Comparative Power Analysis of RISC Processor using Machine Learning Algorithms

A DISSERTATION

SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE AWARD OF THE DEGREE

OF

MASTER OF TECHNOLOGY

in

VLSI DESIGN AND EMBEDDED SYSTEMS

Submitted by:

PRIYA

2K19/VLS/11

Under the guidance of

Dr. MALTI BANSAL
Dept. of Electronics & Communication Engineering,
Delhi Technological University (DTU)

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING
DELHI TECHNOLOGICAL UNIVERSITY
(Formerly Delhi College of Engineering) Delhi-110042

JUNE 2021

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING
DELHI TECHNOLOGICAL UNIVERSITY
(Formerly Delhi College of Engineering) Delhi-110042

CANDIDATE'S DECLARATION

I Priya, Roll No. 2K19/VLS/11, student of M.Tech (VLSI and Embedded Systems), hereby declare that the Project Dissertation titled **"Comparative Power Analysis of RISC Processor using Machine Learning Algorithms"** which is submitted by me to the Department of Electronics and Communication Engineering, Delhi Technological University, Delhi, in the partial fulfillment of the requirement for the award of the degree of Master of Technology, has been done under the guidance of my project mentor **Dr. Malti Bansal**, Assistant Professor, Department of Electronics & Communication Engineering, Delhi Technological University (DTU), is original and not copied from any source without proper citation. This work has not previously formed the basis for the award of any Degree, Diploma Associateship, Fellowship or other similar title or recognition.

Place: Delhi **PRIYA**
Date:25.04.2021 (2K19/VLS/11)

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING
DELHI TECHNOLOGICAL UNIVERSITY
(Formerly Delhi College of Engineering),Delhi-110042

CERTIFICATE

I hereby certify that the Project Dissertation titled **"Comparative Power Analysis of RISC Processor using Machine Learning Algorithms"** which is submitted by **PRIYA**, **2K19/VLS/11** of Electronics and Communication Department, Delhi Technological University, Delhi, in the partial fulfillment of the requirement for the award of the degree of Master of Technology, is a record of the project work carried out by student under my supervision. To the best of my knowledge this work has not been submitted in part or full for any Degree or Diploma to this University or elsewhere.

Place: Delhi
Date: 25.04.2021

Dr. MALTI BANSAL
Assistant Professor
Department of ECE

ACKNOWLEDGEMENT

A successful project can never be prepared by the efforts of the person to whom the project is assigned, but it also demands the help and guardianship of people who helped in completion of the project. I would like to thank all those people who have helped me in this research and inspired me during my study.

With profound sense of gratitude, I thank my Research Supervisor **Dr. Malti Bansal**, Assistant Professor, Department of Electronics & Communication Engineering, Delhi Technological University (DTU), for her encouragement, support, patience and her guidance in this project work. I heartily appreciate the guidance given by her in the project presentation that has improved my presentation skills with her comments and advices.

I take immense delight in extending my acknowledgement to my family and friends who have helped me throughout this project work.

PRIYA
2K19/VLS/11

ABSTRACT

The exigency of automation and advancements ushers machine learning and artificial intelligence to expand it dimensionality into numerous domains like Internet of Things (IoT), Military, product and market analytics, language and sentiment analysis and Very Large Scale Integration of chips is also one of them. ML has started transuding as an significant application in the development and evolution of the Computer Aided tools and technologies that are involved in VLSI domain in the design of ASICs or FPGAs. The SoCs, Macros, IPs and Processors etc. otherwise would take large turnaround time to spin off as a final chip. As RISC Processors are being extensively used due to their less flexibility and high performance as compared to CISC Processors. We focused on the study, design and imposition of a RISC Processor on the Register Transfer Level (RTL) and apply machine learning algorithm to predict and analyze its power consumption. And also we compare which machine learning algorithm fits the power dataset of RISC Processor in the best possible manner in terms of the performance metrics.

In the thesis, a 32-bit, MIPS based RISC Processor is implemented which supports basic Instruction Set Architecture (ISA) to perform few simple arithmetic computations like ADD, SUB, MUL etc. This processor is simulated using Xilinx VIVADO and its power is calculated and stored as a dataset under various input conditions often called as dataset features or attributes. The obtained power dataset is analyzed graphically to know on which factor different types of power majorly depends. Furthermore, machine learning algorithms are applied to obtain the classification report stating the accuracy, precision and recall on the power dataset of the RISC Processor to state which Machine learning algorithm is best fit for the Power dataset generated. In a nutshell, the thesis focuses on the application and relative analysis on performance metrics of machine learning algorithms on the power dataset of RISC Processor.

CONTENTS

LIST OF FIGURES

LIST OF ABBREVIATIONS

RISC	Reduced Instruction Set Computer
CISC	Complex Instruction Set Computer
IR	Instruction Register
ID	Instruction Decode
PC	Program Counter
NPC	Next Program Counter
IF	Instruction Fetch
EX	Execution
MEM	Memory Access
WB	Write Back
IM/DM	Instruction Memory/Data Memory
ML	Machine Learning
AI	Artificial Intelligence
SVM	Support Vector Machine
IF	Isolation Forest
LOF	Local Outlier Factor
KNN	K-Nearest Neighbor
NB	Naïve Bayes
HDL	Hardware Description Language
RTL	Register Transfer Level
FPGA	Field Programmable Gate Array
IP	Intellectual Property

LIST OF TABLES

CHAPTER 1

INTRODUCTION

1.1 INTRODUCTION TO RISC PROCESSOR

In the late 1970s, Researcher who worked with IBM, John Coke and his team developed the first processor prototype that deployed Reduced Instruction Set Computer (RISC) architecture [1]. This architecture has reduced number of instructions which are simpler to implement. Also, RISC architecture has fixed and same bits of instructions, register file and memory which allows the fetch, decode, execute, access to the memory and write back to register file, each operation to be performed in exactly 1 clock cycle. Due to this reason, RISC architecture is less flexible but delivers high performance by increasing the speed of operations.

It only has two operations to be performed with the memory i.e. LOAD and STORE. All the other operations and arithmetic computations are done with the register files. Since interactions with the memory takes more time or more number of clock cycles than what is taken by the registers present in the architecture, and RISC processors mainly performs operations on registers only, makes RISC Processors faster and efficient enough to implement pipelining in the best way possible [2]. LD(load) directive allows to access the data/instructions from the memory and ST(store) directive allows the processor to store the data /results back into the memory.

However, there is other type of processors also based on Complex Instruction Set Computer (CISC) Architecture. The main difference is that they support variable sizes of instructions for example a CISC Processor of 32 bit can support instructions of 8 bit, 16 bit or 32 bit which means that one instruction may or may not get fetched, decoded and executed in one clock cycle. This adds to the complex nature of CISC processor. Although CISC processors are flexible in terms of executing any type of instructions but, performance is greatly affected since many instructions in this processor might take more cycles or more memories to be processed. Speeds of such processors are very slow comparative to the RISC processors.

Most of the operations in the CISC Processor involve the interactions with the memory which adds to more number and types of instructions. Since interaction with the memory is slower process than the interaction with register, this is one more reason that why CISC Processors are slower. CISC Processors have lesser numbers of Registers but definitely have more number of instructions and more number of modes to address those instructions. The major disadvantage of CISC processor is that the pipelining implementation is never effectively done in CISC Processor which adds to the fact that CISC Processors although being too flexible but compromises in the performance part.

1.2 RISC PROCESSOR ARCHITECTURE

The RISC processor that has been implemented has 32-bit core with separate data and instruction memory (Harvard architecture) where instruction memory is generally read only memory and data memory is both read-write memory. The register transfer level code is written in Verilog language and named as RISC.v. It is instantiated in the test bench named RISC_TB.v file. The code also has a double bank of 16 registers (Register File) to allow two simultaneous reads in addition to one write operations from the register file. Also, it has a set of instructions of fixed size but different structure depending on the addressing modes and the specific instructions.

There are five stages incorporated in executing a instruction in 32 bit RISC i.e. IF, ID, EX, MEM and WB as shown in Figure below. Figure 1 shows the basic

Architecture which shows the pipelining stages in the RISC Processor.

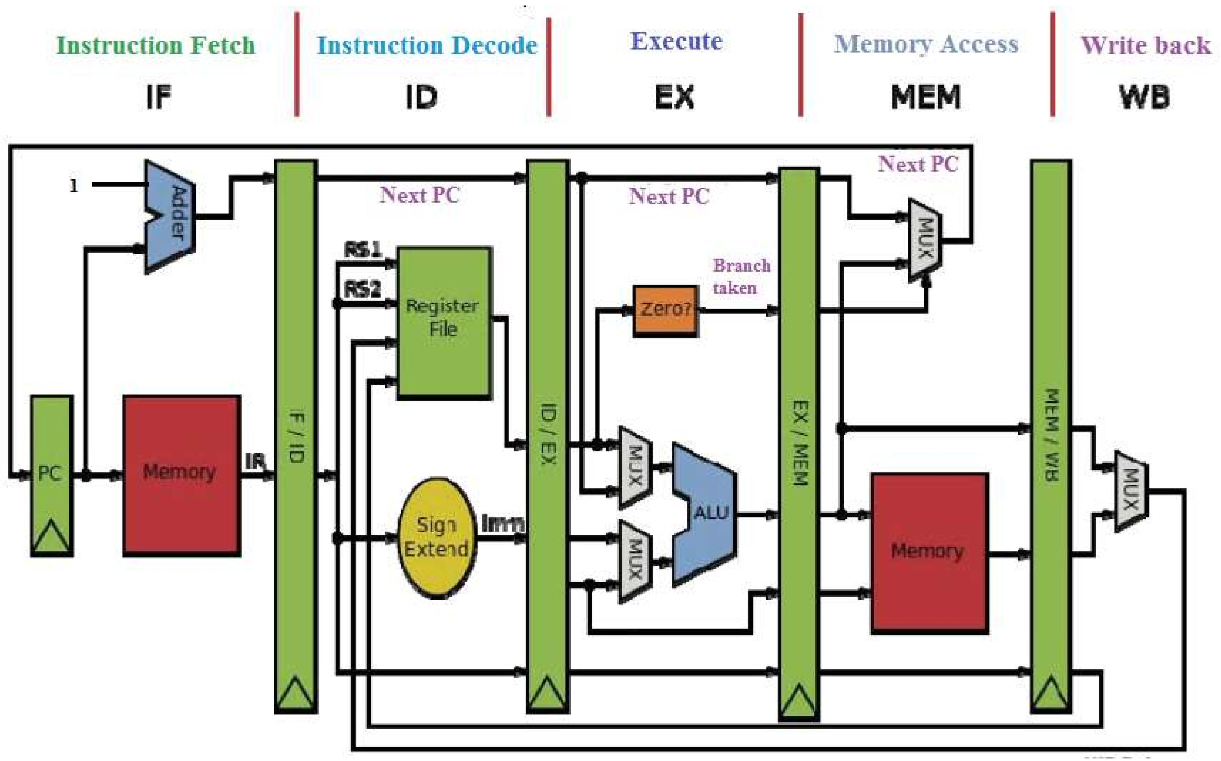


Figure 1.1 Basic Architectural block diagram of RISC Processor [3].

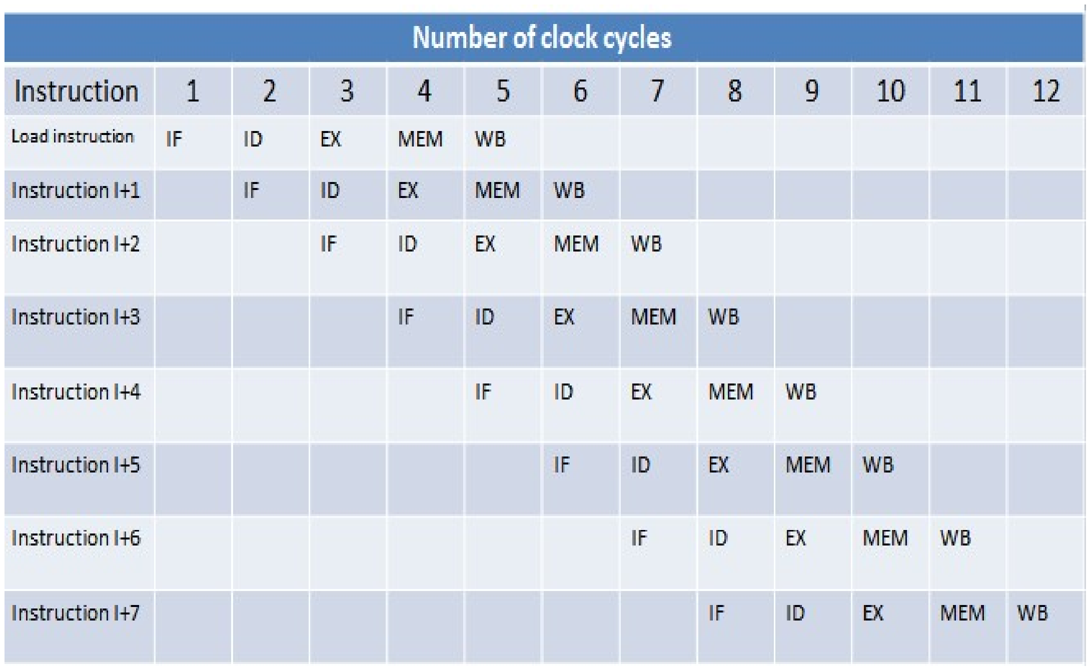
There are five stages in a processor pipeline to perform a specific operation. This is coded in hexadecimal instruction format. The IF stage allows fetching of succeeding instruction by accessing the address of IM stored in program counter and stores the fetched instruction into the Instruction Register (IR). Next stage is the Instruction Decode (ID) which calculates the next PC (NPC), decodes the Instruction stored in IR and performs read operations from the register file (reading rs and rd registers). In the Execution stage, based on the opcode and funct decoded in the previous stage, ALU performs the operations on the operands and stores the value in intermediate register called ALUOut. Memory Access stage allows operations on data to be performed to and from the memory such as load and store(LD/ST). Final stage is the Write Back Stage which perform the Write operation on the register file [3].

1.3 PIPELINING AND ITS HAZARDS

Pipelining is one of the prominent attribute of RISC Processors which when incorporated to the architecture helps in improving the throughput and performance of the processors[3]. Pipelining divides the entire execution of a Instruction (particularly a task like arithmetic computations) into multiple stages, often the operations executing at these stages are called micro-operations. These stages help the processors to work on multiple numbers of instructions at a time.

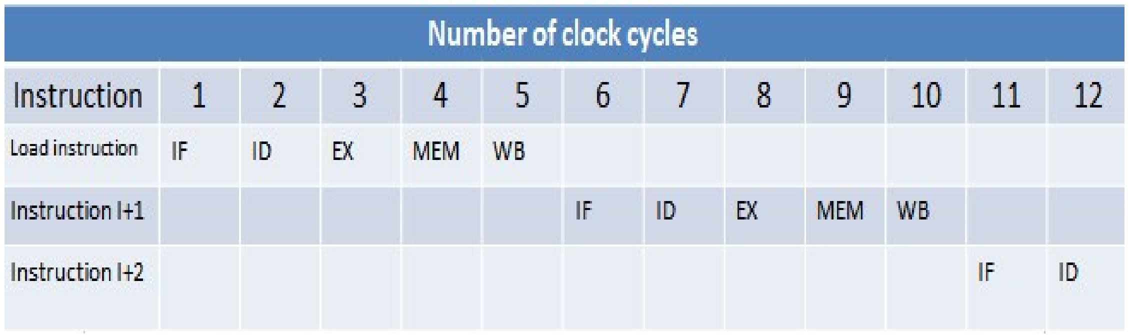
For example: In the RISC processor designed, there are five stages as we discussed in the previous section. Suppose we have our Instructions say, I_0, I_1, I_2, I_3…so on. In first cycle of clock, I_0 will be fetched. In the second clock cycle, I_0 will move to second stage i.e. I_0 will be decoded. And by the time I_0 is decoded, processor can fetch I_1. Processor handles two instructions simultaneously in different stages. In the third clock cycle, I_0 moves to third stage i.e. execution, I_1 moves to decode stage and I_2 moves fetch stage. Here, Processor handles three instructions simultaneously in different stages. In the fourth clock cycle, I_0 moves to fourth stage i.e. memory access, I_1 moves to execution stage and I_2 moves decode stage and I_3 moves to fetch stage. So now, Processor handles four instructions simultaneously in different stages. In the fifth clock cycle, I_0 moves to fifth stage i.e. write back, I_1 moves to memory access stage and I_2 moves execution stage and I_3 moves to decode stage, I_4 moves to fetch stage. So now, Processor handles five instructions simultaneously in different stages. In the coming next cycles, new instruction enters fetch stage and one Instruction completes the entire execution and gets out of the pipeline.

This simultaneous execution of multiple instructions at a time increases the efficiency of the processor. The processor does sit idle waiting for the next instruction.



	Number of clock cycles											
Instruction	1	2	3	4	5	6	7	8	9	10	11	12
Load instruction	IF	ID	EX	MEM	WB							
Instruction I+1		IF	ID	EX	MEM	WB						
Instruction I+2			IF	ID	EX	MEM	WB					
Instruction I+3				IF	ID	EX	MEM	WB				
Instruction I+4					IF	ID	EX	MEM	WB			
Instruction I+5						IF	ID	EX	MEM	WB		
Instruction I+6							IF	ID	EX	MEM	WB	
Instruction I+7								IF	ID	EX	MEM	WB

Figure 1.2 Instruction execution in pipelined processor



	Number of clock cycles											
Instruction	1	2	3	4	5	6	7	8	9	10	11	12
Load instruction	IF	ID	EX	MEM	WB							
Instruction I+1						IF	ID	EX	MEM	WB		
Instruction I+2											IF	ID

Figure 1.3 Instruction execution in non-pipelined processor

If we have k number of stages and n number of instructions, then the overall execution of all the instructions will take (n+(k-1) clock cycles which very less than the non- pipelined execution where number of clock cycles for the same is (n*k). For example 50 instructions in 5 stages of pipeline takes 54 cycles in a pipelined processor where as non-pipelined processor would take 250 clock cycles. Here, non-pipelined processor is 5 times slower and time consuming than the pipelined one. This is the reason why performance of pipelined processors like RISC is very high. As far as throughput is concerned, pipelined processor automatically has higher throughput since throughput is number of instructions per clock cycle. Pipelined processor after first instruction completes its entire execution; the every next clock cycle will then complete one instruction means its nearly one instruction per clock cycle. In case of non-pipelined processor, in every 5 clock cycle, one instruction is executed. So, the throughput is very less in non-pipelined processor.

Also, the most important requirement while implementation of pipelining is the storing the results of the micro-operations because if each stage is connected to next one directly, there is possibility that data may change before it is used in the next stage while the current stage is working on new data changes the input to next stage which needs to work on previous data. So, to store the results of each stage such that it does not change, we need to use latches or registers in between two stages so that the output of a particular stage is not changing in between the micro-operations and this result in the successful functioning of pipeline. These latches/registers are the part of register file of Processor used to store the data. They are named as ID_IF_IR, IF_EX_IR, IF_EX_A, IF_EX_B, IF_EX_NPC etc.

Hazards in pipelining: There are some challenges faced during the proper implementation of pipelining. 100% implementation of pipelining of Processors is not possible due to some reasons such as data hazards, structural hazards and control hazards.

1) Data hazards: This can be understood with the help of the following example. I_0 : ADD R5 R3 R4; I_1 : ADD R6 R5 R7. In this example, I_0 instruction executing first, writes the final result in 5th stage and I_1 instruction reads the data stored in R5 in the second stage. So, I_1 accesses the data before the final result gets written in register R5. In such a case, I_1 accesses the wrong data present in the R5. Hence, it leads to the incorrect result. This hazard can be removed by introducing the dummy instruction. For example: I_0 : ADD R5 R3 R4; I_1 : ADD R5 R5 R0; I_2 :

ADD R6 R5 R7. Here, I_1 is dummy instruction. Another solution can be renaming the registers or using different registers. Latter solution needs more memory and clock cycle and former one only need extra clock cycle.

2) Structural hazards: This hazard occurs when there is conflict at different stages for different instruction to use the same resource such as memory or a register simultaneously. If both the instructions are trying to access the memory, let's say one for reading the instruction and another for reading the data from memory. In such case, one instruction gets the resource but other one waits for it thereby killing the pipelining. Solution for this hazard is that to use different memory for the instruction such as Instruction memory and different memory to store and access the data.

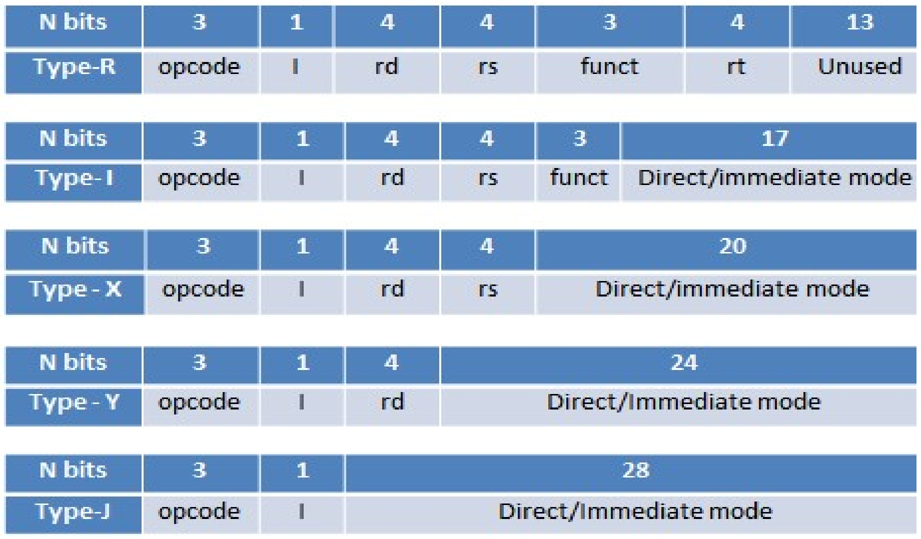
3) Branch hazards: This hazard occurs when there are some branch instructions in the program. Whenever branch is encountered the execution of upcoming instruction has to be stopped and the execution has to be carried out from the address indicated by the branch instruction. It is generally not the next instruction indicated by the Program Counter. It has the address other than the next instruction. Hence, the next instruction has to be stopped from changing or writing the value in the memory. Also, we cannot stop the entire processor at a time after encountering a branch instruction because previous instructions are also in the ahead stages and they should be allowed to complete their write operations to the memory. Solution to above hazard is the use of two flag registers such as Halted and Branch Taken. Whenever branch instruction is decoded, these flag registers should be set to 1. Next instructions that are already fetched should write in the memory only when Branch Taken register is set to 0. If Branch Taken register is set to 1, write operations for those instructions should be disabled. And instruction fetching should start again when Halted register is set to 0 after the execution of

the branch instruction.

Due to presence of these three hazards during the pipelining of the instruction, the pipelining is never implemented in the excellent manner but it does gives us advantages over CISC processor where the implementation of pipelining is even poorer.

1.4 INSTRUCTION SET FORMAT

A designer can control the distribution of bits in the instruction frame depending upon the applications to be performed and operations to be performed. Also, it can be designed in a way so that processor can run in various addressing modes. So, while deciding the position and number of the fields in each type of Instruction Frame, it was taken into account which bits were going to be used and which ones were not in each case, mainly so that there is no need to change their position in other configurations/ modes that shared that field (rd , rs , funct and rt). The following Figure 2 depicts the distribution of the bits in the instruction frame.



N bits	3	1	4	4	3	4	13
Type-R	opcode	I	rd	rs	funct	rt	Unused

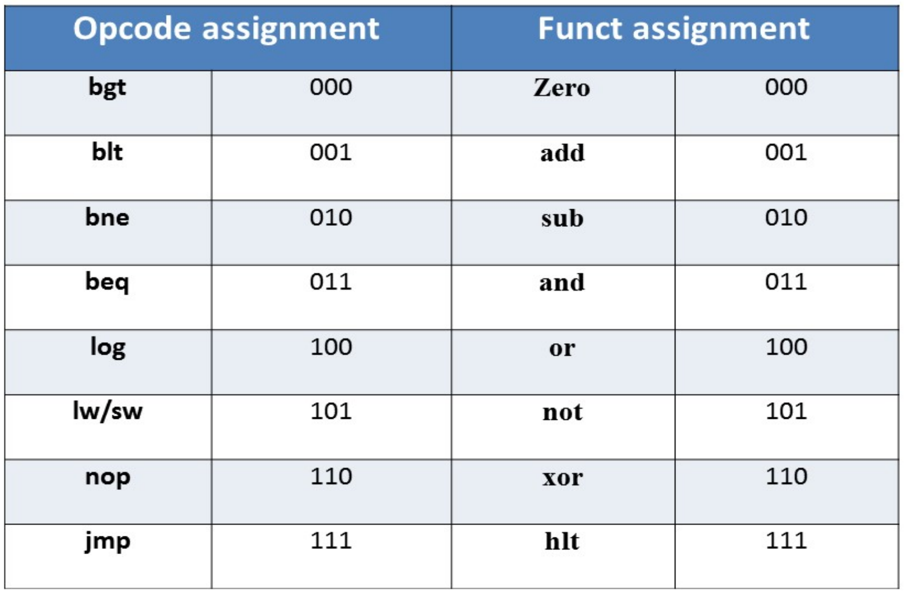
N bits	3	1	4	4	3	17
Type-I	opcode	I	rd	rs	funct	Direct/immediate mode

N bits	3	1	4	4	20
Type - X	opcode	I	rd	rs	Direct/immediate mode

N bits	3	1	4	24
Type - Y	opcode	I	rd	Direct/Immediate mode

N bits	3	1	28
Type-J	opcode	I	Direct/Immediate mode

Figure 1.4 Distribution of the bits in the Instruction Frame.



Opcode assignment		Funct assignment	
bgt	000	Zero	000
blt	001	add	001
bne	010	sub	010
beq	011	and	011
log	100	or	100
lw/sw	101	not	101
nop	110	xor	110
jmp	111	hlt	111

Figure 1.5 Opcode and Funct assignment in Instruction Frame.

Each of the four immediate type distributions corresponds to a specific type with 2 bits that are used to identify them in the Processor: I = 00, X = 01, Y = 10, J = 11.

When deciding the encoding of the opcode and the arithmetic and logical functions (funct field), this frame had to be re-used throughout the design as conditional jump share a bit that differentiates them from the rest of the instructions to use it directly in the control signals. And to save program bits, arithmetic-logical functions share opcode and use a funct field.

For Example, If the functions are to be 8 and fit in 3 bits, the load and store commands took advantage of the fact that they did not need to use the bit that differentiated immediate from non-immediate functions, and were joined in the same opcode, differentiating them precisely with that bit. It was also necessary to make a change in the encoding of the opcode because it was chosen for the address 000 for the beq function, which when resetting the microcontroller caused that the equal operands (they were 0 due to the reset), it always executed the jump command and did not it never worked. To decide the code of the “funct” field, it is only necessary to comment that when required only 6 functions, there were two valid codes that had no implementation. So it was decided to make those codes give an output with the 32 bits at the logic level low to take advantage of this function in the ALU to generate a one in the zero output and thus implement by means of “funct” = 000 or “funct” = 111 the unconditional jumps. All instructions have an associated "funct" function, but only arithmetic operations indicate this function in a field of its plot, the rest are generated by the decoder according to the opcode it receives.

Bit "I" (4th bit) differentiates if it is immediate operation from non-immediate except for lw / sw which identifies precisely whether it is lw or sw. The default value for “I” bit is 0 which is non-immediate or lw where as I = 1 is immediate or sw.

The decoder generates the following control signals : RegWrite: which helps in activating the write input of the register bank. MemtoReg: which helps in indicating whether the command output comes from the ALU (0) or from memory.

(i) MemWrite: helps in activating the write input of the data memory.

(ii) Branch: helps in indicating if the order is a jump (1) or not (0).

(iii)Funct: will tell which ALU operation to be performed (it only generates it if opcode is not 100). Type : iindicates the type immediately, if it is 17 bits (00), 20 bits (01), 24 bits (10) or 28 bits (11).

(iv) ALUsrc: iindicates if the operation is on a register (0) or an immediate (1) [in the case of jump indicates if it is addressed to imm + reg (1) or only to imm (0)].

(v) mbs: by default it is always low level, except for conditional jump bgt and blt, since it indicates that the output that is evaluated at the output of the ALU to jump is the heavier bit instead of zero.

(vi) Output: by default it is always at a low level, except for the conditional jumps bne and bgt, since it inverts the signal that indicates whether the jump occurs, since for bne and bgt it would be 0 instead of 1.

