A Thesis

entitled

FPGA Implementation of a Support Vector Machine based Classification System and its Potential Application in Smart Grid

by

Xiaohui Song

Submitted to the Graduate Faculty as partial fulfillment of the requirements for the

Master of Science Degree in Electrical Engineering

_____

Dr. Hong Wang, Committee Chair

_____

Dr. Lingfeng Wang, Co-Committee Chair

_____

Dr. Weiqing Sun, Committee Member

_____

Dr. Patricia R. Komuniecki, Dean
College of Graduate Studies

The University of Toledo

December 2013

An Abstract of

FPGA Implementation of a Support Vector Machine based Classification System and its Potential Application in Smart Grid

by

Xiaohui Song

Submitted to the Graduate Faculty as partial fulfillment of the requirements for the Master of Science Degree in Electrical Engineering

The University of Toledo
December 2013

Support Vector Machines (SVMs) is a popular classification and regression prediction tool that uses supervised machine learning theory to maximize the predictive accuracy. This paper focuses on the field programmable gate array (FPGA) implementation of a Support Vector Machine classification system. Owing to the advanced parallel calculation feature provided by FPGA, a fast data classification can be achieved by the FPGA-based two-class SVM classifier. The classification system works both in linear mode or non-linear mode, depending on the dimensions of the classification. Simulated results demonstrate that the classification system is effective in fast data classification, as well as a promising technique used in Smart Grid to strengthen the communication security.

For my lovely, brilliant wife, Lu Wang. Your love, patience, support and understanding have encouraged me to finish this study and thesis

Acknowledgements

First, I would like to express the deepest appreciation to my advisors Dr. Hong Wang and Dr. Lingfeng Wang, who have the attitude and the substance of a genius: they continually and convincingly conveyed a spirit of adventure in regard to research and scholarship, and an excitement in regard to teaching. Without their guidance and persistent help this thesis would not have been possible.

I would like to thank my committee member Dr. Weiqing Sun, for his dedication and precious time from a busy schedule.

In addition, I would like to express my gratefulness and respect to my parents and friends. Their encouragements are responsible for all that I have and will achieve.

Contents

List of Tables

List of Figures

List of Abbreviations

ASIC……. Application Specific Integrated Circuit
ASSP..... Application Specific Standard Product

CIA……..... Confidentiality, Integrity, and Availability
CORDIC……..... Coordinated Rotation Digital Computer
COTS……..... Commercial off The Shelf

DSP……..... Digital Signal Processing

ERM……..... Empirical Risk Minimization

FPGA……....Feild Programmable Gate Array

GPU……....Graphics Processing Unit

LE……..... Logic Element
LNS……..... Logarithmic Number Systems

MAC……..... Multiply ACcumulate

NIPS ……..... Neural Information Processing Systems
NRE……..... Nonrecurring Engineering

OEM……..... original equipment manufacturer

SoC……....Systems on a Chip
SRAM……..... Static Random Access Memory
SRM ……..... Structural Risk Minimization
SV……..... Support Vectors
SVM ……..... Support Vector Machine

Chapter 1

Introduction

1.1 Background and Motivation

For last two decades, machine-learning researchers have been working on the area of improving classifier effectiveness, at the same time, the exploration of machine-learning has led to a new generation of state-of-the-art classification algorithms, such as support vector machines (SVMs) [1], boosted decision trees, regularized logistic regression, neural networks, and random forests. Many of these algorithms, including support vector machines, have been applied with success to information analysis problems, especially data classification and regression. Support vector machines appeared in the early nineties as optimal margin classifiers in the context of Vapnik's statistical learning theory [1]. Since then, SVMs have been successfully applied to massive information analysis problems in the real-world, often providing satisfied results compared with many other algorithms. The SVMs algorithm process the data by

minimizing an empirical risk in a well-posed and consistent way based on regularization theory. SVM classification is a kind of large-margin classifier: it is a vector space based machine learning algorithm where the goal is to find a decision boundary between two classes that is maximally far from any point in the training data. The basis of SVMs is the projection of the low-dimensional training data in a higher dimensional feature space; it is easier to separate the input data in this higher dimensional feature space. Moreover, through this projection it is possible that training data–which cannot be separated linearly in the low-dimensional feature space–can be separated linearly in the high-dimensional space using Kernel Functions.

Machine Learning is considered to be a subfield of artificial intelligence and focuses on the development of techniques and algorithms which enable the computer to learn. Its purpose is to solve practical problems using Machine Learning theory, and many algorithms are developed which enable the machine to learn and perform real-world tasks and activities. As a Machine Learning method, the support vector machine was initially popular with the NIPS community and is now an active part of the Machine Learning research around the world. SVM becomes famous when it is used for pattern recognition; it gives accuracy comparable to sophisticated neural networks with elaborated features in a handwriting recognition task [2]. It is also being used for many

applications, such as hand writing analysis, face analysis and so forth, especially for pattern classification and regression–based applications. The foundations of Support Vector Machines (SVM) have been developed by Vapnik and gained popularity due to many promising features, such as better empirical performance. The formulation uses the Structural Risk Minimization (SRM) principle–which has been shown to be superior [3]–to the traditional Empirical Risk Minimization (ERM) principle, a method used by conventional neural networks. SRM minimizes an upper bound on the expected risk, where as ERM minimizes the error on the training data. It is this difference which equips SVM with a greater ability to generalize, which is the goal in statistical learning. SVMs were developed in order to solve the classification problem, but recently they have been extended to solve regression problems [4].

Due to the powerful Machine Learning algorithm and high prediction accuracy, the applications of SVMs progressively increased in last decade, especially in classification problems and pattern recognition problems, as well as providing a good general performance for a wide range of regression and classification tasks. By optimizing the use of the available computing resources, the performance of the SVMs can be maximized. Implementing SVM classifiers on suitable computing devices like FPGAs can exploit the potential of custom precision algorithms. FPGAs are

semiconductor devices, which contain programmable logic elements called ‘logic blocks’. With the development of integrated circuit technologies, modern FPGAs contain coarse grain components, such as memory blocks and embedded multipliers or digital signal processing blocks (DSPs). The implementation of complex combinational functions–such as multipliers onto the programmable logic blocks–has enabled the FPGA devices to boost their performance efficiency. Nowadays, FPGA devices offer a vast amount of DSP blocks and a hierarchy of different memory sizes, thus providing a high level of flexibility and large amounts of parallel computational power. The reprogrammable feature of FPGAs offers a significant advantage against application-specific cases, and targets different classification problems which may vary in size, dimensionality, and dynamic range constraints. Additionally, modern FPGA devices are able to provide equal or superior performance at a lower power cost than general purpose processing units [5].

1.2 Related Work

This section overviews some previous FPGA- or GPU- mapped works on the SVM classification. There exists a fair amount of work on accelerating both the SVM training and classification for general-purpose processors and DSPs, which aim to provide higher performance on such platforms. The work in [6] presents an evaluation of

SVM implementation on embedded processor architectures, and proposes architectural modifications in order to improve their performance. An analysis was performed in [7] where critical parts of the SVM algorithm were mapped between hardware and software, demonstrating how hardware can be used to accelerate SVM computations. An attempt to implement SVMs on a microcontroller was presented in [8], and dealt with issues such as limited memory and hardware. Recently, Graphics Processing Units (GPUs) have been utilized in the implementation of SVMs [9], Their parallel nature shows significant speedups when compared with general-purpose processors. However, caution need to be taken when implementing using GPUs. Efficient programming was needed in order to provide high performance, primarily because of GPU's fixed hardware, especially the interconnection which may not suit the computation or data flow of some applications.

Due to the potential real-time information processing performance advantages they offer for both data training and classification, hardware implementations of SVMs have gained noticeable interest in recent years. Significant progress has been made in the implementation of SVMs on custom hardware, mostly on FPGAs. A homogeneous FPGA-based architecture for the SVM training was introduced in [10], and the results can be potentially extended for the acceleration of the SVM classification. Another homogeneous work was presented in [11], where a parallel FPGA co-processor is

proposed for the inner product calculations using the available DSP units of the targeted device. The kernel computations are performed by the host CPU, unlike in [10], where floating-point pipelines are utilized for the Kernel Functions. The integrated solution in [11] targets a large FPGA device and succeeds in accelerating the SVM classification. Nevertheless, this paper didn't exploit the heterogeneity and the fully custom-arithmetic potential of modern FPGA devices, nor did it target the precision requirements of the training problem. The multipliers were implemented solely by hard-logic DSP blocks, and the large amount of the FPGA's soft-logic was not efficiently utilized. The work in [12] presents an in-depth analysis of their SVM training architecture on a Xilinx Virtex II device. This paper could potentially be exploited for a classification solution. However, due to the resource constraints of the targeted device, it didn't exploit the parallelization potential of modern FPGAs. In [13], a novel implementation based on logarithmic number systems (LNS) was presented. The LNS-based implementation of the SVM kernel was also adopted in [14] in order to produce a hardware-friendly approach. These works focus more on the potential of using LNS for the SVM problem rather than the acceleration of the problem, because the targeted devices are small and only one Multiply-ACcumulate (MAC) unit was used for all the dot-product evaluations. The FPGA architecture proposed in [15] employs a hardware-friendly approach for the kernel

evaluations based on CORDIC algorithm, the paper wasn't optimized in utilizing FPGA logic resources in order to speed-up the SVM classification. The SVM classification was used for video shot boundary detection in [16]. Only linear SVMs were targeted and the FPGA device was used for the dot-product mapping of the SVM algorithm.

For the other works, such as [17] and [18], implement the SVM classification problem on the parallel computing resources of a GPU using NVidia's compute unified device architecture [19] programming environment. Their main differences are related to the chosen floating-point precision for the kernel computations and the usage of the host CPU for the processing of some part of the kernel evaluations, before the results are fed to the GPU. Furthermore, the GPU work in [20] targets a geometric interpretation of the SVM training problem [21] based on Gilbert's algorithm [22], while the classification implementation is similar to [17] and [18].

1.3 Project Objectives

Based on the issues listed above, a FPGA-based SVM classification system is presented in order to achieve a fast two-class data classifier. This work focuses on an FPGA implementation for the two-class SVM classifier, including both linear and non-linear classification which fully exploits the parallel processing power of the FPGA

computing resources and offers scalability and adaptively to the targeted classification problems under the available resource constraints. The proposed system has two main characteristics:

1) It is a strong computational tool with the great power and high prediction accuracy to solve the data classification problem using the SVM Machine Learning algorithm.
2) Owing to the advanced parallel calculation feature provided by FPGA, the system provides a fast two-class data classification with a satisfying time consumption and field integration ability.

The objective is to resolve fast classification problems under the precision requirements using modern FPGA devices, as well as to build a multilayer two-way communication network using the FPGA-based SVM classification system in order to secure the privacy and integrity of communications between parties in Smart Grid for future applications.

1.4 Synopsis of Thesis

This thesis presents and discusses the design and testing of the FPGA based SVM classification system. It begins by reviewing the Machine Learning–based algorithm,

which is called the Support Vector Machine, in order to provide the basic information, knowledge and ideas about how the SVM algorithm provides class prediction for the unknown data based on the labeled data (training dataset).

The SVM algorithm for the classification system is discussed in the next step, the section includes the mathematics expression of the algorithm and details of the algorithm flow for FPGA implementation. This is done by first figuring out the mathematical expression of the SVM algorithm and how to transform the mathematic functions to a FPGA–applicable algorithm. Then, it is time to think about how to apply the proposed algorithm on the target device. The design of modules is discussed next, which can work both separately and in parallel, Then the modules are mapped appropriately in order to construct a classification system. Next, a FPGA–based SVM classification system is built. The system can work in both linear and non-linear modes. After that, the testing results are collected and analyzed in order to draw a conclusion of the research. In the last section, the probable future work is discussed.

