IMPLEMENTATION OF FPGA-BASED OBJECT TRACKING ALGORITHM

A PROJECT REPORT

Submitted by

G. SHRIKANTH (21904106079)

KAUSHIK SUBRAMANIAN (21904106043)

in partial fulfillment for the award of the degree

of

BACHELOR OF ENGINEERING

In

ELECTRONICS AND COMMUNICATION ENGINEERING

SRI VENKATESWARA COLLEGE OF ENGINEERING, SRIPERUMBUDUR

ANNA UNIVERSITY: CHENNAI 600 025

APRIL 2008

ANNA UNIVERSITY: CHENNAI 600 025

BONAFIDE CERTIFICATE

Certified that this project report **"IMPLEMENTATION OF FPGA-BASED OBJECT TRACKING ALGORITHM"** is the bonafide work of "**KAUSHIK SUBRAMANIAN (21904106043) AND G. SHRIKANTH (21904106079)"** who carried out the project work under my supervision.

SIGNATURE	**SIGNATURE**
Prof. R. Narayanan	Mr. N. Venkateswaran
Head of the Department	**SUPERVISOR**
Department of Electronics and Communication Engineering	Assistant Professor
Sri Venkateswara College of Engineering, Pennalur,	Department of Electronics and Communication Engineering
Sriperumbudur - 602105	Sri Venkateswara College of Engineering, Pennalur,
	Sriperumbudur - 602105

EXTERNAL EXAMINAR	**INTERNAL EXAMINAR**

ACKNOWLEDGEMENT

We are personally indebted to a number of people who gave us their useful insights to aid in our overall progress for this project. A complete acknowledgement would therefore be encyclopedic. First of all, we would like to give our deepest gratitude to **our parents** for permitting us to take up this course.

Our sincere thanks and heartfelt sense of gratitude goes to our respected Principal, **Dr. R. Ramachandran** for all his efforts and administration in educating us in his premiere institution. We take this opportunity to also thank our Head of the Department, **Prof. R. Narayanan** for his encouragement throughout the project.

We would like to express our gratitude to our Internal Coordinator, **Prof. Ganesh Vaidyanathan** for his commendable support and encouragement for the completion of our project with perfection.

We also convey our sincere thanks to our internal guide, **Prof. N Venkateswaran** for his invaluable suggestions throughout the project and for his technical support rendered during the course of our project.

ABSTRACT

In this project we propose to use Image Processing algorithms for the purpose of Object Recognition and Tracking and implement the same using an FPGA.

In today's world most sensing applications require some form of digital signal processing and these are implemented primarily on serial processors. While the required output is achievable, it can be beneficial to take advantage of the parallelism, low cost, and low power consumption offered by FPGAs (Spartan 3E). The Field Programmable Gate Array (FPGA) contains logic components that can be programmed to perform complex mathematical functions making them highly suitable for the implementation of matrix algorithms.

The individual frames acquired from the target video are fed into the FPGA. These are then subject to segmentation, thresholding and filtering stages. Following this the object is tracked by comparing the background frame and the processed updated frame containing the new location of the target. The results of the FPGA implementation in tracking a moving object were found to be positive and suitable for object tracking.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1

INTRODUCTION TO OBJECT TRACKING AND SYSTEM DESIGN

1.1 OVERVIEW

1.1.1 Basic Object Tracking

Object tracking is the process of locating a moving object in time using a camera. The algorithm analyses the video frames and outputs the location of moving targets within the video frame.

A few examples of established motion models are:

- To track objects in a plane, the motion model is a 2D transformation of an image of the object (the initial frame)
- When the target is a 3D object, the motion model defines its aspect depending on its 3D position and orientation
- The image of deformable objects can be covered with a boundary box, the motion of the object is defined by the position of the nodes of the bounding box.

The role of the tracking algorithm is to analyze the video frames in order to estimate the motion parameters. These parameters characterize the location of the target. They help identify several other factors such as average speed, number of direction changes, total time in motion and also information about the shape and size of the target.

1.1.2 Methods of Implementation

The two major components of a visual tracking system –

- Target Representation and Localization
- Filtering and Data Association

Target Representation and Localization is mostly a bottom-up process. Typically the computational complexity for these algorithms is low. The following are the common Target Representation and Localization algorithms:

- Blob tracking: Segmentation of object interior (for example blob detection, block-based correlation).
- Mean-shift tracking: An iterative localization procedure based on the maximization of a similarity measure.
- Contour tracking: Detection of object boundary (e.g. Active Contours, Watershed Algorithm)
- Visual feature matching: Registration

Filtering and Data Association is mostly a top-down process, which involves incorporating prior information about the scene or object, dealing with object dynamics. The computational complexity for these algorithms is usually much higher. The following are some common Filtering and Data Association algorithms:

- Kalman filter: An optimal recursive Bayesian filter for linear functions and Gaussian noise.

- Particle filter: Useful for sampling the underlying state-space distribution of non-linear and non-Gaussian processes.

1.2 IMAGE PROCESSING SYSTEM

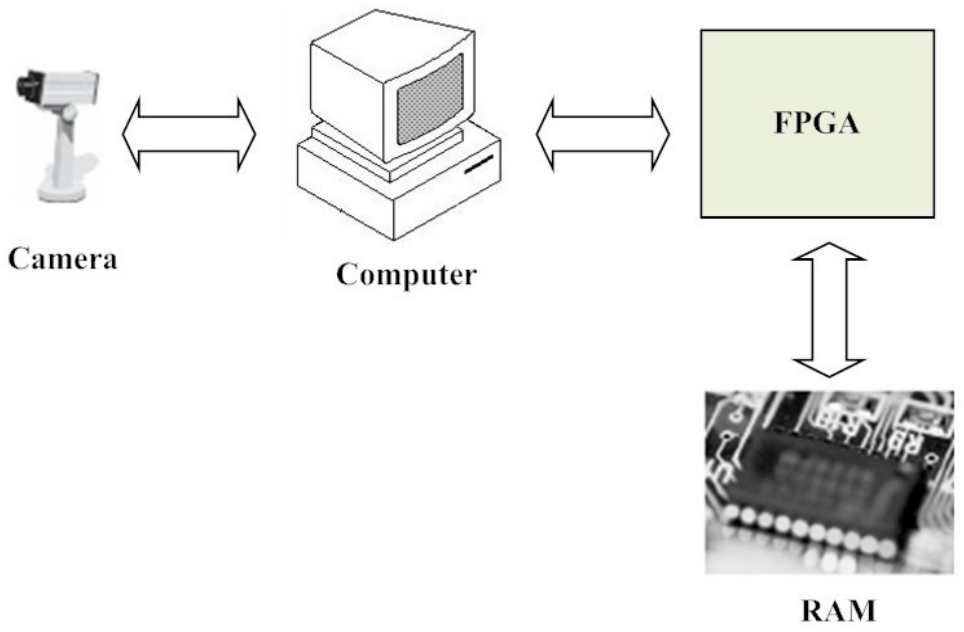


Figure 1.1: Layout of the Image Processing System

1.2.1 System Environment

Our aim is to work in an unstructured environment. An unstructured environment is one which has no artificial blue/green screen. This provides greater system flexibility and portability but can make reliable segmentation more difficult. As this environment requires the need to distinguish the objects of interest from any other objects that may be present within the frame. This limitation may be overcome by restricting the target objects to saturated and distinctive colors to enable them to be distinguished from the unstructured background. Augmenting the unstructured environment with structured color in this way is a compromise that enables a much simpler segmentation algorithm to be used. Another method to maintain the color distribution is to keep the background environment a constant. This way, only the target is in motion and the system is able to track its motion in a 2-D frame.

1.2.2 Image Acquisition

The image capture is performed using a color video camera which produces a stream of RGB pixels. The Casio camera is mounted on a tripod stand with a fixed Background that contains the object to be tracked. A brief 5 – 10 second video is recorded in .avi format. The temporal resolution requirements of the application have to be considered. We require a lower resolution as it will have a significantly higher acquisition rate for the observation of faster events. The size of the video frame is set to 640x480 pixels.

The video obtained is read in the computer using Matlab. The software processes the entire video and converts it into Image frames at the rate of 10 frames per second. Depending on the accuracy required and computational capability of the System, the frames can be interlaced.

1.2.2.1 Frame Generation

The video is fed in the Matlab program. The program reads the .avi file and converts it to frames. The frames are produced at the rate of 10 frames per second. Consider a 10 second video, a total of 100 frames will be produced in RGB format. These frames are then stored as individual bitmap files (total of 100 files). The bitmap files are arranged in the order of their occurrence in the video. The first frame is selected as the Base – Background Frame. The remaining bitmap files are used for the process of Object Recognition and Tracking.

1.2.2.2 Background and Object Identification

It is important that the object needs to be differentiated from the background. The color elements must be eliminated and the recognition is done in gray scale. With a still environment, the background frame is selected as the first frame. Considering the 10 second video, the 50th frame is randomly selected as the Object frame – these two frames form the basis for Object Recognition. It gives information about the shape and size of the object.

1.3 ALGORITHM DESIGN FOR OBJECT RECOGNITION

The following modules make up the Object Recognition stage: Grayscale Conversion, Delta Frame Generation, Thresholding, Noise Filtering and Image Enhancement. Figure 1.2 shows the Algorithm Flow.

