FPGA Based Visual Object Tracking

By

GC Obaid Ullah Khan

GC Mehmood Ahmad

GC Usama Bin Sana

FC Saleh Abdullah Kaifi

Submitted to the Faculty of Department of Electrical Engineering, Military College of Signals, National University of Sciences and Technology, Islamabad in partial fulfillment for the requirements of a B.E Degree in Telecom Engineering

May 2015

Abstract

In Contemporary technological world, high speed computers and economical and low cost video cameras are easily accessible. Therefore, a great interest has been placed in the field of object tracking due to the requirement of analyzing the automated videos. In visual object tracking the target of interest is tracked in consecutive video frames. It has enormous applications in security and surveillance systems, medical imaging, robotics, and traffic control.

The current thesis illustrates the implementation of visual object tracking algorithms on Field programmable gate arrays (FPGA). FPGAs are the most proficient, high speed and inexpensive devices, provides fast execution of IP algorithms which is not achievable with the serial microcontrollers. Verilog Spartan 3E board is used to execute the project logic and the visual object tracking algorithms.

It is hereby certified that the contents of the project report entitled “FPGA based Visual Object tracking”, submitted by the syndicate of

1. GC Obaid Ullah Khan
2. GC Usama bin Sana
3. GC Mehmood Ahmad
4. FC Saleh Kaifi

have been found satisfactory as per the requirement of the B.E. Degree in Electrical (Telecom) Engineering.

Supervisor:

Lecturer Moiz Ahmad Pirkani

MCS, NUST

_____

DECLARATION

We hereby declare that no content of work presented in this thesis has been submitted in support of another award of qualification or degree either in this institution or anywhere else.

DEDICATED TO

Almighty Allah,

Faculty for their help

And our parents for their support

ACKNOWLEDGEMENT

Nothing happens without the will of Allah Almighty. We thank Allah Almighty for giving us knowledge and strength to accomplish this task successfully.

We would like to thank our project supervisor, Lecturer Moiz Ahmed Pirkani. Without his support and encouragement, it would not have been possible to complete this project.

We would like to express our special gratitude and thanks to Lec Moiz Ahmed Pirkani for his unflinching support in truly testing times.

We would also like to thank our colleagues for helping us in developing the project and people who have willingly helped us with their abilities.

Last but not least, we are very thankful to our parents, who bore with us in times of difficulty and hardship. Without their consistent support and encouragement we could not have accomplished our targets successfully.

Table of Contents

CONTENTS **PAGE NUMBER**

List of Figures

List of Tables

Acronyms

VOT	Visual Object Tracking
FPGA	Field Programmable Gate Array
IOBs	Input output blocks
RAM	Random Access Memory
USB	Universal Serial Bus
UART	Universal Asynchronous Receiver/Transmitter
IP	Image Processing
IP Core	Intellectual property core
LSI	Large scale integration
DSP	Digital Signal Processing
DCM	Digital Clock Manager
ARWZ	Architecture Wizard
VGA	Visual Graphics Array
.coe	Coefficient File
.bit	Bitstream File

Chapter 1

Introduction

The fundamental idea of this project is to design such a system that tracks the object of interest in consecutive frames of a video. Image processing algorithms is used to do this. First the images are stored in memory of FPGA and afterward by using Verilog the image processing algorithms are implemented in FPGA and finally displayed on VGA. The basic block diagram of the project is shown in the Figure 1-1-1.

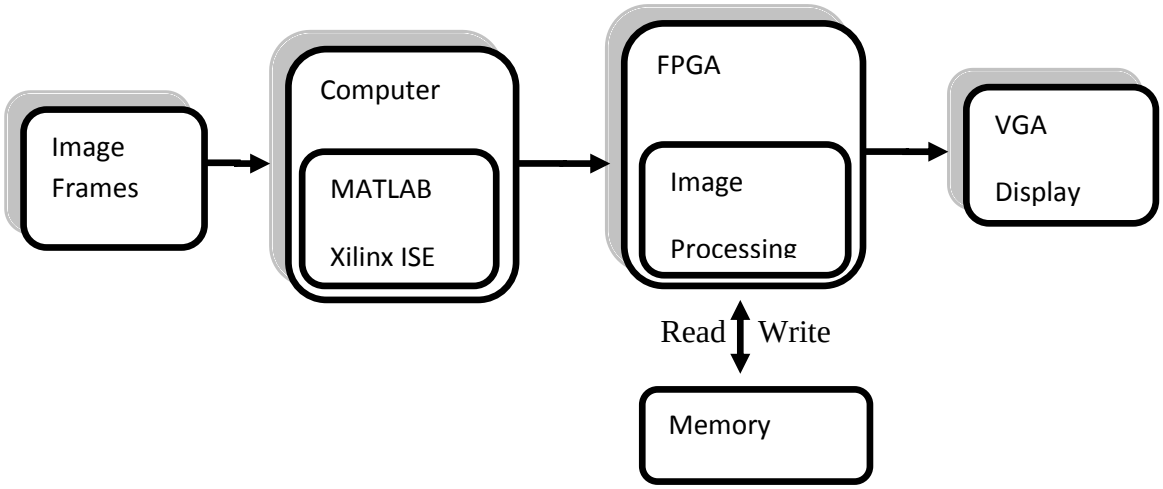


Figure 1-1 Basic Block Diagram

1.1 Statement of Need

In present world the need of flexible and high speed image processing is greatly growing. While data of image is incredibly large therefore this cannot be done using the conventional microprocessors. This is because the image pixel is scanned one by one by microprocessor and the data is processed successively. This method of image processing is excessively slow for real time because the processing speed becomes low.

As a result, reconfigurable hardware in the form of FPGAs has been proposed to acquire high performance for computationally intensive DSP application. FPGAs have programmable logic blocks with the small width and the programmable interconnections provide an immense flexibility for the real time, parallel processing design.

1.2 Visual Object Tracking

To mark the position of a target of interest in consecutive image frames of a video is called visual object tracking. After the reception of video, different algorithms of image processing are implemented to analyze the frame. These algorithms not only find the location of the desired object but also give much more information about it. This information can be the object's shape, size, average speed and directions. These algorithms involve real time processing at high rates.

1.3 Field Programmable Gate Arrays

Field programmable means that the function of FPGA is explained by the programmer not the manufacturer. Therefore its functioning is carried out by scripting some program depending on the particular application. This is how FPGAs provide flexibility to the users. FPGAs are prefabricated chips that contains logic block that are interrelated and interconnected. These interconnections are programmable wires. Complex functions can be completed on the logic blocks. There are a number of IOBs. These fast and bidirectional IOBs communicate and correspond with the outside environment of the FPGA. FPGA's have their built in on-chip memory.

1.3.1 FPGA Programming

To program the FPGA specific software is needed. In the current project we used the Xilinx ISE 14.2 to write, compile and synthesize the codes. It translates the code and then places and routes to generate the final programming file which is the .bit file. This software is also used to burn these files onto the FPGA.

1.4 Project Applications

Applications of object tacking are escalating day by day. A list of few applications is given below

- Surveillance systems
- Traffic control
- Medical Imaging
- Mobile Robotics
- Human Computer interaction
- Sports Analytics

1.5 Project Scope

The following tasks are to be performed in this project:

- ✓ Studying the basic image processing algorithms
- ✓ Writing image processing algorithms in MATLAB
- ✓ Software based Visual Object Tracking by using MATLAB
- ✓ Checking the compatibility of the proposed design for the kit
- ✓ Investigating features of the Spartan 3E starter board
- ✓ Executing UART and SDRAM through Verilog
- ✓ Initializing BRAM and implementing DCM through IP Cores
- ✓ Transferring image data to FPGA
- ✓ Implementing IP algorithms through Verilog coding
- ✓ Studying VGA and exhibiting the data on VGA monitor
- ✓ Study Architecture of MicroBlaze soft core processor

Chapter 2

Spartan-3E FPGA Starter Kit Board

2.1 SPARTAN 3E (XC3S500E) FPGA

SPARTAN 3E is used for the execution and completion of image processing algorithms. The Spartan-3E FPGA is entrenched with the 90nm technology. Spartan-3E FPGAs deliver up to 1.6 million system gates, up to 232 IOBs, and versatile platform FPGA architecture with the lowest cost per-logic in the industry.

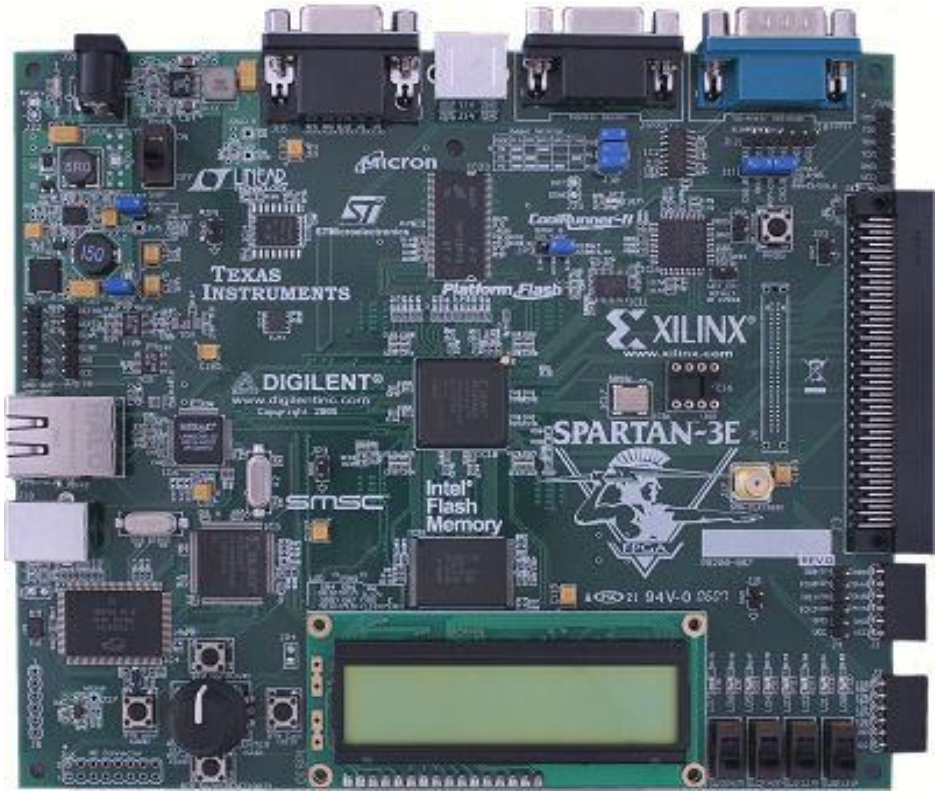


Figure 2-1Spartan 3E Starter Kit

2.1.1 Overview of architecture and features

FPGAs have integrated blocks of generally used functionality such as RAM, clock management, and DSP. The following are the vital mechanism in an FPGA.

2.1.1.1 Configurable Logic Blocks (CLBs)

The CLB is the basic logic unit in a FPGA. It gives the basic logic and storage functionality for a target application design. Every CLB comprises of a configurable switch matrix with 4 or 6 inputs, a number of selection circuitry (MUX, etc), and flip-flops. The switch matrix is extremely flexible and can be configured to handle combinatorial logic, shift registers or RAM.

