AENSI Journals

Australian Journal of Basic and Applied Sciences

ISSN:1991-8178

Journal home page: www.ajbasweb.com

Design and FPGA Implementation of Systolic Array Architecture For Vector Computation

[1]Selvakumar, R., [2]Dr. Dharmishtan K. Varughese

[1]*Karpagam College of Engineering, Department of ECE, Research Scholar, Coimbatore, India*
[2]*Karpagam College of Engineering, Department of ECE, Professor Coimbatore, India*

ARTICLE INFO

Article history:
Received 25 October 2014
Received in revised form
26 November 2014
Accepted 29 December 2014
Available online 15 January 2015

Keywords:
Optical flow, Vectors, Systolic array, Latency and Image resolution

ABSTRACT

Optical flow is mainly used for collision and obstacle detection in unmanned air vehicles. Most of the optical flow algorithms are more suited for software implementation and do not provide the measurement density along with the accuracy and the computational speed necessary for real time implementations. Hence the design of specialized hardware for optical flow computation to provide higher computational speeds with reasonable accuracy is required. In this paper systolic array based architecture for optical flow vector computation is designed for 32x32 image resolution which can be extended to 256x256 resolution images. Optical flow vectors have higher dynamic range of fractional numbers which if implemented in floating point arithmetic consume a lot of hardware resources and power, hence floating to fixed point conversion has been performed. This Paper method uses fewer adders, multipliers and dividers than actually required along with flexibility of reuse of existing data for further iterations without actually recalculating. Insignificant bits at intermediate levels are truncated to reduce multiplier input bit widths such that accuracy of vectors will remain unchanged. Optical flow computation processor has been implemented on Virtex-5 FPGA. Total power consumed is 1.125W. Computational time for 32x32 images is 550.4 micro seconds for 100 iterations and 16 micro seconds for each iteration. When extended to process 256x256 images optical flow processor designed can process 1000 images/sec, whereas existing optical flow processors can process 60 images per sec.

To Cite This Article: Selvakumar, R., Dr. Dharmishtan K. Varughese, Design and FPGA Implementation of Systolic Array Architecture For Vector Computation. ***Aust. J. Basic & Appl. Sci.,*** *9(2): 133-142, 2015*

INTRODUCTION

Optical flow based navigation and control solutions for unmanned air vehicles (UAVs) are becoming appealing since scientific evidence is mounting on the fact that flying insects rely on some form of optical flow to perform quick and highly accurate navigation routines. This paper presents design and implementation of hardware architecture for optical flow computation processor targeted for unmanned vehicle navigation applications.

Despite ongoing efforts, significant research issues have to be addressed before wide scale application of optical flow based guidance and navigation control schemes. This is mostly due to the fact that most optical flow algorithms do not yet provide the measurement density along with the accuracy and the computational speed necessary for real time implementations. Hardware implementation of optical flow computation is necessary for real time applications like unmanned vehicle navigation.

Design and implementation of Systolic array architecture for optical flow on FPGA is focused on real time applications especially unmanned vehicle navigation. Optical flow estimation for real time applications like unmanned vehicle navigation requires higher computational speeds with reasonable accuracy. Current software implementations will provide good accuracy but requires higher computational times which make them unsuitable for real time application so it is required to design and implement specialized hardware for optical flow estimation which can be adopted for real time applications. This paper shows systolic array based architecture design to compute optical flow vectors in real time with high computational speeds and reasonable accuracy required for real time applications.

Corresponding Author: Selvakumar R, Karpagam College of Engineering, Department of ECE, Research Scholar, Coimbatore, India.

Related work:

Horn and Schunck(1980) proposed an algorithm based on gradient method for determining optical flow. According to them "Optical flow cannot be computed locally, since only one independent measurement is available from the image sequence at a point, while the flow velocity has two components. A second constraint is needed. A method for finding the optical flow pattern is presented which assumes that the apparent velocity of the brightness pattern varies smoothly almost everywhere in the image". With respect to the above statement they proposed gradient based optical flow estimation algorithm with brightness constancy and spatial coherence constraints. The proposed method is iterative method which gradually smoothens the measurements to obtain more accurate estimates of optical flow vectors.

