POLITECNICO DI TORINO

Collegio di Ingegneria Elettronica, delle Telecomunicazioni e Fisica (ETF)

Master of Science in Communications and Computer Networks Engineering

Master Degree Thesis

Trace-buffer Based Electric Bug Localization

Supervisor
Prof. Giovanni Squillero

Candidate
Jiao Li
student no. 224739

KIT Supervisor
Karlsruher Institut für Technologie
Chair of Dependable Nano Computing (CDNC)
Prof. Dr. Mehdi Baradaran Tahoori
KIT Tutor
M.Sc. Eng. Arunkumar Vijayan

April 2018

Abstract

The aim of this thesis is to collect workload data based on simulations and predict run-time parameters based on the machine learning algorithms. Targeting at investigating the upcoming false-behavior of a chip.

This thesis is composed by five main parts which are:

- the **introductory chapter**, highlighting background, key terms, why a new trace-buffer based electrical bug localization method is needed, the goal of this work, outlining the constraints and issues which are generated mainly by previous studies;
- the second chapter describes which methodology is used in our proposal and a brief overview is given in the **program structure** section in order to better understand this work;
- in the third chapter, the most important one, **benchmark circuit selection** is presented firstly with a comparison of ISCAS'99, ITC'99 and LEON3 processor, then the whole procedures of generating experimental data sets are illustrated, starting from **experimental setup** to the **final training dataset** generation with the Leon3 microprocessor; we focus on the **model selection** which concentrates on false behavior prediction using machine learning in the last section of this chapter, explaining the machine learning algorithm which is used in this thesis and its peculiarities that make it different from other machine learning algorithms and previous works; the unbalanced class problems are also illustrated;
- in the following, the fourth chapter focuses on the **evaluation of the final results and performance**; the experimental results generated by using the same workload data-set and by using different workloads both for training and testing the mode are demonstrated and both electrical bugs localization and the trace signal selection are considered;
- the **conclusions** of this thesis is presented in the last chapter.

We could observe from the experimental result that the trace buffer based electrical debugging using machine learning techniques is efficient than the traditional methods. After considering both accuracy score for locating the electrical bugs and the trace signal selection, RFC genrated the best result.

Acknowledgements

I would like to thank my family for the love and support.

Key Words Post-silicon validation, machine learning, trace-buffer, electrical bug localization.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Preliminary Introduction and Motivations

In pre-silicon verification, the formal verification and simulation-based verification are two major methods. Since the complexity of modern electronic systems increased rapidly and this makes the traditional pre-silicon verification, which detects and fixes bugs before manufacturing, becomes inefficient for detecting the functional bugs and some electrical bugs. It makes post-silicon debugging become into existence and become more important nowadays.

The expeditiously increased complexity of modern electronic systems also makes the effectiveness of the debug strategy to become more important. The previous studies and works provided an overview of various post-silicon validation activities, techniques, and the corresponded issues.

Post-silicon validation is critical for integrated circuit designing. Increasing the abilities of control and observation of the integrated circuits' internal behavior level is required in order to be able to find the root-cause design bugs. And this level is much higher than what the manufacturing test generally needs.

In the modern complex integrated circuits, excessive power dissipation may cause run-time errors and the device destruction due to overheating, while device lifespan may shorten by reliability issues. Therefore monitoring the states, which corresponded to the code execution, timing, and data accesses, in order to monitor the electrical bug become critical. But storing all system states is not feasible.

After satisfying the critical path and avoid redundant information. A subset of traceable signals are effective and could be used for debugging in run-time, which

introduces the trace-buffer based debugging technologies.

Machine learning shows its excellent performance in many real-world problem classes in recent years, many machine learning applications were developed successfully. Let computer learn from historical experiences then automatically improve its upcoming choice decision based on learning models or learning algorithms, which takes my interests to combine it with electrical debugging.

1.2 Goal of the Project

This work aims to collect workload data based on simulations and predict run-time parameters based on the machine-learning model built on these data, targets at exploring the electrical bug detection capabilities of a post-silicon debugging framework by marking informed decisions on trace signal selection by analyzing the trace buffer data to extract voltage droop profile of the power delivery network (PDN) for bug localization and detection.

1.3 Background, Previous Studies and Key Terms

1.3.1 Post-silicon Validation

Since the traditional pre-validation is nearly impossible to detect and fix all bugs before manufacture due to sheer design complexity. As stated in section 1.1, it brings post-silicon debugging into existence and makes it become more important nowadays.

Post-silicon validation has significant overlap with pre-silicon design verification and manufacturing (or production) testing, it is the process which the manufactured design, for example, a chip is tested for all functional correctness at run-time, encompassing all the validation efforts that are poured onto a system after the first several prototypes are manufactured in a lab setup, but before the final product release. As a result, a number of functional bugs survived into manufactured silicon, detecting and diagnosing them is the job of post-silicon validation, so that they do not escape into the released system.

A simplified overview of pre-silicon and post-silicon validation flow is demonstrated in the Fig.1.1 below.

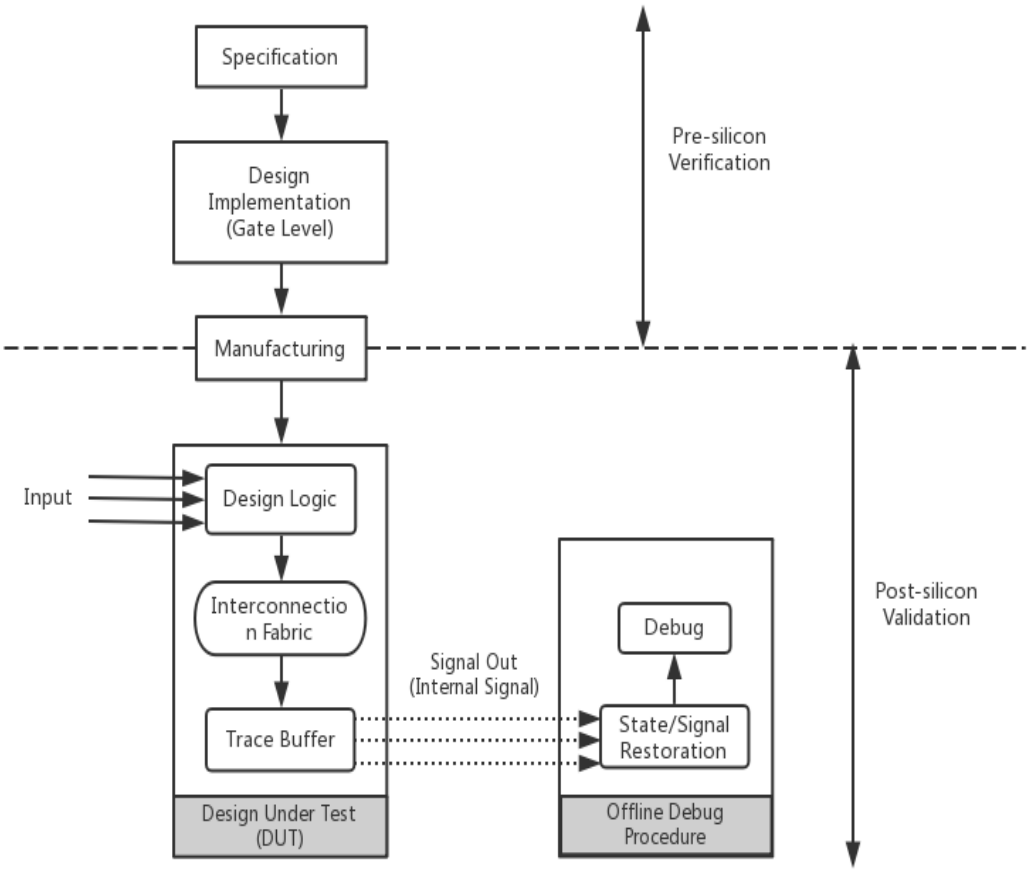


Figure 1.1: Simplified overview of pre-silicon validation and post-silicon validation flow

In the pre-silicon validation stage, as showing in the Figure 1.1, the debugging process remains mainly manually and the debugging time grows significantly with the increasing complexity of the integrated circuits. There are different approaches for pre-silicon debugging automation, like *Binary Decision Diagrams* (BDD) [4], and *Boolean Satisfiability* (SAT) [5]. The debugging approaches based on SAT showed their robustness and effectiveness in a variety of design scenarios from diagnosis to debugging properties. And the design bugs at gate level are divided into three main classes: logic bugs, algorithmic bugs, and synchronization bugs [6].

The post-silicon validation contains two parts mainly, *Design Under Test* (DUT) and *Offline Debug Procedure* (ODP). Logic synthesis is applied after verifying the design against the specification and fixing the bugs in the design, it converts the

design to a gate level circuit, and the new circuit is checked again against the design, which is called equivalence checking. The resistor-transistor level (RTL) design is usually created by a place-and-route process for chip manufacturing.

Signals selected through pre-silicon analysis are funneled to trace buffer through interconnection fabric from which the states are restored offline to assist in debugging. Then the design is fabricated in silicon as a chip.

1.3.2 PDN and Electrical Bug

Power Delivery Network (PDN) includes all the involved devices and wiring which distributed across the die and packages of the chip and discrete components, it distributes power and ground voltages from pad locations to all devices in a design over a network of conductors. The design of PDN has to consider many different operating modes resulting from clock gating, power gating, multiple power domains, dynamic voltage, and frequency-scaled operation and also has to be implemented using fewer routing layers and cheaper packages due to cost consideration [7].

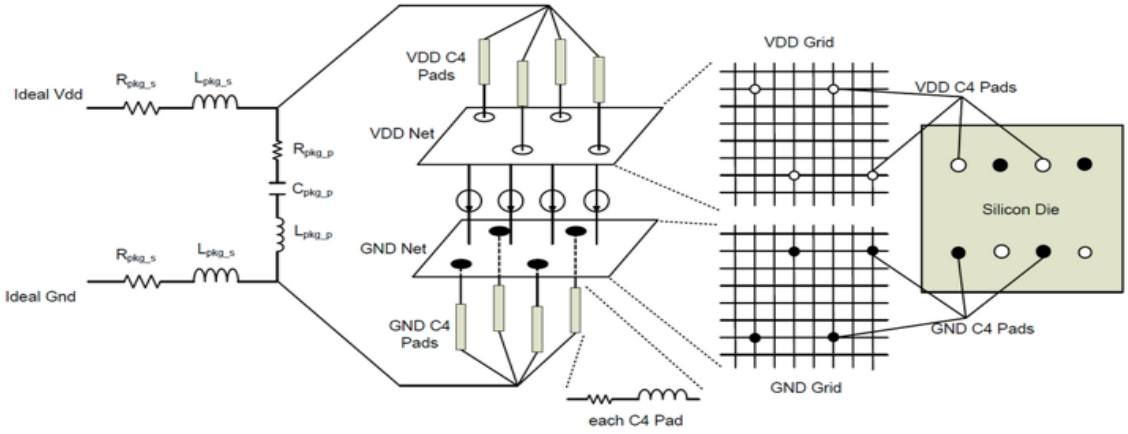


Figure 1.2: PDN example: on-chip model [1]

The power dissipation of designed chips has grown rapidly with the increased complexity of modern integrated circuits. For reducing the power dissipation, the operating voltage is scaled down, therefore shrinking the voltage drop which is allowed on the chip power and ground distribution for the correct operation of the circuits. For high-performance processors, since it needs to deliver a very large amount of power, the issues and related challenges become obvious.

In a digital circuit, most of the power in CMOS circuits is consumed during charging and discharging of the load capacitance during each clock cycle, it means gates consume power when making logical transition.

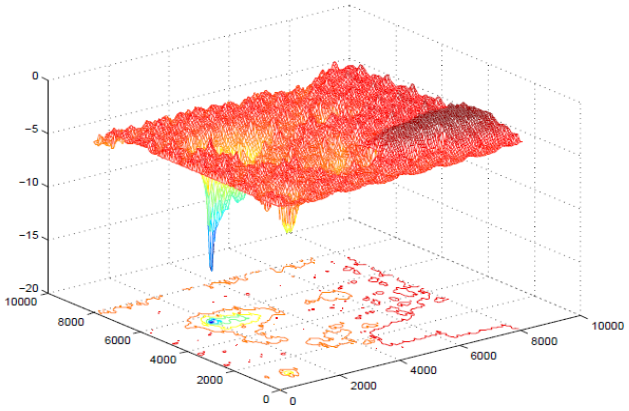


Figure 1.3: Voltage drop contour plot. Z-axis is the percentage change [2]

Most of electric bug due to systematic design problems and it may lead the chip to generate incorrect outputs. The majority of failures caused due to electrical bugs, are timing failures. For instance, the PDN is not well designed and generates high voltage droop in some region, as shown in Figure 1.3, which means the increased delay across the circuit is caused by the percentage of V_{dd} reducing. They are hard to test and they only happen when the chip is running under specific workload scenarios.

1.3.3 Trace-buffer

Digital trace usually provides the detailed history of states which represents code execution, timing, and data accesses. These information could be used for debugging and performance analysis. It captures states in run-time but does not impact the execution of electronic system.

A simplified view of how the trace buffer works, is shown in Figure 1.4 below.