1.5 Outline of Thesis

The remainder of the thesis is organized as follows. In Section 2, the Machine Learning theory and support vector machine algorithm are introduced. The design of the

SVM algorithms used in the classification system is detailed in Section 3. The FPGA architecture mapping of the proposed algorithm is presented in Section 4. In Section 5, the implementation results of the SVM classification system are analyzed. Conclusions and future work are given in the final section.

Chapter 2

Literature Review

2.1 Outline

This chapter consists of two parts. The first part presents the background knowledge of Machine Learning and the Support Vector Machine algorithm, which are important in understanding this study. The SVM algorithm is one of the most popular Machine Learning based algorithms. The second part introduces the current status and advantages of modern PFGAs.

2.2 Support Vector Machine Concepts and Applications

Kernel-based techniques, such as Support Vector Machines, Bayes Point Machines, Kernel Principal Component Analysis, and Gaussian Processes, represent a major development in Machine Learning algorithms. Support vector machines (SVMs)

are a group of supervised learning methods that can be applied to classification or regression.

Support Vector Machines represent an extension of nonlinear models of the Generalized Portrait Algorithm developed by Vapnik and Lerner [23]. The SVM algorithm is based on the Statistical Learning theory and the Vapnik–Chervonenkis (VC) dimension [24]. The Statistical Learning theory, which describes the properties of learning machines that allow them to give reliable predictions, was reviewed by Vapnik in three books: *Estimation of Dependencies Based on Empirical Data* [25], *The Nature of Statistical Learning Theory* [1], *and Statistical Learning Theory* [26]. In the current formulation, the SVM algorithm was developed at AT&T Bell Laboratories by Vapnik et al [27–33].

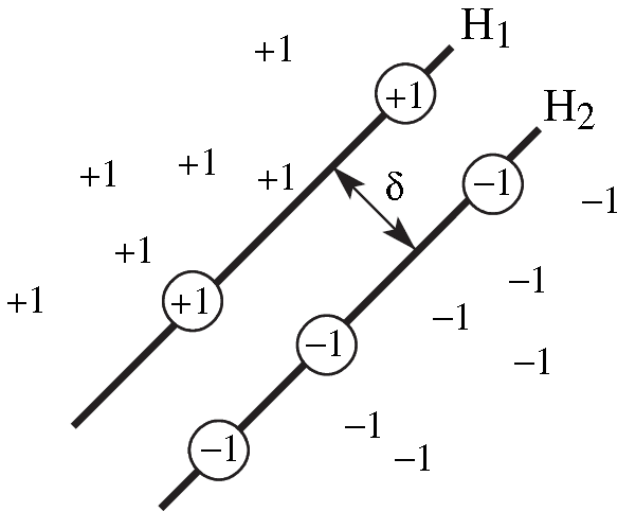

Figure 2-1: Maximum separation hyperplane.

SVM models were originally defined for the classification of linearly separable classes of objects. Such an example is presented in Figure 2-1. For these two-dimensional objects that belong to two classes (class +1 and class -1), it is easy to find a line that separates them perfectly.

For any particular set of two-class objects, an SVM finds the unique hyperplane having the maximum margin (denoted with δ in Figure 2-1). The hyperplane H_1 defines the border with class +1 objects, whereas the hyperplane H_2 defines the border with class -1 objects. Two objects from class +1 define the hyperplane H_1, and three objects from class -1 define the hyperplane H_2. These objects, represented inside circles in Figure 2-1, are called support vectors. A special characteristic of SVM is that the solution to a classification problem is represented by the support vectors that determine the maximum margin hyperplane.

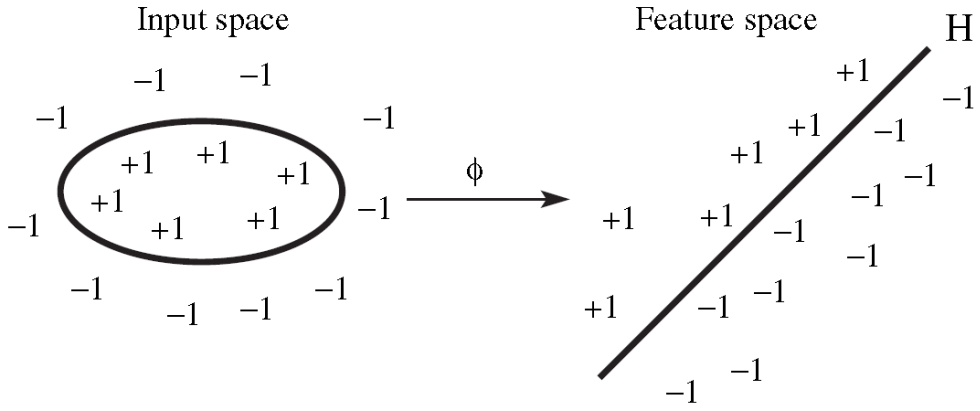

Figure 2-2: Linear separation in feature space.

SVM can also be used to separate classes that cannot be separated with a linear classifier (Figure 2-2, left). In such cases, the coordinates of the objects are mapped into a feature space using nonlinear functions called feature functions ϕ. The feature space is a high-dimensional space in which the two classes can be separated with a linear classifier (Figure 2-2, right).

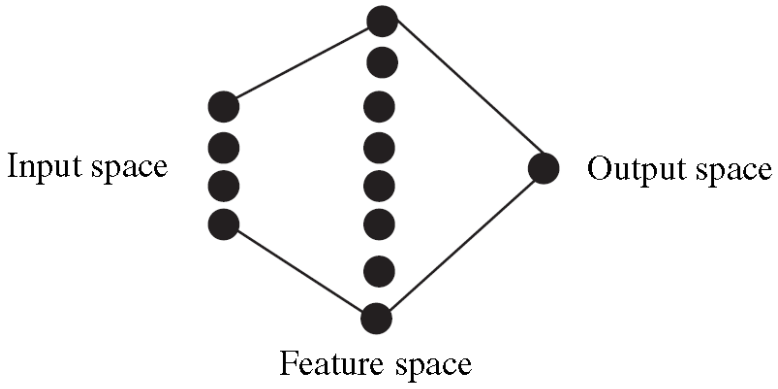

Figure 2-3: Support vector machines map the input space into a high-dimensional feature space.

As presented in Figure 2-2 and 2-3, the nonlinear feature function ϕ combines the input space (the original coordinates of the objects) into the feature space, which can even have an infinite dimension. Because the feature space is high dimensional, it is not practical to directly use feature functions ϕ when computing the classification hyperplane. Instead, the nonlinear mapping induced by the feature functions is computed with special nonlinear functions called kernels. Kernels have the advantage of operating in the input

space, where the solution of the classification problem is a weighted sum of kernel functions evaluated at the support vectors.

To illustrate the SVM capability of training nonlinear classifiers, consider the patterns from Table 2.1. This is a synthetic dataset of two-dimensional patterns, designed to investigate the properties of the SVM classification algorithm. All figures from this chapter presenting SVM models for various datasets were prepared with a slightly modified version of Gunn's MATLAB toolbox. In all figures, class +1 patterns are represented by +, whereas class -1 patterns are represented by black dots. The SVM hyperplane is drawn with a continuous line, whereas the margins of the SVM hyperplane are represented by dotted lines. Support vectors from the class +1 are represented as + inside a circle, whereas support vectors from the class -1 are represented as a black dot inside a circle.

Partitioning of the dataset from Table 2.1 with a linear kernel is shown in Figure 2-4a. It is obvious that a linear function is not adequate for this dataset because the classifier is not able to discriminate the two types of patterns; all patterns are support vectors. A perfect separation of the two classes can be achieved with a degree 2 polynomial kernel (Figure 2-4b).

Table 2.1:

Linearly Non-separable Patterns Used for the SVM Classification Models in Figure 2-4 and 2-5.

Pattern	x1	x2	Class
1	2	4.5	1
2	2.5	2.9	1
3	3	1.5	1
4	3.6	0.5	1
5	4.2	2	1
6	3.9	4	1
7	5	1	1
8	0.6	1	-1
9	1	4.2	-1
10	1.5	2.5	-1
11	1.75	0.6	-1
12	3	5.6	-1
13	4.5	5	-1
14	5	4	-1
15	5.5	2	-1

This SVM model has six support vectors, namely, three from class +1 and three from class -1. These six patterns define the SVM model and can be used to predict the class membership for new patterns. The four patterns from class +1 situated in the space region bordered by the +1 margin and the five patterns from class -1 situated in the space region delimited by the -1 margin are not important in defining the SVM model, and they can be eliminated from the training set without changing the SVM solution.

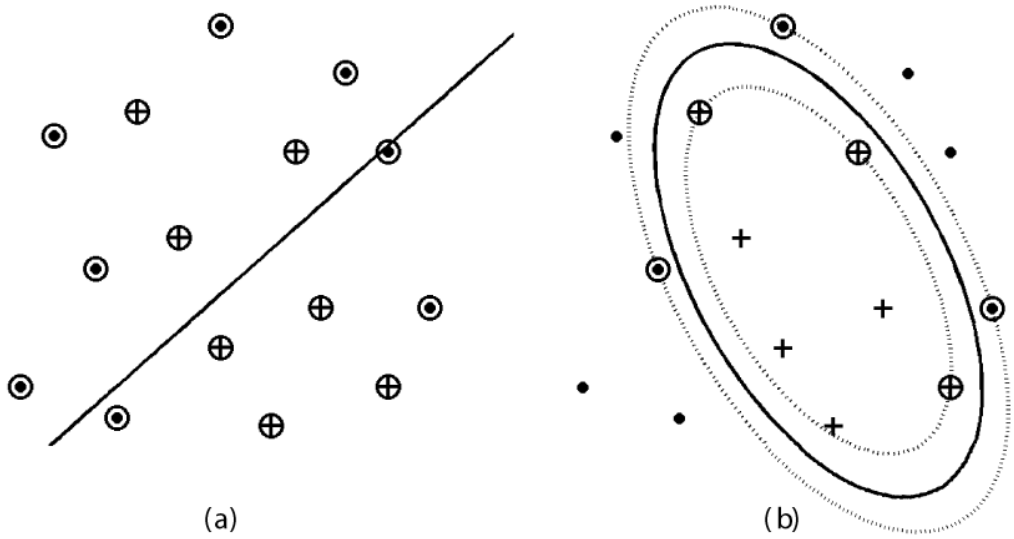

Figure 2-4: SVM classification models for the dataset from Table 2.1.

In this example, the use of nonlinear kernels provides the SVM with the ability to model complicated separation hyperplanes. However, because there is no theoretical tool to predict which kernel will give the best results for a given dataset, experimenting with different kernels is the only way to identify the best function. An alternative solution to discriminate the patterns from Table 2.1 is offered by a degree 3 polynomial kernel (Figure 2-5a) that has seven support vectors, namely, three from class +1 and four from class -1. The separation hyperplane becomes even more convoluted when a degree 10 polynomial kernel is used (Figure 2-5b). It is clear that this SVM model, with 10 support vectors (4 from class +1 and 6 from class -1), is not an optimal model for the dataset from Table 2.1.