2.1.1.2 Configurable IO blocks (IOBs)

A Configurable input/output (I/O) Block helps in conducting signals onto the chip and dispatching them back off again. The polarity of the output can generally be programmed for active high or active low output, and frequently the slow rate of the output can be programmed for fast or slow rise and fall times. There are characteristically flip-flops on outputs so that clocked signals can be output instantly to the pins without encountering significant delay, more easily meeting the setup time requirement for external devices. In the same way, flip-flops on the inputs decrease delay on a signal before reaching a flip-flop, consequently dropping the hold time requirement of the FPGA.

2.1.1.3 Interconnect

While the CLB imparts the logic capability, flexible interconnect routing routes the signals between CLBs and to and from I/Os. Routing comes in a number of flavors, from that designed to be linked between CLBs to fast horizontal and vertical long lines spanning the device to global low-skew routing for Clocking and other global signals.

2.1.1.4 Memory

Distributed RAM, Embedded Block RAM, SRAM, DDR RAM is accessible in nearly all FPGAs, which permits for on-chip memory for design. In addition there is single and dual port RAMs.

2.1.1.5 Digital clock management

Digital clock management is produced by mainly FPGAs that give precision clock synthesis essential for many designs.

2.2 Key Components and Features

The key features of the Spartan-3E Starter Kit board are:

- Xilinx XC3S500E Spartan-3E FPGA
    - Up to 232 user-I/O pins
    - 320-pin FBGA package
    - Over 10,000 logic cells
- Xilinx 4 Mbit Platform Flash configuration PROM
- Xilinx 64-macrocell XC2C64A CoolRunner™ CPLD
- 64 MByte (512 Mbit) of DDR SDRAM, x16 data interface, 100+ MHz
- 16 MByte (128 Mbit) of parallel NOR Flash (Intel StrataFlash)
    - FPGA configuration storage
    - MicroBlaze code storage/shadowing
- 16 Mbits of SPI serial Flash (STMicro)
    - FPGA configuration storage
    - MicroBlaze code shadowing
- 2-line, 16-character LCD screen
- PS/2 mouse or keyboard port
- VGA display port
- 10/100 Ethernet PHY (requires Ethernet MAC in FPGA)
- Two 9-pin RS-232 ports (DTE- and DCE-style)

• On-board USB-based FPGA/CPLD download/debug interface

• 50 MHz clock oscillator

• SHA-1 1-wire serial EEPROM for bitstream copy protection

• Hirose FX2 expansion connector

• Three Digilent 6-pin expansion connectors

• Four-output, SPI-based Digital-to-Analog Converter (DAC)

• Two-input, SPI-based Analog-to-Digital Converter (ADC) with programmable-gain pre-amplifier

• ChipScope™ SoftTouch debugging port

• Rotary-encoder with push-button shaft

• Eight discrete LEDs

• Four slide switches

• Four push-button switches

• SMA clock input

• 8-pin DIP socket for auxiliary clock oscillator

2.3 Why FPGA?

When the number of objects to be detected (tracked) enhance, processing becomes hard and complicated by a digital processor such as a PC. The reason is because a great number of operations are to be performed on each frame. Such a great number of operations per second could be accomplished efficiently with the help of FPGA (Field Programmable Gate Arrays). The FPGA's offer high level of parallelism which formulates processing of the video frames rapid and proficient. FGPA's comprises of reprogrammable logic blocks. These logic blocks are interconnected and

permit application particular hardware implementation. Thus they propose a high level of flexibility.

FPGA's are a cooperation between a processor based system and the application specific IC's (ASIC). There are a great number of benefits of using FPGA. Some of them are listed below:

- Parallelism
- Time to market
- Flexibility
- Reliability
- Low Power Consumption
- Performance Speed

Chapter 3

Image Processing

3.1 A survey on Visual Object Tracking

Visual object tracking has developed into a considerable technique in the past few years. A great number of papers have been published in this field in the past decade. One major cause behind the immense attraction of people in this field is that it is a contemporary challenging scientific problem. Secondly, visual object tracking is an element of a lot of high level problems such as mobile robotics, human computer interaction, surveillance systems, traffic control, sports analytics, medical imaging etc. They include problem like motion detection, activity understanding and event detection etc. Other major reason behind the development of this field is the higher obligation and advancement of HW/SW technology. With the advancement of this technology new tracking algorithms are being developed. The main constraint in the field is the performance of the system, as a large data set is mandatory to perform the complex operation on the image. As the algorithms being used turn out to be more complex and advance, it is required that the processing of these algorithms is high-speed and is less prone to error. To address this issue, the processing has been comprehensive for the serial processor used in the computer to another substitute that can carry out several operations on every pixel of the image consequential in greater number of operations per second. This is entirely accomplished by the use of FPGA. In our experimental results, we will show that there is a considerable enhancement and improvement in performance that can be attained as compared to processor based design.

3.2 Steps involved

A lot of work has been done in the field and various algorithms for object tracking have been proposed. Object tracking is mainly completed in two steps.

1. Object detection
2. Tracking of the detected object

There are various algorithms to perform the above mentioned tasks. We look at the overview of these.

3.2.1 Object Detection

Different methodologies used in object tracking are briefly described one by one in the section below.

3.2.1.1 Object Representation:

When an object of interest is to be tracked from the scene, it is required to be characterized and represented in some form. Different algorithms used for object tracking involve different representation and it is generally based on application.

Shape based tracking is possible by the shape information of the moving region of object. It is possible by representing object shape by points, geometric shapes, Object silhouette and contour, articulated shape models and skeletal models etc.

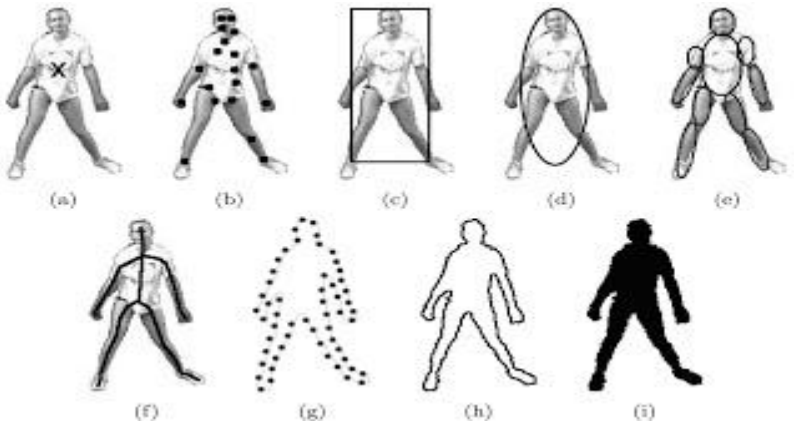


Figure 3-1 Object Representation

Object can also be showed on the basis of other than shape, or other features are joined with the shape representation. Some feature representations are probability densities of object's appearance; templates formed by taking help of shapes, joining shape and appearance of object or multi-view appearance models.

3.2.1.2 Object features:

Features of an object comprise of its color, boundaries, texture and optical flow. On the basis of particular application, the features for tracking are being selected. This is possible by visualizing that how the targeted object is unlike from the background. A

variety of methods are used for feature selection. Higher the prominence of the feature higher would its discrimination.

3.2.1.3 Strategies for detection of object:

An important step in tracking is object detection; once a detecting scheme has been applied the focus of attention is then only on the moving region. Only these regions are then measured for further processes. A number of the approaches for object detection are briefly described below.

Point detectors: in this technique the interested points are detected. These points should be clear, well positioned, significant texture in its localities and or view-point changes. These are useful in image identical and recognition.

Background subtraction: this technique is generally used for relatively static background. In it the difference between two consecutive frames of the video is made and then the thresholding is completed to extract the moving segment. As it uses pixel by pixel difference of the current image with the reference, therefore it is exceptionally sensitive to the changes in environment.

Image Segmentation:

This technique partitions image into small regions to identify objects. Different techniques have been used for segmentation by different people. Some of them are:

- Color based segmentation e.g. K-means clustering and Mean shift clustering.
- Transform methods e.g. watershed segmentation
- Texture methods e.g. texture filters
- Thresh holding methods e.g. Otsu's method.

3.2.2 Tracking

The purpose of tracking is to create trajectory of object in the frame. Classification of object tracking algorithms is given below.

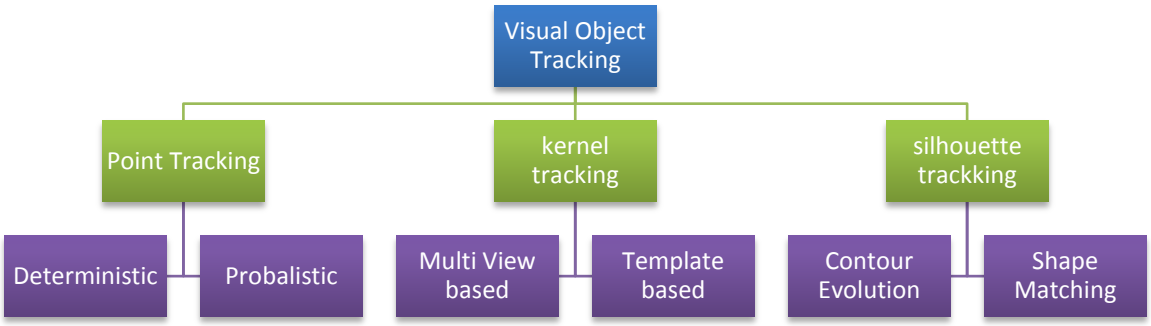


Figure 3-2 Types of Tracking

3.2.2.1 Point tracking:

Once the object is spotted the function of point tracking is to show it in the form of points, preceding object state is then taken as a reference. Subsequently these points are connected to make the object trajectory. Different approaches can be applied e.g. multi point correspondence, parametric transformation or contour evaluation. MCE tracker, GOA tracker, Kalman filter JPDAF, PMHT are the most common point tracking algorithms that have been used.

3.2.2.2 Kernel tracking

This is the tracking that depends on the object features, shape and appearance. Kernel of any shape can be selected to track the object; motion of the kernel represents the motion of the object in successive frames. The algorithms for kernel tracking are, mean-shift, KTL, layering.

3.2.2.3 Silhouette tracking

The object region is tracked by matching or contour evaluation by using the information of the object like its volume, surface density by its shape models. A silhouette is shaped in the object region. The representative work for silhouette tracking are state space models, vibrational models, heuristic models, Hausdorff, histogram.

In our project we used the kernel tracking algorithms to track the object in software implementation. While for the hardware implementation on FPGA we had to use relatively simpler technique. So the technique of background subtraction has been used in FPGA.

Chapter 4

Software Simulations using MATLAB

4.1 Basic Image Processing Algorithms

Following are the several fundamental image processing algorithms required for Visual object tracking in our project. We choose a simple image and demonstrate various operations, learned in the Image Processing, on MATLAB.

Figure 4-1 Test Image

4.1.1 Grayscale Conversion

Grayscale or intensity image is an image where each pixel is a sample that holds intensity information. The value of each pixel ranges from 0-255 and each pixel is of 8-bits or 1 Byte.

