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.Fig.1 represents the attributes of RC computing with respect to Processor and ASIC.

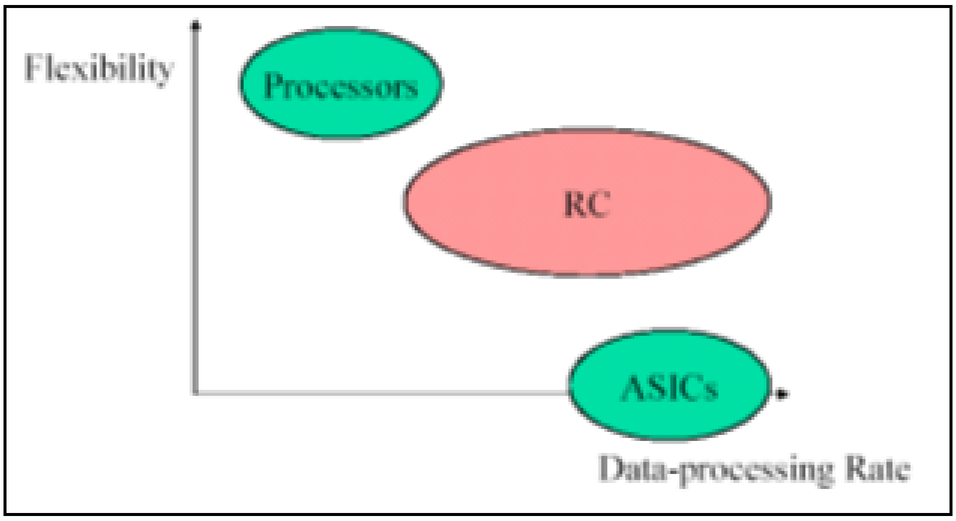


Figure 1 : Flexibility vs Data-Processing rate

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].

Fig.2 gives an overall picture of how the reconfigurable modules are swapped in and out of the reconfigurable partition which will be created during the floorplanning.

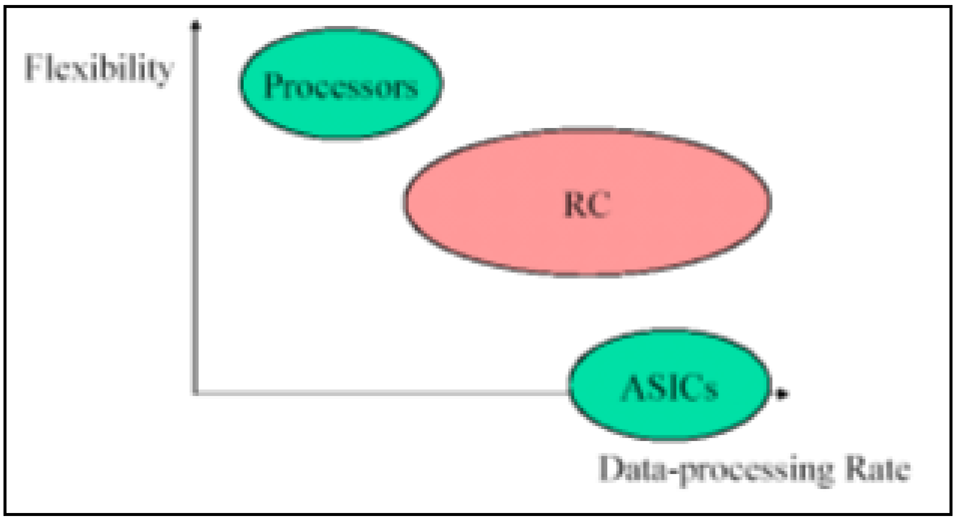


Figure 2 : Partial dynamically reconfigurable architecture

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

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. Fig.3 shows the basic block diagram of a target recognition system.