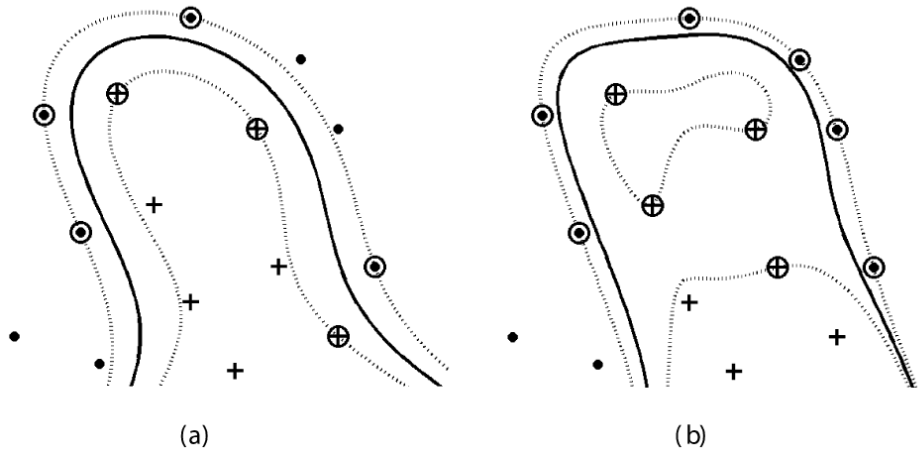

Figure 2-5: SVM classification models obtained with the polynomial kernel for the dataset from Table 2.1: (a) polynomial of degree 3; (b) polynomial of degree 10.

2.3 Modern Field Programmable Gate Array Devices

The Field Programmable Gate Array (FPGA) is a semiconductor device that can be programmed after manufacturing. Instead of being restricted to any predetermined hardware function, an FPGA allows you to program product features and functions, adapt to new standards, and reconfigure hardware for specific applications even after the product has been installed in the field, hence the name "field-programmable". FPGA can be used to implement any logical function that an application-specific integrated circuit (ASIC) could perform, but the ability to update the functionality after shipping offers advantages for many applications [34].

Unlike previous generation FPGAs using I/Os with programmable logic and interconnects, today's FPGAs consist of various mixes of configurable embedded SRAM,

high-speed transceivers, high-speed I/Os, logic blocks, and routing. Specifically, an FPGA contains programmable logic components called Logic Elements (LEs) and a hierarchy of reconfigurable interconnects that allow the LEs to be physically connected. You can configure LEs to perform either complex combinational functions or merely simple logic gates, like AND and XOR. In most FPGAs, the logic blocks also include memory elements, which may be simple flipflops or more complete blocks of memory [35].

As FPGAs continue to evolve, the devices have become more integrated. Hard intellectual property blocks built into the FPGA fabric provide rich functions, while also lowering power and cost and freeing up logic resources for product differentiation. Newer FPGA families are being developed with hard embedded processors, transforming the devices into systems on a chip (SoC) [36].

Compared to ASICs or ASSPs, FPGAs offer many design advantages, including:

1) 1.Performance—Taking advantage of hardware parallelism, FPGAs exceed the computing power of Digital Signal Processors (DSPs) by breaking the paradigm of sequential execution and accomplishing more per clock cycle. BDTI, a noted analyst and benchmarking firm, released benchmarks showing how FPGAs can deliver many times the processing power per dollar of a DSP

solution in some applications. Controlling inputs and outputs (I/O) at the hardware level also provides faster response times and specialized functionality in order to closely match application requirements.

2) Time to market—FPGA technology offers both flexibility and rapid prototyping capabilities in the face of increased time-to-market concerns. You can test an idea or concept and verify it in hardware without going through the long fabrication process of custom ASIC design. You can then implement incremental changes and iterate on an FPGA design within hours, instead of weeks. Commercial off-the-shelf (COTS) hardware is also available with different types of I/O already connected to a user-programmable FPGA chip. The growing availability of high-level software tools decreases the learning curve with layers of abstraction and often offers valuable IP cores (prebuilt functions) for advanced control and signal processing.

3) Cost—The Nonrecurring Engineering (NRE) expense of custom ASIC design far exceeds that of FPGA-based hardware solutions. The large initial investment in ASICs is easy to justify for OEMs shipping thousands of chips per year, but many end–users need custom hardware functionality for the tens to hundreds of systems in development. The very nature of programmable

silicon means you have no fabrication costs or long lead times for assembly. Because system requirements often change over time, the cost of making incremental changes to FPGA designs is negligible when compared to the large expense of respinning an ASIC [37].

4) Reliability—While software tools are the programming environment, FPGA circuitry is truly a "hard" implementation of program execution. Processor-based systems often involve several layers of abstraction to help schedule tasks and share resources among multiple processes. The driver layer controls hardware resources and the OS manages memory and processor bandwidth. For any given processor core, only one instruction can be executed at a time, and processor-based systems are continually at risk of time-critical tasks preempting one another. FPGAs, which do not use OSs, minimize reliability concerns with truely parallel execution and deterministic hardware dedicated to every task.

5) Long-term maintenance—As mentioned earlier, FPGA chips are field-upgradable and do not require the time and expense involved with ASIC redesign. For example, digital communication protocols, have specifications that can change over time, and ASIC-based interfaces may cause maintenance

and forward-compatibility challenges. Furthermore, FPGA chips are reconfigurable and can keep up with future modifications that might be necessary. As a product or system matures, you can make functional enhancements without spending time redesigning hardware or modifying the board layout.

Chapter 3

Support Vector Machine

3.1 Outline

At first, an introduction about support vector machine algorithm and its applications are given. Then in the third subsection, details of linear SVM algorithm are presented including mathematical expression and geometric interpretation. And non-linear SVM algorithm with the Kernel Function is introduced. The whole chapter is a mathematical understanding of the SVM algorithm, prepared for the FPGA implementation in next chapter.

3.2 Algorithm Introduction

Support Vector Machines (SVM) [1] are considered one of the most powerful classification tools due to their state-of-the-art Machine Learning algorithm based on the Vapnik-Chervonenkis learning theory [38]. SVMs can be defined as supervised learning

models which construct a hyperplane or a set of hyperplanes in a high dimensional feature space, and used for classification and regression analysis. The algorithm is trained with a Machine Learning theory to maximize predictive accuracy using optimization theory that implements a learning bias derived from statistical learning theory [39]. With the strong regularization properties–which refer to the generalization of the model to new data–SVM can be an efficient tool with which to solve classification problems [40]. A classification task usually involves training and testing data which consist of some data instances [41]. Each instance in the training set contains one target values and several attributes. The goal of SVM is to produce a model which predicts target value of data instances in the testing set, which are given only the attributes [42]. In SVMs, a dataset consisting of pairs of input vectors and desired outputs is called the training dataset, which is used to design and construct the decision function of the system and, hence, this procedure is usually considered as an instance of supervised learning. Known labels help indicate whether the system is performing in a right way or not. This information points to a desired response, validating the accuracy of the system, or be used to help the system to learn to act correctly [43]. During the training phase, the system identifies the Support Vectors (SVs) [44], which are those data points that can best build a separation model for the classes. Those vectors are then used to predict the class of any future data point

during the classification phase. The classification phase is a step which gives prediction of unknown samples. According to the prediction model from the training phase, new data can be classified based on different key features. During the classification phase, training datasets can be updated with newly obtained data and work as an "online" model to provide most accurate prediction.

3.3 Linear Support Vector Machine

Multi-class classifications can be break up into two-class classification units and non-linear classification problem can be solved by replacing inner product calculation with Kernel Functions.

Given 2 classes C_1 and C_2, $\mathbf{T} = \{(\mathbf{X_1}, y_1) \ (\mathbf{X}_2, y_2) \ \cdots \ (\mathbf{X}_N, y_N)\}$ is a training dataset consisting of samples taken from C_1 and C_2, where $\mathbf{X}_n \in \mathbf{R}^M$, $y_n \in \{1, -1\}$. If $\mathbf{X}_n$ belongs to class C_1, then $y_n = 1$; If $\mathbf{X}_n$ belongs to class C_2, then $y_n = -1$. Finding a real function $g(\mathbf{X})$ in $\mathbf{R}^M$, for any new sample with unknown class, have

$$\begin{cases} g(\mathbf{X}) > 0, & \mathbf{X} \in C_1 \\ g(\mathbf{X}) < 0, & \mathbf{X} \in C_2 \end{cases} \quad \text{or} \quad \begin{cases} \text{sgn}\{g(\mathbf{X})\} = 1, & \mathbf{X} \in C_1 \\ \text{sgn}\{g(\mathbf{X})\} = -1, & \mathbf{X} \in C_2 \end{cases} \tag{1}$$

$g(\mathbf{X})$ is the decision (classification) function. When $g(\mathbf{X})$ is linear function, it's called linear SVM, and when $g(\mathbf{X})$ is non-linear function, it's called non-linear SVM.

As showed in Figure 3-1, the goal of linear SVM is to find a classification line $g(\mathbf{X})$ between C_1 and C_2. It is known that under high-dimensional circumstance, $g(\mathbf{X})$ is a hyperplane. For linear separable classes C_1 and C_2, more than one hyperplanes can be applied to separate them accurately. Assuming that two classes can be separated by hyperplane l, lying on each of l are two parallax hyperplanes l_1 and l_2 with no learning sample points between them. The region bounded by them is called the "margin". Thus the objective of SVMs is to maximize the distance between the classes' hyperplanes or, in other words, to maximize the "margin". After some paragraphs, you may have another heading, in which case you will use a subsection heading as below.

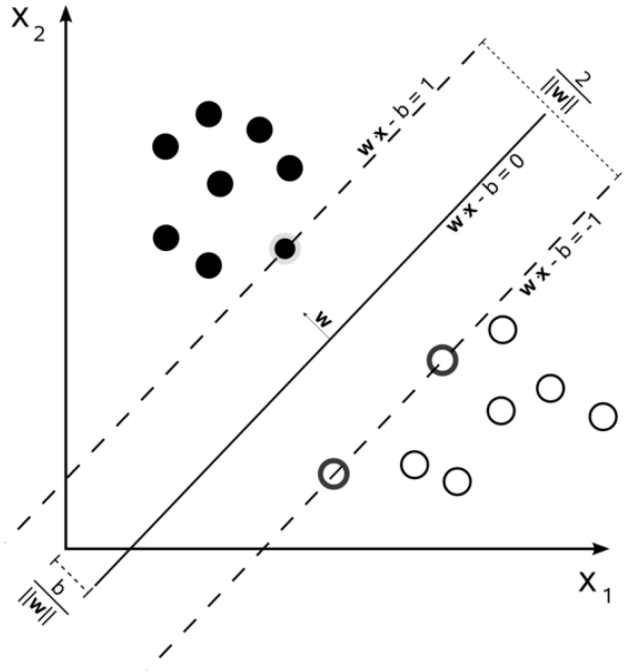

Figure 3-1: SVM separating hyperplane.

The expression of the separating hyperplane is:

$$g(\mathbf{X}) = \langle \mathbf{W} \cdot \mathbf{X} \rangle + b = 0 \tag{2}$$

Where $\langle \cdot \rangle$ denotes an inner product. and $\mathbf{W}$ the normal vector to the hyperplane. The parameter $\frac{b}{\|\mathbf{W}\|}$ determines the offset of the hyperplane from the origin along the normal vector $\mathbf{W}$.