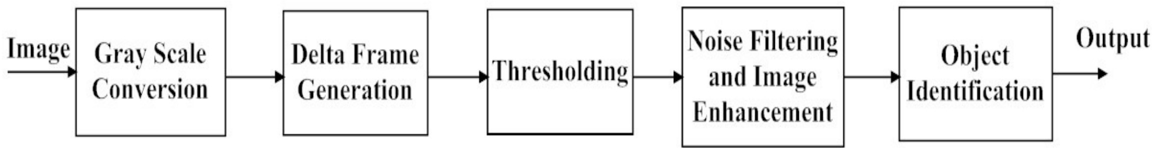


Figure 1.2: Object Recognition Algorithm Flow

1.3.1 Grayscale Conversion

The bitmap files have been generated and the Background and Object frame have been selected. These files are present in RGB format at a resolution of 640x480 pixels. These frames are then converted to grayscale within a range of 0-255. This reduces the coherent effect of the environment and allows us to easily separate the object from the background.

1.3.2 Delta Frame Generation

Once the Gray Scale Conversion has been completed, the respective frames are subtracted from one another. The resulting frame is called the Delta Frame. This method of image subtraction eliminates the background and brings the object into focus, giving us information about its shape and size. The Delta frame also reduces the number of pixels that the system will have to process.

1.3.3 Thresholding

In order to further enhance the resolution of the delta frame Gray Scale Thresholding is done. Example – Figure 1.3. The individual pixels in the grayscale image are marked as object pixels if their value is greater than some threshold value (initially set as 80) and as background pixels otherwise.

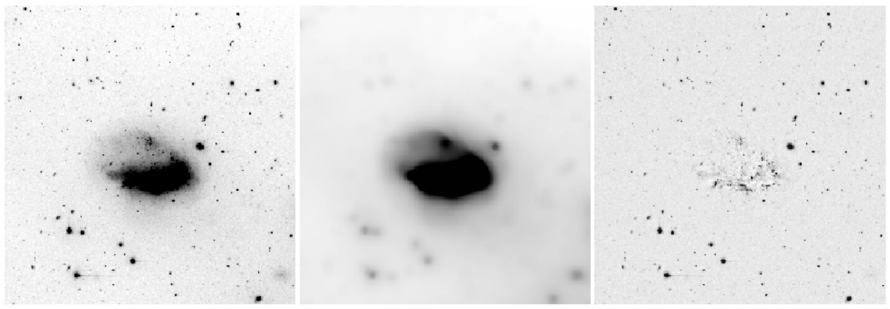


Figure 1.3: Gray Level Thresholding

In this case the object pixel is given a value of “1” while a background pixel is given a value of “0.” The thresholding can also be made adaptive when a different threshold is used for different regions in the image. The initial threshold value is set by considering the mean or median value. The approach is justified if the object pixels are brighter than the background. Else an iterative method has been implemented to obtain the value. The algorithm is as follows –

1. An initial random threshold (T) is chosen.
2. The image is segmented into object and background pixels using the above threshold. This creates two sets:
    1. G_1 = {f(m,n):f(m,n)>T} (object pixels)
    2. G_2 = {f(m,n):f(m,n) T} (background pixels)
3. The average of each set is computed.
    1. m_1 = average value of G_1
    2. m_2 = average value of G_2
4. A new threshold is created that is the average of m_1 and m_2
    1. $T' = (m_1 + m_2)/2$

1.3.4 Noise Filtering

The median filter is normally used to reduce noise in an image. The median filter is considered to do a better job than the mean filter of preserving useful detail in the image. The filter considers each pixel in the image in turn and looks at its nearby neighbors to decide whether or not it is representative of its surroundings. It then replaces the pixel value with the median of the neighboring pixel values. The median is calculated by first sorting all the pixel values from the surrounding neighborhood into numerical ascending order and then replacing the pixel being considered with the middle pixel value. (If the neighborhood under consideration contains an even number of pixels, the average of the two middle pixel values is used.) An example is shown below -

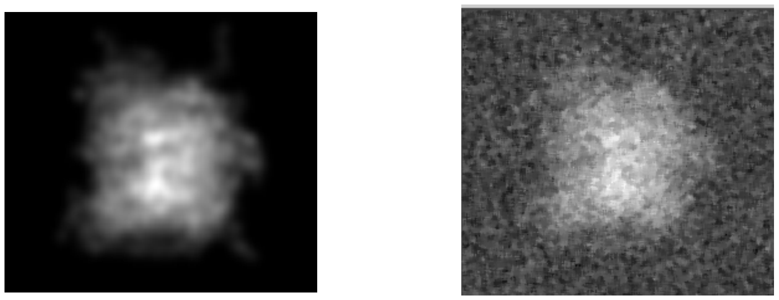


Figure 1.4: Example of Median Filter

The main advantages of the median filter:

- The median is a more robust average than the mean and so a single very unrepresentative pixel in a neighborhood will not affect the median value significantly.
- Since the median value must actually be the value of one of the pixels in the neighborhood, the median filter does not create new unrealistic pixel values. Thus the median filter is much better at preserving sharp edges than the mean filter.

In general, the median filter allows a great deal of high spatial frequency detail to pass while remaining very effective at removing noise on images where less than half of the pixels in a smoothing neighborhood have been effected.

1.3.5 Image Enhancement

The aim of image enhancement is to improve the interpretability or perception of information about the object. We have designed a spatial domain method which operates directly on the pixels. The filtered image is subjected to Edge Detection by the use of Sobel Operators. The outer boundary of the object is acquired. This gives us a noise free output image containing only on the boundary.

This image is then superimposed on the Object frame which was originally selected. This produces the required Enhanced Image. The object can be easily recognized using the acquired representation and its shape and size can be approximated.

1.4 OBJECT TRACKING

1.4.1 Optimal Frame Rate

The algorithm previously described helps identify the object and gives us information about its shape and size. Now in order to track it, we must select the frames acquired from the video. Considering a 10 second video, 100 frames are produced. The frames are then fed into the Matlab program at the rate of 1 frame per second. This is under the impression that the rate that we choose contains the complete motion of the object. The optimal frame rate considered is 1 frame per second. Further complexity is reduced if we alter the input frame rate to 4 frames per second.

1.4.2 Determining the Objects Position

In most applications, the center of gravity is used for tracking the target as it is a geometric property of any object. The center of gravity is the average location of the weight of an object. It can be used to describe the motion of the object through space in terms of the translation of the point of the object from one place to another.

In general, determining the center of gravity is a complicated procedure because the mass may not be uniformly distributed throughout the object. In order to simplify the problem we assume the object is composed of uniform material. We perform the preprocessing algorithms on the image to acquire a noise free enhanced image. An operator then scans the entire length of the image frame for the first white pixel. This is a clear indication of the 2D position of the object within that time frame. This is iterative process and it repeated over all the frames.

1.4.3 Comparative Tracking

The background frame is kept as the standard base. The following frames contain the updated location of the target. The optimal frame rate is chosen as previously explained. Each frame is fed into the program which subjects the frame to process of object recognition to achieve a noise free enhanced image containing only the object.

At the rate of 1 frame per second, the enhanced image is fed to the tracking program. This analyzes each frame and computes the first white pixel that represents the object. This is under the assumption that object is made up of uniform material. This point is then plotted on a new image as the first position of the object. Subsequent frames are collected, subtracted from the background, filtered, enhanced and then the points are computed. The updated locations of the pixel points are collected to provide the approximate path taken by the object. The iterative program acquires the frames and plots the individual points – Object Path.

1.5 MATLAB SIMULATION

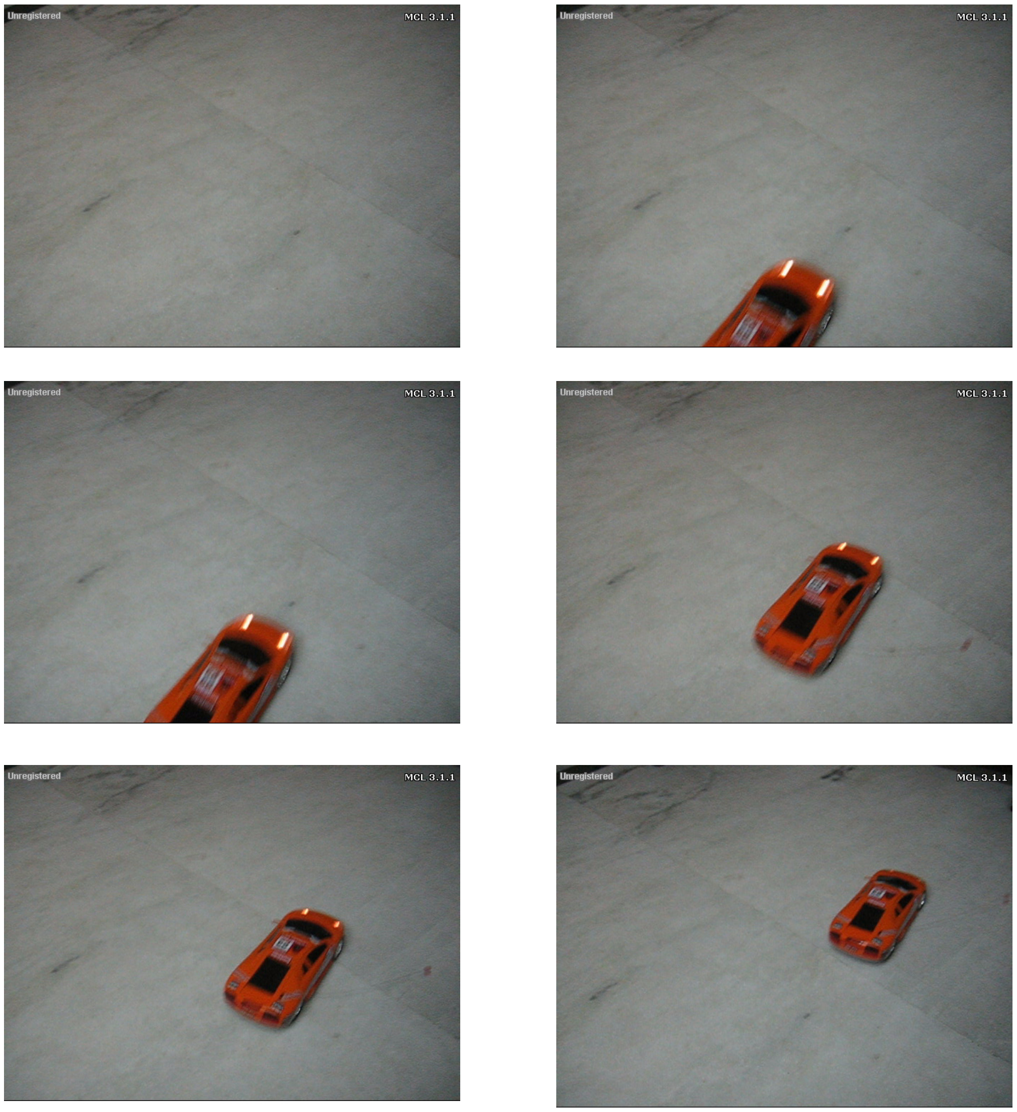
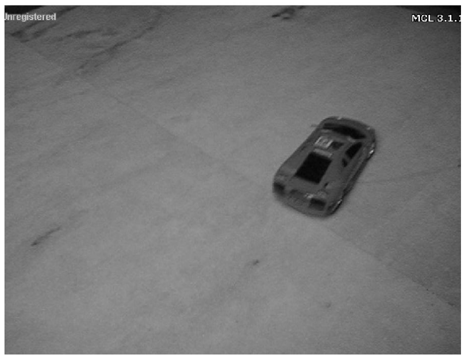
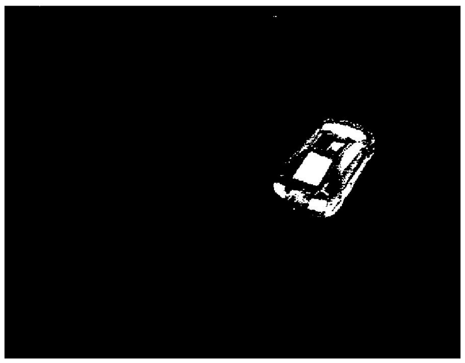


