ACKNOWLEDGEMENT

This humble endeavor bears the imprint of many people who assisted us for the accomplishment of the Final Year BE Project. I take this opportunity to express my heartfelt gratitude and appreciation to all those who have helped me directly or indirectly towards the successful completion of this project.

I express my most sincere and grateful acknowledgement to **PES Institute of Technology** for giving me an opportunity to pursue my Bachelor Degree in Electronics & Communication and thus guiding me for a bright future.

I would like to sincerely thank our project guide, **Dr.Vijay Holimath, Research Associate Professor of the Department of Electronics and Communication,PES Institute of Technology, Bangalore**, for his consistent valuable guidance, advice and persistent encouragement throughout the project work.

I am grateful to **Dr T S Chandar, the Head of the Department of Electronics and communication, PES Institute of Technology, Bangalore** for his encouragement and support in our endeavour. Last, but not the least, I would like to thank my parents and classmates for their constant support, motivation and amendments throughout the project.

ABSTRACT

Reconfigurable computing (RC) is an emerging viable computing option for applications that require high performance. RC systems are a combination of hardware/software data processing platforms that implement computationally intensive algorithms in Field Programmable Gate Array (FPGA) hardware devices. The use of Dynamic Partial Reconfiguration (DPR) in FPGA based systems is an attempt at combining the flexibility of software with the speed of Application Specific Integrated Circuits (ASICS). Consequently, it enables significant reduction of space, time and power consumption over conventional software only systems.

Automated Target Recognition (ATR) is one of the most popular naval applications which can be used to facilitate the safety and security measures. Here, the visual representations of the signal i.e. images are used for target recognition.

In this target recognition problem, there are two classes, namely, target and clutter. Principal Component Analysis (PCA) is used as the feature extractors for the images obtained from the sensor. The feature vectors so obtained for every image is passed as an input to a feedforward Artificial Neural Network (ANN) which is employed as a 2-class classifier.

The training code for PCA and ANN was written in Matlab. Testing code for the same is written in Xilinx C. The reconfigurable hardware design was synthesized and loaded to Virtex-5 XC5VFX70T board. The test results obtained from the hardware were validated with Matlab results. The implementation of the algorithms on the reconfigurable hardware was found to yield faster results.

THESIS ORGANIZATION

This section provides the brief structure and organization of this report. The overall project is divided in terms of detailed chapters for better understanding and interpretation. A brief description of what each chapter contains is shown below:

Chapter 1 describes Introduction about the project and Motivation for carrying it out.

Chapter 2 specifies the outcomes of the project that can be expected from the reconfigurable design and implementation of the image processing algorithms on the RC hardware.

Chapter 3 mainly illustrates the working principle of a Reconfigurable Hardware, Literature survey and previous work, scope of the present work, brief overview of this project and briefly discussed the advantages and disadvantages.

Chapter 4 of the report will describe the overall hardware embedded design, Floorplanning and the rest of the flow involved in the hardware design of the project. This chapter discusses in brief all the hardware modules that form a part of the application.

Chapter 5 enumerates the implementation of image processing algorithms on software and hardware platforms. Validation of the test results is done.

Chapter 6 includes conclusion, final outcome of the project, and future improvements that can be implemented to obtain better results.

Bibliography lists out the various books and documents that form the basic idea, & concept for the design, implementation and rest of the work carried out in this project.

Appendix I enumerate the various tools used for the design which is implemented on FPGA.

Appendix II enumerates the reports obtained during implementation.

ABBREVIATION

Word	**Abbreviation**
ASIC	Application Specific Integrated Circuits
RC	Reconfigurable Computing
FPGA	Field Programmable Gate Array
PR	Partial Reconfiguration
ATR	Automatic Target Recognition
EDK	Embedded Development Kit
XPS	Xilinx Platform Studio
SDK	Software Development Kit
PCA	Principal Component Analysis
ANN	Artificial Neural Network
PLB	Processor Local Bus
BRAM	Block Random Access Memory
ILMB	Instruction Local Memory Bus
DLMB	Data Local Memory Bus
JTAG	Joint Test Action Group
RM	Reconfigurable Module

Table 0.1: Abbreviation

Contents

List of Figures

List of Tables

Chapter 1
Introduction

1 Introduction

There are two primary methods in traditional computing for the execution of algorithms. The first is to use an Application Specific Integrated Circuit (ASIC), to perform the operations in hardware. Because these ASICs are designed specifically to perform a given computation, they are very fast and efficient when executing the exact computation for which they were designed. However, the circuit cannot be altered for any other computation. Microprocessors are a far more flexible solution. Processors execute a set of instructions to perform a computation. By changing the software instructions, the functionality of the system is altered without changing the hardware. However, the downside of this flexibility is that the performance suffers, and is far below that of an ASIC. The processor must read each instruction from memory, determine its meaning, and only then execute it. This results in a high execution overhead for each individual operation[1].

Reconfigurable computing (RC) is intended to fill the gap between hardware and software, achieving potentially much higher performance than software, while maintaining a higher level of flexibility than hardware. Typical RC systems yield 10X to 100X improvement in processing speed over conventional CPU-based "software- only" systems [2]. RC systems merge the advantages of ASICs and General purpose processors. Figure.1.1 represents the attributes of RC computing with respect to Processor and ASIC.

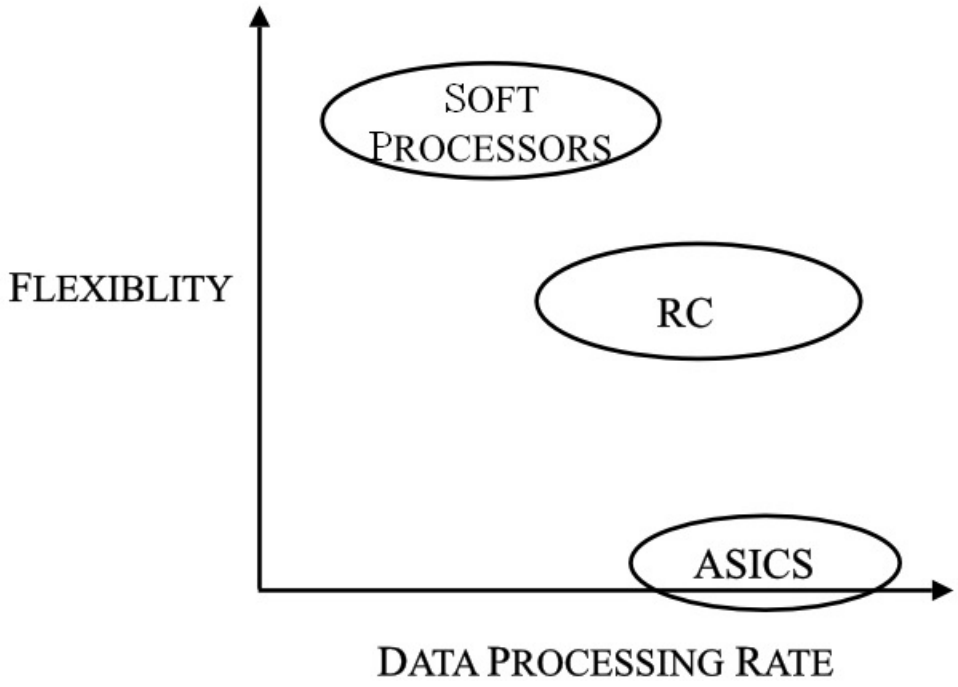

Figure 1.1: Flexibility vs Data-Processing rate

Partial Reconfiguration (PR) or Run-Time Reconfiguration (RTR) is implemented. It is defined as the ability to modify or change the functional configuration of the device during operation, through either hardware or software changes. PR is based upon the concept of virtual hardware, which is similar to virtual memory. The physical hardware is much smaller than the sum of the resources required by each of the configurations. Therefore, instead of reducing the number of configurations that are mapped, we instead swap them in and out of the actual hardware as they are needed[3].

To demonstrate the working of partially reconfigurable design, a naval application is developed.The

Naval Application involves a Target Recognition System which takes input image from a sensor and categorizes it as a target or a clutter. This is achieved using Principle Component Analysis(PCA) and Neural Network(NN)

1.1 Automatic Target Recognition

Automatic Target Recognition (ATR) is ability of an algorithm or device to recognize the targets based on the data received by the sensors. Target recognition can be done in various ways, namely audio representation of the signal, visual representation of the data etc. Here, the visual representations of the signal i.e. images are used for target recognition. In real-time scenarios, the images from Forward Looking Infrared (FLIR) sensor are used for target recognition. The datasets are generated using MATLAB, Photoshop and Picasa.

In recognition problems, there are two major steps, namely feature extraction and feature classification. Figure.1.2 shows the basic block diagram of a target recognition system.

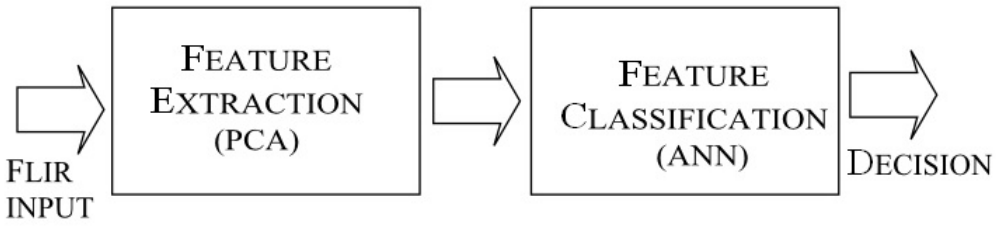

Figure 1.2: A basic block diagram for target recognition

Feature extraction is employed when the input data in an application is too large to be processed and is redundant. In this case the input data will be raw pixels. In order to obtain the reduced representation of the initial data, feature extraction is used. Hence, feature extraction is closely related to dimensionality reduction.

Principal Component Analysis is one of the most efficient techniques to achieve dimensionality reduction and get non-redundant representation of a large data. It uses an Information Theory approach wherein the most relevant image information is encoded in a group of images that will best distinguish every image. It transforms the target and clutter images in to a set of basis images, which essentially are the principal components of the images. The Principal Components (or Eigenvectors) basically seek directions in which it is more efficient to represent the data. This is particularly useful for reducing the computational effort.

Such an information theory approach will encode not only the local features but also the global features. When we find the principal components or the Eigenvectors of the image set, each Eigenvector has some contribution from each image used in the training set.