Jose L. Martin *et al*.(2004) proposed an approach for hardware implementation of optical flow constraint equation proposed by Horn and Schunck. In this approach FPGA's along with FIFO memories were used to implement the constraint equation on hardware. The approach is iterative with pipelined hardware to take advantage of iterative computations. The implemented hardware can process 256x256 images with 60 images /sec. Due to the use of iterative approach achievable computational speeds is limited as compared to array implementation of the constraint equation.

Aitzol Zuloaga *et al*.(1998) proposed hardware architecture for implementing optical flow constraint equation. The proposed solution uses pipelined approach for iterative implementation and uses FIFO and delay sequencers for synchronizing the inputs as well as feedback vectors. Optimal partitioning of blocks to suite the implementation is the major advantage for the proposed architecture.

Marco Mammarella *et al*.(2008) made an effective comparison of optical flow algorithms for aircraft navigation applications and provided the efficiency, accuracy and feasibility of different optical flow algorithms.

Nan Lu *et al*.(2008) proposed an improved motion detection method for real time survivalence. Lucas-Kanade based optical flow estimation algorithm is used for real time video sequences. The author proposed a 3D sober operator for optical flow computation. Sober operator proposed are templates which can be used for calculating optical flow gradient terms with less complexity.

Hongche Liu *et al*.(1994) have analyzed different optical flow algorithms for speed vs accuracy trade-offs and also compared algorithms with higher flexibility in handling different motions.

ZhaoyiWei *et al*.(2008) proposed an FPGA based embedded motion estimation sensor. The proposed architecture is implemented with embedded development kit available in FPGA's.Micro blaze soft processor is used as processor in FPGA with interfaces and peripherals the drivers for which are developed in c language. A tensor based algorithm is used and the hardware blocks were modeled for major calculations and the embedded processor is used to control the data interface with the hardware.

Summary of related work:

- Hardware implementation of optical flow is required due to the fact that most optical flow algorithms do not yet provide the measurement density along with the accuracy and the computational speed necessary for real time implementations
- Gradient based methods are proved to be faster and less computation intensive
- Horn & Schunck algorithm which is a gradient based algorithm is chosen for implementation due to its higher computational speeds and it is less computation intensive
- Systolic array architecture identified in the literature requires improving the local communication employed to reduce the I/O bottleneck imposed by memory elements.
- Conventional pipelined implementation of Horn and Schunck algorithm is suitable for non-iterative computations which do not provide accuracy required for real time applications
- Horn and Schunck constraint equations require optimal partitioning to eliminate dependencies between different computations
- Conventional optical flow processors are implemented with floating point arithmetic which requires more hardware resources and higher power consumption

Methodology:

Horn and Schunck algorithm which is gradient based technique is chosen due to higher computational speeds and less computation intensive for hardware implementation. Horn and Schunck equations are restructured for reducing the dependencies in the calculations and the floating point to fixed point conversion is carried out to reduce the hardware utilization and power consumed. Optical flow vectors are calculated using Matlab in software to obtain the properties of optical flow vectors. Optical flow computation processor based on systolic array architecture is modeled with Verilog HDL and simulated with ModelSim for functional verification of the system. Xilinx ISE is used to synthesize and implement the design on the Virtex 5 FPGA.

Design and Implementation:

Need for Systolic Array:

- High-performance, special-purpose computer system is required to meet specific application
- I/O and computation imbalance is a notable problem
- The concept of Systolic architecture can map high-level computation into hardware structures
- Systolic system works like an automobile assembly line
- Systolic system is easy to implement because of its regularity and easy to reconfigure
- Systolic architecture can result in cost-effective, high-performance special-purpose systems for a wide range of problems

Design of Systolic Array Architecture:

Systolic array architecture has been designed in such a way to take advantage of local communication, regularity and modularity of the processing elements (PE's). This structure provides reduced access to memory.

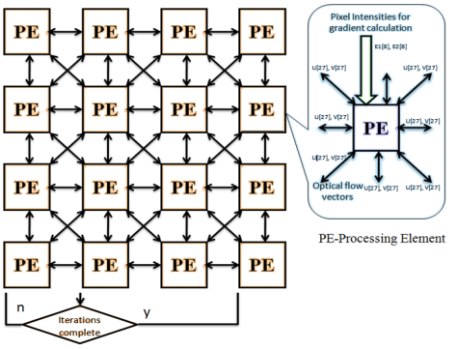

Fig. 1: Systolic array architecture for Optical flow.