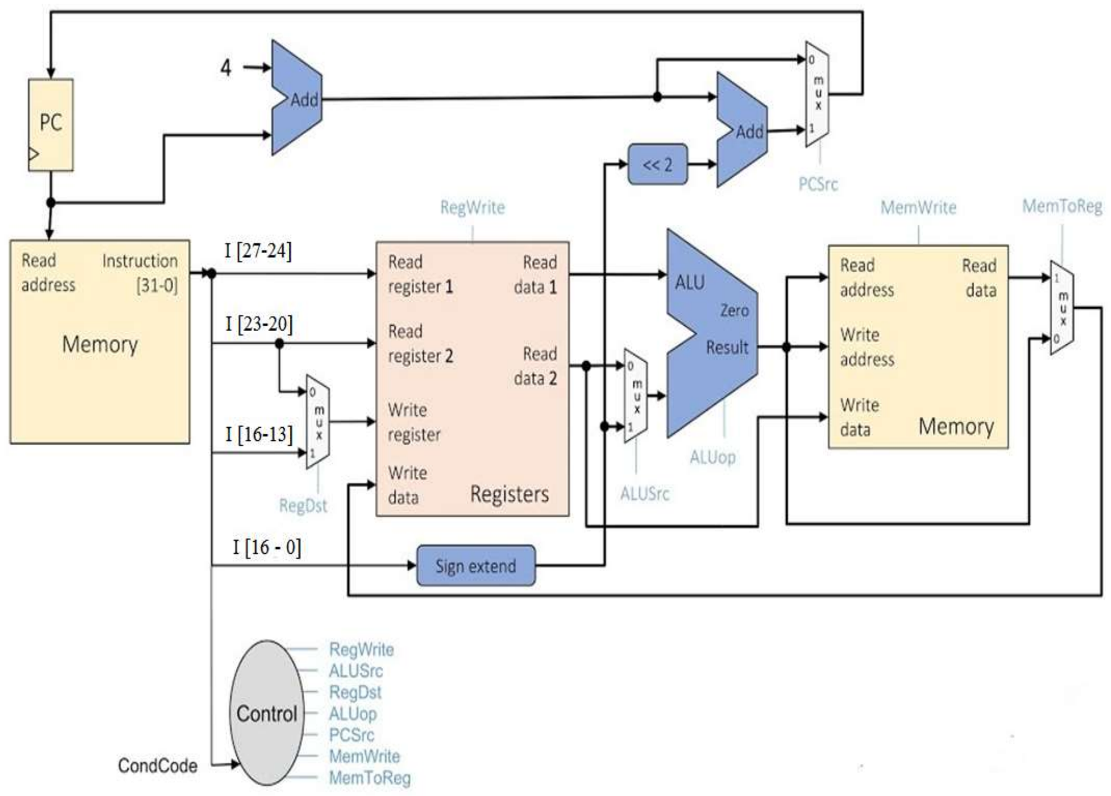


Figure 1.6 Distribution of Instruction bits in the Datapath

1.5 ADVANTAGES OF RISC PROCESSORS

1. Power Consumption: RISC processors consume and dissipate lesser power due to execution of simpler operation and dealing with the simpler hardware. The lesser the hardware, the lesser will be power consumption at device level. It has two advantages. Longer use of battery. No need form cooling of device. Smaller device design without noise.

2. Simpler hardware: All the components of processor are fabricated on a same chip. Smaller chips allow a semiconductor manufacturer to place more parts on a single silicon wafer, which can lower the per-chip cost dramatically.

3. Less area: The chip area used in the design of control unit is considerably reduced. More area is available for additional features.

4. Excellent performance: since in RISC processor, pipelining can be implemented in the excellent way. Number of cycle to execute a program drastically reduces

leading to the excellent performance. Also, as no. of bit processing per second is increasing, the operations are performed at faster rate which increases the performance.

5. Addressing modes: RISC processors use less number of addressing modes since majority of the read and write operations are performed on the registers. Very few operations such as only load and store are the operations are done with memory. This is the main reason lesser ways of accessing the instructions are available since lesser operation are on memory.

6. Memory: These processors have lesser use of memory hence memory consumption to store the data is less. Only results of the computation and data on which computation is done are stored in memory. Instead large numbers of on-chip registers are used to store intermediate results. This is one of the reasons that these processors are faster since we know that interactions with memory can take multiple cycles.

1.6 APPLICATIONS OF PROCESSOR

1. Single board Microcomputers: simplest, cheapest general purpose processor based device with the minimum possible hardware and software attached. Used for universities and educational institutional for delivering training to the student and in industries for evaluation of the processor or for building prototype systems.

2. Terminals: Used for communication between a device and its user. Instructions and data to be fed into the computer are given by the help of keyboard and mouse, and the output of the computer is displayed on the monitor screen.

3. Personal computers: Used for playing games and learning simple programming and 16-bit processors are used for word processing, payroll, business accounts, medical record keeping and inventory control.

4. CAD Machines: execute powerful functions, with their increased word length and memory size. Used to make powerful microcomputer. Used as an

engineering workstation and computer aided design (CAD) machines.

5. Communication: Used in a wide range of communication equipment. Used in digital telephone sets, telephone exchanges and modems. Widespread use of processors in radio, telephone, television and satellite communication, teleconferencing, railway reservation systems at the national level and air reservation systems at the international level.

6. Instrumentation: Processor based frequency meters, frequency synthesizers, function generator, spectrum analyzer and controllers are used in medical instrumentation, e.g. patient monitoring in intensive care unit, pathological analysis and the measurements of parameters like blood pressure and temperature.

7. Control: Controllers are used in home appliances such as microwave oven, washing machine. Used in controlling various PVT parameters such as speed, temperature and pressure.

8. Automation and Publication: Instead of typing, drafting and fling, processor based systems are applied in word processing, excel sheet operation, storing and retrieving the information from secondary devices. In publishing houses, used for automating phototypesetting directly from the output of the word process or system.

9. Consumers: Used in toys, entertainment equipment and home appliances with novel features.

CHAPTER 2
MACHINE LEARNING PERSPECTIVES

2.1 INTRODUCTION TO MACHINE LEARNING IN VLSI

Machine learning is a field growing rapidly which has brought up a revolution in the world by combining the advancements in Artificial Intelligence(AI) and data science that comes up with the systems and solutions which can learn from the prior experiences or preexisting information or data and improve without being implicitly and comprehensively programmed for the same i.e. evolving the methods or solutions without help of human intelligence which is more of developing its own intelligence to deal with a problem. In a nutshell, Machine Learning is the enabling the computers with human intelligence[4]. Machine learning revolves around finding and creating effective and efficient learning algorithms that assists the machines to analyze the available data and train itself to accurately predict for the unknown data samples.

In 1950's, a computer program for playing checkers was created by Arthur Samuel who worked with IBM in the AI domain. In this program, Samuel used a scoring mechanism using the locations of the pieces on board and tried calculating the probability of each side winning. Using a minimax technique, the program selects its next move and this technique eventually got developed into Minimax algorithm. A variety of mechanisms were also developed by Samuel to allow his program become better. His program remembered all the locations on the board that was valued for the reward function. His program had learned from the available data samples and predicted the reward function. Arthur Samuel was the first who coined the term Machine Learning [5]. ML is progressively developing in numerous spheres like Quantum computing, Robotics, Data Mining, VLSI, Automation, Signal Processing, Artificial Intelligence, Internet of Things (IoT), Medical, Military etc. As ML is majorly working on neural networks and logical & computational algorithms, it makes a system both smartly accurate and highly beneficial. Due its accuracy, reliability, efficiency and ability to improve, ML is excessively effective over the human biological intelligence [6].

2.2 THE BASIC PARADIGM OF MACHINE LEARNING

The basic model of Machine Learning consists of few steps starting from the Collection of data samples from the experiences. This is the preexisting data samples that help the machine to learn a given task. Data is conditioned by removing errors, missing values, repeated values etc. Next step chooses an algorithm model according to aptness with different tasks/problems /data.. Now, the model is trained iteratively followed by the evaluation in which the model is tested against unknown data samples. Further, tuning of the model parameter is done to increase the performance of the model proceeded with the final step of making the predictions in the real scenario. The performance in the tasks improves as the machines gains experience while executing the tasks and updating the model each time [5]. The basic model and its process has been represented in the block diagram Figure2.1.

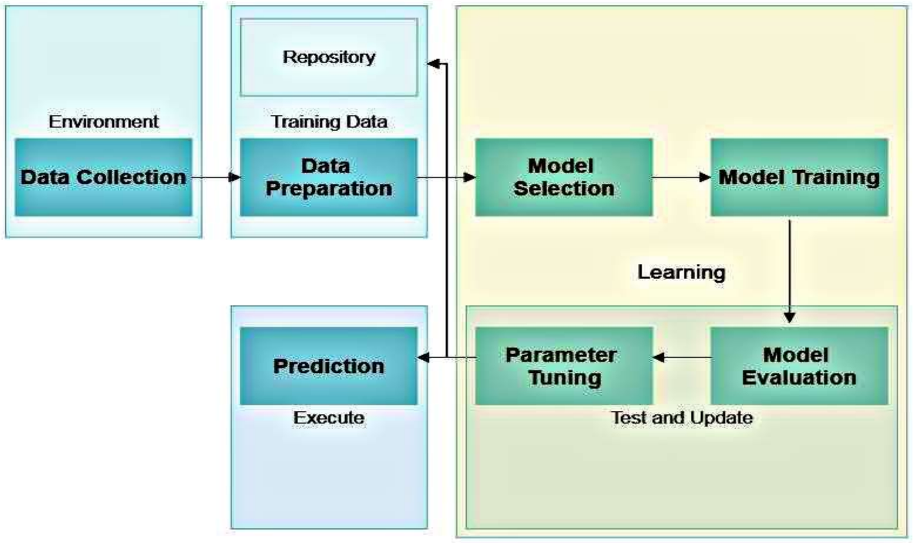


Figure 2.1 Basic model of Machine Learning with Process Steps

2.3 AREAS OF MACHINE LEARNING

ML has uncountable number of applications and it offers solutions to many real time issues. Some of the areas where machine learning is being progressively applied are mentioned below[5]: i)Face Detection and Recognition ii) Visual Perception iii)Classification iv)Adaptive Systems v) Modeling vi) Speech and Image processing vi)Automation vii) Problem Solving viii) Genetics ix)Anomalies Detection x)Games xi)Internet of Things (IoT) xii) Quantum computing xiii) Medical Diagnosis xiv) VLSI xv) Stock Market Trading xvi) Virtual Personal Assistant xvii) Online Fraud Detection xviii) Speech Recognition

2.4 MACHINE LEARNING ALGORITHMS

The highest level of abstraction in machine learning methods is based on the source of data/information that directs into the learning[7]. It is broadly classified into three categories. These are:

i) Unsupervised Learning
ii) Supervised Learning
iii) Semi-Supervised.

In unsupervised learning, only input data is available and some structure or label needs to be developed to distinguish between the input data samples. In supervised learning, input and corresponding output data samples are available along with the structure/ labels. In semi-supervised learning, only some fraction of input data samples have corresponding output pairs i.e. few of them are labeled or structured[8]. Figure 2.2 represents the basic classification of machine learning methods and the algorithms used for learning in different methods.

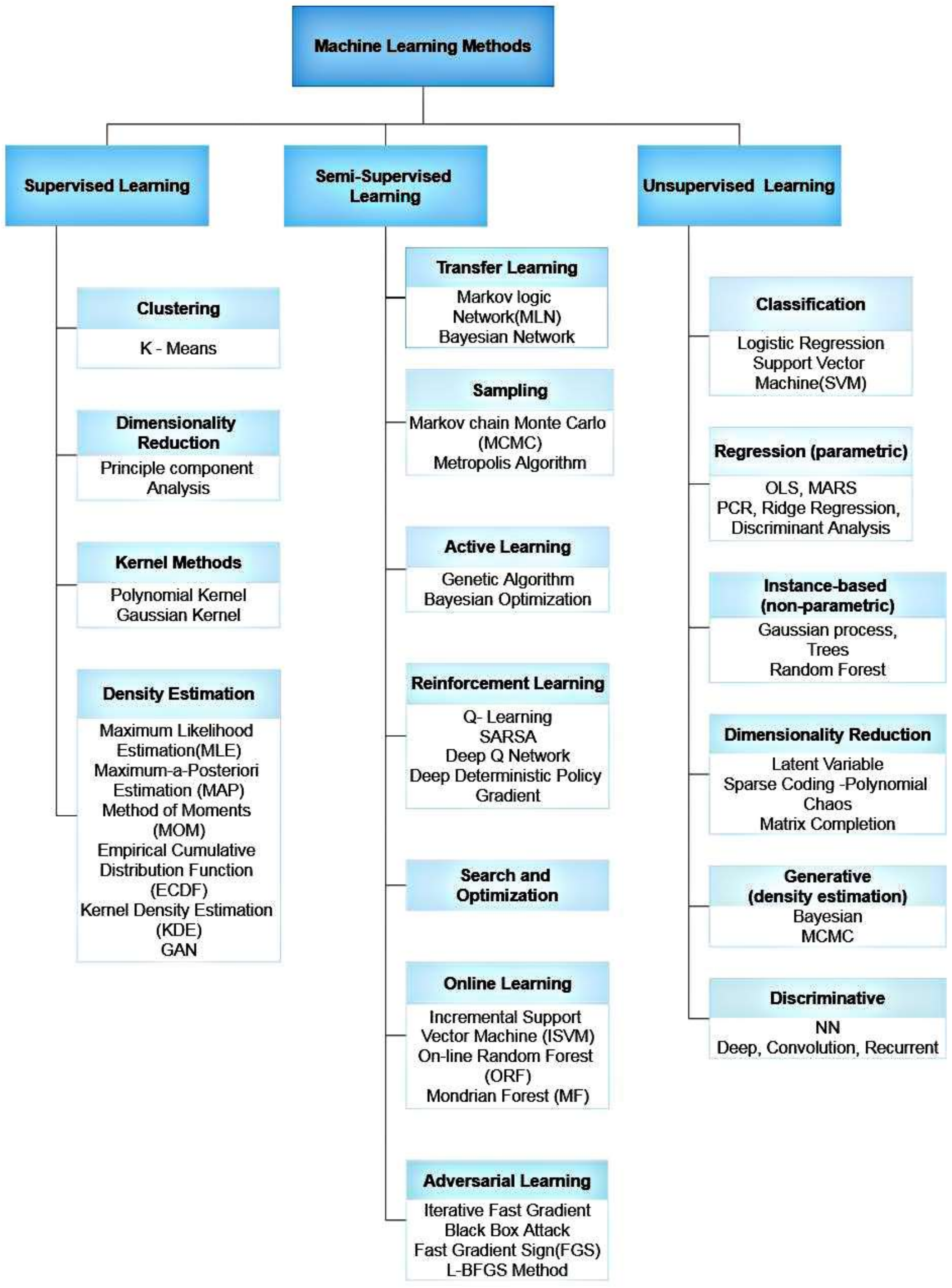


Figure 2.2 Classification of Machine Learning Algorithms

A brief summary of different Machine Learning algorithms has been represented in the form of table that briefly lists the Uses, Advantages and its Drawbacks which can be useful in the appropriate selection of algorithm is given below.

Table 2.1 Machine Learning algorithms and their Uses, Advantages and its Drawbacks.

S. no.	***Machine Learning Algorithms***			
	Name of Algorithm	***Uses***	***Advantages***	***Drawbacks***
1.	Gradient Descent	To minimize cost function	Efficient , Stable error gradient	Never converges for too high and too low learning rate
2.	Linear Regression	Models continuous variables, prediction, Data analysis process	Easier to understand, Easy to avoid Over fitting	Not a good fit for nonlinear relationships, cannot handle complex pattern, over simplifies real word issues
3	Multi –Variate Regression Analysis	Used on number of independent variable and single dependent variable	Deeper insight to relationship between variables Models complex real time issues, Realistic and practical	Complex, High knowledge is required for modeling, sample size needs to be high, difficult to analyze
4	Logistic Regression	Used on classification problem	Simple to implement, Ease of regularization, Efficient in computation and training, no scaling required, reliable	Unable to solve nonlinear problem, prone to over fitting, does not work well unless all the variables are identified.
5	Descion Tree	Used on regression and classification problem	suitable for regression Classification problem, easy to interpret and handle, capability to fill missing values, high performance due to efficiency of tree traversal.	unstable, difficult to control size of tree, it may be prone to sampling error and it gives a locally optimal solution- not optimal solution. Prone to Over-fitting

S. no.	***Machine Learning Algorithms***			
	Name of Algorithm	***Uses***	***Advantages***	***Drawbacks***
6	Support Vector Machine	Used on regression and classification problem	Handles both semi structured and structured data, can handle complex function, less probability of over fitting, scales up the high dimensional data. does not get stuck in local optima.	Low performance in large data sets, difficult to find appropriate kernel function, Does not work in noisy dataset. No probability estimates. Difficult to understand
7	Bayesian Learning	To handle incomplete data sets.	Prevents over-fitting, no removal of contradictions required.	Prior selection is not easy. distribution can be influenced by prior, wrong predictions possible, complex computation
8.	Naïve Bayes	Used on binary and multi-class classification problems	easy to implement, gives good performance , less training data required, scales linearly with predictors and data samples, handles continuous, discrete data. insensitive to irrelevant features	Model often outperforms, too simple, cannot be applied directly, requires retraining, stops scaling when data points are high, more runtime memory required, complex computation for more variables.
9.	K Nearest Neighbour	Used on classification problems	Simple and easy to implement, cheap and flexible classification, suitable for multi modal classes	Expensive, computation is distant and intense, less accuracy, no generalization, data large sets

2.5 DRAWBACKS OF MACHINE LEARNING

Despite Machine learning being very effective and offering a technical advancement in various domains, it has numerous drawbacks related to a specific problem or a particular machine learning algorithm. The drawbacks of ML are mentioned as following.

(i) **Volume of Data** : When the training and learning process is carried out, a large volume of data is required and used. The data used in this process should be non-partisan and unbiased consistency and high quality which might need the generation of more data and hence, more time, space and power is required for better quality of results.

(ii) **Authentic and Dependable Resources**: are required in case learning algorithms show time consuming errors and complexity. It is very important to check that the algorithms that has been assisted in the process is producing the desired output or not because to get the desired output we need an accurate learning algorithm with high performance.

(iii) **Selection and availability of accurate algorithm** : is also a challenge. Machine learning still needs a lot of improvements in algorithms and the software that performs the analysis on datasets.

(iv) Moreover, due to large volume of data, **Error susceptibility** is also high which needs to be taken care of while using a particular dataset and learning algorithms.

(v) **Drawbacks related to a specific machine learning algorithm** such as non-linearity, sampling errors, overfitting, noisy datasets, incomprehensible datasets, low performance, complex and expensive computation, insufficient runtime memory etc.

CHAPTER 3

LITERATURE SURVEY - MACHINE LEARNING ALGORITHMS

The power dataset of RISC Processor is used in the implementation of five different Machine Learning Algorithms to find the Accuracy, precision and Recall. These Five algorithms are studied and summarized as follows.

3.1 SUPPORT VECTOR MACHINE

For the statistical analysis of classification based power dataset, an appropriate ML algorithm can be Support Vector Machine (SVM) [9]. It is applicable to both binary and multi class dataset. To define a Binary classification problem using SVM, a hyper-plane is constructed and optimized. The optimization of this hyper-plane involves the maximization of margin separating the two different classes [9]. SVM dealing with multiple classes combines many binary classifiers. Figure 5 shows the two types of classification where support vector machine is applicable.

Let us assume that a set of n number of data vectors are available which can be written in a form (Xi,Yi) where i=1,2,…,n given that Yi belongs to the range between -1 to 1 and Xi belongs to R^N. Here, X denotes the data vector, Y denotes the binary class and i denote the i^{th} number of data vector or binary class.

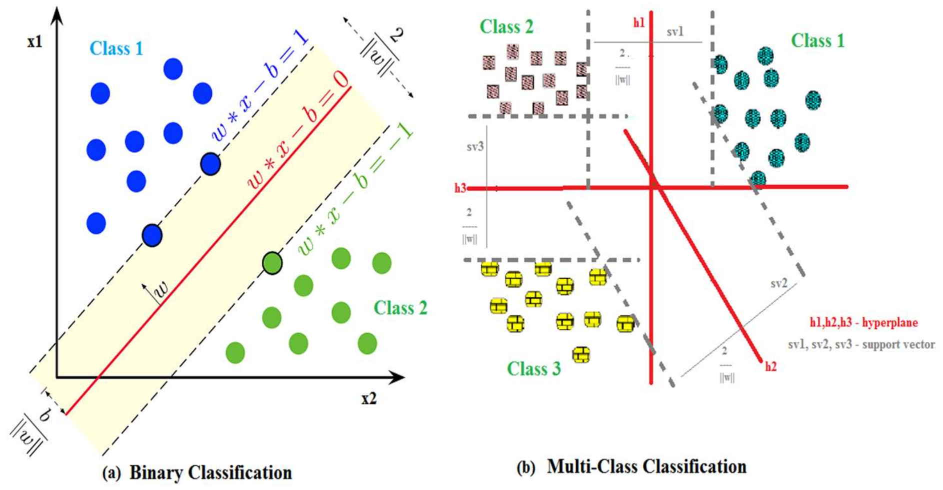


Figure 3.1 Types of SVM Classification

This type of SVM classification aims at constructing the following optimization function for the hyper plane that accurately distinguishes the difference between the class label/tag of input data sample to be tested X[10].

$$Y(X) = sgn[w. \varphi(X) + b] \quad (1)$$

such that $\varphi(X)$ is a mapped function of input data X. It is nonlinear in nature and maps to high dimensional characteristics space of available data. The difference between two support vectors is $2/\|w\|$.We obtain the optimization by maximizing the difference between the support vectors.

The general separating hyper-plane equation where input data is lying on the hyper plane itself, is written as $w\cdot\varphi(X) + b = 0$. When two classes are linearly separable, the expression for two separating hyper plane are written as w.Xi + b is greater than or equal to +1 given that Yi is +1 and w .Xi + b is less than or equal to −1 given that Yi is −1[11]. On combining both the conditions, we get the following expression.

$$Yi(w \cdot Xi + b) - 1 \geq 0 \quad (2)$$

and its optimization is obtained by $\min\{(1/2)\|w\|^2\}$. But, when the classes cannot be separated by linear expression, we need to modify the hyper plane expression as well as its optimization by using slack variable which helps in penalizing the non-linearity problem in separating the two classes. The modified hyper plane expression is given as

$$Y(w \cdot Xi + b) > 1 - \xi i. \quad (3)$$

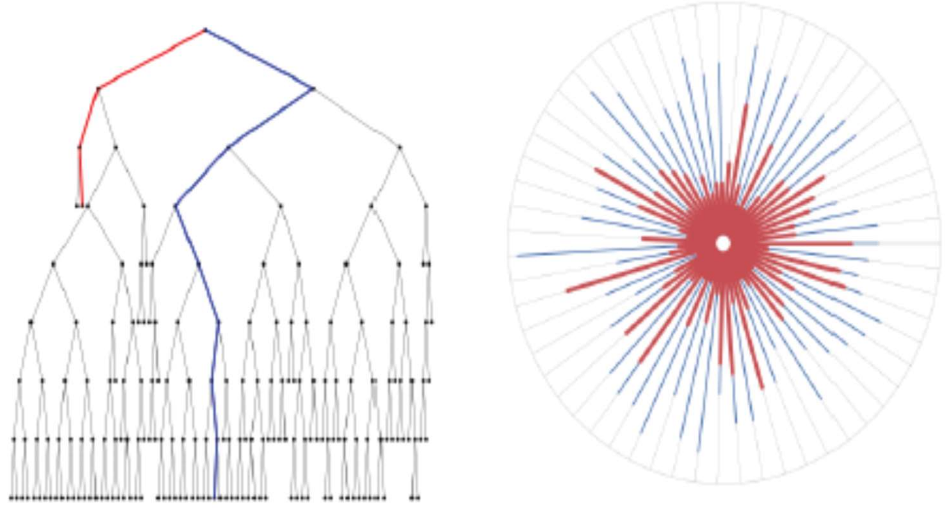
The optimization obtained from modified expression is given as : $\min[\{(1/2)\|w\|^2\}+P\sum \xi_i]$, where $P\sum \xi_i$ is the penalty term [11] and ξ_i is the slack variable in which value of i=1,2,…,n. these parameters are selected by the user depending on their requirements. The more the value of C, the higher is the penalty for wrongly classifying the data [10].

3.2 ISOLATION FOREST (IF) ALGORITHM

The Isolation Forest (IF) algorithm is applied when at least one class is an anomaly i.e. one class is very small in proportion to the other class in terms of numbers of samples or size and there is notable difference between the two classes[12]. While constructing a random tree, the instances or attributes of the entire dataset is partitioned in a repetitive manner until all the attributes/instances are completely separated from each other. In such a case when tree is generated by partitioning, the anomalies will have distinguishable shortest path among the entire dataset because anomalies will be detected in smaller number of partitions only. So, they are the fastest one to be detected when a large number of random trees are produced together, also called Random Forest Trees [14].

Consider an initial node T, which is called to be a parent node or root node in an Isolation Tree. The node T can be an exterior-node possessing no sub nodes or an interior-node having performed 1 test and 2 sub nodes, say (T_l ,T_r). An assessment is carried where the attribute value of an instance (m) and attribute value of a randomly chosen instance often called split value(s) are compared. Depending on the condition- $m < s$, it will segregate the data into two data points' i.e T_l and T_r. This, building up of isolation tree is performed with repetitive segregation of entire data sample X by randomly choosing an instance and respective split value considering a data sample X where X has n number of instances/ attribute. The process is repeated till the tree attains a certain height or the data sample left = 1. An isolation tree can simply be called a binary tree in which every node has either zero or two sub nodes. If all instance/attributes are mutually exclusive and separable, then instance/attribute are isolated as root node and the total count of root node is equal to the total number of instances in the dataset X i.e. n_r and $n_r - 1$ is the no. of sub nodes. $2n_r - 1$ is the total number of nodes of an isolation Tree. Memory required for isolation forest algorithm grows in a linear fashion with n_r.

[14].Path Length of isolation tree is the number of branches covered while moving from root/main node of the tree to the last terminating node of the tree[12].



(a) Representation of a single tree in a forest.

(b) Representation of a full forest where each radial line corresponds to a tree.

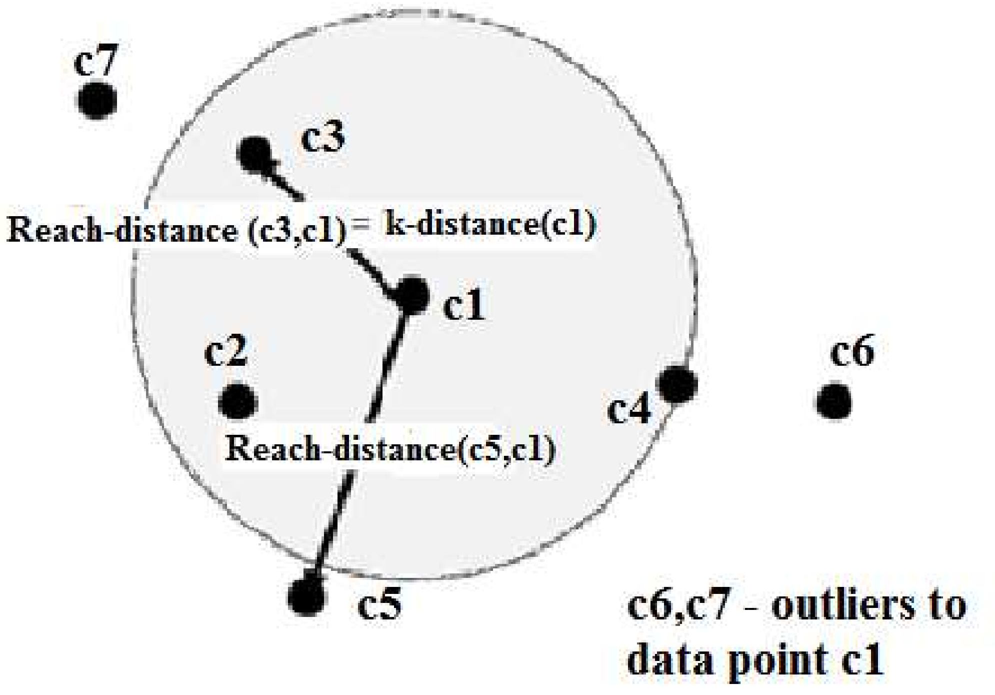

c7
c3
Reach-distance (c3,c1)= k-distance(c1)
c1
c2
c6
c4
Reach-distance(c5,c1)
c5
c6,c7 - outliers to data point c1

Figure 3.2 (i) Representation of a tree in a forest and as a radial line [13], (ii) Working of LOF algorithms [16].

3.3 LOCAL OUTLIER FACTOR (LOF) ALGORITHM

Local Outlier Factor (LOF) algorithm is used to demarcate the outlier samples from the usual or ordinary samples in the dataset which is done with the help of comparison of dataset points. To ascertain whether a given sample point is outliers or not, we calculate the deviation of data points from the other and the data point with less density is labeled as outlier. Normal data is given the output as '1' and outlier as '0' [15]. For example- noise detection in pixels, fraud detection in ATM transaction [17]. The distance between the data points are measured by term k-distance (k-dis(c5))of a sample point c5 which is stated as the Euclidean distance between data points c5 & c1 i.e. d(c1.c5) such that at least k data points have d(c1,ci) less than d(c1,c5) where i ={2,3,4,..n} and k should be greater than one[18].