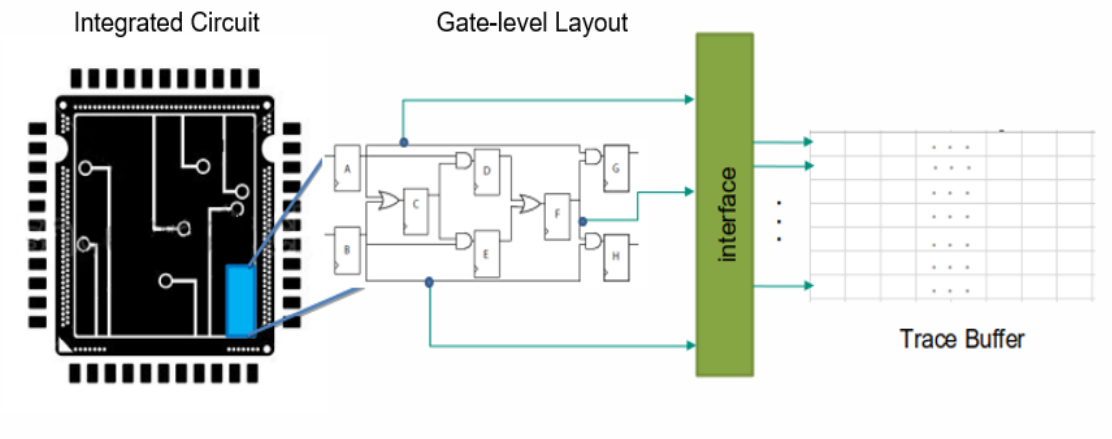


Figure 1.4: Example of trace-buffer

Embedded Trace Buffer (ETB) is an on-chip circular memory buffer, as shown in figure 1.5, whose physical size is usually limited between two thousand and eight thousand cycles, which means it could trace few hundred signals for few thousand cycles in a design with millions of signals and continuously stores the traced information.

The circular trace buffer, as shown in Figure 1.5, is also applied in our proposal. It connected to a subset of FFs and always stores the last N cycle contents of these traced FFs, it keeps overwriting itself as a circular queue.

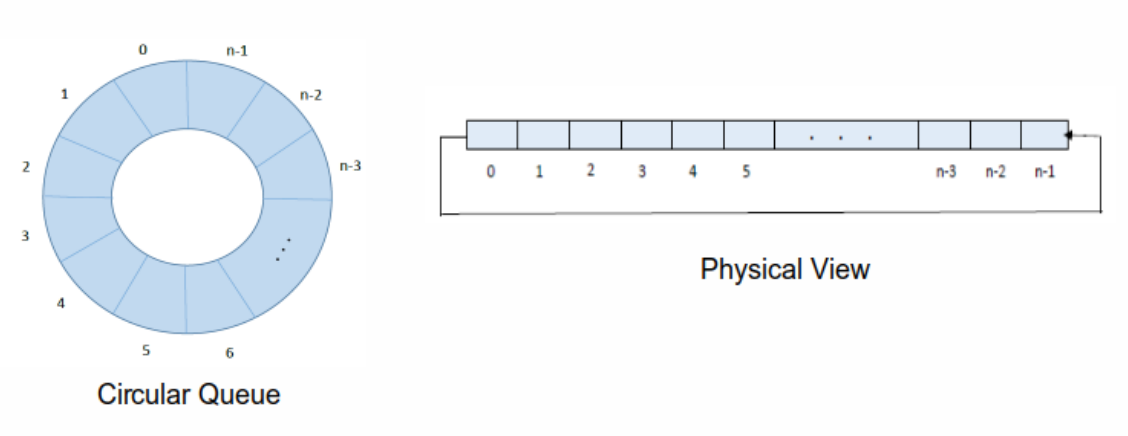


Figure 1.5: Circular buffer

Usage of the trace buffer for run-time debugging reduces the complexity and difficulty compared to the traditional debug methods, which usually requires to stop

and re-start the electronic system. Obviously, it can also reduce the amount of time needed to find these failures significantly. The number of signals that can be traced per cycle is limited to trace buffer width. The trace buffer depth will limit the maximum number of values recorded per signal [8].

1.3.4 SA, SP and SR

SA : *Switching Activity.*
It plays a significant roll in the integrated circuits as what is already mentioned in section 1.3.2, gates consume power when making logical transition, so the power dissipation depend on the extent of circuit switching activity strongly [9], [10]. And in paper [11], the following equation is provided for calculating dynamic power:

$$P_d = \alpha f_c C V_{dd}^2$$

where α stands for switching activity or signal activity of an actual node, f_c is the clock frequency and V_{dd} is the supply voltage.

The clocked nodes have highest switching activities with $\alpha = 1$ in general, it is the number of clock cycles for each cycle. Therefore the clock signals are very critical for dynamic power consumption. And for data, different activities depend on different workloads. Random data have an activity of 0.25, and in the matter of fact, the real data normally change between 0.01 and 0.25 [11].

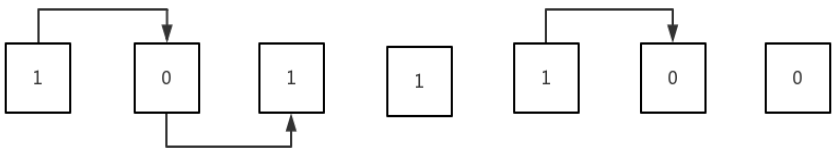


Figure 1.6: Switch Activities example

In another words, for example a sequence of states showed in the Fig. 1.6, only the change between adjacent states, like 1 to 0 or 0 to 1, counted as a switch.

SP : *Signal Probability (Logical Signal Probability).*
In paper [12], " 1 " was chosen as the "reference value". The definitions of the probability of a logic signal and the probability that a signal equal to zero are given as Eq.1.1 (Definition 1) and Eq.1.2 (Definition 2) in the following.

Definition 1: The probability of a (logic) signal, expressed as

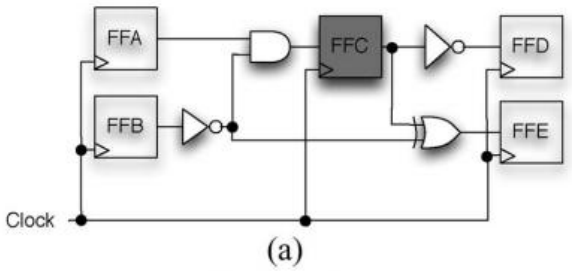
$$a = P(A = 1) \tag{1.1}$$

"for signal A, is a real number on the interval [0, 1] which expresses the probability that signal A equals 1. We use the convention here that upper case letters correspond to signal names (Boolean variables) and lower case letters represent the corresponding probabilities. Since Boolean algebra is based on two-valued variables, we give the following definition."

Definition 2: The probability that signal A = 0 is given as

$$P(A = 0) = 1 - P(A = 1) = 1 - a \tag{1.2}$$

Several lemmas which relate Boolean operations to corresponding operations on probabilities and their proofs can be found in [12].
Sets of rules for calculating the node signal probabilities in different cases with different methods were summarized in tables in [13].



(a)

Clock cycles

	0	1	2	3	4
FFA	1	1	X	X	X
FFB	0	0	X	X	X
FFC	0	1	1	0	X
FFD	X	1	0	0	1
FFE	X	1	0	X	X

(b)

Figure 1.7: Sample circuit for state restoration. (a) CUD. (b) Restored data in sequential elements.

SR : *State Restoration (Ratio).*
State restoration ratio was mentioned firstly in [14], and a simple example was considered as shown in the Figure 1.7.

The easiest way for debugging the circuit which is showed in Fig.1.7(a), all five *flip-flops* (FFs) were sampled for five consecutive clock cycles, as shown in Figure 1.7(b). For storing all the data during silicon debugging in this case, a size of 5×5 trace buffer was used, it means only the sampled flip-flops is needed to be monitored instead of all the signals, and knowing the trace of one FF will be able to reconstruct some of the missing data of the circuit. How the states restoration works is shown in the following.

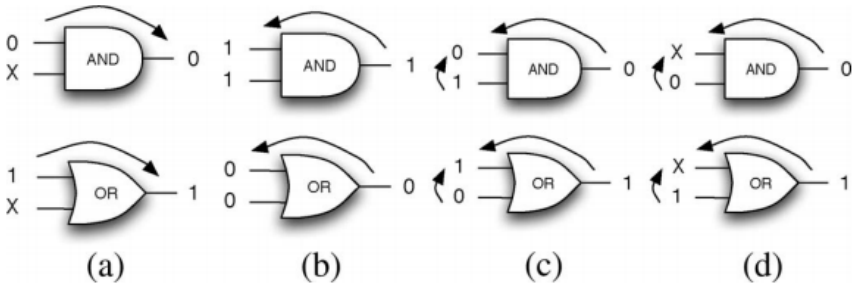


Figure 1.8: Principal operations for state restoration. (a)Forward. (b)Backward. (c)Combined. (d)Not defined.

Forward restoration: reconstructing the output when knowing of inputs.
Backward restoration: restoring the unknown inputs when knowing the output.

The main idea of state restoration is to forward propagate and backward justify the known values from a trace signal to other nodes in a circuit in order to reconstruct the missing data. This is achieved by applying the Boolean relations between state elements [14], the principal operations for SR is demonstrated in the Figure 1.8 as showing above, both forward restoration, backward restoration, combined case and not defined case are illustrated.

Singal or *state restoration ratio* (SRR) is a common used metric for measuring the signal restoration and the quality of trace signal selection, which was defined in [15] with more detailed explanations, is also presented in the following.

$$SRR = \frac{N_{Traced\_states} + N_{Restored\_states}}{N_{Traced\_states}}$$

1.4 Machine Learning

1.4.1 Introduction and Glossaryon Machine Learning

Machine learning is used firstly by Arthur Samuel who is expert in the field of computer gaming and artificial intelligence in 1959 [16]. It is one of the fastest growing fields recent years in computer science. Its wide usage covers not only science but also the daily life with the rapidly growing of the digital technologies.

The amount of data become large thus capturing key features and useful information from the data or mapping input to output became less trivial. But the certain patterns do exist in the data. Algorithms are needed in order to solve these problems on the computer, so-called learning algorithms in machine learning area, which contains a sequence of instructions that should be implemented to transform the input to output.

Machine learning is a subject which is devoted to study how to use the experiences to improve the performance of system through calculation methods. The principle of machine learning is generating *learning models*, which could automatically improve or optimize the performance with given data through *learning algorithms*, then it could make accurate *predictions*. And the given data are usually described as the *experiences*, here refers to the historical information or data which are available to the learner. The learning model could provide the accurate prediction after learning when new experience or data is given to it.

The process that generating the learning model from data, is called *learning* or *training*. The data-set is used in the learning process is called *training data* and each sample inside that is called *training sample*. And all the gathered training samples together is called *training set*. The learned model which corresponded to the potential patterns or regularities of the data is called *hypothesis*, and these regularities are so-called *ground-truth*. The learning process is to find out or approximate the ground-truth. The 'result' information is needed in order to build a model which is possible to make the prediction.

$$\begin{aligned} Error\ Rate &= E(f;D) \\ &= \frac{1}{m}\sum_{i=1}^{m} I(f(x_i) \neq y_i) \\ &= \frac{n}{m} \end{aligned} \tag{1.3}$$

The total number of wrong samples over the whole sample set after classification is called *error rate*. For example, there are m samples in total and n of them were

classified wrongly. Thus, the error rate [17] could be calculated as in eq. 1.3. And accordingly, the accuracy is:

$$\begin{aligned} Accuracy &= 1 - Error\ Rate \\ &= 1 - E(f;D) \\ &= \frac{1}{m}\sum_{i=1}^{m} I(f(x_i) = y_i) \\ &= 1 - \frac{n}{m} \end{aligned} \tag{1.4}$$

where f stands for the leaner performance, *loss function* I is one when $f(x_i)$ and y_i are equal, is zero otherwise, and D is the given sample set with$m$ samples which could be represented as following.

$$D = \{(x_1, y_1), (x_2, y_2), ..., (x_m, y_m)\}$$

In general, the difference between the predictions and the actual output is called *error*, the error made by learner on the training set is *training error* or *empirical error*, and the errors made on the new samples is *generalization error*. Obviously, a learner with small generalization error is wanted. However, the new samples are unknown in advance, so what actually is tried to do is to minimize the empirical error.

Error rate and accuracy are used the mostly but can not satisfy all the task requirements, therefore also other performance measurements like *precision* and *recall* [18] were introduced. According to the real output set and predicted output set, the samples could be classified as *True Positive* (TP), *False Positive* (FP), *True Negative* (TN) and *False Negative* (FN). And they could be represented in a so-called *confusion matrix* as showed in the table 1.1.

Actual	Prediction Results	
	Positive Class	Negative Class
Positive Class	TP	FN
Negative Class	FP	TN

Table 1.1: Confusion matrix

Therefore the precision and recall could be calculated as in eq.1.5 and eq.1.6.

$$Precison = P = \frac{TP}{TP + FP} \tag{1.5}$$

$$Recall = R = \frac{TP}{TP + FN} \tag{1.6}$$

And obviously, $TP + FP + TN + FN = Total_{samples}$.

Using the learning algorithms to build up a good and useful approximation in order to detect certain patterns or regularities instead of all is the niche of machine learning. Machine learning plays an important role in transforming the digital data into useful products and services, and it shows its excellent performance in many real-world problem classes and many machine learning applications were developed successfully in recent years.

1.4.2 Problem Catalog

According to the training data-set is labeled or not, the learning tasks could be separated mainly into two kinds, *supervised learning* and *unsupervised learning*. The learning tasks could be classified as showed in Figure 1.9.

- **Supervised Learning / Predictive models**
  Predictions are made for the future outcome based on the historical data. Normally, what needs to be learned and how it needs to be learned is given from the beginning. Some examples of algorithms used are: Nearest neighbour, Naive Bayes, Decision Trees, Regression etc.

  - **Semi-supervised learning** only an incomplete training signal, which means a training set with some missing target outputs, is given to the machine.
  - **Active learning** only the training labels are obtained by the machine for a limited set of instances, and also has to optimize its choice of objects to acquire labels for.
  - **Reinforcement learning** training data (in form of rewards and punishments) is given only as feedback to the program's actions in a dynamic environment [19], and usually the machine is trained to take specific decisions based on the business requirement to maximize efficiency or performance, it is an example of machine learning. An example of algorithm used in reinforcement learning is Markov Decision Process.