Figure1.5: Frame Generation using Matlab

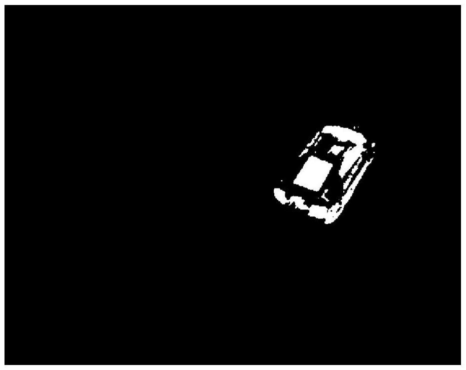
a) Gray Level Conversion



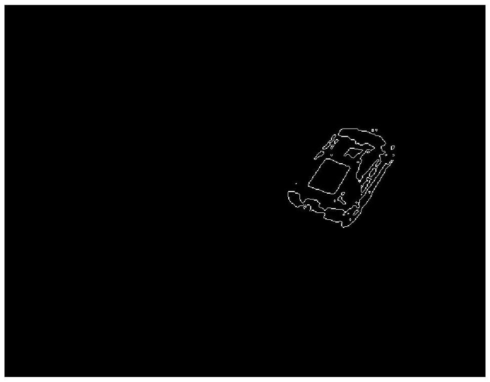
b) Threshold



c) Median Filter (Noise Removal)



d) Edge Detection



e) Enhanced Image

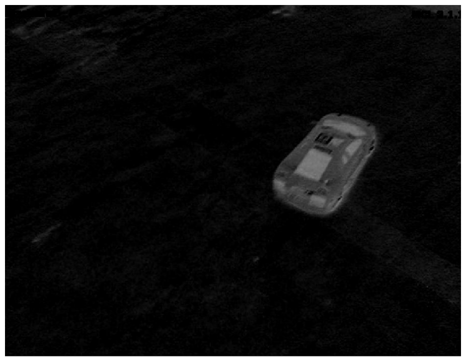


Figure 1.6: Step-wise generation of Enhanced Image

Path of the Object Tracked

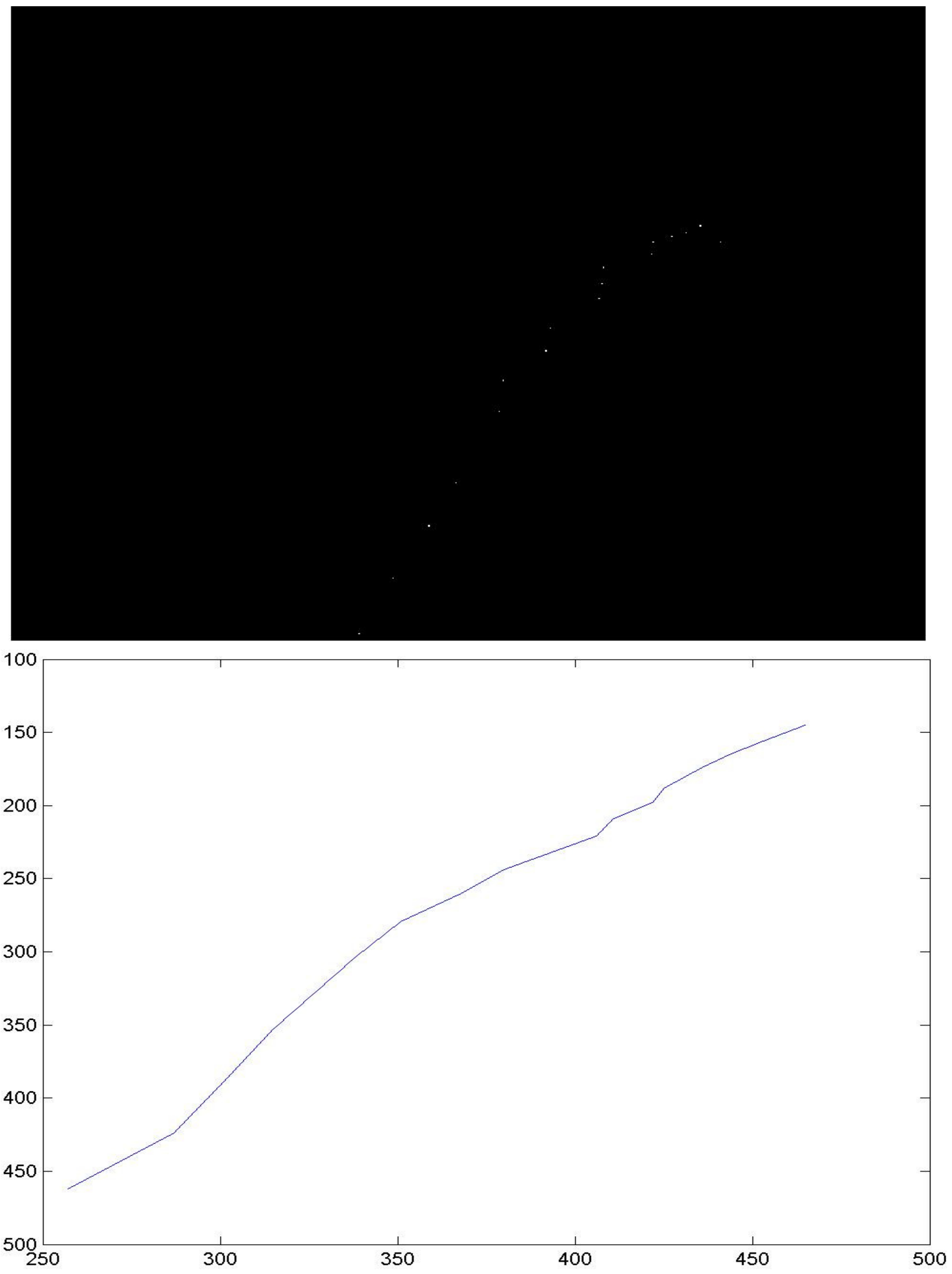


Figure 1.7: Object Path obtained

CHAPTER 2
FPGA IMPLEMENTATION OF OBJECT TRACKING ALGORITHM

2.1 THE ADVANTAGE OF USING FPGAs

Image processing is difficult to achieve on a serial processor. This is due to the large data set required to represent the image and the complex operations that need to be performed on the image. Consider video rates of 25 frames per second, a single operation performed on every pixel of a 768 by 576 color image (Standard PAL frame) equates to 33 million operations per second. Many image processing applications require that several operations be performed on each pixel in the image resulting in an even large number of operations per second. Thus the perfect alternative is to make use of an FPGA. Continual growth in the size and functionality of FPGAs over recent years has resulted in an increasing interest in their use for image processing applications. In a recent benchmarking test conducted by Berkeley Design Technology, Inc. (BDTi), the following results were acquired –

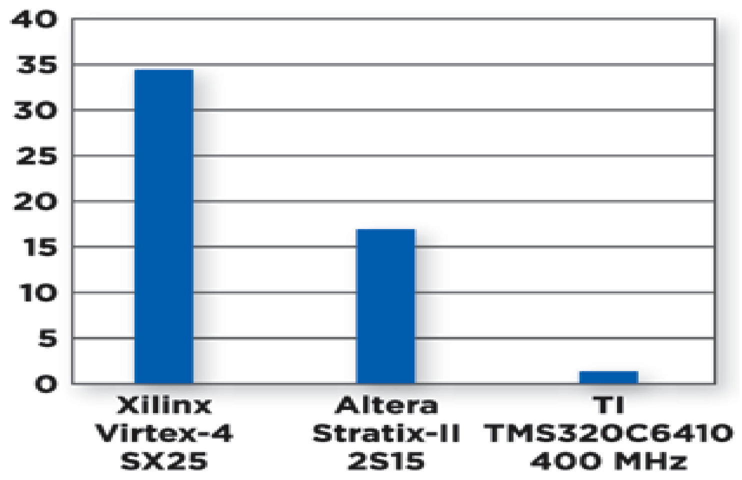


Figure 2.1: Benchmarking Test conducted by BDTi

The main advantage of using FPGAs for the implementation of image processing applications is because their structure is able to exploit spatial and temporal parallelism. FPGA implementations have the potential to be parallel using a mixture of these two forms. For example, the FPGA could be configured to partition the image and distribute the resulting sections to multiple pipelines all of which could process data concurrently. Such parallelization is subject to the processing mode and hardware constraints of the system.