K-distance neighborhood of a sample/data point say c1 is the k-distance neighborhood of c1 consists of every data point in the sample space which are less than k-distance (d (c1, c5)) distant apart from c1 as represented all the data points in a circle in figure 3.2(ii). k reach-distance (c1, c5). Local Outlier Factor (LOF) of a sample/data point say c1. is can be defined as the mean of the ratio of c1's local reachability density and data points in k nearest neighbors e.g. c2, c3, c4, where local reachability density of c1 is{mean(k-reach-distance of data points)}$^{-1}$ which are in the k-distance neighborhood of the data point c1 i.e. c2, c3, c4 etc. to the data point c1 itself and k-reach-distance (c1,c5) is defined as real distance from c5 to c1[18].

3.4 K-NEAREST NEIGHBOR(KNN) CLASSIFIER

The most commonly applied machine learning algorithm is K-Nearest-Neighbor classifier for the classification based problems. This algorithm is very simple, efficient and has high accuracy [19]. In KNN, the classification is done on the basis Euclidean distance. The data points close to the Kth nearest neighbor are given the label of Kth neighbors group. The only disadvantage of KNN classifier is that when the dataset volume is huge, it consumes more time to find out the K-nearest neighbors [20][21].

The working of K-Nearest Neighbor classifier is elucidated in accordance with the following steps: first step is to pick minimum number of neighbors say K. Second step is to determine the Euclidean distance of those K no. of neighbor data points**.** Third step is to consider only the K nearest neighbors. Next step is that out of these K neighbors, segregate the data points categorizing them according to proximity and for each new category, set new data points considering the fact of keeping neighbor as maximum as possible. Last step makes the KNN classifier ready for a use as a model. While choosing the value of K, following points should be considered: If value of K is very small, the model becomes sensitive to outliers and If value of K is large, classifier may include in the neighborhood, those data points which are part of other categories. KNN Classifier can be improved on accuracy by adding weights to the sample features and adjusting the value of K[21].

3.5 NAÏVE BAYES(NB) ALGORITHM

NB is a plain classification machine learning algorithm which uses mathematical Bayes theorem [23] and has higher accuracy [22]. Naïve Bayes theorem is of mainly three types. i)Multinomial ii)Gaussian iii)Bernoulli[24].

It detects and predicts the internet traffic while assuming that that the sample features/attributes are mutually exclusive to each other i.e. independent of one another [26]. Every machine learning algorithm has two main stages i.e. training the model and testing/predicting the unknown sample using the trained model. This trained model here works on Naïve Bayes algorithm [25]. In the initial training process, data set comprising of input features/attributes and selected output is fed to the model implementing NB classifier algorithm. From this training data, NB Classifier calculates the independent probabilities e.g. P(A), P(B) of the data whose output class has to be predicted i.e. target data. In the next step, dependent probabilities of input attributes of data set is calculated with the help of already calculated target data probabilities e.g.(A|B) or P(B|A) etc. Next is the testing phase where data to be tested i.e. test data's new input attributes are fed to the NB classifier model. Finally, the probability of output class of test data given that probability of input attribute of test data is already known; P(A|B), is calculated. The expression for the posteriori probability is P(A|B) given as

$$P(A|B) = [P(B|A)*P(A)]/P(B) \quad (4)$$

Such that P(B|A) is the likelihood probability; P(A) and P(B) are the independent probabilities in the expression 4[24].

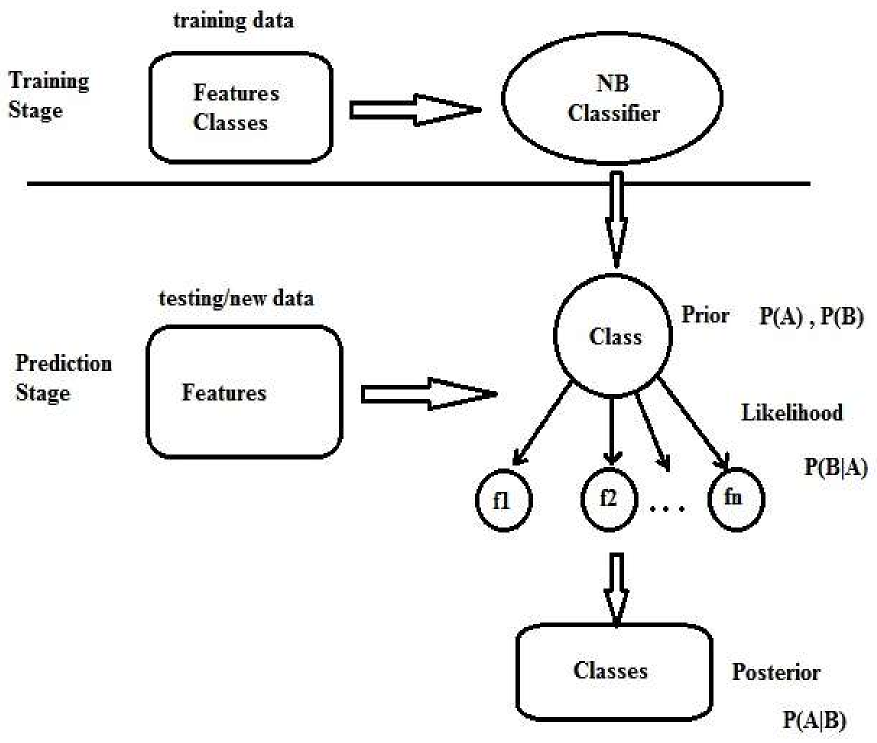


Figure 3.3 Algorithm representation for Naïve Bayes classifier[24]

Naive Bayes(NB) Classifier have good speed and performance since they take lesser starting up time and are accurate in producing results.

Performance Metrics

Every machine learning algorithm is evaluated on the basis of some parameters also called performance metrics that measures how a machine learning algorithm is performing on particular problem or dataset. These parameters are named as Precision, Recall, F1-Score and support. Let us briefly understand their meaning.

With respect to the power dataset we have used, power consumption more than 3000W is beyond tolerable limits and it is unacceptable.

We have divided the power dataset as class ‘0’ when power is under tolerable limit and class ‘1’ when beyond the tolerable limit. When data is predicted as class’0’ when it was actually class ‘0’, it is called True Positive (TP). When data is predicted as class’1’ when it was actually class ‘0’, it is called False Negative(FN). When data is predicted as class’0’ when it was actually class ‘1’, it is called False Positive(FP). When data is predicted as class’1’ when it was actually class ‘1’, it is called True Negative(TN).

i) **Precision (P)**: It is the ratio of True Positive to the sum of True Positive and False Negative.

ii) **Recall (R):** It is the ratio of True Negative to the sum of True Negative and False Positive.

iii) **Support (S):** Number of samples in Class ‘0’ i.e. Total number of power data samples which are under tolerable limit in Target.

iv) **F1-Score:** It is the harmonic mean of the precision and recall. i.e. {2*(P*R)}(P+R)

CHAPTER 4
TOOLS USED

4.1 INTRODUCTION TO VERILOG HDL

Verilog HDL is widely used as design description language for field programmable gate arrays and application specific integrated circuits in the automation of electronic design circuits. Verilog HDL is a generic language which needs a simulator to run the design code. Due to its generic nature, it is quite possible to code a test bench which verifies and checks whether the design code written is working correctly or not.

4.2 FEATURES OF VERILOG

Following are the major features and capabilities that Verilog provides which makes itself different from other hardware languages.

- Exchange medium between chip vendors and CAD Tool users.
- Open Source, comprehensible, readable to machine.
- Flexible design methodologies: top to down, bottom to up, mixed.
- Supports hierarchy.
- Not technology specific, but supports the technology related features.
- Support various hardware technologies.
- Synchronous and asynchronous timing models supported.
- An IEEE and ANSI standard; hence models are portable.
- Three different description styles: structural, behavioral, data flow supported.
- Test benches allowed to be written to test other Verilog models.
- A wide range of abstraction level is supported.
- It does not, support modeling at or below the transistor level.
- Propagation delays, min max delays, setup and hold timing, timing constraints, spike detection can be described.

There are several abstraction levels in which a digital circuit can be designed which are shown in Fig.4.1. The lowest level from where we start designing a digital circuit is the transistor level; it deals with the discrete transistors. The transistors are connected with each other to make a circuit. Gate level is higher level in the abstraction where we combine logic gates to form the circuit. With the use of logic gates, standard combinational as well as sequential circuits can be made.

These combinational and sequential circuits are put together to form the larger circuits such as processor. It becomes an easier to solve a digital problem when we solve it from lower level to higher abstraction level. The RTL consists of data path and control circuit along with their connections that makes the data flow within these functional units. Behavioral level is the highest level of abstraction which describes the operation of the larger circuits.

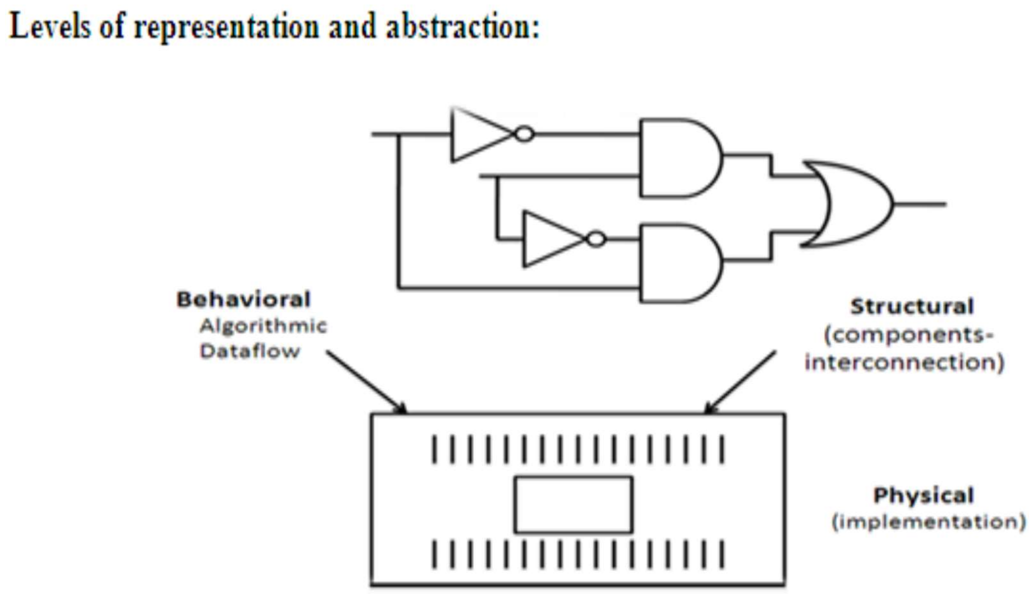


Figure 4.1: Level of Representation and Abstraction [28]

4.3 SIMULATION

The stimulation is done with the help of test benches. They check and tell whether the output generated is correct or not. The block representation of stimulation is shown in Figure 4.2.

4.4 ADVANTAGES OF VERILOG

The main advantages of Verilog can be listed as following:

- Description of a concurrent system which is not possible in other programming languages like Pascal, BASIC, C, low level assembly language.
- Translates the gates and wires of CPLD or FPGA which no other language can do.
- It is just like an actual hardware being configured.

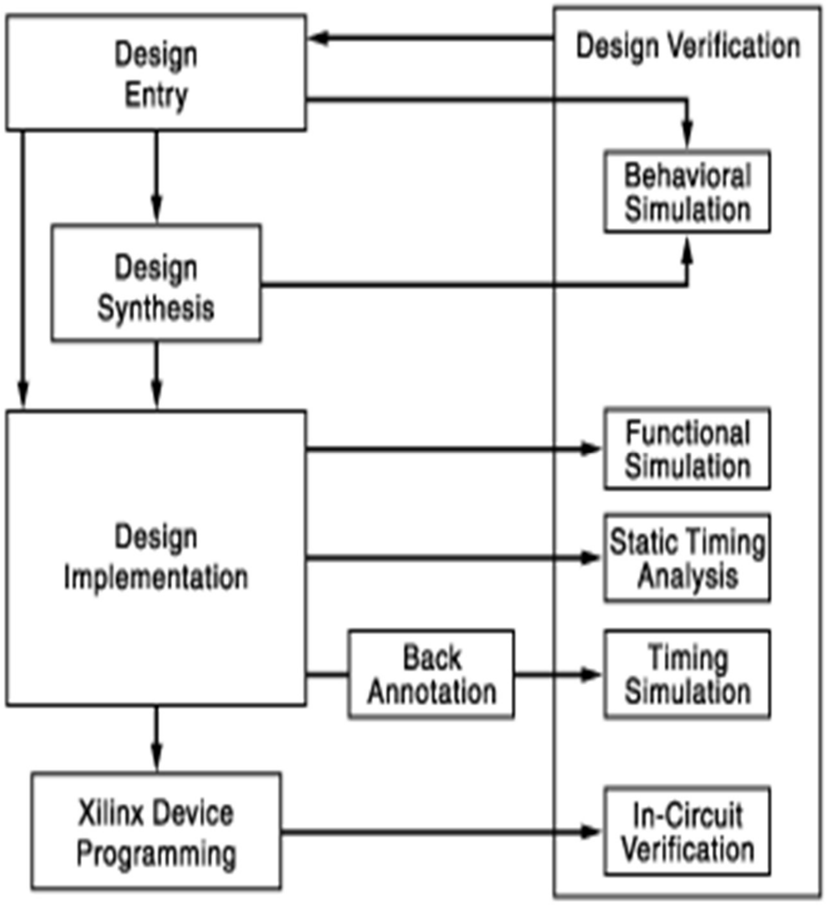


Figure 4.2: FPGA Design Flow Overview[27]

4.5 XILINX VIVADO

4.5.1 INTRODUCTION TO XILINX VIVADO SOFTWARE

XILINX VIVADO is a software tool mainly used for the synthesis, simulation, analysis and implementation of digital circuit design. It helps the designers to simulate and synthesize the digital designs, performing the analysis on timing constraint and examining the RTL schematics. It also helps simulating a design's response to various stimulus, and also configure the targeted device. It supersedes the Xilinx ISE with added feature SoC development and High Level synthesis. Vivado came up in April 2012[29] and is an integrated design environment with system to IC level tools which is built on a shared data paradigm and a common debug environment. Vivado incorporates ESL design tools for synthesis and verification of C based IP; standard packaging of RTL IP for reuse; standard IP integration of all kinds of system blocks; and their verification[30].

- **Vivado HLS** helps C, C++ and SystemC programs to be directly targeted into Xilinx devices without the need to manually create RTL. Vivado HLS increases designer's productivity, and supports C++ classes, templates, functions and operators[31].
- **Vivado Simulator** bolsters mixed language like Tcl scripts, IP and its verification.
- The **Vivado IP Integrator** combines and configure IP from the large Xilinx IP library.
- **Vivado Tcl Store** develops add on to Vivado, and used to add and change Vivado's capabilities. Vivado is based on Tcl scripting language. Internal functions can be called and controlled using Tcl scripts[32].

CHAPTER 5

SIMULATION RESULTS

5.1 EXRIMENTAL SETUP

The RTL code for the RISC Processor is written in Verilog Hardware Description Language and the software used for its behavioral simulation, RTL analysis, RTL synthesis and schematic generation and design implementation is XILINX VIVADO 2019.1 version.

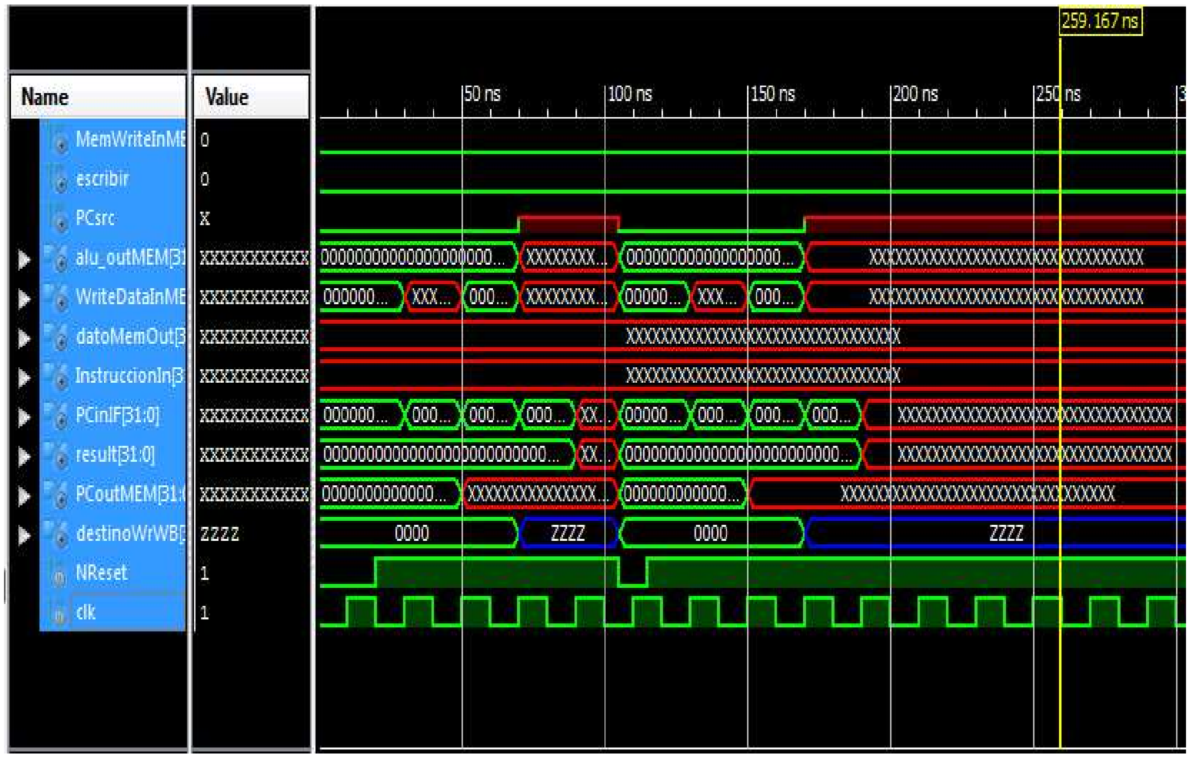


Figure 5.1 Simulation Result of 32 bit RISC Processor

Figure 5.1 shows that 32 bit RISC processor so designed is working correctly and the simulation results are generated. It is just to show that this RTL code is the basic model of RISC Processor and its simulation results showing that the RTL code is bug/error free. On the analysis of RTL code, the elaborated design of the RTL schematic is shown as in the figure 5.2. The RTL code synthesizes to a circuit as shown in the figure depending upon constraint files and design code files depicting number of logic cells and their interconnections when implemented on a FPGA.

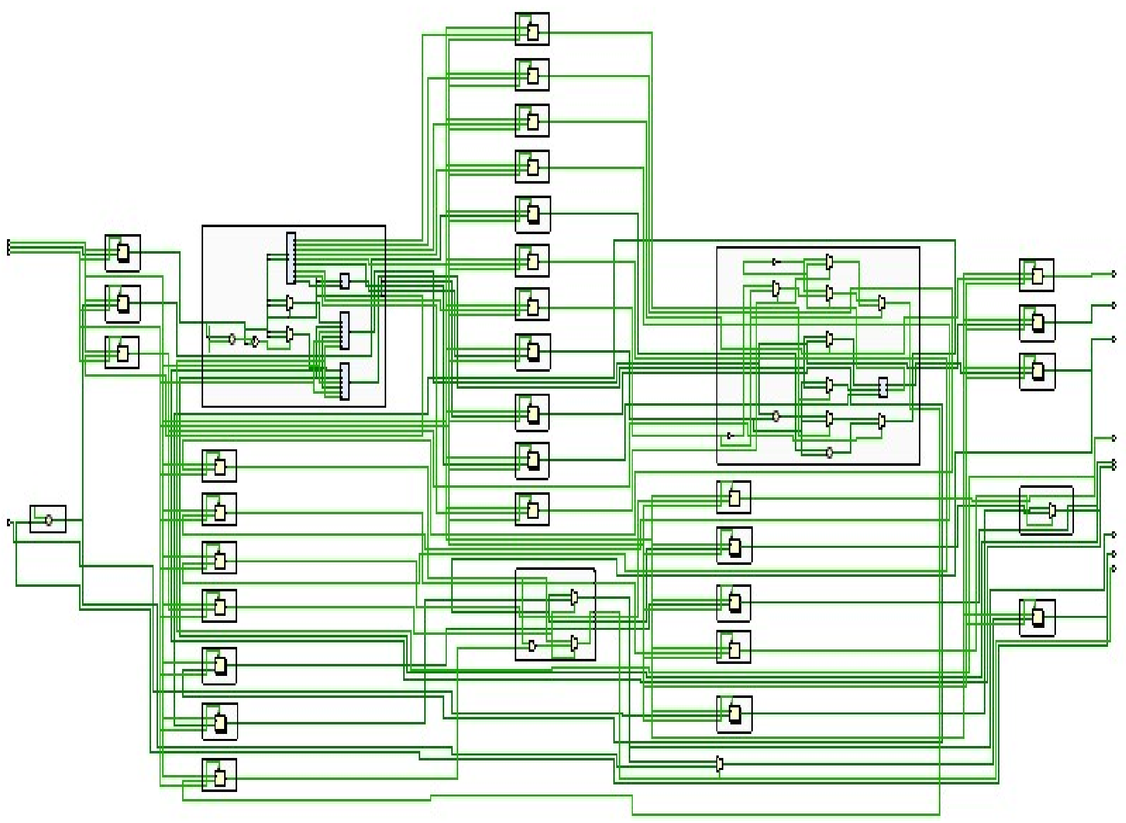
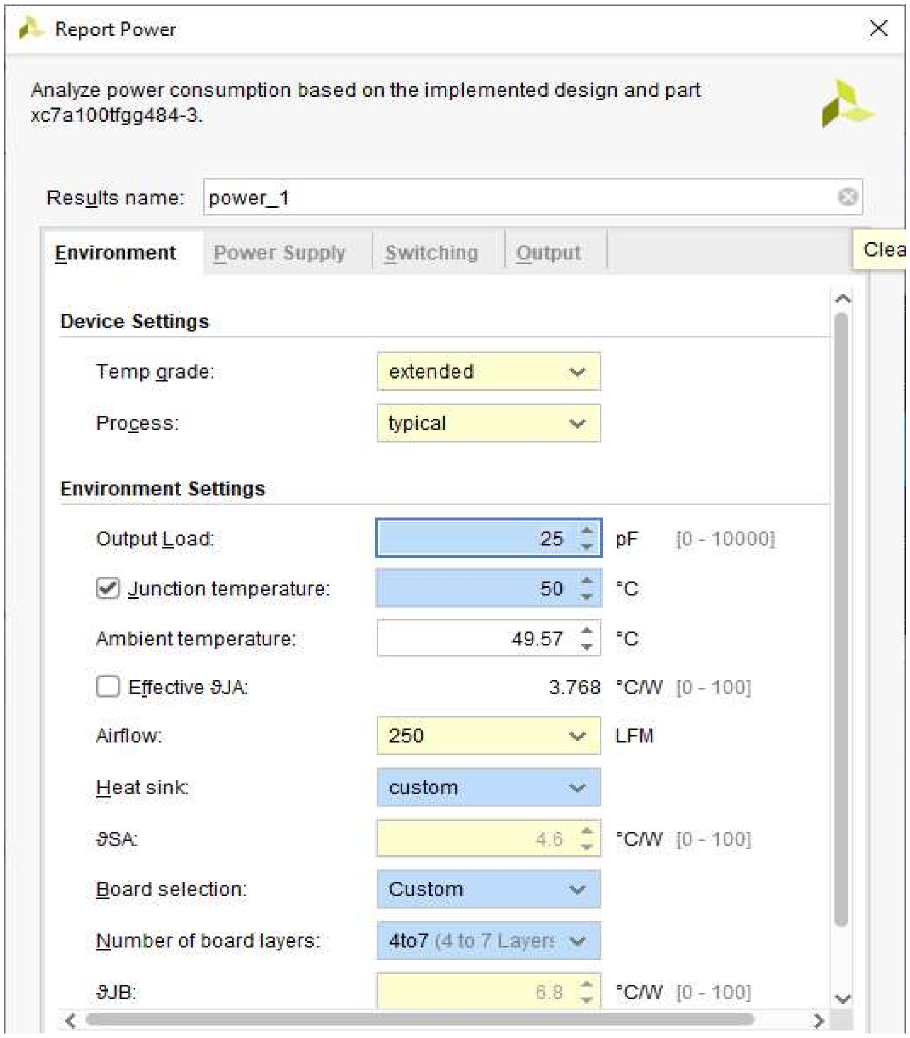


Figure 5.2 Register Transfer Level schematic of RISC Processor after implementation stage.

After the design and simulation of 32 bit RISC Processor, its power reports has been generated giving variety of inputs such as Output Load, Junction Temperature, Ambient Temperature,Effective SJA, Airflow, Heat Sink, Theata, Board Selection, No. of board layers, Board Temperature , Thermal Margin etc. The output report consisted of Logic power, On chip dynamic power, Static power and I/O power, Total on chip power, signal power, class etc. All the inputs and outputs given here are the features/attributes of a data sample. Approximately 50 data samples has been recorded and maintained in .csv file.


Report Power

Analyze power consumption based on the implemented design and part xc7a100tfgg484-3.

Results name: power_1

Environment | Power Supply | Switching | Output

Device Settings

Temp grade: extended

Process: typical

Environment Settings

Output Load: 25 pF [0 - 10000]

☑ Junction temperature: 50 °C

Ambient temperature: 49.57 °C

☐ Effective ϑJA: 3.768 °C/W [0 - 100]

Airflow: 250 LFM

Heat sink: custom

ϑSA: 4.6 °C/W [0 - 100]

Board selection: Custom

Number of board layers: 4to7 (4 to 7 Layers

ϑJB: 6.8 °C/W [0 - 100]

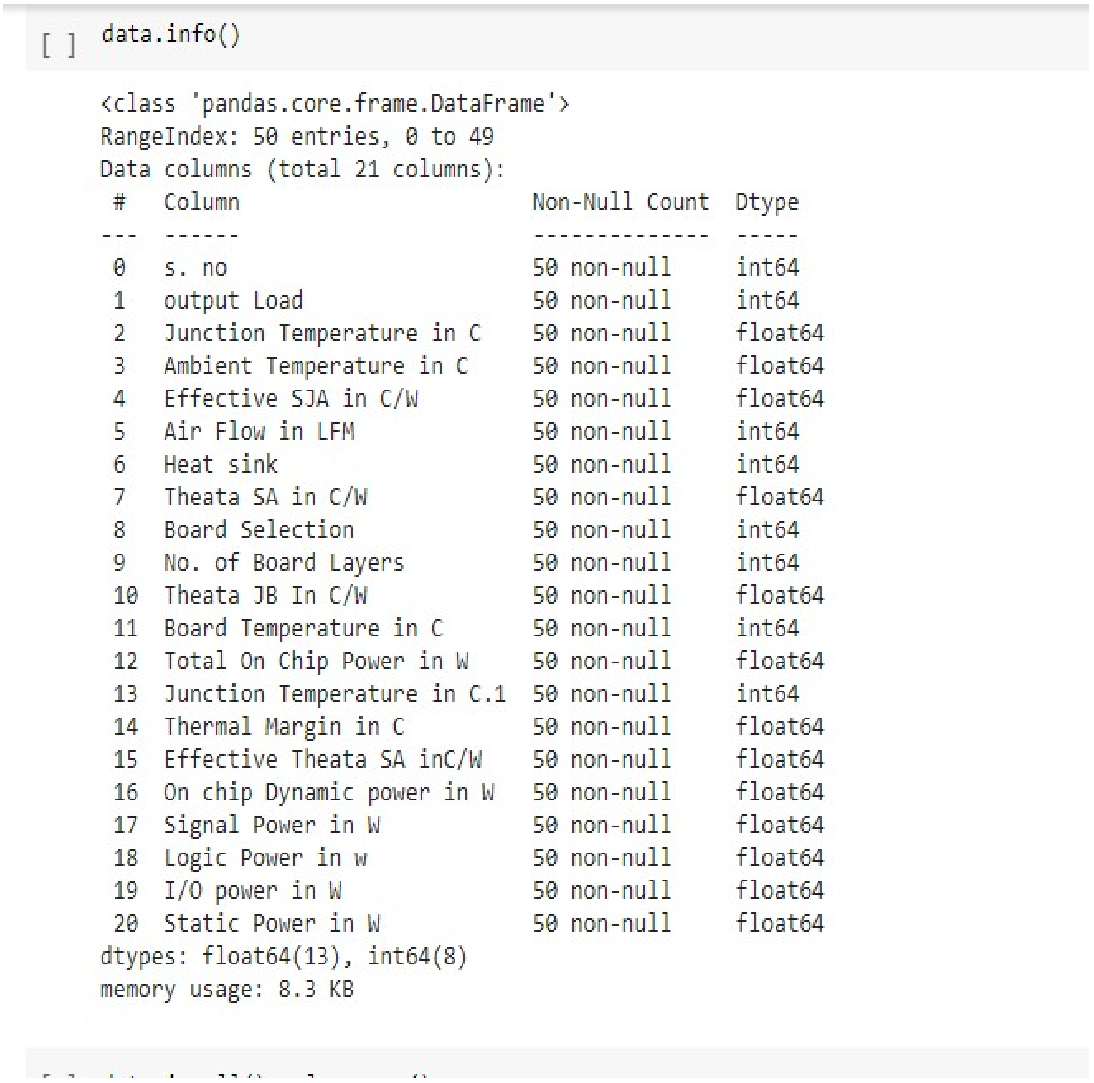
Figure 5.3 Power Report and required Input features.