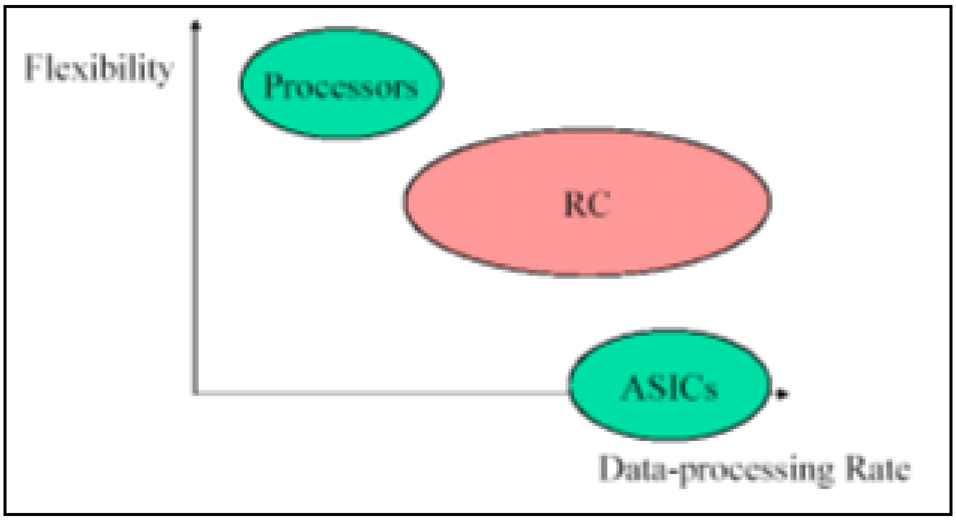


Figure 3 : A basic block diagram for target recognition

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.Fig.4 shows the basic architecture of the feedforward Artificial Neural Network (ANN).

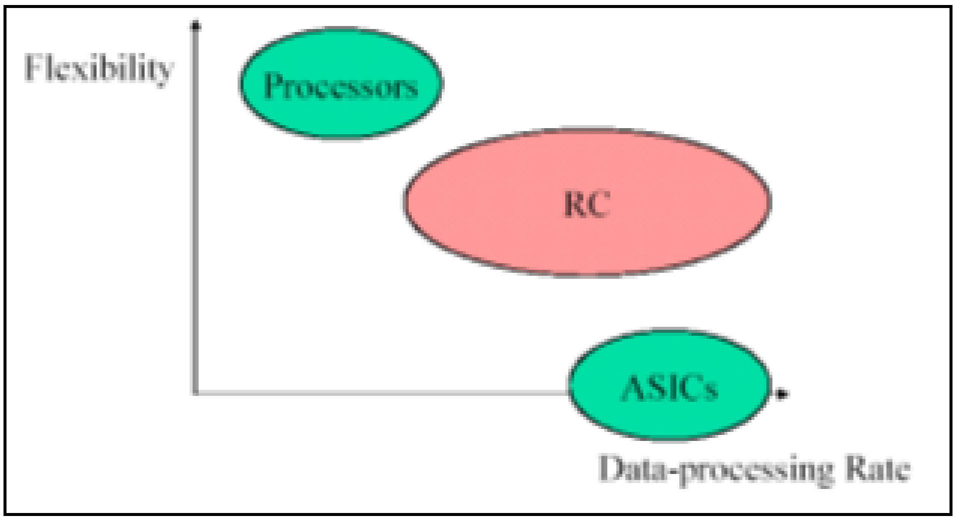


Figure 4 : Basic Architecture of Artificial Neural Network

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 reconïňĄgurable 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 reconïňĄgurable 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 reconïňĄgurable computing include a reduction in size and component count (and hence cost), improved time-to-market, and improved ïňĆexibility and upgradability. These advantages are especially important for embedded applications [4].

Chapter 2
Project Deliverables

2 Project Deliverables

2.1 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.

2.2 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.

2.3 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 5 illustrates the premise behind Partial Reconfiguration.