Figure 4-2 Gray Scaled Image

4.1.2 Thresh holding

Images thresh holding means transforming a grayscale image to a binary image i.e. the pixel values are either 1 or 0. We set a specific pixel value i.e. X for thresh holding and describe that values above X will be measured 1 and that below X would be considered 0.

Output = 0 if Image<=X; Output = 255 if Image>X

The below image is a thresholded image which has either black or white pixel values.

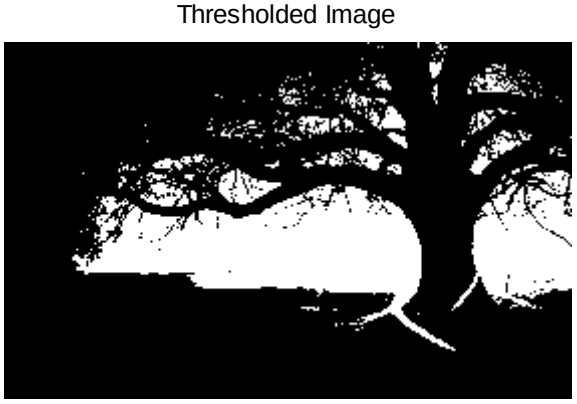


Figure 4-3 Thresholded Image

4.1.3 Image Enhancement using Arithmetic Operations

Image Enhancement is a technique used to operate and transform an image to our need by different operations on the image. It may be **Spatial Transformations** (directly dealing with image pixels) or **Frequency Domain Transformations** (dealing with the Fourier transform of the image). The best way to improve an image is by addition, subtraction, multiplication and division. Subtraction and multiplication is used to improve the image below:

Figure 4-4Arithmetically Enhanced Image

4.1.4 Edge Detection:

The process of identifying the points in an image where intensity of an image changes abruptly or jumps is called Edge Detection. Edge Detection is best performed on a grayscale image.

Figure 4-5 Edge Detection of the Image

4.1.5 Image Dilation

Image Dilation is one of the fundamental operations of Image processing on which other operations are based. It can be most simply stated as to probe and expand higher pixel values in a grayscale image or the bright area in the image.

Dilated Image

Figure 4-6 Dilated Image

4.1.6 Image Erosion

Image Erosion is another of the essential operations of Image processing on which other operations are based. Opposite to Image Dilation it expands lower pixel values in a gray scale image or the darker area in the image.

Eroded Image

Figure 4-7 Eroded Image

4.1.7 Image closing

Image Closing contrary to image opening can plainly be explained as the Erosion of the Dilation of an Image. It is one more of the image enhancement techniques.

Figure 4-8 Closed Image

4.1.8 2D Histogram

Image Histogram is graphical representation of the intensity distribution in an image. It plots number of pixels on the y-axis and intensity values on the x-axis. By knowledge of the histogram of an image, viewer is able to know the whole intensity distribution in the image.

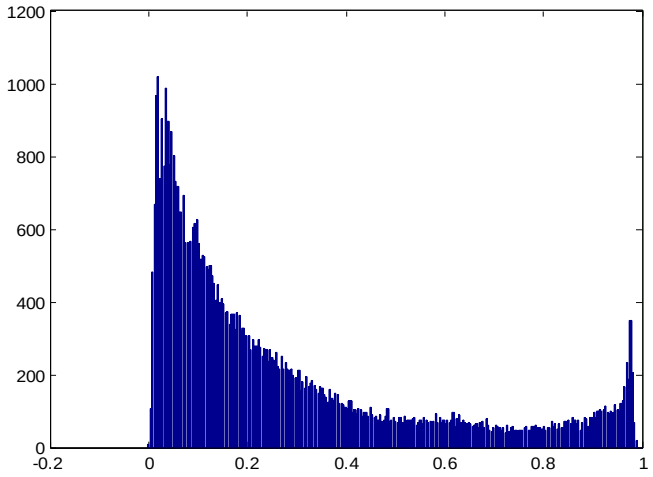


Figure 4-9 2-D Histogram

4.1.9 Histogram Equalizer

Histogram Equalization is a technique of image enhancement using contrast adjustment. The contrast of the below image is increased to get the better image.

Figure 4-10 Histogram Equalizer

4.2 Object Tracking Using MATLAB

Our video was simple with only one black object, a ball on a white background.

Kernel Tracking

The method used to track the object was Normalized 2D cross Correlation. This type of tracking is called Kernel Tracking.

4.2.1 Normalized Cross Correlation

Cross Correlation is a measure of resemblance of two images which are spaced in time. The idea is to chose the target object and cross correlate it with the image samples. The tracking is done on the basis of the maximization of similarity in the image.

Normalized Cross Correlation function is depicted by the command ***normxcorr2.***

The command uses following algorithm in its processing:

1. It computes the cross correlation in the spatial or frequency domain.

2. Calculates local sums by pre-computing running sums.

3. Uses the local sums to normalize the cross-correlation to get correlation coefficients.

4.2.2 Steps Involved in Object Detection:

Following steps were involved in software based Object Tracking:

Figure 4-11 Object Detection

First of all the video is converted to frames. MATALAB generates 15 frames per second; this rate can be adjusted to the desired frame rate. The next steps involve Grayscale Conversion, thresh holding, image enhancement, edge detection, object identification and finally object tracking using normalized cross correlation.

The algorithms in this section have already been discussed in the previous section.

4.2.2.1 Target Object

Given below is the target object:

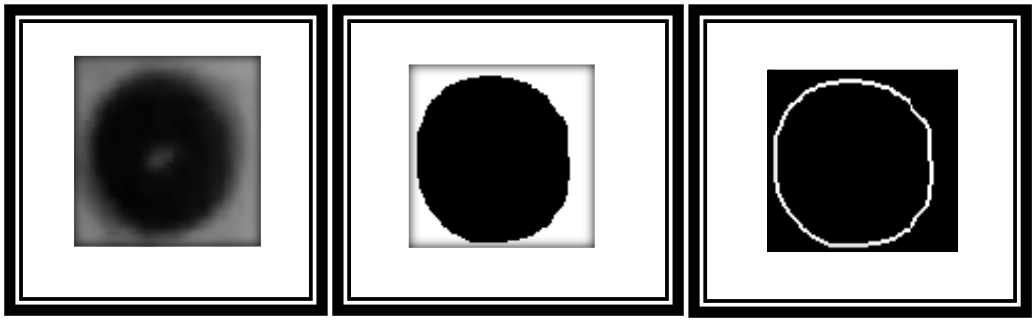


Figure 4-11 1 is the original image, 2 is thresholded image and 3 uses edge detection.

4.2.2.2 Object Tracking original Video Frames:

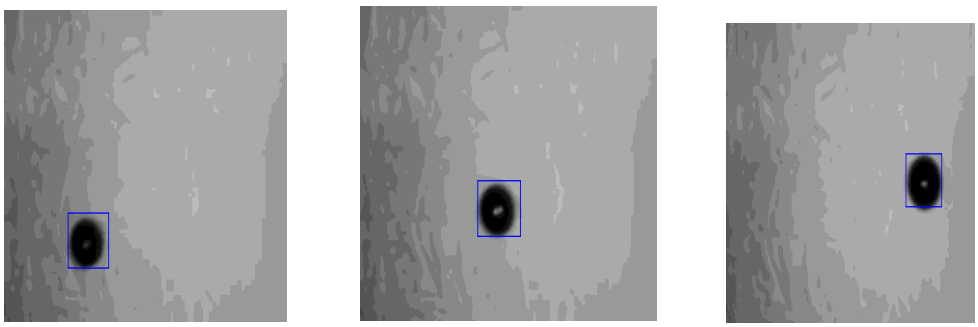


Figure 4-12 Object being Tracked

Chapter 5

VGA Display

5.1 VGA Display Port

The Spartan 3E board includes a VGA exhibit port via DB15 connector. The Spartan 3E FPGA drives five signals to show the respective data on the screen. The signal are the red, blue and green, one that show colors on the screen. Combination of theses three signals give eight colors as shown in the table 5.1. The other two signal are the horizontal synchornisation signal and the vertical synchronization signal written as hsync and vsync respectively. These two signal assist to show the pixel data on the screen in a correct manner.

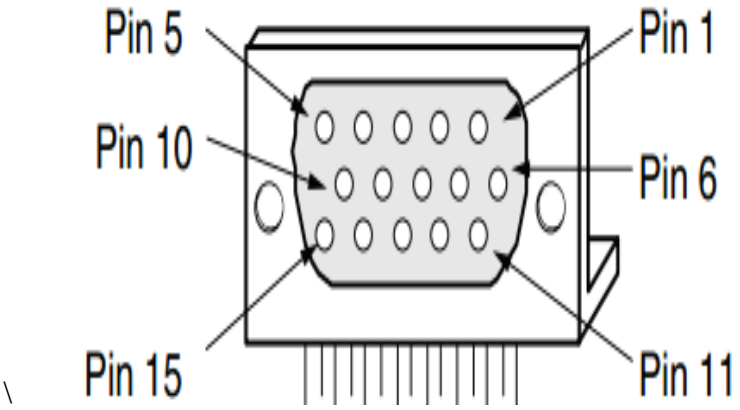


Figure 5-1 DB15 Connector

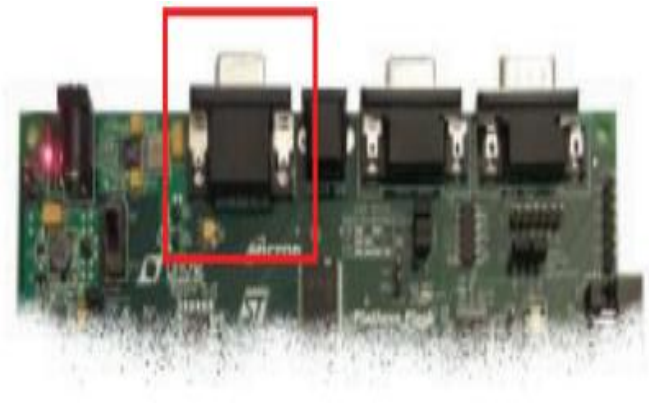


Figure 5-2 VGA Connector on FPGA

The figure below shows the pins of the VGA port and their connections. The FPGA pin numbers are also shown in the figure.