These data samples are used as training sample for the analysis of power dataset and implementation of ML Algorithms to obtain their comparative performance measuring parameters such as macro average and weighted average of Precision, Recall, F1-Score and Accuracy.

Implementation of Machine Learning :

The first step is the collection of data samples. Here, the collection of different types of power consumption is the data on which the machine learning algorithm will be applied. A comma separated file is created from the different input of power samples. This data is analyzed first hence is called power analysis of RISC

Processor. Figure 5.4 represents the data information mainly the input features or attributes and their related power outcomes included in the dataset created from various power samples of RISC Processor.



```
[ ]  data.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 50 entries, 0 to 49
Data columns (total 21 columns):
 #   Column                      Non-Null Count  Dtype  
---  ------                      --------------  -----  
 0   s. no                       50 non-null     int64  
 1   output Load                 50 non-null     int64  
 2   Junction Temperature in C   50 non-null     float64
 3   Ambient Temperature in C    50 non-null     float64
 4   Effective SJA in C/W        50 non-null     float64
 5   Air Flow in LFM             50 non-null     int64  
 6   Heat sink                   50 non-null     int64  
 7   Theata SA in C/W            50 non-null     float64
 8   Board Selection             50 non-null     int64  
 9   No. of Board Layers         50 non-null     int64  
 10  Theata JB In C/W            50 non-null     float64
 11  Board Temperature in C      50 non-null     int64  
 12  Total On Chip Power in W    50 non-null     float64
 13  Junction Temperature in C.1 50 non-null     int64  
 14  Thermal Margin in C         50 non-null     float64
 15  Effective Theata SA inC/W   50 non-null     float64
 16  On chip Dynamic power in W  50 non-null     float64
 17  Signal Power in W           50 non-null     float64
 18  Logic Power in w            50 non-null     float64
 19  I/O power in W              50 non-null     float64
 20  Static Power in W           50 non-null     float64
dtypes: float64(13), int64(8)
memory usage: 8.3 KB
```

Figure 5.4 Data Information of Power samples of RISC Processor

There are five types of power dissipation calculated and recorded from VIVADO which can enumerated as static power dissipation, I/O power dissipation, Signal Power dissipation, Logic Power dissipation and On chip power dissipation. Further Figure 5.5 to Figure 5.9 all the types of power are represented in the form of bar and pie. They represent what is the power consumption in majority of the samples.

5.2 EXPERIMENTAL RESULTS

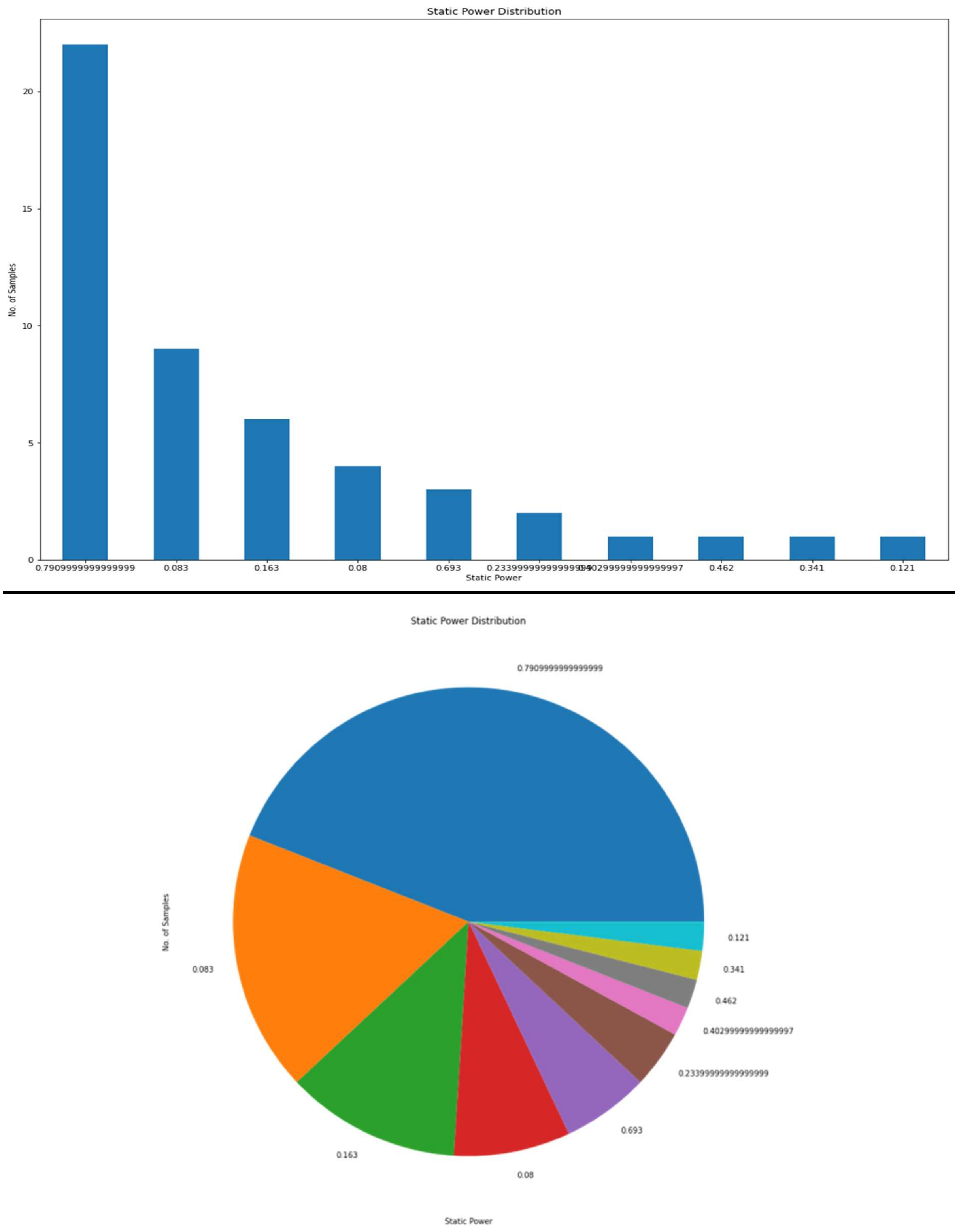
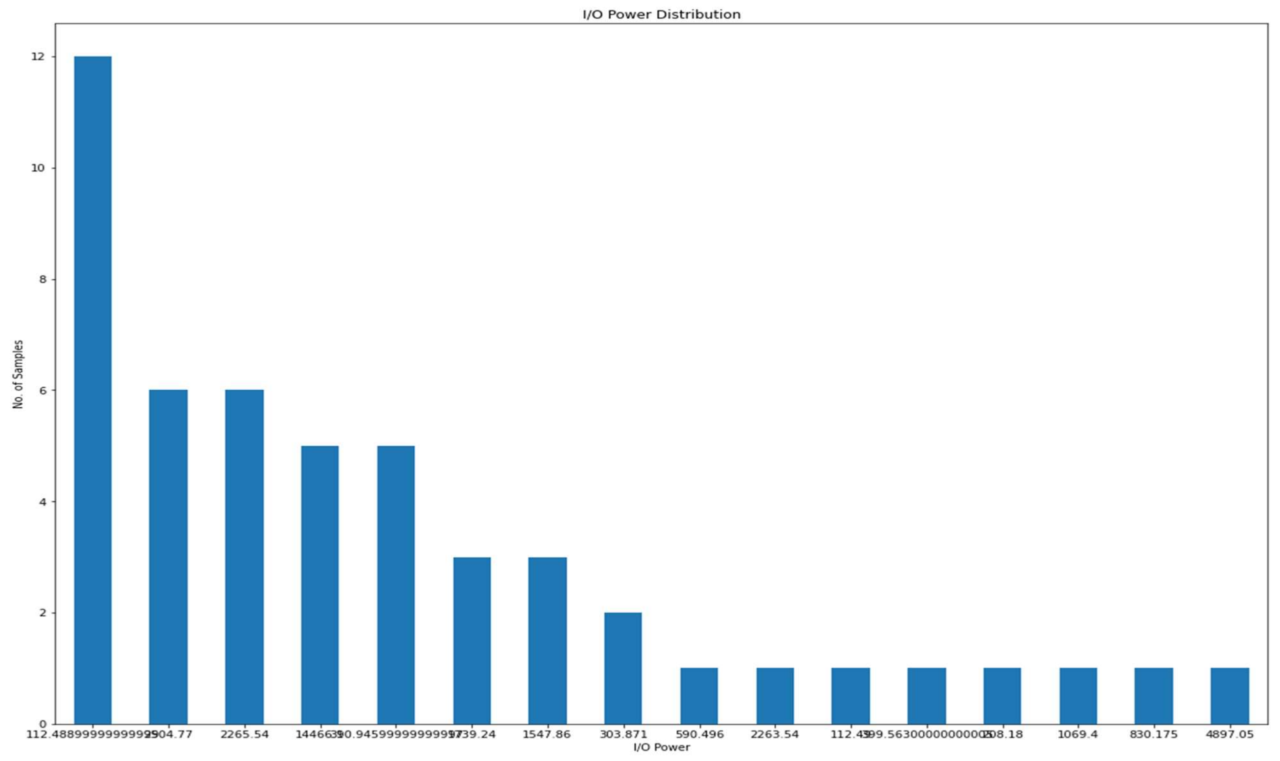


Figure 5.5 Static power distribution of RISC Processor



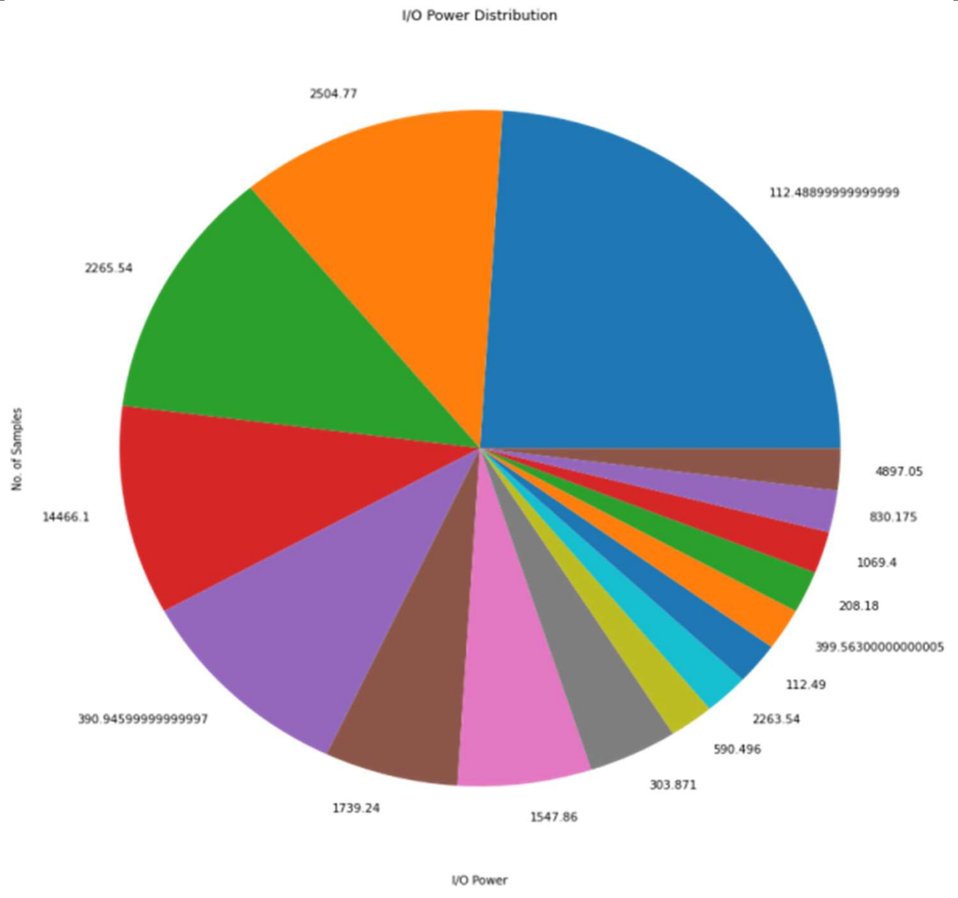
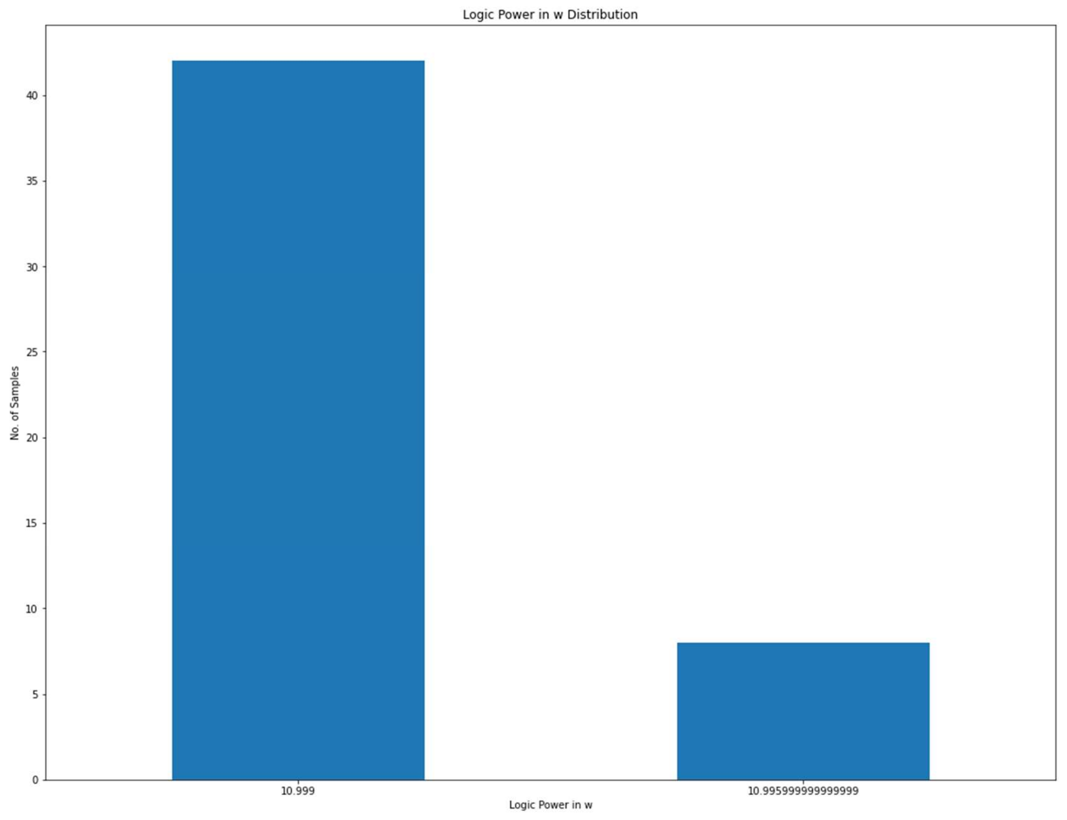


Figure 5.6 Input/Output power distribution of RISC Processor



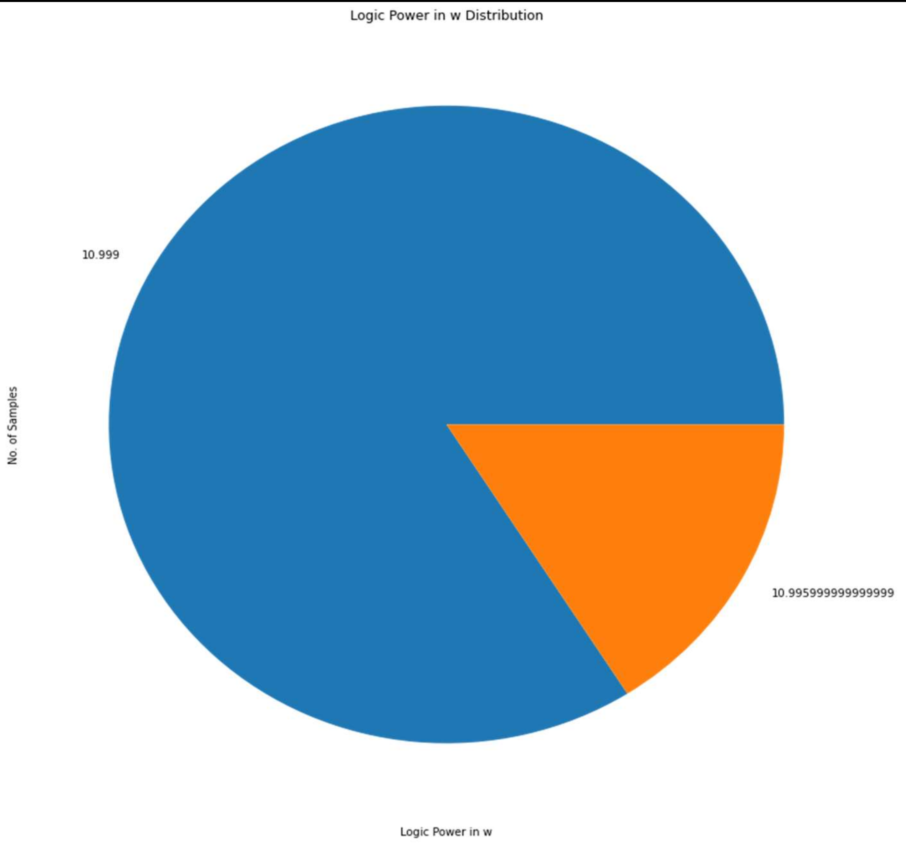
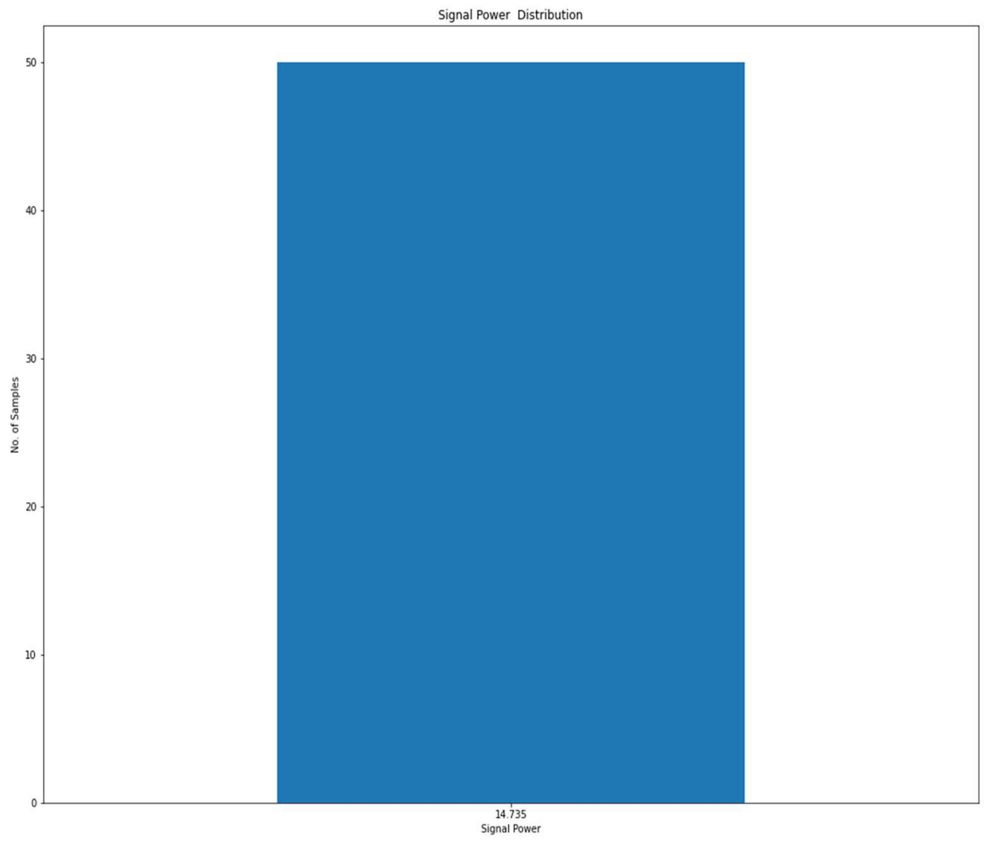


Figure 5.7 Logic power distribution of RISC Processor

The three main type of powers out total five power are discussed below.
(i) Static Power (ii) Input/Output Power (iii) Logic Power
Figure 5.5 shows the distribution of static power consumed by the RISC Processor. Static power consumption by RISC processor is the power consumed by Processor when there are no switching activities taking place. This power is mainly due to the leaky current when the processor is in OFF state. Maximum number of samples consuming static power equal to 0.790 W, rest of the samples consuming 0.043 W, 0.143W, 0.08W, 0.693W and so on. There are various input features on which static power depends. This variation in Static power consumption depends on what input was given to generate the power report of RISC Processor.

Similarly, for the Input/Output power distribution among the samples is shown in Figure 5.6. I/O power is power consumed by the inputs and output ports of the processor while transitioning the signal from high to low or low to high. Maximum number of samples consuming 112.489W and rest of the samples consuming 2504.77W, 2265.54W, 14466.1W, 390.95W, 1739.24W and so on. Maximum I/O power reaches 14466.1W under some input conditions.

Figure 5.7 shows Logic Power distribution of the RISC Processor. Logic power consumed by the processor is the power consumed by the logic implementation present inside the processor. Here, all the samples consume nearly same amount of power i.e 10.990W and 10.995W treated as differently but not much difference in the logic power consumption can be inferred that it does not much depend upon the variation of input features in the power report of the processor. This processor consumes same logic power i.e. it depends on processor internal implementation rather than outside application of the various inputs.



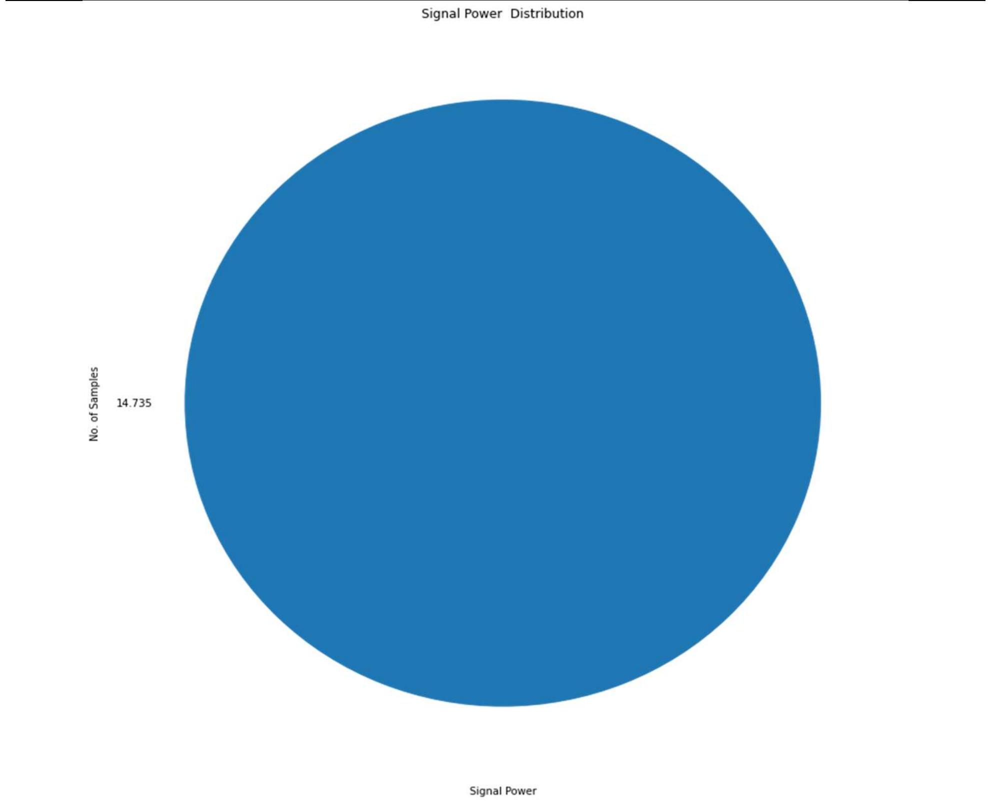
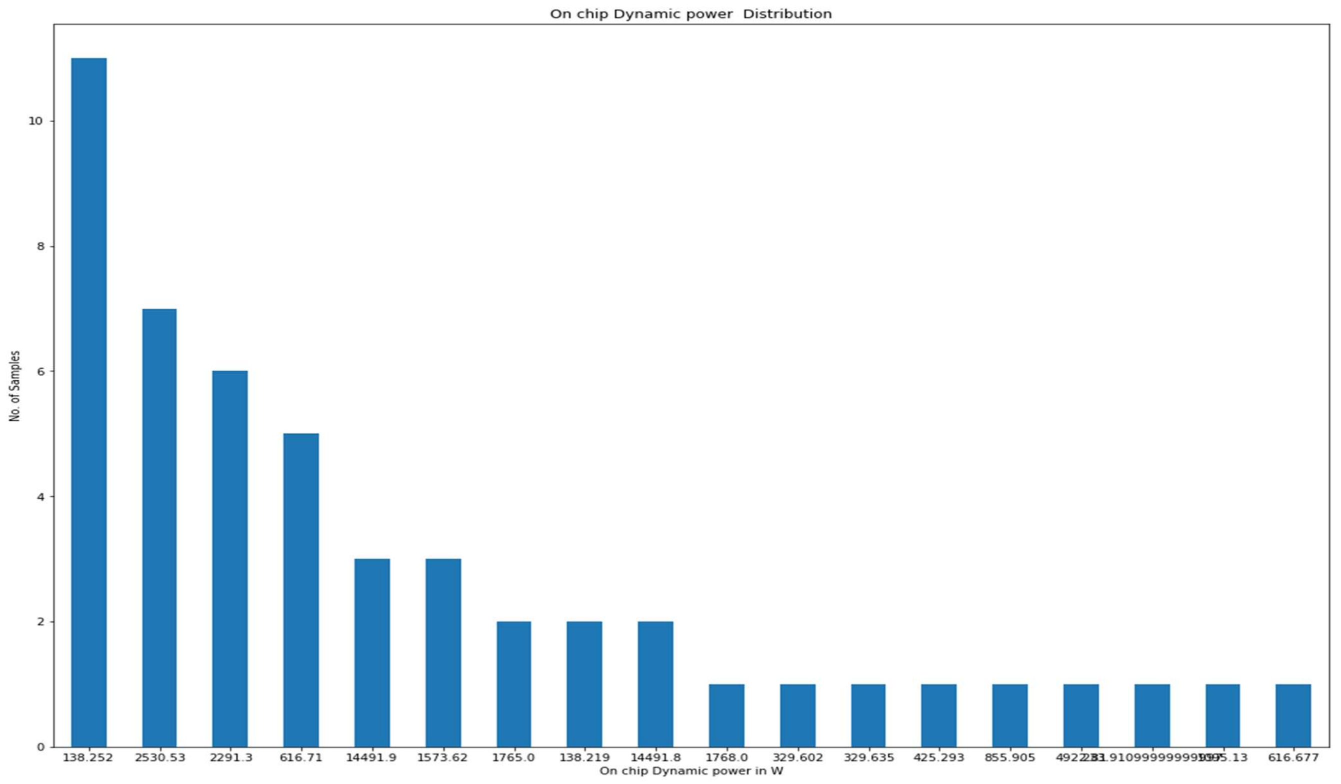


Figure 5.8 Signal power distribution of RISC Processor



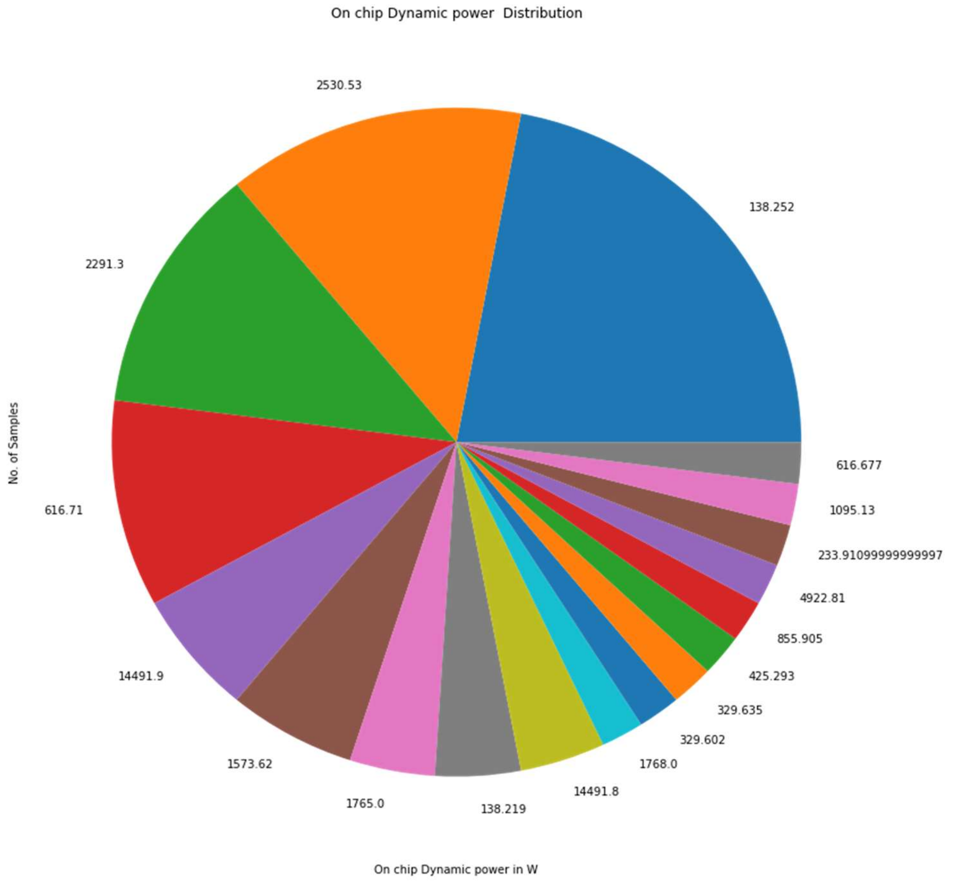


Figure 5.9 On chip power distribution of RISC Processor

Two more power out of total five powers are described below.