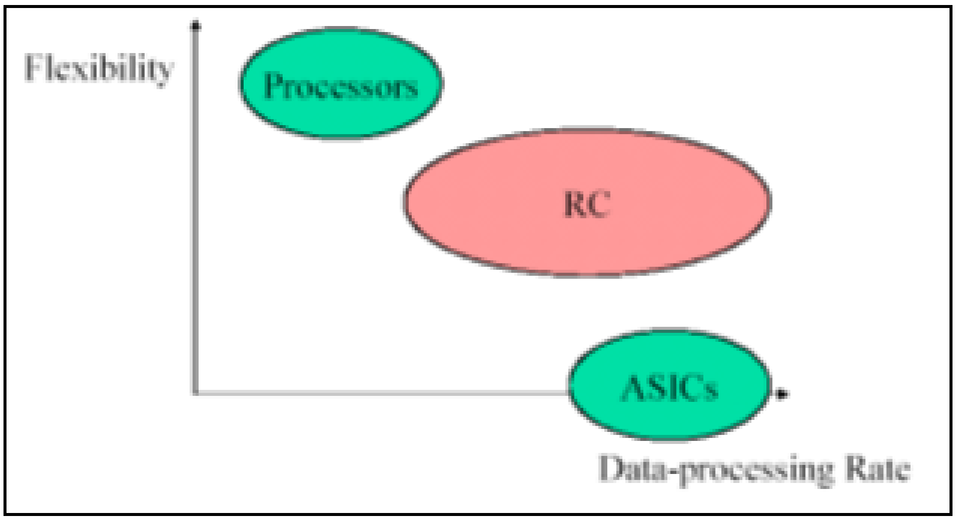


Figure 5 : Basic Premise of Partial Reconfiguration

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 âĂIJAâĂİ 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,

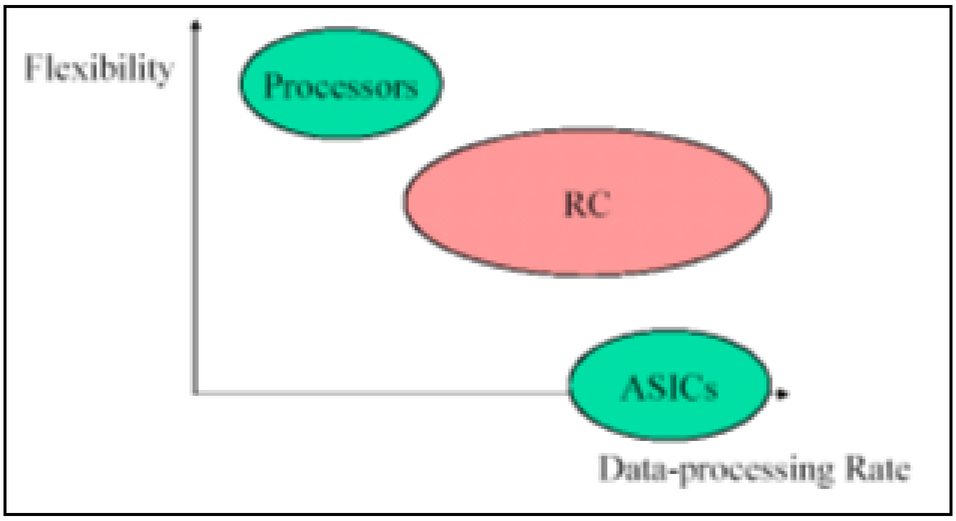


Figure 1 : Image

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 âĂŸlearning ruleâĂŹ which modifies the weights of the connections according to the input patterns that it is presented with. We make use of âĂŸdelta ruleâĂŹ 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. «formula» where Sj is the sum of all relevant products of weights and outputs from the previous layer i, wij represents the relevant weights connecting layer i with layer j, ai represents the activations of the nodes in the previous layer i, aj 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

  «formula»

  where Ep is total error over the training pattern, Â¡ is a value applied to simplify the functionâĂŹs derivative, n represents all output nodes for a given training pattern, tjn represents the target value for node n in output layer j, and ajn 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

  «formula»

  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,

  «formula»

  where, âĆň is the learning rate. A higher value for âĆň 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.

  «formula»

  âĆň is 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.

  – 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, âĂIJImplementation of Floating-Point Operations for 3D Graphics on a Coarse-Grained Reconfigurable ArchitectureâĂİ 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.

E.Manikandan and K.A.Karthigeyan, âĂIJDesign of Parallel Vector/Scalar Floating Point Co-Processor For Reconfigurable ArchitectureâĂİ 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.

ALI AZARIAN AND MAHMOOD AHMADI, âĂIJRECONFIGURABLE COMPUTING ARCHITECTUREâĂİ 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.