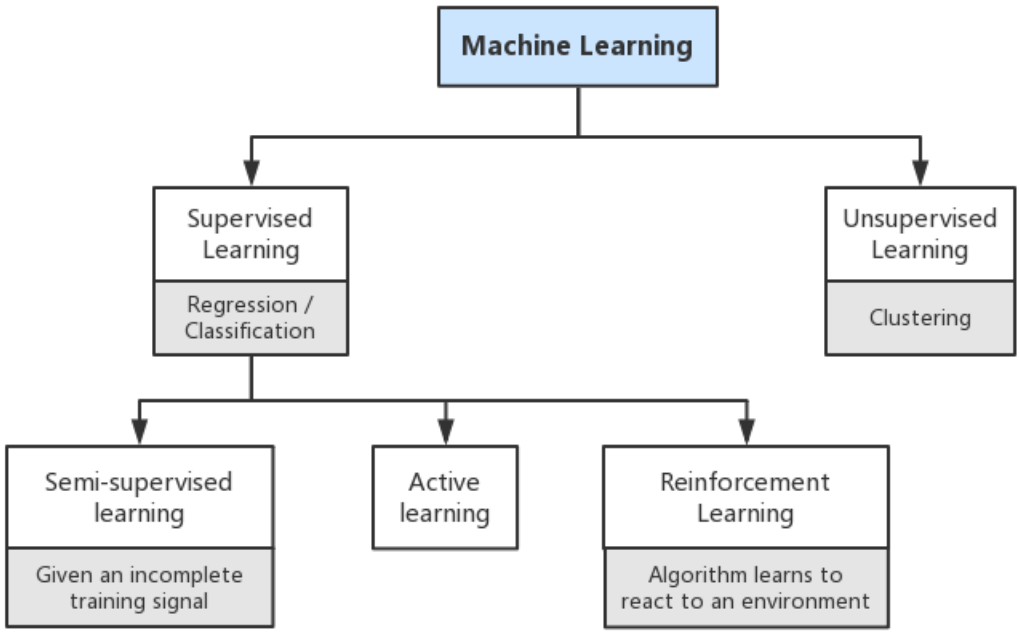


Figure 1.9: Machine Learning problem classification

- **Unsupervised learning / Descriptive models**
  It is used to train descriptive models where no target is set and no single feature is important than the other. Example of algorithm used here is: K-means Clustering Algorithm etc.

If the discrete values, like 'yes' or 'no' or other labels with a 'name', are going to be predicted, this type of learning tasks is called *classification*. If the continues values, like 0.95, 0.33, without giving any labels, are going to be predicted, this type of learning tasks is classified as *regression*. When the learning task only involves two different classes, it is binary classification problem, and usually, one class is called positive class and another one is called negative class; when several classes are involved, it is usually a *multi-class classification* problem. There is also *clustering*problem which means separating the training set into different groups and each group is called a *cluster*. And these auto-grouped clusters may correspond to some potential conceptual divisions. Classification and regression problem belongs to the first kind and clustering belongs to the second kind.

1.4.3 History and Previous Studies

From 1950s to early 1970s, *artificial intelligence* (AI) already existed as an academic discipline, which targeted on letting machines learn from data. Several approaches were mentioned includes e.g., neural networks. The probabilistic reasoning was also employed [20]. Machine learning was the inevitable product of the development of AI to a certain stage. It was reorganized as a separated field later, the goal was changed from achieving AI to solving practical natured problems.

In the 1980s, one of the mainstreams of "learning from samples" was the study of logic and symbolic reasoning, whose representatives included decision trees and logical learning. The typical decision tree learning based on information theory, with the objective of minimizing the entropy of information, directly simulated the decision making of concepts by people in tree process. The most famous representative of logical learning is *Inductive Logic Programming* (ILP), it could be regarded as an intersection of machine learning and logic programming. Another mainstream is connectionism based on neural networks, and unlike the symbolism, it generated the black-box model.

In the mid-nineties of the twentieth century, statistical learning became the mainstream of study, the most representative is *Support Vector Machine* (SVM) and kernel methods. And early twenty-first centuries, the connectionism became popular again, especially deep learning which could be regarded as neural networks with many layers in a narrow sense.

Same methods are employed in both machine learning and data mining, the overlapping is significant. The main difference between them is that they have different goals. Machine learning focuses on prediction and data mining focuses on the discovery of the unknown properties in the data. Some of the machine learning problems could be described as optimization problems since they are formatted as the minimization of some loss function on a training set. The difference between the prediction problem and optimization problems in machine learning is that they have the different goal of generalization. Optimization algorithms minimize the loss on a training set, machine learning is concerntrated on minimizing the loss of unseen samples [21].

Recent years, there are many successful applications of machine learning in various fields, for example, statistics, AI, speech and handwriting recognition in commercially available systems and network optimization or signal processing in the telecommunications area, etc..

Machine learning was already used in the post-silicon debugging stage. A learning-based signal selection approach was presented in [22], to reduce the computation overhead of the existed simulation-based approaches, a fast signal selection based on machine learning techniques was demonstrated to improve the restoration ratio. Our approach aims to build a predictive model by applying machine learning techniques to reduce the trace buffer size and directly locate the upcoming failures.

Chapter 2

Methodologies and Experiment Set-up

2.1 Program Structure

A overview of program structure, Figure 2.1, is provided in the following with a simplified process explanations in order to understand this work better.

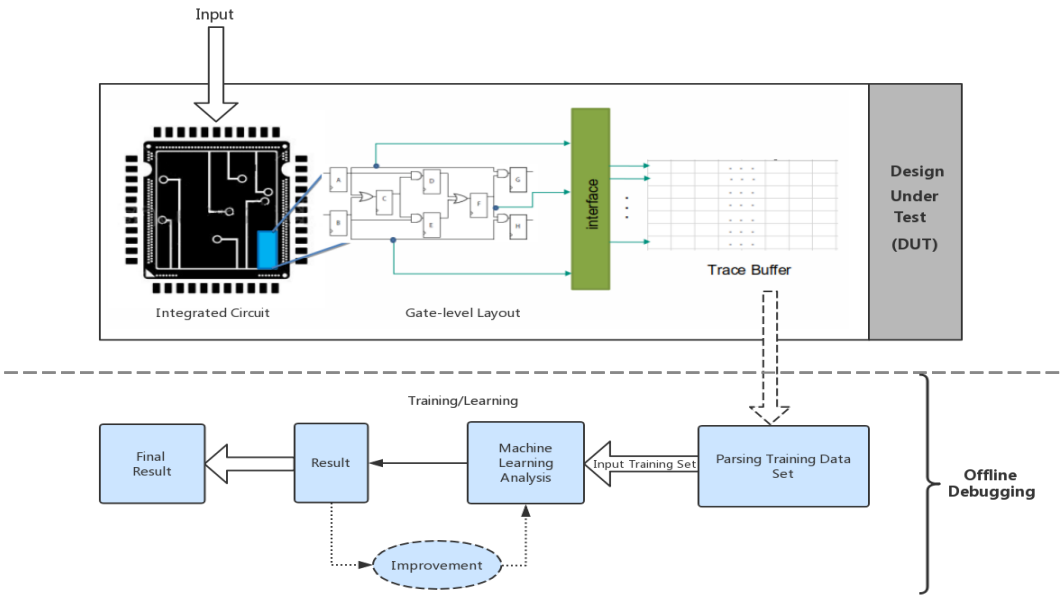


Figure 2.1: Simplified View of Program Structure

As what was already mentioned in chapter 1.2, the goal of this thesis is focusing on the post-silicon debugging stage, to locate electrical bugs due to timing errors and reduce number of traced signals due to the physical size limitation by using machine learning methods.

In the beginning, the initial simulations were implemented in both the ISCAS'89 and ITC'99 benchmark suits. For realistic PDN behavior, the microprocessor benchmark LEON3 was explored, it was firstly synthesized by using some modern synthesizing tool and then several applications, which belong to the MiBench programs, were implemented in post-layout simulation. The behaviors during the simulation of all the components on the chip were monitored and recorded.

Also, extracting transient spatial voltage droop map for a real simulation data of LEON3 was set up. Then the net-list information and simulation results for voltage droops were re-formatted in order to make them compatible with the machine learning algorithms. In the final training data set which contains only the flip-flops, maximum droop of each simulation segment and the grid location where the maximum droop occurred.

In the end, several machine learning algorithms are compared to solve, e.g., unbalanced classes problem, in order to reduce the trace buffer size and locate the electrical bugs in the offline debugging part. The detailed illustrations, issues and solutions are introduced in the following chapter 3 *Implementations and Experiments*.

2.2 Benchmark Circuits Selection

2.2.1 ISCAS'89, ITC'99 and LEON3 Processor

The experiments were implemented firstly on parts of sequential benchmark circuits of ISCAS'89 [23] benchmarks and several benchmarks, for example b15 and b17, from ITC'99 suit [24].

Although ISCAS'89 benchmarks are already extended in terms of size and complexity comparing to ISCAS'85 [25] benchmarks, but both ISCAS'89 and ITC'99 suit are still too small for testing our proposal. Since a simulation with more realistic PDN behavior is wanted, so LEON3 Processor, which is presented in the following, was chosen. There are several versions of LEON processor, details are presented in the user's manual [26] and the main differences are listed in the following.

- *LEON2 :* dose not support SMP and has a five-stage pipeline.
- *LEON3 :* includes SMP support and a seven-stage pipeline.
  (It is a synthesisable VHDL model which is highly configurable, and particularly suitable for system-on-a-chip (SoC) designs.)

The full source code of LEON3 processor is available under the GNU GPL license and proprietary license, which allows it to be used in proprietary applications. This is also one of the reasons it was chosen.

2.2.2 Grouping

According to the size and location of each component that were specified, to group all of them on the chip to different grids is the principle idea of *Grouping*. In the meanwhile, the consumed power of each component was also grouped for the later transient voltage simulation. The total number of *grids* was defined by the tester, and they were introduced in order to locate the electrical bugs to a known location for debugging. The size and location information of all the logical components were given by the library files.

The LEON3 microprocessor was synthesized by using some modern synthesizing tool first and after the placement and routing were done on the synthesized microprocessor net-list, the total number of components became 33922 in the final layout with 2366 registers only.

Although the recorded the switching power values of all the components were parsed as the input of the voltage simulation. Since only the transient voltage simulation is interested in our proposal, therefore, only the generated voltage values, include maximum voltage droops and maximum package droop, of all the Flip-Flops (FFs) were extracted for the later usage.

The total number of grids to be divided is also important since the aim of this thesis is to locate the electrical bugs. Number of rows, which were divided, was defined as $Rows$ and number of columns was defined as $Columns$ in grouping section. Then the total number of grids became obvious.
As

$$Total\_Grids = Rows * Columns \tag{2.1}$$

The original version of grouping was developed to retrieve all the divided grids in order to locate each component. Which gave the time complexity T as shown in

the equation 2.2.

$$T(Rows, Columns) \sim O(Rows * Columns) \tag{2.2}$$

In a realistic scenario, since the total number of grids was large, retrieval of all grids became time consuming. Then an improvement on the grouping program was made to locate row number first then column. And the time complexity of the improved algorithm was demonstrated in equation 2.3.

$$T(Rows, Columns) \sim O(Rows + Columns) \tag{2.3}$$

When the number of rows and columns were the same, e.g., defined as n, then the equation 2.2 became $O(n^2)$, and the improved case with equation 2.3 became $O(n)$. The improvement is obvious, but in our final simulation, the total number of used grids is 25 according to the Voltspot limitation, but in a real world problem, the total of grids needed is much more than this.

A normal microprocessor nowadays has more than billions of components, assume there is a microprocessor which contains one billion of components and they are uniformly distributed on the chip, total number of grids for grouping is 25 which means, there are forty million components in each grid.

2.2.3 Extract the spatial voltage droop map

In this stage, VoltSpot [27, 28, 29, 1] was used for transient voltage simulation, detailed specifications could be found in the papers mentioned before or through the user manual. And for each workload segment, the spatial voltage droop map for $5 * 5$ grid is extracted.

2.2.3.1 Transient Simulation

Constant power (static power) dissipated by the cell in steady state after the vector is applied and all transitions are stabilized. Usually it is less than 1% to larger than 30% of total power when cell active but the majority of power when cell inactive. And dynamic power is part of the power that dissipated when an input transition is being made, for example charging or discharging the internal and external capacitive loads.

The influence of parasitics should be considered while analyzing the switching behavior of an integrated circuit. Any realistic calculation of the transient behavior

has to include the capacitive effects associated with junctions areas and conduction wires. A summary of the physical fundamentals including theories and equations used for transient simulation is also given in [30, 31].

In the steady simulation, only the IR drop is considered. But for the extensive transient simulations on a large scale power delivery structures, analyzing power delivery fluctuation caused by dynamic IR drop and Ldi/dt drop, as well as package and on-chip resonance, are required.

In order to check the generated voltage droop information is correct and to better understand each workload, the IR drop distribution is plotted which could also make the understanding of power distribution of the microprocessor of one simulation segment after Grouping easier.