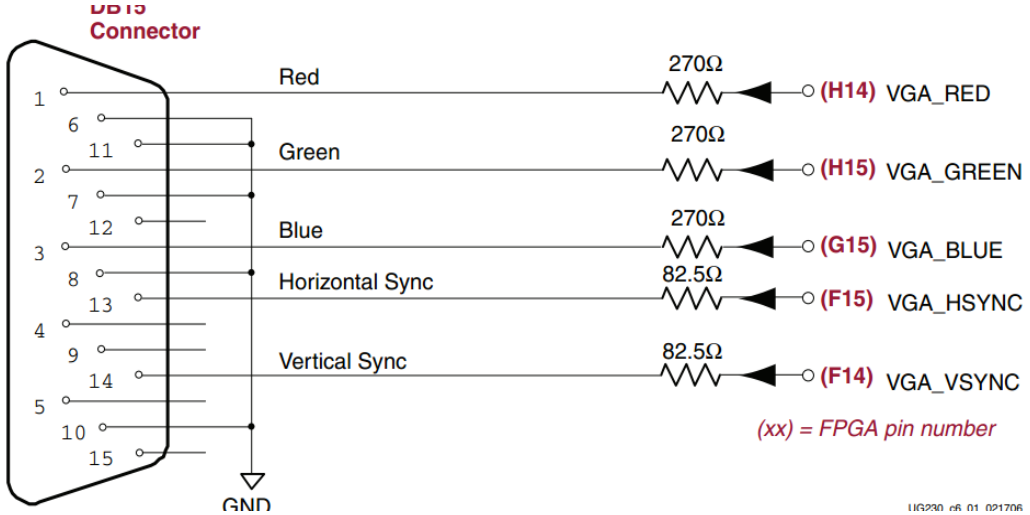


Figure 5-3VGA Signals

The three color pins of the VGA port are used to present 8 different colors in the screen. Table 5-1 shows the 3 bit color codes.

Table 5-1 3-bit display color codes

VGA_RED	VGA_GREEN	VGA_BLUE	Resulting Color
0	0	0	**Black**
0	0	1	**Blue**
0	1	0	**Green**
0	1	1	**Cyan**
1	0	0	**Red**
1	0	1	**Magenta**
1	1	0	**Yellow**
1	1	1	**White**

5.2 VGA signal timings

In our project we used the CRT based monitor that applies an electron beam that has been modulated and the information of images is exhibited on the phosphor screen. The electron beams are redirected by a strong magnetic field in top to bottom and left to right direction. The beam returns back in some time. As shown in the figure 6-4 the information is shown on the screen during the time when the beam is moving forward. The resolution of the exhibited data depends on the size and frequency of the beams.

As shown in Figure 6-4, the VGA controller produces the horizontal sync (HS) and vertical sync (VS) timings signals and the data is displayed on each pixel clock. The time to show one pixel on the screen defines the pixel clock. The VS signal explains the refresh frequency of the display, or the frequency at which all information on the display is redrawn. The minimum practical refresh frequency is 60 Hz and can go up to 120Hz.

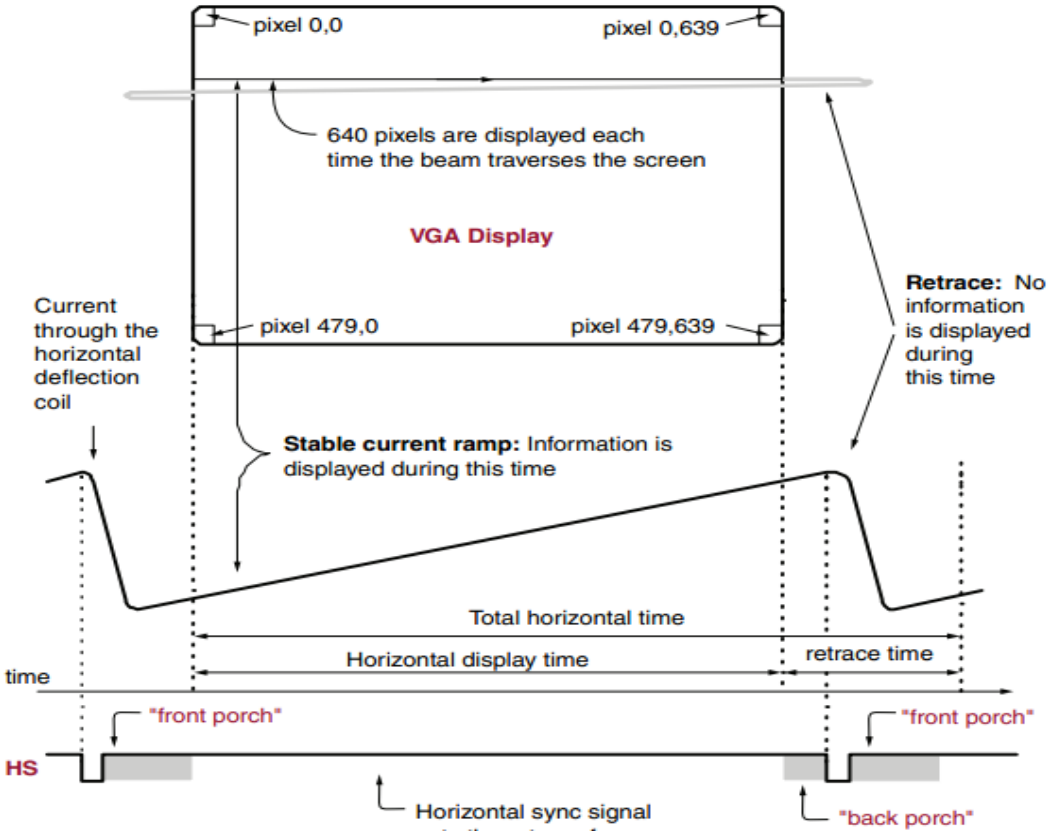


Figure 5-4 CRT Display Timings

The signal timings in Table 6-2 are derived for a 640-pixel by 480-row display using a 25 MHz pixel clock and 60 Hz ± 1 refresh. Figure 6-4 shows the relation between each of the timing symbols. The timing for the sync pulse width (TPW) and front and back porch intervals (TFP and TBP) are based on observations from various VGA displays. The front and back porch intervals are the pre- and post-sync pulse times. Information cannot be shown during these times.

Table 5-2 VGA Timing

Symbol	Parameter	Vertical Sync			Horizontal Sync	
		Time	Clocks	Lines	Time	Clocks
T_S	Sync pulse time	16.7ms	416,800	521	32 µs	800
T_{DISP}	Display time	15.36ms	384,000	480	25.6µs	640
T_{PW}	Pulse width	64 µs	1,600	2	3.84µs	96
T_F	Front porch	320µs	8,000	10	640ns	16
T_{BP}	Back porch	928µs	23,200	29	1.92µs	48

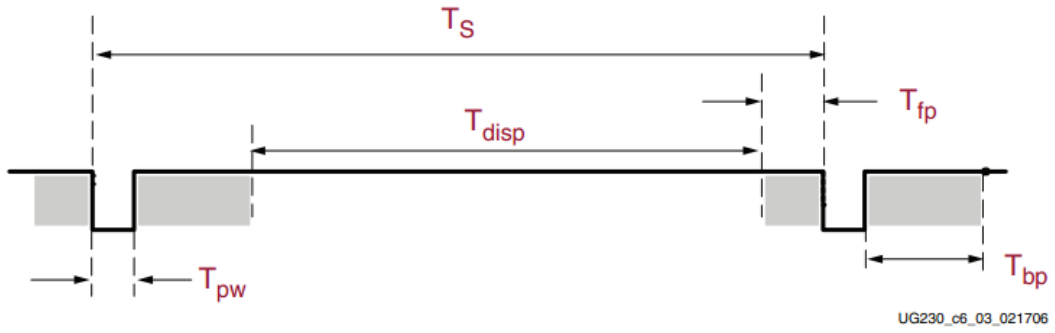


Figure 5-5 VGA Control Timing

Two separate counters are used that are clocked by the pixel clock. The horizontal counter controls the horizontal timings and the vertical counter controls the vertical timings of the pixel data. The source code used for VGA display is shown in the Appendix.

5.3 Implementation of the VGA display

The VGA timing was first set for 640x480 display screen. The first step was to display the eight colors on the screen vertically. This was successfully achieved. The display that was seen on the monitor is shown in the figure 6-6.

The next step was to display an image. The image data was stored in Bram and was read from there.

Finally tracking of the target Object was achieved and then displayed on the VGA monitor.

Chapter 6

Hardware Based Implementation

6.1 Block Diagram of the Proposed Approach

In order to achieve the task of successful tracking of an object, following series of steps was observed in our specific approach towards VOT.

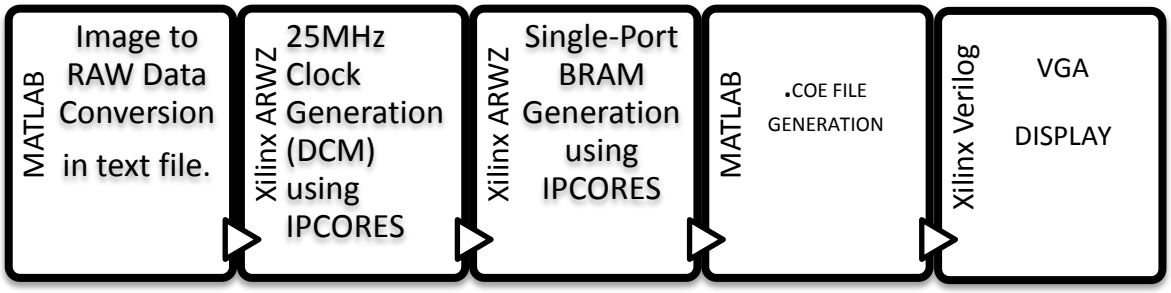


Figure 6-1Steps involved in hardware based implementation of VOT

6.2 Image to RAW Data Conversion

6.2.1 Raw Data

Raw data of an image is the digital value of each pixel in the image, initiating from the pixel at top left corner to the pixel at the bottom right corner. The range of pixel data may fluctuate depending on the bits of data of each image pixel. For gray scale image each pixel can have a value between 0~255. For monochrome image the data is single bit so only 2 values are possible. i.e. '0' or '1'.

In a digital camera when the imaging chip is exposed to capture the image, it records the amount of light that has hit each pixel. This is recorded as a voltage level. The camera's analog to digital converter circuitry changes the analog voltage signal to a digital value. Depending on the camera's circuitry if 8 bits of data are recorded then each pixel can handle 255 brightness levels.

6.2.2 Implementation

We took out images from a short video and each of these images was reduced to the pixel size of 200x150. From these images firstly a text file was produced using

MATLAB which contained the raw data of the image i.e. the values of each pixel without any header or extreme information. The code for writing raw data of an image to text file is in the appendix.

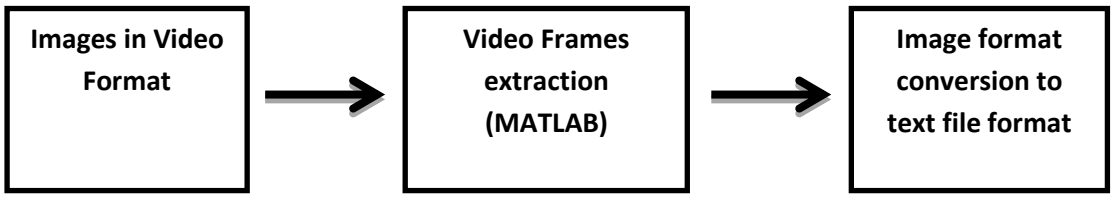


Figure 6-2 Image to Raw data conversion

6.3 Digital Clock Manager (DCM)

6.3.1 Introduction

The Digital Clock Manager (DCM) primitive in Xilinx FPGA parts is used to execute the following:

- Delay locked loop (DLL)
- Digital frequency synthesizer (DFS)
- Digital phase shifter (DPS)
- Digital spread spectrum (DSS)

Spartan 3E Starters Board has 8 internal global clock buses, and 4 Digital Clock Managers. Out of these 4 DCM's a single DCM is used in this project.