Every image in the training set can be represented as a weighted linear combination of these basis matrices. The number of Eigen images that we obtain therefore would be equal to the number of images in the training set. Let that number be M. Some of these Eigen images are more important in encoding the variation in images of the dataset, thus we could also approximate all images using only the K most significant Eigen images.

Classification is done by the 2-class neural network classifier. During the training, the neural network learns the weights and the bias terms from the training data, which will be the feature vectors of every

image. There are three layers, namely input layer, hidden layer and output layer.Figure.1.3 shows the basic architecture of the feedforward Artificial Neural Network (ANN).

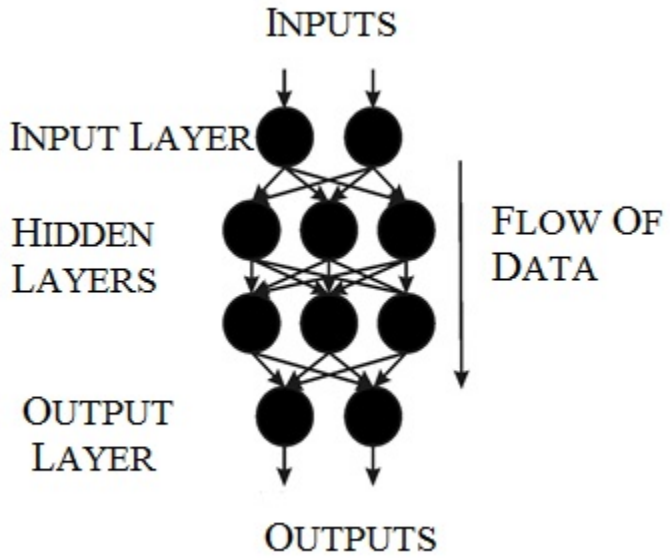

Figure 1.3: Basic Architecture of Artificial Neural Network

The circle in the above figure is the neuron which is the basic building block of a neural network.

The number of neurons can be varied based on recognition problem. The number of neurons in the input layer depends upon the dimension of the feature vector. The number of neurons in the hidden layer can be varied such that desired level of accuracy is achieved. The number of classes decides the number of neurons in the output layer.

1.2 Motivation

Nowadays, the demands for FPGA-based embedded systems with higher performance in terms of powerful computational ability and fast processing time are rising rapidly. Latest applications ported to embedded systems (e.g., pattern recognition, scalable video rendering, communication protocols) demand a large computation power, while must respect other critical embedded design constraints, such as, short time-to-market, low energy consumption or reduced implementation size. Increasing number of processors does not always translate to linear speedup because not all portions of the application can be parallelized. Here is where the dynamically loading and unloading modules (PR) at run time can be an alternative over the existing methods.

The speed of RC systems is much greater than conventional software systems. Another compelling advantage is reduced energy and power consumption. In a reconfigurable system, the circuitry is optimized for the application, such that the power consumption will tend to be much lower than that for a general-purpose processor. Various surveys report that moving critical software loops to reconfigurable hardware results in average energy savings of 35% to 70% with an average speed up of 3 to 7 times, depending on the particular device used. Other advantages of reconfigurable computing include a reduction in size and

component count (and hence cost), improved time-to-market, and improved flexibility and upgradability. These advantages are especially important for embedded applications [4].

Chapter 2
Project Deliverables

2 Project Deliverables

1. Embedded Design of reconfigurable hardware using EDK and Xilinx PlanAhead
   - Design with single memory module and controllers.
   - Design with two memory modules connected in parallel to achieve enhanced memory for the software application.
2. Implementation of PCA and ANN in MATLAB
   - The training and testing code written in MATLAB.
   - Trained weights and biases are saved in IEEE 754 format to text files.
3. Implementation of PCA and ANN in Xilinx C
   - Testing Code is written in Xilinx C using the reconfigurable hardware.
   - The accuracy of the results is verified with MATLAB results.

Chapter 3
Background

3 Background

This chapter begins with a more detailed description of Partial Reconfiguration. The underlying equations in the image processing algorithms are explained in this chapter.

3.1 Partial Reconfiguration

Partial Reconfiguration (PR) is modifying a subset of logic in an operating FPGA design by downloading a partial configuration file via Internal Configuration Access Port (ICAP)[1].

FPGA technology provides the flexibility of on-site programming and re-programming without going through re-fabrication with a modified design. PR takes this flexibility one step further, allowing the modification of an FPGA design during run-time by loading a partial configuration file, usually a partial BIT file. After a full BIT file configures the FPGA, partial BIT files can be downloaded to modify reconfigurable regions in the FPGA without compromising the integrity of the applications running on those parts of the device that are not being reconfigured.

Figure.3.1 illustrates the premise behind Partial Reconfiguration.

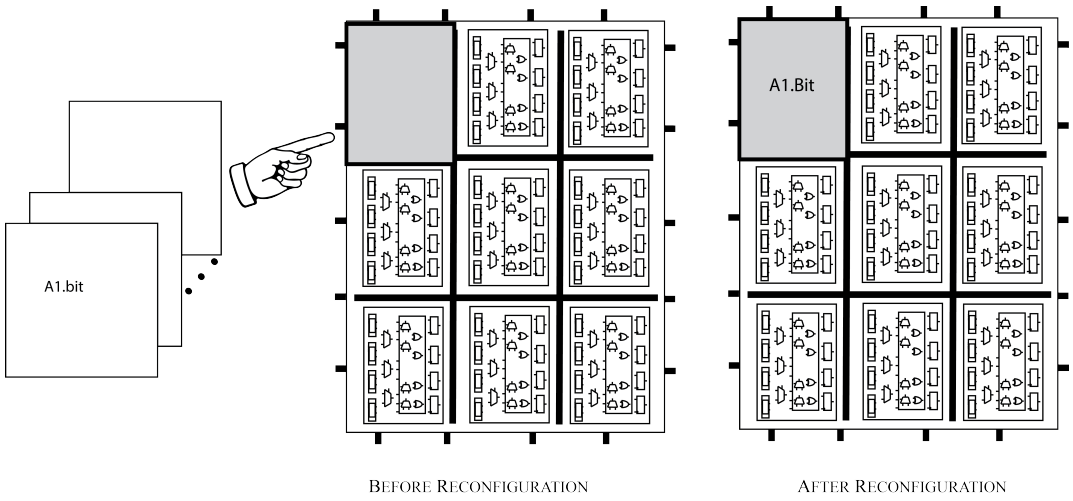

Figure 3.1: Basic Premise of Partial Reconfiguration

As shown, the function implemented in Reconfig Block A is modified by downloading one of several partial BIT files, A1.bit, A2.bit, A3.bit, or A4.bit. The logic in the FPGA design is divided into two different types, reconfigurable logic and static logic. The gray area of the FPGA block represents static logic and the block portion which is labeled Reconfig Block A represents reconfigurable logic.

The static logic remains functioning and is completely unaffected by the loading of a partial BIT file. The reconfigurable logic is replaced by the contents of the partial BIT file.

The basic premise of Partial Reconfiguration is that the FPGA hardware resources can be time-multiplexed similar to the ability of a microprocessor to switch tasks. Because the FPGA device is switching tasks in hardware, it has the benefit of both flexibility of a software implementation and the performance of a hardware implementation.

There are many reasons why the ability to time multiplex hardware dynamically on a single FPGA device is advantageous. These include:

- Reducing the size of the FPGA device required to implement a given function, with consequent

reductions in cost and power consumption.

- Providing flexibility in the choices of algorithms or protocols available to an application âĂć Enabling new techniques in design security.
- Improving FPGA fault tolerance.
- Accelerating configurable computing.

3.2 Principle Component Analysis

In order to implement target recognition, PCA[1]. is used for feature extraction. The algorithm can be enumerated as follows,

1. Obtain M training images, I_1,I_2... I_M It is very important that the images are mean-centered.
2. Represent each image I_i as a vector Γ_i as discussed below.

$$A = \begin{bmatrix} a_{11} & a_{12} & ... & a_{1N} \\ a_{21} & a_{22} & ... & a_{2N} \\ : & : & & : \\ a_{N1} & a_{N2} & ... & a_{NN} \end{bmatrix} \tag{3.1}$$

   This is concatenated to obtain Γ_i

$$\Gamma_i = \begin{bmatrix} a_{11} \\ : \\ a_{1N} \\ : \\ a_{2N} \\ : \\ a_{NN} \end{bmatrix} \tag{3.2}$$

3. Find the average image vector ψ
4. Subtract the mean image from each image vector Γ_i to get a set of vectors ϕ_i. The purpose of subtracting the mean image from each image vector is to be left with only the distinguishing features from each image and *removing* in a way information that is common.

$$\phi_i = \Gamma_i - \psi \tag{3.3}$$

5. Find the Covariance matrix C:

$$C = AA^{\mathrm{T}} \tag{3.4}$$

   where

$$A = \begin{bmatrix} \phi_1 & \phi_2 & ... & \phi_M \end{bmatrix} \tag{3.5}$$

   The Co-variance matrix has simply been made by putting one modified image vector obtained in one column each. C is a N^2 X M matrix

6. We then calculate the Eigen vectors u_i of C, C is a N^2 X N^2 matrix and it would return N^2 Eigen vectors each being N^2 dimensional. For an image this number is huge.

7. Consider the matrix A^TA instead of AA^T. A is a N^2 X M matrix, thus A^TA is a M X M matrix. If we find the Eigen vectors of this matrix, it would return M Eigen vectors, each of dimension M X *1*, these Eigenvectors are denoted as v_i. Now from some properties of matrices, it follows that:

$$u_i = Av_i \tag{3.6}$$

   This implies that using v_i, we can calculate the M largest Eigen vectors of AA^T. It is obvious that $M \ll N^2$ as M is simply the number of training images.

8. Find the best M Eigen vectors of $C = AA^T$ by using the relation discussed above. That is: $u = Av_i$.

9. Select the best K Eigen vectors, the selection of these Eigen vectors is done heuristically.

10. Now each image in the training set (minus the mean), ϕ ¡ can be represented as a linear combination of these Eigen vectors u_i:

$$\phi_i = \sum_{j=1}^{k} w_j u_j \tag{3.7}$$

    where, u_jś are Eigen Images. These weights can be calculated as :

$$w_j = u_j^T \phi_i \tag{3.8}$$

    Each normalized training image is represented in this basis as a vector:

$$A = \begin{bmatrix} w_1 \\ w_2 \\ : \\ w_k \end{bmatrix} \tag{3.9}$$

    where, i = 1,2...M. Such a vector corresponding to every image in the training set is calculated which are the features.

