds to 146 times increase in the computation performance. The power rating of the design is 844.38 mW which is only the 1/40 of the power consumed by the 1.66 GHz processor.

In addition to the high performance, the resource usage of the design is very low. This leaves a large amount of resources unused. These resources may be used for further computations or the FPGA device can be replaced with a cheaper one. There are smaller devices available in the same Cyclone II family as EP2C70 as listed in Fig. 3.1. If a custom PCB is to be designed then the use of smaller devices such as EP2C15 to EP2C35 would be beneficial in terms of cost and power consumption but still providing sufficient resources.

If the design is wanted to be modified or improved to suit different applications, some constraints should be considered. For example, when one requires a higher frame rate operation, a straightforward approach would be to increase the operating frequencies of synchronous modules. However, the design operates near to the maximum achievable clock frequencies. The design can be modified to increase the frequency by using more pipeline stages. Or, to increase the maximum achievable clock frequency without any change in the design, a higher speed grade device can be used. In both ways it should be noted that higher clock rates will eventually introduce a power penalty.

Another need may be to increase the image size to be processed. In current implementation, the image size is constrained with the available memory resources on board. To increase the image size, a higher capacity memory should be used. Dynamic memories(DRAMs) can provide the required memory space, however, the low bandwidth of them may cause a decrease in maximum achievable frame rate.

It can also be required to process images taken from multiple cameras. In this scenario, the design can be easily duplicated as much as the FPGA device meets the required resources. However, since there is a single memory, the memory accesses should be multiplexed for each camera. Being already the bottleneck of the design, further multiplexing of memory will decrease the frame rate dramatically. Therefore, we suggest using dedicated memories for processing images taken from each camera.

CHAPTER 6

CONCLUSION

In this thesis, we presented the design and implementation of a high performance FPGA hardware with a small footprint and low power consumption that is capable of providing over-realtime optical flow data. The motivation behind this work was the lack of a suitable hardware for mobile robotic platforms that is capable of computing optical flow vector field in real time which is a factor that prevents the mobile robotics community to efficiently utilize some successful techniques presented in computer vision literature. Our motivation and similar studies presented in literature are explained in the first chapter. The results and weaknesses are discussed to state the possible improvements.

We implemented a well known optical flow algorithm proposed by Horn & Schunck which is briefly explained in Chapter 2. It yields a high density optical flow vector field with reasonable accuracy. This method is suitable for implementation on hardware using low logic and memory resources and delivering a high performance. The arithmetic operations involved in the computation can be implemented using fixed point representation with small word lengths.

In Chapter 3, we discuss the requirements of the hardware platform in terms of resources that will be used in implementation of the optical flow computation. We explained the selection criteria and the specifications of the DE2-70 FPGA development board. The DSP blocks provide an efficient implementation of arithmetic operations involved in the optical flow computation and SSRAM is a suitable memory for massive data transfers such as image frames. RS232 communication provides an easy to use interface, however it is only suitable for offline transfer of test sequences.

The use of HDLs and software design tools in hardware design simplifies the design procedure and debugging.

The proposed hardware design is explained in details using data flow charts and state machine diagrams in Chapter 4. In designing of the hardware, we used the top-down design methodology which simplifies the design task and allows the partitioning of the whole design into easily manageable subparts. We also discuss the design alternatives and the selected approaches together with a discussion of the selection procedure. The main advantage of the hardware design comes from the parallelized and pipelined implementations of the functions.

In Chapter 5, the proposed hardware design is tested using one synthetic and two real test sequences that are frequently used for performance evaluations of optical flow methods in the literature. The error between the proposed hardware implementation and floating point PC implementation is compared using angular error rate and end point error rate measures. We achieved a relative 1.319° average angular error rate with 0.509° standard deviation and 0.024 endpoint error at most. The analysis of the proposed hardware implementation is done in terms of resource usage, power consumption and computation time. The overall ratings are partitioned between the sub-design modules indicating the dominant modules effecting the whole performance criteria the most. At the end of the chapter, we make comments on the results and compare the computation time and power consumption results of our design with a software implementation on a PC of the day. The hardware can compute optical flow vector field on a 256x256 image pair in 3.89ms which is 146 times faster than software implementation while consuming only 844.38 mW of power which is 1/40 of the power consumed by the 1.66 GHz processor of the PC.

The comparison between the hardware implemented and a software implemented (Matlab) system using the same algorithm showed that the hardware implementation achieved a superior performance in terms of speed, power consumption and compactness in charge of a reasonable decrease in accuracy. In conclusion, the FPGA implementation of optical flow computation can provide hardware acceleration to vision applications on robotic platforms while delivering real-time speeds, low power consumption and reasonable accuracy at an affordable cost.