And the hyperplanes l_1 and l_2 can be described by the expression：

$$\begin{aligned} l_1 &: \ \langle \mathbf{W} \cdot \mathbf{X} \rangle + b = 1 \\ l_2 &: \ \langle \mathbf{W} \cdot \mathbf{X} \rangle + b = -1 \end{aligned} \tag{3}$$

The distance between these two hyperplanes is $2 / \|\mathbf{W}\|$, then the problem to maximize the "margin" become the problem to minimize $\|\mathbf{W}\|$. In order to simplify the calculation, substitute $\|\mathbf{W}\|$ with $\frac{1}{2}\|\mathbf{W}\|^2$, then the problem can be expressed as a constrained optimization problem:

$$\begin{aligned} \min_{\mathbf{W},b} \quad & \frac{1}{2}\|\mathbf{W}\|^2 \\ s.t. \quad & y_n \cdot \left(\langle \mathbf{W} \cdot \mathbf{X}_n \rangle + b \right) \geq 1, \quad n = 1, \ 2, \ \cdots, \ N \end{aligned} \tag{4}$$

Using Lagrange multipliers solve the constrained optimization problem, we can get the classification function:

$$g(\mathbf{X}) = \sum_{n=1}^{N} \alpha_n y_n \langle \mathbf{X}_n, \mathbf{X} \rangle - \sum_{n=1}^{N} \alpha_n y_n \langle \mathbf{X}_n, \mathbf{X}_k \rangle + y_k \tag{5}$$

Apply Wolfe's dual form to solve (4) and transform the constrained optimization problem to a concise form:

$$\begin{aligned} \max_{\alpha_i} \quad & -\frac{1}{2}\sum_{i=1}^{p}\sum_{j=1}^{p} y_i y_j \alpha_i \alpha_j \left\langle \mathbf{X}_i \cdot \mathbf{X}_j \right\rangle + \sum_{i=1}^{p} \alpha_i \\ s.t. \quad & \sum_{i=1}^{p} y_i \alpha_i = 0 \\ & \alpha_i \geq 0, \quad i = 1,\ 2,\ \cdots,\ p \end{aligned} \tag{6}$$

3.4 Non-linear Support Vector Machine

In many real-world classification problems, it is often not feasible to linearly separate the data in the original space. SVMs can overcome this problem by mapping the input space to a higher dimensional one where a linear separation may be feasible. Then the non-linear SVM can be expressed as:

$$\begin{aligned} \max_{\alpha_i} \quad & -\frac{1}{2}\sum_{i=1}^{p}\sum_{j=1}^{p} y_i y_j \alpha_i \alpha_j \left\langle \Phi(\mathbf{X}_i) \cdot \Phi(\mathbf{X}_j) \right\rangle + \sum_{i=1}^{p} \alpha_i \\ s.t. \quad & \sum_{i=1}^{p} y_i \alpha_i = 0 \\ & \alpha_i \geq 0, \quad i = 1,\ 2,\ \cdots,\ p \end{aligned} \tag{7}$$

Where $\Phi(\bullet)$ is a mapping function that transform non-linear problem to linear separable problem. Actually, there is no necessity to find $\Phi(\bullet)$ if we can calculate $\left\langle \Phi(\mathbf{X}_i) \cdot \Phi(\mathbf{X}_j) \right\rangle$. Introducing Kernel Function:

$$K(\mathbf{X}_i, \mathbf{X}_j) = \left\langle \Phi(\mathbf{X}_i) \cdot \Phi(\mathbf{X}_j) \right\rangle \tag{8}$$

By replacing dot product with kernel function. This allows the algorithm to fit the maximum-margin hyperplane in a transformed feature space. It is a feasible way to achieve non-linear SVM classification.

Common Kernel Functions:

Polynomial: $K(\mathbf{X}\cdot\mathbf{Y}) = (\mathbf{X}\cdot\mathbf{Y}-1)^p$ (9)

Gaussian radial basis function: $K(\mathbf{X}\cdot\mathbf{Y}) = e^{-\frac{\|\mathbf{X}-\mathbf{Y}\|^2}{\sigma}}$ (10)

Hyperbolic tangent: $K(\mathbf{X}\cdot\mathbf{Y}) = \tanh(\lambda\mathbf{X}\cdot\mathbf{Y}-\delta)$ (11)

Out of many possible Kernel Functions, of special interest are those which satisfy Mercer's condition [45] and can be expressed as an inner product in the high-dimensional space. By applying the kernel, there is no need to explicitly map the data to the higher-dimensional space [46].

Chapter 4

FPGA Architecture Mapping

4.1 Outline

In this chapter, the detailed algorithm flow for the FPGA implementation is given to build the classification system. Then computing modules used to construct the classifiers are designed and tested. After that, the system FPGA architecture mapping is presented and the data processing flow are introduced.

4.2 Linear SVM

The SVM classification algorithm can be expressed with several concepts and equations, but to implement the algorithm, a feasible expression needs to be introduced in the implementation.

For the training phase of the algorithm, the classification function needs to be constructed. Using $\max_{\alpha_n \in [0,\infty), n=1,\cdots,\infty} L_* \left(\mathbf{W}, b, \mathbf{A} \right)$, solve for Lagrange multiplier α_n in (7). Let:

$$\frac{\partial}{\partial \alpha_n} L(\mathbf{W}, b, \mathbf{A}) = -y_n \sum_{k=1}^{N} \alpha_k y_k \langle \mathbf{X}_n \cdot \mathbf{X}_k \rangle + 1 = 0 \quad n = 1,\ 2,\ \cdots,\ N \tag{12}$$

Then we can get all the Lagrange multipliers α_1, α_2, $\cdots$, α_N, as a result, the normal vector of the classification function $g(\mathbf{X}) = \langle \mathbf{W} \cdot \mathbf{X} \rangle + b$ is:

$$\mathbf{W} = \sum_{n=1}^{N} \alpha_n y_n \mathbf{X}_n \tag{13}$$

According to the geometrical significance of linear constrained convex optimization problem, only constrains with $\alpha_k > 0$ is effective. Within the SVM classification problem, the training samples with $\alpha_k > 0$ are support vectors, and they are just on the two sides of the "margin". All the support vectors have to meet constrain:

$$1 - y_k \cdot (\langle \mathbf{W} \cdot \mathbf{X}_k \rangle + b) = 0 \tag{14}$$

Considering $|y_k| = 1$, we can easily construct the classification function:

$$g(\mathbf{X}) = \sum_{n=1}^{N} \alpha_n y_n \langle \mathbf{X}_n, \mathbf{X} \rangle - \sum_{n=1}^{N} \alpha_n y_n \langle \mathbf{X}_n, \mathbf{X}_k \rangle + y_k \tag{15}$$

To solve SVM classification problem with PFGA, we need to design a compatible expression of the algorithm rather than apply the equations and functions mentioned above.

Table 4.1:

Linear SVM classification algorithm.

Algorithm 1: Linear SVM classifier

```
set matrix x=[ Test sample ];
    matrix X=[ Training samples ];
    matrix Y=[ Class identity ];
    matrix B=Y';
for i=1 to n,
for j=1 to n,
multiply matrix X(i,:) by matrix X(j,:)'
set matrix A(i,j)=X(i,:)*X(j,:)';
end
end
for k=1 to n,
    for l=1 to n,
    do multiply matrix A(k,l) by matrix Y(l)
    set matrix A(k,l)=A(k,l)*Y(l);
    end
    end
divide matrix A by matrix B
set matrix C=A\B;
for m=1 to n,
     multiply matrix C(m) by matrix Y(m) and matrix X(m,:)
     set matrix Z(m,:)=C(m)*Y(m)*X(m,:);
     end
set matrix W=[sum of the first column of matrix Z, sum of the second column of matrix
Z];
calculate parameter b=Y(1)-W*X(1,:)';
build the classification function G=W*x'+b
classification result G = Ans
```

$$\begin{cases} \text{sgn}\{G\} = 1, & \mathbf{X} \in C_1 \\ \text{sgn}\{G\} = -1, & \mathbf{X} \in C_2 \end{cases}$$

The rationale behind the design of the SVM classifier is the exploitation of the parallel computational power offered by the FPGA resources, as well as the high memory bandwidth offered by the FPGA internal memories in the most efficient way. As we can see in Algorithm 1, the computation of g(X) involves matrix–vector operations, which can be very complicated using FPGAs during actual calculating procedure. The problem can be divided into smaller segments and parallel units can be used to implement the segments. Therefore, the matrix calculating is performed by processing units, such as adders and multipliers. These processing units perform parallel computations architecture in the design in order to significantly speed up the decision function.

To construct the classifier, the computing modules are needed first. For the linear SVM classifier, as we can see from Table 4.1, matrix calculation is the key issue. Performing matrix calculating will cost a huge amount of computational resource and significantly increase the computation time. Due to this issue, matrix calculations are broken into computing units which can work in parallel in our design, modules like adders and multipliers are designed and assembled to construct the classification system.

4.2.1 Computing Modules for Linear SVM

The computing modules designed for the system are 18 bit signed fixed-point adders and multipliers with 1 sign bit 7 bit before decimal point 10 bit after. A fixed point package is used to build the computing units. The fixed-point math packages are based on the VHDL 1076.3 numeric_std package and use the signed and unsigned arithmetic from within that package. This makes them highly efficient as the numeric_std package is well supported by simulation and synthesis tools. The package defines two new types "ufixed" which is unsigned fixed point, and "sfixed" which is signed fixed point [47].

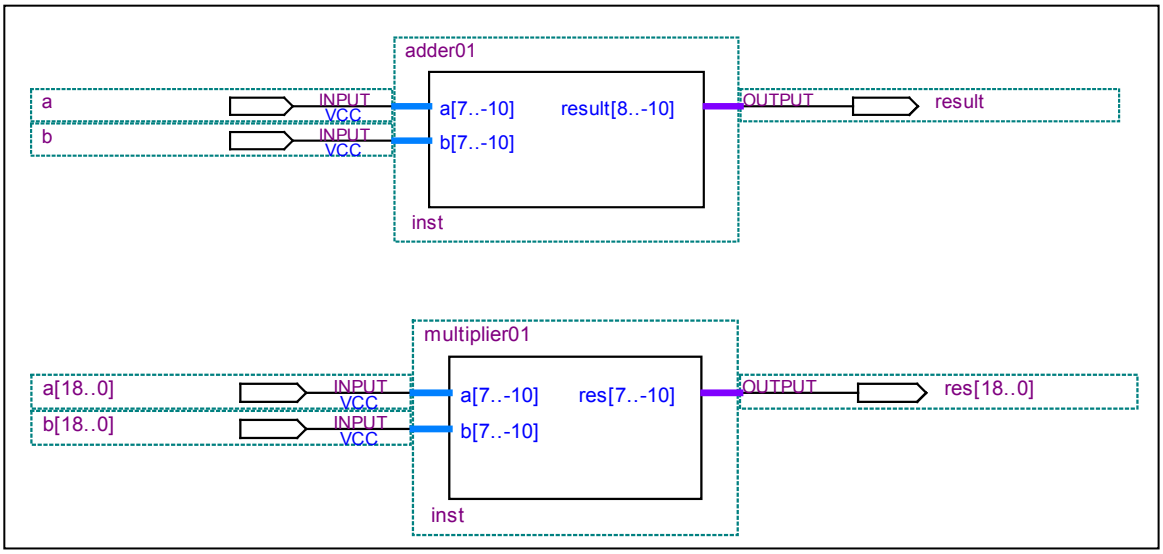

Figure 4-1: Computing modules for linear SVM classifier.