The above mentioned steps are implemented during the training session. After the eigen images are obtained, the testing is done. The steps implemented during testing session are as follows:

1. We normalize the incoming probe Γ as:

$$\phi = \Gamma - \psi \tag{3.10}$$

2. We then project this normalized probe onto the Eigen-space (the collection of Eigen vectors) and find out the weights.

$$w_i = u_i^T \phi \tag{3.11}$$

3. The normalized probe ϕ can then simply be represented as:

$$\Omega = \begin{bmatrix} w_1 \\ w_2 \\ : \\ w_k \end{bmatrix} \tag{3.12}$$

After the feature vector (weight vector) for the probe has been found out, we pass it as an input to the classifier.

3.3 Artificial Neural Network

After obtaining the features of the images, they are classified using an ANN[2.]. Neural networks are typically organized in layers. Layers are made up of a number of interconnected 'nodes' which contain an 'activation function'. Patterns are presented to the network via the 'input layer', which communicates to one or more 'hidden layers' where the actual processing is done via a system of weighted 'connections'. The hidden layers then link to an 'output layer' where the answer is output.

Most ANNs contain some form of *learning rule* which modifies the weights of the connections according to the input patterns that it is presented with. We make use of *delta rule* for backwards propagation of errors. With the delta rule, as with other types of backpropagation, *learning* is a supervised process that occurs with each cycle or *epoch* (i.e. each time the network is presented with a new input pattern) through a forward activation flow of outputs, and the backwards error propagation of weight adjustments. In other words, when a neural network is initially presented with a pattern it makes a random guess as to what it might be.

It then sees how far its answer was from the actual one and makes an appropriate adjustment to its connection weights. Also, within each hidden layer node is a hyperbolic tangent (tanh) activation function which polarizes network activity, adds non-linearity to the inputs and helps it to stabilize. Backpropagation performs a gradient descent within the solution's vector space towards a 'global minimum' along the steepest vector of the error surface. The global minimum is that theoretical solution with the lowest possible error. Learning Rate is multiplied by the error and then weights are updated. Learning rate helps the network to overcome obstacles (local minima) in the error surface and settle down at or near the global minimum.

Once a neural network is 'trained' to a satisfactory level it may be used as an analytical tool on other data. To do this, the user no longer specifies any training runs and instead allows the network to work in forward propagation mode only. New inputs are presented to the input pattern where they filter into and are processed by the middle layers as though training were taking place, however, at this point the output is retained and no back propagation occurs. The output of a forward propagation run is the predicted model for the data which can then be used for further analysis and interpretation.

The brief steps for implementation of a feed forward ANN are as follows:

- The weights are randomly initialized which will prevent the network to be stuck in local minima while updating weights.
- During forward propagation through a network, the output (activation) of a given node is a function of its inputs. The inputs to a node, which are simply the products of the output of preceding nodes with their associated weights, are summed and then passed through an activation function before

being sent out from the node.

$$S_j = \sum_i w_i j a_i \tag{3.13}$$

$$a_j = f(S_i) \tag{3.14}$$

where S_j is the sum of all relevant products of weights and outputs from the previous layer i, w_{ij} represents the relevant weights connecting layer i with layer j, a_i represents the activations of the nodes in the previous layer i, a_j is the activation of the node at hand, and f is the activation function.

In this application case tanh function is used as an activation function.

- The error function is commonly given as the sum of the squares of the differences between all target and actual node activations for the output layer. For a particular training pattern (i.e., training case), error is thus given by

$$E_p = \frac{1}{2} \sum_n (t_{jn} - a_{jn})^2 \tag{3.15}$$

where Ep is total error over the training pattern, $\frac{1}{2}$ is a value applied to simplify the function s derivative, n represents all output nodes for a given training pattern, t_{jn} represents the target value for node n in output layer j, and a_{jn} represents the actual activation for the same node.

Error over an entire set of training patterns (i.e., over one iteration, or epoch) is calculated by summing all Ep is given by

$$E = \sum_p E_p = \frac{1}{2} \sum_p \sum_n (t_{jn} - a_{jn})^2 \tag{3.16}$$

where E is total error, and p represents all training patterns.

- Gradient descent learning uses this error function for the modification of weights along the most direct path in weight-space to minimize error, change applied to a given weight is proportional to the negative of the derivative of the error with respect to that weight. The negative of the derivative of the error function is required in order to perform gradient descent learning.

- Backpropagation Algorithm is used to update the weights between input layer and hidden layer as well as weights between hidden layer and output layer.

- By using the chain rule,

$$\begin{aligned} \frac{\partial Error}{\partial weight\_hid\_out} &= \frac{\partial Error}{\partial output} * \frac{\partial Output}{\partial weight\_hid\_out} \\ &= error * hidden\_val \end{aligned} \tag{3.17}$$

$$\Delta weight\_hid\_out = \epsilon * error * hidden\_val \tag{3.18}$$

where, ϵ is the learning rate. A higher value for ϵ will necessarily result in a greater magnitude of change. Because each weight update can reduce error only slightly, many iterations are required in order to satisfactorily minimize error.

$$\begin{aligned}\frac{\partial Error}{\partial weight\_inp\_hid} &= \frac{\partial Error}{\partial outputh} * \frac{\partial outputh}{\partial inph} * \frac{\partial inph}{\partial weight\_in\_hid} \\ &= \frac{\partial Error}{\partial output} * \frac{\partial output}{\partial outputh} \\ &= error * weight\_hid\_out * (1 - hval^2) * input\end{aligned} \tag{3.19}$$

$$\Delta weight\_in\_hid = \epsilon * error * weight\_hid\_out * (1 - hval^2) * input \tag{3.20}$$

ϵ the learning rate. Backpropagation and updating of weights is done over large epochs and the error almost approaches zero. When the error in the network is negligibly small, then the network is said to have learnt the pattern from the training set. Once the neural network is trained, the weight matrix will be saved. Given a test image, recognition will be done using the weights and biases learnt during training. The following is done during testing,

- During testing, only forward propagation is implemented. The following equations are implemented.

$$S_j = \sum_i w_i j a_i$$

$$a_j = f(S_i)$$

- Based on the magnitude of the final result, the classification is done.

3.4 Literature Survey

[1] Manhwee Jo, V.K.Prasad Arava, Hoonmo Yang and Kiyoung Choi, Implementation of Floating/-Point Operations for 3D Graphics on a Coarse-Grained Reconfigurable Architecture In this paper the author presents how we can perform various floating-point operations on a coarse grained reconfigurable array of integer processing elements. They also demonstrate the effectiveness of their approach through the implementation of various floating-point operations for 3D graphics and give a glimpse on the performance analysis as well. The basic idea is the use of multiple Processing Elements in the array to perform single floating point operation. In order to achieve this, they propose a method to extend the design of each Processing Element without a significant increase in cost.

[2] E.Manikandan and K.A.Karthigeyan, Design of Parallel Vector/Scalar Floating Point Co-Processor For Reconfigurable Architecture In the existing FPGA soft processor systems, we make use of dedicated hardware modules to speed up parallel applications. In this paper the authors explain about the alternative approach of using a soft vector processor as a general purpose accelerator. To achieve this an autonomous Floating Point Vector Co-processor (FPVC) is implemented that works independently in an embedded system. This is a four stage RISC pipeline that supports single-precision and 32-bit integer arithmetic operations. They also show that FPVC is easier to implement at the cost of decrease in performance compared to the custom data path.

[3] Ali Azarian and Mahmood Ahmadi, Reconfigurable Computing Architecture In this survey, the authors give us a glimpse of an overview of programming logics and Configurable Logic Block (CLB) and Look Up Table (LUT) as logic elements. They also introduce us to reconfigurable computing models like static and dynamic, single and multi-context and partial reconfiguration architectures. Ali and

Mahmood define Reconfigurable Computing as the process of changing the structure of a reconfigurable device at star-uptime respectively at run-time and involves the use of reconfigurable devices, such as Field Programmable Gate Arrays (FPGAs), for computing purposes. Later in their work, they explain the principle involved in Static and Dynamic Reconfiguration. The Static one involves one time configuration followed by multiple execution, but the Dynamic one involves reconfiguration after an execution cycle as well. They also gave a basic idea of Look Up Table Computation and the need for dedicated Computational Blocks an described common interconnect strategies.

[4] Pierre Baldi and Kurt Hornik, Neural Networks and Principal Component Analysis: Learning from Examples without Local Minima Considering the problem of learning from examples in layered linear feed-forward neural networks using optimization methods such as Back propagation algorithm, the author shows that there is a unique minimum corresponding to the projection onto the subspace generated by the first principal vectors of a covariance matrix associated with the training patterns. Neural networks can be viewed as circuits of highly interconnected units with modifiable interconnection weights. They can be classified, for instance, according to their architecture, algorithm for adjusting the weights, and the type of units used in the circuit. The network consists of n input units, p hidden units and n output units. In addition to its simplicity, error back-propagation can be applied to nonlinear networks and to a variety of problems without having any detailed a priori knowledge of their structure or of the mathematical properties of the optimal solutions.

Chapter 4
Hardware Architecture

4 Hardware Architecture

This chapter describes the hardware architecture which is designed to implement the target recognition algorithms. The hardware cores and peripherals which form a part of the design are explained in detail.

The feature extraction and classification algorithms are run on XC5VFX70T . The FPGA belongs to the Virtex-5 family.

Xilinx Platform Studio (XPS) is used to configure and build the hardware specification of the embedded system.It provides an integrated environment for creating the software and hardware specification flows for an Embedded Processor system. It also provides a graphical system editor for connection of processors, peripherals and buses.

4.1 Hardware modules of the embedded design

4.1.1 Processor

The FPGA board supports two microprocessors. They are Microblaze (soft-core microprocessor) and PowerPC(hard-core embedded microprocessor).

We have used MicroBlaze[2] in this embedded application, since we tailor our project to specific needs (i.e.: Flash, UART, General Purpose Input/Output peripherals and etc.). As a soft-core processor, MicroBlaze is implemented entirely in the general-purpose memory and logic fabric of Xilinx FPGAs.The Micro Blaze processor is a 32-bit Harvard Reduced Instruction Set Computer (RISC) architecture optimized for implementation in Xilinx FPGAs with separate 32-bit instruction and data buses running at full speed to execute programs and access data from both on-chip and external memory at the same time. MicroBlaze is a load/store type of processor; it can only load/store data from/to memory. It cannot do any operations on data in memory directly; instead the data in memory must be brought inside the MicroBlaze processor and placed into the general-purpose registers to do any operations.