The Digital Clock Manger allows:

- Multiplying of an external clock
- Division of an external clock
- Clock skew compensation by phase difference correction.
- Phase shifting of clock signals

6.3.2 DCM Module I/O Signals

A single DCM has following main input output signals:

Table 6-1 DCM I/O Signal

Signal	I/O	Description
RST	I	If C_EXT_RESET_HIGH = 0, an inverter is inserted. Then three DFFs are inserted before this signal is connected to the equivalent pin of DCM.
CLKIN	I	If C_CLKIN_BUF = true, a BUFG is inserted; otherwise, this signal is connected to the equivalent pin of DCM directly.
CLKFB	I	If C_CLKIN_BUF = true, a BUFG is inserted; otherwise, this signal is connected to the equivalent pin of DCM directly.
CLK0	O	If C_CLKIN_BUF = true, a BUFG is inserted; otherwise, this signal is connected to the equivalent pin of DCM directly.
CLK90	O	If C_CLK90_BUF = true, a BUFG is inserted; otherwise, this signal is connected to the equivalent pin of DCM directly.
CLK180	O	If C_CLK180_BUF = true, a BUFG is inserted; otherwise, this signal is connected to the equivalent pin of DCM directly.
CLK270	O	If C_CLK270_BUF = true, a BUFG is inserted; otherwise, this signal is connected to the equivalent pin of DCM directly.
CLK2X	O	If C_CLKFX_BUF = true, a BUFG is inserted; otherwise, this signal is connected to the equivalent pin of DCM directly.
CLK2X180	O	If C_CLK2X180_BUF = true, a BUFG is inserted; otherwise, this signal is connected to the equivalent pin of DCM directly.
CLKDV	O	If C_CLKDV_BUF = true, a BUFG is inserted; otherwise, this signal is connected to the equivalent pin of DCM directly.
CLKFX	O	If C_CLKFX_BUF = true, a BUFG is inserted; otherwise, this signal is connected to the equivalent pin of DCM directly.
CLKFX180	O	If C_CLKFX180_BUF = true, a BUFG is inserted; otherwise, this signal is connected to the equivalent pin of DCM directly.
LOCKED	O	Connect to the equivalent pin of DCM directly

6.3.3 Functional Description

Functional Description of a variety of signals is used in the figure below:

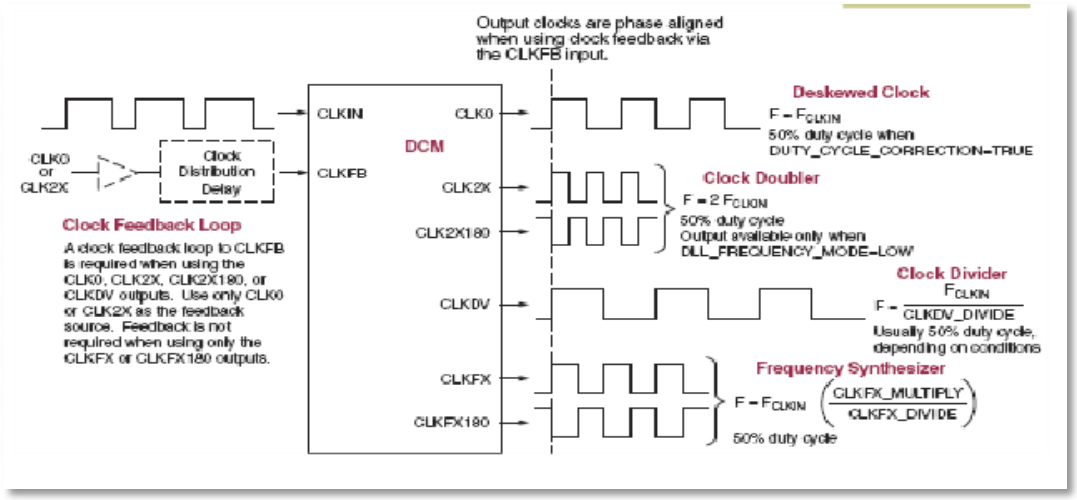


Figure 6-3 Function Diagram of DCM

6.3.4 25MHz Clock Generation

Since VGA and BRAM both work on a clock frequency of 25MHz. Hence a clock of 25MHz was generated from external FPGA clock of 50MHZ using a single DCM block. This single DCM block was generated using IP-Cores Xilinx Architecture Wizard (ARWZ) in Xilinx Design Suite. The DCM module had the RST checked but the output of the clock was not “Locked”. DFS is enabled

Signals used in out generated DCM include:

- RST: reset for the DCM
- CLKIN: external clock of 50MHz
- CLKFB: internal feedback of DCM
- CLK0: buffered input clock (dived by 2 for this particular case)

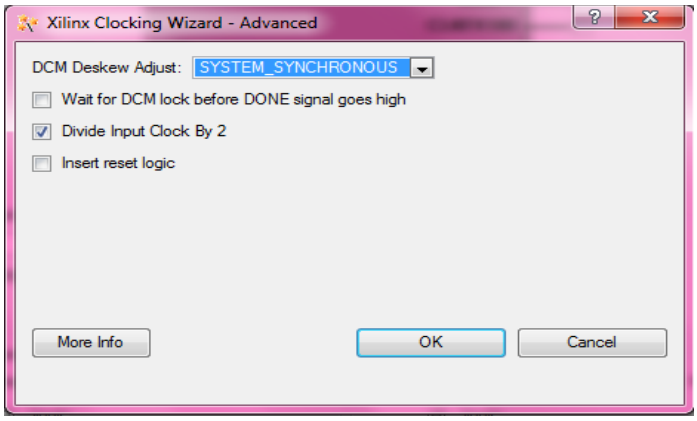


Figure 6-4 ARWZ settings for DCM

Attributes in the particular Application:

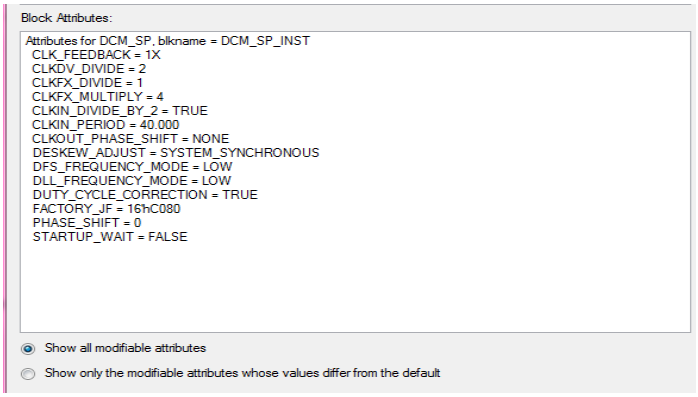


Figure 6-5Attributes of the generated clock

The CLK0 signal was provided as clock to the BRAM and VGA blocks.

6.4 Single Port BRAM using IP-CORES

6.4.1 BRAMs in SPARTAN 3E

Spartan 3E has a total of 20 BRAMS available out of which we used 18 BRAMS for our project.

Table 6-2 XC3S500E BRAM Specifications

Device	RAM Columns	RAM Blocks per Columns	Total RAM Blocks	Total RAM bits	Total RAM Kbits
XC3S500E	2	10	20	368640	360

6.4.2 Single-port BRAM Module I/O Signals

A single-port BRAM module has following main input output signals:

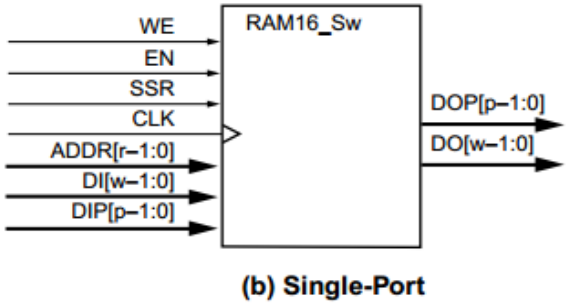


Figure 6-6 Single Port BRAM

Table 6-3 BRAM I/O Signals

Signal	I/O	Description
DI	I	Data Input bus
DIP	I	Parity Data Input Bus (available only for byte-wide and wider organizations)
ADDR	I	Address Bus
CLK	I	Clock
SSR	I	Synchronous Set/Reset
EN	I	Clock Enable
WE	I	Write Enable
DO	O	Data Output Bus.
DOP	O	Parity Data Output (available only for byte-wide and wider organizations)

6.4.3 IP-Core Generator block

It specifies the depth and width of the data in the core generator block. In our design we needed only a single pixel in 1 row of memory for convenience in our VGA Display process as a result the width and depth were specified as demonstrated in the figure. The depth was 30000 for images of 200x150 pixels each which took our memory space of 6 BRAMs.

Figure 6-7 BRAM width and Depth

6.4.4 Content Initialization

It is used to specify the initial RAM contents for a CORE Generator block RAM function, generate a coefficients (.coe) file. A simple example of a coefficients file appears in Figure 7. At a minimum, define the radix for the initialization data—i.e., base 2, 10, or 16—and then indicate the RAM contents starting with the data at location 0, followed by data at subsequent locations. To include the coefficients file, locate the appropriate section in the CORE Generator wizard and check Load Init File, as shown in Figure 8. Then, click Load File and select the coefficients file.

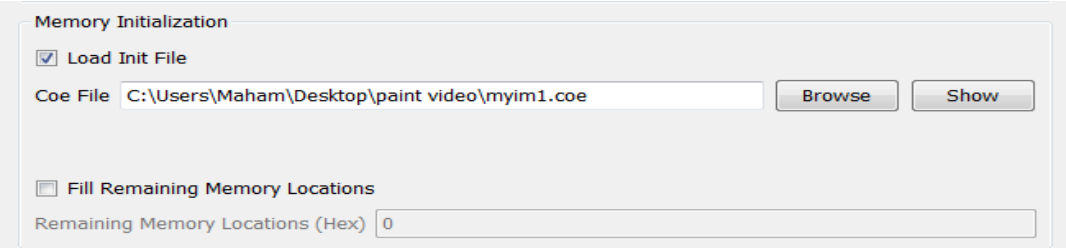


Figure 6-8 Image content Initialization

6.5 .coe File Generation

There are mainly two main methods of generating a .coe file.

1) Using IP core generators for generation the .coe file.

2) Writing a .coe file

6.5.1 Using IP-CORE Generator

IP-Core generators have a wizard that guides through the .coe file generation process and through it you can manually write desired data values into the BRAM. This is a good procedure to be followed when data to be stored is small.

6.5.2 Writing .Coe File

The problem with the above technique was that image data is enormous and it was not possible for us to manually write all the data to create the .Coe file so we switched to the 2nd approach.

In this approach we generated a MATLAB code that would write the data in a text file in the format same as that of a .Coe file and then the extension was changed to .txt to .Coe. This generated .Coe file was used in the BRAM for data of pixels up to (300x300) pixels which utilized about 15/20 BRAMs. MATLAB code is attached in the Appendix.

6.6 VGA Display

VGA was primarily tested and 8 colors were exhibited on it in order to ensure that it works.