Figure 1 shows two dimensional systolic arrays of 4x4 elements. Basic processing element which is called 'PE' is regular and modular. Since all the PE's do the same function except for I/O handling modifications for boundary PE's, it is possible to design one PE and replicate it to arrive at any structure like 32x32 or 256x256 without much effort. Being regular will provide more reliability and reduced effort of debugging, since debugging single PE is required instead of debugging the entire system or array.

In general each PE would take image intensities as input, calculates gradients, laplacians and provides optical flow vectors. Each PE communicates the calculated optical flow vectors with the neighboring PE's which will be utilized for calculating laplacians or average velocity vectors. Each PE computes flow vectors for a single pixel.

Processing element (PE):

Block diagram of the processing element is shown in Figure 2 which includes the details of the communication between basic building blocks namely adder/subtractor multiplier IP and divider IP. Top level inputs for processing element are clk, rst and pixel intensities of previous and current images. Over signal indicates the completion of the computation and indicates the availability of the optical flow vectors.

Controller is the key block of the processing element which is designed with the control flow. Controller block communicates with all the basic blocks in the processing element and controls the operation of all the blocks including the scheduling of inputs and collection of outputs. Different handshaking signals between the controller and the other blocks will provide the synchronization of the data among all the blocks.

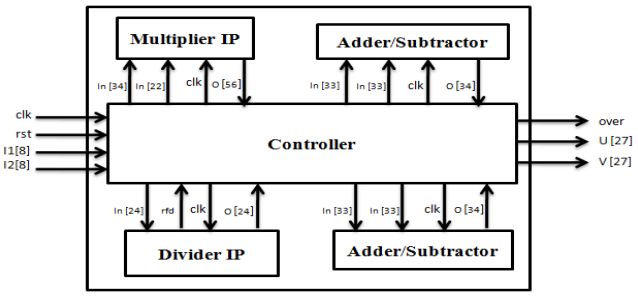

Fig. 2: Processing element block diagram.

Since design needs to build array structure for higher resolution images which requires more elements in the array, we are limited to use only one adder, multiplier and divider for each processing element (PE). Splitting into smaller terms provides flexibility in building the hardware especially when certain terms require to be computed once and used repeatedly. Figure 3 shows the method used for computing optical flow vectors for 32x32 images using 8x4 systolic array architecture.

32x32 images require the use of 8x4 systolic array architecture iteratively with the help of memories for 32 iterations to computer optical flow vectors. Initially 32x32 images are loaded into 32x32 memories from which 8x4 pixel elements will be stored in the second level memories which will have direct access to the systolic array architecture for optical flow computation. After the completion of vector calculation over signal is used as load signal to load the images. If the 32 iterations are complete next set of images will be loaded and if the 32 iterations is not complete second level memories will be loaded with the next 8x4 elements.

When calculating optical flow vectors with iterations more than one it can be observed that gradient calculations will remain same as iterate for the same image measurements or pixel intensities such that they can be calculated once in the first iteration and can be reused until all the iterations complete and new set of images are input to the system, and the laplacian and calculations will vary for each iterations based on the previous estimates of neighboring pixel or PE vectors, so they need to be calculated for each time iteration happens. If we can go further as we have separated the laplacian equation into smaller terms we can restrict to only fewer terms that need to be computed for every iteration by utilizing the existing values for other terms which were completely independent. We can observe from the Figure 3 terms on the right side are independent with respect to iterations and can be computed only once. On the other hand terms on the left are dependent on the outputs or vectors from the previous iterations and are required to be computed for every iteration. Let's say for 100 iterations we need to compute these terms 100 times and the terms on the right need to be computed only once out of 100. This provides a higher advantage in the overall performance of the system especially when the iterations are more