6.1 Future Work

The presented hardware implementation on DE2-70 board enables processing image frames up to a size of 256k pixels. The constraint on the image size is caused by the low capacity of SSRAM available on board. Processing images with larger sizes requires a higher capacity memory. One option is to use 64MB SDRAM in trade of a noticeable performance decrease.

The FPGA development board we utilized is a general purpose board designed for education and research purposes by TerASIC. So there are many capabilities and interfaces available on-board such as AC97 sound codec. Although it can be used for a wide application area, the unused devices take up space and causes the board to be larger. They also consume power even they are not in use. It is possible to overcome these disadvantages by designing a custom PCB with required components and devices only.

The proposed hardware is designed to take input image sequences from PC for testing purposes. An interface to take image sequences from a CMOS camera in high speed is being designed currently. However, it is not mentioned in the scope of this thesis and planned to be used for applications on our robotic platform. With some modifications, the same hardware can also compute realtime optical flow data on images taken from a stereo camera.

There are also some techniques available to compute a more accurate optical flow field. One of them is the multi-resolution approach that is frequently used with differential techniques. This technique can yield a more accurate optical flow estimation even in large displacement cases. However, our hardware development board has limited memory resources to store the required data. The use of fast and more capacity DDR SDRAMs may enable the implementation of multi-resolution method. On the other hand, the addition of multiresolution computations will introduce an additional delay to the optical flow computation and the maximum achievable frame rate will decrease eventually. If we assume that the speed of the motion field is constant, processing at lower frame rate will correspond to more displacements in consecutive frames. Although, multiresolution method can handle large displacements of pixels, the delay

of computations increases the displacements even more. Therefore, the advantages and drawbacks of implementing multiresolution method should be analyzed carefully.

Before optical flow computation, the input images are generally subject to some prefiltering process to increase the accuracy of the algorithm. Since filtering takes less time compared to optical flow computation, this hardware design excludes the implementation of a filtering algorithm. The filtering algorithm can also be implemented in hardware to get a complete system on a chip.

We used the spatiotemporal gradient estimates presented in the original paper of Horn & Schunck. Alternative methods that yield better estimates are presented in literature. However, those methods require more computational power than the original one. Alternative methods can also be implemented if a more accurate flow field is required in trade of performance.

To increase the accuracy of the fixed point implementation, the division operation can be implemented using a fixed point representation with larger word length than the current 32 bits representation. However, it should also be considered that, this will inevitably introduce an increase in the computation time.

Finally, the achieved results motivates us to implement more computer vision algorithms on hardware. There are still many computer vision tasks that can not be utilized in mobile robotic applications because of the mentioned performance concerns. Although the hardware design is more complicated and requires much more effort than the software design, the academic studies seem to utilize even more hardware in the future.

REFERENCES

[1] Altera. *DE2-70 User Manual*, v1.01 edition, 2007.

[2] Altera. *Cyclone II Handbook*, February 2008.

[3] Altera. *Quartus II Handbook*, July 2010.

[4] P. Anandan. A unified perspective on computational techniques for the measurement of visual motion. In *International Conference on Computer Vision (ICCV)*, pages 219–230, 1987.

[5] P. Arribas and F.-H. Macia. Fpga board for real time vision development systems. In *Proceedings of the Fourth IEEE International Caracas Conference on Devices, Circuits and Systems*, pages T021–1–T021–6, 2002.

[6] P. C. Arribas and F. J. Alonso. Fpga real time lane departure warning hardware system. In *Computer Aided Systems Theory (EUROCAST)*, pages 725–732, 2007.

[7] P. C. Arribas and F. M. H. Maciá. Fpga implementation of the horn&shunk optical flow algorithm for motion detection in real time images. In *Design of Circuits and Integrated Systems Conference*, pages 616–621, 1998.

[8] P. C. Arribas and F. M. H. Maciá. Fpga implementation of camus correlation optical flow algorithm for real time images. In *Vision Interface Proceedings*, pages 7–9, 2001.

[9] P. C. Arribas and F. M. H. Macia. Fpga implementation of santos-victor optical flow algorithm for real-time image processing: an useful attempt. In *VLSI Circuits and Systems*, volume 5117, pages 23–32, 2003.

[10] S. Baker, D. Scharstein, J. Lewis, S. Roth, M. Black, and R. Szeliski. A database and evaluation methodology for optical flow. In *IEEE 11th International Conference on Computer Vision (ICCV).*, pages 1–8, Oct. 2007.