By applying fixed-point package to designed computing units, the computational accuracy is slightly dropped because of the restriction of the fixed bits. Any data with accuracy higher than $1/2^{10}$ can't be process and expressed precisely. A single computing

error for the adder and multiplier can reach to $1/2^9$ and $D/2^9$ separately (where D is one of the input data). Training data sampled in a small area can cause an error which has significant influence on the results. Although the chance of the error mentioned above occurred is very low, by avoiding specific sampling area, this error can be controlled blow 1.5%, integrated with all possibilities, this error can be ignored. Overall, the total computational accuracy is satisfied while the computation resource consumption is limited within an acceptable range.

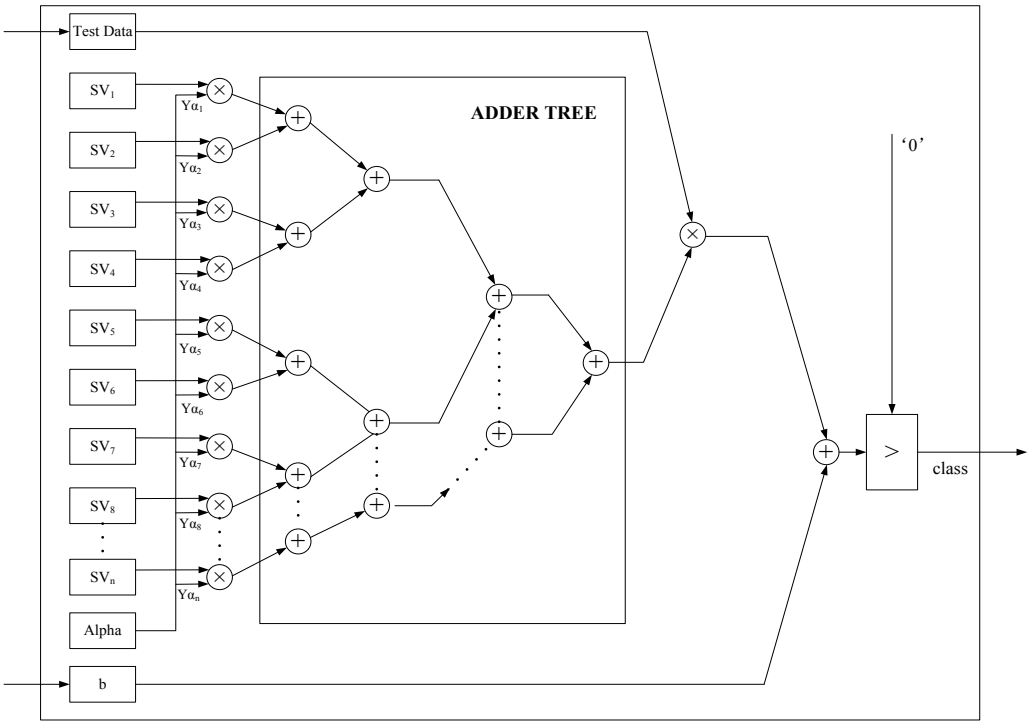

Figure 4-2: Linear SVM classifier architecture.

After appropriate mapping, the assembled computing units will run synergistically to achieve the classification system. Every computing unit on the same hierarchy works in parallel to boost the performance of the system.

The proposed FPGA architecture for the linear SVM classifier is shown in Figure 4-2. The SVs (training datasets) are loaded into the internal FPGA memories, while the classification data points are streamed into the FPGA through the Test data units. The multiply units construct the polynomials of the classification function with training dataset and Lagrange multipliers, and all the multiply units works in parallel. The adder tree in the architecture builds the classification function g(X) with the parameter b and polynomials, which multiply units constructed. After unclassified data is streamed into the FPGA board, classification function g(X) runs a classifying procedure with it based on the training dataset. Results are then sent to the comparing unit and the class of the unknown points can be recognized according to the comparison result. The top level mapping code is showed in Table 4.2 This design uses integer and 18 bit signed fixed point binary data to fulfill the calculation, with 1 sign bit 7 bit before decimal point, and 10 bit after.

Table 4.2:

Top-level Architecture Mapping of Linear SVM classification.

Architecture 1: Linear SVM classifier

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.fixed_pkg.all;
entity fixed_top is
port(sv11,sv12,sv21,sv22,sv31,sv32,
        sv41,sv42,sv51,sv52,sv61,sv62,
        sv71,sv72,sv81,sv82,sv91,sv92,
        sv101,sv102,ya1,ya2,ya3,ya4,ya5,
        ya6,ya7,ya8,ya9,ya10,test1,
        test2,b: in    sfixed(7 downto -10);
        res: out    sfixed(7 downto -10));
end fixed_top;
architecture structure of fixed_top is
component adder01 is
port(a, b: in sfixed(7 downto -10);
        res: out sfixed(7 downto -10));
end component;
component multiplier01 is
port(a, b: in    sfixed(7 downto -10);
        res: out    sfixed(7 downto -10));
end component;
signal a1,a2,a3,a4,a5,a6,a7,a8,a9,a10,
            b1,b2,b3,b4,b5,b6,b7,b8,b9,b10,
            c1,c2,c3,c4,c5,c6,c7,c8,c9,c10,
            d1,d2,d3,d4,e1,e2,f1,f2,g1,g2,
            h1:sfixed(7 downto -10);
begin
u1:multiplier01 port map(sv11,ya1,a1);
u2:multiplier01 port map (sv21,ya2,a2);
u3:multiplier01 port map (sv31,ya3,a3);
u4:multiplier01 port map (sv41,ya4,a4);
u5:multiplier01 port map (sv51,ya5,a5);
u6:multiplier01 port map (sv61,ya6,a6);
```

```
u7:multiplier01 port map (sv71,ya7,a7);
u8:multiplier01 port map (sv81,ya8,a8);
u9:multiplier01 port map (sv91,ya9,a9);
u10:multiplier01 port map (sv101,ya10,a10);
u11:multiplier01 port map (ya1,sv12,b1);
u12:multiplier01 port map (ya2,sv22,b2);
u13:multiplier01 port map (ya3,sv32,b3);
u14:multiplier01 port map (ya4,sv42,b4);
u15:multiplier01 port map (ya5,sv52,b5);
u16:multiplier01 port map (ya6,sv62,b6);
u17:multiplier01 port map (ya7,sv72,b7);
u18:multiplier01 port map (ya8,sv82,b8);
u19:multiplier01 port map (ya9,sv92,b9);
u20:multiplier01 port map (ya10,sv102,b10);
u21:adder01 port map (a1,a2,c1);
u22:adder01 port map (b1,b2,c3);
u23:adder01 port map (a3,a4,c2);
u24:adder01 port map (b3,b4,c4);
u25:adder01 port map (a5,a6,c5);
u26:adder01 port map (b5,b6,c7);
u27:adder01 port map (a7,a8,c6);
u28:adder01 port map (b7,b8,c8);
u29:adder01 port map (a9,a10,c9);
u30:adder01 port map (b9,b10,c10);
u31:adder01 port map (c1,c2,d1);
u32:adder01 port map (c3,c4,d3);
u33:adder01 port map (c5,c6,d2);
u34:adder01 port map (c7,c8,d4);
u35:adder01 port map (d1,d2,e1);
u36:adder01 port map (d3,d4,e2);
u37:adder01 port map (e1,c9,f1);
u38:adder01 port map (e2,c10,f2);
u39:multiplier01 port map (f1,test1,g1);
u40:multiplier01 port map (f2,test2,g2);
u41:adder01 port map (g1,g2,h1);
```

```
u42:adder01 port map (h1,b,res);
end architecture structure ;
```

4.3 Non-linear SVM

Based on the comparison results of accuracy and performance of different Kernel Functions, we applied Gaussian radial basis function: $K(\mathbf{X}\cdot\mathbf{Y}) = e^{-\frac{\|\mathbf{X}-\mathbf{Y}\|^2}{\sigma}}$ in our non-linear SVM algorithm. The classification function constructed for non-linear SVM is:

$$g(\mathbf{X}) = \sum_{n=1}^{N} \alpha_n y_n K(\mathbf{X}_n, \mathbf{X}) - \sum_{n=1}^{N} \alpha_n y_n K(\mathbf{X}_n, \mathbf{X}_k) + y_k \tag{16}$$

4.3.1 Computing Modules for Non-linear SVM

The computing modules designed for the non-linear system are adders, multiplier, square calculators and exp calculators:

1) Adder01 is an 18bit integer adder to add up the results from square calculation modules which shows the square of the difference between support vector and test point.

2) Adder02 is an 18bit 3 ports fixed-point adder; it adds up b and total of the multipliers, and accuracy is $1/2^{14}$.

3) Adder03 is an 18bit 11 ports fixed-point adder; it sums up the results of the multipliers, and accuracy is $1/2^{14}$.

4) Multiplier is an 18bit 3 ports fixed-point multiplier; it multiplies the results of the exp calculator by the parameter $y\alpha$.

4) Square is a square calculation module which shows the square of the difference between support vector and test point.

5) Exp_cal is a table-driven calculation module which shows the calculation result of exp(-x/0.72).

Table 4.3:

Exp calculator design code.

Architecture 2: Exp calculator

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.fixed_pkg.all;
entity exp_cal is
port (a: in integer range 0 to 262140;
b: out sfixed(3 downto -14));
end exp_cal;
architecture behavior of exp_cal is
begin
with a select
b<= "00000000000000000" when 0 ,

```
"0000001111111110101" when 1 ,
"0000000111111111010" when 2 ,
"0000000001111111110" when 3 ,
"0000000000011111101" when 4 ,
"0000000000000111111" when 5 ,
"0000000000000000011" when 6 ,
                    …
"0000000000000000000" when others;