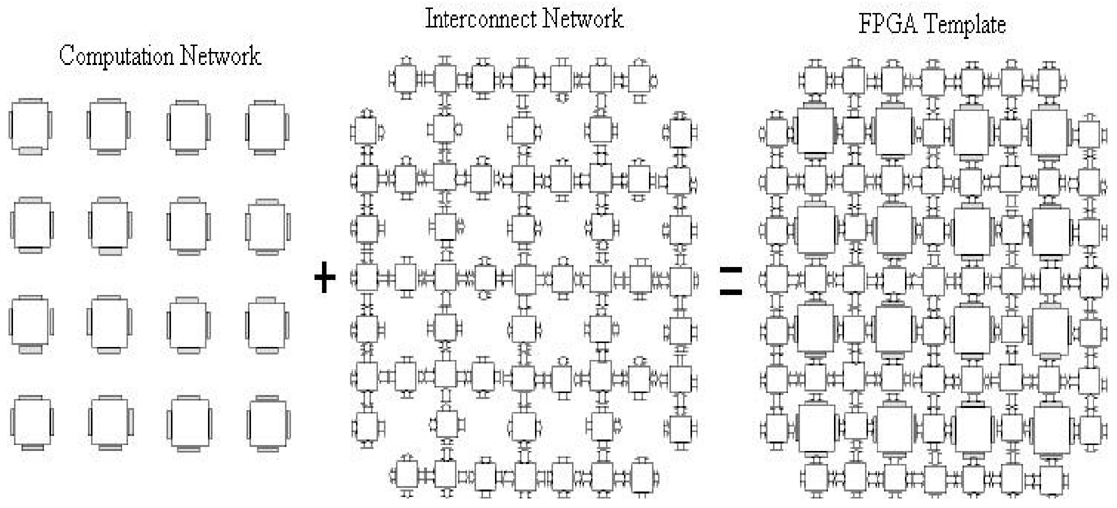


Figure 2.2: Programmable Logic Blocks of an FPGA

In Figure 2.2, an FPGA consists of a matrix of logic blocks that are connected by an interconnect network. Both the logic blocks and the interconnect network are reprogrammable allowing application specific hardware to be constructed, while at the same time maintaining the ability to change the functionality of the system with ease. As such, an FPGA offers a compromise between the flexibility of general purpose processors and the hardware-based speed of ASICs.

2.1.1 Hardware Constraints

There are three modes of processing: stream, offline and hybrid processing. In stream processing, data is received from the input device in a raster nature at video rates. Memory bandwidth constraints dictate that as much processing as possible can be performed as the data arrives. In offline processing there is no timing constraint. This allows random access to memory containing the image data. The speed of execution in most cases is limited by the memory access speed. The hybrid case is a mixture of stream and offline processing. In this case, the timing constraint is relaxed so the image is captured at a slower rate. While the image is streamed into a frame buffer it can be processed to extract the region of interest. This region can be processed by an offline stage which would allow random access to the region's elements.

2.1.1.1 Timing Constraints

If there is no requirement on processing time then the constraint on timing is relaxed and the system can revert to offline processing. This is often the result of a direct mapping from a software algorithm. The constraint on bandwidth is also eliminated because random access to memory is possible and desired values in memory can be obtained over a number of clock cycles with buffering between cycles. Offline processing in hardware therefore closely resembles the software programming paradigm; the designer need not worry about constraints to any great extent. This is the approach taken by languages that map software algorithms to hardware. The goal is to produce hardware that processes the input data as fast as possible given various automatic and manual optimization techniques.

2.1.1.2 Bandwidth Constraints

Frame buffering requires large amounts of memory. The size of the frame buffer depends on the transform itself. In the worst case (rotation by 90º, for example) the whole image must be buffered. A single 24-bit (8-bits per color channel) color image with 768 by 576 pixels requires 1.2 MB of memory. FPGAs have very limited amounts of on-chip RAM. The logic blocks themselves can be configured to act like RAM, but this is usually an inefficient use of the logic blocks. Typically some sort of off-chip memory is used but this only allows a single access to the frame buffer per clock cycle, which can be a problem for the many operations that require simultaneous access to more than one pixel from the input image. For example, bilinear interpolation requires simultaneous access to four pixels from the input image. This will be on a per clock cycle basis if real-time processing constraints are imposed.

2.2 SPARTAN 3E (XC3S500E) FPGA

2.2.1 Overview of Features and Layout

The Spartan-3E FPGA is embedded with the 90nm technology at the heart of its architecture. This reduces the die size and cost, increases manufacturing efficiency, and addresses a wider range of applications. You can integrate embedded processing, digital signal processing (DSP), and connectivity capabilities into Spartan-3E devices at no extra cost. These are supported with customized tools (ISE and EDK), JTAG probes, IP cores, design services, and training. The Spartan-3E diagram shown in Figure 2.3 allows users to easily migrate to different densities across multiple packages and supports 18 different single-ended and differential I/O standards.

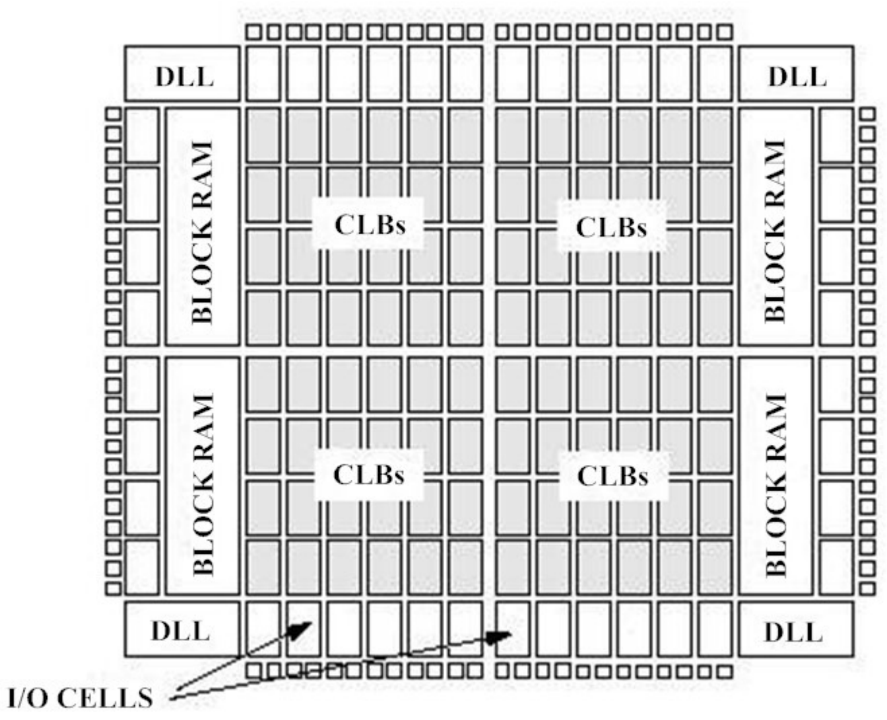


Figure 2.3: Spartan 3E Layout

The main advantages are High Speed Connectivity, High Performance DSP Solutions and Lowest Cost Embedded Processing Solutions.

1. High Speed Connectivity

System connectivity consists of physical parallel I/O interfaces and the protocols required for higher bandwidth. The Spartan-3E device I/O pins support full functionality for fast, flexible electrical interfaces. The PCI-Express slots are 100 MHz compatible. Also there are 18 I/O standards, DDR I/O registers, DCMs.

2. High Performance DSP Solutions

Spartan-3E FPGAs help you efficiently build DSP solutions that handle. Up to 9.1 billion multiply and accumulates (MACs) per second. There are up to 36, 18x18 embedded multipliers for implementing compact DSP structures such as MAC engines, and adaptive and fully parallel FIR filters. The Block RAM can be used for storing partial products and coefficients.

3. Lowest Cost Embedded Processing Solutions

The effective fractional cost of incorporating the MicroBlaze™ (32-bit soft processor) into a Spartan-3E FPGA is very less. The Xilinx MicroBlaze with Spartan-3E FPGA (Figure 2.4) can be used to integrate the entire processing engine, all control functions, and additional supporting logic into a single cost-effective platform. The Embedded Development Kit (EDK) offers a common development environment for Spartan Series FPGAs with MicroBlaze.