Figure 6-9 VGA test results

For display, an image of 320x240 was chosen and shown on the VGA as out test image which used about 15 BRAMs.

Figure 6-10 VGA Image Display

FPGA was linked with the laptop for storing image data and burning the code in the FPGA and FPGA was attached to another monitor through DB-15 connector to show the image.

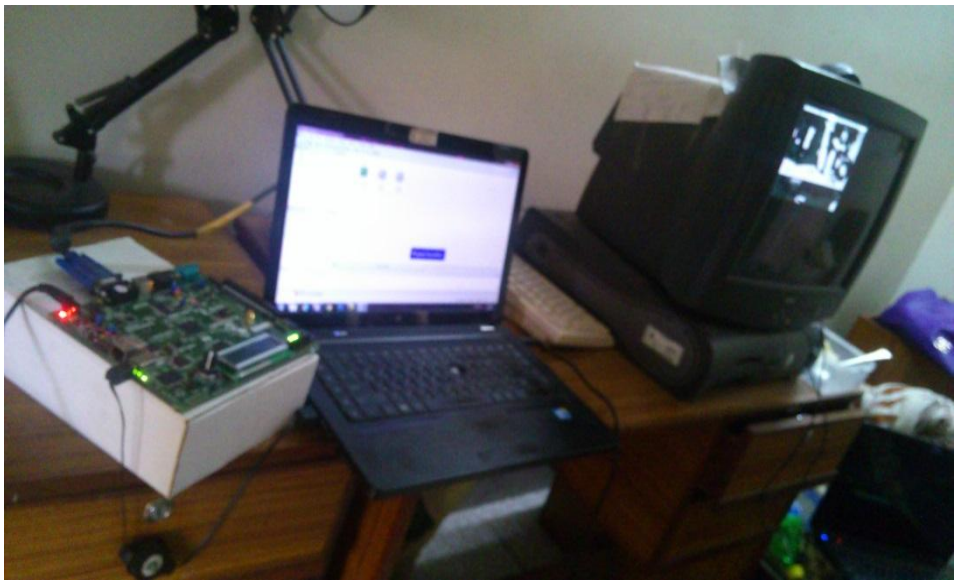


Figure 6-11Setup for VGA Image display

Chapter 7

MicroBlaze and Embedded Development Kit

7.1 MicroBlaze Architecture

MicroBlaze is an embedded and fixed soft core processor. It is an optimized version of a reduced instruction set computer (RISC) designed by Xilinx, Inc. It is executed in many Xilinx FPGAs. Fig shows the block diagram of MicroBlaze

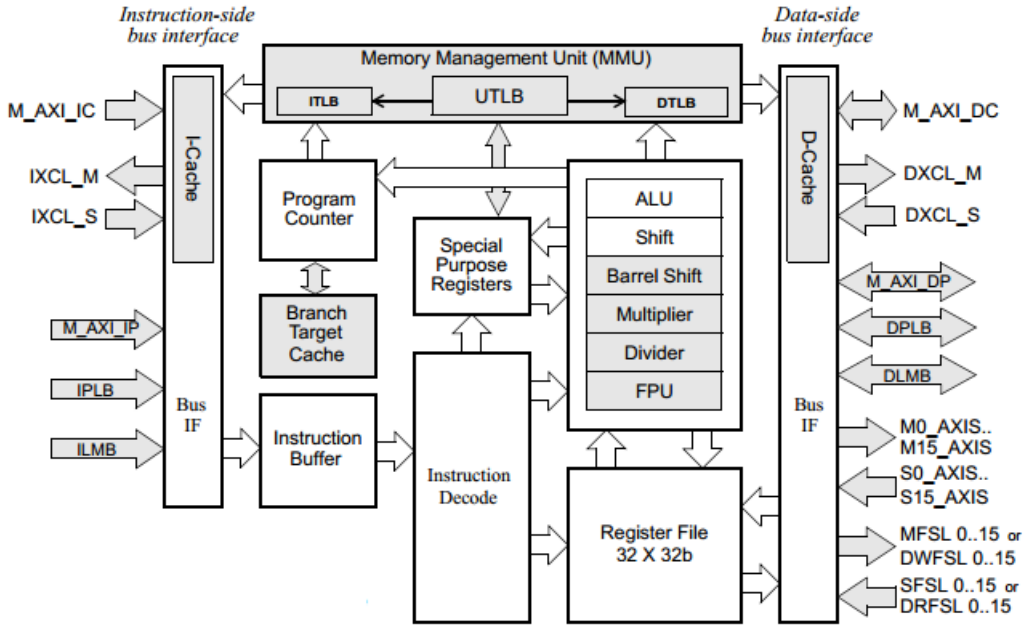


Figure 7- 1: MicroBlaze Block Diagram

7.1.1 Features

Some of the features of MicroBlaze are given below

- 32 bit general purpose registers
- 32 bit instruction word
- 32 bit address bus
- LMB BRAM memory
- Instruction and Data Caches
- Single issue pipeline stages

7.1.2 Data Types and Endianness

MicroBlaze make use of Big-Endian, bit-reversed format to symbolize data. The data types which the hardware supports for MicroBlaze are word, half word and byte.

7.1.3 Instructions

All MicroBlaze instructions are 32 bits wide. Instructions are of 2 types, Type A or Type B. Type A instructions have up to two source register operands and one destination register operand. Type B instructions have one source register; a 16-bit immediate operand and a single destination register operand.

7.1.4 Registers

MicroBlaze has an orthogonal instruction set architecture. It has 32-bit general purpose registers and up to seven 32-bit special purposes registers, depending on configured options.

7.1.5 Pipeline Architecture

MicroBlaze uses pipeline architecture for the execution of instructions. The pipeline is divided into five stages: Fetch (IF), Decode (OF), Execute (EX), Access Memory (MEM), and Write back (WB).

For the majority of instructions, every stage takes one clock cycle to complete. A particular instruction takes five clock cycles to complete, and one instruction is completed on every cycle. A few instructions entail multiple clock cycles in the execute stage to complete. This is accomplished by stalling the pipeline.

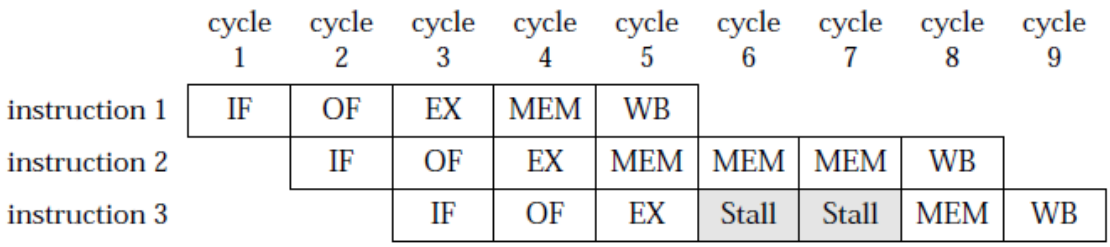


Figure 7- 2: Stages of Pipelining

7.1.6 *Memory Architecture*

MicroBlaze is implemented with Harvard memory architecture, i.e. instruction and data accesses are done in separate storage and signal conduits. Each address space hasa 32 bit range which can handle up to 4GByte of instructions and data memory. Both instruction and data an interface of MicroBlaze is 32 bit wide.

Data and instruction accesses must be aligned, unless the processor is configured to support unaligned exceptions.

MicroBlaze uses memory mapped I/O. The processor has up to three interfaces for memory accesses:

- **Local Memory Bus (LMB)**

  It provides on chip access to dual port block RAM.

- **On-Chip Peripheral Bus(OPB)**

  It provides access to on- chip and off-chip peripherals.

- **Xilinx Cache Link**

  It is used with the external memory controllers.

The LMB memory address range must not overlap with OPB or XCL ranges.

7.1.7 Floating Point Unit (FPU)

The MicroBlaze floating point unit uses IEEE 754 single precision floating point format, which includes definitions for infinity, not-a-number (NaN) and zero. It also supports addition, subtraction, multiplication, division, and comparison instructions, implements round-to-nearest mode, generate sticky status bits for: underflow, overflow, and invalid operation.

7.1.8 Fast Simplex Link (FSL)

MicroBlaze can be configured with Fast Simplex Link (FSL) interfaces, each comprises of one input and one output port. The FSL channels are dedicated unidirectional point to-point data streaming interfaces. Each FSL gives a low latency dedicated interface to the processor pipeline. Thus they are ideal for extending the processors execution unit with custom hardware accelerators.

7.2 Embedded Development Kit (EDK)

EDK is software available in ISE design suite. It facilitates the user to build processor based system.

The software has a good graphical design and many wizards existing to configure the embedded processor and peripherals.

The tools provided with EDK are designed to help in all phases of the surrounded design process, as demonstrated in the following figure.

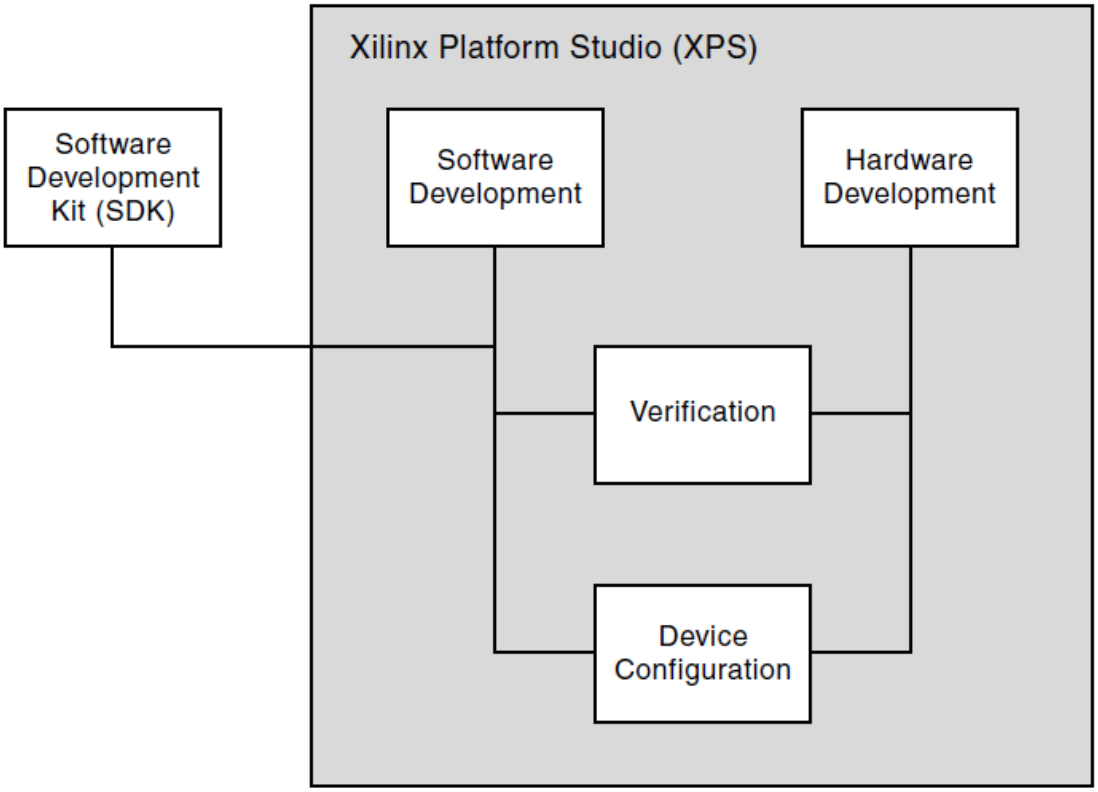


Figure 7- 3: Tools within EDK software

- Xilinx Platform Studio (XPS)
- Software Development Kit (SDK)

7.2.1 Xilinx Platform Studio (XPS)

XPS presents an environment for creating software and hardware specification flows for embedded processor systems. It gives a graphical system editor for connection of processors, peripherals, and buses. It is applied to add cores, edit core parameters, and makes bus and signal connections to make an MHS file, create and modify MSS file, produce and view system block diagrams. By using XPS the following peripherals can easily be connected to the embedded processor.