(iv) Signal Power (v) On chip Dynamic Power

Figure 5.8 shows the distribution of signal power of RISC processor that what amount of power is consumed by the signals in the processor. Here, all the samples consume fixed amount of signal power i.e. 14.735W which can also infer that signal power of the processor does not vary with the variation in the input features. It is fixed under all conditions.

Figure 5.9 shows On chip dynamic power distribution of RISC processor. It is the power consumed by the processor during the switching activities caused to the transistors used in the implementation of the processor circuit. Maximum numbers of samples consume 138.252W On chip dynamic power. Rest of the samples consumes 2530.53W, 2291.3W, 616.71W, 14491.9W, 1573.62W, 1765W and so on. The maximum and unacceptable on chip dynamic power consumed is 14491.9W. The variation in the on chip dynamic power consumption depends upon the variation in the values of Input features.

Factors affecting various power of the processor:

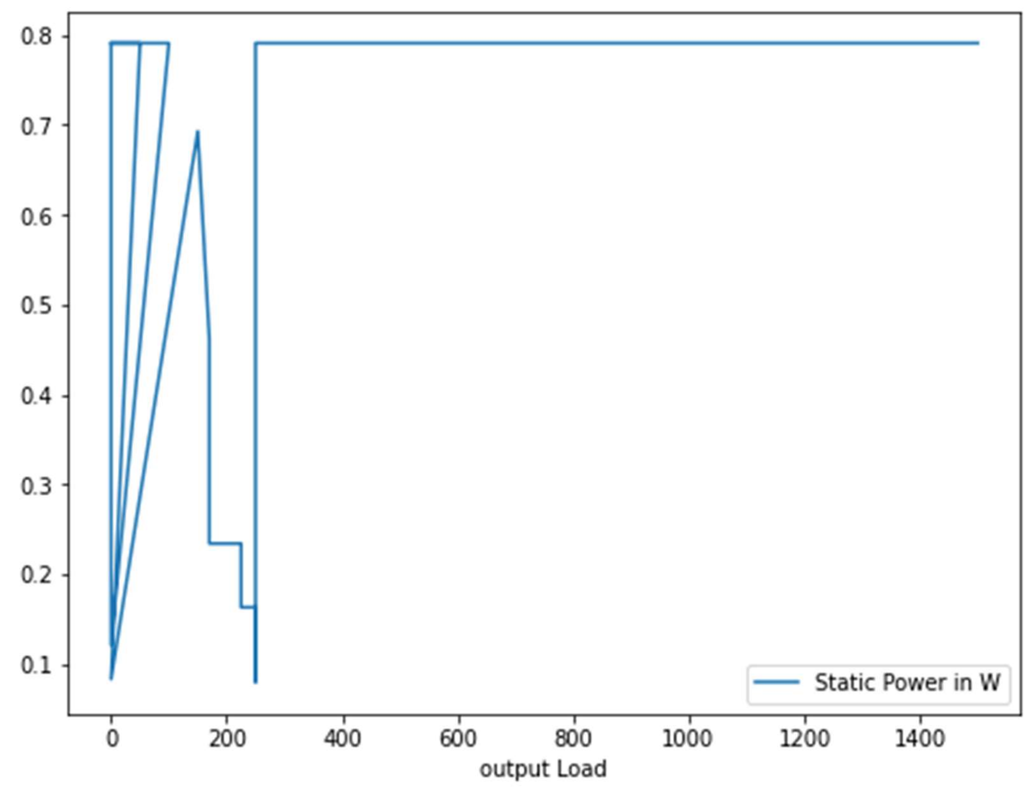
There are two factors on which power of the processor depends and have been listed as (i) Output Load Capacitance (ii) PVT conditions – Junction Temperature

(i) Output Load Capacitance (fF):

The output load capacitance is the major factor in the variation of power of RISC Processor. We have found experimentally the variation of different kinds of power with respect to the output load capacitance.

- Static Power

Figure 5.10 (i) shows the dependence of Static power on the output load capacitance of the processor. From the graph, Static power depends on various other factors when output load is less than approximately 250µF showing undefined behavior.



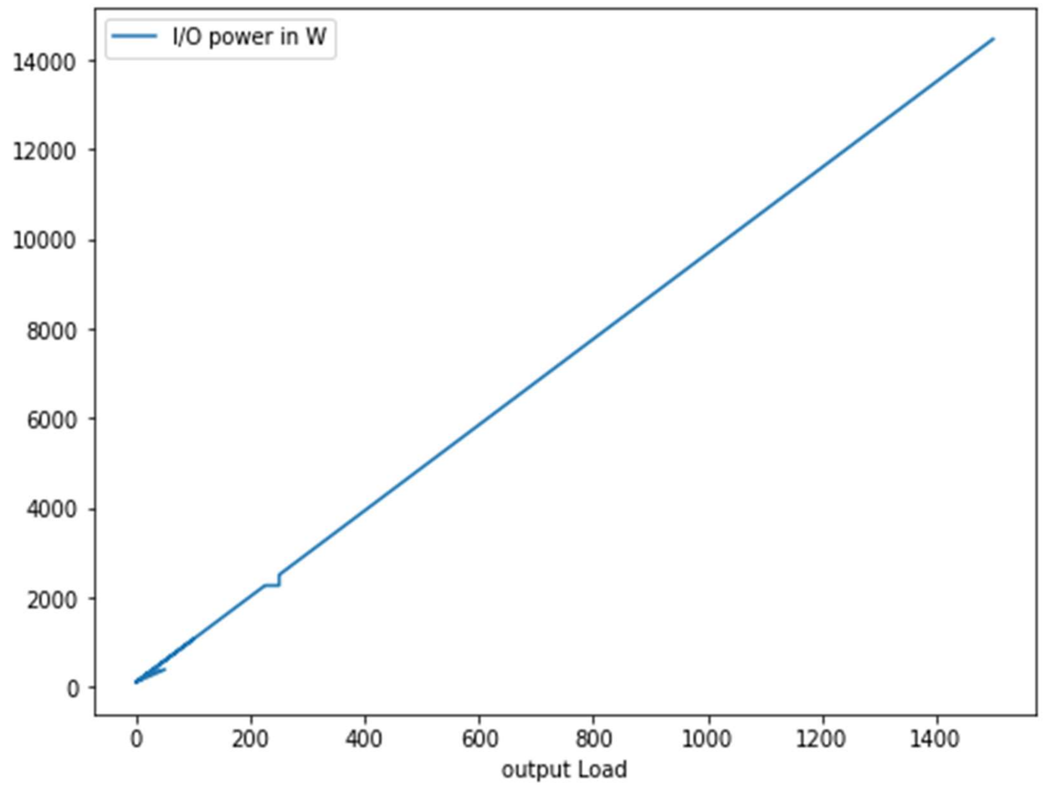


Figure 5.10 (i) Static power vs Output Load of RISC Processor **(ii)**I/O power vs output load of RISC Processor .

This is also due to the fact that some samples in the dataset may have varied other input factor but not much variation is done with output load. When output load exceeds 250μF, static power becomes constant to approximately 0.8W. If the output load increases further, there is no change in the static power dissipation. Hence it saturates after 250 μF.

- I/O Power

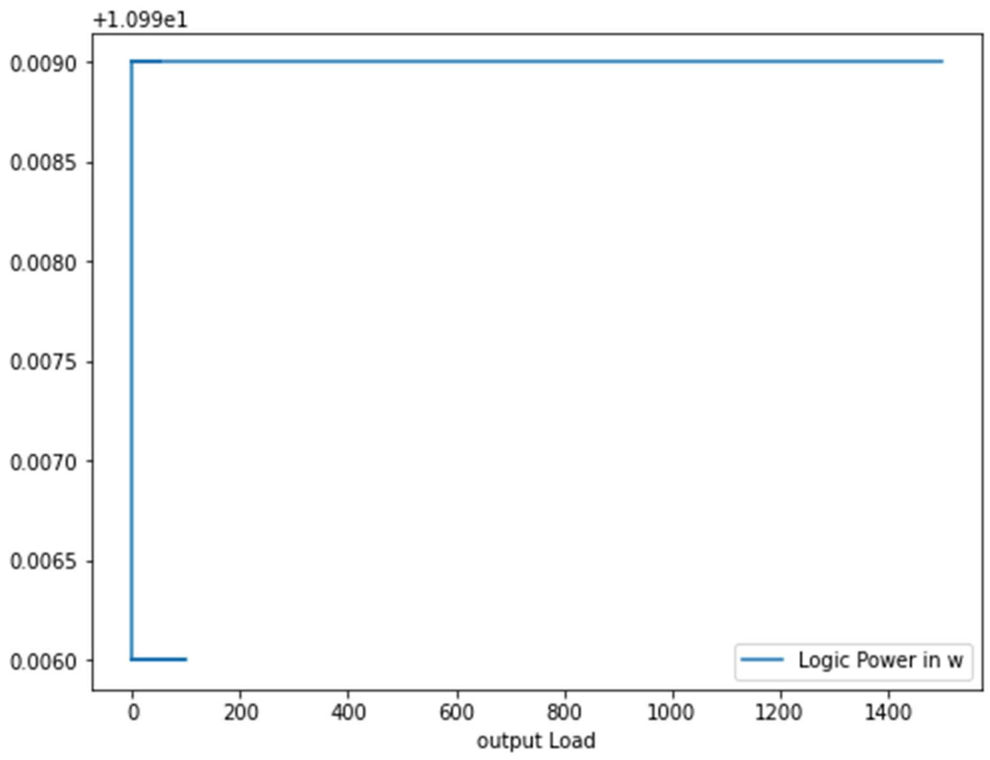
Figure 5.10 (ii) shows the dependence of I/O power on the output load capacitance of the processor. From the graph, it is clear that the I/O power of the power directly depends upon upon the output load capacitance. It varies linearly with the output load since I/O power also depends upon the switching activities and also power needed by the input and output pins of the design to function properly. From the graph, we can deduce that I/O Power α C_L .

- Logic Power

Figure 5.11 shows variation of Logic power with respect to output Load of RISC Processor. Logic power has two constant values when output Load is less than125μF i.e 0.0060W and 0.0090W. Beyond output load capacitance of 125μF, the processor consumes constant power of 0.0090W. This also infers that for low values of load capacitance, other factors also affect the power but for higher values, output load capacitance becomes the dominant factor on which power depends. Logic power does not vary with output load. It remains mostly constant.

- Signal Power

Similarly in figure 5.11 (ii) signal power variation is shown with respect to Output Load of RISC processor. Signal power is constant and equal to 14.735W and does not vary with the output load of the RISC processor.



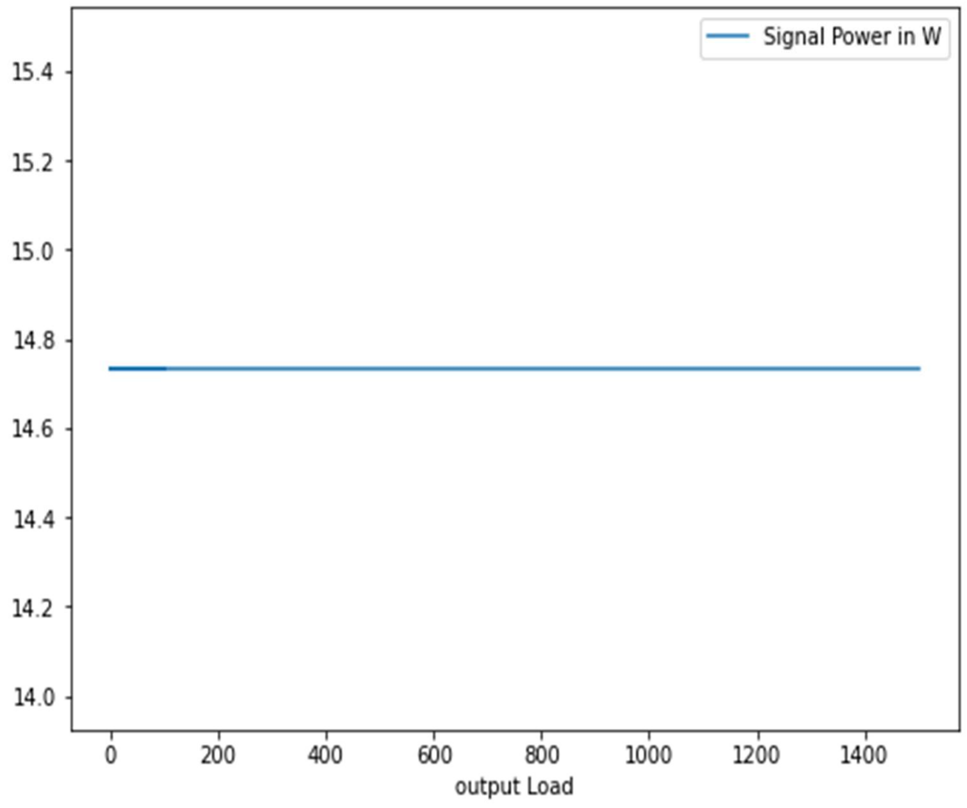
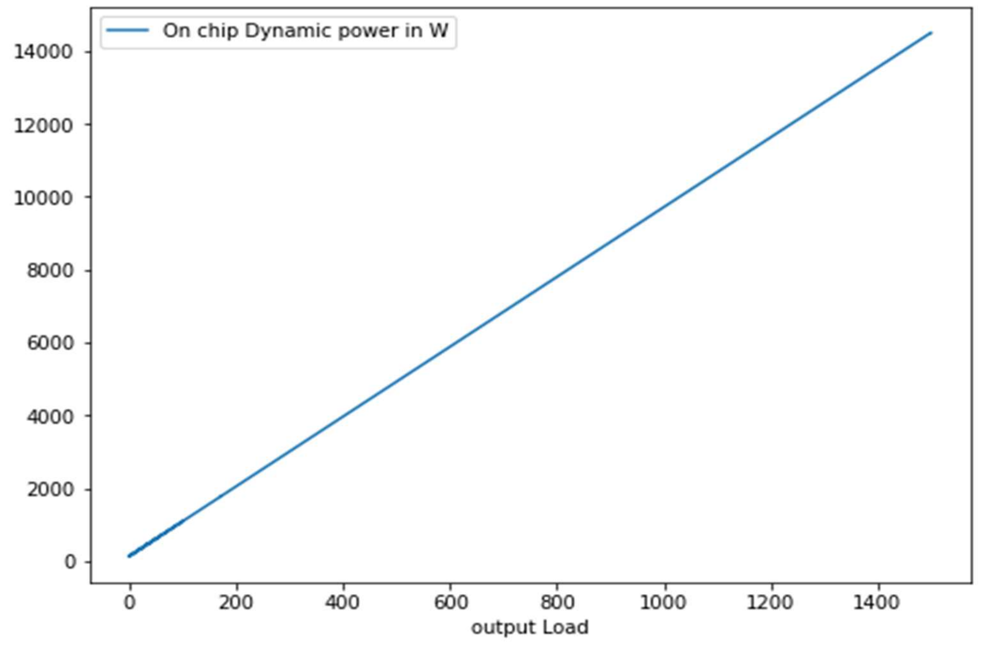


Figure 5.11 (i) Logic power vs Output Load of RISC Processor **(ii)** Signal power vs Output Load of RISC Processor

- On Chip Dynamic Power

Figure 5.12 (i) shows the variation of on chip dynamic power with respect to output load. The graph shows that as the output load increases dynamic power increases linearly. Also theoretically, from the expression, $P_{dynamic} = C_L V_{DD}^2 f$.



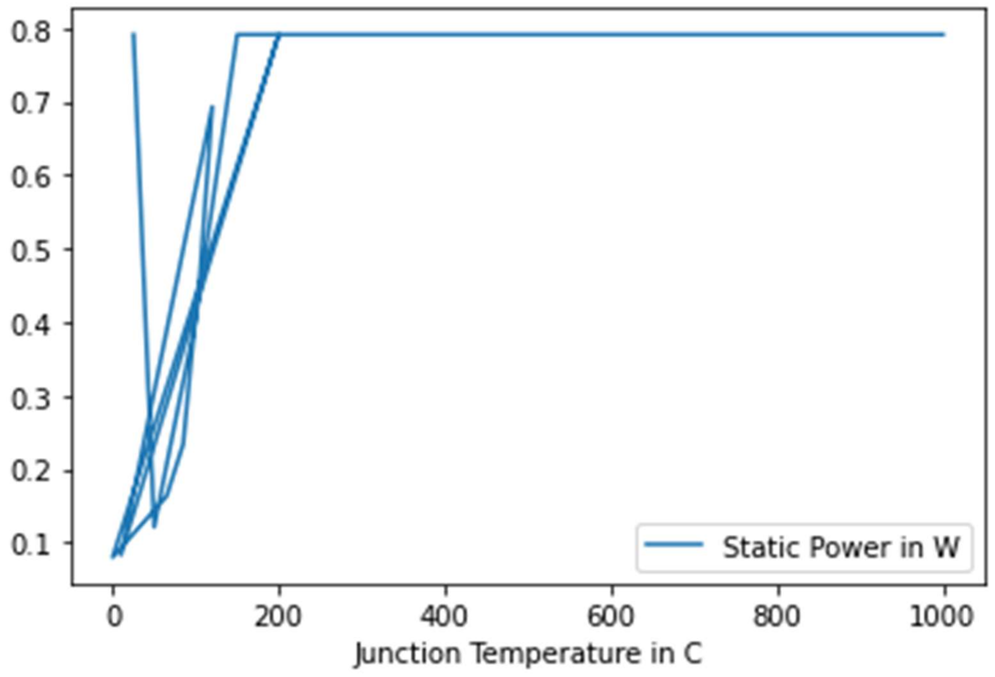


Figure 5.12 (i) On Chip Dynamic power vs Output Load of RISC Processor **(ii)**

Static Power vs Junction Temperature of RISC processor.

The dynamic on/off chip power of a digital circuit depends upon the switching activities on input and output i.e. charging and discharging of output load capacitance and $P_{dynamic} \alpha C_L$. Hence, graph shows the correct relationship between the dynamic power and output load.

(ii) PVT conditions - Junction Temperature :

The power of processor depends upon the PVT conditions as well. Process, Voltage and Temperature are the reasons power of processor may vary. In this thesis, we have focused on Temperature specifically. Temperature can be Ambient Temperature or Junction Temperature. Ambient Temperature comes into play when Processor chip is fabricated and in the surrounding how the power of processor varies. Junction Temperature is important while designing the processor chip and simulating and improving the design for better power consumption.

Factor considered here on which the power of RISC processor depends is Junction temperature. Analysis of all the power is done with respect to the parameter junction temperature. Figure 5.12 (ii) to Figure 5.14 (ii) shows the behavior of power when junction temperature was varied in C.

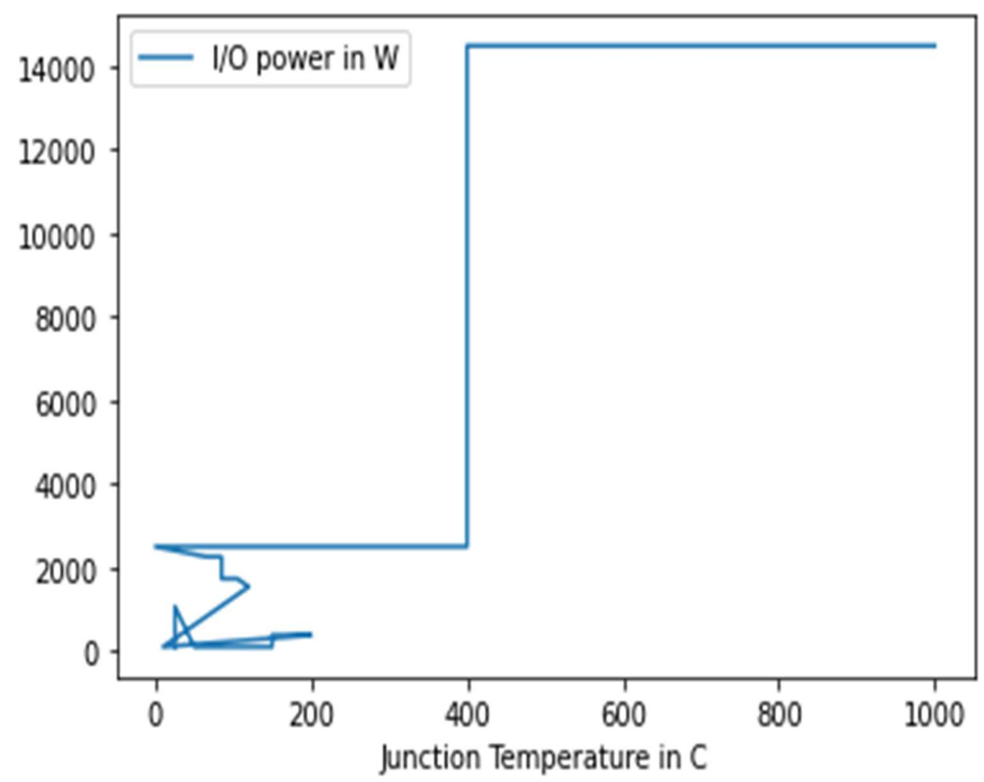
- Static Power

As shown in Figure 5.12 (ii) Static power initially varies linearly with some undefined behavior due to dependence on other factors too and after around 200 C, Static power is constant and does not change with respect to junction temperature.

- I/O power

Similarly, in figure 5.13 (i) when junction temperature is less than 400 C, I/O is constant i.e around 2500W and beyond this temperature, I/O power increases and becomes constant at around 14000W.

.



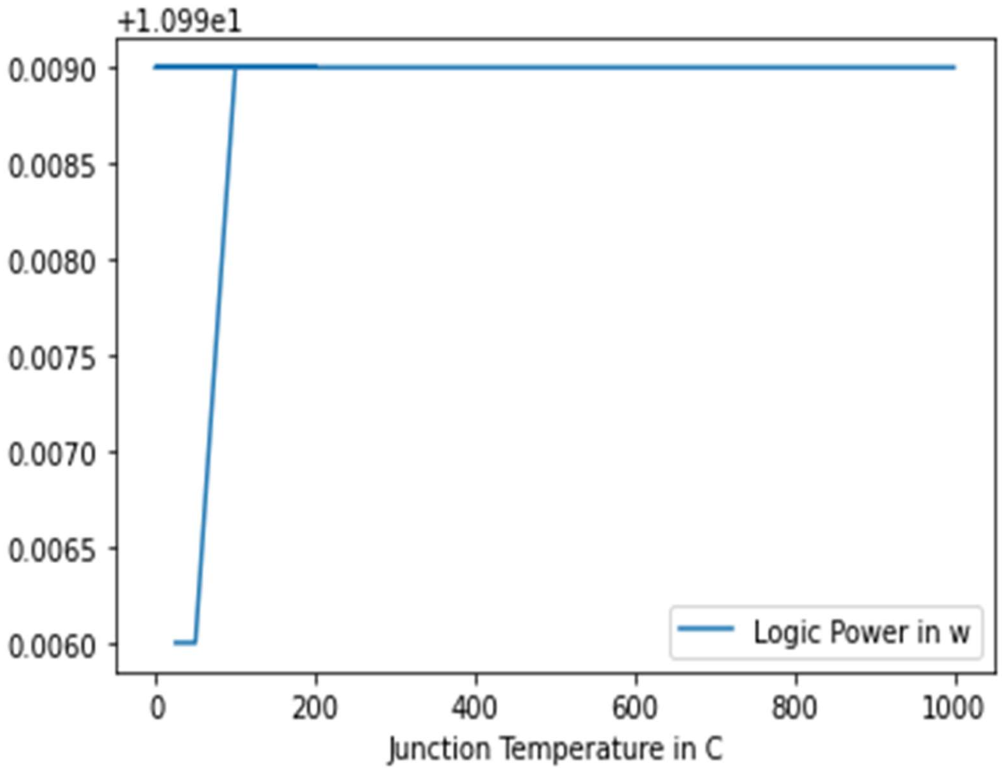


Figure 5.13 (i) I/O power vs Junction Temperature of RISC Processor **(ii)** Logic Power vs Junction Temperature of RISC processor

- Logic Power

For Logic power figure 5.13(ii) shows its variance with Junction Temperature i.e. increases linearly from 0.0060W to 0.0090W when Junction Temperature is less 150 C and after constant at 0.0090W if further increases in Junction Temperature

- Signal Power

Figure 5.14 (i) shows Signal power variation with the Junction Temperature of RISC Processor and figure 5.14 (ii) shows On chip Dynamic Power variation with the Junction Temperature of RISC processor. Signal power remains constant to 14.735W with the increase in the value of Junction Temperature.

- On Chip Dynamic Power

On chip dynamic power shows undefined behavior and it can be approximated as constant at around 2500W when junction temperature is less than 400 C and constant to approximately at 14000W when Junction Temperature is more than 400 C. So, an abrupt change in On chip dynamic power is visible when Junction Temperature was increased beyond a fixed value.

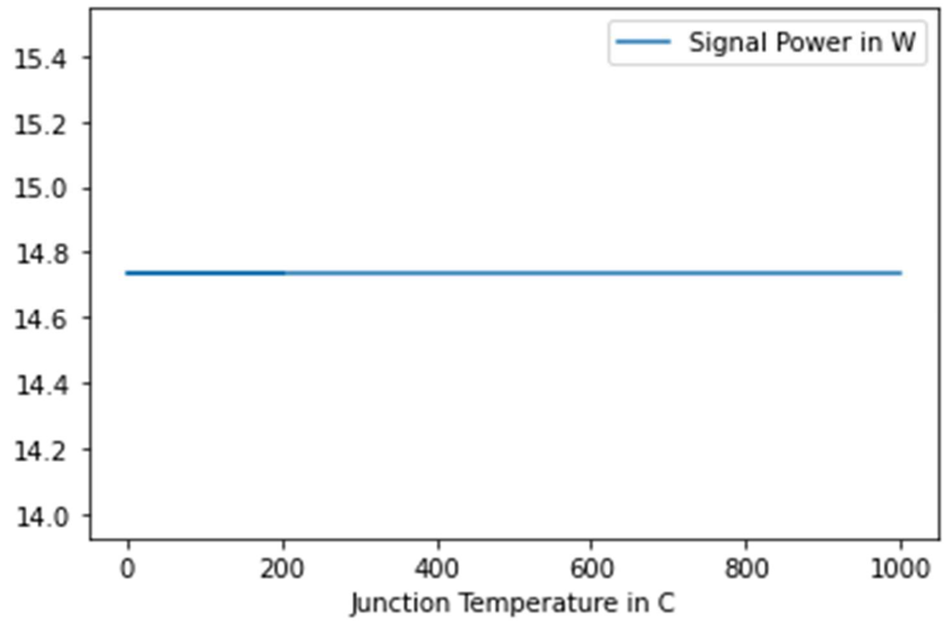


Figure 5.14 (i) Signal power vs Junction Temperature of RISC

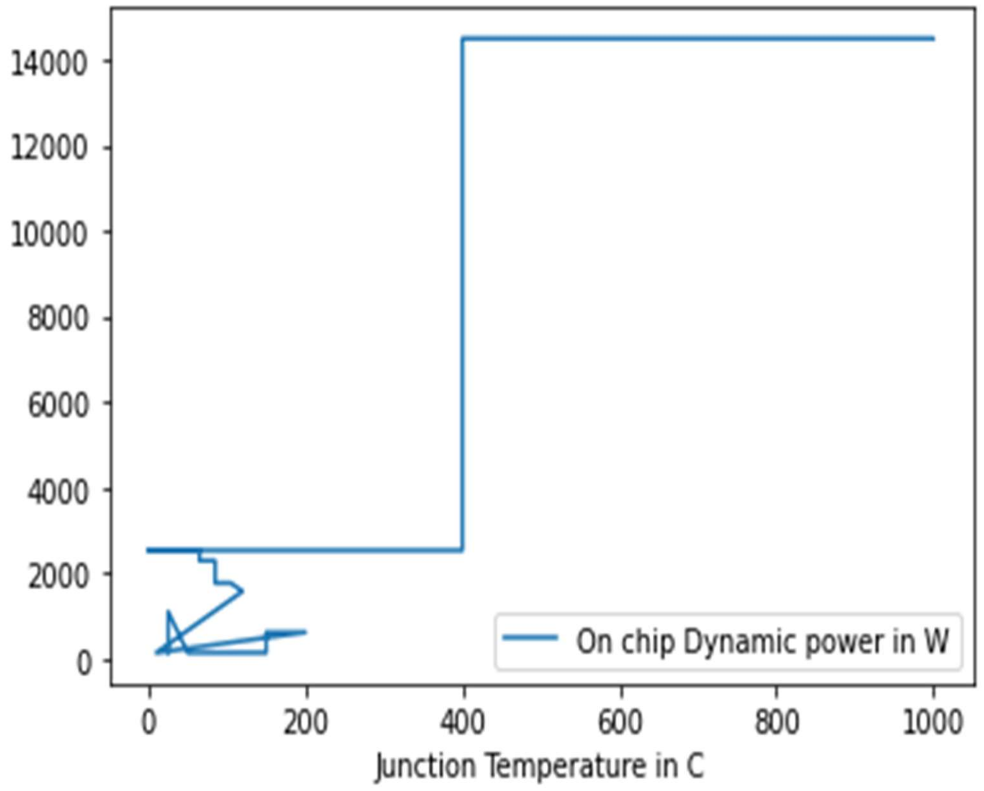


Figure 5.14 (i) Signal power vs Junction Temperature of RISC Processor **(ii)** On chip Dynamic Power vs Junction Temperature of RISC processor

CORRELATION MATRIX

Correlation of Input and Output Features :

The correlation matrix gives the idea about the amount of correlation between the the input and output features. To know the relation between the various input and output features among themselves and among each other. Higher the value of co - variance means they are highly related to each other.