end behavior;
```

The fixed point package is also used to build the computing units. By applying fixed-point package to designed computing units, the computational accuracy can be satisfied while the computation resource consumption is limited within an acceptable range. After appropriate mapping, the assembled computing units will run synergistically to achieve the classification system. Every computing unit on the same hierarchy works in parallel to boost the performance efficiency of the system.

Table 4.4

Non-linear SVM classification algorithm.

Algorithm. 2: Non-linear SVM classifier

set matrix x=[Test sample];
matrix X=[Training samples];
matrix Y=[Class identity];
matrix B=Y';
for i=1 to n,
for j=1 to n,
calculate classification function parameters using matrix X and Gaussian radial basis function and give the result to matrix

A(i,j)=exp(-((X(i,1)-X(j,1))^2+(X(i,2)-X(j,2))^2)/(2*(0.6)^2));
end
end
for k=1 to10,
    for l=1 to 10,
    multiply matrix A(k,l) by matrix Y(l) and give the result to matrix A(k,l);
    end
    end
dived matrix A by matrix B and give the result to matrix C;
for q=1 to 10,
calculate C(q)*Y(q)*exp(-((X(q,1)-X(1,1))^2+(X(q,2)-X(1,2))^2)/(2*(0.6)^2)) and give the result to h(q);
end
calculate the sum of all the elements in matrix h and give the result to matrix H;
calculate parameter b=Y(1)-H;
for p=1 to 10,
calculate matrix
g(p)=C(p)*Y(p)*exp(-((X(p,1)-x(1))^2+(X(p,2)-x(2))^2)/(2*(0.6)^2));
end
calculate the sum of matrix g and give the result to matrix W;
build the classification function G=W+b

$$\begin{cases} \text{sgn}\{G\} = 1, & \mathbf{X} \in C_1 \\ \text{sgn}\{G\} = -1, & \mathbf{X} \in C_2 \end{cases}$$

By replacing the dot product with the Kernel Function, we construct a non-linear SVM classification algorithm. As we can see in Table 4.4–besides the matrix calculation–the exponential function calculation is introduced. Neither fixed point calculation nor floating point calculation for exponential function can be efficiently implemented on FPGA. The advantage of parallel computing cannot be fully played out for massive exp computing. In the proposed algorithm, we designed a table-driven

exponential function calculation module to fulfill the actual exp calculation. The application of the table-driven exponential module saved a large amount of PFGA computing resources, and the accuracy is reliable according to the simulation test. The matrix calculating is also performed in parallel by processing units similar to linear algorithm. The parameter σ in the Gaussian radial basis function is set to 0.6 in our design.

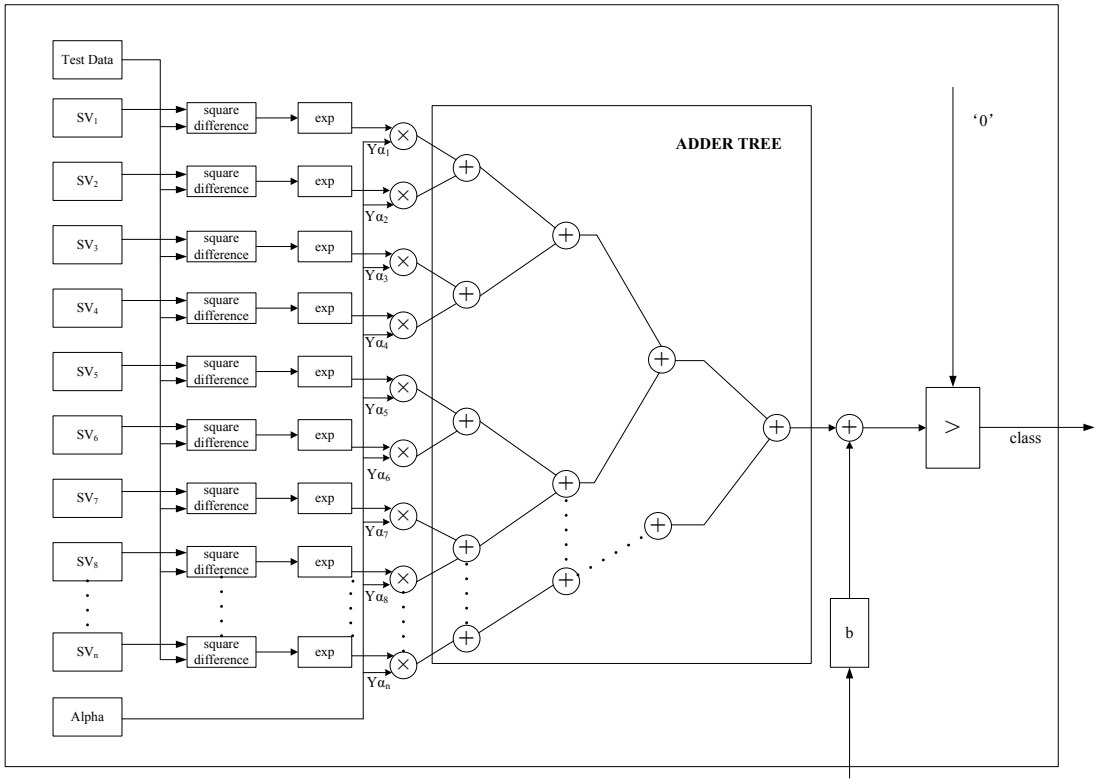

Figure 4-3: Non-linear SVM classifier architecture.

The non-linear SVM classifier FPGA architecture is shown in Figure 4-3. Like the linear design, training datasets are loaded in the internal FPGA memories through SV

units, and then together with test data, they are streamed into square difference units. These units calculate the square difference of test points and training points. Next, results are sent to exp units which perform exponential function calculation in order to achieve the Kernel Function. These exp units work as table-driven modules. Not only they fulfill the exponential function calculation but also the rest of the Gaussian radial basis function including parameters. The top level mapping code is showed in Table 4.5

Table 4.5:

Top-level Architecture Mapping of Non-linear SVM classification.

Architecture 3: Non-linear SVM Classifier

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.fixed_pkg.all;
entity nonlinear_top is
port(sv11,sv12,sv21,sv22,sv31,sv32,
        sv41,sv42,sv51,sv52,sv61,sv62,
        sv71,sv72,sv81,sv82,sv91,sv92,
        sv101,sv102,x1,x2: in integer range 0 to 262140;
        ya1,ya2,ya3,ya4,ya5,ya6,ya7,
        ya8,ya9,ya10,b: in sfixed(3 downto -14);
        res: out    sfixed(3 downto -14));
end nonlinear_top;
architecture structure of nonlinear_top is
component adder01 is
port(a, b: in integer range 0 to 262140;
        res: out integer range 0 to 262140);
end component;
```

```
component multiplier01 is
port(a, b: in    sfixed(3 downto -14);
        res: out    sfixed(3 downto -14));
end component;
component square is
port (a, b: in integer range 0 to 262140;
        res: out    integer range 0 to 262140);
end component;
component exp_cal is
port ( a: in integer range 0 to 262140;
           b: out sfixed(3 downto -14));
end component;
component adder02 is
port(a, b: in sfixed(3 downto -14);
        res: out sfixed(3 downto -14));
end component;
component adder03 is
port(a1,a2,a3,a4,a5,a6,a7,a8,a9,a10: in sfixed(3 downto -14);
        res: out sfixed(3 downto -14));
end component;

signal a1,a2,a3,a4,a5,a6,a7,a8,a9,a10,
           a11,a12,a13,a14,a15,a16,a17,a18,
           a19,a20,b1,b2,b3,b4,b5,b6,b7,b8,
           b9,b10: integer range 0 to 262140;
signal c1,c2,c3,c4,c5,c6,c7,c8,
           c9,c10,d1,d2,d3,d4,d5,d6,d7,d8,
           d9,d10,e1:sfixed(3 downto -14);
begin
u1:square port map(sv11,x1,a1);
u2:square port map(sv12,x2,a2);
u3:square port map(sv21,x1,a3);
u4:square port map(sv22,x2,a4);
u5:square port map(sv31,x1,a5);
u6:square port map(sv32,x2,a6);
```

```
u7:square port map(sv41,x1,a7);
u8:square port map(sv42,x2,a8);
u9:square port map(sv51,x1,a9);
u10:square port map(sv52,x2,a10);
u11:square port map(sv61,x1,a11);
u12:square port map(sv62,x2,a12);
u13:square port map(sv71,x1,a13);
u14:square port map(sv72,x2,a14);
u15:square port map(sv81,x1,a15);
u16:square port map(sv82,x2,a16);
u17:square port map(sv91,x1,a17);
u18:square port map(sv92,x2,a18);
u19:square port map(sv101,x1,a19);
u20:square port map(sv102,x2,a20);
u21:adder01 port map(a1,a2,b1);
u22:adder01 port map(a3,a4,b2);
u23:adder01 port map(a5,a6,b3);
u24:adder01 port map(a7,a8,b4);
u25:adder01 port map(a9,a10,b5);
u26:adder01 port map(a11,a12,b6);
u27:adder01 port map(a13,a14,b7);
u28:adder01 port map(a15,a16,b8);
u29:adder01 port map(a17,a18,b9);
u30:adder01 port map(a19,a20,b10);
u31:exp_cal port map(b1,c1);
u32:exp_cal port map(b2,c2);
u33:exp_cal port map(b3,c3);
u34:exp_cal port map(b4,c4);
u35:exp_cal port map(b5,c5);
u36:exp_cal port map(b6,c6);
u37:exp_cal port map(b7,c7);
u38:exp_cal port map(b8,c8);
u39:exp_cal port map(b9,c9);
u40:exp_cal port map(b10,c10);
u41:multiplier01 port map (c1,ya1,d1);
```

```
u42:multiplier01 port map (c2,ya2,d2);
u43:multiplier01 port map (c3,ya3,d3);
u44:multiplier01 port map (c4,ya4,d4);
u45:multiplier01 port map (c5,ya5,d5);
u46:multiplier01 port map (c6,ya6,d6);
u47:multiplier01 port map (c7,ya7,d7);
u48:multiplier01 port map (c8,ya8,d8);
u49:multiplier01 port map (c9,ya9,d9);
u50:multiplier01 port map (c10,ya10,d10);
u51:adder03 port map (d1,d2,d3,d4,d5,d6,d7,d8,d9,d10,e1);
u52:adder02 port map (e1,b,res);
end architecture structure ;
```

The table-driven module saved a large amount of computing resources by transferring massive floating calculations into a fast table look up mechanism. Due to the mathematical properties of Gaussian radial basis function, the table can be restricted to an acceptable size. The adder tree and multipliers construct the classification functions with parameter b. Class identities of test data will be shown after the comparing unit. The calculating units in the architecture use integer and 18 bit signed fixed point binary data to fulfill the calculation, with 1 sign bit, 2 bit before the decimal point and 14 bit after the decimal point.

Chapter 5

Implementation Results

5.1 Testing Environment Devices

The targeted device for the proposed architecture was the Altera's Cylone II EP2C70F896. The results can be easily expanded to other targeted devices by changing the resource constraints of the design flow. The architecture is captured in VHDL and the fixed-point modules are generated by the Altera tools and packages [48]. The targeted operating frequency is between 200–250 MHz For the testing data, we create 4 random sampling datasets for linear and non-linear SVM classifier separately.

5.2 Result Analysis of Linear Classifier

For linear SVM design, we build 4 different two-class linear datasets (dataset A, B, C, D) to test the classifier. The size of each dataset is 400, 20 of which will be used for SVM training, with a testing size of 50. According to the cumulative results of each

datasets, the test results given by the classification system is satisfying and the consumed time meets our requirements for the design. The results accuracy, however, may be slightly different based on the amount of training data and the density of the support vectors.

Table 5.1：

Testing results of linear models.

	Accuracy	Recognition rate
Model A	99.58%	100%
Model B	99.63%	98%
Model C	99.62%	100%
Model D	99.61%	100%

As we can see from Table 5.1, based on 4 different linear test models, the calculation error is around 0.39% (e_1), and the recognition rate is satisfactory. The recognition rate is related to the training and test datasets we choose, the classification results can vary with the selection of the testing points. Based on the algorithm proposed above and the calculation error obtained from the amount of testing calculation, an expected recognition error (E_1) is inevitable. With the algorithm accuracy Δ_1 ($1/2^{10}$), the expected recognition error can be calculated as $E_1=2(e_1+\Delta_1)$ =0.975%. If applied on a certain amount of test data, the actual recognition error will regress towards E_1. Although the time required for different classification models is unpredictable, it is far easier and more reliable than a PC with a 2.27-GHz Intel i5 duo processer with 3 GB of RAM, for

the computing time is reduced by approximately 30%-50%. Figure 5-1 depicts the detailed comparison in terms of time usage.