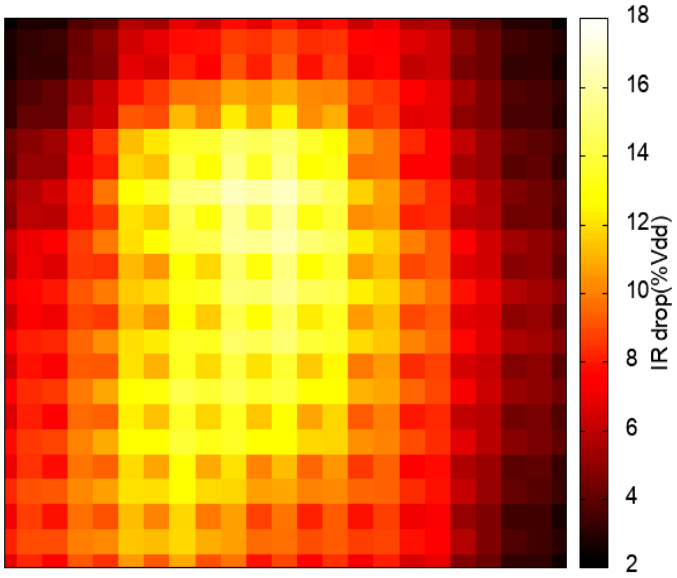


Figure 2.2: Example of power distribution after Grouping for one segment data

As showed in Figure 2.2, different colors represented different level of IR Dropson the microprocessor which were measured in percentage of V_{dd}, brighter color means larger voltage drop and darker means smaller or no voltage drop. The IR drop difference for each grid is obvious and it can be easily discovered that considering a certain simulation segment, the larger IR drops happened in adjacent grids.

2.2.3.2 Spatial Voltage Droop Extraction

For extracting the spatial voltage droop information, both the workload segments, which contained the power information and the floorplan, that contained the grid size and location information of the chip was defined in grouping, should be provided to the VoltSpot platform.

As mentioned in previous subsection 2.2.3.1, the transient operations should be implemented in current stage in order to capture the switches and analyze the states changing in the later process. Thus, several simulation segments' records, which contains the power information, are needed in one simulation time.

In order to avoid the final data set is too small, the *sliding window approach* is introduced during transient simulation.

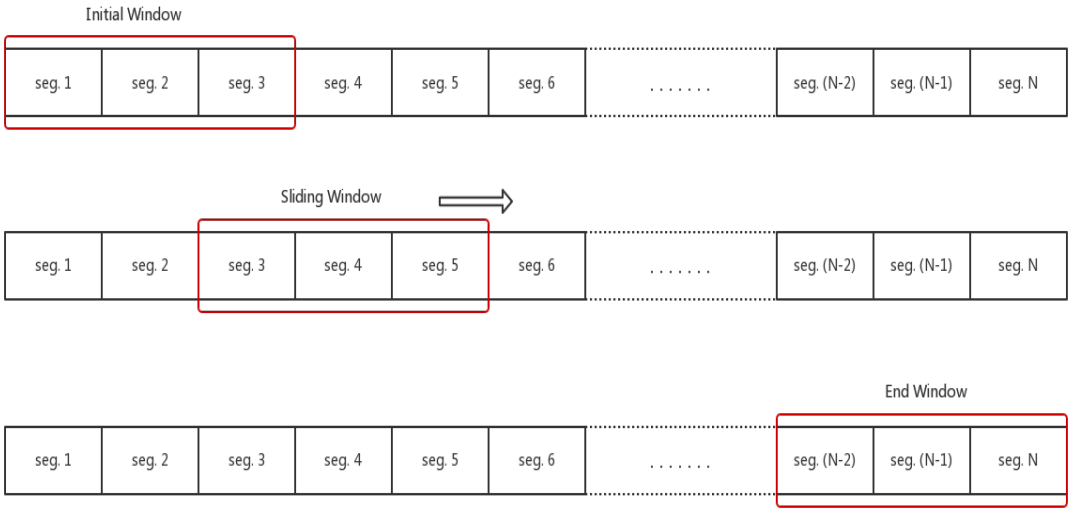


Figure 2.3: Example of sliding window approach with window size = 3

The example which is given in Figure 2.3 shows the principle idea of sliding window approach with the window size equal to three and the total number of workload segments equals to N.

Each workload segment after grouping named as *seg.x* and x are integer numbers for ordering the segments according to the simulation time.

And for example, in the initial window, first three workload segments should be simulated, and the end window is generated when the last segment is reached. Therefore the total number of resulted recordings after transient voltage simulation grows simply from $N/window\_size$ to $(N - window\_size + 1)$.

Chapter 3

Experiments and Implementations

3.1 Final Data Set Generation

For preparing the training data set, feature vectors and labels, also called X values and y values, should be parsed into a proper format after chosen, in order to make it compatible to the machine learning algorithms.

First of all, the maximum was found among the all the voltage droops in each simulation time slot or in another word, in each simulation segments, after extracting and recording the voltage droop values as explained in section 2.2.3. And here in our experiment, 500 was chosen due to the physical memory size of the computer and the total simulation time. The corresponded grid location in the chip was formatted as (a, b) where a and b represented for integer numbers, and totally $5 * 5 = 25$ grids were used, therefore, the location was labeled from (0, 0) to (4, 4).

And the workload segments after post-layout simulation contained states changing information of the traced signals that monitored by the trace-buffer, all the recorded FFs were used as feature vectors for training, and either SA or SP, that were illustrated in section 1.3.4, could be used as the feature values. And SA was chosen to be used as the feature values in this thesis.

Since the states changing information was known for each FF in each simulation segments, thus the SA could be easily calculated as:

$$SA \ = \ P\ (State\ switches) = \frac{N_{1->0} + N_{0->1}}{N_{Total\ number\ of\ traced\ states}}$$

Finally, using SAs of different flip-flops in the design as feature vectors (X variables) and the coordinates of voltage hot-spots as y variable. Then another y variable max

droop is added for later checking. Collecting SAs for several workload segments, each of them equivalents to one simulation segment of 10 clock cycles, i.e., the information in a single SAIF file, and their corresponding hot-spot. Each one of such a pair was a sample in the data set, thus the final data set is generated.

For example, there are 100 flip-flops and the grid size is 55, the data set example is showed in the table below.

FF_1SA	FF_2SA	FF_3SA	...	$FF_{100}SA$	Hotspot grid location [y1]	Max Voltage droop [y2]
0.015	0.0	0.025		0.01	(1,3)	182.838286

Table 3.1: Data Set Example

As showing in the table 3.1, FF_1SA represented for the first flip-flop's switching activity of the first workload segment (the SAIF file which recorded the first simulation step), and in the real data-set, it appeared as the real name of a flip-flop.

In the end, three workloads, *Sha_small*, *Bitcount* and *Basicmath*, were generated based on the MiBench programs and were formatted as illustrated in Table 3.1.

3.2 Training Data Set Analysis

After generating the data sets of three different workloads from MiBench, all of them were analyzed firstly in order to understand them better and then implement a suitable machine learning algorithm on them.

All the generated data sets of three workloads were formatted as a matrix of size 500 by 2373. And as mentioned previously, 25 grids were used which means that at most 25 classes could show up. But after grouping the segments in each workload by different class labels, the results showed that not all of the grid labels could be observed and there was also an unbalanced classes distribution, which means that data-sets with a disproportionate ratio of observations in each class.

An example of unbalanced class problem is demonstrated in the table 3.2 below. Given both the grid location label and the percentage of the samples that could be observed.

Class Distribution	0	1	2	3	4
0	(0, 0) 0.8%	(0, 1) 0.8%	(0, 2) 0%	(0,3) 0%	(0, 4) 0%
1	(1, 0) 75.6%	(1, 1) 10.4%	(1, 2) 0.8%	(1, 3) 0.8%	(1, 4) 0.8%
2	(2, 0) 2%	(2, 1) 1.4%	(2, 2) 0.8%	(2, 3) 0.8%	(2, 4) 0.8%
3	(3, 0) 0%	(3, 1) 1.2%	(3, 2) 0%	(3, 3) 0%	(3, 4) 1%
4	(4, 0) 1%	(4, 1) 1%	(4, 2) 0%	(4, 3) 0%	(4, 4) 0%

Table 3.2: Unbalanced classes example - Sha_small

It could be easily see from the example scenario that is illustrated in the table, the number of observed sample belonged to class (1, 0) is significantly higher than those belonging to the other classes.

This problem is predominant in scenarios where anomaly detection is critical. The predictive models that were developed using conventional machine learning algorithms could be biased and inaccurate in this case. And usually, the class distribution, proportion or balance of classes are not taken into account by the machine learning algorithms because in general they are designed to improve accuracy by reducing errors.

3.3 Model Selection

Usually, many learning algorithms could be chosen for dealing with the real-world tasks. Even for the same algorithm, different models could be generated with different parameter settings.

The ideal solution for learning algorithms selection and parameter setting is to evaluate the generalization error, then choosing the model with the smallest generalization error. Thus, a testing set is needed for testing the ability of predicting new samples of a learner. And the testing error on the testing set is regarded as the approximation of the generalization error.

Therefore in the following subsections, the approaches for dealing with the unbalanced classes were discussed firstly; then the most commonly used machine learning algorithms were tried and tested to see the performance; several machine learning algorithms were analyzed and detailed illustrations were presented; in the end, four learning algorithms were determined for the final machine learning experiments.

3.3.1 Approaches for Handling Unbalanced Classes

For dealing with the unbalanced data-set, either to improve the implemented algorithms or work on the training set through data processing before provided these data to the machine learning algorithms, the possible techniques that could be used are summarized below also in the Figure 3.1, the detailed information could be found in [32] and some key methods were also illustrated in the following.

- *Data level approaches*
  - *Random Under-Sampling* deleting the samples in the majority class randomly in order to balance all the classes.
  - *Random Over-Sampling* increasing the number of samples in the less distributed class by randomly duplicating them.

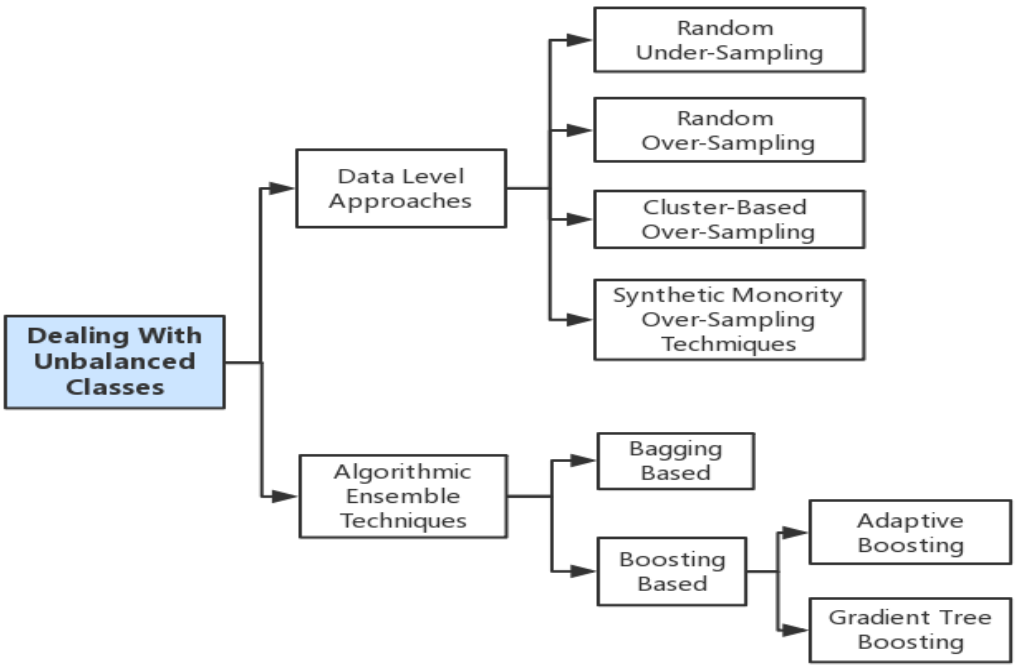


Figure 3.1: Summary of approaches for unbalanced classes

  - *Cluster-Based Over-Sampling* over-sample each cluster in order to make all clusters have an equal number of samples, in another works, all classes have the same size.

  - *Synthetic Minority Over-Sampling Technique* avoids over-fitting problem. Synthetic instances are created and added to the original data-set and new data-set is used as a sample to train the learning models.

- *Algorithmic Ensemble Techniques*

  - *Bagging Based* creating strong learners in order to reduce over-fitting problem.

  - *Boosting Based* combining weak learners in order to create a strong learner which can make the accurate predictions.

As summarized before, there are several under-sampling or over-sampling techniques for re-scaling data-set, since the data-set is generated from the real world scenario, the sample distribution is unknown and unique from workload to workload. Re-sampling the data set seems like an improper technique for current task. Since the useful information that is critical for building rule classifiers could be discarded in the under-sampling techniques and more information could be added in the over-sampling techniques, so they will not be the accurate representations of the distribution. Therefore, the algorithmic ensemble techniques are preferred and in the final implementation, the boosting based algorithms *Gradient Boosting Classifier* (GBC) and an alternative of it in case of multi-class problems, *Random Forest Classifier* (RFC), were chosen and applied.

3.3.2 Gradient Boosting Classifier

The Gradient Boosting Classifier (GBC) or Gradient Tree Boosting is a generalization of boosting to arbitrary differentiable loss functions [33]. It could be regarded as the combination of gradient descent and boosting. GBC was studied in order to improve the the current performance, a simple example of AdaBoost is presented in the Figure 3.2, 3.3 firstly and then a simplified view of the structure is showed in the Figure 3.4.

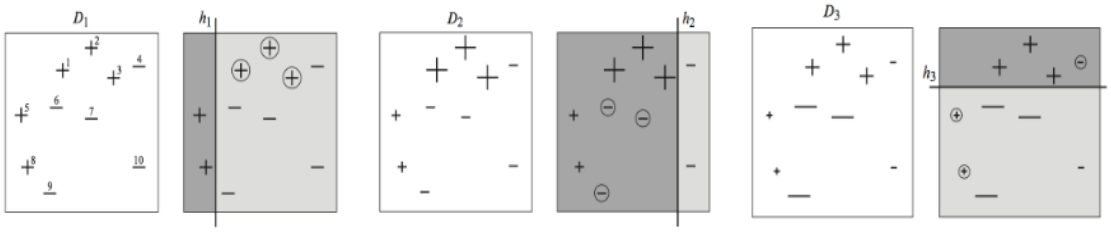


Figure 3.2: Structure view AdaBoost
Source [3] Figure 1.1

From the figures 3.2 and 3.3 which demonstrated the behaviors of AdaBoost, there are three pairs of boxes, the left white box represents the distribution D_t with the size of each example scaled in portion to its weight under the distribution. And each box on the right shows the weak hypothesis h_i, where darker shading indicates the region of the prediction to be positive and the remaining region stands for the prediction to be negative. In addition, the misclassified samples are circled. Finally, the final classifier or combined classifier is computed as the sign of the weighted sum of the three weak hypotheses [3].