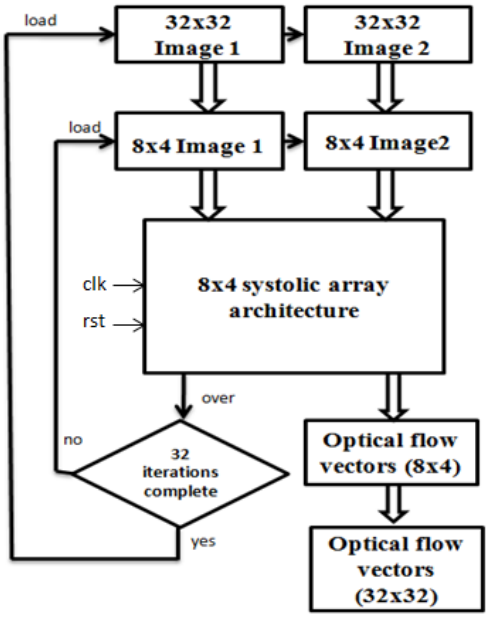

Fig. 3: 8x4 systolic array architecture design for 32x32images.

Processing Element Design:

Processing element is the most important element in the systolic array architecture which performs the major functionality to estimate the optical flow vectors. Figure 4 shows the general block diagram of optical processing element which models the optical flow equations for estimating optical flow vectors.

Processing element contains registers or memories internally for faster access to the data from the external memories which work serial providing one output for each clock. Internal registers also be useful in communicating intermediate optical flow vectors between different processing elements. This avoids the major I/O bottleneck with reduced dependency on the memories for performing operations

- Inputs: 8xE1[8] and 8xE2[8], 8xU[27] and 8xV[27], rst and clk
- Outputs: U[27], V[27] and load/over
- Clock: System clock with 9ns time period.

Inputs include pixel intensities including current pixel and neighbouring pixels for the current image and the previous image, optical flow vectors from the neighbouring pixels calculated in the previous iteration or for the previous image sequence.Horn and Schunck equations are required to design a PE or optical flow computation processor.

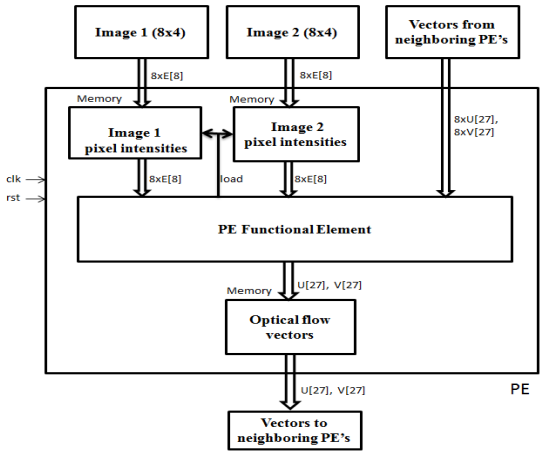

Fig. 4: Processing Element design.

Horn and Schunck(1980) equations required to design a PE or optical flow computation processor are

$$E_x = \frac{1}{4}\{E_{i,j+1,k} - E_{i,j,k} + E_{i+1,j+1,k} - E_{i+1,j,k} + E_{i,j+1,k+1} - E_{i,j,k+1} + E_{i+1,j,+1k+1} - E_{i+1,j,k+1}\}$$

$$E_y = \frac{1}{4}\{E_{i+1,j,k} - E_{i,j,k} + E_{i+1,j+1,k} - E_{i,j+1,k} + E_{i+1,j,k+1} - E_{i,j,k+1} + E_{i+1,j,+1k+1} - E_{i,j+1,k+1}\}$$

$$E_t = \frac{1}{4}\{E_{i,j,k+1} - E_{i,j,k} + E_{i+1,j,k+1} - E_{i+1,j,k} + E_{i,j+1,k+1} - E_{i,j+1,k} + E_{i+1,j,+1k+1} - E_{i+1,j+1,k}\}$$

$$\bar{u}_{i,j,k} = \frac{1}{6}\{u_{i-1,j,k} + u_{i,j+1,k}\} + u_{i+1,j,k} + u_{i,j-1,k}\} + \frac{1}{12}\{u_{i-1,j-1,k} + u_{i-1,j+1,k}\} + u_{i+1,j11,k} + u_{i,j-1,k}\}$$

$$\bar{v}_{i,j,k} = \frac{1}{6}\{v_{i-1,j,k} + v_{i,j+1,k}\} + v_{i+1,j,k} + v_{i,j-1,k}\} + \frac{1}{12}\{v_{i-1,j-1,k} + v_{i-1,j+1,k}\} + v_{i+1,j11,k} + v_{i,j-1,k}\}$$

$$u^{n+1} = u^{-n} - E_x\,(E_x\,u^{-n} + E_y\,v^{-n} + E_t)/(\propto^2 + E_x^2 + E_y^2)$$

$$v^{n+1} = v - E_{xy}\,(E_x\,u^{-n} + E_y\,v^{-n} + E_t)/(\propto^2 + E_x^2 + E_y^2)$$