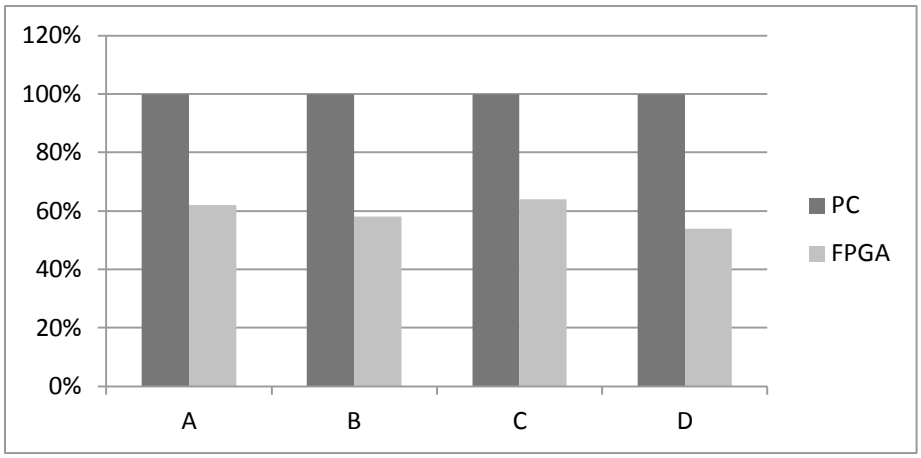

Figure 5-1：Linear SVM time consumption comparison.

5.3 Result Analysis of Non-linear Classifier

For the non-linear SVM classification design, 4 different 400-size nonlinear datasets are built to test the non-linear classifier. The training size for each dataset is 20 and the testing size is 40. The cumulative results are also satisfactory. The calculation error is around 0.041%, which is better than the linear design.

Table 5.2：

Testing results of non-linear models.

	Accuracy	Recognition rate
Model A1	99.95%	97.5%
Model B1	99.96%	100%
Model C1	99.94%	100%
Model D1	99.99%	95%

From Table 5.2, high recognition rates are obtained for the test models and datasets we chose. Like the linear classification, we obtain a 0.041% calculation error (e_2), which we get from the amount of testing calculations and the non-linear algorithm accuracy (Δ_2). The expected recognition rate error for non-linear system (E_2) can be calculated as $E_2=2(e_2+\Delta_2)$ =0.832%. Due to the table-driven exp units we introduced into our design, the time consumption for the non-linear SVM classification system significantly dropped; compared to the 2.27-GHz Intel i5 duo processer with 3 GB of RAM PC, the computing time is reduced by approximately 60%. Figure 5-2 is the detailed time consumption.

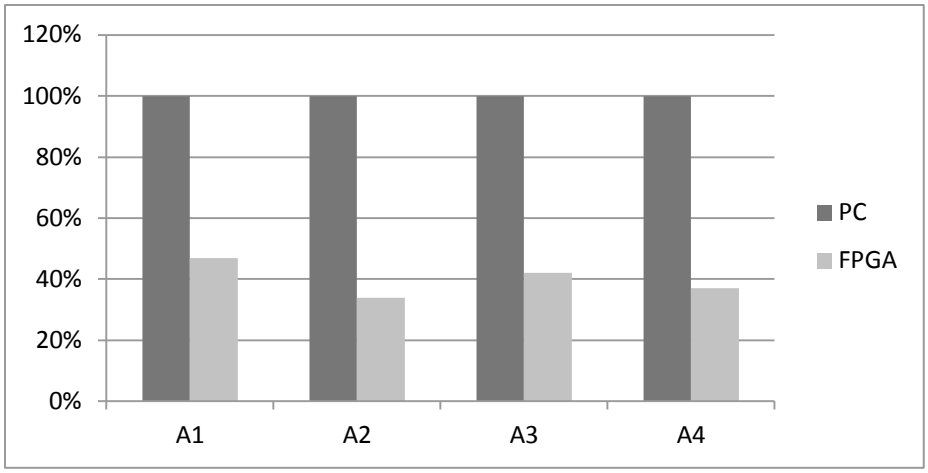

Figure 5-2: Non-linear SVM time consumption comparison.

For the time consumption result showed in Figure 5-1 and Figure 5-2, compare to a PC with a 2.27-GHz Intel i5 duo processer, we can see a marked time consumption reduction with PFGA implementation. As a matter of fact, the operating frequency of

targeted FPGA in this work is between 200–250 MHz, it's about 20 times less than the 2.27-GHz Intel i5 duo processer. The actual time consumption reduction ratio is significantly higher than what we can see from Figure 5-1 and Figure 5-2. Further test can be accomplished by implement SVM on a FPGA board with a higher operating frequency then a PC processor. As one of the latest version of advanced FPGAs, Altera's Stratix 10 devices are manufactured with the revolutionary Intel 14 nm 3D Tri-Gate transistor technology, the targeted operating frequency can reach to over one gigahertz. If we implement our SVM classification system on Altera's Stratix 10 sevice for the future research, with the strong computing power of industry's first gigahertz FPGAs, the real parallel computation advantage of PFGA will be revealed. Using expanded FPGA boards to solve complicated real-world problem with massive computational work is also a feasible way to test the time consumption preference of FPGA based SVM.

Chapter 6

Conclusion and Future work

6.1 Summary and Conclusions

This paper presents a FPGA-based SVM classification system which can be used for fast data classification. The implementation results show that the designed FPGA implementation of SVM classification system works adequately as a fast two-class classification system with a high-accuracy and satisfying computing time. The performance of the SVM classifier as a fast recognition classification system fulfills the proposed requirements. For the further work, it is very promising that smart meters embedded with SVM classifiers can provide fast intrusion detection in order to protect the whole secure communication system, like a firewall.

6.2 Future Work

The SVM classification system is usually applied to cyber-security area. Most of these applications are used as a fast intrusion detector by running a classification procedure with the unknown data. Compare to the traditional SVM implementation, our FPGA-based SVM classification system has broad application space, especially in cyber infrastructure security field such as Smart Grid security. The adaptability and excellent performance offered by FPGA-based SVM classifier provides a feasible solution for Smart Grid security issues.

The core of the Smart Grid is the use of intelligent communication networks as the platform that enables grid instrumentation, analysis, and control of utility operations, from power generation to transmission and distribution. One of the most important foundations of a Smart Grid is the interoperability that enables all of the required devices, technologies, applications, and agents (energy producers, consumers, and operators) to interact in the Smart Grid network. Although Smart Grid communications can assist in transforming the energy industry–such as by playing a critical role in maintaining high levels of reliability, performance, and manageability–they also introduce the need for integrated security infrastructures [49]. It is inevitable that adding digital intelligence and two-way functionality to the power grid will increase the risk of cyber attacks and vulnerabilities like confidentiality, integrity, and availability (CIA) [50]. More endpoints

and interconnected networks mean more ways for security problems to get in and proliferate [51]. Many of the technologies being deployed to support Smart Grid projects such as smart meters, sensors, and advanced communications networks can increase the vulnerability of the grid to attack. [52] In addition, a Smart Grid equipped with intelligent electronic devices cannot survive if the communications infrastructure is insecure and vulnerable to cyber attacks. Devising effective strategies for securing the computing and communication networks that will be central to the performance and availability of the envisioned electric power infrastructure and for protecting the privacy of Smart Grid-related data is our top priority [51]. Due to this issue, a reliable two-way communication solution with security mechanisms regarding to the cyber-physical security of the Smart Grid [53] is extremely important.

For future work, a multilayer, two-way communication network using the FPGA-based SVM classification system will be built in order to secure the privacy and integrity of communications between parties in Smart Grid. The real-time communication ability of the Smart Grid will enable utilities to optimize and modernize the power grid in order to realize its full potential [54]. From Figure 6-1, we can see the whole network combined with electricity suppliers, local maintenance facilities, smart buildings and smart meters. At the top layer of the network, the electrical supplier will gather real-time

power usage report from smart meters, making the most efficiency power plan according to all the data received. Smart meters [55] on the lowest layer play a key role in securing the whole communication network. For consumers, smart meters protect users' privacy by hiding unnecessary personal information from the electricity supplier. In terms of electricity suppliers, all irrelevant information except for the consumption report will be blocked by smart meters just like a firewall to prevent malicious intrusions. This new communication network will be constructed using various communication paths, including fiber optic cable, twisted pair, broadband over power line, and wireless technologies [43]. Smart meters are extremely attractive targets for malicious intrusions, for their vulnerabilities can easily be monetized [52]. There are enormous amounts of communications going through smart meters all the time, and checking the safety of every communication will tremendously delay the whole communication system. In fact, it is also impractical to apply complicated devices and mechanisms on smart meters in order to run a security check with all the communications.

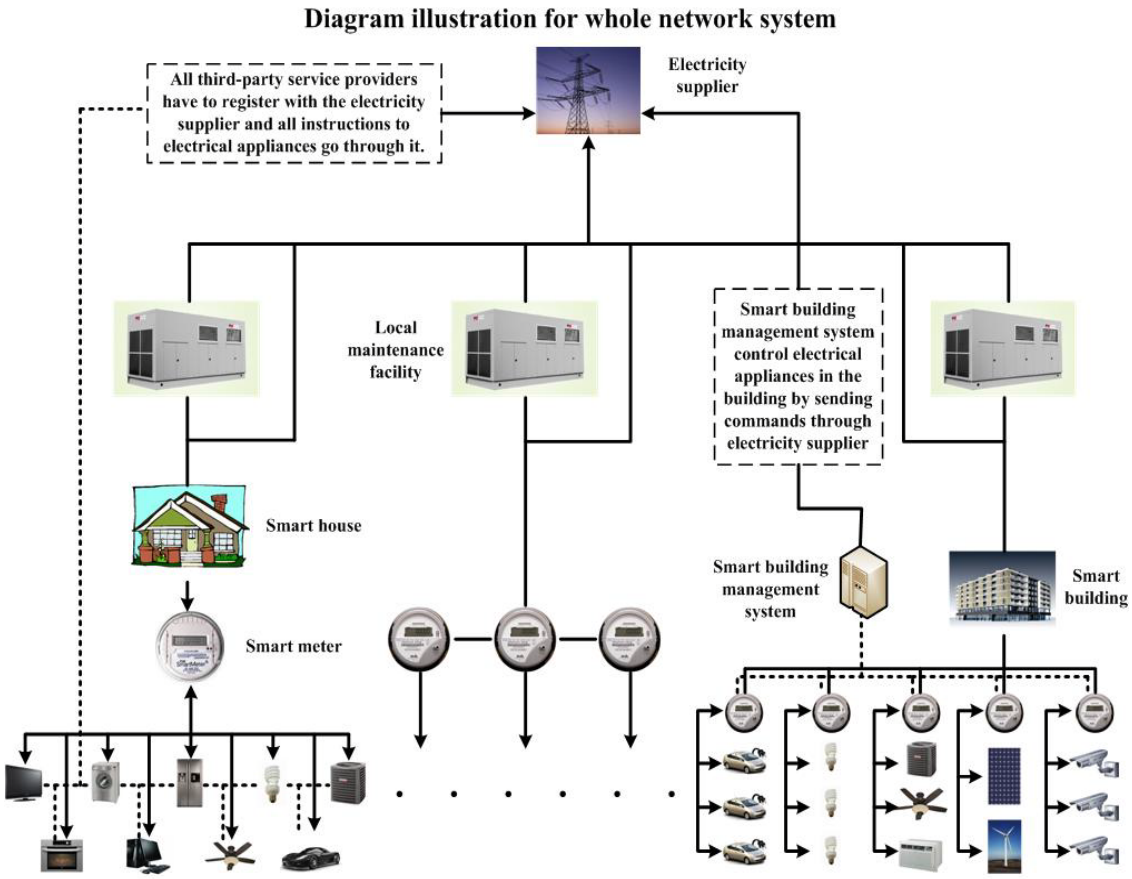

Figure 6-1: Diagram illustration for a multilayer two-way communication network.