PIERRE BALDI AND KURT HORNIK, âĂIJNEURAL NETWORKS AND PRINCIPAL COMPONENT ANALYSIS: LEARNING FROM EXAMPLES WITHOUT LOCAL MINIMAâĂİ 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

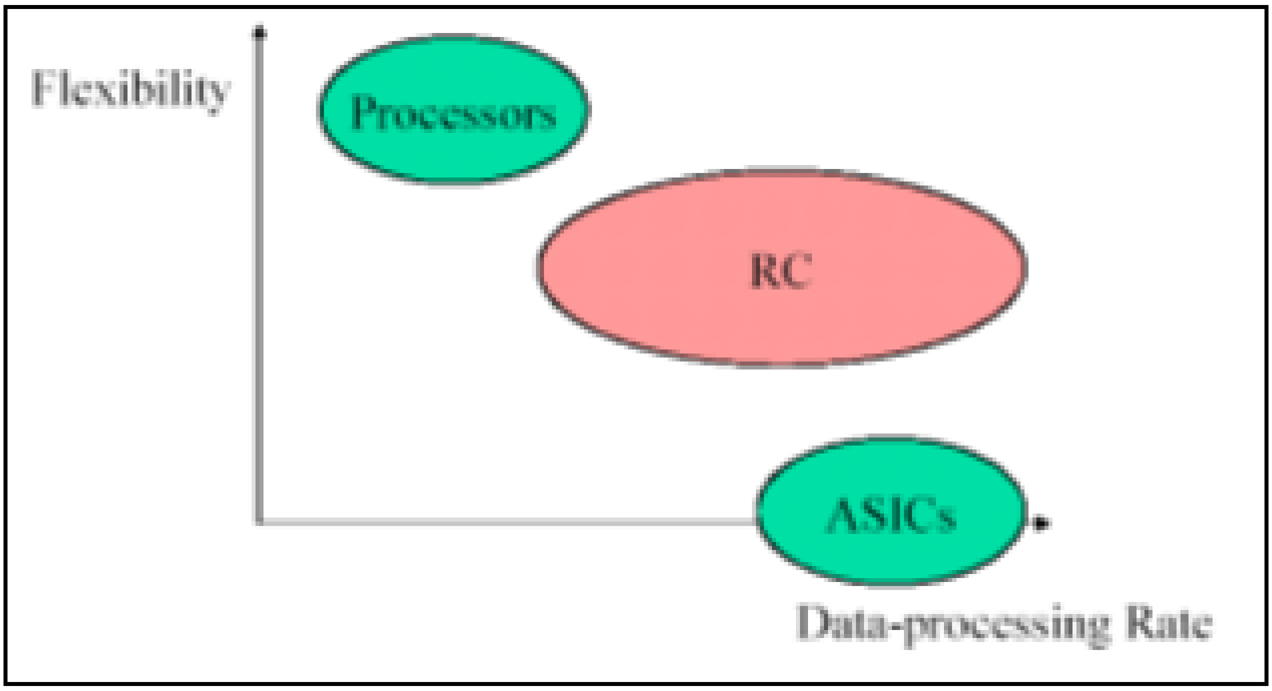
The XPS HWICAP (Hardware ICAP) IP [9.]enables an embedded microprocessor to read and write the FPGA configuration memory through the Internal Configuration Access Port (ICAP) at run time. Using ICAP we can write software programs for an embedded processor that modifies the circuit structure and functionality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