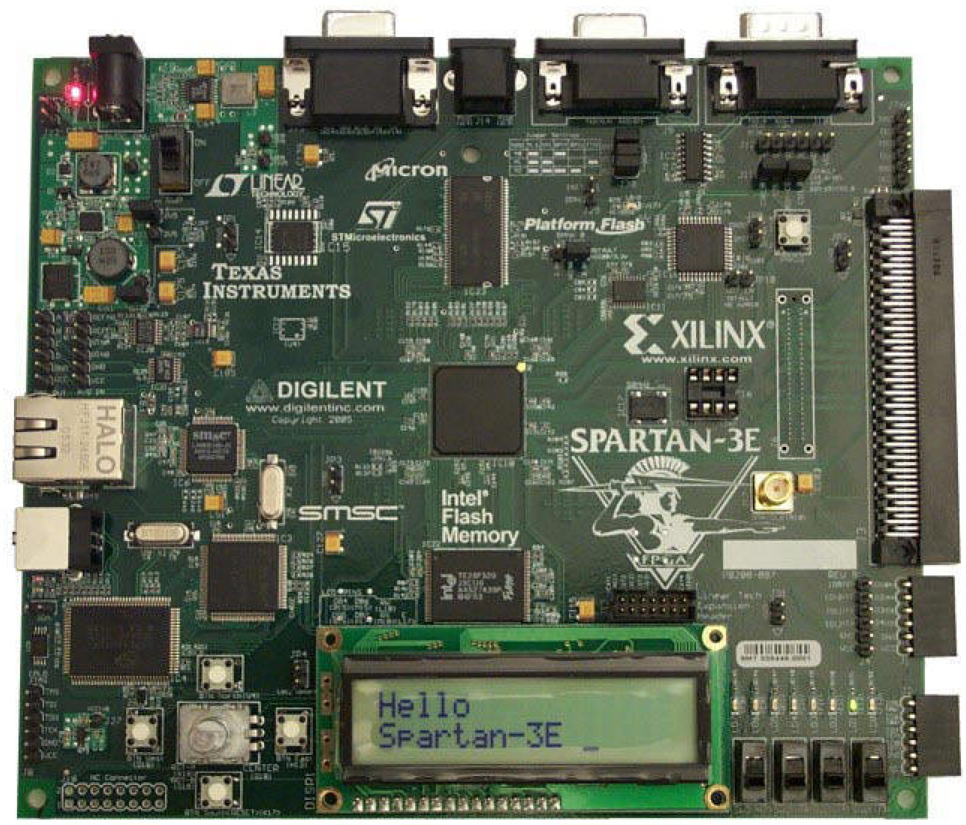


Figure 2.4: Spartan 3E Starter Kit

2.3 DEVELOPMENT TOOLS

2.3.1 Xilinx Embedded Development Kit 8.1i

The FPGA/FPGA chip is supported with a complete set of software and hardware development tools - Xilinx Embedded Development Kit (EDK) and Xilinx Platform Studio (XPS) tools development software. This tool is used to create a simple processor system. The microprocessors available for use in Xilinx Field Programmable Gate Arrays (FPGAs) with Xilinx EDK software tools can be broken down into two broad categories. There are soft-core microprocessors (MicroBlaze) and the hard-core embedded microprocessor (PowerPC).

EDK uses Intellectual-Property Interface (IPIF) library to implement common functionality among various processor peripherals. It is verified, optimized and highly parameterizable. It also gives you a set of simplified bus protocol called IP Interconnect (IPIC). Using the IPIF module with parameterization that suits your needs will greatly reduce your design and test effort because you don't have to re-invent the wheel. This is done in EDK with a wizard that walks you through the entire process.

2.3.2 MicroBlaze – Virtual Microprocessor

The MicroBlaze is a virtual microprocessor that is built by combining blocks of code called cores. MicroBlaze is an embedded soft core that includes the following features:

- Thirty-two 32-bit general purpose registers.
- 32 bit instruction word with three operands and two addressing modes.
- Separate 32-bit instruction and data buses that conform to IBM's OPB (On-chip Peripheral Bus) specification.
- 32-bit address bus
- Single issue pipeline

The MicroBlaze is a full 32-bit RISC CPU that is embedded in the Xilinx FPGA SPARTAN 3E and Virtex-4 FPGA families. It can be run at speeds up to 100MHz and is the best choice for CPU-intensive tasks in Xilinx FPGA based systems.

2.4 ALGORITHM MAPPING

The FPGA implementation is divided into blocks, each block implementing a separate portion of the algorithm. This approach allowed for concurrent development and for testing of individual blocks. The inbuilt finite state machine (FSM) controls each block. In addition, a high-level FSM controls the interaction of the blocks. Each computational block is implemented in C and checked for proper functionality with simulators (ISE Simulator)

The Algorithm primarily consists on mapping low-level operations like local filters. Conceptually, each pixel in the output image is produced by sliding an N×N window over the input image and computing an operation according to the input pixels under the window and the chosen window operator. The result is a pixel value that is assigned to the centre of the window in the output image as shown below in Figure 2.5.

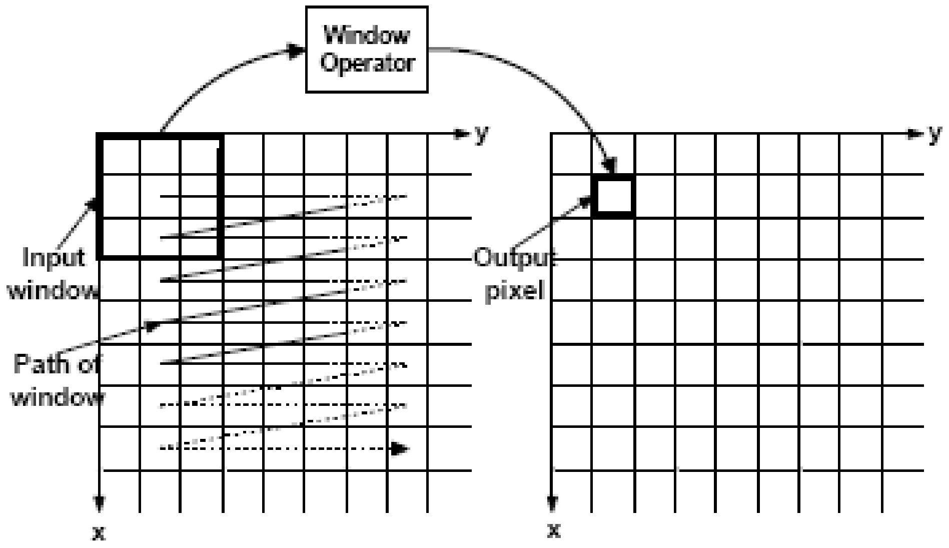


Figure 2.5: Mapping the Window Operation

For processing purposes, the straightforward approach is to store the entire input image into a frame buffer, accessing the neighborhood pixels and applying the function as needed to produce the output image. If processing of the video stream is required N×N pixel values are needed to perform the calculations each time the window is moved and each pixel in the image is read up to N×N times. Memory bandwidth constraints make obtaining all these pixels each clock cycle impossible. Input data from the previous N-1 rows can be cached using a shift register (or circular memory buffer) for when the window is scanned along subsequent lines.

Instead of sliding the window across the image, the above implementation now feeds the image through the window. Introducing the row buffer data structures adds additional complications. With the use of both caching and pipelining there needs to be a mechanism for adding to the row buffer and for flushing the pipeline. This is required when operating on video data, due to the horizontal blanking between lines and the vertical blanking between frames. If either the buffer or the pipeline operated during the blanking periods the results for following pixels would be incorrect due to invalid data being written to them. This requires us to stop entering data into the row buffers and to stall the pipeline while a blanking period occurs. A better option is to replicate the edge pixels of the closest border. Such image padding can be considered as a special case of pipeline priming. When a new frame is received the first line is pre-loaded into the row buffer the required number of times for the given window size. Before processing a new row the first pixels are also pre-loaded the required number of times, as is the last pixel of the line and the last line. Figure 2.6 shows the implementation of the Row Buffers for Window Operations.

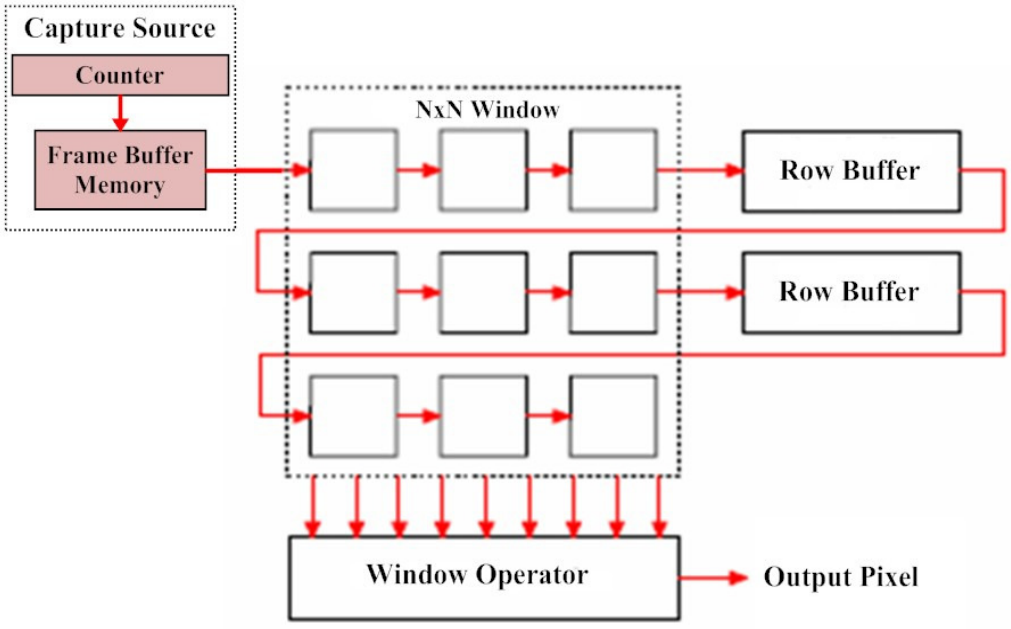


Figure 2.6: Window Operation using Buffers

2.4.1 Memory Interfacing and C Compiler

Because the Spartan 3E FPGA that is used in the design does not have enough internal RAM for image storage, the processing blocks were interfaced with five on-board 256K×36-bit pipelined DDRAM devices. To reduce the hardware computation time, each sub-block can read and write within the same clock cycle; each sub-block was connected to two memory chips while active. Typically, a computational block reads its inputs from one memory and writes its outputs to another. It is also necessary to control/arbitrate the FPGA internal block RAM, which is used for storage of computed thresholds and other parameters. The memory interface provides the computational blocks with a common interface and hides some of the complex details.