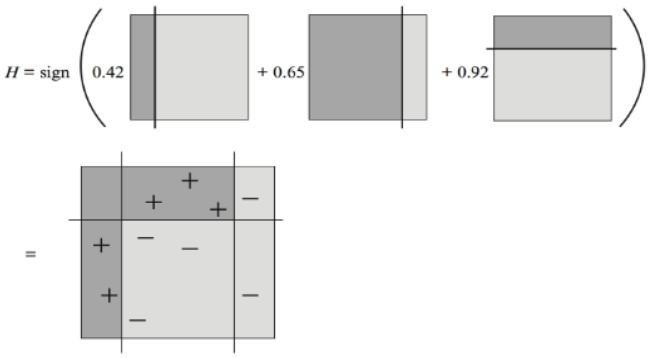


Figure 3.3: Structure view of AdaBoost
Source [3] Figure 1.2

What could be easily see from Figure 3.4 is that Gradient Boosting fits and additive model in a forward stage-wise manner, a weak learner is introduced in each stage. Thus many learning models are trained sequentially in the Gradient Boosting instead of one, and each model minimizes the loss function by using, for example the standard approach: *Gradient Descent Method* (GDM), which is $y = ax + b + e$. Gradient Boosting works on weak learners or classifiers and *Decision Trees* are used

as base learners. It tries to boost them into a strong learner.

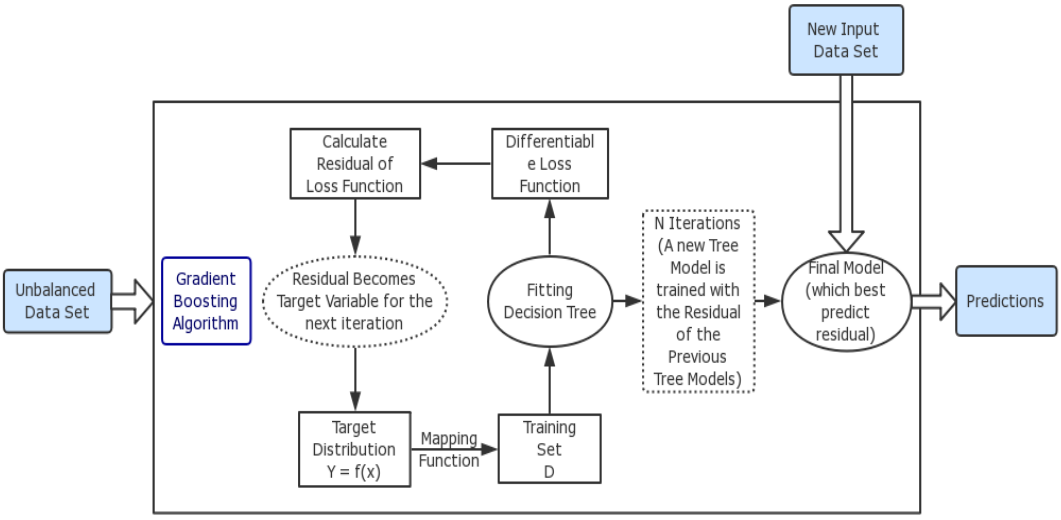


Figure 3.4: Structure view of Gradient Boosting classifier

First of all, the learner is built by Gradient Boosting on the original training data-set to do the prediction, then the losses are calculated which usually as the difference between the real value and the predicted value that calculated by the first learner. Then the losses are used to build a new learner in the second step and in general it is improved comparing the previous one, the detailed demonstrations could be found in [3].

At each step, the residual of the loss function is calculated by implementing the GDM and the new residual of the loss function becomes a target variable for the next iteration. Thus the learner or classifier is improved in each iteration compared to the previous one with different residuals.

In order to fit the Gradient Boosting Tree well, all the three parameters, e,g,. Shrinkage parameter, tree depth and the total amount of trees, should be trained properly. If the parameters are not tuned correctly, the over-fitting problem may be generated.

3.3.3 Random Forest Classifier

In the LEON3 microprocessor scenario, there are more than two thousand feature values that each with a small quantity of information in each workload. The accuracy of a single tree classifier is slightly better than a random choice of class, but the combined tree classifiers can improve the accuracy. And as mentioned in paper [34], "because of the Law of Large Numbers they do not overfit, injecting the right kind of randomness makes them accurate classifier".

The sklearn.ensemble module [35] includes RandomForest algorithm which based on randomized decision trees.

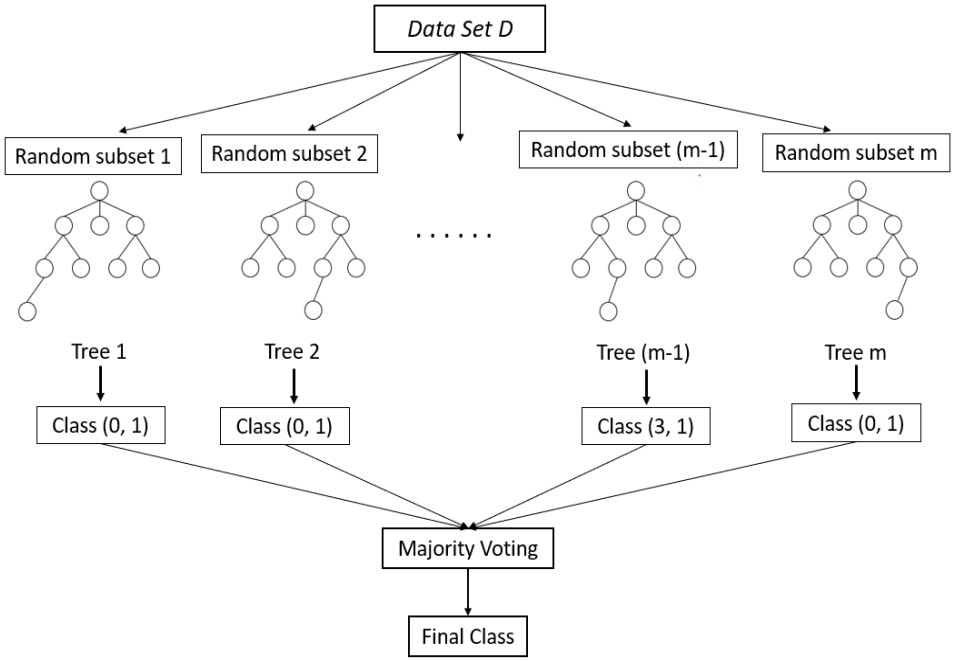


Figure 3.5: Simplified view of Random Forest with classes voted by each classifier

Random forest is an ensemble method and a meta estimator that each time it takes a subset of samples from data-set to build a decision tree. After generating a large number of trees, for example *Tree 1* to *Tree m* as showed in Figure 3.5, the corresponded class is generated according to each tree, and the final class is got after the most popular class is voted in order to get a more accurate and stable prediction and control overfitting, these procedures are called *random forests*.

When constructing the trees, the split that is chosen when splitting a node is not the best split among all features but among a random subset of features, thus even

with the same training data, the best-found split may vary, this could be controlled by fixing the random state. Because of the randomness, the bias of the forest usually slightly increases, but since the averaging is used, the decreased variance usually could compensate from the increased bias, therefore an overall better model is generated.

In the implementation of sklearn, the prediction of a group of samples is given as the averaged prediction of the individual classifiers [36] instead of letting each classifier vote for a single class as illustrated in Figure 3.5. And the base estimator is *Decision Tree Classifier*. In the matter of fact, the final prediction according to the majority voting of independent judges is better than the best judge.

The size of the trees should be controlled by setting, for example the maximum depth of the trees, by considering the complexity of the tree and the memory consumption, but since our data-set is small, this could be simply ignored. And the results of forests are competitive with boosting and adaptive bagging methods.

3.3.4 Cross Validation

Three workloads were generated previously and for each of them the corresponded data set contains 500 samples and for each sample that contains 2373 features, for simplicity m and n were used to replace the numbers, then the data set could be represented as following.

$$\boldsymbol{D} = \{(x_{1,\ 1}, x_{1,\ 2}..., x_{1,\ n},\ y_1), (x_{2,\ 1}, x_{2,\ 2}..., x_{2,\ n},\ y_2), ..., (x_{m,\ 1}, x_{m,\ 2}..., x_{m,\ n},\ y_m)\}$$

where $x_{m,\ n}$ stands for the *n-th* feature of the *m-th* sample and y_m stands for the label of *m-th* sample.

These data sets were separated properly into training set $\boldsymbol{S}$ and testing set $\boldsymbol{T}$, thus each workload could be used both for training and testing. The most commonly used techniques are *hold-out*, *cross-validation*, *bootstrapping* and *parameter tuning*. Estimating the expected level of fit of a learning model to a data set and the trained model is independent of the training data set was used is the goal of cross-validation.

The cross-validation techniques could be divided into two kinds, exhaustive and non-exhaustive.

- ***Exhaustive cross-validation methods*** all possibilities are tried in order to divide the original sample set into a training set and a validation set for learning and testing.

- ***Non-exhaustive cross-validation methods*** not all the possible splittings are computed of the original sample set. They are the approximations of *leave-p-out cross-validation (LpO CV)* which divided the original data-set into different partitions with a equal size. P observations as the validation set and others are used for training.

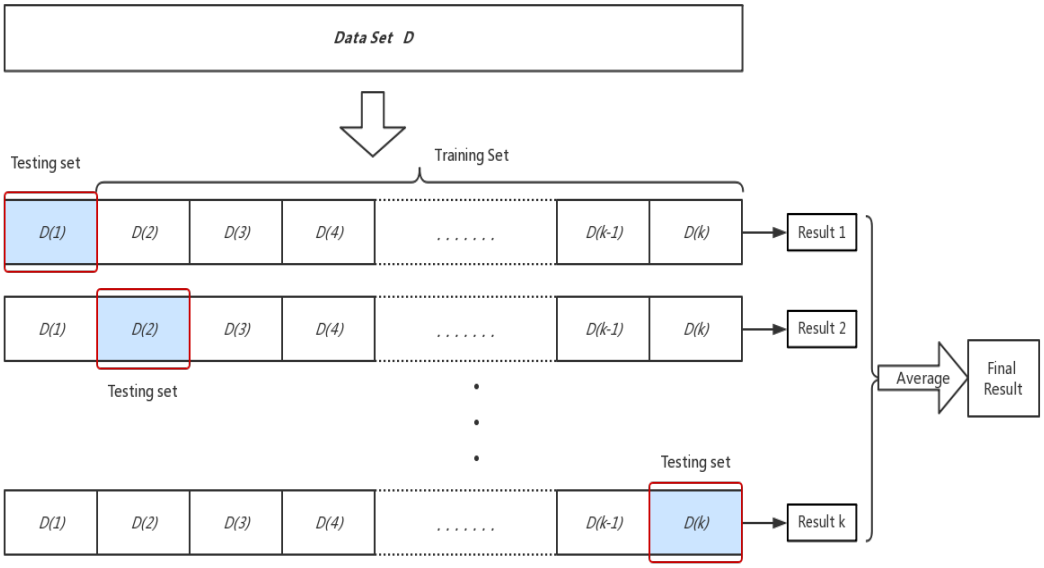


Figure 3.6: Structure view of k-fold cross-validation

In k-fold cross-validation method, the original sample set is divided randomly into k subsets *D(1)* to *D(k)* as showed in Fig. 3.6, and each of them contains an equal number of samples. One of the subsets is used as the validation set for testing the model and the *(k - 1)* remained subsets are used for training the model, then the cross-validation process could be repeated k times (the folds) [37].

The number of wrong predicted samples when testing $D(i)$ is n_i, and n for the whole data set, then the error rate of the learning model could be calculated as equation 3.1.

$$E = \sum_{i=1}^{k} n_i \ / \ n \tag{3.1}$$

Obviously, each of the k subdivided sample sets is used exactly once as the validation data. In the end, averaging the k results from the folds to produce the final estimation.

When data set D contains m samples and $k = m$, then this special case of cross-validation is called *Leave-One-out(LOO)*. Since there is only one sample in each subset, thus obviously LOO is not influenced by the ways of the sample partitioning. And the training set has only one sample less than the original data set, therefore in most cases, the model evaluated by LOO is similar to the expected learning model that trained by data set D.

The computational consumption of training m models increases with the increased size of the data set. In order to improve the accuracy of the learning model, the k-fold cross-validation could be repeated t times and every time the original data set is divided randomly into equal subsets.

The error rate for one model could be calculated as showed in equation 3.1, in the t times testing the error rate of generated t models could be represented as E_1, E_2, ..., E_t. Therefore the error rate of the final learning model is defined as e and could be calculated as in Equation 3.2.

$$e = \sum_{i=1}^{t} E_i \ / \ t \tag{3.2}$$

After considering both the model accuracy and the computational consumption, 10-fold cross-validation is chosen and the final model is generated after repeating each experiment 10 times.

3.3.5 Electrical Bug Localization Based on the Most Commonly Used Machine Learning Algorithms

Besides the algorithms that chosen in the section 3.3.1, the most commonly used algorithms were tested firstly in order to find more suitable algorithms based on experiments in order to locate the electrical bugs and compare their performance in the final implementation.