To secure smart meters [56] on the lowest level, we then introduce a Support Vector Machine classification system based on FPGA as the security mechanism running on smart meters. The SVM classification system will be integrated into smart meters and work as a firewall for the whole communication system, and after being trained with the datasets that contain known attack types, SVM classification system will look through all the communications with a quick inspection. Based on the data features, SVM classifier will then detect attacks from normal communication. Regular communications are allowed in the network without significant delay, while malicious intrusions and

unknown datasets are sent to the upper levels, such as local maintenance facilities and electricity suppliers to be processed. The time consumption is acceptable. FPGA chips running SVM classifier integrated into Smart Grid is also a feasible choice.

References

[1] **V. Vapnik**, *The Nature of Statistical Learning Theory,* Berlin, Germany: Springer-Verlag, 1995.

[2] **Andrew Moore**, Tutorial of SVM, Available: http://www.cs.cmu.edu/~awm

[3] **Burges C.**, "A tutorial on support vector machines for pattern recognition," In *Data Mining and Knowledge Discovery*, Kluwer Academic Publishers, Boston, 1998, (Volume 2).

[4] **V. Vapnik, S. Golowich, and A. Smola.** "Support vector method for function approximation, regression estimation, and signal processing," In M. Mozer, M. Jordan, and T. Petsche, editors, *Advances in Neural Information Processing Systems* 9, 281– 287, Cambridge, MA, 1997. MIT Press.

[5] **Markos Papadonikolakis, Christos-Savvas Bouganis**, "Novel Cascade FPGA Accelerator for Support Vector Machines Classification," *IEEE Transactions on Neural Networks and Learning Systems*, Vol. 23, No. 7, July 2012.

[6] **S. Dey, M. Kedia, N. Agarwal, and A. Basu**, "Embedded Support Vector Machine: Architectural Enhancements and Evaluation," Proc. *20th Int'l Conf. Very Large-Scale Integration (VLSI) Design*, pp. 685-690, 2007.

[7] **R. Pedersen and M. Schoeberl**, "An Embedded Support Vector Machine," Proc. *Fourth Workshop Intelligent Solutions in Embedded Systems*, pp. 1-11, 2006.

[8] **A. Boni, F. Pianegiani, and D. Petri**, "Low-Power and Low-Cost Implementation of SVMs for Smart Sensors," *IEEE Trans. Instrumentation and Measurement*, vol. 56, no. 1, pp. 39-44, Feb. 2007.

[9] **B. Catanzaro, N. Sundaram, and K. Keutzer**, "Fast Support Vector Machine Training and Classification on Graphics Processors," Proc. *25th Int'l Conf. Machine Learning*, pp. 104-111, 2008.

[10] **M. Papadonikolakis and C.-S. Bouganis**, "A scalable FPGA architecture for non-linear SVM training," in Proc. *Int. Conf. FPT Technol*., Dec. 2008, pp. 337–340.

[11] **S. Cadambi, I. Durdanovic, V. Jakkula, M. Sankaradass, E. Cosatto, S. Chakradhar, and H. Graf**, "A massively parallel FPGA-based coprocessor for support vector machines," in Proc. *17th IEEE Symp. Field Programm. Custom Comput*. Mach., Apr. 2009, pp. 115–122.

[12] **D. Anguita, A. Boni, and S. Ridella**, "A digital architecture for support vector machines: Theory, algorithm, and FPGA implementation," *IEEE Trans. Neural Network.*, vol. 14, no. 5, pp. 993–1009, Sep. 2003

[13] **F. Khan, M. Arnold, and W. Pottenger**, "Hardware-based support vector machine classification in logarithmic number systems," in Proc. *IEEE Int. Symp. Circuits System*., vol. 5. May 2005, pp. 5154–5157.

[14] **K. Irick, M. DeBole, V. Narayanan, and A. Gayasen**, "A hardware efficient support vector machine architecture for FPGA," in Proc. Annu. *IEEE Symp. Field-Programm. Custom Comput*. Mach., Apr. 2008, pp. 304–305.

[15] **M. Ruiz-Llata and M. Yèbenes-Calvino**, "FPGA implementation of support vector machines for 3D object identification," in Proc. *19th Int. Conf. Artif. Neural Netw.* I, 2009, pp. 467–474.

[16] **C. Hsu, M.-K. Ku, and L.-Y. Liu**, "Support vector machine FPGA implementation for video shot boundary detection application," in Proc. *IEEE Int. SOC Conf*., Sep. 2009, pp. 239–242.

[17] **B. Catanzaro, N. Sundaram, and K. Keutzer**, "Fast support vector machine training and classification on graphics processors," in Proc. *25th Int. Conf. Mach. Learn*., 2008, pp. 104–111.

[18] **A. Carpenter**. (2009). CUSVM: A Cuda Implementation of Support Vector Classification and Regression [Online]. Available: http://patternsonascreen.net/cuSVM.html

[19] **NVidia**. (2008). NVIDIA CUDA Compute Unified Device Architecture, Programming Guide, Santa Clara, CA [Online]. Available: http://www.nvidia.co.uk/cuda.

[20] **M. Papadonikolakis, C.-S. Bouganis, and G. Constantinides**, "Performance comparison of GPU and FPGA architectures for the SVM training problem," in Proc. *Int. Conf. Field-Programm. Technol*., 2009, pp. 388–391.

[21] **S. Martin**, "Training support vector machines using Gilbert's algorithm," in Proc. *5th IEEE Int. Conf. Data Mining*, Washington, DC, Nov. 2005, pp. 306–313.

[22] **E. G. Gilbert**, "An iterative procedure for computing the minimum of a quadratic form on a convex set," *SIAM J. Control*, vol. 4, no. 1, pp. 61–80, 1966.

[23] **V. Vapnik and A. Lerner**, "Pattern Recognition Using Generalized Portrait Method Automat," *Remote Contr*., 24, 774–780 (1963)..

[24] **V. Vapnik and A. Chervonenkis**, *Theory of Pattern Recognition*, Nauka, Moscow, Russia, 1974.

[25] **V.Vapnik**, *Estimation of Dependencies Based on Empirical Data*, Nauka, Moscow, Russia, 1979.

[26] **V. Vapnik**, *Statistical Learning Theory*, Wiley-Interscience, New York, 1998.

[27] **C. Cortes and V. Vapnik**, "Support-Vector Networks," *Mach. Learn*, 20, 273–297 (1995).

[28] **B. Scho¨ lkopf, K. K. Sung, C. J. C. Burges, F. Girosi, P. Niyogi, T. Poggio, and V. Vapnik**, "Comparing Support Vector Machines with Gaussian Kernels to Radial Basis Function Classifiers," *IEEE Trans. Signal Process*., 45, 2758–2765 (1997).

[29] **O. Chapelle, P. Haffner, and V. N. Vapnik**, "Support Vector Machines for Histogram-based Image Classification.," *IEEE Trans. Neural Netw.*, 10, 1055–1064 (1999).

[30] **H. Drucker, D. H. Wu, and V. N. Vapnik**, "Support Vector Machines for Spam Categorization ," *IEEE Trans. Neural Netw.*, 10, 1048–1054 (1999)..

[31] **V. N. Vapnik**, "An Overview of Statistical Learning Theory," *IEEE Trans. Neural Netw.*, 10, 988–999 (1999).

[32] **V. Vapnik and O. Chapelle**, "Bounds on Error Expectation for Support Vector Machines," *Neural Comput*., 12, 2013–2036 (2000)..

[33] **I. Guyon, J. Weston, S. Barnhill, and V. Vapnik**, "Gene Selection for Cancer Classification Using Support Vector Machines," Mach. Learn., 46, 389–422 (2002).

[34] **Wikipedia**, Field-programmable gate array, [Online] Available: https://en.wikipedia.org/wiki/Field-programmable_gate_array.

[35] **Introduction to FPGA and Verilog Programming,** [Online] Available: http://coep.vlab.co.in/?sub=29&brch=88&sim=228&cnt=1

[36] **Ian Kuon, Russell Tessier and Jonathan Rose** "FPGA Architecture: Survey and Challenges," *Foundations and Trends in Electronic Design Automation* Vol. 2, No. 2 (2007) 135–253.

[37] **Altera**, FPGAs, [Online] Available: http://www.altera.com/products/fpga.html

[38] **Christos Kyrkou, Theocharis Theocharides**, "A parallel hardware architecture for real-time object detection with support vector machines," *IEEE Transactions on Computers*, vol. 61, no.6, June 2012.

[39] **G. Garg, Vijander Singh, Mudita Grover, Nidhi ,J.R.P Gupta.** "Optimal Kernel

Learning for EEG based Sleep Scoring System," *International Journal of Biological & Medical Research* 2011; 2(4): 1220 – 1225.

[40] **Vikramaditya Jakkula**,Tutorial on Support Vector Machine (SVM) [Online] Available : www.cs.columbia.edu/~kathy/cs4701/documents/jason_svm_tutorial.pdf.

[41] **Duda R. and Hart P.**, *Pattern Classification and Scene Analysis*, Wiley, New York 1973.

[42] **Nello Cristianini and John Shawe-Taylor**, *An Introduction to Support Vector Machines and Other Kernel-based Learning Methods*, Cambridge University Press, Boston, 2000.

[43] **Daintree Networks.** (2007). what's so good about mesh networks? [Online]. Available: http://www.daintree.net/downloads/whitepapers/mesh-networking.pdf.

[44] **C.J.C. Burges**, "A Tutorial on Support Vector Machines for Pattern Recognition," *Data Mining and Knowledge Discovery*, vol. 2, no. 2, pp. 121-167, 1998.

[45] **J. Mercer**, "Functions of positive and negative type and their connection with the theory of integral equations," *Phil. Trans. Royal Soc.* London, vol. 209, nos. 441–458, pp. 415–446, 1909.

[46] **B. Scholkopf and A. J. Smola**, *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond. Cambridge*, MA: MIT Press, 2001.

[47] **David Bishop**, "Fixed- and floating-point packages for VHDL 2005," Eastman Kodak Company, Rochester, NY.

[48] **M. Langhammer**. "Floating point datapath synthesis for fpgas," In FPL'08, 2008.

[49] **P. McDaniel and S. McLaughlin**. "Security and Privacy Challenges in the Smart Grid," *IEEE Security and Privacy*, 2009, May. 7(3)**.**

[50] **C. Bennett, and D. Highfill**, "Networking AMI Smart Meters," Energy 2030 Conference, 2008. ENERGY 2008. IEEE, 17-18 Nov. 2008.

[51] **NIST**, NIST Framework and Roadmap for Smart Grid Interoperability Standards, Release 2.0. [Online] Available: http://www.nist.gov/smartgrid/framework-022812.cfm.

[52] **P. McDaniel, S. W. Smith**, "Security and Privacy Challenges in the Smart Grid." [Online]. Available: http://www.patrickmcdaniel.org/pubs/sp-smartgrid09.pdf.

[53] **DOE**, The Smart Grid: An Introduction [Online]. Available: http://energy.gov/oe/downloads/smart-grid-introduction-0.

[54] **S. Keemink, B.Roos**. (2008) Security analysis of Dutch smart metering systems, [Online]. Available: http://staff.science.uva.nl/~delaat/sne-2007-2008/p33/report.pdf.

[55] **F. Cleveland**, "Cyber security issues for advanced metering infrastructure (AMI)," *2008 IEEE Power and Energy Society General Meeting - Conversion and Delivery of Electrical Energy in the 21st Century*, pp. 1–5, July 2008.

[56] **WirelessMess**, An Introduction to Smart Meters, [Online]. Available: http://wirelessmess.org/introduction-to-smart-meters/