2.4.1.1 C Compiler

Xilinx MicroBlaze Processor Supports Linux and C-to-FPGA Acceleration Embedded systems can be developed to create hardware-accelerated, single-chip applications that take advantage of the MicroBlaze processor features and C-to-hardware acceleration for complex, performance-critical applications. The addition of memory management to the MicroBlaze processor provides embedded systems designers with a powerful new alternative for hardware-accelerated embedded systems. By offloading critical C-language processes to dedicated hardware coprocessors, the system as a whole can operate at a slower clock speed, consume less power and yet provide vastly more processing performance than would be possible using a discrete processor.

Using the automated C-to-hardware compiler tools and interactive optimizers, performance gains well in excess of 100X over software-only approaches, in applications that include image processing, DSP and secure communications have been reported.

The MicroBlaze configurable soft processor includes configurable coprocessor capabilities through its high-performance Fast Simplex Link (FSL) accelerator interface. The compiler automatically parallelizes and pipelines C-language algorithm and generates FSL interfaces, with little or no need for hardware design experience or hardware description language (HDL) coding. The automatic C-to-HDL capabilities of Microblaze dramatically accelerate system design.

2.4.1.2 Program Files

Input Files

1. MHS File

The Microprocessor Hardware Specification (MHS) file defines the hardware component. The MHS file serves as an input to the Platform Generator (Platgen) tool. An MHS file defines the configuration of the embedded processor system, and includes the following:

- Bus architecture
- Peripherals
- Processor
- System Connectivity

2. MSS File

The Microprocessor Software Specification (MSS) is used as an input file to the Library Generator (Libgen). The MSS file contains directives for customizing OSs, libraries, and drivers.

3. UCF File

The User Constraints File (UCF) specifies timing and placement constraints for the FPGA Design.

Output Files

1. Block Memory Map

A BMM file is a text file that has syntactic descriptions of how individual Block RAMs constitute a contiguous logical data space. When updating the FPGA bitstream with memory initialization data, the Data2Mem utility uses the BMM file to direct the translation of data into the proper initialization form. This file is generated by the Platform Generator (Platgen) and updated with physical location information by the Bitstream Generator tool.

2. ELF File

The Executable and Linkable Format (ELF) is a common standard in computing. An executable or executable file, in computer science, is a file whose contents are meant to be interpreted as a program by a computer. Most often, they contain the binary representation of machine instructions of a specific processor, but can also contain an intermediate form that requires the services of an interpreter to be run.

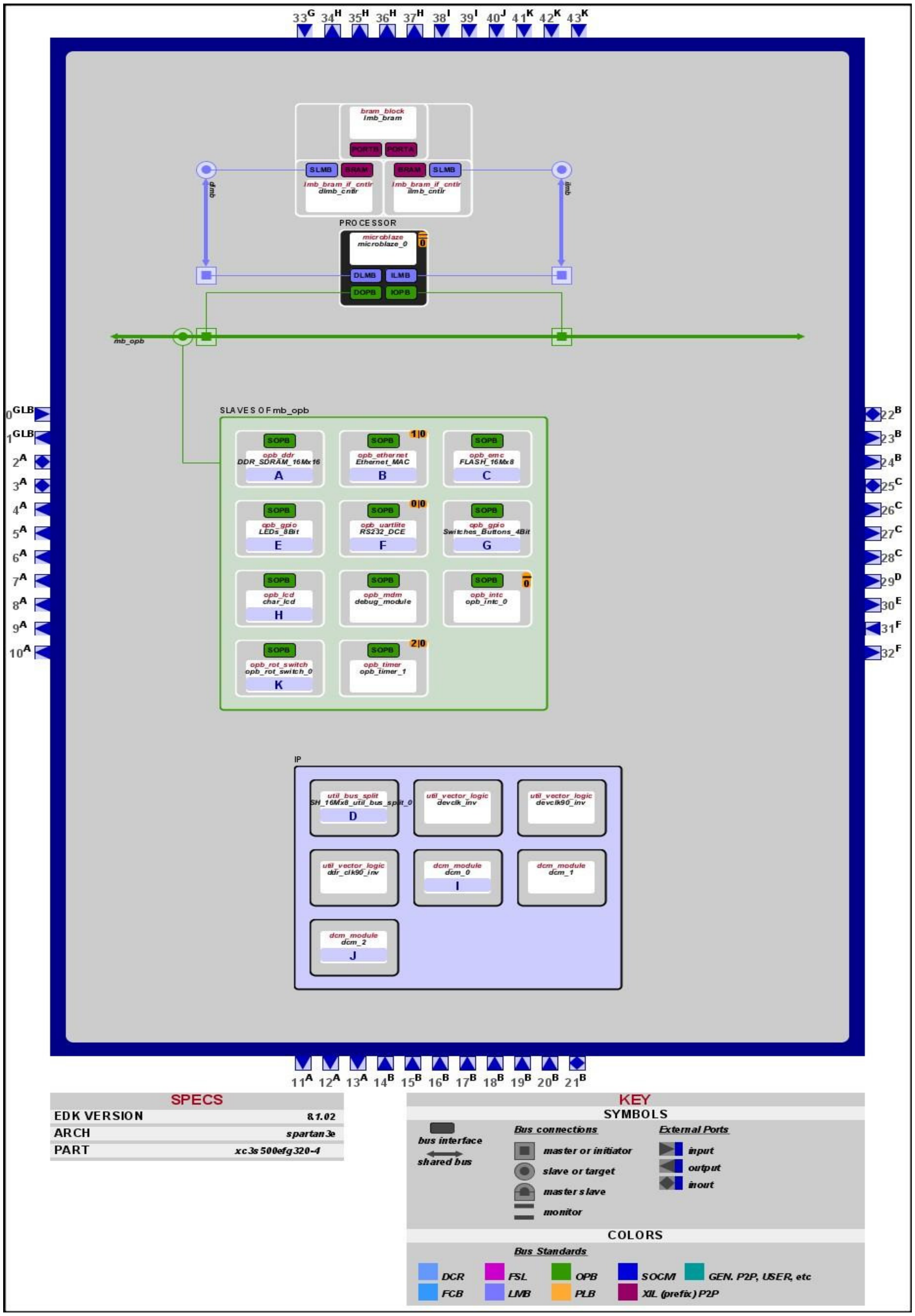


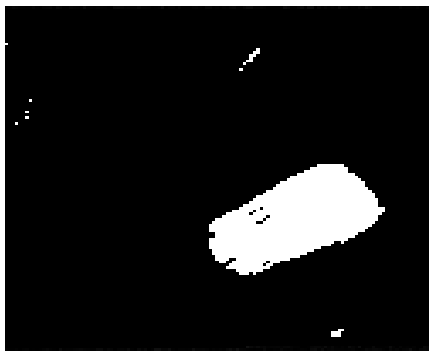
Figure 2.7: Synthesized Block Diagram

2.5 SIMULATION

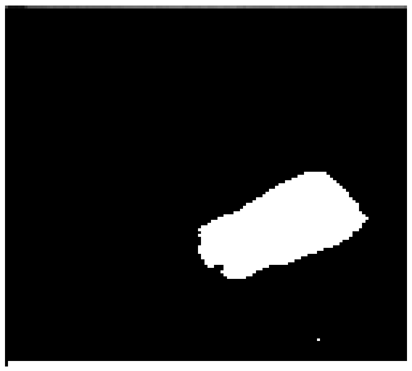
2.5.1 Hyper Terminal

The pixels generated through the FPGA are viewed using Matlab. The pixel values are displayed using a Hyper Terminal Connection. The interface can be established by appropriately setting the baud rate and COM port. The pixels obtained are copied to a text file which is read by Matlab. The program converts the text file to the required image. Figure 2.8 shows Images obtained from the pixel values computed using the FPGA.