Figure 15 represents the correlation between each i.e. input and output feature of a data sample

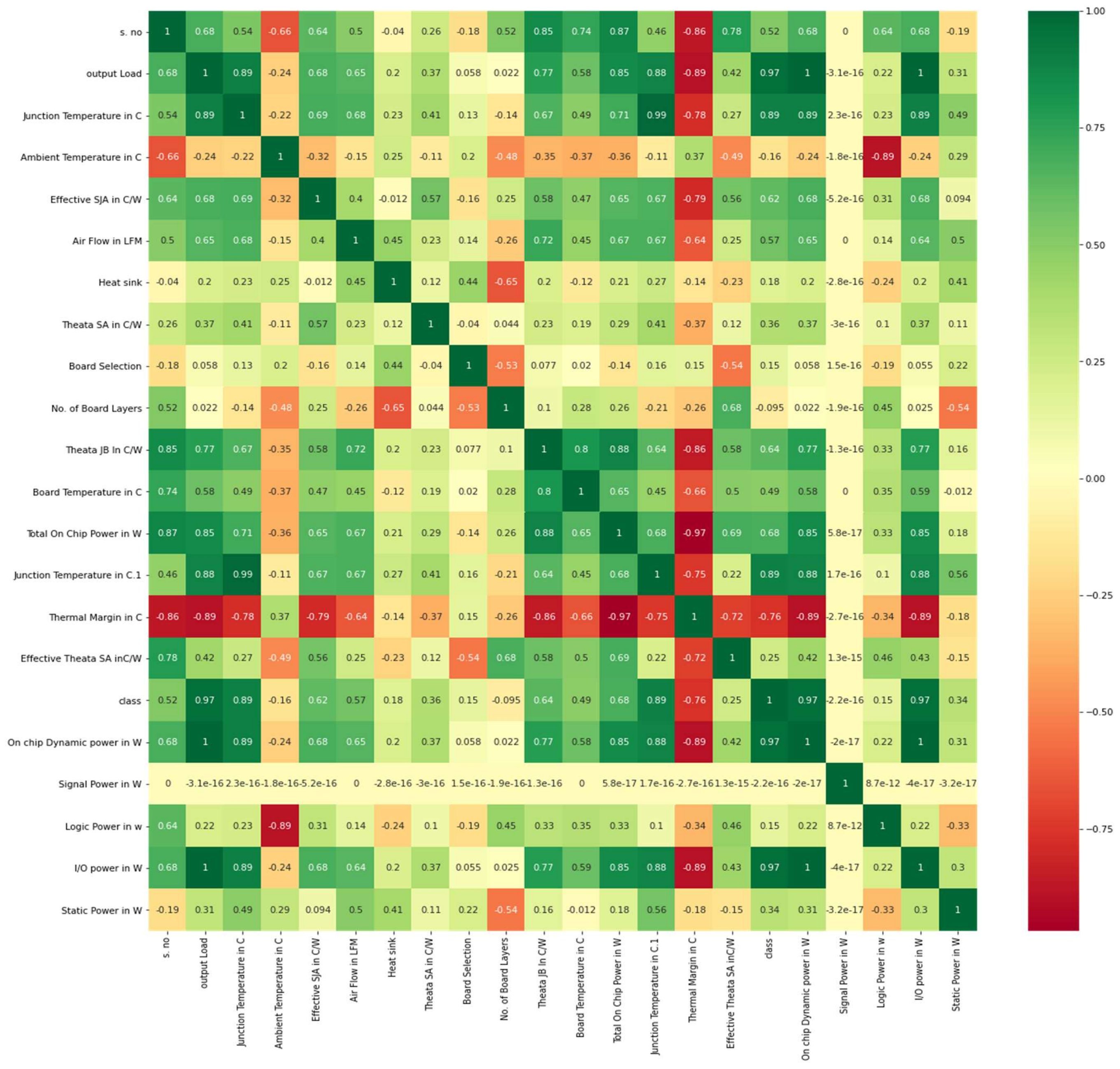


Figure 15 Correlation of each features in dataset

. They are also called correlation heatmap mainly helpful in the dataset analysis. It plots the covariance of the features of the data sample. It varies from -1 to +1. From the graph, the correlation between the output load and junction temperature is 0.89, output load and air flow is 0.65, on chip power and output load is 1, Logic power and output load is 0.22, I/O power and output load is 1 etc

Table 1 Comparative classification report showing performance metric of Machine Learning algorithms.

S.No.	Classification Report of Machine Learning Algorithms				
1.	**Isolation Forest**		**Accuracy Score 0.88 No. of class ‘1’ 3**		
	Performance Metrics	**Precision**	**Recall**	**F1-Score**	**Support**
	Class 0	0.95	0.90	0.93	21
	Class 1	0.60	0.75	0.67	4
	Accuracy			0.88	25
	Macro Average	0.77	0.83	0.80	25
	Weighted Average	0.89	0.88	0.89	25
2.	**Local Outlier Factor**		**Accuracy Score 0.96 No. of class ‘1’ 1**		
	Performance Metrics	**Precision**	**Recall**	**F1-Score**	**Support**
	Class 0	1.00	0.95	0.98	21
	Class 1	0.80	1.00	0.89	4
	Accuracy			0.96	25
	Macro Average	0.90	0.98	0.93	25
	Weighted Average	0.97	0.96	9.96	25
3.	**Support Vector Machine**		**Accuracy Score 0.56 No. of class ‘1’ 11**		
	Performance Metrics	**Precision**	**Recall**	**F1-Score**	**Support**
	Class 0	0.86	0.57	0.69	21
	Class 1	0.18	0.50	0.27	4

	Accuracy			0.56	25
	Macro Average	0.52	0.54	0.48	25
	Weighted Average	0.75	0.56	0.62	25
4.	**Naïve Bayes Classifier**		**Accuracy Score 0.84 No. of class ‘1’ 4**		
	Performance Metrics	**Precision**	**Recall**	**F1-Score**	**Support**
	Class 0	0.84	1.00	0.91	21
	Class 1	0.00	0.00	0.00	4
	Accuracy			0.84	25
	Macro Average	0.42	0.50	0.46	25
	Weighted Average	0.71	0.84	0.77	25
5.	**K-Nearest Neighbor Classifier**		**Accuracy Score 0.84 No. of class ‘1’ 4**		
	Performance Metrics	**Precision**	**Recall**	**F1-Score**	**Support**
	Class 0	0.84	1.00	0.91	21
	Class 1	0.00	0.00	0.00	4
	Accuracy			0.84	25
	Macro Average	0.42	0.50	0.46	25
	Weighted Average	0.71	0.84	0.77	25

Based on the performance metrics in the Table 1, The algorithm for the highest accuracy is the Local Outlier Factor i.e. 0.96, highest precision is Local Outlier factor i.e. 1.00, highest Recall is Naïve Bayes and KNN Classifier for Class'0'and Local Outlier Factor for class '1', highest F1-Score is Local Outlier Factor. Following Figure 12 shows the comparative graphical analysis of these five machine learning algorithms (SVM, IF, LOF, KNN, NB) on the basis of performance metrics such as Precision, Recall, F1 score and Accuracy. On the basis of the graph, Local Outlier Factor (LOF) is the best suited machine learning algorithm for the analysis of Power dataset of a RISC Processor.

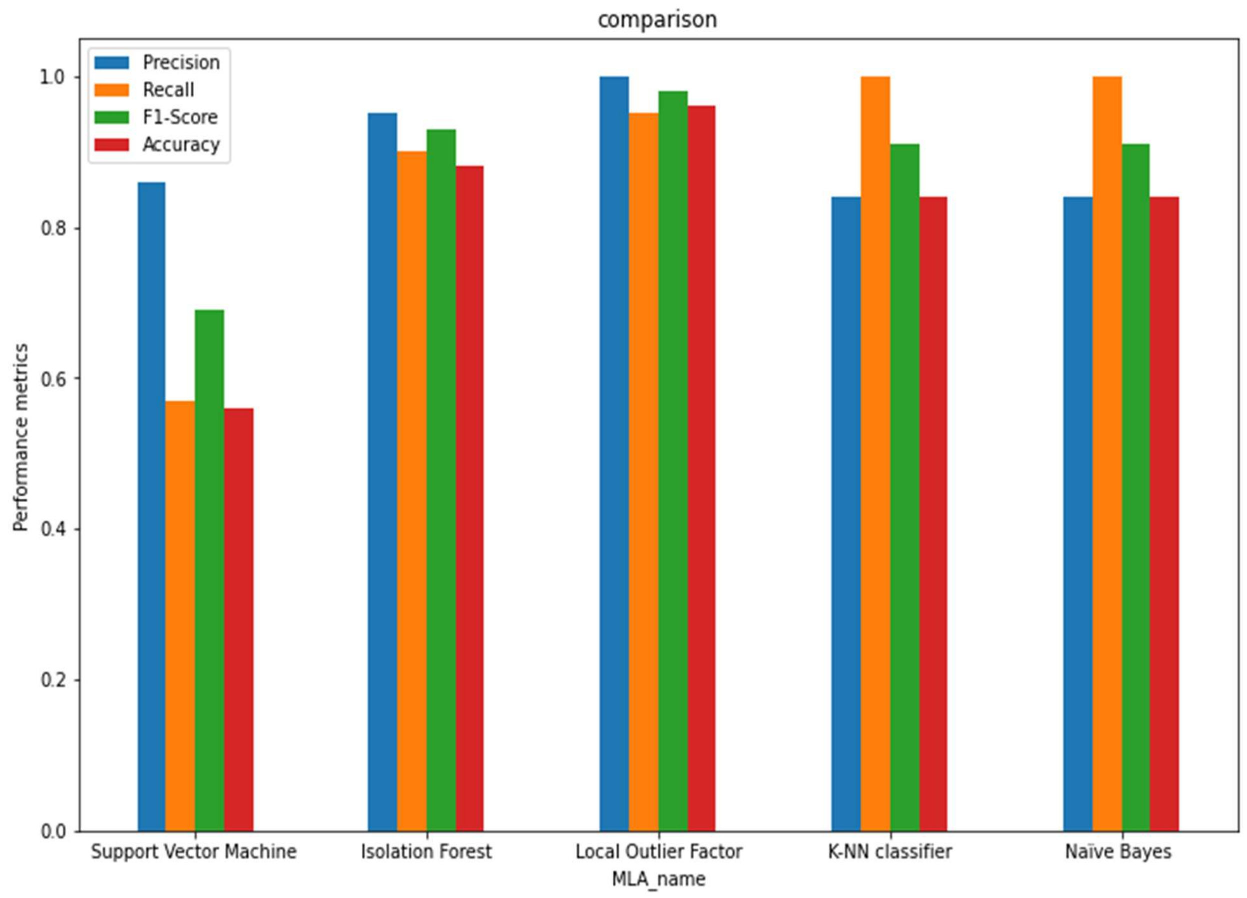


Figure 16 Comparative graphical analysis showing performance metric of Machine Learning algorithms.

CHAPTER 6
CONCLUSION AND FUTURE SCOPE

The design architecture is written in VERILOG HDL code using Xilinx VIVADO tool for synthesis and simulation. RISC processor is designed to carry out various arithmetic and logic instructions. The various blocks like ALU are simulated and synthesized using the software Xilinx VIVADO Simulator. Constraint files were added for timing and Implemented by selecting the ARTIX-7 FPGA Board. This thesis also comprehends the relevance and utilization of ML algorithms at the RTL level of a digital circuit design. We have studied and simulated a 32 bit MIPS RISC Processor. On implementation, power reports were generated and recorded in a .csv file.

This dataset was analyzed and understood that power dissipation mainly depends on the output capacitive load and vaguely on the Junction Temperature. On the analyzing the power dataset, we were able to find out that dynamic and signal power varied linearly with the output load capacitance and signal, I/O and static power were approximately close to a constant value. On generating the correlation matrix, we identified the covariance of input and output features among each other. Furthermore, we implemented five ML algorithms – KNN Classifier, Isolation Forest (IF), Local outlier Factor (LOF), SVM, and NB classifier. The order of accuracy on the power data set of these algorithms is as follows: Local outlier factor > Isolation Forest >KNN Classifier = NB Classifier > Support Vector Machine. Along with the accuracy, other performance metrics have also been evaluated and arranged in a tabular form.

We can improve the performance, power and area of the implemented processor by reducing the instruction and finding out machine learning algorithms that can fit well for the power analysis. It can also be improved by adding more strength to the instruction set, which has the capability perform the required operations which in turn increases the speed/performance of the RISC Processor. We further can look forward to the application of more ML algorithms with an extensive dataset on the same processor. Moreover, we can modify the RISC processor and perform the analysis using same machine learning algorithms to know the application of ML algorithms at the RTL level of an IP or SoC. Machine learning has a very bright scope at every abstraction level of designing a chip.

We need apt machine algorithms that can find and optimize the critical path of the design to improve the timing performance of RISC Processor. Area and power consumption can be reduced by low power techniques such as logic restructuring and pin ordering etc.

References

[1] https:// www.ibm.com/ibm/history/ibm100/us/en/icons/risc/

[2] Preetam bhosale, Hari Krishna Murti, "FPGA implementation of Low Power Pipelined 32 Bit RISC Processor " IJITEE Volume-1, Issue-3,August 2012

[3] Pranjali S. Kelgaonkar and Shilpa Kodgire, "Design of 32 Bit MIPS RISC processor based on Soc," *International Journal of Latest Trends in Engineering and Technology (IJLTET)*, Volume-6, Issue-3, January 2016.

[4] M. Xue and C. Zhu, "A Study and Application on Machine Learning of Artificial Intellligence," 2009 International Joint Conference on Artificial Intelligence, Hainan Island, 2009, pp. 272-274, doi: 10.1109/JCAI.2009.55.

[5] Samuel, Arthur (1959). "Some Studies in Machine Learning Using the Game of Checkers". *IBM Journal of Research and Development*. **3** (3):210-229. CiteSeerX 10.1.1.368.2254.

[6] A. Nayak and K. Dutta, "Impacts of machine learning and artificial intelligence on mankind," *2017 International Conference on Intelligent Computing and Control (I2C2)*, Coimbatore, 2017.

[7] S. Ray, "A Quick Review of Machine Learning Algorithms," *2019 International Conference on Machine Learning, Big Data, Cloud and Parallel Computing (COMITCon)*, Faridabad, India, 2019, pp. 35-39.

[8] **Elfadel**, Ibrahim (Abe) M., **Boning**, Duane S., **Li**, Xin (Eds.), Machine Learning in VLSI Computer-Aided Design, 2019, Springer International Publishing, Springer Nature Switzerland AG.

[9] Qiang He and Jun-Fen Chen, "The inverse problem of support vector machines and its solution," 2005 International Conference on Machine Learning and Cybernetics, Guangzhou, China, 2005, pp. 4322-4327 Vol. 7.

[10] Yi-Min Huang and Shu-Xin Du, "Weighted support vector machine for classification with uneven training class sizes," 2005 International Conference on Machine Learning and Cybernetics, Guangzhou, China, 2005, pp. 4365-4369 Vol. 7.

[11] A. Mathur and G. M. Foody, "Multiclass and Binary SVM Classification: Implications for Training and Classification Users," in *IEEE Geoscience and Remote Sensing Letters*, vol. 5, no. 2, pp. 241-245, April 2008.

[12] D. Xu, Y. Wang, Y. Meng and Z. Zhang, "An Improved Data Anomaly Detection Method Based on Isolation Forest," *2017 10th International Symposium on Computational Intelligence and Design (ISCID)*, Hangzhou, China, 2017, pp. 287-291.

[13] S. Hariri, M. C. Kind and R. J. Brunner, "Extended Isolation Forest," in IEEE Transactions on Knowledge and Data Engineering, vol. 33, no. 4, pp. 1479-1489, 1 April 2021.

[14] F. T. Liu, K. M. Ting and Z. Zhou, "Isolation Forest," 2008 Eighth IEEE International Conference on Data Mining, Pisa, Italy, 2008, pp. 413-422.

[15] H. Budiarto, A. Erna Permanasari and S. Fauziati, "Unsupervised Anomaly Detection Using K-Means, Local Outlier Factor and One Class SVM," 2019 5th International Conference on Science and Technology (ICST), Yogyakarta, Indonesia, 2019, pp. 1-5.

[16] Breunig, Markus M., et al. "LOF: identifying density-based local outliers." ACM sigmod record. Vol. 29. No. 2. ACM, 2000

[17] W. Wang and P. Lu, "An Efficient Switching Median Filter Based on Local Outlier Factor," in IEEE Signal Processing Letters, vol. 18, no. 10, pp. 551-554, Oct. 2011.

[18] F. Zhang, F. Yin and G. Huang, "An Optimized LOF algorithm Based on Tree structure," 2020 3rd International Conference on Artificial Intelligence and Big Data (ICAIBD), Chengdu, China, 2020, pp. 167-171.

[19] F. Sanei, A. Harifi and S. Golzari, "Improving the precision of KNN classifier using nonlinear weighting method based on the spline interpolation," 2017 7th International Conference on Computer and Knowledge Engineering (ICCKE), Mashhad, Iran, 2017, pp. 289-292.

[20] Belur V. Dasarathy, " Nearest Neighbor (NN) Norms: NN Pattern Classification Techniques", Mc Graw-Hill Computer Science Series, IEEE Computer Society Press, Las Alamitos, California, pp.217- 224,1991

[21] M. Manjusha and R. Harikumar, "Performance analysis of KNN classifier and K-means clustering for robust classification of epilepsy from EEG signals," 2016 International Conference on Wireless Communications, Signal Processing and Networking (WiSPNET), Chennai, India, 2016, pp. 2412-2416.

[22] S. F. Rodiyansyah and E. Winarko, "Klasifikasi Posting Twitter Kemacetan Lalu Lintas Kota Bandung Menggunakan Naive Bayesian Classification," FMIPA UGM, vol. 6, no. 1, pp. 91–100, 2012

[23] S. Ernawati, E. R. Yulia, Frieyadie and Samudi, "Implementation of The Naïve Bayes Algorithm with Feature Selection using Genetic Algorithm for Sentiment Review Analysis of Fashion Online Companies," 2018 6th International Conference on Cyber and IT Service Management (CITSM), Parapat, Indonesia, 2018, pp. 1-5.

[24] M. Dixit, R. Sharma, S. Shaikh and K. Muley, "Internet Traffic Detection using Naïve Bayes and K-Nearest Neighbors (KNN) algorithm," 2019 International Conference on Intelligent Computing and Control Systems (ICCS), Madurai, India, 2019, pp. 1153-1157.

[25] R.S. Anu Gowsalya, S. Miruna, Joe Amali, “Naive Bayes Based Network Traffic Classification Using Correlation Information”, International Journal of Advanced Research in Computer Science and Software Engineering Volume 4 issue 3

[26] Andrew W. Moore, Denis Zeuy, “ Internet Traffic Classification Using Bayesian Analysis Techniques”, SIGMETRICS '05 Proceedings of the 2005 ACM SIGMETRICS international conference on Measurement and modeling of computer systems

[27] www.Xilinx.com/itp/xilinx10/isehelp/ise_c_fpga_design__ow_overview.htm.

[28] A verilog HDL Primer, J.Bhasker

[29] "Xilinx Inc, Form 8-K, Current Report, Filing Date Apr 25, 2012". secdatabase.com. Retrieved May 6, 2018.

[30] EDN. "The Vivado Design Suite accelerates programmable systems integration and implementation by up to 4X." Jun 15, 2012. Retrieved Jun 25, 2013.

[31] "Vivado Design Suite 2014.1 Increases Productivity with Automation of UltraFast Design Methodology and OpenCL Hardware Acceleration". SAN JOSE: Market Watch. 2014-04-16.

[32] Morris, Kevin (2014-05-06). "Viva Vivado!, Xilinx Tunes-Up Tools". Electronic Engineering Journal.

APPENDIX A

Procedure to calculate the power samples using Xilinx Vivado software

1. Start with the Xilinx Vivado by double-clicking the Vivado desktop icon .

2. By clicking on the **Create Project**, a New Project window will open.

3. Choose **Next** so as to open Project Name dialog box.

- Type our project name
- Choose a location where we want to save your project files.
- Select **Create project subdirectory** check box
- Click **Next**.

4. A Project Type page will appear, select **RTL Project**, and click **Next**.

5. Next will be the **Add Sources** page, set Target language to **Verilog**.

6. Set the simulator language to **Mixed** and click **Next**.

6. Choose and add your constraint file in the Add Constraints page, go for **Next** button.

7. Next, we have a Default Part page,

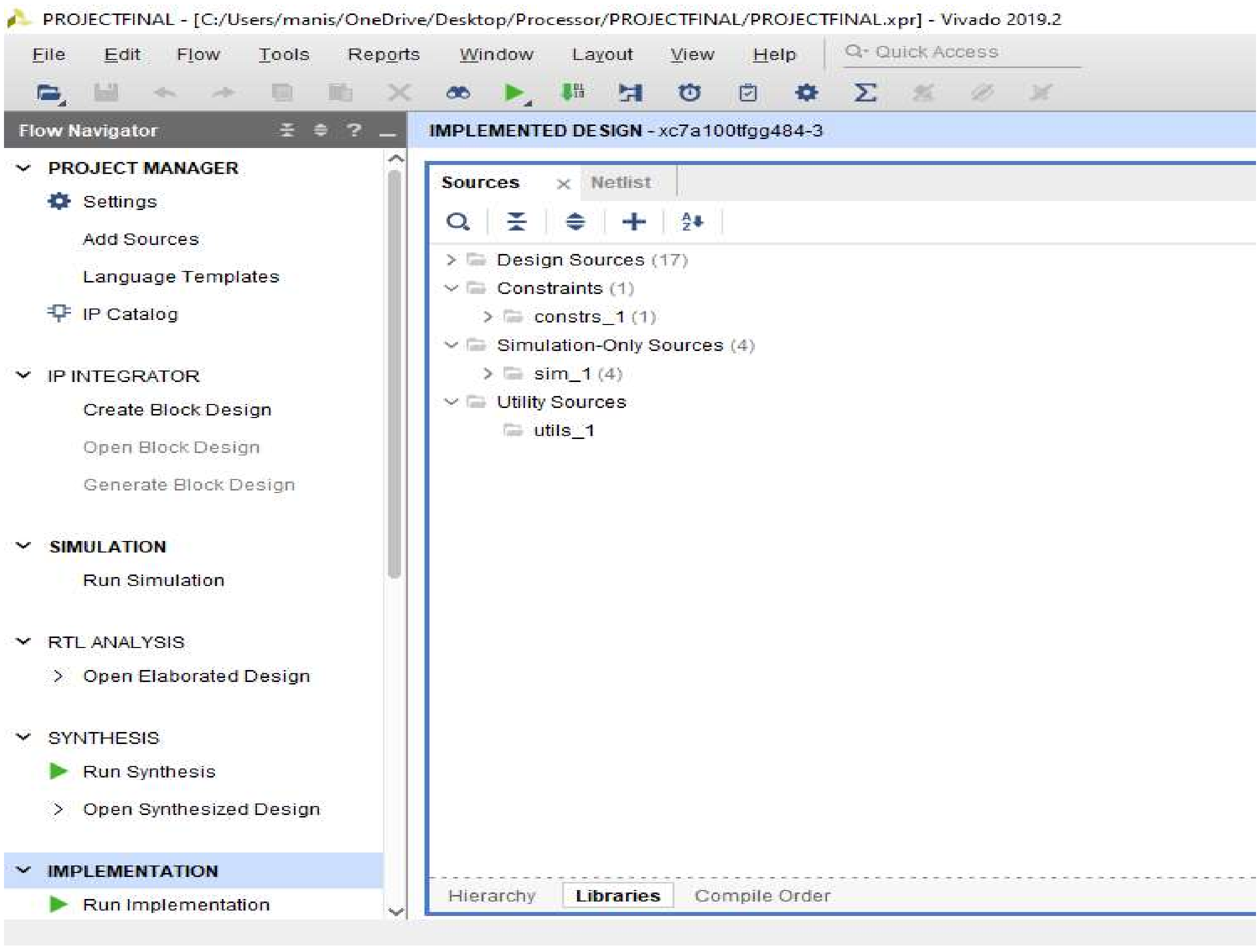
- Select **Boards**.
- Select **All** to view all versions of the supported boards in Xilinx Vivado.
- Select the version of the Artix-7 Evaluation Board.
- Click on the **Next** button.

After this, we write the design code and simulation code in verilog for the 32 bit RISC processor and the file for it is with .v extension which is included in the Design sources. This design source and simulation source can be successfully simulated for by clicking on the **Run Simulation.**

This will check the behvioral working of the code whether the design code with the help of applied simulation source is prodcing the correct output or not. The

Figure shows the sources included for the implemented design.

Along the side we can see different views like Simulation, RTL analysis,Synthesis and implementation. We can simulate our design code by clicking on the **Run Simulation** tab. We can generate the RTL schematic by clicking on the **Open Elaborated Design** tab. We can synthesize the code by choosing **Run Synthesis** tab and can view the implementation by choosing the **Run Implementation** tab.


PROJECTFINAL - [C:/Users/manis/OneDrive/Desktop/Processor/PROJECTFINAL/PROJECTFINAL.xpr] - Vivado 2019.2

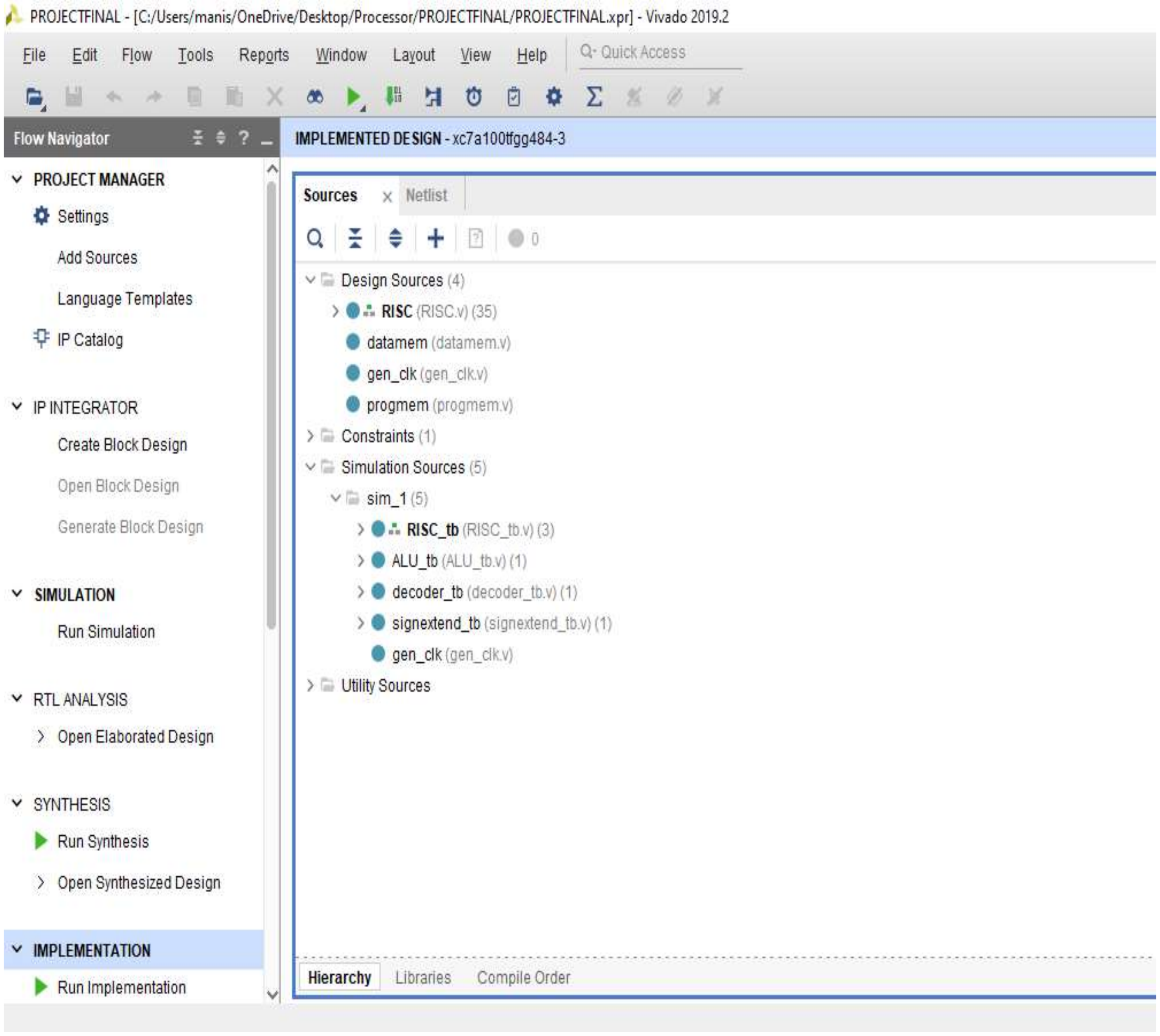
IMPLEMENTED DESIGN - xc7a100tfgg484-3

- Design Sources (17)
- Constraints (1)
  - constrs_1 (1)
- Simulation-Only Sources (4)
  - sim_1 (4)
- Utility Sources
  - utils_1

Figure Sources included in the Implemented Design.