- Quad SPI/SRAM/NOR/NAND Flash
- Gigabit Ethernet
- UART

- I2C
- SPI
- CAN
- GPIO

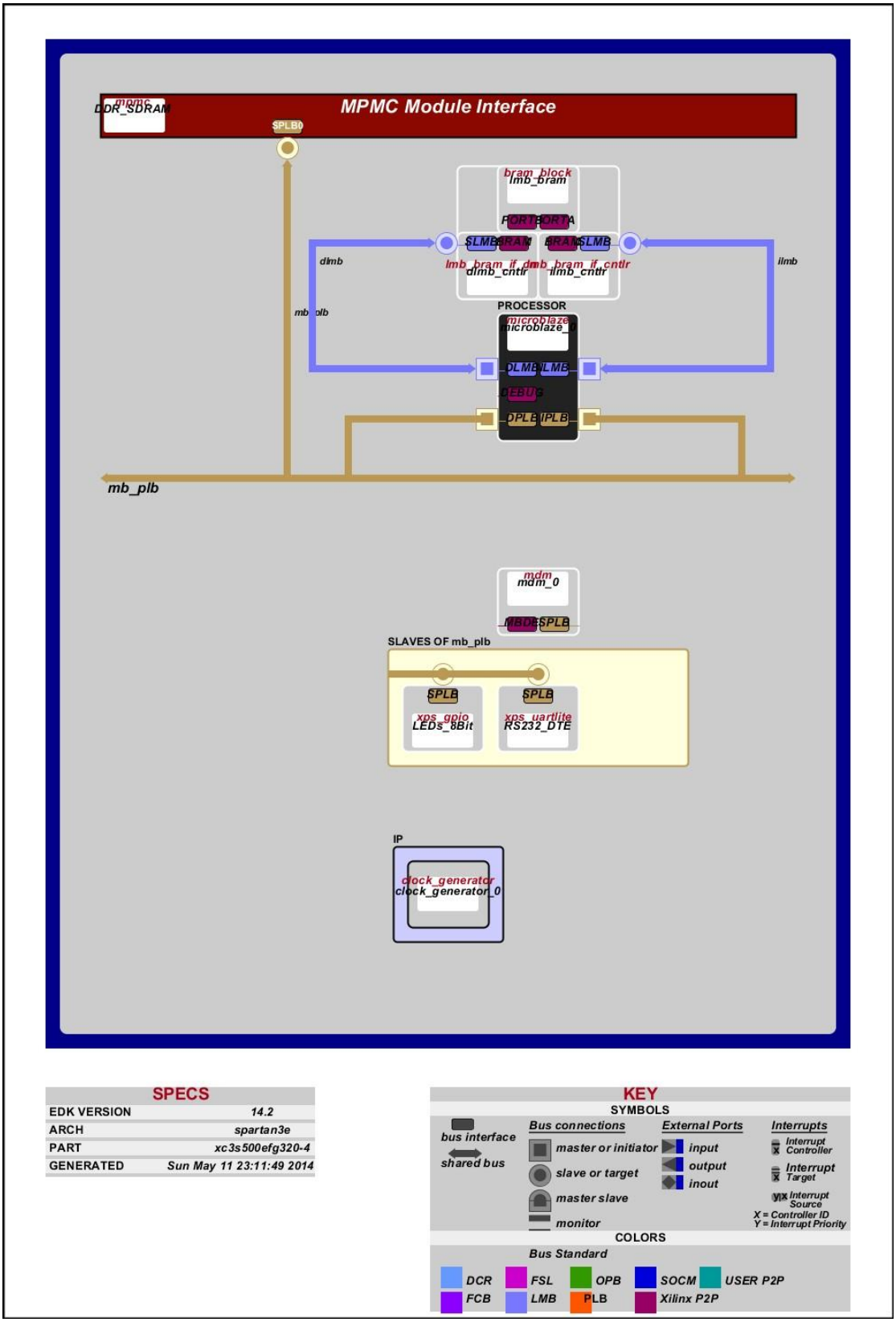
7.2.2 Xilinx Software Development Kit (SDK)

The Xilinx SDK is a GUI which provides a development environment for software application projects. SDK is based on the Eclipse open-source standard. It has C/C++ code editor, compilation environment; application build configuration and automatic make file generation

7.2.3 Experiment done on EDK

Firstly, image data is transformed to text file. Image data is sent to MicroBlaze by using UART. MicroBlaze stored the image data in DDR SDRAM. The data is read with the help of DDR SDRAM and exhibited on hyper terminal of PC.

Fig illustrates the block diagram of the overall hardware system.



The functionality code of the above hardware is written in C language using SDK.

Chapter 8

Conclusions and Recommendations

8.1 Tasks Completed

In summing up, we have learnt a lot from this project. We have got deep understanding of embedded system (Spartan 3e) and related software's (Xilinx ISE, EDK).

The tasks we completed in this project are:

- Implementation of BRAM (using ip core)
- VGA display.
- Implementation of Digital clock manager (using ip core)
- Object tracking in images stored in BRAM.
- MicroBlaze hardware and software design.

8.2 Results

8.2.1 FPGA Design Utilization Summary in our Project

Table 8-1 Design Summary of the project

Logic Utilization	**Used**	**Available**	**Utilization:**
Number of Slice Flip Flops	42	9,312	1%
Number of 4 input LUTs	144	9,312	1%
Number of occupied Slices	87	4,656	1%
Number of Slices containing only related logic	87	87	100%
Number of Slices containing unrelated logic	0	87	0%
Total no. of 4 input LUTs	158	9,312	1%
Number used as logic	144		
Numbers used as route-thru	14		
Number of bonded IOBs	31	232	13%
DCM	1	4	25%

Number of RAMB16s	18	20	90%
Number of BUFGMUXs	2	24	8%
Average Fanout of Non-Clock Nets	3.79		

8.2.2 BRAM Utilization

Table 8-2 BRAM utilized

Image Size Chosen	120x90 (8-bit gray scale image)
No. of BRAM used for a single Image	6
Memory Consumed per Frame	86.4 Kbits
Total RAM Kbits used in the project	259.2 Kbits

8.2.3 Processing time comparison

The comparison of the timing of the software and hardware implementations illustrates that the performance of FPGA is approximately 600 times better than the MATLAB simulations. As data increases so does the effectiveness comparison of the two approaches.

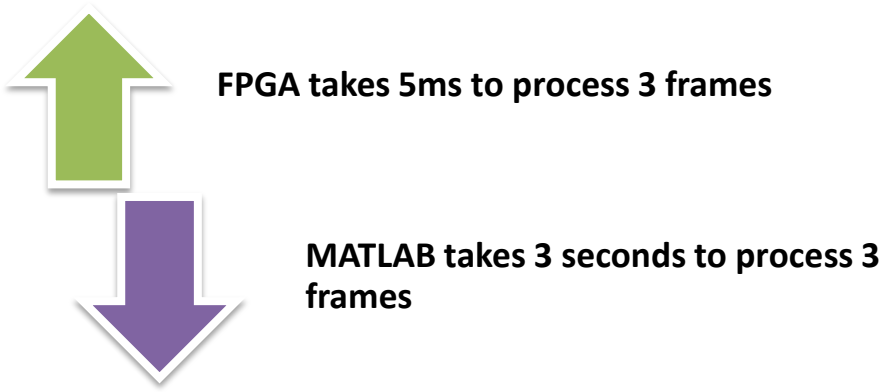


Figure 8-1 Comparison

8.3 Future Work

In this project we have done tracking on few images because of the constraints on the availability of BRAMs in Spartan 3E. So the future tasks are to:

- Use FPGA with better image processing platforms like the Cyclone® III FPGAs and Zynq series. These FPGAs have more memory; more IOBs and

can perform extensive differentiation and integration on system level. In addition they also have enhanced cost, performance and productivity.

- Perform more advance image processing algorithm for object tracking.
- Camera interfacing
- Implement 8 bit VGA.

References

[1]. Xilinx, MicroBlaze Processor Reference Guide, Embedded Development Kit EDK 14.1
URL: http://www.xilinx.com/support/documentation/sw_manuals/xilinx14_1/mb_ref_guide.pdf

[2]. Xilinx, Embedded System Tools, Reference Manual, Embedded Development Kit
URL: https://www.xilinx.com/support/documentation/sw_manuals/edk10_est _rm.pdf

[3]. Xilinx, Application Note, "Using Block RAM in Spartan-3 Generation FPGAs", XAPP463 (v2.0) March 1, 2005.

[4]. Xilinx, Application Note, "Digital Clock Manager (DCM) Module", DS485 April 24, 2009.

[5]. Xilinx, "Spartan-3E FPGA Starter Kit Board User Guide", UG230 (v1.2) January 20, 2011.

[6]. EE3810, "Advanced Features of Xilinx FPGA", dcm_module.

[7]. Alper Yilmaz, "Object tracking: A survey", University of Central Florida, ACM Computing Surveys (CSUR) Surveys Homepage archive Volume 38 Issue 4, 2006

[8]. Rafael C. Gonzalez, Richard Eugene Woods, 2nd edition, "Digital Image processing".

[9]. Radi H.R., Caleb W.W.K., M.N.Shah Zainudin., M.Muzafar Ismail., "The Design and Implementation of VGA Controller on FPGA", University Teknikal Malaysia Melaka Hang Tuah Jaya 76100 Durian Tunggal, Melaka, Malaysia.

[10]. ZAHEERUDDIN SYED, MUNWAR SHAIK, "FPGA IMPLEMENTATION OF VGA CONTROLLER", KITS warangal, Kakatiya University, Hasanparthy, Warangal -506015,Andhara Pradesh.