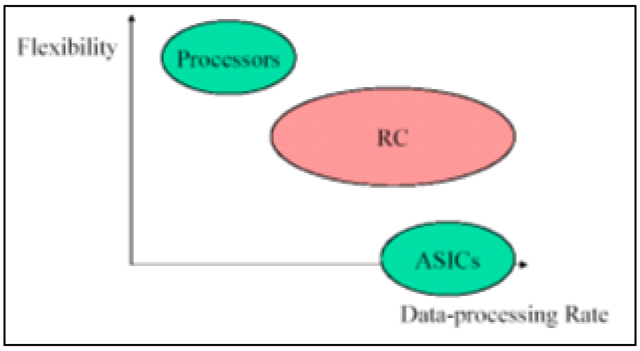


Figure 6 : Hardware embedded design block diagram of the application

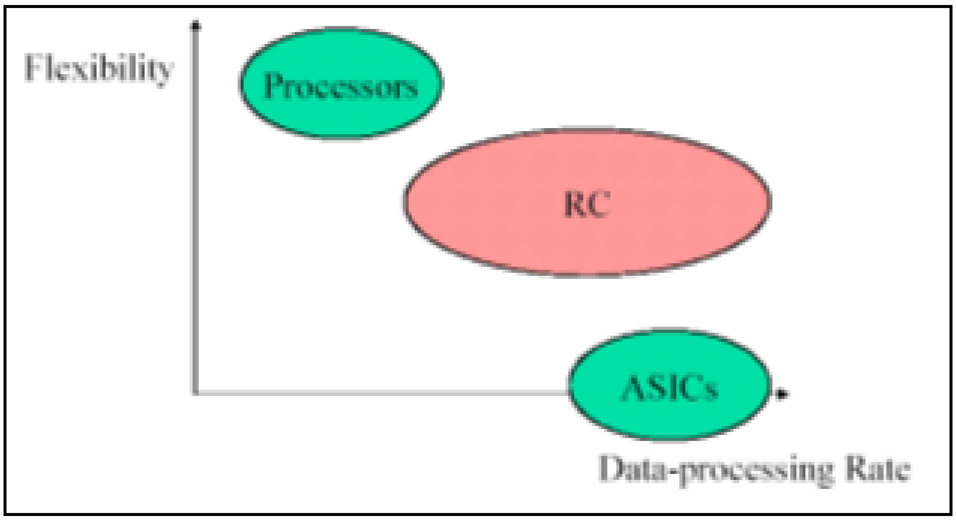


Fig.7: Bus Interfaces in the application

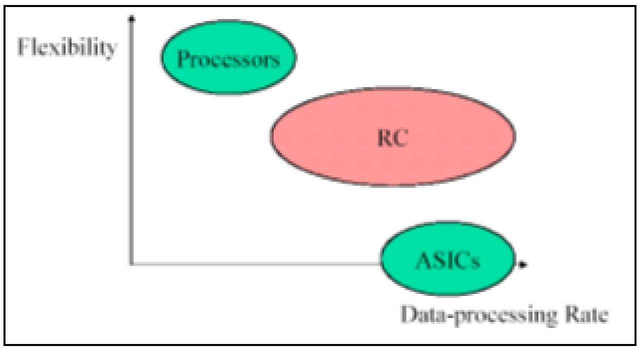


Fig.8: Address Map for hardware modules

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.

END oF REPORT

$$\begin{aligned}(x+y)^3 &= (x+y)^2(x+y)\\ &= (x^2+2xy+y^2)(x+y)\\ &= (x^3+2x^2y+xy^2)+(x^2y+2xy^2+y^3)\\ &= x^3+3x^2y+3xy^2+y^3\end{aligned} \tag{4.1}$$

Phasellus viverra nulla ut metus varius laoreet. Quisque rutrum. Aenean imperdiet. Etiam ultricies nisi vel augue. Curabitur ullamcorper ultricies

4.2 Heading on level 2 (subsection)

Lorem ipsum dolor sit amet, consectetuer adipiscing elit.

$$A = \begin{bmatrix} A_{11} & A_{21} \\ A_{21} & A_{22} \end{bmatrix} \tag{4.2}$$

Aenean commodo ligula eget dolor. Aenean massa. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Donec quam felis, ultricies nec, pellentesque eu, pretium quis, sem.

4.2.1 Heading on level 3 (subsubsection)

Nulla consequat massa quis enim. Donec pede justo, fringilla vel, aliquet nec, vulputate eget, arcu. In enim justo, rhoncus ut, imperdiet a, venenatis vitae, justo. Nullam dictum felis eu pede mollis pretium. Integer tincidunt. Cras dapibus. Vivamus elementum semper nisi. Aenean vulputate eleifend tellus. Aenean leo ligula, porttitor eu, consequat vitae, eleifend ac, enim.

Heading on level 4 (paragraph) Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Aenean commodo ligula eget dolor. Aenean massa. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Donec quam felis, ultricies nec, pellentesque eu, pretium quis, sem. Nulla consequat massa quis enim.

5 Lists

5.1 Example for list (3*itemize)

- First item in a list
  - First item in a list
    - First item in a list
    - Second item in a list
  - Second item in a list
- Second item in a list

5.2 Example for list (enumerate)

1. First item in a list
2. Second item in a list
3. Third item in a list