Splitting the equations into smaller terms in order to efficiently handle and model the complex equations required to calculate optical flow.

$$Temp = 1/(\propto^2 + E_x^2 + E_y^2)$$

$$P_{term} = E_x * Temp$$

$$Q_{term} = E_y * Temp$$

$$U_{term} = (E_x\,U_b + E_y\,V_b + E_t\,) * P_{term}$$

$$V_{term} = (E_x\,U_b + E_y\,V_b + E_t\,) * Q_{term}$$

$$U = U_b - U_{term}$$

$$V = V_b - V_{term}$$

In general each PE would take image intensities as input, calculates gradients, laplacians and provides optical flow vectors. Each PE communicates the calculated optical flow vectors with the neighbouring PE's which will be utilized for calculating laplacians or average velocity vectors. Each PE computes flow vectors for a single pixel.

- Basic building blocks of a processing element (PE) are
- Adder/Subtractor
- Multiplier
- Divider

Since it is required to build array structure for higher resolution images which requires more processing elements in the array, usage of adder, multiplier and divider IP's for each processing element (PE) is limited. Splitting into smaller terms provides flexibility in building the hardware especially when certain terms require to be computed once and used repeatedly.

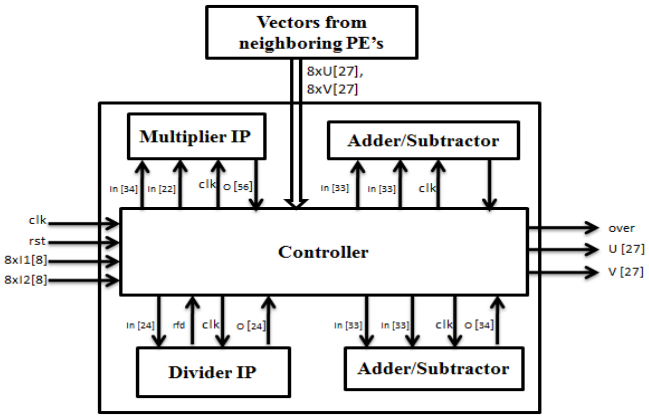

Fig. 5: Processing element functional block diagram.

Figure 5 shows the detailed block diagram of the processing element which includes the details of the communication between basic building blocks namely adder/subtractor multiplier IP and divider IP. Top level inputs for processing element are clk, rst and pixel intensities of previous and current images and the optical flow vectors from neighboring processing elements. Over signal indicates the completion of the computation and indicates the availability of the optical flow vectors.

Figure 6 shows the control chart for modeling the optical flow equations. For obtaining optical flow vectors with reasonable accuracy it is needed to calculate the optical flow vectors for more number of iteration. Number of iterations will determine the accuracy of the obtained optical flow vectors; this will provide trade-off for accuracy with the cost of speed or performance.

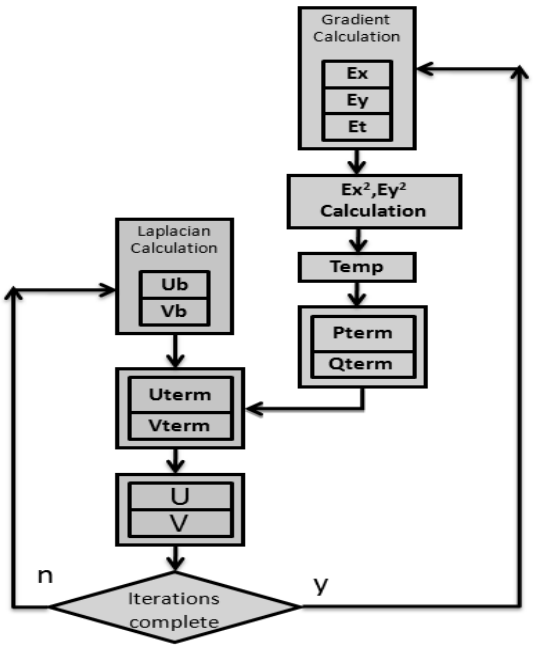

Fig. 6: Flow chart for optical flow equation modeling.