The first machine learning experiment was initialized with all X values to predict y values by using *Logistic Regression* (LR), *Linear Discriminant Analysis* (LDA), *K-Neighbors Classifier* (KNN), *Classification and Regression Trees* (CART), *Naive Bayes classifiers* (NB) and *Support Vector Machines* (SVMs). K-folder validation was used with k equal to 5 and the testing set is 20% of samples from the training set. And the results are illustrated in the table 3.3 and 3.4, both represented with the mean accuracy and the standardization of the accuracy inside the brackets.

Workloads	LR	LDA	KNN
Sha_small	0.518777 (0.085266)	0.371764 (0.107236)	0.545377 (0.068075)
Bitcount	0.549147 (0.081467)	0.464011 (0.072403)	0.586842 (0.074013)
Basicmath	0.788691 (0.057645)	0.633997 (0.068901)	0.831792 (0.022248)

Table 3.3: Testing resukts on the most commonly used algorithms - part1

Workloads	CART	NB	SVM
Sha_small	0.409104 (0.071136)	0.214011 (0.073290)	0.588051 (0.087534)
Bitcount	0.399787 (0.076108)	0.160171 (0.075059)	0.632077 (0.074303)
Basicmath	0.775960 (0.036514)	0.186629 (0.051801)	0.789331 (0.034154)

Table 3.4: Testing resukts on the most commonly used algorithms - part2

From the comparison above, what could be easily seen is that LDA, CART and NB should be discarded since they have really low accuracy, even in the debugging stage, the accuracy should not lower than 80%.

The KNN's results seemed good. If the training sets were unbalanced, and if they were uniformly distributed, the probability that any random query point will be classified to the class with more examples becomes higher. So, the closest neighbor of the query point may still belong to the class with fewer examples, but if rest $(k-1)$ points belong to the other class, because of its higher density in the space, the point will get mis-classified. The weights of classes could be considered when applying KNN. But since a generalized learning model is wanted that could work for different workloads and the weights of each class in each workload are different, and it is not feasible to change the classes' weights every time in the run-time of a chip. Therefore, the KNN is discarded either.

3.3.6 Logistic Regression

Since each of our data-sets contains 25 classes, therefore the Logistic Regression for multi-class is implemented which is named *Multinomial Logistic Regression* or *Multiclass Logistic Regression (MLR)*. As mentioned in [38], "in statistics, multinomial logistic regression is a classification method that generalizes logistic regression to multiclass problems, i.e. with more than two possible discrete outcomes". It estimates a separate binary LR model for each dummy variable and is used to predict the probabilities of the different potential outcomes of a categorically distributed

dependent variable then give a set of independent variables, in order to understand the relationship between nominal variables without ordering.

Assume there are m features in the data-set, the training algorithm could be implemented either using one-vs-rest scheme or using cross- entropy loss, then (m-1) independent binary choices could be got, therefore, the result of MLR is (m-1) binary logistic regression models. Each model carries the effect of predictors on the probability of success in that class compared to the reference class. And each model has its own intercept and regression coefficients, in another word, the predictors can affect each class differently.

3.3.7 Support Vector Machine (SVM)

Support vector machines (SVMs) are a set of supervised learning methods that could be used for classification, regression and outliers detection, in the current task, the classification method is implemented that is called *SVC*.

In order to deal with the multi-class problem, SVC used a one-vs-one approach. And the number of classes in each workload is 25, then

$$N_{Total\_class} * \frac{N_{Total\_class} - 1}{2} = 25 * \frac{25 - 1}{2} = 300$$

classifiers are built totally and each of them trains data from two classes.

A hyper-plane or a set of hyper-planes in a high or infinite dimensional space are created by SVC, good separation means the large distance between nearest training data points of any class which is called functional margin and usually the margin should be enlarged in order to lower the generalization error of a classifier.

Since the classes are not uniformly distributed, and this should be compensated by giving more importance to some certain classes, therefore in the implementation of sklearn, the class weight and sample weight can be used. The mathematical explanations are illustrated in paper [39, 40].

3.3.8 Trace Signal Selection

As what is already mentioned in section 1.3.3, the implementation of the trace buffer for run-time debugging reduces the complexity compared to the traditional debug

techniques, which usually requires to stop and restart the electronic system.

Since the width and depth of trace buffer is limited by the number of signals that can be traced per cycle and the maximum number of values recorded per signal. And the physical size of embedded trace buffer is usually limited between two thousand and eight thousand cycles, which means only few hundred signals for few thousand cycles could be traced continuously stored and in a design with millions of signals.

There are more than two thousand FFs in each generated workloads but only few hundred signals or less than one hundred signals in the design could be traced at the same time. Therefore, the feature selection or elimination method should be implemented necessarily.

The principal concept of *Recursive Feature Elimination (RFE)* is feature ranking with recursive feature elimination. An external estimator is given to assign the weights to features and selecting features by considering smaller sets of features each time recursively is the object of recursive feature elimination [41].

First of all, training the estimator on the original set of features and the importance of each attribute, through the coefficient of the attribute or the feature importance, is obtained. Then, pruning the less important features from the current set of features. This procedure is repeated recursively on the new feature set after pruning until the desired number of features are selected ultimately. In the sklearn [35], there is a method called *RFECV* which performs RFE in a cross-validation loop to select the optimal number of features.

3.3.9 Short Summary on Model Selection

Summarizing this chapter in short.

First of all, the final data-set generation was illustrated including feature values calculation and label choosing. All the SA of FFs are used as the feature values, maximum voltage droop location are the labels and after learning, the upcoming failure location of the new data-set should be predicted.

Then the data set was analyzed and the classes distribution was presented in order to better understand the current machine learning task.

After that, the last and the most important section **Model Selection** was presented. Since our machine learning task is a unbalanced multi-class classification problem. The approaches for handling the unbalanced class problems were discussed firstly; then, several machine learning techniques which are suitable for current task, like Gradient Boosting Classifier, Random forest Classifier and Support Vector Machines for multi-class problem, were analyzed and demonstrated with details; a simple experiment on the most commonly used machine learning algorithms was also implemented in order to see the results and choose more possible algorithms for the final experiments. The importance and necessity of using cross-validation and trace signal selection were also discussed in the meanwhile.

In the end, only the Gradient Boosting Classifier and, Random Forest Classifier, LR and SVM were chosen and implemented in the final experiments in chapter 4 after the discussions .

Chapter 4

Results and Performance Evaluation

This chapter focuses on the **evaluation of final results and performance**. First of all, the chosen machine learning algorithms were trained and tested by using the same workload, then the improved experiments were implemented by using different workload at training and testing stage in order to simulate a more realistic scenario. Detailed results and performances were illustrated in the following sections.

4.1 Training and Testing Using the Same Workload

The same workload was implemented for both training and testing in current section.

In the training stage, 10-fold cross-validation was applied, which means breaking the data-set randomly and equally into ten parts, 9 subsets were used for training the models and the rest one was used for testing in each fold, and the result is the average of the 10 folds' results. Even the whole workload data-set was used, but because of the randomness when separating the data-set, the result generated by running the experiment each time was slightly different than other times, thus each experiment was repeated for ten times. And the final result for the entire experiment was the average of the ten results. In the final testing procedure, testing set size was set to 0.2 which means 20% of the total samples in each workload were chosen for testing the learned model.

Since in the original data-set, all the feature values are signal activities that are calculated as the probability of states switched, in other word the adjacency states

are different, which means all the attribute values are already between zero and one. Therefore, the implementation of data-set normalization or standardization is not necessary.

RFE was implemented firstly in order to reduce the number of flip-flops are selected as trace signals to the trace buffer, then the new training data-set after selection was regarded as the new input data-set to the machine learning algorithms, which were selected previously, for building the learning models, these were repeated recursively. All three workloads were tested and the current machine learning results were demonstrated in the following.

Workload	Testing	GradientBoosting	RandomForest	LR	SVM
Sha_small	Average	96.25% [67.52%]	97% [73.09%]	94% [6.37%]	97% [19.87%]
Bitcount		96.15% [80.94%]	97% [66.64%]	96% [2.15%]	97% [1.31%]
Basicmath		92.1% [11.26%]	90.7% [13.54%]	81% [0.04%]	77% [1.31%]
Sha_small	Best	96% [2.15%]	96% [28.30%]	94% [6.37%]	97% [19.87%]
Bitcount		96% [0.46%]	97% [1.31%]	96% [2.15%]	97% [1.31%]
Basicmath		88-94% [1.31%]	90% [0.46%]	81% [0.04%]	77% [1.31%]

Figure 4.1: Training and testing with same workload, validation size = 0.2

From Fig. 4.1 that was presented above, four different machine learning algorithms were tested on each workload after trace signal selection. In the columns which were labeled as *Average*, the final accuracy scores were calculated as the average of the results that were generated after running the experiment for totally 10 times, the average number of selected features were also presented inside the brackets; the columns with label *Best* was just the best result among all the previously generated results considering both the accuracy score and the number of selected trace signals.

The performances of four algorithms for each workload vary a little bit. Comparing the performance of a single machine learning method on three different workloads, the accuracy scores generated on workload Basicmath are much lower than other workloads. Gradient Boosting Classifier, Random Forest Classifier and SVM have almost the same accuracy score both in the average and the best case when testing on the workload Sha_small and Bitcount and all of them are higher than 96%; but if comparing only the number of selected FFs as the trace signals, they performed much better on the workload Bitcount.

Logistic Regression has almost the same performance, considering both the accuracy score and the number of selected trace signals, not matter in average or best result scenario; SVM in the Basicmath scenario has the lowest accuracy score 77% and the second lowest accuracy score 81%, in both the average and best case.

Since the debugging stage is concentrated, thus all the machine learning algorithms with the accuracy score larger than 80% should be acceptable. And it is easy to find that almost all the machine learning techniques always have the accuracy scores that are greater than 90%.

Workload (Trace Signal Reduction)		GradientBoosting	RandomForest	LR	SVM
Sha_small	Average	67.52%	73.09%	6.37%	19.87%
	Best	2.15%	28.30%	6.37%	19.87%
Bitcount	Average	80.94%	66.64%	2.15%	1.31%
	Best	0.46%	1.31%	2.15%	1.31%
Basicmath	Average	11.26%	13.54%	0.04%	1.31%
	Best	1.31%	0.46%	0.04%	1.31%

Table 4.1: Trace signal selection result testing on the same workload in percentage of reduced signals

Taking a closer look on the percentage of selected trace signals on the Table 4.1. All the data are represented in the percentage of total number of flip-flops that were selected as trace signals to trace buffer. The results vary according to different workloads.

Among three workloads, GBC and RFC have the selection percentage of trace signals that are much higher in the best result than in the average result. The performances of average and best scenario result for each workload are the same when testing LR and SVM, but the SVM has a slightly better performance on the Bitcount workload, LR performed better on the other workloads.

Comparing all the best results of the trace signal selection percentage for each technique, there is no much difference among all the learning techniques on the workload Basicmath; GBC always has the best performance in Sha_small and Bitcount among all the algorithms. Comparing only the average result of all the scenarios, the performances of LR and SVM are much better than GBC and RFC.

4.1.1 Short Summary

The performance of each algorithm varies according to different workloads. All the chosen algorithms could provide pretty good accuracy score on workload Sha_small and Bitcount, with lowest accuracy score 94%. And LR and SVM could give good performance no matter in the best result or in the average case in terms of the trace signal selection percentage.

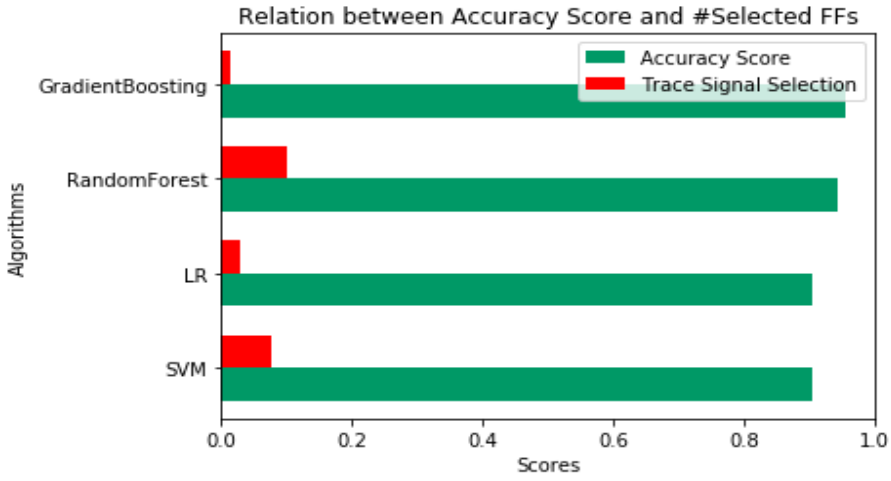


Figure 4.2: ML Results - Training and Testing by using the same workload

The overall machine learning results are presented in the bar-chart that is showed above. And what could be easily found is that Gradient Boosting Classifier has the best performance with the highest accuracy score and the lowest trace signal selection rate.

4.2 Training and Testing Using Different Workloads

Since in a real-world scenario, what kind of workload will show up in the future is not predictable, and since there are more than two thousand features for each sample in our original data-set, therefore, the combinations of all the attribute values which were used for training the model with a really high probability that they will never show up again in the future. Thus, training and testing the machine learning algorithms by using different workloads were proposed.

And for example, there are three previously generated workloads, Sha_small,

Bitcount, and Basicmath. Each time one of them was utilized for training the learning model firstly and then one of the rest two workloads was chosen and implemented as the testing set. In another word, each learning model was trained by one workload and was tested by other two different workloads. And in the end, there were 6 different scenarios. With the same experimental settings as demonstrated in section 4.1, the experiments were repeated for 10 times and both the average accuracy and the best accuracy score were calculated for each algorithm and recorded separately in columns labeled as *Average* and *Best*, the average number of FFs which were selected as trace signals after feature selection and the best result among all the experimental results were also presented inside the brackets after the accuracy score in percentage of total number of FFs.