Appendix

A.1 Code for tracking and display(main module)

```
module vga_top(
        input wire clk,                 //master clock = 50MHz
        input wire clr,         //right-most pushbutton for reset BTN east
        output wire red,                //red vga output
        output wire green,      //green vga output
        output wire blue,               //blue vga output
        output wire hsync,              //horizontal sync out
        output wire vsync,                      //vertical sync out
        output wire [2:0]douta,
        output wire [2:0]douta2,
        //input [5:0]addra,
        input wea
        );


// VGA display clock interconnect
wire dclk;
wire [14:0]addra;
//wire [13:0]addra1;
wire [2:0]douta3;
wire [2:0]douta4;
```

```
wire [2:0]douta5;


// generate display clock
clockdiv U1 (
    .CLKIN_IN(clk),
    .RST_IN(RST_IN),
    .CLKIN_IBUFG_OUT(CLKIN_IBUFG_OUT),
    .CLK0_OUT(dclk)
    );




//----------- Begin Cut here for INSTANTIATION Template ---// INST_TAG
bram100 foo(
  .clka(dclk), // input clka
  .wea(wea), // input [0 : 0] wea
  .addra(addra), // input [13 : 0] addra
  .dina(), // input [2 : 0] dina
  .douta(douta) // output [2 : 0] douta
);
// INST_TAG_END ------ End INSTANTIATION Template ---------
```

```
//----------- Begin Cut here for INSTANTIATION Template ---// INST_TAG
bram100_2 foo1(
  .clka(dclk), // input clka
  .wea(wea), // input [0 : 0] wea
  .addra(addra), // input [13 : 0] addra
  .dina(), // input [2 : 0] dina
  .douta(douta2) // output [2 : 0] douta
);
// INST_TAG_END ------ End INSTANTIATION Template ---------

//----------- Begin Cut here for INSTANTIATION Template ---// INST_TAG
bram100_3 foo2(
  .clka(dclk), // input clka
  .wea(wea), // input [0 : 0] wea
  .addra(addra), // input [13 : 0] addra
  .dina(), // input [2 : 0] dina
  .douta(douta4) // output [2 : 0] douta
);
// INST_TAG_END ------ End INSTANTIATION Template ---------

assign douta3=douta~^douta2;
```

```
assign douta5=douta4~^douta2;



// VGA controller
vga640x480 U3(
        .dclk(dclk),
        .clr(clr),
        .hsync(hsync),
        .vsync(vsync),
        .red(red),
        .green(green),
        .blue(blue),
        .douta(douta),
        .douta2(douta2),
        .douta3(douta3),
        .douta4(douta4),
        .douta5(douta5),
        .addra(addra)
        //.addra1(addra1)
        );
```

A.2 VGA display

```
module vga640x480(
        input wire dclk,                    //pixel clock: 25MHz
        input wire clr,                     //asynchronous reset
        output wire hsync,                  //horizontal sync out
        output wire vsync,                  //vertical sync out
        output reg red,//red vga output
        output reg green, //green vga output
        output reg blue,          //blue vga output
        input [2:0]douta,
        input [2:0]douta2,
        input [2:0]douta3,
        input [2:0]douta4,
        input [2:0]douta5,
        output reg [14:0]addra
        //output reg [13:0]addra1
//      output [2:0]LED
        );
//reg [11:0]j;
//parameter jlimit=4000;
parameter address=29999;
//parameter address2=16383;
```

```
// video structure constants
parameter hpixels = 800;// horizontal pixels per line
parameter vlines = 521; // vertical lines per frame
parameter hpulse = 96;          // hsync pulse length
parameter vpulse = 2; // vsync pulse length
parameter hbp = 144;  // end of horizontal back porch
parameter hfp = 784;  // beginning of horizontal front porch
parameter vbp = 31;             // end of vertical back porch
parameter vfp = 511;  // beginning of vertical front porch
// active horizontal video is therefore: 784 - 144 = 640
// active vertical video is therefore: 511 - 31 = 480

// registers for storing the horizontal & vertical counters
reg [9:0] hc;
reg [9:0] vc;

// Horizontal & vertical counters --
// this is how we keep track of where we are on the screen.
// ------------------------
// Sequential "always block", which is a block that is
// only triggered on signal transitions or "edges".
// posedge = rising edge  &  negedge = falling edge
```

```
// Assignment statements can only be used on type "reg" and need to be of the "non-blocking" type: <=
always @(posedge dclk or posedge clr)
begin
	// reset condition
	if (clr == 1)
	begin
		//addra<=17'b0;
		hc<= 0;
		vc<= 0;
	end
	else
	begin
		// keep counting until the end of the line
		if (hc < hpixels - 1)
			hc<= hc + 1;
		else
		// When we hit the end of the line, reset the horizontal
		// counter and increment the vertical counter.
		// If vertical counter is at the end of the frame, then
		// reset that one too.
		begin
```

```
                    hc<= 0;
                    if (vc < vlines - 1)
                        vc<= vc + 1;
                    else
                        vc<= 0;
                end

        end
end

// generate sync pulses (active low)
// ----------------
// "assign" statements are a quick way to
// give values to variables of type: wire
assign hsync = (hc < hpulse) ? 0:1;
assign vsync = (vc < vpulse) ? 0:1;

// display 100% saturation colorbars
// ------------------------
// Combinational "always block", which is a block that is
// triggered when anything in the "sensitivity list" changes.
// The asterisk implies that everything that is capable of triggering the block
```

```
// is automatically included in the sensitivty list.  In this case, it would be
// equivalent to the following: always @(hc, vc)
// Assignment statements can only be used on type "reg" and should be of the "blocking" type: =
always @(posedge dclk)
begin
        if((vc < vbp)&&(hc < hbp))
        begin
                addra<=15'b0;
                //addra1<=14'b0;
                //j<=12'b0;
        end
        // first check if we're within vertical active video range

//      else if(addra!=address)
//      begin
                // now display different colors every 80 pixels
                // while we're within the active horizontal range
                // -----------------
                // display white bar
        else if (vc >= vbp+10 && vc < (vbp+160))
                        begin
```

```
                                        if (hc >= hbp+100 && hc <
(hbp+300)&&(addra!=address))
                                        begin
                                                //{red,green,blue} = douta;
                                                addra<=addra+1'b1;

if(({red,green,blue}==3'b111)&&(douta == 3'b000))
{red,green,blue} = 3'b011;
else if(({red,green,blue}==3'b000)&&(douta3 == 3'b111))
{red,green,blue} = 3'b010;
else
{red,green,blue} = douta;
end
                                    else
                                        begin
                                            red = 0;
                                            green = 0;
                                            blue = 0;
                                        end
                                end
        // we're outside active vertical range so display black
                        else if(vc >= vbp+160 && vc < (vbp+170))
```

```
                    begin
                    red = 0;
                    green = 0;
                    blue = 0;
                    addra<=13'b0;
                    end
            else if (vc >= vbp+170 && vc < (vbp+320))
                begin
                    if (hc >= hbp+100 && hc <
(hbp+300)&&(addra!=address))
                    begin
                        addra<=addra+1'b1;

        //if((hc>=hbp+130)&&(hc<=hbp+150)&&({red,green,blue}==3'b111)&&(vc
>= vbp+130) && (vc < vbp+150))

        if(({red,green,blue}==3'b111)&&(douta3 == 3'b000))
                        {red,green,blue} = 3'b110;
                        else
if(({red,green,blue}==3'b000)&&(douta3 == 3'b000))
                        {red,green,blue} = 3'b011;
                        else
if(({red,green,blue}==3'b000)&&(douta3 == 3'b111))
                        {red,green,blue} = 3'b010;
```

```
                else
                {red,green,blue} = douta2;
                //end


            end
        else
            begin
                red = 0;
                green = 0;
                blue = 0;
            end
end


            else if(vc >= vbp+320 && vc < (vbp+330))
            begin
            red = 0;
            green = 0;
            blue = 0;
            addra<=13'b0;
            end
```

```
                else if (vc >= vbp+330 && vc < (vbp+480))
                        begin
                                if (hc >= hbp+100 && hc <
(hbp+300)&&(addra!=address))
                                begin
                                        //{red,green,blue} = douta;
                                        addra<=addra+1'b1;

        //if((hc>=hbp+130)&&(hc<=hbp+150)&&({red,green,blue}==3'b111)&&(vc
>= vbp+130) && (vc < vbp+150))

        if(({red,green,blue}==3'b111)&&(douta5 == 3'b000))
                                        {red,green,blue} = 3'b110;
                                        else
if(({red,green,blue}==3'b000)&&(douta5 == 3'b000))
                                        {red,green,blue} = 3'b011;
                                        else
if(({red,green,blue}==3'b000)&&(douta5 == 3'b111))
                                        {red,green,blue} = 3'b010;
                                        else
                                        {red,green,blue} = douta4;
                                        //end
```

```
                end
            else
                begin
                    red = 0;
                    green = 0;
                    blue = 0;
                end
        end

    else
      {red,green,blue}=3'b000;
    end
endmodule
```

A.3 Code for Image to RAW data Conversion

```
clc;

clf;

clear all;

hautothresh =
video.Autothresholder('ThresholdOutputPort',true,'ThresholdScaleFactor',1);

Image=imread('C:\Users\Maham\Desktop\paint.jpg');

ImageG=rgb2gray(Image);

z1 = step(hautothresh,ImageG);

%Gray=rgb2gray(Image);

%Image2=not(Image);

F=double(Image);

%save File3.txt -ascii F


dlmwrite('C:\Users\Maham\Desktop\test_file.txt',z1,'delimiter',' ');

k = dlmread('C:\Users\Maham\Desktop\test_file.txt');

imshow(k)
```

A.4 .coe File Generation in MATLAB

```
img = imread('C:\Users\Maham\Desktop\paint video\myim3.jpg');

height = size(img, 1);

width = size(img, 2);

img2=rgb2gray(img);

BW = im2bw(img2,0.7);

imshow(BW);


fid = fopen('C:\Users\Maham\Desktop\paint video\myim3.txt','w');

fprintf(fid,'MEMORY_INITIALIZATION_RADIX=2;\r\n');

 %fprintf(fid,'Coefficient_Width = 3;\r\n' );

fprintf(fid,'MEMORY_INITIALIZATION_VECTOR=\r\n');


for r=1:height

for c=1:width

      %cnt = cnt + 1;

      R = BW(r,c,1);

      G = BW(r,c,1);

      B = BW(r,c,1);


      y = [R;G;B];
```

```
fprintf(fid,'%X%X%X,\r\n',y);
      % fprintf(fid,',\n');
end
end

fclose(fid);
```

A.5 C code for MicroBlaze

```
#include <stdio.h>
#include "platform.h"
#include "xparameters.h"
#include "xutil.h"
#include "xuartlite_l.h"
void print(char *str);
int main()
{
   init_platform();
int uch;
int memory_word, value;
int count;
int* SDRAM_address = (int*)(XPAR_DDR_SDRAM_MPMC_BASEADDR);
print("I am in program \n\r");
```

```
    memory_word = 0;
    // reading from the com port
for(count = 0; count <4; count++)
    {
          uch = XUartLite_RecvByte(STDIN_BASEADDRESS);
             xil_printf("-- input value %c --\r\n", uch);
             // combining 4 digits in 32 bit word
             memory_word = (memory_word << 8)| (0xff & uch);
             xil_printf("-- input value %c --\r\n", memory_word);
    }
    xil_printf("-- input value %c --\r\n", memory_word);
    // putting 32 bit word in SDRAM
    *SDRAM_address = memory_word;
    // reading from SDRAM
value = *SDRAM_address;
    xil_printf("-- Value  = %c --\r\n", value);
print("End of program \n\r");
    cleanup_platform();
return 0;
}
```