Australian Journal of Basic and Applied Sciences, 9(2) February 2015, Pages: 133-142

HDL Modeling:

Figure 7 shows the flow chart for calculating Ex which is the gradient along x direction. Ex involves computation of the average of the differences of the pixel intensities along x direction. Since the availability of limited resources two adders are used to perform serial addition with 8 pixel intensities with the help of FIFO. Serial addition involves addition of previous results with the new operands obtained from the FIFO. Once the FIFO is empty subtraction of the adder results will be performed to obtain Ex value.

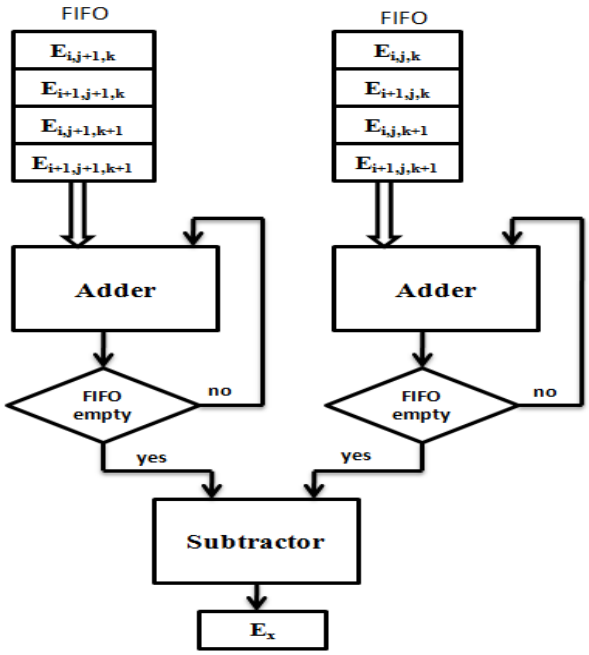

Fig. 7: Flow for calculation Ex.

Ey and Ez calculations can be performed using the similar flow chart with restructured input pixel intensities.

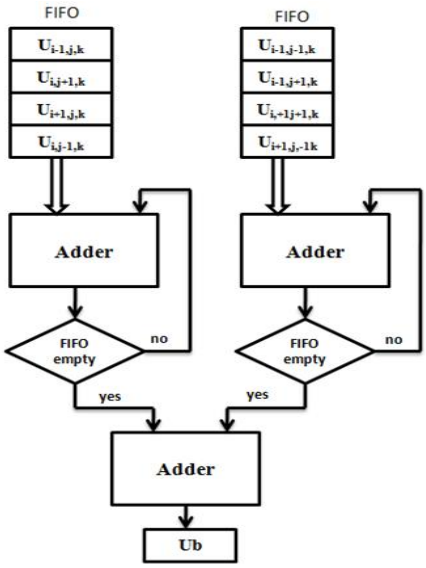

Fig. 8: Ub Flow chart.

Figure 8 shows the flow chart for computing Ub with hardware blocks. V b flow chart is same as Ub flow chart with vectors inputs in y direction. Flow charts for Ub and Vb are similar to the gradient calculation flow charts with different set of inputs. Ub and Vb calculation does not involves subtraction instead addition will be

involved. Ub and Vb values obtained are average velocity vectors which will be called as laplacian along x and y directions.

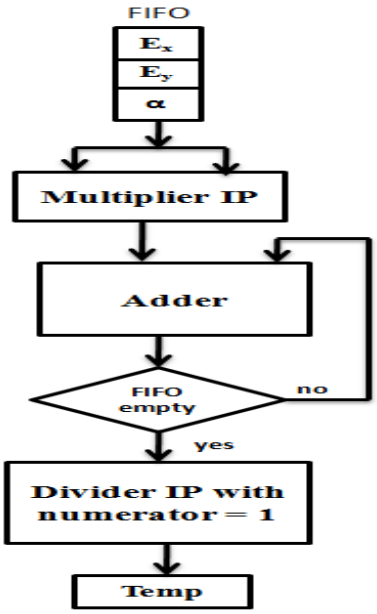

Fig. 9: Temp term flow chart.