4.1.2 Primary I/O Bus

Processor Local Bus (PLB)[2] is the I/O bus which is used in the design. The Xilinx 128-bit PLB provides bus infrastructure for connecting an optional number of PLB masters and slaves into an overall PLB system. In this design there are 2 PLB hosts and 5 PLB slaves.

4.1.3 Memory Module

Block Random Access Memory (BRAM) is used to store the instructions, data .The stack and heap memory is The BRAM Block[3] is a configurable memory module which attaches to a variety of BRAM Interface Controllers. Both Port A and Port B of the memory block can be connected to independent BRAM Interface Controllers: LMB (Local Memory Bus), PLB (Processor Local Bus), and OCM (On-Chip Memory).

The size of the Memory module depends upon the size of the BRAM interface controllers attached to the Port A and Port B of the BRAM. Local memory size is 64KB in this design. The default size of the controllers are 64KB.We have increased the size of the controllers to 128KB so that memory is sufficient for the code, data, heap and stack. Thus, 128KB of total memory is available.

The default stack and heap sizes are 1KB each. Since operations are performed on an image, a larger heap will be required. Also, a large number of function calls will be required to perform pixel-by-pixel

operation. In order to accommodate all the requirements the heap and stack sizes are increased to 16KB each.

4.1.4 BRAM Interface Controllers

The LMB BRAM Interface Controller[4] is the interface between the LMB and the bram_block peripheral. There are two interface controllers connected to the Port A and Port B of the BRAM block. PORT A is connected to Instruction Local Memory Bus (ILMB) Interface controller. PORT B is connected to Data Local Memory Bus (DLMB) Interface controller. Both the controllers are PLB masters.

4.1.5 Local Memory Bus

LMB[5.] is used as an interconnect for Xilinx FPGA-based embedded processor systems. The LMB is a fast, local bus for connecting the MicroBlaze processor instruction and data ports to high-speed peripherals, primarily on-chip BRAM. Since the microprocessor has a Harvard Architecture there are separate buses for data and instructions.ILMB is an interconnect between ILMB interface controller and bram block.DLMB is an interconnect between DLMB interface controller and bram block. The width of both the buses is 32 bits. It is a single master bus.

4.1.6 Debugger

Microblaze Debug Module (MDM)[6.] enables JTAG-based debugging of one or more microblaze processors. JTAG specifies the use of a dedicated debug port implementing a serial communications interface without requiring direct external access to the system address and data buses. MDM supports debugging upto 8 microblaze processors.

MDM is a slave and hence is connected to Slave Processor Local Bus (SPLB).

4.1.7 Communication Peripheral

RS232-UART (Universal Asynchronous Receiver Transmitter) is used for Serial Communication. XPS-UART Lite Interface[7.] connects to the PLB and provides controller interface for Asynchronous data transfer. It supports 8 bit interfaces. It has configurable baud rate. The baud rate of 115200 is used. XPS-UART Lite performs parallel to serial conversion on characters received through PLB and serial to parallel conversion on characters received on the serial peripheral. It supports full-duplex communication. The peripheral acts a slave. Therefore, it is connected to SPLB.

4.1.8 Flash Memory Controller

The XPS System ACE Interface Controller (XPS SYSACE)[8.] is the interface between the PLB and the microprocessor Interface (MPU) of the System ACE Compact Flash solution peripheral. It is connected as a 32-bit Slave on PLB buses. Flash memory is used to store the partial bit files, system.ace file, text files which contain the test image, Eigen images for feature extraction, weight matrix for neural network.

4.1.9 XPS HWICAP Controller

The XPS HWICAP (Hardware ICAP) IP [9.]enables an embedded microprocessor to read and write the FPGA configuration memory through the Internal Configuration Access Port (ICAP) at run time. Using ICAP we can write software programs for an embedded processor that modifies the circuit structure and

functionality during the circuit's operation. The XPS HWICAP includes support for resource reading and modification of the CLB LUTs and Flip-Flops. The XPS HWICAP controller provides the interface necessary to transfer bit streams to and from the ICAP. It is connected to SPLB as it is a slave.

4.1.10 User-Defined Peripheral

There is one user-defined peripheral where instantiation of a module is done without netlist, inputs and outputs of the reconfigurable module are declared. Since there are six inputs to the reconfigurable module, six software accessible registers are used. The peripheral is connected to SPLB.Once the peripheral is generated, it is imported to the XPS design later.

After adding all the required IPs to the XPS design, the design is synthesized and netlist is generated .This is a top-level netlist(.ngc file) which is an input to the Xilinx PlanAhead tool.The XPS also generates User Constraints File(.ucf) and Block Memory map(.bmm) for the application. NGC files are specific netlist files which contain both logical design data and constraints. It is a translation of the VHDL/ Verilog design file into gates optimized for the target architecture. UCF files are used for timing, placement and pinout constraints.

BMM file is a text file that has syntactic descriptions of how individual block RAMs constitutes a contiguous logical data space.

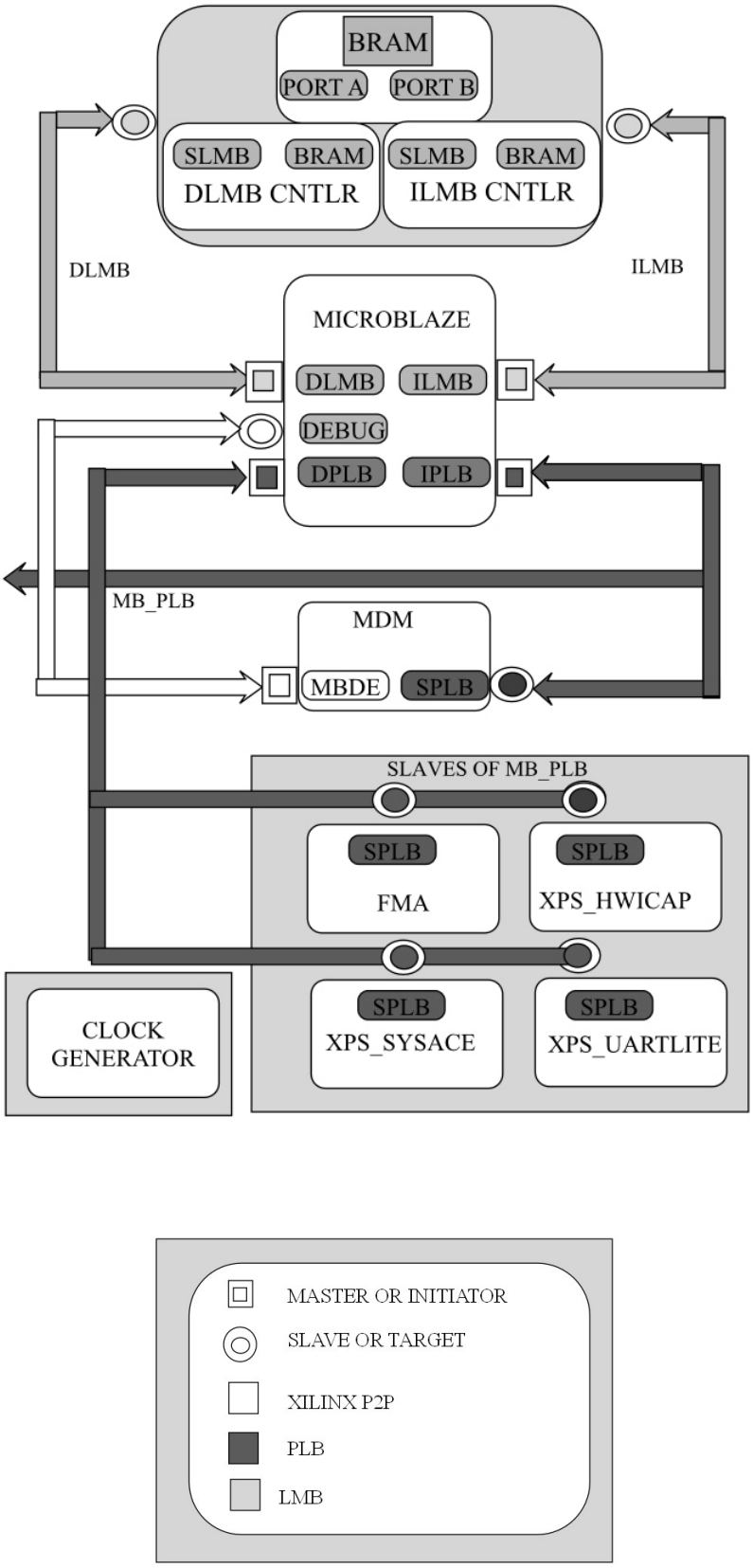

Figure 4.1: Hardware embedded design block diagram of the application

Instance	Base Name	Base Address	High Address	Size	Bus Interface	Bus Name
dlmb_cntlr	C_BASEADDR	0X00000000	0X0001FFFF	128K	SLMB	dlmb
ilmb_cntlr	C_BASEADDR	0X00000000	0X0001FFFF	128K	SLMB	ilmb
SYSACE Compact Flash	C_BASEADDR	0X83600000	0X8360FFFF	64K	SPLB	mb_plb
RS232_Uart_1	C_BASEADDR	0X84000000	0X8400FFFF	64K	SPLB	mb_plb
mdm_0	C_BASEADD	0X84400000	0X8440FFFF	64K	SPLB	mb_plb
xps_hwicap_0	C_BASEADDR	0X86800000	0X8680FFFF	64K	SPLB	mb_plb
fma_0	C_BASEADDR	0XCAA00000	0XCAA0FFFF	64K	SPLB	mb_plb

Table 4.1: Address Map for hardware modules

In this hardware design the upper bound on stack and heap is 16KB.If the user attempts to increase it further, then the instructions and data no longer occupy contiguous sections and therefore an executable is not created. For a highly computationally intensive algorithm, user might require a higher stack and heap sizes. To enable it, the hardware design is modified accordingly. The following amendments are made to the previous embedded design :

1. Along with the previous BRAM block one more BRAM block of 128KB is added to the design. Hence, we get to use 256 KB of memory. Now 128KB of BRAM is used to store code and data. Another 128KB BRAM block is used for the stack and heap memory. The stack and heap memory is 32 KB size each which caters running of a computationally expensive algorithm on the FPGA.
2. The BRAM interface controllers (DLMB controller and ILMB controller) are added to the design. Port connections of the BRAM block have to be made accordingly.
3. Bus interfaces for the controllers are ILMB and DLMB buses respectively.
4. Address Memory Map is modified and addresses are generated for the hardware modules.