Xilinx Vivado facilitates the incorporation of constraint files known as XDC file. It is becomes a challenging task for the novice user to write design constraint file as per own because we have limited knowledge regarding the physical connections of the FPGA pins on the board.

We can add test benches which appear in the Sources view hierarchy. Multiple test benches can be added to the project. All source files like design sources, constraint sources and simulation sources of the project are simulated by clicking on the Simulation option. Figure shows the main source files in design, constrain and simulation. When you add a test bench to the project, you *must* ensure that the associated Design View is set to a Simulation view. For simulation source files, Project Navigator automatically selects the Design View association based on the *file name*.

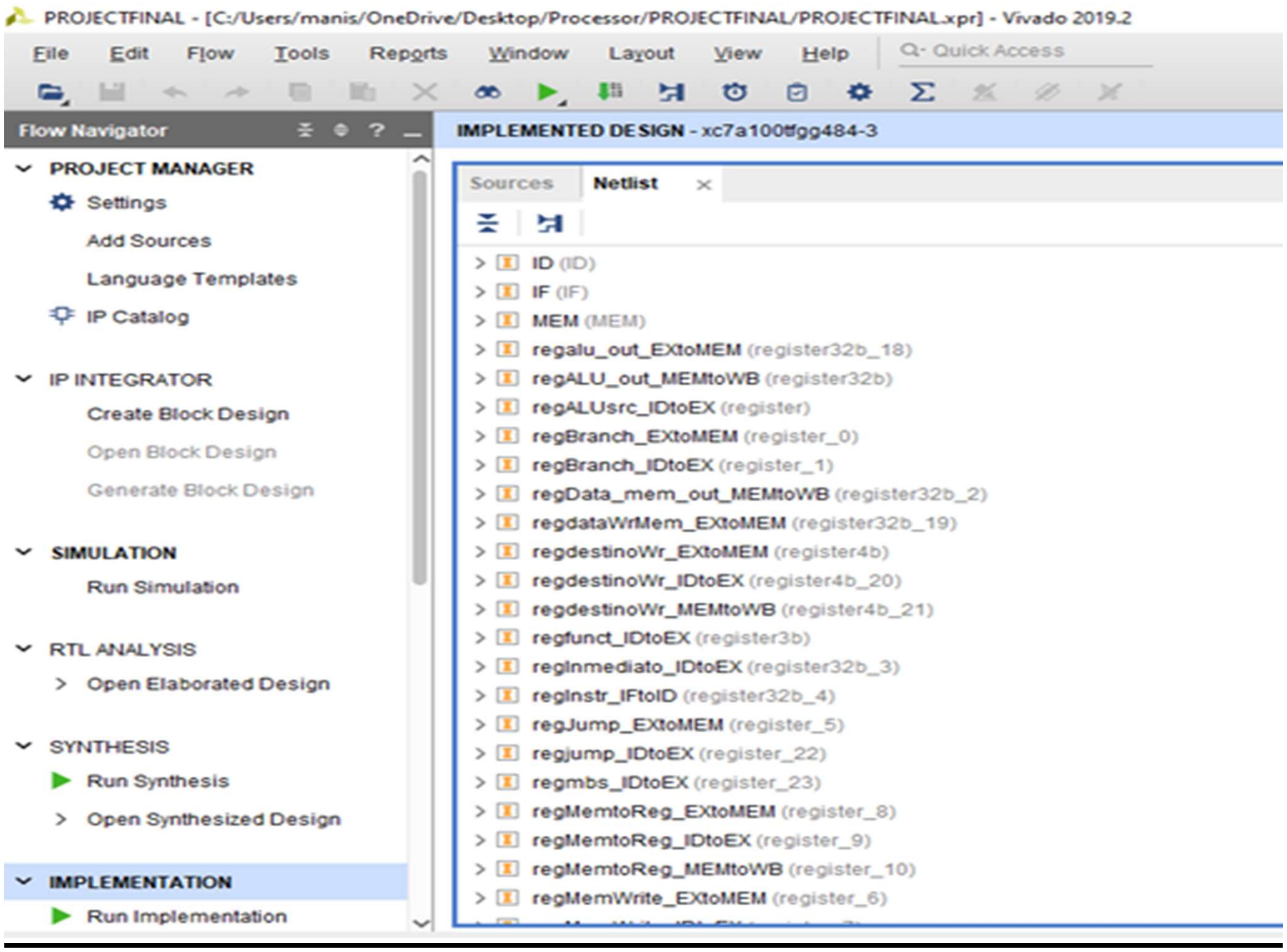

PROJECTFINAL - [C:/Users/manis/OneDrive/Desktop/Processor/PROJECTFINAL/PROJECTFINAL.xpr] - Vivado 2019.2

IMPLEMENTED DESIGN - xc7a100tfgg484-3

```
Design Sources (4)
  RISC (RISC.v) (35)
  datamem (datamem.v)
  gen_clk (gen_clk.v)
  progmem (progmem.v)
Constraints (1)
Simulation Sources (5)
  sim_1 (5)
    RISC_tb (RISC_tb.v) (3)
    ALU_tb (ALU_tb.v) (1)
    decoder_tb (decoder_tb.v) (1)
    signextend_tb (signextend_tb.v) (1)
    gen_clk (gen_clk.v)
Utility Sources
```

Figure Design, constraint and simulation source files.

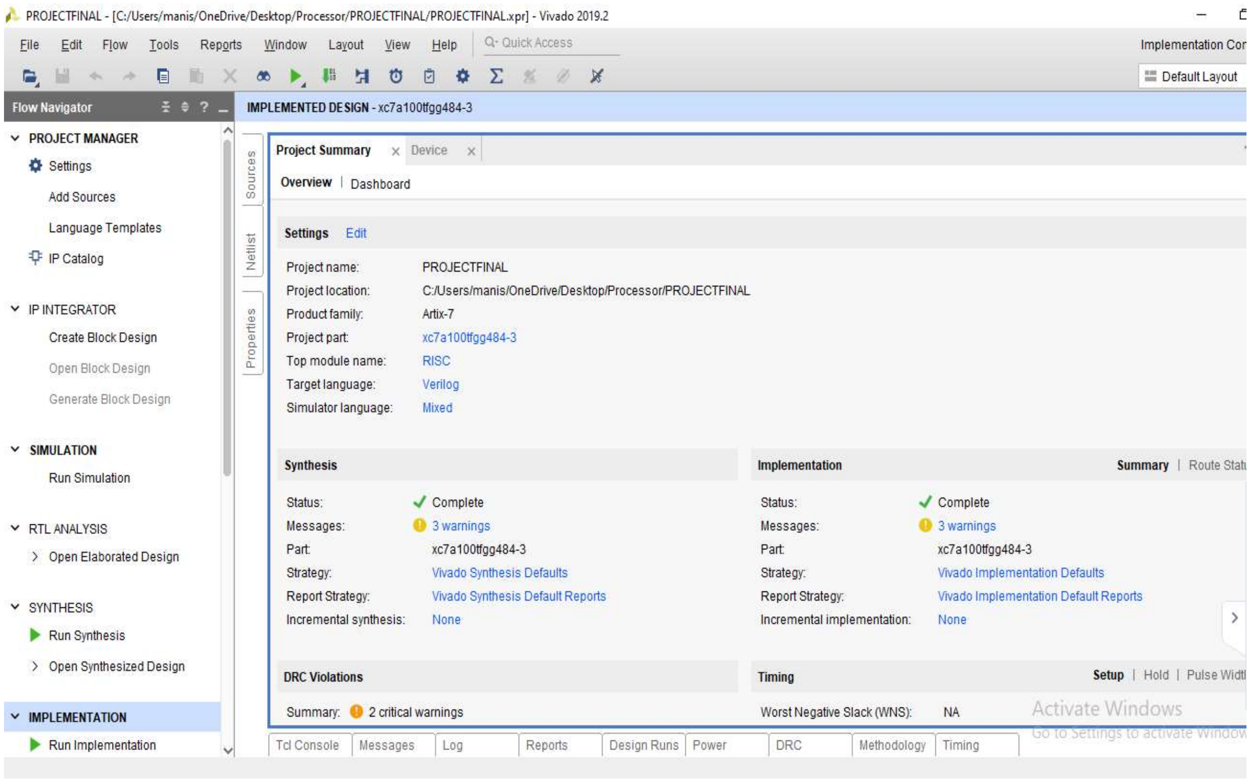
The design description comprising of ports, registers, flops, cells, pins, and nets as design device, user defined modules, library elements instances(BELs), LUTs, FF, RAMs, DSP, etc. is known as Netlist. The following Figure shows the netlist section of the implemented RISC processor design code showing the ID,IF, RegALU, MEM, registers used for ALU_OUT for each stage transition i.e EX to MEM, MEM to WB, ID to EX etc. Each of the netlist refers to digital circuit of the processor components like datapath, control unit, ALU etc.



ID (ID)
IF (IF)
MEM (MEM)
regalu_out_EXtoMEM (register32b_18)
regALU_out_MEMtoWB (register32b)
regALUsrc_IDtoEX (register)
regBranch_EXtoMEM (register_0)
regBranch_IDtoEX (register_1)
regData_mem_out_MEMtoWB (register32b_2)
regdataWrMem_EXtoMEM (register32b_19)
regdestinoWr_EXtoMEM (register4b)
regdestinoWr_IDtoEX (register4b_20)
regdestinoWr_MEMtoWB (register4b_21)
regfunct_IDtoEX (register3b)
regInmediato_IDtoEX (register32b_3)
regInstr_IFtoID (register32b_4)
regJump_EXtoMEM (register_5)
regjump_IDtoEX (register_22)
regmbs_IDtoEX (register_23)
regMemtoReg_EXtoMEM (register_8)
regMemtoReg_IDtoEX (register_9)
regMemtoReg_MEMtoWB (register_10)
regMemWrite_EXtoMEM (register_6)

Figure Netlist of the Implemented Design

The project summary gives the overview of the setting, synthesis, implementation, summary, DRC violations, Timing, Setup, Utilization and power consumed by the design. Following Figure shows the project summary of a Project 1.



PROJECTFINAL - [C:/Users/manis/OneDrive/Desktop/Processor/PROJECTFINAL/PROJECTFINAL.xpr] - Vivado 2019.2

IMPLEMENTED DESIGN - xc7a100tfgg484-3

Settings Edit

Project name:	PROJECTFINAL
Project location:	C:/Users/manis/OneDrive/Desktop/Processor/PROJECTFINAL
Product family:	Artix-7
Project part:	xc7a100tfgg484-3
Top module name:	RISC
Target language:	Verilog
Simulator language:	Mixed

Synthesis

Status:	✓ Complete
Messages:	3 warnings
Part:	xc7a100tfgg484-3
Strategy:	Vivado Synthesis Defaults
Report Strategy:	Vivado Synthesis Default Reports
Incremental synthesis:	None

Implementation Summary | Route Stat

Status:	✓ Complete
Messages:	3 warnings
Part:	xc7a100tfgg484-3
Strategy:	Vivado Implementation Defaults
Report Strategy:	Vivado Implementation Default Reports
Incremental implementation:	None

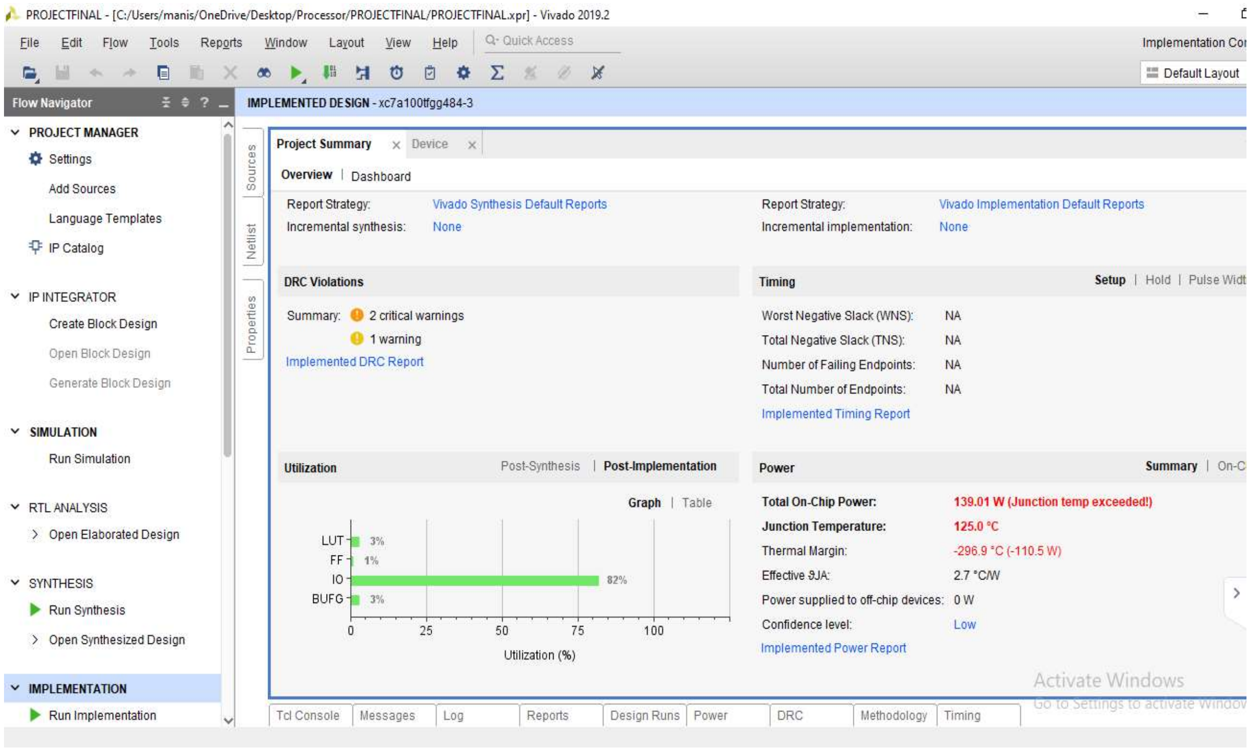
DRC Violations

Summary: 2 critical warnings

Timing Setup | Hold | Pulse Widt

Worst Negative Slack (WNS): NA

Figure The summary of the project 1



PROJECTFINAL - [C:/Users/manis/OneDrive/Desktop/Processor/PROJECTFINAL/PROJECTFINAL.xpr] - Vivado 2019.2

IMPLEMENTED DESIGN - xc7a100tfgg484-3

Report Strategy:	Vivado Synthesis Default Reports
Incremental synthesis:	None

Report Strategy:	Vivado Implementation Default Reports
Incremental implementation:	None

DRC Violations

Summary: 2 critical warnings
1 warning

Implemented DRC Report

Timing Setup | Hold | Pulse Widt

Worst Negative Slack (WNS):	NA
Total Negative Slack (TNS):	NA
Number of Failing Endpoints:	NA
Total Number of Endpoints:	NA

Implemented Timing Report

Utilization Post-Synthesis | Post-Implementation

Power Summary | On-C

Total On-Chip Power:	139.01 W (Junction temp exceeded!)
Junction Temperature:	125.0 °C
Thermal Margin:	-296.9 °C (-110.5 W)
Effective ϑJA:	2.7 °C/W
Power supplied to off-chip devices:	0 W
Confidence level:	Low

Implemented Power Report

Figure The summary of the project 2

Following Figure shows the Device level of the Implemented Design of RISC processor. This is a kind of a Floor Plan of the processor on FPGA, where it shows the schematic of how the design of the processor is placed in FPGA tentatively.
We can see partitions in the overall FPGA area available as X0Y0, X0Y1,X1,Y2 etc. as an individual blocks. Also, we can see that mainly only two blocks X0Y2 and X0Y1are used for the implementation of the processor we designed.

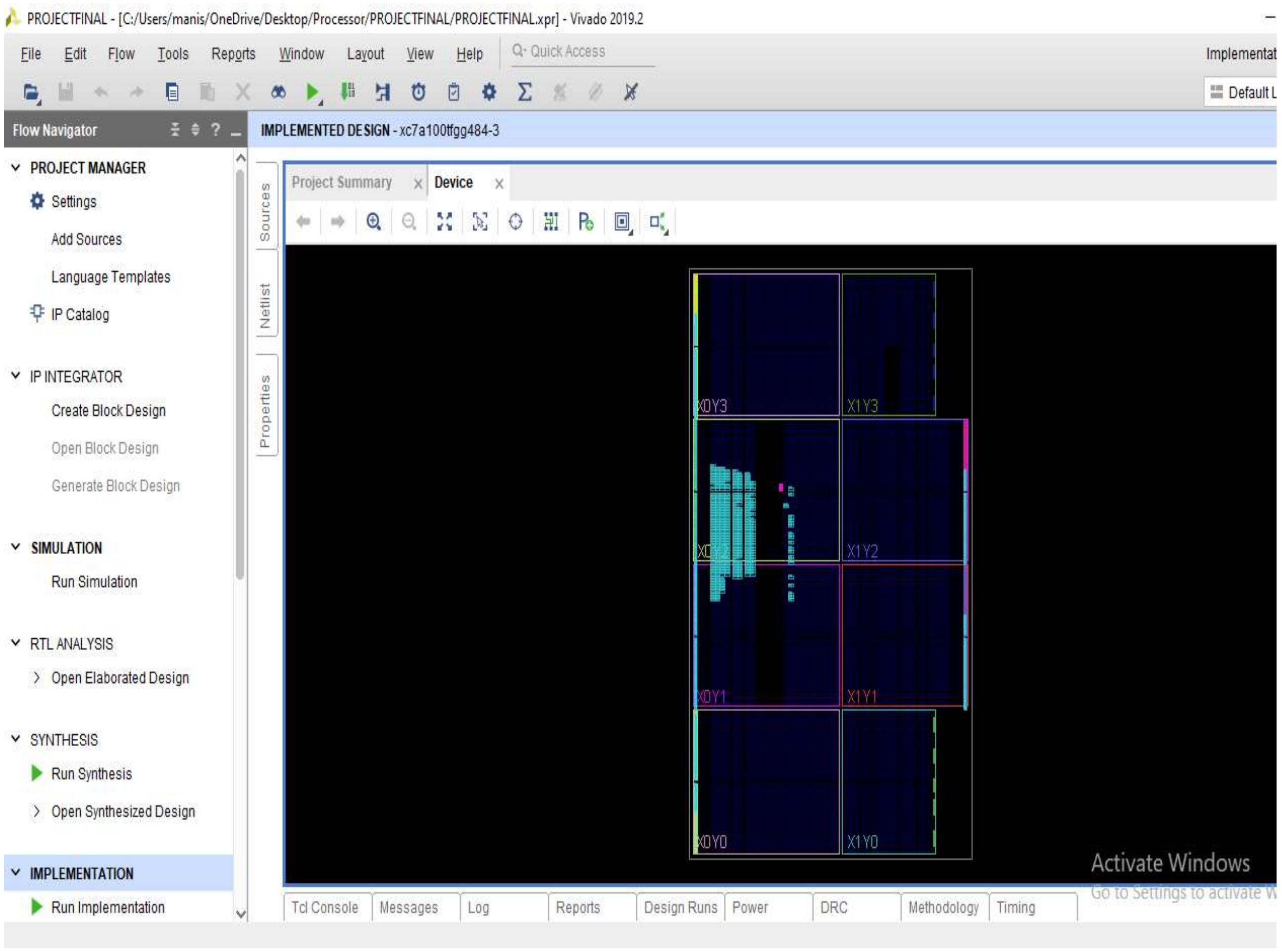


Figure Device of the Implemented Design

After we are completed with the design code, we move on to the next stage of generating the netlist of the design to see what the design code get synthesized to. The Xilinx Vivado incorporates an environment that customizes the IP and analyses the RTL. Elaborated Design in the Flow Navigator helps in detail analysis of code by using various RTL code DRCs to examine and improve the timing or power of the design. This generates simple and an elaborated netlist to also check synthesis of that design code.

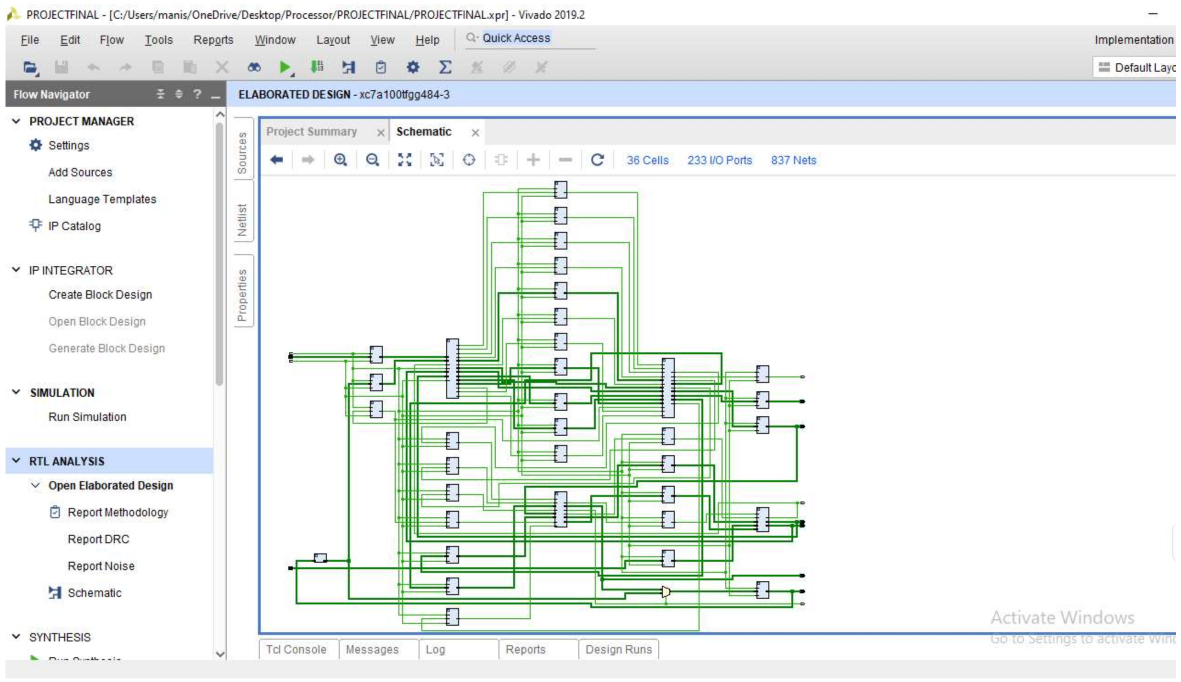


Figure RTL Schematic of the RISC Processor

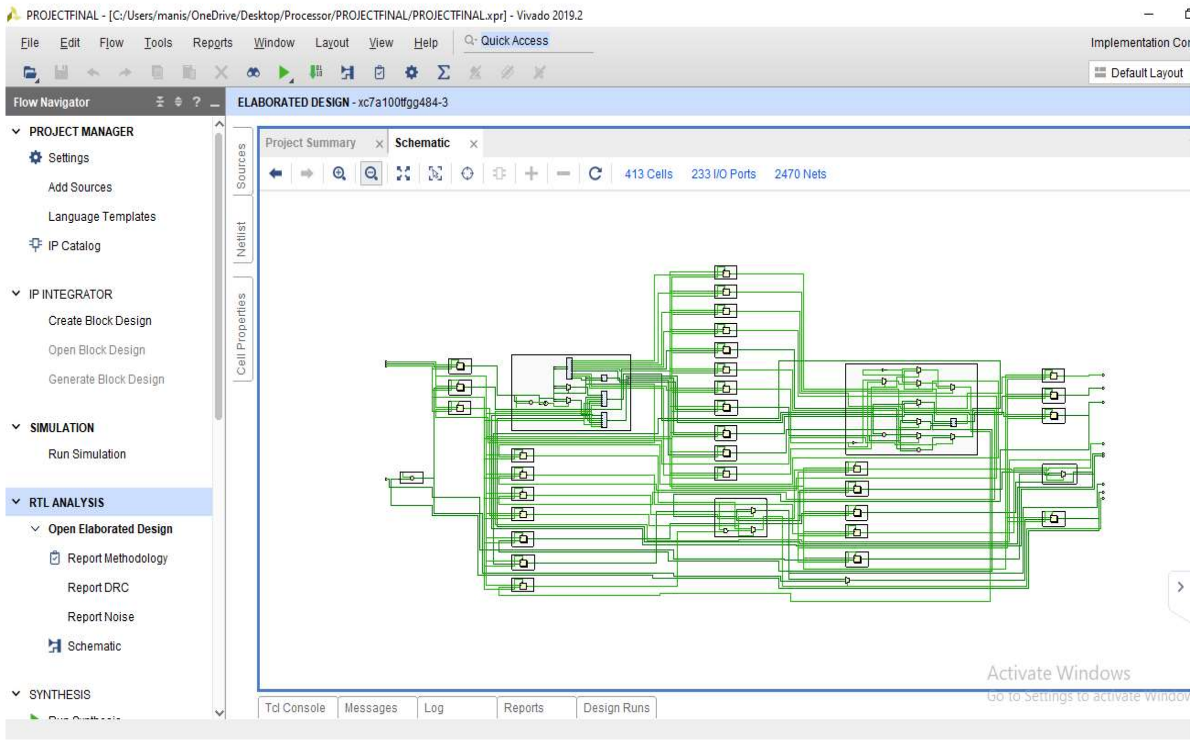


Figure Schematic of the Elaborated RISC Processor Design

The Elaborated RTL Design Schematic allows the numerous analytical views that include an RTL Netlist, Schematic, and Graphical Hierarchy. Cross select feature is included in the view section to lets us debug and optimize the RTL of the design code.