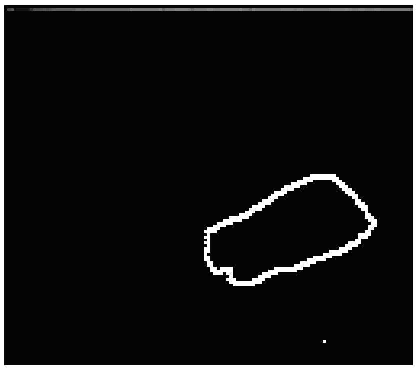
a) Threshold



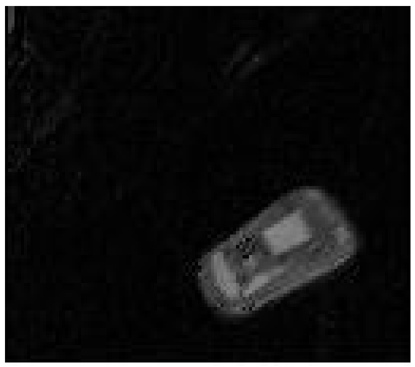
b) Noise Filter



c) Edge Detection



d) Enhanced Image



2.8 Text File converted to Image in Matlab

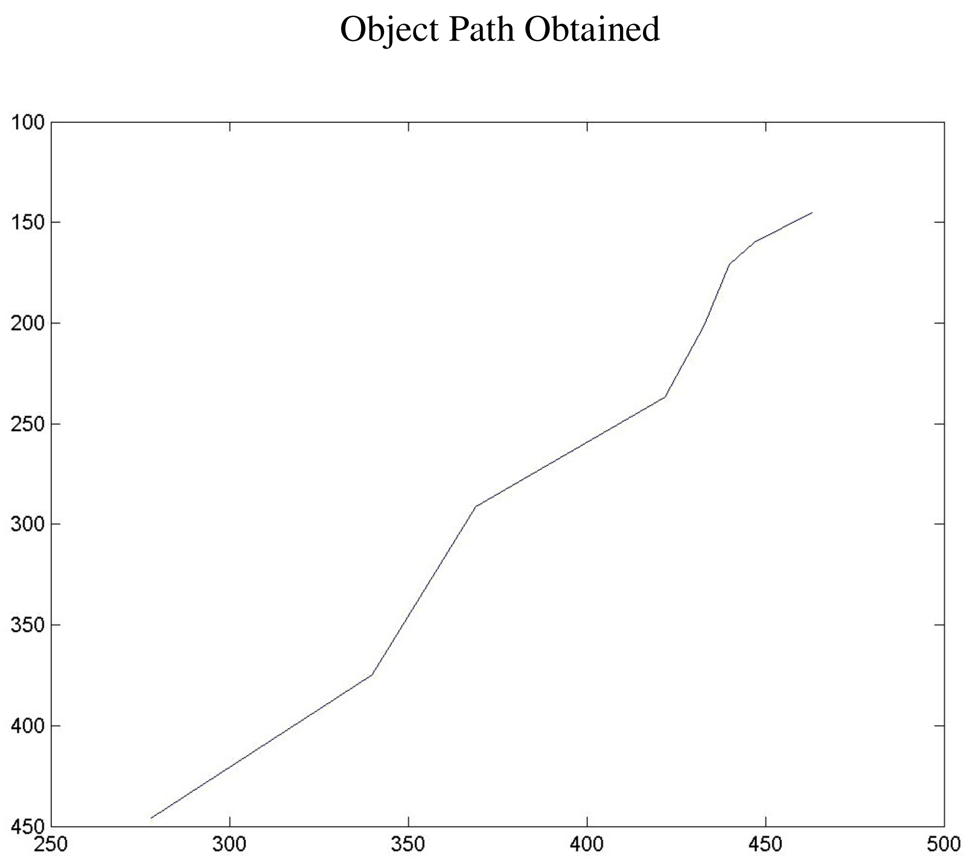


Figure 2.9: Pixel Values obtained from FPGA plotted using Matlab

CHAPTER 3
CONCLUSION

Object identification and tracking requires the use of an efficient signal processing system. Although video processing is achievable on serial processors, it can be beneficial to take advantage of the parallelism, low cost, and low power consumption offered by FPGAs. The successful implementation of this image processing algorithm illustrates that the digital signal processing required for high rate sensing application can be efficiently implemented on FPGA hardware.

The gray scale transformation has been used to remove the coherence of the background and the target to be tracked. The Delta Frame-based segmentation and Thresholding combine two intensive operations into one step, eliminating the need for large numbers of parallel comparators. The resulting optimized enhanced image fits on a small FPGA, such as the Xilinx Spartan- III XC3S500E, with sufficient resources available for an application to make use of the derived tracking information. We have demonstrated this by designing a simple video which contains an object in motion.

When we compare the outputs obtained from Matlab and FPGA, we find the outputs obtained using the Spartan 3E kit are computationally efficient. The pixel values are scaled and the outputs are comparable to the ones obtained using Matlab.

APPENDIX 1

Pseudo Code for Object Tracking C Program

```
//Median Function
Module FindMedian( Values )
Start
Sort the Values
If CountOf(Values) Is Even
    Return Mean(Middle Two Values)
Else
    Return Middle Value
End

Module MainProgram
Start
//Initialize the variables
Initialize MatVal, MatVal1, IntVal, IntVal1 To 0
Initialize cp To 0x25000000
Initialize cp1 To 0x25100000
Initialize NoOfRows,NoOfColumns To 128
Initialize NoOfRows1, NoOfColumns1 To 120
//Read 1st Input
For I = 0 To NoOfRows
    For J = 0 To NoOfColumns
        While( True )
            If Ch Ranges from 0 to 9 Then
```

```
                MatVal = (MatVal * 10) + Ch
            Else If
                RedB[i][j] = MatVal;
                MatVal = 0
                Increment cp
                    If Ch is NewLine or IntVal Is 010 Or 020 Or
                    003 Or 032 Then
                    Increment cp
                End If
                Break Out Of While Loop
            End If
        End While
        Increment cp
    End For
    Input[i][j] = RedB[i][j]
End For
```

//Read 2nd Input

```
For I = 0 To NoOfRows1
    For J = 0 To NoOfColumns1
        While( True )
            If Ch1 Ranges from 0 to 9 Then
                MatVal1 = (MatVal1 * 10) + Ch
            Else If
                RedB1[i][j] = MatVal1;
                MatVal1 = 0
                Increment cp1
```

```
                    If Ch1 is NewLine Or IntVal1 Is 010 Or 020
                    Or 003 Or 032 Then
                    Increment cp1
                End If
                Break Out Of While Loop
            End If
        End While
        Increment cp1
    End For
    Input1[i][j] = RedB1[i][j]
End For
```

//Delta Frame Generation

```
For I = 0 To NoOfRows
    For J = 0 To NoOfColumns
        Seg[i][j] = Input[i][j] - Input1[i][j]
        If Seg[i][j] > 20 Then
            Seg1[i][j] = 255
        Else
            Seg1[i][j] = 0
        End If
    End For
End For
```

//Median Filter

```
For I = 0 To NoOfRows
    For J = 0 To NoOfColumns
        If Not Any Edge Or Corner Then
            bbr = Median( All the Values In Seg1 )
        End If
        FirstFilter[i][j] = bbr
    End For
End For
```
//Thresholding
```
For I = 0 To NoOfRows
    For J = 0 To NoOfColumns
        If FirstFilter[i][j] > 40 Then
            FirstFilter[i][j] = 255
        Else
            FirstFilter[i][j] = 0
        End If
        EdgeImage[i][j] = 255
    End For
End For
```
//Edge Detection
```
For I = 0 To NoOfRows
    For J = 0 To NoOfColumns
        If I == 0 And J == 0 Then
                If FirstFilter[j][i+1] And FirstFilter[j+1][i+1] And
                FirstFilter[j+1][i] Are Equal To 255 Then
                    EdgeImage[j][i] = 0
        End If
        Else If I == 0 And J == Length Then
            If FirstFilter[j+1][i] And FirstFilter[j+1][i-1] And
            FirstFilter[j][i-1] Are Equal To 255 Then
                EdgeImage[j][i] = 0
            End If
        Else If I == Length And J == 0 Then
            If FirstFilter[j-1][i] And FirstFilter[j-1][i+1] And
            FirstFilter[j][i+1] Are Equal To 255 Then
                EdgeImage[j][i] = 0
            End If
        Else If I == Length And J == Length Then
            If FirstFilter[j-1][i-1] And FirstFilter[j][i-1] And
            FirstFilter[j-1][i] Are Equal To 255 Then
                EdgeImage[j][i] = 0
            End If
        Else If I == 0 And J == Length Then
            If FirstFilter[j+1][i] And FirstFilter[j+1][i-1] And
            FirstFilter[j][i-1] Are Equal To 255 Then
                EdgeImage[j][i] = 0
            End If
        Else If I == 1
            If FirstFilter[j][i-1] And FirstFilter[j+1][i-1] And
            FirstFilter[j+1][i] And FirstFilter[j+1][i+1] And
            FirstFilter[j][i+1] Are Equal To 255 Then
                EdgeImage[j][i] = 0
            End If
        Else If I == Length
            If FirstFilter[j-1][i-1] And FirstFilter[j+1][i-1] And
            FirstFilter[j+1][i] And FirstFilter[j-1][i] And
            FirstFilter[j][i-1] Are Equal To 255 Then
                EdgeImage[j][i] = 0
```

```
                End If
            Else If J == 1
                If FirstFilter[j][i-1] And FirstFilter[j-1][i-1] And
                FirstFilter[j-1][i] And FirstFilter[j-1][i+1] And
                FirstFilter[j][i+1] Are Equal To 255 Then
                    EdgeImage[j][i] = 0
                End If
            Else
                If FirstFilter[j-1][i-1] And FirstFilter[j+1][i-1] And
                FirstFilter[j+1][i] And FirstFilter[j+1][i+1] And
                FirstFilter[j][i-1] And FirstFilter[j-1][i] Are Equal To 255
                Then
                    EdgeImage[j][i] = 0
                End If
            End If
        End For
    End For
```