Now, the design has two BRAM modules in parallel as shown in Figure.4.2.

4.2 Reconfigurable Modules(RMs)

There are two reconfigurable modules in the design which are used to implement the computations involved in the algorithms. The arithmetic operations which are performed during the testing session are performed by these reconfigurable modules based on which partial bit stream is loaded on to the reconfigurable partition created during the Floorplanning in Xilinx PlanAhead Tool.

The first RM performs single precision add-fused multiplication and the second RM performs double precision add-fused multiplication. Single Precision multiplier takes 2 32 bit inputs and gives another 32 bit number as an output. Double Precision multiplier takes 2 64 bit inputs and gives another 64 bit number as an output.

The format of a single precision number is as shown in Figure.4.3.

During single precision multiplication, the sign bits of both the inputs are XORed.The exponent 8 bits (with bias of 127 added) of both the inputs are added and mantissa is multiplied. Bias is added to

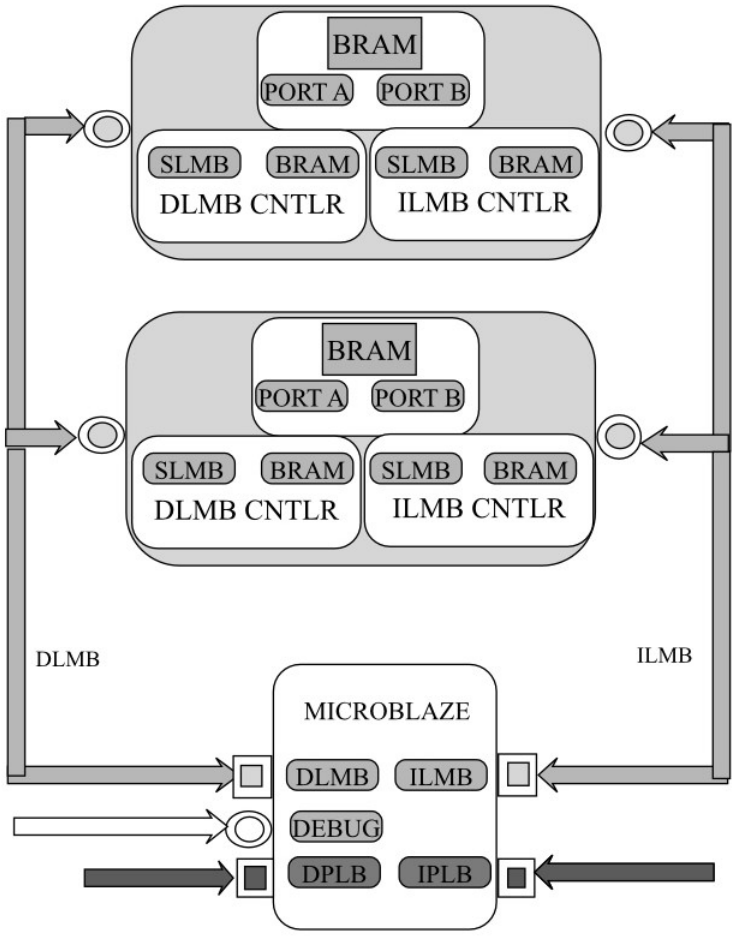

Figure 4.2: Block Diagram with two memory modules and interface controllers

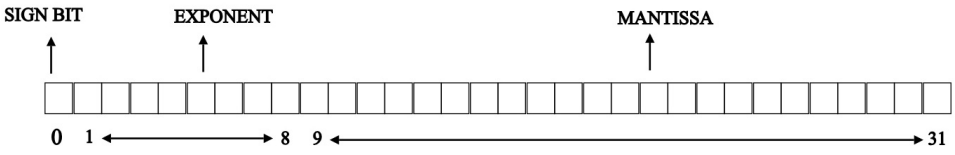

Figure 4.3: IEEE 754 Single Precision Format

restrict the numbers from -127 to 127 between 0 to 255 so that the exponent can be represented as an 8 bit unsigned number. While interpretation of the results the bias terms should be subtracted to get the correct decimal representation of the number.

The format of a double precision number is as shown in Figure.4.4

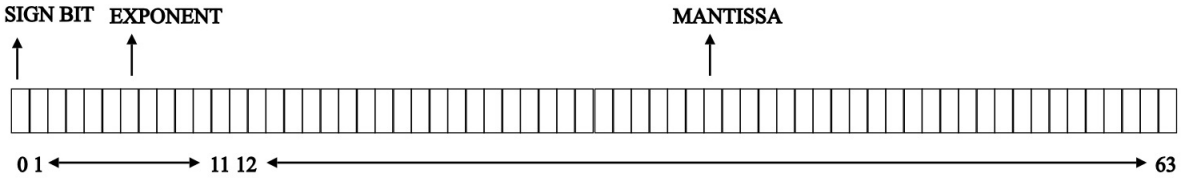

Figure 4.4: IEEE 754 Double Precision Format

Similar operations are performed for a double precision multiplier except for a different bias value which is 1023 to make sure that the exponent value can be represented as an unsigned 11 bit number.The block diagram of the reconfigurable module is shown in Figure.4.5.

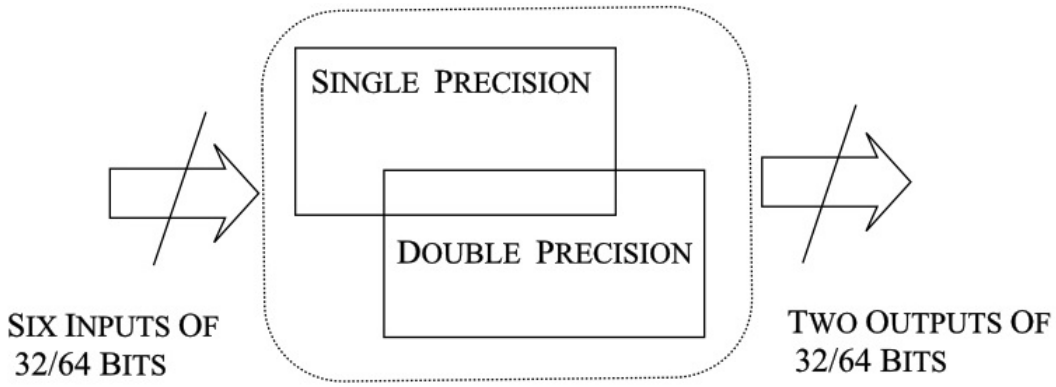

Figure 4.5: Block Diagram of Reconfigurable Module

Based on RM which is loaded into the reconfigurable region the width of the inputs vary. If Single precision partial bitstream is loaded, then the inputs are 32 bits wide and vice-versa.

The operation performed by the RMs is :

$$A \pm B * C \tag{4.1}$$

The general purpose registers in MicroBlaze processor are 32 bit wide. Therefore operation can be performed only at 32 bit data at once. The 64 bit input data is split into two 32 bit data and given as input to the RMs. So, the bifurcation is done as follows,

Addition, Subtraction and Multiplication can be performed by the RMs.The values of the operands have to be so adjusted that the desired operation is performed by the module.

Table 2 lists the values of the operands to perform the desired operation by a single precision RM.

The modules are written in Verilog. The Verilog code is synthesized and .ngc files are generated for both the modules which is the netlist to the user defined peripheral which was added in the XPS.

Operands	Single Precision	Double Precision
First operand	A[31:0]	A[31:0]
Second operand	0	A[63:32]
Third operand	B[31:0]	B[31:0]
Fourth operand	0	B[63:32]
Fifth operand	C[31:0]	C[31:0]
Sixth operand	0	C[63:32]

Table 4.2: Inputs to the Reconfigurable Modules

Operands	Addition	Subtraction	Multiplication
First operand	A[31:0]	A[31:0]	0
Second operand	0	0	0
Third operand	B[31:0]	B[31:0]	B[31:0]
Fourth operand	0	0	0
Fifth operand	3F800000	3F800000	C[31:0]
Sixth operand	0	0	0

Table 4.3: Values of operands for various operations to a single precision RM

4.3 Xilinx PlanAhead

Xilinx PlanAhead 12.4 is a tool for:

- I/O pin planning
- Floorplan Area/Logic/IO
- Analyse Timing/Floorplan Design
- Run DRC
- The input to the tool will be .ngc file and .bmm file of the top level module.
- The output will be the full and partial bitstreams of the design.

Brief Procedure

1. A new PlanAhead with PR enabled has to be created.
2. The .ngc file which is generated by the XPS will be the input to the Xilinx PlanAhead. This top-level .ngc file will have a module which includes the inputs and outputs for reconfigurable modules and all other hardware modules which were added in the XPS.
3. A reconfigurable partition is created and the two reconfigurable modules, namely single and double precision add-fused multipliers added along with their netlists.
4. A reconfigurable partition will be created on left half side of the board. Only Slices will be selected as the modules do not require DSP slices and BRAM memory.
5. For memory mapping, the .bmm file path will be given and double precision module is run first .The static and partial logic both are mapped, placed and routed.

Operands	**Addition**	**Subtraction**	**Multiplication**
First operand	A[31:0]	A[31:0]	0
Second operand	A[32:64]	A[32:64]	0
Third operand	B[31:0]	B[31:0]	B[31:0]
Fourth operand	B[32:64]	B[32:64]	B[32:64]
Fifth operand	00000000	00000000	C[31:0]
Sixth operand	3FF00000	3FF00000	C[32:64]

Table 4.4: Values of operands for various operations to a double precision RM

6. For the other module static logic is imported from the double module and partial logic is implemented .Later, single module is mapped, placed and routed.

7. The partial and full bit files are generated .

8. The download.bit file is generated by giving the path of the .bmm file, executable file and bit file of double module using data2mem.

9. The system.ace file is generated which will be loaded onto the FPGA which contains the download.bit file and JTAG settings.