All the results of experiments that used the same training workload were organized into one table like showed in the Fig. 4.3, which means all the machine learning techniques were trained by wokload Sha_small. Then the corresponded *trace signal selection* information were presented in a table like Table 4.2 in percentage of total number of flip-flops.

Training Workload	Testing Workload		GradientBoosting	RandomForest	LR	SVM
Sha_small	Bitcount	Average	90.2% [64.78%]	90.6% [59.81%]	90.3% [26.02%]	91.2% [80.60%]
	Basicmath		88.4% [58.46%]	89.9% [57.70%]	85.5% [26.02%]	86.1% [100%]
	Bitcount	Best	90.4% [1.31%]	90.4% [0.46%]	84% [0.04%]	90.8% [2.99%]
	Basicmath		88% [1.31%]	88.4% [0.46%]	88% [0.04%]	90.0% [7.21%]

Figure 4.3: ML results with different testing set - Sha_small as training set

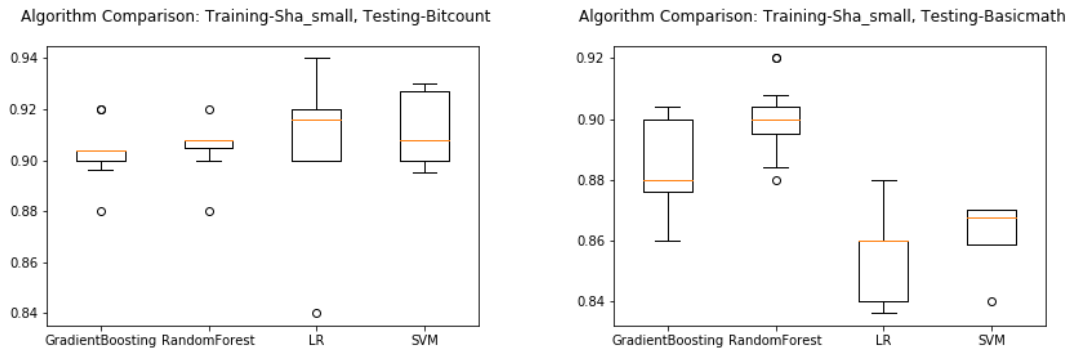


Figure 4.4: ML Algorithms Trained by Sha_small

From Figure 4.3, what could be easily found is that the performances of GBC

and RFC in case of average and best results are really close considering both the accuracy score and the number of selected FFs; for LR and SVM, the learning model that tested by Bitcount workload has the better results in the average rather than the results in the columns that were labeled as best; in the contrary, the model tested by Basicmath has always the better performances in the best results. The accuracy scores were presented in the box-plot and the average scores were marked by orange lines inside each box. As showed in the Figure 4.4 above, algorithms which trained by Sha_small and tested by Bitcount was compared on the left and on the right hand side the algorithms were tested by Basicmath.

In the LR case which was tested on workload Bitcount, the best result is presented with accuracy score 84%, which is 6.3% lower than the average result, it is abnormal and it happened because the number of reduced trace signal is also considered since one of our aim is reducing the trace buffer size, better result was generated with a higher percentage of selected trace signals which was not recorded as the best result. Similar phenomenon happened also in the SVM and GBC on workload Bitcount with a smaller difference. Besides this lowest score, all others are larger than 86%.

Considering the same learning algorithm for different testing workloads in terms of average accuracy score, SVM varies from 86.1% to 91.2% which has the biggest accuracy score difference which is 5.1%, LR has 4.8% variety, RFC has the smallest vary difference which is only 0.7%.

Trace Signal Reduction	GradientBoosting	RandomForest	LR	SVM	
Total FFs	2371	2371	2371	2371	
FFs after reduction	31	11	2	71	171
Selection Percentage	1.31%	0.46%	0.04%	2.99%	7.21%

Table 4.2: Trace signal selection consider best scenario in Sha_small

If considering only the number of trace signals after feature selection that appeared in the best results, GBC, RFC and LR selected less than 2% of the total FFs as the trace signals, tested both on the workload Bitcount and Basicmath. And for SVM, depends on the testing data-set, in other word, different workload, consider both the best and the worst results appeared previously, the number of selected trace signals varies from about 3% in the best scenario to around 7% in the worst case. Trace signal selection is 7% means that only 171 FFs were selected after implementing the feature elimination method in current LEON3 microprocessor scenario

which seems like good; but in a real processor that has more than 8 billion transistors, monitoring 7% of them means that there are more than 500 million signals need to be monitored, and it is not feasible. Therefore, GBC, RFC and LR worked much better than SVM in terms of trace signal selection. The detailed data were summarized in Table 4.2.

Training Workload	Testing Workload		GradientBoosting	RandomForest	LR	SVM
Bitcount	Sha_small	Average	84.4% [45.09%]	90.0% [33.95%]	90.3% [0.63%]	87.7% [60.19%]
	Basicmath		81.6% [53.94%]	88.7% [37.20%]	87.7% [0.55%]	87.7% [73.98%]
	Sha_small	Best	92% [0.46%]	88% [0.46%]	90.8% [0.04%]	85% [7.21%]
	Basicmath		91% [0.46%]	92.7% [0.46%]	86% [0.04%]	80% [8.90%]

Figure 4.5: ML results with different testing set - Bitcount as training set

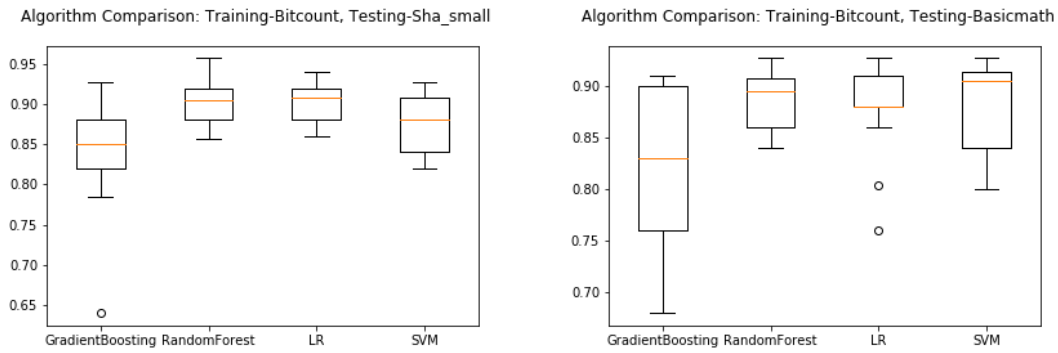


Figure 4.6: ML Algorithms Trained by Bitcount

Moving to the scenario that all the machine learning methods were trained by using workload Bitcount and then tested by other workloads, as showed in Fig. 4.5. The average accuracy score for GBC varies from 81.6% to 84.4%, RFC and LR very from around 88% to around 90%, and SVM has always 87.7% for both Sha_small and Basicmath. If considering only the accuracy score, RFC and LR works better in this scenario. The learning model that generated by SVM and tested by workload Basicmath, has the lowest accuracy 80% in the *Best* column. The best accuracy score is lower than the average as what happened and explained in the previous scenarios. It is the lowest accuracy score overall and since we are in the debugging stage which means the current result is still acceptable.

Trace Signal Reduction	GradientBoosting	RandomForest	LR	SVM	
Total FFs	2371	2371	2371	2371	
FFs after reduction	11	11	2	171	211
Selection Percentage	0.46%	0.46%	0.04%	7.21%	8.90%

Table 4.3: Trace signal selection consider best scenario in Bitcount

As demonstrated in the Table 4.3, less than 1% of the total FFs were selected as trace signals after applying RFE considering only the best cases appeared in the experiment results of GBC, RFC and LR. For SVM, the trace signals selection changes from around 7% to 9%. And what could be noticed is that in some of the scenarios, the best result among ten generated results has a better accuracy score than the average case with small amount of features after selection. Less number of features can represent the data set better, which seems like violated the general sense and it was abnormal. It is possibly because of the the current data-set does not cover all the classes, future experiments should be explored.

Training Workload	Testing Workload		GradientBoosting	RandomForest	LR	SVM
Basicmath	Bitcount	Average	91.2% [34.80%]	97.1% [26.23%]	94.6% [0.25%]	77.7% [9.83%]
	Sha_small		86.5% [37.66%]	94.6% [27.67%]	92.3% [0.25%]	84.7% [10.46%]
	Bitcount	Best	87.5% [0.04%]	96% [3.84%]	87.5% [0.04%]	77.5% [1.31%]
	Sha_small		94% [0.46%]	94% [3.84%]	88% [0.04%]	83.2% [1.31%]

Figure 4.7: ML results with different testing set - Basicmath as training set

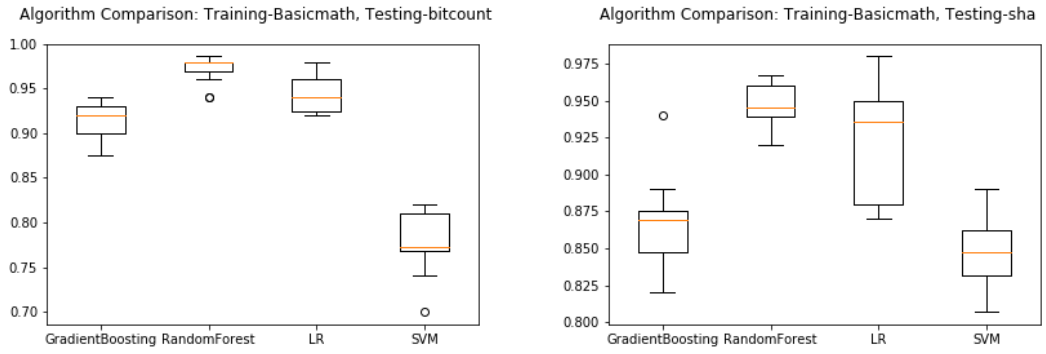


Figure 4.8: ML Algorithms Trained by Basicmath

From the Fig. 4.7, what could be easily found is that usually the average accuracy score is a little bit higher than the best case but with a much larger number

of selected FFs. Since the aim of this thesis is not only to locate the electrical bugs but also to reduce the number of trace signals, so the trace signal selection is also considered and it makes the current results reasonable.

Comparing the performance of four algorithms for the same workload, it is easy to find that Random Forest Classifier always has the best result and all of them were greater than 94%; SVM always has the worst results. But all of them do not vary much. Considering only the average result among all, the best accuracy score 97.1% was generated by using Bitcount as the testing set in Random Forest Classifier case; the worst result is around 77.5% which was generated by SVM also in Bitcount scenario, and it is too low even in the debugging stage.

Trace Signal Reduction	GradientBoosting		RandomForest	LR	SVM
Total FFs	2371		2371	2371	2371
FFs after reduction	2	11	91	2	31
Selection Percentage	0.04%	0.46%	3.84%	0.04%	1.21%

Table 4.4: Trace signal selection consider best scenario in Basicmath

In the Table 4.4, RFC has the highest selection percentage which is a little bit lower than 4%. In the SVM scenario, around 1% of the total FFs were selected, and less than 0.5% of the total number of trace signals were chosen in the GBC and LR scenario which independent from the testing set.

4.2.1 Short Summary

As showed in Fig. 4.9, comparing only the average accuracy score for each learning algorithm among all different scenarios, what could be easily understood is that GBC, RFC and LR could be used to predict and locate the electrical bugs with a pretty high accuracy score. And among these three techniques, RFC almost always has the best accuracy score. The accuracy scores that generated by SVM vary from 77.7% to 91.2% depends on the different testing workloads, even it is in the debugging stage, the score lower than 80% should not be acceptable.

Training Workload	Testing Workload	GradientBoosting		RandomForest	
		Average	Best	Average	Best
Sha_small	Bitcount	90.2% [64.78%]	90.4% [1.31%]	90.6% [59.81%]	90.4% [0.46%]
	Basicmath	88.4% [58.46%]	88% [1.31%]	89.9% [57.70%]	88.4% [0.46%]
Bitcount	Sha_small	84.4% [45.09%]	92% [0.46%]	90.0% [33.95%]	88% [0.46%]
	Basicmath	81.6% [53.94%]	91% [0.46%]	88.7% [37.20%]	92.7% [0.46%]
Basicmath	Bitcount	91.2% [34.80%]	87.5% [0.04%]	97.1% [26.23%]	96% [3.84%]
	Sha_small	86.5% [37.66%]	94% [0.46%]	94.6% [26.23%]	94% [3.84%]
Training Workload	Testing Workload	LR		SVM	
		Average	Best	Average	Best
Sha_small	Bitcount	90.3% [26.02%]	84% [0.04%]	91.2% [80.60%]	90.8% [2.99%]
	Basicmath	85.5% [26.02%]	88% [0.04%]	86.1% [100%]	90.0% [7.21%]
Bitcount	Sha_small	90.3% [0.63%]	90.8% [0.04%]	87.7% [60.19%]	85% [7.21%]
	Basicmath	87.7% [0.55%]	86% [0.04%]	87.7% [73.98%]	80% [8.90%]
Basicmath	Bitcount	94.6% [0.25%]	87.5% [0.04%]	77.7% [9.83%]	77.5% [1.31%]
	Sha_small	92.3% [0.25%]	88% [0.04%]	84.7% [10.46%]	83.2% [1.31%]

Figure 4.9: ML Results - Training and Testing by using different workloads

For GBC, RFC and LR, the trace signal reduction percentage is always higher than 96% and in most of the scenarios higher than 99%. But for SVM, depends on the different testing data-set, in more than half of the cases the reduction percentage is lower than 93%, GBC, RFC and LR worked much better in terms of trace signal reduction than SVM.

And the relation between accuracy score and the percentage of total FFs which were selected as trace signals was represented in bar chart and was showed in the Figure 4.10. It could be easily found that when the percentage of traced FFs decreasing, the accuracy score does not decrease rapidly. Then the final experimental results that presented in current section of the more realistic scenario were summarized in the Figure 4.11.