[11] J. Barron, D. Fleet, and S. Beauchemin. Performance of optical flow techniques. *International Journal of Computer Vision (IJCV)*, 12(1):43–77, February 1994.

[12] M. Black and P. Anandan. A framework for the robust estimation of optical flow. In *Fourth International Conference on Computer Vision*, pages 231–236, May 1993.

[13] J. M. Bodily. An optical flow implementation comparison study. Master's thesis, Brigham Young University, April 2009.

[14] T. Browne, J. Condell, G. Prasad, and T. McGinnity. An investigation into optical flow computation on fpga hardware. In *International Machine Vision and Image Processing Conference, IMVIP '08*, pages 176–181, Sept. 2008.

[15] T. A. Camus. *Real-Time Optical Flow.* PhD thesis, Brown University, Providence, RI, USA, 1994.

[16] J. Chase, B. Nelson, J. Bodily, Z. Wei, and D.-J. Lee. Real-time optical flow calculations on fpga and gpu architectures: A comparison study. In *16th International Symposium on Field-Programmable Custom Computing Machines, FCCM '08*, pages 173–182, April 2008.

[17] C. Claus, A. Laika, L. Jia, and W. Stechele. High performance fpga based optical flow calculation using the census transformation. pages 1185 –1190, jun. 2009.

[18] J. Diaz, E. Ros, R. Agis, and J. L. Bernier. Superpipelined high-performance optical-flow computation architecture. *Computer Vision and Image Understanding*, 112(3):262 – 273, 2008.

[19] J. Diaz, E. Ros, S. Mota, and R. Agis. Real time optical flow processing system. In *International Conference on Field Programmable Logic and Applications (FPL)*, pages 617–626, 2004.

[20] D. J. Fleet and A. D. Jepson. Computation of component image velocity from local phase information. *International Journal of Computer Vision*, 5(1):77–104, 1990.

[21] H. Haussecker and P. Geissler. *Handbook of computer vision and applications*, volume 3. Academic Press, 1999.

[22] D. Heeger. Model for the extraction of image flow. *Journal of the Optical Society of America A*, 4:1455–1471, Aug. 1987.

[23] B. Horn and B. Schunck. Determining optical flow. 17(1-3):185–203, August 1981.

[24] B. Lucas and T. Kanade. An iterative image registration technique with an application to stereo vision. In *Proceedings of Imaging Understanding Workshop*, pages 121–130, 1981.

[25] W. MacLean. An evaluation of the suitability of fpgas for embedded vision systems. In *Workshops, 2005. CVPR Workshops. IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 131–131, June 2005.

[26] J. Martin, A. Zuloaga, C. Cuadrado, J. Lazaro, and U. Bidarte. Hardware implementation of optical flow constraint equation using fpgas. *Computer Vision and Image Understanding Journal*, 98(3):462–490, June 2005.

[27] Y. Mizukami and K. Tadamura. Optical flow computation on compute unified device architecture. pages 179–184, 2007.

[28] H.-H. Nagel. On a constraint equation for the estimation of displacement rates in image sequences. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 11(1):13–30, Jan 1989.

[29] H. Niitsuma and T. Maruyama. High speed computation of the optical flow. In *International Conference on Image Analysis and Processing (ICIAP)*, pages 287–295, 2005.

[30] M. Otte and H.-H. Nagel. Optical flow estimation: advances and comparisons. In *ECCV '94: Proceedings of the third European conference on Computer vision (vol. 1)*, pages 51–60, Secaucus, NJ, USA, 1994. Springer-Verlag New York, Inc.

[31] M. Proesmans, L. van Gool, E. Pauwels, and A. Oosterlinck. Determination of optical flow and its discontinuities using non-linear diffusion. In *ECCV '94: Proceedings of the third European conference on Computer Vision (Vol. II)*, pages 295–304, Secaucus, NJ, USA, 1994. Springer-Verlag New York, Inc.

[32] J. Santos-Victor and J. S. victor Giulio S. Uncalibrated obstacle detection using normal flow. Technical report, University of Genova, Italy, 1996.

[33] R. Strzodka and C. Garbe. Real-time motion estimation and visualization on graphics cards. In *VIS '04: Proceedings of the conference on Visualization '04*, pages 545–552, Washington, DC, USA, 2004. IEEE Computer Society.

[34] Z. Wei, M. Martineau, D.-J. Lee, and M. Martineau. A fast and accurate tensor-based optical flow algorithm implemented in fpga. In *Applications of Computer Vision, 2007. WACV '07. IEEE Workshop on*, pages 18–18, Feb. 2007.