Figure 9 shows the flow chart for calculating Temp term which involves all the building blocks namely adder, multiplier and divider. Temp calculation involves the computation of the sum of Ex2,Ey2 and α2 terms calculated with the help of a single multiple IP with serial accumulation with adder. Once the FIFO is empty adder output is fed to the divider IP as denominator with a constant numerator of 1. Divider IP has divide by zero indication flag which will be used to prevent the system to enter into unknown state. Divider IP will provides the result in 2's complement format with fractional number which will be left shifted to represent in fixed point representation.

RESULTS AND DISCUSSIONS

Flow Vector Computation Processor Results:

PE's were arranged in systolic array structure as shown in Figure 4 to form optical flow processor that can process 32x32 images. Apart from PE's few memories and synchronization circuitry were used to build the systolic array based optical flow processor. As per the available resources in Virtex-5 FPGA an 8x4 array is implemented and is utilized in iterative fashion to process 32x32 images. Memories were utilized to store intermediate results. Implementation results are shown in Table 1

Table 1: Hardware utilization results.

Total Recourses	Recourses used	Utilization in %	Time Period (ns)	9.04
Registers (69120)	52152	77.3%	Quiescent Power	0.96
LUTs (69120)	61154	88.4%	Dynamic Power(W)	0.16
DSP48E (64)	40	62.3%	Total Power	1.125

Validation of Results:

Optical flow vectors computed with the designed systolic array based optical flow processor needs to be verified for correctness and correlation with Matlab computed optical flow vectors. Optical flow vectors obtained from simulation of systolic array based optical flow processor will be in floating point converted and in two's complement format which needs to be converted back to decimal format representing fractional numbers in order make comparison with Matlab computed vectors. There is no automatic method to convert fractional binary numbers to decimal so we need to manually convert the simulation results to decimal format shifting the binary number 19 bits to right. In order to further simplify the manual effort simulation constructs in verilog HDL are used to covert 2's complement to signed magnitude format, shift the binary number by 19 bits right and separate fractional and integer parts. Simulation will write the converted and separated values in a text file

at the end of simulation such that manual conversion of fractional part was done manually in order to compare with Matlab results

Optical flow vectors U,V obtained from simulation and Matlab are tabulated in Table 2 where the deviated values are highlighted. Observe the values much smaller are the ones which got deviated as we have used only 27 bits to represent the optical flow vectors which can represent numbers as small as 7.629e-6 so numbers having significant information less than this will be truncated which is the reason for deviation[9]. For unmanned navigation applications accuracy obtained with 27 bits is sufficient because navigation control algorithm will treat vectors less than certain threshold as zeros considering very less or no-motion.