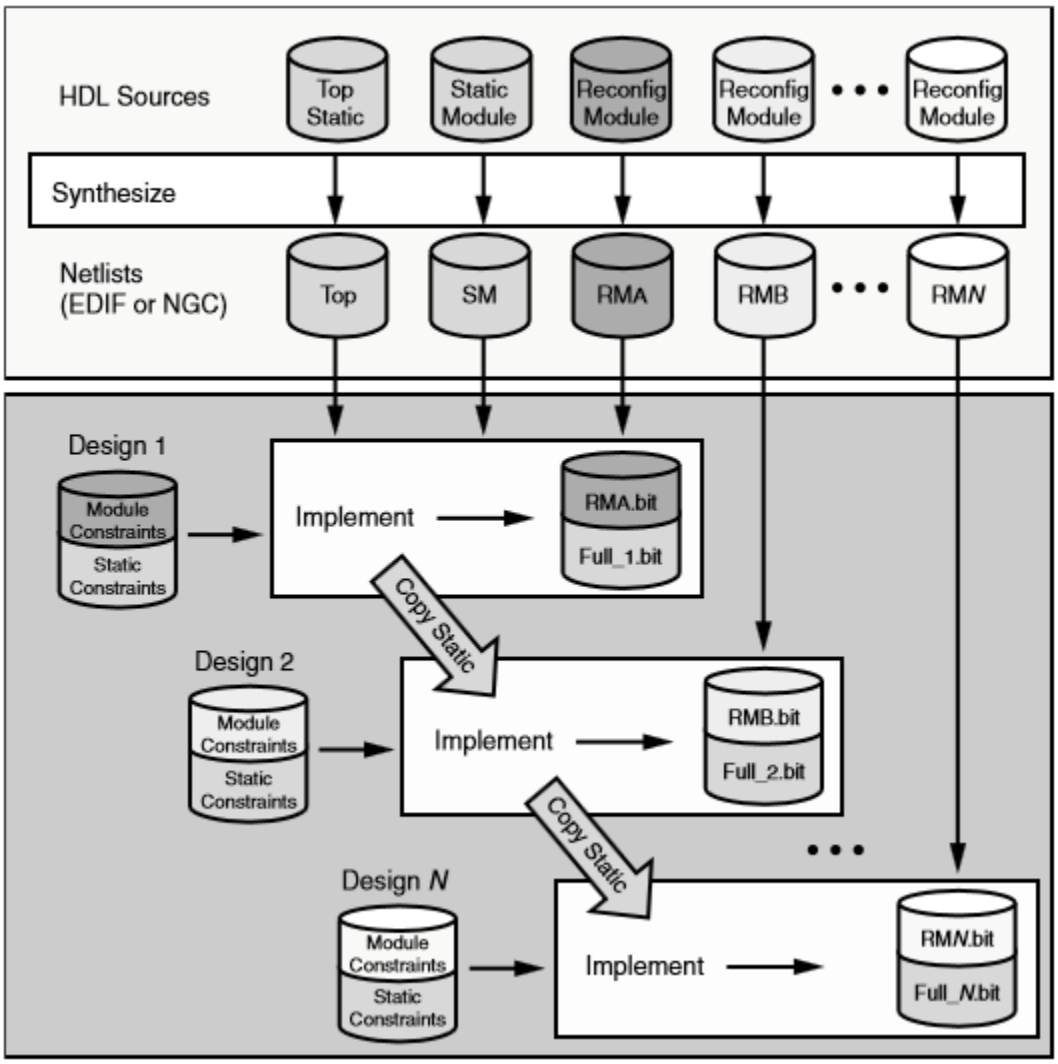

Figure 4.6: Overview of the Partial Reconfiguration Software Flow

The top gray box represents the synthesis of HDL source to netlists for each module. The appropriate netlists are implemented in each design to generate the full and partial BIT files for that configuration. The static logic from the first implementation is shared among all subsequent design implementations[5].

Figure.4.7 shows the reconfigurable partition created during Floorplanning.

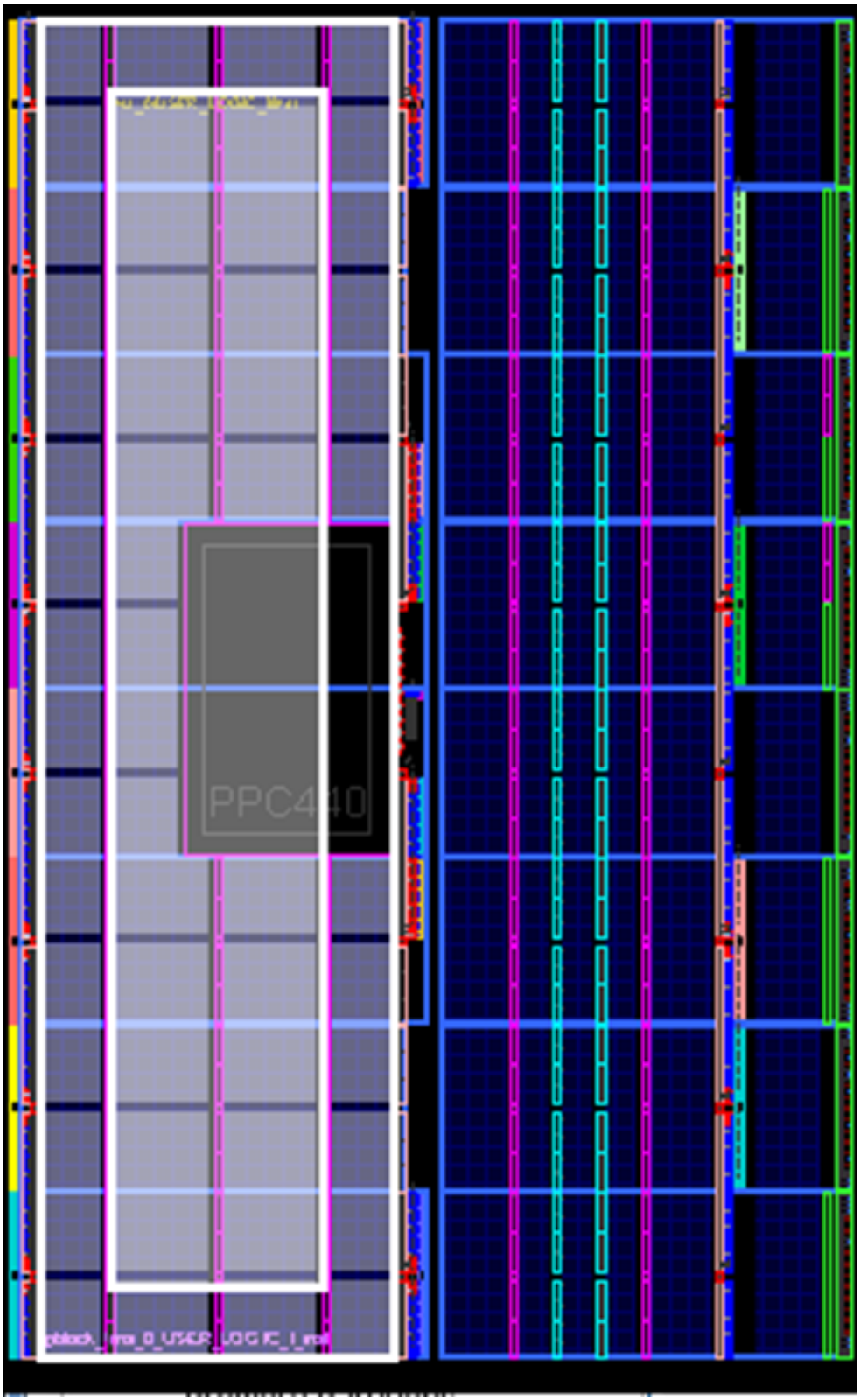

Figure 4.7: Reconfigurable Partition

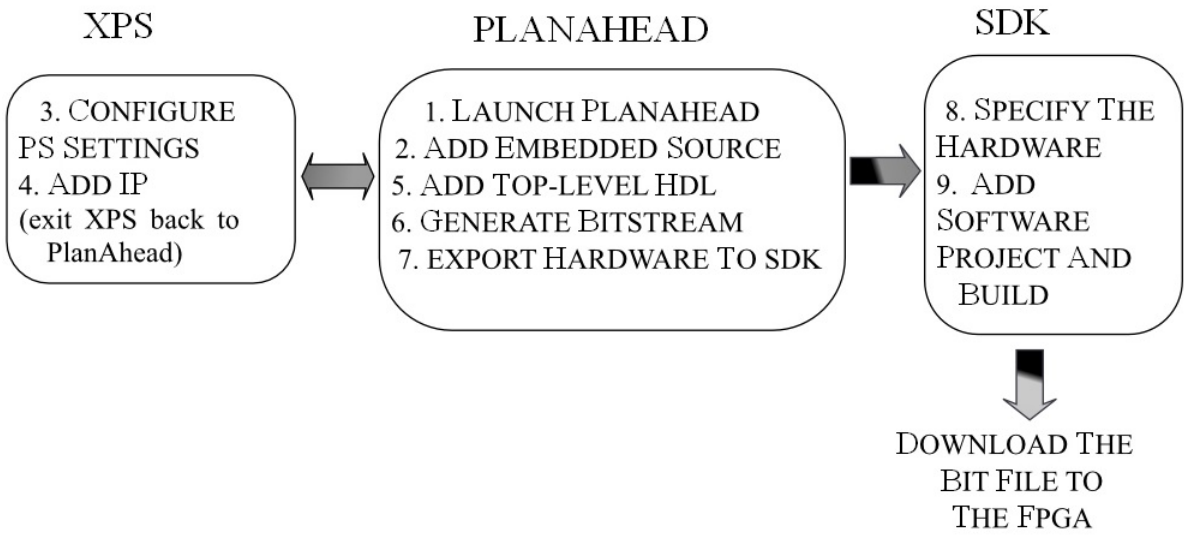

Figure 4.8: Hardware flow of the project

The Xilinx Platform studio is used for selecting the type of processor, type of interconnect system, memory modules, type of controllers, communication peripherals, etc. The wrapper function for all the modules is provided by XPS which is later used in PlanAhead for checking the area and timing constraints. Once the top-level module is generated,PlanAhead is used for mapping, place and route of the design. Bitstream generation is done. SDK is launched ,all required libraries are included. The stack and heap sizes are changed as per requirement. Finally an .elf file is generated for the Xilinx C code. Finally the partial bit files and system.ace file is loaded to the Flash memory. Output is tested on the hyperterminal.