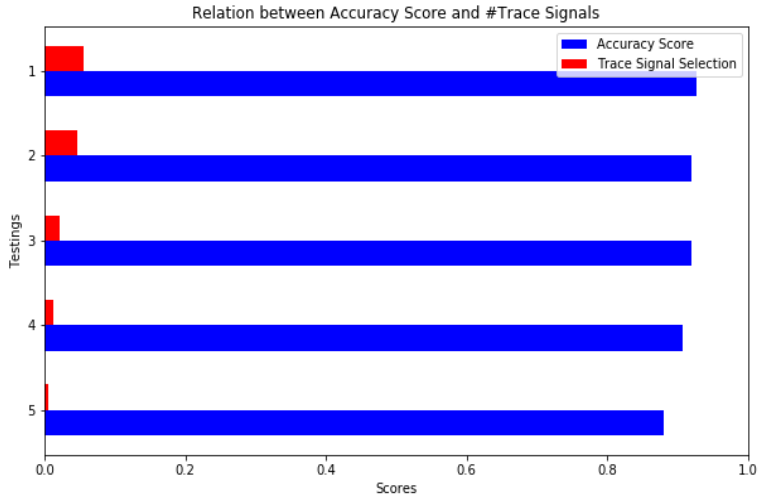


Figure 4.10: Relation between Accuracy Score and Number of Selected Trace Signals

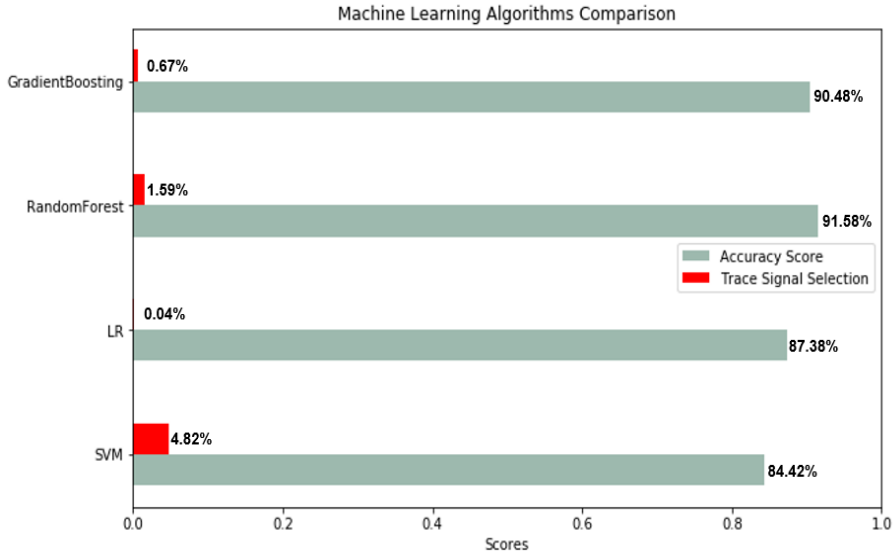


Figure 4.11: ML Results - Training and Testing by using different workloads

Then what could be easily see is that RFC has the highest accuracy 91.58% with 1.59% of total number of flip-flops are selected as trace signals to the trace buffer, it is the best performance compared to GBC, LR and SVM. And 1.59% of total number of flip-flops means that only 38 FFs were selected to be trace in the LEON3 microprocessor scenario.

What we could say is that RFC is the best algorithms with the highest accuracy score, the low percentage of signals need to trace and small variety.

Chapter 5

Conclusion

This work aims to collect workload data based on simulations and predict run-time parameters based on the machine-learning model built on these data, targets at exploring the electrical bug detection capabilities of a post-silicon debugging framework by marking informed decisions on trace signal selection by analyzing the trace buffer data to extract voltage droop profile of the PDN for bug localization and detection.

From the beginning of this work, the processor benchmark LEON3 was explored in order to simulate a realistic PDN behavior; then three applications which belong to the MiBench programs, were implemented in post-layout simulation. The behaviors during the simulation of all the components on the chip were monitored and recorded. A preparation to extract transient spatial voltage droop profile for real simulation data of LEON3 was set up. Then the net-list and simulation results for voltage droops were formatted in order to make them compatible with the machine learning algorithms. In the final training data set contains only the FFs, maximum droop of each simulation segment and the grid location. In the end the machine learning algorithms are applied.

Three different workloads data based on simulations were collected and through the data-set analysis results illustrated in the previous chapters, several data-set re-sampling techniques like *Random Under-Sampling and Over-Sampling*, *Cluster-Based Over-Sampling* etc. and some algorithmic ensemble techniques like GBC, RFC etc. which could be used for handling unbalanced class problems were illustrated with details. After considering the characteristics of our data-set and the objects of this thesis, *Gradient Boosting Classifier (GBC)*, *Random Forest Classifier (RFC)*, *Logistic Regression (LR)* and *Support Vector Machines (SVMs)* were chosen for the final experiments.

And since one of the object of this thesis is to reduce the dimension of trace buffer. In order to reduce the number of features, in other word, the total number of FFs that are monitored and recorded inside the trace buffer, the cascaded machine learning algorithms that were explored with the feature elimination technique *Recursive Feature Elimination (RFE).*

Conclusion We could observe from the experimental result that the trace buffer based electrical debugging using machine learning techniques is efficient than the traditional methods. Through the results that were illustrated previously, what could be easily understood is that GBC, RFC and LR could be used to predict and locate the electrical bugs with a pretty high accuracy score. After considering both accuracy score for locating the electrical bugs and the trace signal selection, GBC, RFC and LR genrated better results than SVM. Among them the feature reduction percentage does not very much and RFC almost always works well considering accuracy score. RFC has the highest accuracy 91.58% with 1.59% of total number of flip-flops are selected as trace signals to the trace buffer, it is the best performance compared to GBC, LR and SVM. And 1.59% of total number of flip-flops means that only 38 FFs were selected to be trace in the LEON3 microprocessor scenario. So, we believe that RFC is the best algorithms with the highest accuracy score, the low percentage of signals need to trace and small variety with current scenarios.

Future work more workload could explored and combined together for training in order to simulate a more realistic scenario, the trace signals reduction based on the machine learning techniques could be investigated deeper in order to reduce the trace buffer length and width more. And current work could be possibly improved by other machine learning algorithms or methods, for example neural network.

Bibliography

[1] Runjie Zhang, Ke Wang, Brett H Meyer, Mircea R Stan, and Kevin Skadron. Architecture implications of pads as a scarce resource: Extended results. *University of Virginia, Tech. Rep. CS-2014*, 1, 2014.

[2] Haihua Su, Frank Liu, Anirudh Devgan, Emrah Acar, and Sani Nassif. Full chip leakage estimation considering power supply and temperature variations. In *Proceedings of the 2003 international symposium on Low power electronics and design*, pages 78–83. ACM, 2003.

[3] Robert E Schapire and Yoav Freund. *Boosting: Foundations and algorithms*. MIT press, 2012.

[4] S. B. Akers. Binary decision diagrams. *IEEE Transactions on Computers*, C-27(6):509–516, June 1978.

[5] João P Marques-Silva and Karem A Sakallah. Boolean satisfiability in electronic design automation. In *Proceedings of the 37th Annual Design Automation Conference*, pages 675–680. ACM, 2000.

[6] Kypros Constantinides, Onur Mutlu, and Todd Austin. Online design bug detection: Rtl analysis, flexible mechanisms, and evaluation. In *Microarchitecture, 2008. MICRO-41. 2008 41st IEEE/ACM International Symposium on*, pages 282–293. IEEE, 2008.

[7] Luciano Lavagno, Louis Scheffer, and Grant Martin. *EDA for IC implementation, circuit design, and process technology*. CRC Press, 2006.

[8] Xiao Liu and Qiang Xu. *Trace-based post-silicon validation for VLSI circuits*. Springer, 2016.

[9] Radu Marculescu, Diana Marculescu, and Massoud Pedram. Switching activity analysis considering spatiotemporal correlations. In *Proceedings of the 1994 IEEE/ACM International Conference on Computer-aided Design*, ICCAD '94, pages 294–299, Los Alamitos, CA, USA, 1994. IEEE Computer Society Press.

[10] Michael G. Xakellis and Farid N. Najm. Statistical estimation of the switching activity in digital circuits. In *Proceedings of the 31st Annual Design Automation Conference*, DAC '94, pages 728–733, New York, NY, USA, 1994. ACM.

[11] Christer Svensson and Atila Alvandpour. Low power and low voltage cmos digital circuit techniques. In *Proceedings of the 1998 International Symposium*

on Low Power Electronics and Design, ISLPED '98, pages 7–10, New York, NY, USA, 1998. ACM.

[12] K. P. Parker and E. J. McCluskey. Probabilistic treatment of general combinational networks. *IEEE Transactions on Computers*, C-24(6):668–670, June 1975.

[13] S. Ercolani, M. Favalli, M. Damiani, P. Olivo, and B. Ricco. Estimate of signal probability in combinational logic networks. In *[1989] Proceedings of the 1st European Test Conference*, pages 132–138, Apr 1989.

[14] H. F. Ko and N. Nicolici. Algorithms for state restoration and trace-signal selection for data acquisition in silicon debug. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 28(2):285–297, Feb 2009.

[15] K. Rahmani, S. Proch, and P. Mishra. Efficient selection of trace and scan signals for post-silicon debug. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 24(1):313–323, Jan 2016.

[16] Arthur L Samuel. Some studies in machine learning using the game of checkers. *IBM Journal of research and development*, 3(3):210–229, 1959.

[17] Ji-Hyun Kim. Estimating classification error rate: Repeated cross-validation, repeated hold-out and bootstrap. *Computational statistics & data analysis*, 53(11):3735–3745, 2009.

[18] David Martin Powers. Evaluation: from precision, recall and f-measure to roc, informedness, markedness and correlation. 2011.

[19] Nasser M Nasrabadi. Pattern recognition and machine learning. *Journal of electronic imaging*, 16(4):049901, 2007.

[20] Stuart Jonathan Russell, Peter Norvig, John F Canny, Jitendra M Malik, and Douglas D Edwards. *Artificial intelligence: a modern approach*, volume 2. Prentice hall Upper Saddle River, 2003.

[21] Pat Langley. The changing science of machine learning. *Machine Learning*, 82(3):275–279, 2011.

[22] K. Rahmani, P. Mishra, and S. Ray. Scalable trace signal selection using machine learning. In *2013 IEEE 31st International Conference on Computer Design (ICCD)*, pages 384–389, Oct 2013.

[23] F. Brglez, D. Bryan, and K. Kozminski. Combinational profiles of sequential benchmark circuits. In *IEEE International Symposium on Circuits and Systems,*, pages 1929–1934 vol.3, May 1989.

[24] F. Corno, M. S. Reorda, and G. Squillero. Rt-level itc'99 benchmarks and first atpg results. *IEEE Design Test of Computers*, 17(3):44–53, Jul 2000.

[25] Special session: Recent algorithmsfor gate-level atpg with fault sirnularion and their performance assessment. In *IEEE International Symposium on Circuits and Systems,*, pages pp. 663–698, June 1985.

[26] Jiri Gaisler, Edvin Catovic, Marko Isomaki, Kristoffer Glembo, and Sandi Habinc. Grlib ip core user's manual. *Gaisler research*, 2007.

[27] R. Zhang, K. Wang, B. H. Meyer, M. R. Stan, and K. Skadron. Architecture implications of pads as a scarce resource. In *2014 ACM/IEEE 41st International Symposium on Computer Architecture (ISCA)*, pages 373–384, June 2014.

[28] R. Zhang, K. Mazumdar, B. H. Meyer, K. Wang, K. Skadron, and M. Stan. A cross-layer design exploration of charge-recycled power-delivery in many-layer 3d-ic. In *2015 52nd ACM/EDAC/IEEE Design Automation Conference (DAC)*, pages 1–6, June 2015.

[29] R. Zhang, K. Mazumdar, B. H. Meyer, K. Wang, K. Skadron, and M. R. Stan. Transient voltage noise in charge-recycled power delivery networks for many-layer 3d-ic. In *2015 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*, pages 152–158, July 2015.

[30] W Fichtner. Physics and simulation of small mos devices. In *Electron Devices Meeting, 1982 International*, pages 638–641. IEEE, 1982.

[31] R. E. Bank, W. M. Coughran, W. Fichtner, E. H. Grosse, D. J. Rose, and R. K. Smith. Transient simulation of silicon devices and circuits. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 4(4):436–451, October 1985.

[32] H. He and E. A. Garcia. Learning from imbalanced data. *IEEE Transactions on Knowledge and Data Engineering*, 21(9):1263–1284, Sept 2009.

[33] Jerome H Friedman. Greedy function approximation: a gradient boosting machine. *Annals of statistics*, pages 1189–1232, 2001.

[34] Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

[35] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[36] Leo Breiman et al. Arcing classifier (with discussion and a rejoinder by the author). *The annals of statistics*, 26(3):801–849, 1998.

[37] Payam Refaeilzadeh, Lei Tang, and Huan Liu. Cross-validation. In *Encyclopedia of database systems*, pages 532–538. Springer, 2009.

[38] William H Greene. *Econometric analysis*. Pearson Education India, 2003.

[39] I. Guyon, B. Boser, and V. Vapnik. Automatic capacity tuning of very large vc-dimension classifiers. In *Advances in Neural Information Processing Systems*, pages 147–155. Morgan Kaufmann, 1993.

[40] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine Learning*, 20(3):273–297, Sep 1995.

[41] Isabelle Guyon, Jason Weston, Stephen Barnhill, and Vladimir Vapnik. Gene selection for cancer classification using support vector machines. *Machine learning*, 46(1-3):389–422, 2002.