Table 2: Validation results

U_{matlab}	$U_{simulation}$	V_{matlab}	$V_{simulation}$
3.8189e-4	3.8125e-4	-6.7852e-5	-6.103e-5
7.5977e-6	0	-3.2465e-4	-3.24e-4
-0.0012	-0.0012	-8.6859e-5	-7.019e-5
-0.0012	-0.0012	5.6065e-5	5.3389e-5
-0.0022	-0.0022	4.9803e-5	4.5763e-5
-0.0032	-0.0032	5.3384e-6	0
-0.0026	-0.0026	-4.2687e-4	-4.26e-4
-0.0014	-0.0014	1.3671e-5	0.7629e-5
3.1900e-4	3.126e-4	-2.6739e-4	-2.67e-4
-2.7056e-4	-2.66e-4	-5.5041e-4	-5.50e-4
-0.0014	-0.0014	-2.7484e-4	-2.74e-4
-0.0019	-0.0019	1.5009e-4	1.50e-4
-0.0033	-0.0033	5.5991e-4	5.59e-4
-0.0049	-0.0049	6.7407e-4	6.74e-4
-0.0041	-0.0041	0.0011	0.0011
-0.0026	-0.0026	6.0247e-4	6.024e-4
3.9393e-4	3.93e-4	-2.0476e-4	-2.047e-4
-3.5307e-4	-3.53e-4	-1.9170e-4	-1.917e-4
-9.4438e-4	-9.44e-4	2.6419e-4	2.641e-4
-0.0018	-0.0018	2.3024e-4	2.302e-4
-0.0032	-0.0032	3.5176e-4	3.517e-4
-0.0051	-0.0051	7.1840e-4	7.184e-4
-0.0044	-0.0044	2.2090e-4	2.209e-4
-0.0032	-0.0032	4.2068e-5	3.8146e-5
2.7398e-5	2.2879e-5	-4.1679e-5	-3.8146e-5
1.7832e-4	1.7832e-4	6.3025e-5	6.1035e-5
-4.4709e-4	-4.4709e-4	1.2820e-4	1.282e-4
-0.0012	-0.0012	1.0986e-4	1.098e-4
-0.0018	-0.0018	3.0351e-4	3.035e-4
-0.0041	-0.0041	4.1754e-4	4.175e-4
-0.0040	-0.0040	1.5326e-4	1.53e-4
-0.0021	-0.0021	-1.7281e-4	-1.72e-4

Contribution:

New methodology for optical flow computation using Systolic array architecture is implemented on FPGA. Floating to fixed point conversion is performed to reduce hardware utilization and power. Flow charts required for developing Verilog HDL code are developed to model processing element with the basic building blocks namely adder/subtractor, multiplier and divider. Processing element dataflow between different iterations and for different clock cycles is discussed which provides the advantage of reduced latency or computational speed for optical flow computation by avoiding redundant operations. This paper provides an approach for specialized implementation for optical flow computation on FPGA to provide higher computational speeds with reasonable accuracy for real time applications.

Conclusion:

The designed architecture requires 57 clock cycles and for other iterations requires19 cycles. Hence for 100 iterations only 1767 clock cycles are required instead of 5700 clock cycles for computing optic flow. Calculations are made to convert floating point representation to fixed point representation which provided 600% reduction in FPGA resource consumption which helped in designing an array of 8x4 processing elements. Systolic array based optical flow processor works at a clock speed of 9.044ns. Total dynamic power consumed is 0.16W, leakage power is 0.96W with a total power consumption of 1.125W. Time required to compute Optical Flow vectors for 8x4 images with 100 iterations is 17µs. Time required to compute Optical Flow vectors for 32x32 images with 100 iterations is 243.2 µs. Optical flow processor is implemented on Viretx5 FPGA the results are correlated with Matlab obtained results and the results are accurate till the range of 10^{-6} which will be sufficient for unmanned air vehicle navigation applications.

REFERENCES

Aitzol Zuloaga, L. Jode Martin and Joseba Ezquerra, 1998. Hardware Architecture for Optical Flow Estimation in Real Time, Proceedings of the ICIP Conference on Image Processing

Hongche Liu, Tsai-Hang Hang, Martin Herman and Rama Chellappa, 1994. Accuracy vs Efficiency Trade-offs in Optical Flow Algorithms,Center for Automation Research/Department of Electrical Engineering, University of Maryland

Jose Martin, L., Aitzol Zuloaga, Carlos Cudrado, L. Zesus Azaro and Unai Bidarte, December 2004. Hardware Implementation of Optic Flow Constraint Equation using FPGA, Computer vision and image understanding, Science Direct.

Kung, H.T., 1982. Why systolic architectures?, January 1982. IEEE Computer Society, 15: 37.

Marco Mammarella and Giampiero Campa, 2008. A comparison of Optical Flow Algorithms for Real time Aircraft guidance and Navigation, AIAA Guidance Navigation and Control Conference and Exhibit.

Nan Lu, Jihong Wang, Q.H. Wu and Li Yang, 2008. An Improved Motion Detection Method for Real-Time Surveillance, IAENG International Journal of Computer Science.

Zhaoyi Wei, Dah-Jye Lee, E. Brent Nelson, K. James Archibald and B. Barrett Edwards, 2008. FPGA-Based Embedded Motion Estimation Sensor, Electrical and Computer Engineering Department, Brigham Young University.

s