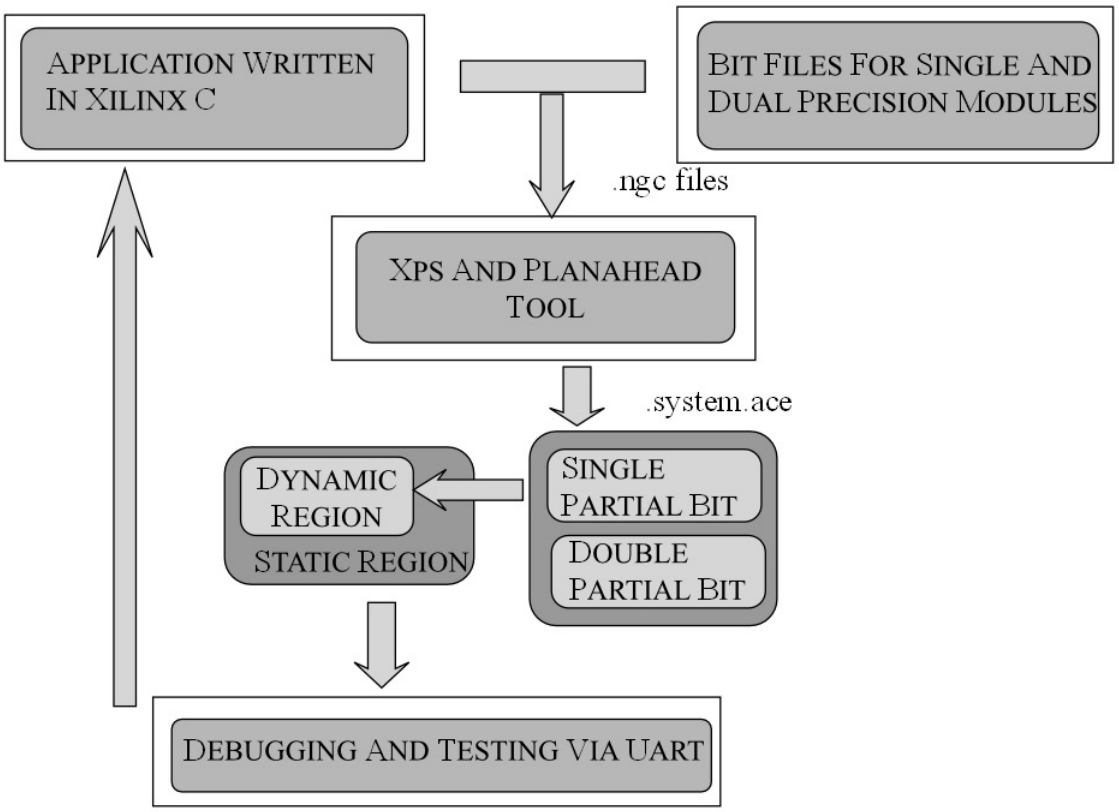

Figure 4.9: System Block Diagram

In the system block diagram, the .ngc files of the single and double precision modules which are the reconfigurable modules,the top level .ngc file(obtained from XPS) are given as an input to the PlanAhead. Executable is generated from the software which is written in Xilinx C. By making use of the hardware specified by the PlanAhead and the executable generated by the SDK system.ace file is generated. Once the bit files and system.ace files are loaded to the FPGA.,the dynamic logic keeps changing based on which partial bit is loaded via ICAP whereas the static logic keeps functioning throughout the program. Debugging and testing is done via UART on the HyperTerminal. In case of errors, the changes are made in the software and the process is repeated until the desired output is obtained.

Chapter 5
Software Design and Results

5 Software Design and Results

5.1 Training dataset and test dataset

The images are assumed to be taken from Forward Looking Infra-red (FLIR) sensor. The dataset is generated using Photoshop and Picasa by applying appropriate filters to a RGB image. The size of the training set is 200 with 100 images belonging to each class namely, clutter and target. The test set has some 30 images.

Figure.5.1 shows the sample target and clutter image which are used in for target recognition.

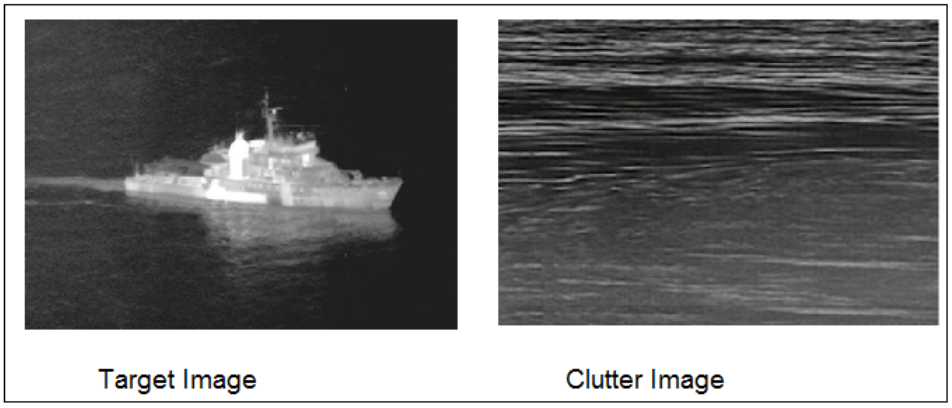

Figure 5.1: Sample Target and Clutter Image

In order to prevail over the memory constraints of the hardware the image resolution is chosen to be 20 X 20.Before doing any computation on the pixel values, the images will be resized to 20 X 20.

5.2 Training

Training is done using MATLAB. The feature extraction is done on the training set and the required results are stored into a text file which will be used during the test session. The mean and standard deviation of the PCA feature is noted. Every feature vector is normalized accordingly.

The neural network is trained using the feature vectors. The weights are stored in the text file which is used during testing scenario.

Figure.5.2 shows the block diagram of feature extraction using PCA for training dataset.

The above figure is the block diagram of feature extraction which is implemented in MATLAB. The following results are stored into Flash memory for testing:

- Test Image
- Mean Image of the training set
- Eigen Images

The reconfigurable modules assume the input to be in IEEE 754 hexadecimal format. Therefore the input to the reconfigurable modules is converted from decimal format to IEEE 754 hexadecimal format. The feature extraction process deals with pixel-by-pixel computations. In order to make the calculations more efficient single precision arithmetic is employed. The inputs are converted into single precision

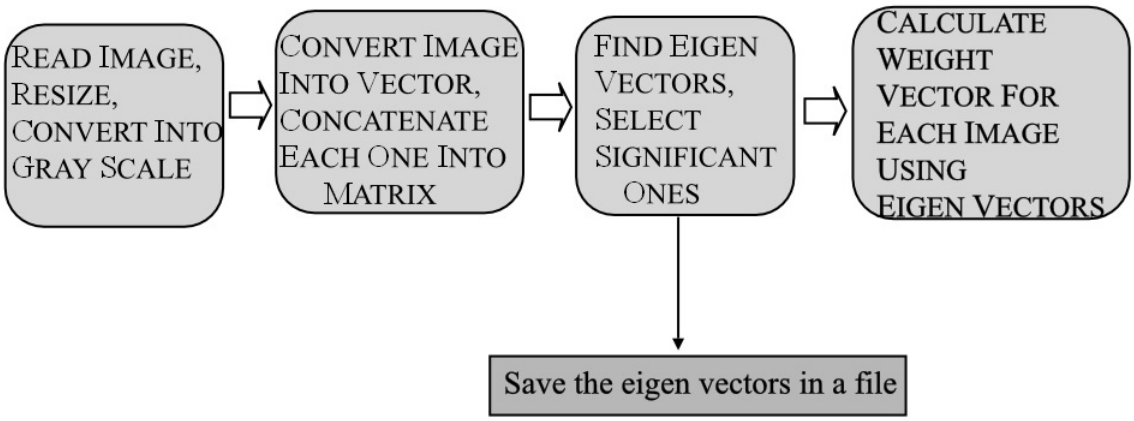

Figure 5.2: Feature Extraction using PCA for training

format and single precision RM is loaded. The inputs are passed to the single precision RM according to Table 2. Figure.5.3 shows the block diagram for pre-processing of inputs to the single precision RM.

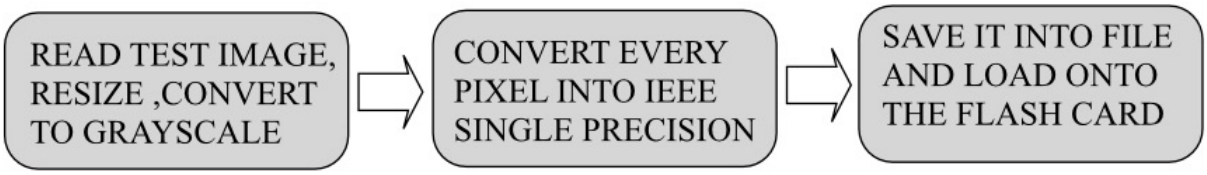

Figure 5.3: Pre-processing of the test image

Similarly, the mean image is converted into IEEE hexadecimal format and saved into a file. The Eigen images are also converted into the same desired format and saved into a separate file which will be used later.

The mean and standard deviation of the feature vectors is found and noted down which will be used during testing.

Once the feature vectors are obtained and desired results are stored into a file, classification is done using feedforward artificial neural network. Figure.5.4 depicts the overall block diagram of classification process during training.

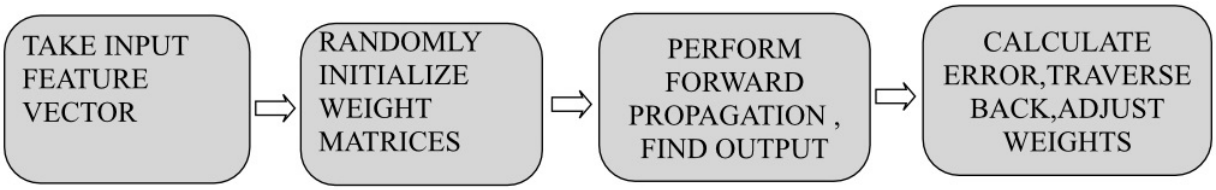

Figure 5.4: Training the neural network for classification

In order to achieve better accuracy without the loss in the performance, double precision arithmetic is employed for classification. Therefore, the weight matrix is converted into IEEE double precision format and stored into a file. After training , there will be four .txt files with

5.3 Testing

Figure.5.5 shows the block diagram for feature extraction of a test image on the board.