//Image Enhancement and Object Tracking

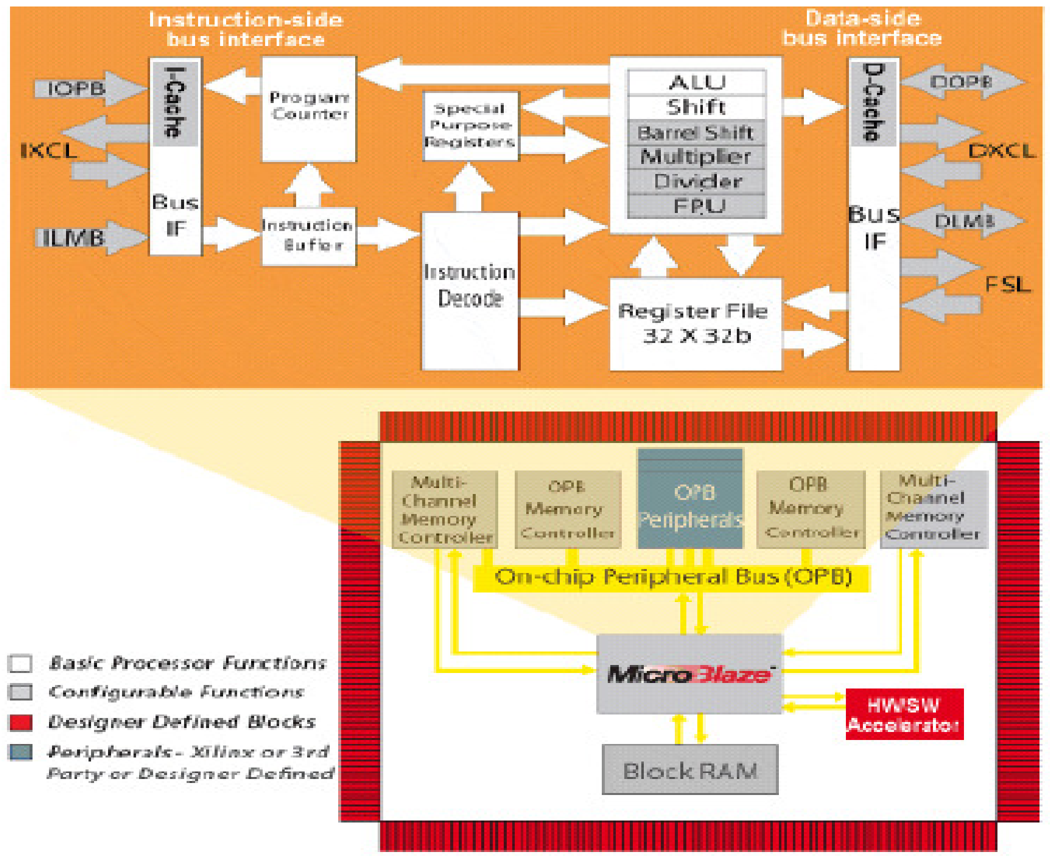
```
For I = 0 To NoOfRows
    For J = 0 To NoOfColumns
        Seg[i][j] = Input[i][j] - Input1[i][j]
        If Seg[i][j] = 1 Then
            Store [i],[j]
        End If
    End For
End For
End
```

APPENDIX 2

Development Tools: Microblaze Virtual Microprocessor

The MicroBlaze core is organized as a Harvard architecture with separate bus interface units for data accesses and instruction accesses. MicroBlaze does not separate between data accesses to I/O and memory (i.e. it uses memory mapped I/O). The processor has up to three interfaces for memory accesses: Local Memory Bus (LMB), IBM's On-chip Peripheral Bus (OPB), and Xilinx Cache Link (XCL). The LMB provides single-cycle access to on-chip dual-port block RAM (BRAM). The OPB interface provides a connection to both on-chip and off-chip peripherals and memory. The CacheLink interface is intended for use with specialized external memory controllers. MicroBlaze also supports up to 8 Fast Simplex Link (FSL) ports, each with one master and one slave FSL interface. The FSL is a simple, yet powerful, point-to-point interface that connects user developed custom hardware accelerators (co-processors) to the MicroBlaze processor pipeline to accelerate time-critical algorithms.

The backbone of the architecture is a single-issue, 3-stage pipeline with 32 general-purpose registers, an Arithmetic Logic Unit (ALU), a shift unit, and two levels of interrupt. This basic design can then be configured with more advanced features to tailor to the exact needs of the target embedded application such as: barrel shifter, divider, multiplier, single precision floating-point unit (FPU), instruction and data caches, exception handling, debug logic, Fast Simplex Link (FSL) interfaces and others. This flexibility allows the user to balance the required performance of the target application against the logic area cost of the soft processor.



The items in white are the backbone of the MicroBlaze architecture while the items shaded gray are optional features available depending on the exact needs of the target embedded application. Because MicroBlaze is a soft-core microprocessor, any optional features not used will not be implemented and will not take up any of the FPGAs resources.

Instruction Operations:

The MicroBlaze pipeline is a parallel pipeline, divided into three stages: Fetch, Decode, and Execute. In general, each stage takes one clock cycle to complete. Consequently, it takes three clock cycles (ignoring delays or stalls) for the instruction to complete. Each stage is active on each clock cycle so three instructions can be executed simultaneously, one at each of the three pipeline stages. MicroBlaze implements an Instruction Prefetch Buffer that reduces the impact of multi-cycle instruction memory latency. While the pipeline is stalled by a multi-cycle instruction in the execution stage the Instruction Prefetch Buffer continues to load sequential instructions. Once the pipeline resumes execution the fetch stage can load new instructions directly from the Instruction Prefetch Buffer rather than having to wait for the instruction memory access to complete. The Instruction Prefetch Buffer is part of the backbone of the MicroBlaze architecture and is not the same thing as the optional instruction cache.

Stack:

The stack convention used in MicroBlaze starts from a higher memory location and grows downward to lower memory locations when items are pushed onto a stack with a function call. Items are popped off the stack the reverse order they were put on; item at the lowest memory location of the stack goes first and etc.

Registers:

The MicroBlaze processor also has special purpose registers such as: Program Counter (PC) can read it but cannot write to it, Machine Status Register (MSR) to indicate the status of the processor such as indicating arithmetic carry, divide by zero error, a Fast Simplex Link (FSL) error and enabling/disabling interrupts to name a few. An Exception Address Register (EAR) that stores the full load/store address that caused the exception. An Exception Status register (ESR) that indicates what kind of exception occurred. A Floating Point Status Register (FSR) to indicate things such as invalid operation, divide by zero error, overflow, underflow and denormalized operand error of a floating point operation.

Interrupts:

MicroBlaze also supports reset, interrupt, user exception, break and hardware exceptions. For interrupts, MicroBlaze supports only one external interrupt source (connecting to the Interrupt input port). If multiple interrupts are needed, an interrupt controller must be used to handle multiple interrupt requests to MicroBlaze. An interrupt controller is available for use with the Xilinx Embedded Development Kit (EDK) software tools. The processor will only react to interrupts if the Interrupt Enable (IE) bit in the Machine Status Register (MSR) is set to 1. On an interrupt the instruction in the execution stage will complete, while the instruction in the decode stage is replaced by a branch to the interrupt vector (address 0x10). The interrupt return address (the PC associated with the instruction in the decode stage at the time of the interrupt) is automatically loaded into general-purpose register R14. In addition, the processor also disables future interrupts by clearing the IE bit in the MSR. The IE bit is automatically set again when executing the RTID instruction.

C Compiler:

Writing software to control the MicroBlaze processor must be done in C/C++ language. Using C/C++ is the preferred method by most people and is the format that the Xilinx Embedded Development Kit (EDK) software tools expect. The EDK tools have built in C/C++ compilers to generate the necessary machine code for the MicroBlaze processor.

EDK Interface:

The MicroBlaze processor is useless by itself without some type of peripheral devices to connect to and EDK comes with a large number of commonly used peripherals. Many different kinds of systems can be created with these peripherals, but it is likely that you may have to create your own custom peripheral to implement functionality not available in the EDK peripheral libraries and use it in your processor system.

The processor system by EDK is connected by On-chip Peripheral Bus (OPB) and/or Processor Local Bus (PLB), so your custom peripheral must be OPB or PLB compliant. Meaning the top-level module of your custom peripheral must contain a set of bus ports that is compliant to OPB or PLB protocol, so that it can be attached to the system OPB or PLB bus.

REFERENCES

1. Chi-Jeng Chang, Pei-Yung Hsiao, Zen-Yi Huang (2006). ‘Integrated Operation of Image Capturing and Processing in FPGA’, IJCSNS International Journal of Computer Science and Network Security, VOL.6 No.1A, pp 173-179.

2. Christopher T. Johnston, Kim T Gribbon, Donald G. Bailey (2005) ‘FPGA based Remote Object Tracking for Real-time Control’, 1st International Conference on Sensing Technology November 21-23, 2005 Palmerston North, New Zealand.

3. Crookes D., Benkrid K., Bouridane A., Alotaibi K., and Benkrid A.(2000), ‘Design and implementation of a high level programming environment for FPGA-based image processing’, Vision, Image and Signal Processing, IEE Proceedings, vol. 147, Issue: 4 , Aug, 2000, pp. 377 -384.

4. Hong C.S,. Chung S.M, Lee J.S. and Hong K.S. (1997), ‘A Vision-Guided Object Tracking and Prediction Algorithm for Soccer Robots’, IEEE Robotics and Automation Society, Vol. 1 pp: 346-351.

5. L. Baumela and D. Maravall, "Real-time target tracking," Aerospace and Electronic Systems Magazine, IEEE, vol. 10, no. 7, pp. 4-7, 1995.

6. L. Kilmartin, M. O Conghaile (1999), ‘Real Time Image Processing Object Detection and Tracking Algorithms’, Proceedings of the Irish Signals and Systems Conference, NUI, Galway, June 1999, pp. 207-214.