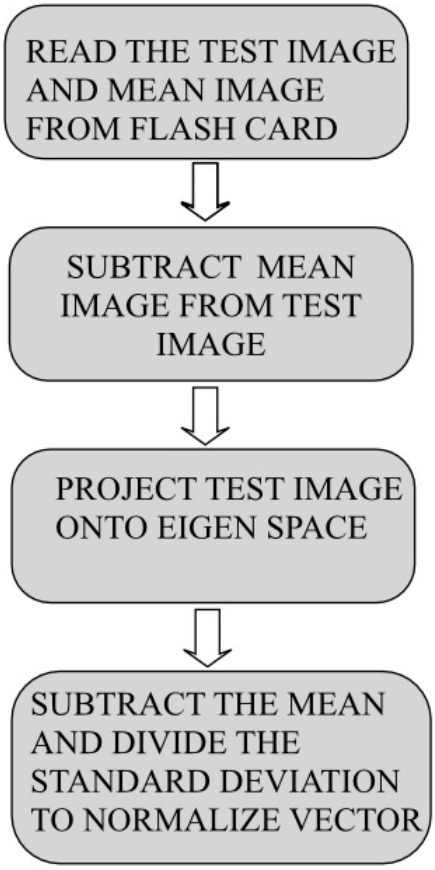

Figure 5.5: Feature Extraction on board

The subtraction and the projection of the image onto eigen space is done by the Single precision RM.

The single precision partial bit file will be loaded beforehand .Later the operations are performed.

Figure.5.6 depicts the classification of test image on the board.

The matrix multiplication which is carried out between the various layers of the neural network performed by double precision RM and every input being in IEEE double precision format. The inputs to the double precision RM is passed according to Table 3. The decimal output which is either positive or negative is displayed on the HyperTerminal. If the output is positive then it is target or vice-versa.

The testing is done on board as well as MATLAB so that the test results shown by the board can be validated.

There are possibilities of memory bloating where the results are corrupt therefore a proper balance has to be maintained between the stack and heap variables so that the program uses the allocated stack and heap memory efficiently.

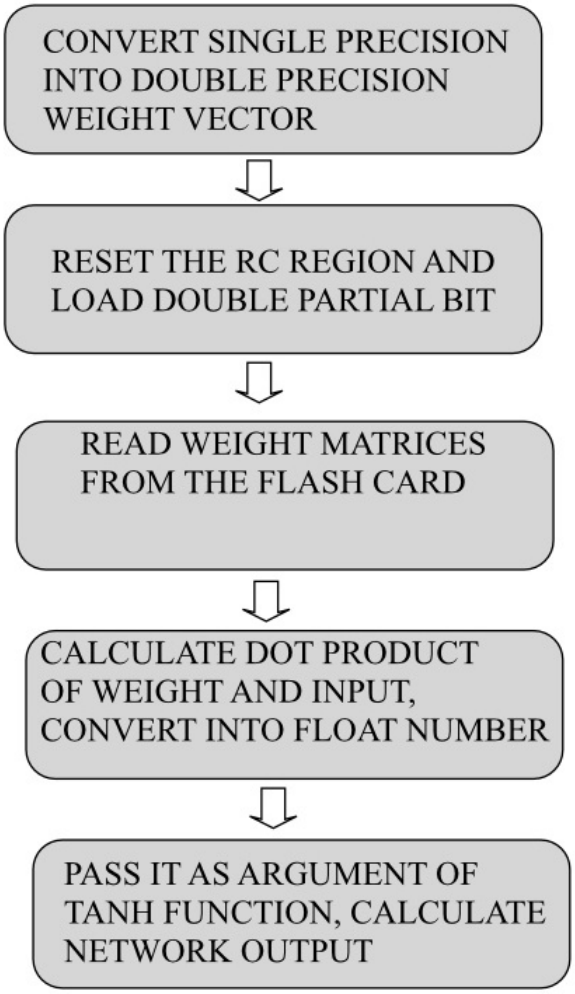

Figure 5.6: Classification of test image on board.

5.4 Pseudo Code

Begin

- Include
  - *Xilinx specific header files for input/output*
  - *Header files for ICAP*
  - *Header files for flash memory*
  - *Header files for UART communication*
  - *Basic math header files*
- *Initialize* the *Flash card interface controller* and *ICAP device*
- *Load* the *Single precision bit file*
- *Read* the *Test image* and *Mean image*
- *Subtract* the *Mean image* from *Test image*
- *Multiply* the *Eigen images* with the *difference* such that feature vector is obtained
- Convert the single precision feature vector into double precision
- Reset the reconfigurable region and load the double partial bit
- Normalize the feature vector by subtracting mean and dividing by standard deviation
- Read the weight matrix from the flash card
- Perform matrix multiplication of the feature vector with input-hidden layer weight matrix
- Convert the matrix into decimal number
- Pass it as an argument to hyperbolic tangent
- Perform matrix multiplication of the result with hidden-output layer weight matrix
- Based on the magnitude of the final output, take the decision.

End

5.5 Results

The variables during training are saved into .mat file which is later loaded during testing. Figure.5.7 shows a screenshot of MATLAB Command line where feature extraction is performed on three test images and is classified accordingly.

```
>> load('training3.mat')
>> testing
clutter

ans =

   -93

clutter

ans =

   -81

target

ans =

    95
```

Figure 5.7: Testing in MATLAB

Figure.5.8 shows a screenshot of the HyperTerminal where the same test images are classified.

```
outpred is -93
              clutter
outpred is -81
              clutter
outpred is 95
             target
```

Figure 5.8: Testing on Board

Test results from MATLAB and board go well in hand with each other. Since there is no support for printing double numbers on the board, the final predicted output is scaled by 100, converted into an integer and then displayed.

Therefore, if the predicted result is close to 100 then the image is classified as target and if it is found close -100 then it classified as clutter.

Chapter 6
Conclusion and Future Work

6 Conclusion and Future Work

6.1 Conclusions

In this project a partially reconfigurable hardware with two different architectures is proposed. Target Recognition for naval applications is implemented on the reconfigurable hardware. Huge improvement in processing speed over software implementation is achieved. Minimal amount of hardware resources are utilized. The area/gate utilization of the FPGA is small thus reduces the hardware resources needed.

The training was done on MATLAB and the testing was done on the FPGA.The functional correctness of the design was verified by using MATLAB.

Therefore, due to reprogram ability of FPGAs, the proposed architecture possessed the speed of hardware while retaining the flexibility of a software Implementation.

6.2 Future Work

This design has a single FPU .The architecture can be multi-FPU which increases the performance of the design .Pipelining can be implemented with a multi-FPU architecture. The proposed design implements PCA for feature extraction and ANN for classification .In future, other feature extraction and classification algorithms can be implemented. For achieving a better accuracy and to build a reliable system the decision can made be by combining the results of more than one recognition algorithms.

Bibliography

Bibliography

References

6.3 Journals, Proceedings, Transactions and Publication Papers

[1] Katherine Compton and Scott Hauck, *An Introduction to Reconfigurable Computing*. IEEE Computer, 2000.

[2] Clay Gloster, *Floating Point Functional Cores for Reconfigurable Computing Systems*. 2003

[3] Ali Azarian and Mahmood Ahmadi, *Reconfigurable Computing Architecture Survey and Introduction*. 2nd IEEE International Conference on Computer Science and Information Technology. Vol.03, pp.269-274, Aug 2009.

[4] T.J. Todman, G.A. Constantinides, S.J.E. Wilton, O. Mencer, W. Luk and P.Y.K. Cheung, *Reconfigurable computing: architectures and design methods*.

6.4 Websites

[5] `https://onionesquereality.wordpress.com/tag/eigenfaces/`

[6] `http://www.webpages.ttu.edu/dleverin/neural_network/neural_networks.html`

[7] `http://pages.cs.wisc.edu/~bolo/shipyard/neural/local.html`

6.5 Datasheets, Manuals and User Guides

[8] UG702, *Partial Reconfiguration User Guide*, April 24, 2012.

[9] UG081, *MicroBlaze Processor Reference Guide*, November 15,2011.

[10] DS531, *LogiCORE IP Processor Local Bus plb*, September 21, 2010.

[11] DS444, *IP Processor Block RAM (BRAM) Block*, March 2, 2010.

[12] DS452, *IP Processor LMB BRAM Interface Controller*, March 2, 2010.

[13] DS445, *Local Memory Bus*, December 2, 2009.

[14] DS641, *MicroBlaze Debug Module*,July 23, 2010.

[15] DS571, *XPS UART Lite*,April 19,2010.

[16] DS583, *XPS SYSACE Interface Controller*,July 20,2009.

[17] DS586,*LogiCORE IP XPS HWICAP*,July 23, 2010.

Appendix I
Tools

A1 Tools

In this section, a brief discussion on the tools that are made use of in this project is provided. The various tools and environment used in the design, implementation are listed below:

- Design Language
  - MATLAB
  - Xilinx C
  - Verilog
- Software Tools

Hardware Design and Synthesis	**Xilinx Platform Studio 12.4**
Floorplanning and Bitstream generation	Xilinx PlanAhead 12.4
Executable generation	Xilinx SDK 12.4

Table 6.1: Values of operands for various operations to a double precision RM

- Hardware Specifications

Device	XC5VFX70T
Family	Virtex 5
Package	FFG665C

Table 6.2: Values of operands for various operations to a double precision RM

Appendix II
Reports

A2 Reports

Pr_verify report is used to check the input and output nets of all the RMs .It checks for any errors and if the static logic is same throughout all the modules. The report is as follows:

```
Analyzing Designs:
D:\pca_ann\project_1\project_1.runs\double \double_routed.ncd
D:\pca_ann \project_1project_1.runs \single \single_routed.ncd
Number of matched proxy logic bels = 257
Number of matched external nets = 259
Number of matched global clock nets = 3
Number of matched Reconfigurable Partitions = 0
SUCCESS!
```

```
Analyzing Designs:
D:\pca_ann\project_1\project_1.runs\single\single_routed.ncd
D:\pca_ann\project_1\project_1.runs\config_bb\config_bb_routed.ncd
Number of matched proxy logic bels = 257
Number of matched external nets = 259
Number of matched global clock nets = 3
Number of matched Reconfigurable Partitions = 0
SUCCESS!
```

```
Analyzing Designs: D:\pca_ann\project_1\project_1.runs\config_bb\config_bb_routed.ncd
D:\pca_ann\project_1\project_1.runs\double\double_routed.ncd
Number of matched proxy logic bels = 257
Number of matched external nets = 259
Number of matched global clock nets = 3
Number of matched Reconfigurable Partitions = 0
SUCCESS!
C:\Xilinx\12.4\ISE_DS\ISE\bin\nt64\unwrapped\pr_verify.exe
D:\pca_ann\project_1\project_1.runs\double\double_routed.ncd
D:\pca_ann\project_1\project_1.runs\single\single_routed.ncd
D:\pca_ann\project_1\project_1.runs\config_bb\config_bb_routed.ncd => PASS
```