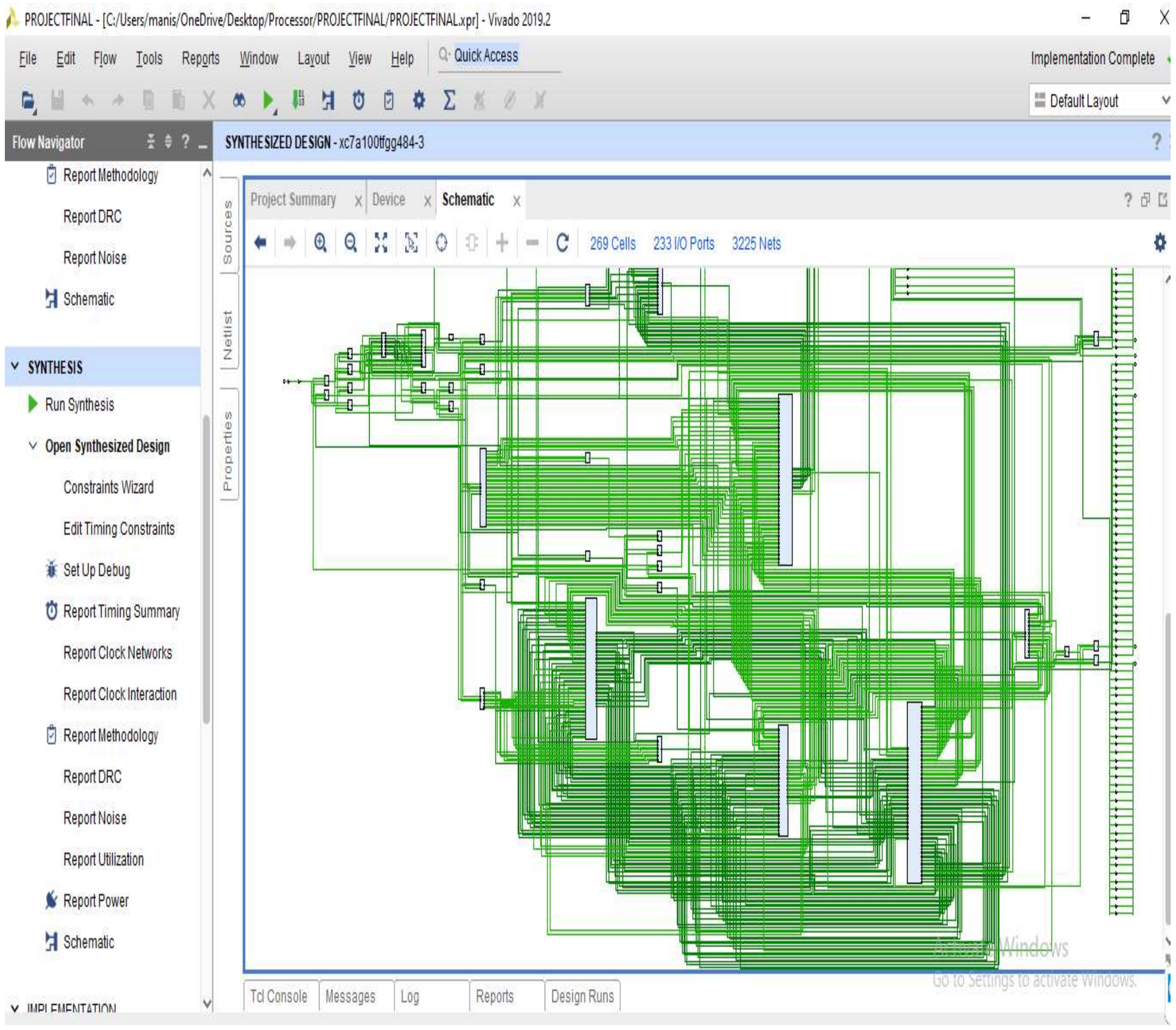


Figure Schematic of the Elaborated RISC Processor Design(Zoomed Version)

Next step is the checking the functional correctness of the design code of Processor. So we simulate and check its behavioral. After the design is working correctly, we generate the Power Report filling in the required inputs.

APPENDIX –B

Python Code used while modeling machine learning algorithm and analyzing both power dataset and different types of power consumptions.

CODE :

//Loading the libraries

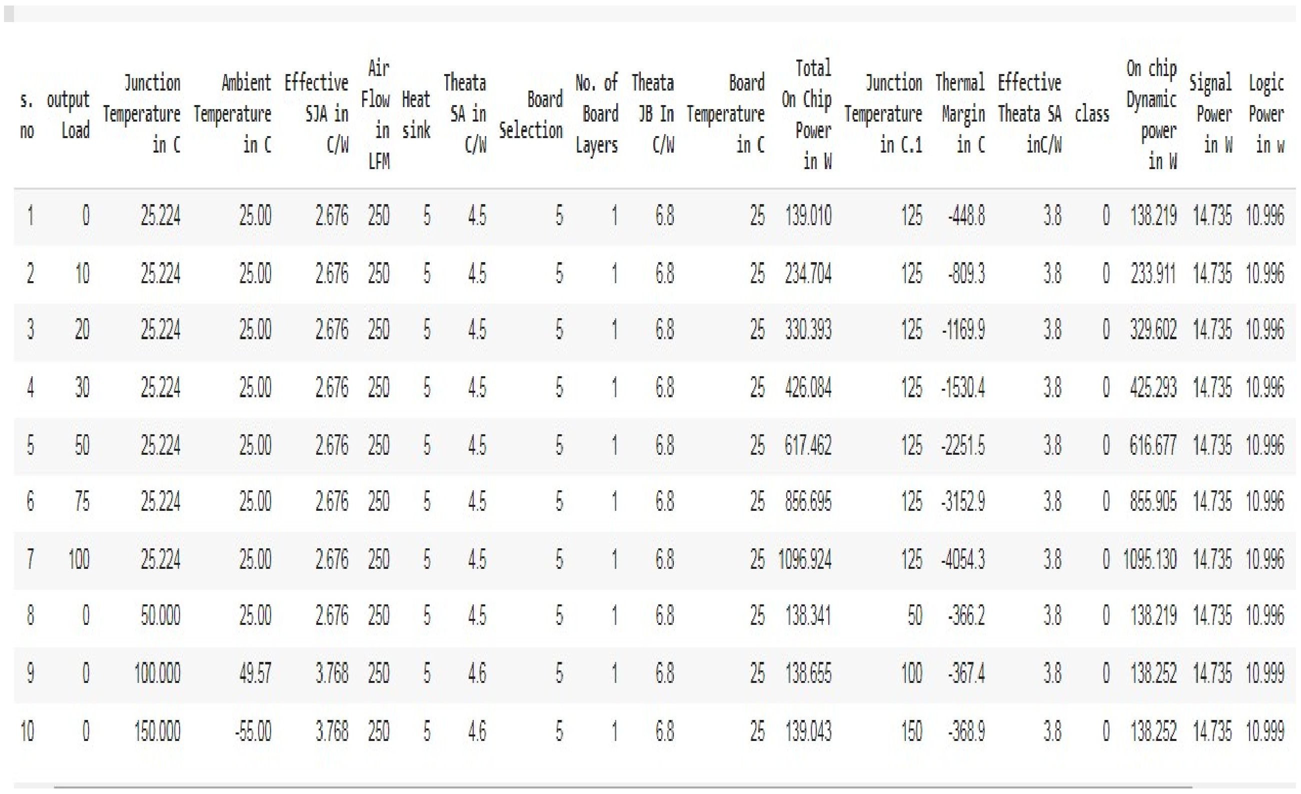
```
import numpy as np

import pandas as pd

import sklearn

import scipy

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.metrics import classification_report,accuracy_score

from sklearn.ensemble import IsolationForest

from sklearn.neighbors import LocalOutlierFactor

from sklearn.naive_bayes import GaussianNB

from sklearn.neighbors import KNeighborsClassifier

from sklearn.svm import OneClassSVM
from pylab import rcParams

rcParams['figure.figsize'] =12,8

RANDOM_SEED = 42
```

//Uploading the dataset

// Reading the dataset – first 10 entries

```
data = pd.read_csv('processor2.csv',sep=',')

data.head(n=10)
```

Following Figure showing first ten entries in the Power dataset.

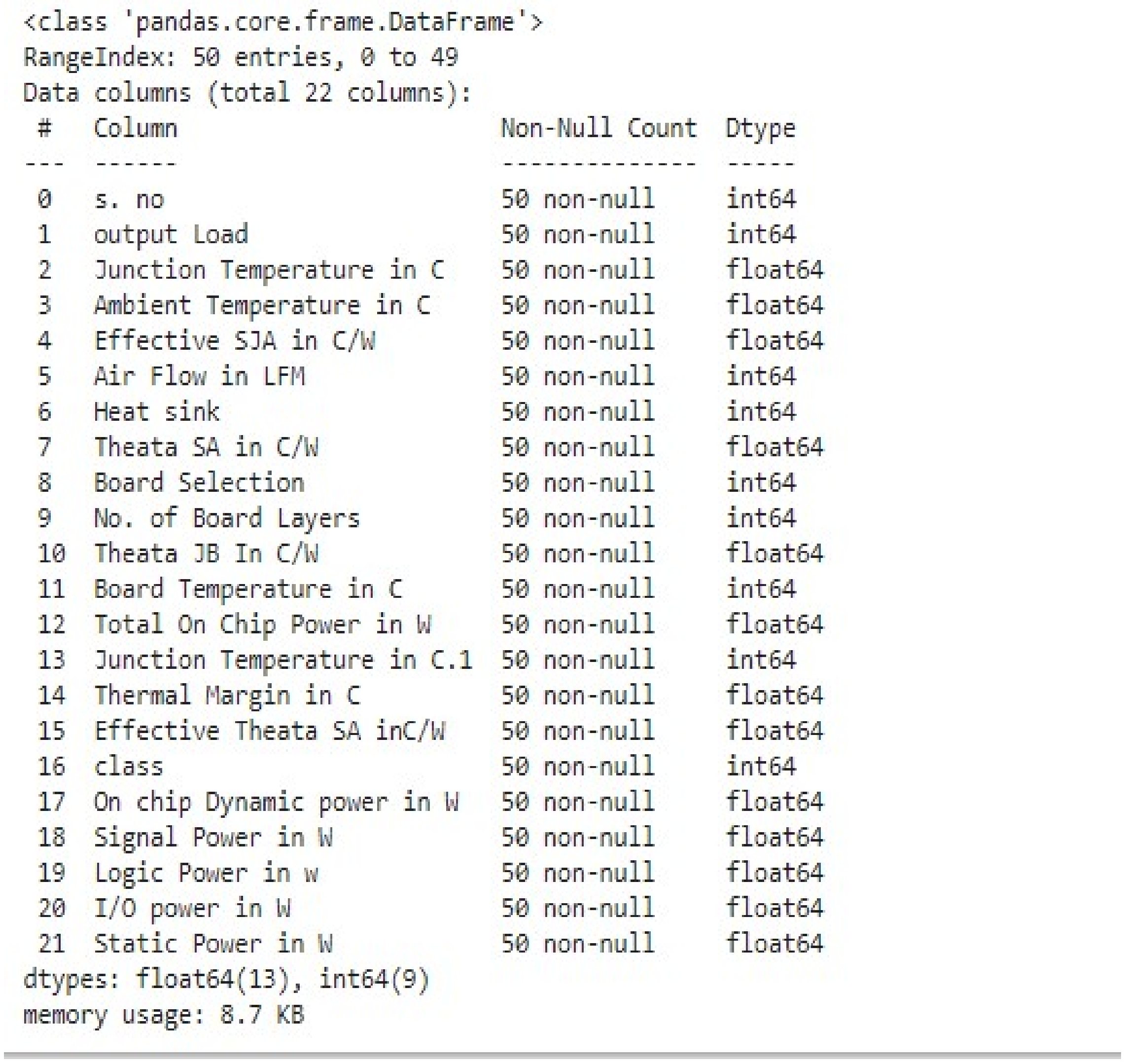


s. no	output Load	Junction Temperature in C	Ambient Temperature in C	Effective SJA in C/W	Air Flow in LFM	Heat sink	Theata SA in C/W	Board Selection	No. of Board Layers	Theata JB In C/W	Board Temperature in C	Total On Chip Power in W	Junction Temperature in C.1	Thermal Margin in C	Effective Theata SA inC/W	class	On chip Dynamic power in W	Signal Power in W	Logic Power in w
1	0	25.224	25.00	2.676	250	5	4.5	5	1	6.8	25	139.010	125	-448.8	3.8	0	138.219	14.735	10.996
2	10	25.224	25.00	2.676	250	5	4.5	5	1	6.8	25	234.704	125	-809.3	3.8	0	233.911	14.735	10.996
3	20	25.224	25.00	2.676	250	5	4.5	5	1	6.8	25	330.393	125	-1169.9	3.8	0	329.602	14.735	10.996
4	30	25.224	25.00	2.676	250	5	4.5	5	1	6.8	25	426.084	125	-1530.4	3.8	0	425.293	14.735	10.996
5	50	25.224	25.00	2.676	250	5	4.5	5	1	6.8	25	617.462	125	-2251.5	3.8	0	616.677	14.735	10.996
6	75	25.224	25.00	2.676	250	5	4.5	5	1	6.8	25	856.695	125	-3152.9	3.8	0	855.905	14.735	10.996
7	100	25.224	25.00	2.676	250	5	4.5	5	1	6.8	25	1096.924	125	-4054.3	3.8	0	1095.130	14.735	10.996
8	0	50.000	25.00	2.676	250	5	4.5	5	1	6.8	25	138.341	50	-366.2	3.8	0	138.219	14.735	10.996
9	0	100.000	49.57	3.768	250	5	4.6	5	1	6.8	25	138.655	100	-367.4	3.8	0	138.252	14.735	10.999
10	0	150.000	-55.00	3.768	250	5	4.6	5	1	6.8	25	139.043	150	-368.9	3.8	0	138.252	14.735	10.999

Figure First Ten entries in power dataset data.info()

// Above command prints the information about the dataset as all the columns and their Non-null count and Dtype

The following figure shows the information about the dataset as all the columns and their Non-null count and Dtype. Also, total entries as 50, all the input and output as the column, which will also be referred as instances. And each entries will be referred as samples.



```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 50 entries, 0 to 49
Data columns (total 22 columns):
 #   Column                        Non-Null Count  Dtype  
---  ------                        --------------  -----  
 0   s. no                         50 non-null     int64  
 1   output Load                   50 non-null     int64  
 2   Junction Temperature in C     50 non-null     float64
 3   Ambient Temperature in C      50 non-null     float64
 4   Effective SJA in C/W          50 non-null     float64
 5   Air Flow in LFM               50 non-null     int64  
 6   Heat sink                     50 non-null     int64  
 7   Theata SA in C/W              50 non-null     float64
 8   Board Selection               50 non-null     int64  
 9   No. of Board Layers           50 non-null     int64  
 10  Theata JB In C/W              50 non-null     float64
 11  Board Temperature in C        50 non-null     int64  
 12  Total On Chip Power in W      50 non-null     float64
 13  Junction Temperature in C.1   50 non-null     int64  
 14  Thermal Margin in C           50 non-null     float64
 15  Effective Theata SA inC/W     50 non-null     float64
 16  class                         50 non-null     int64  
 17  On chip Dynamic power in W    50 non-null     float64
 18  Signal Power in W             50 non-null     float64
 19  Logic Power in w              50 non-null     float64
 20  I/O power in W                50 non-null     float64
 21  Static Power in W             50 non-null     float64
dtypes: float64(13), int64(9)
memory usage: 8.7 KB
```

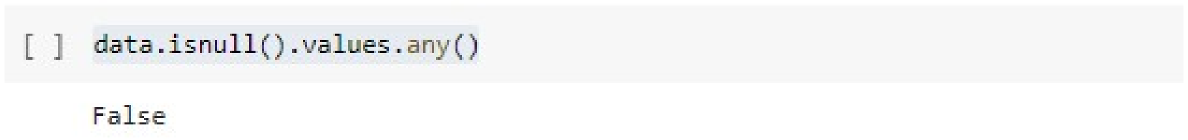
Figure Information about the dataset as all the columns and their Non-null count and Dtype

//Checking dataset if containing any null values or missing values

//Checking if the dataset is consistent

data.isnull().values.any()

Following Figure shows the output of the above snippet.



```
[ ]  data.isnull().values.any()

     False
```

//Power analysis

//Plotting the distribution of the different types of Power generated from the

//power report

// Static Power Distribution

count_classes = pd.value_counts(data['Static Power in W'], sort = True)

count_classes.plot(kind = 'pie', rot=0)

plt.title("Static Power Distribution")

plt.xlabel("Static Power")

plt.ylabel("No. of Samples")

// Logic Power in w Distribution

count_classes = pd.value_counts(data['Logic Power in w'], sort = True)

count_classes.plot(kind = 'pie', rot=0)

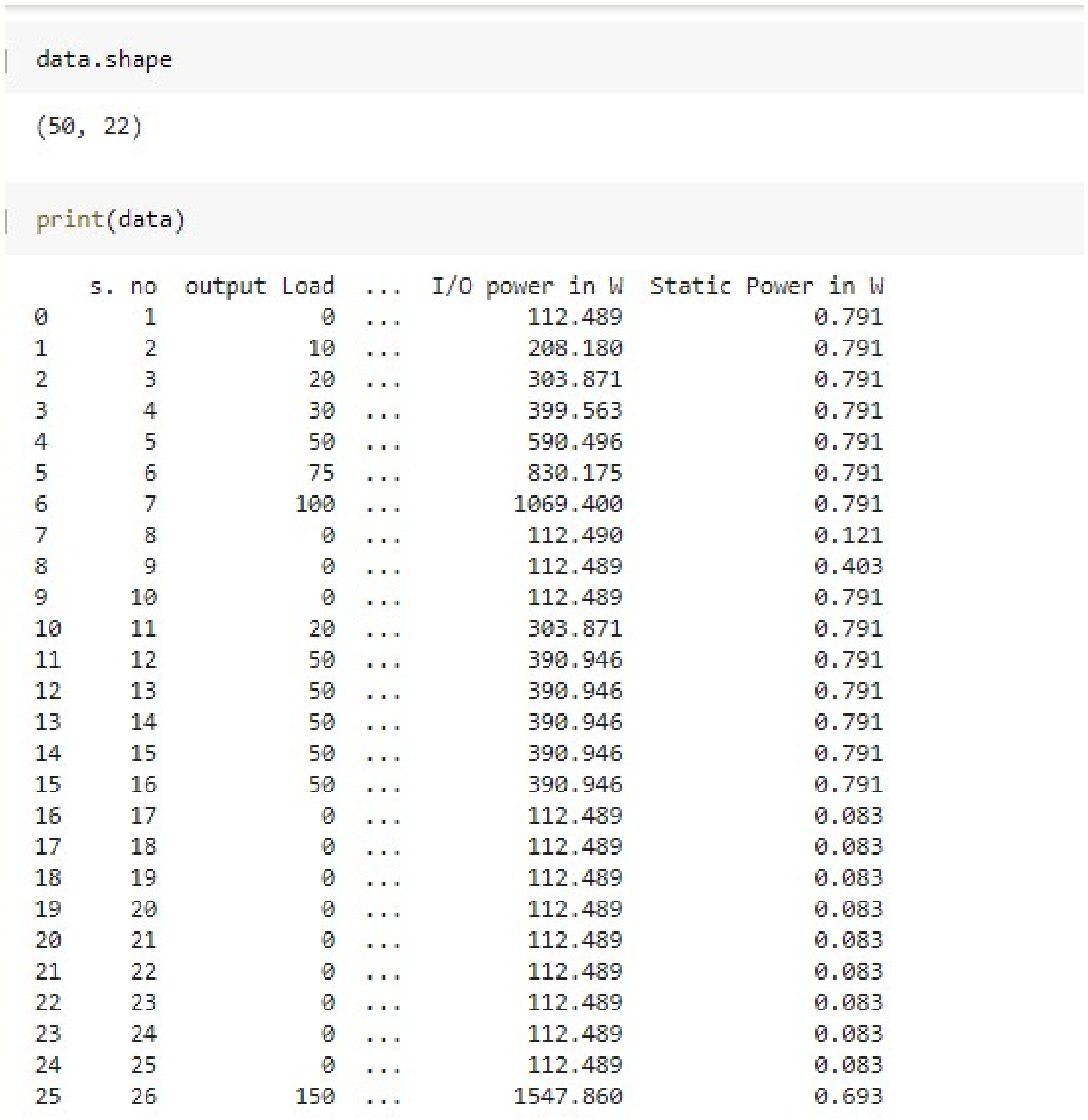
```
plt.title("Logic Power in w Distribution")

plt.xlabel("Logic Power in w")

plt.ylabel("No. of Samples")

// I/O Power Distribution

count_classes = pd.value_counts(data['I/O power in W'], sort = True)

count_classes.plot(kind = 'pie', rot=0)


plt.title("I/O Power Distribution")


plt.xlabel("I/O Power")

plt.ylabel("No. of Samples")

// Signal Power Distribution

count_classes = pd.value_counts(data['Signal Power in W'], sort = True)

count_classes.plot(kind = 'pie', rot=0)


plt.title("Signal Power  Distribution")


plt.xlabel("Signal Power")

plt.ylabel("No. of Samples")

//On chip Dynamic power in W

count_classes = pd.value_counts(data['On chip Dynamic power in W'], sort = True
)

count_classes.plot(kind = 'pie', rot=0)
```

plt.title("On chip Dynamic power Distribution")

plt.xlabel("On chip Dynamic power in W")

plt.ylabel("No. of Samples")

Following Figure shows data shape command and printing the data.



```
data.shape
```

```
(50, 22)
```

```
print(data)
```

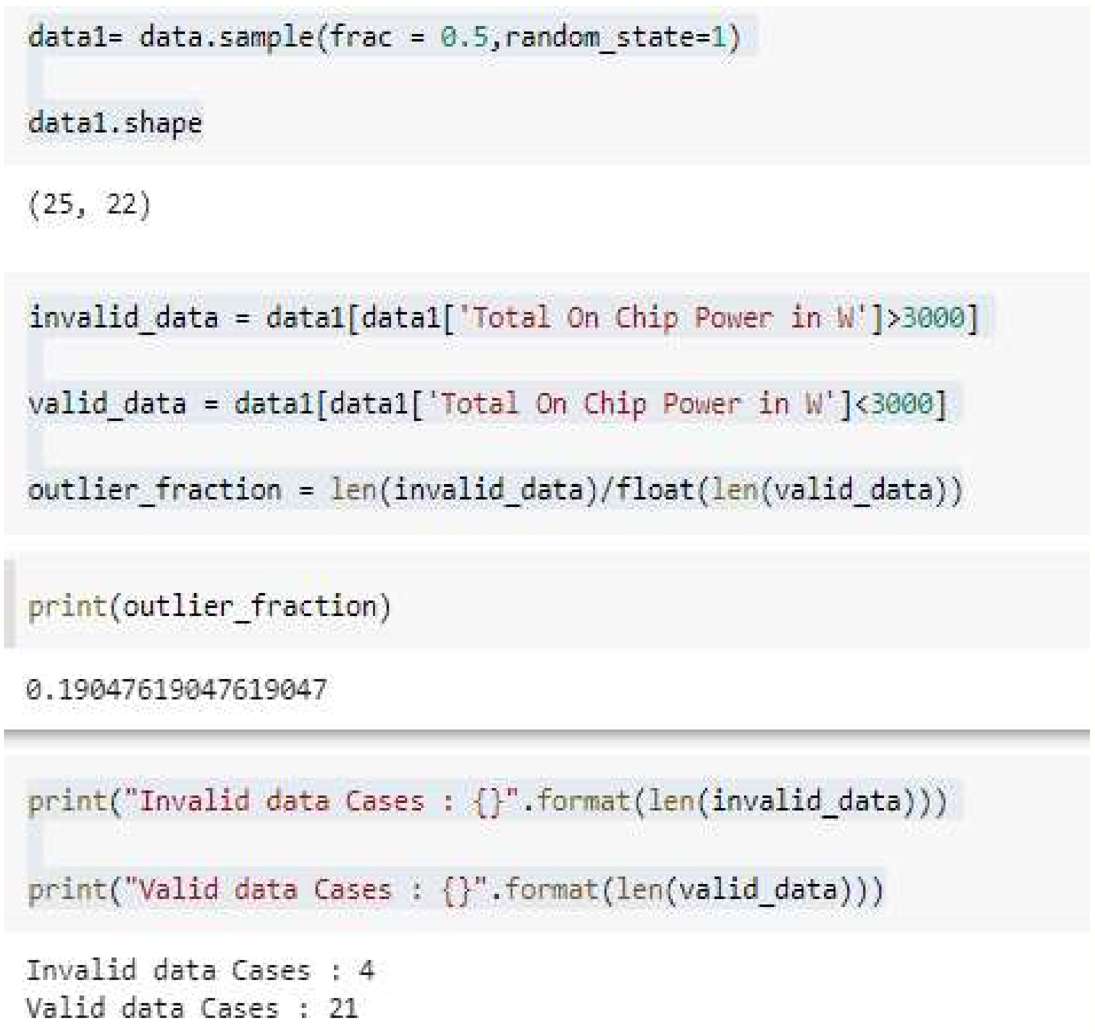
```
    s. no  output Load  ...  I/O power in W  Static Power in W
0       1            0  ...         112.489              0.791
1       2           10  ...         208.180              0.791
2       3           20  ...         303.871              0.791
3       4           30  ...         399.563              0.791
4       5           50  ...         590.496              0.791
5       6           75  ...         830.175              0.791
6       7          100  ...        1069.400              0.791
7       8            0  ...         112.490              0.121
8       9            0  ...         112.489              0.403
9      10            0  ...         112.489              0.791
10     11           20  ...         303.871              0.791
11     12           50  ...         390.946              0.791
12     13           50  ...         390.946              0.791
13     14           50  ...         390.946              0.791
14     15           50  ...         390.946              0.791
15     16           50  ...         390.946              0.791
16     17            0  ...         112.489              0.083
17     18            0  ...         112.489              0.083
18     19            0  ...         112.489              0.083
19     20            0  ...         112.489              0.083
20     21            0  ...         112.489              0.083
21     22            0  ...         112.489              0.083
22     23            0  ...         112.489              0.083
23     24            0  ...         112.489              0.083
24     25            0  ...         112.489              0.083
25     26          150  ...        1547.860              0.693
```

Figure data shape command and printing the data

```
//Plotting the different types of power with respect to output load and
//Junction Temperature
data.plot(x='output Load', y='On chip Dynamic power in W')

data.plot(x='output Load', y='Signal Power in W')

data.plot(x='output Load', y='Logic Power in w')

data.plot(x='output Load', y='I/O power in W')

data.plot(x='output Load', y='Static Power in W')

data.plot(x='Junction Temperature in C', y='On chip Dynamic power in W')

data.plot(x='No. of Board Layers', y='On chip Dynamic power in W')

data.plot(x='Junction Temperature in C', y='On chip Dynamic power in W')

data.plot(x='Junction Temperature in C', y='Signal Power in W')

data.plot(x='Junction Temperature in C', y='Logic Power in w')

data.plot(x='Junction Temperature in C', y='I/O power in W')

data.plot(x='Junction Temperature in C', y='Static Power in W')

data1= data.sample(frac = 0.5,random_state=1)

data1.shape

invalid_data = data1[data1['Total On Chip Power in W']>3000]
```

```
valid_data = data1[data1['Total On Chip Power in W']<3000]

outlier_fraction = len(invalid_data)/float(len(valid_data))

print(outlier_fraction)

print("Invalid data Cases : {}".format(len(invalid_data)))

print("Valid data Cases : {}".format(len(valid_data)))
```

Figure showing output of the above commands



```
data1= data.sample(frac = 0.5,random_state=1)

data1.shape
```

```
(25, 22)
```

```
invalid_data = data1[data1['Total On Chip Power in W']>3000]

valid_data = data1[data1['Total On Chip Power in W']<3000]

outlier_fraction = len(invalid_data)/float(len(valid_data))
```

```
print(outlier_fraction)
```

```
0.19047619047619047
```

```
print("Invalid data Cases : {}".format(len(invalid_data)))

print("Valid data Cases : {}".format(len(valid_data)))
```

```
Invalid data Cases : 4
Valid data Cases : 21
```

Figure output of data1 shape and outlier fraction command

```
//Correlation

// Generating the correalation Matrix

import seaborn as sns

#get correlations of each features in dataset

corrmat = data1.corr()

top_corr_features = corrmat.index

plt.figure(figsize=(20,20))

//plotting the heat map

g=sns.heatmap(data[top_corr_features].corr(),annot=True,cmap="RdYlGn")

//Create independent and Dependent Features

columns = data1.columns.tolist()

//Filter the columns to remove data we do not want

columns = [c for c in columns if c not in ["class"]]

//Store the variable we are predicting

target = "class"

// Define a random state

state = np.random.RandomState(42)

X = data1[columns]

Y = data1[target]

X_outliers = state.uniform(low=0, high=1, size=(X.shape[0], X.shape[1]))

//Print the shapes of X & Y
```

```
print(X.shape)

print(Y.shape)

//Define the outlier detection methods

//Defining all the five Machine Learning Algorithms

classifiers = {

   "Isolation Forest":IsolationForest(n_estimators=100, max_samples=len(X),

                              contamination=outlier_fraction,random_state=state, verbo

se=0),

   "Local Outlier Factor":LocalOutlierFactor(n_neighbors=20, algorithm='auto',

                                   leaf_size=30, metric='minkowski',

                                   p=2, metric_params=None, contamination=outlier_fra

ction),

   "Support Vector Machine":OneClassSVM(kernel='rbf', degree=3, gamma=0.1,n

u=0.05,

                              max_iter=-1),

           "GaussianNB" : GaussianNB(priors= None , var_smoothing=1e-09),

          "KneighborsClassifier": KNeighborsClassifier(algorithm='auto', leaf_size

=30, metric='minkowski',

metric_params=None, n_jobs=None, n_neighbors=5, p=2,

weights='uniform')

}
```

```
n_outliers = len(invalid_data)

for i, (clf_name,clf) in enumerate(classifiers.items()):

 //Fit the data and tag outliers

//Fitting the Machine Learning Algorithms

    if clf_name == "Local Outlier Factor":

        y_pred = clf.fit_predict(X)

        scores_prediction = clf.negative_outlier_factor_

    elif clf_name == "Support Vector Machine":

        clf.fit(X)

        y_pred = clf.predict(X)

    elif clf_name =="GaussianNB":

        clf.fit(X, y_pred)

        y_pred = clf.predict(X)

    elif clf_name =="KneighborsClassifier":

        clf.fit(X, y_pred)

        y_pred = clf.predict(X)

    else:

        clf.fit(X)

        scores_prediction = clf.decision_function(X)

        y_pred = clf.predict(X)
```

//Reshape the prediction values to 0 for Valid transactions , 1 for Fraud trans actions

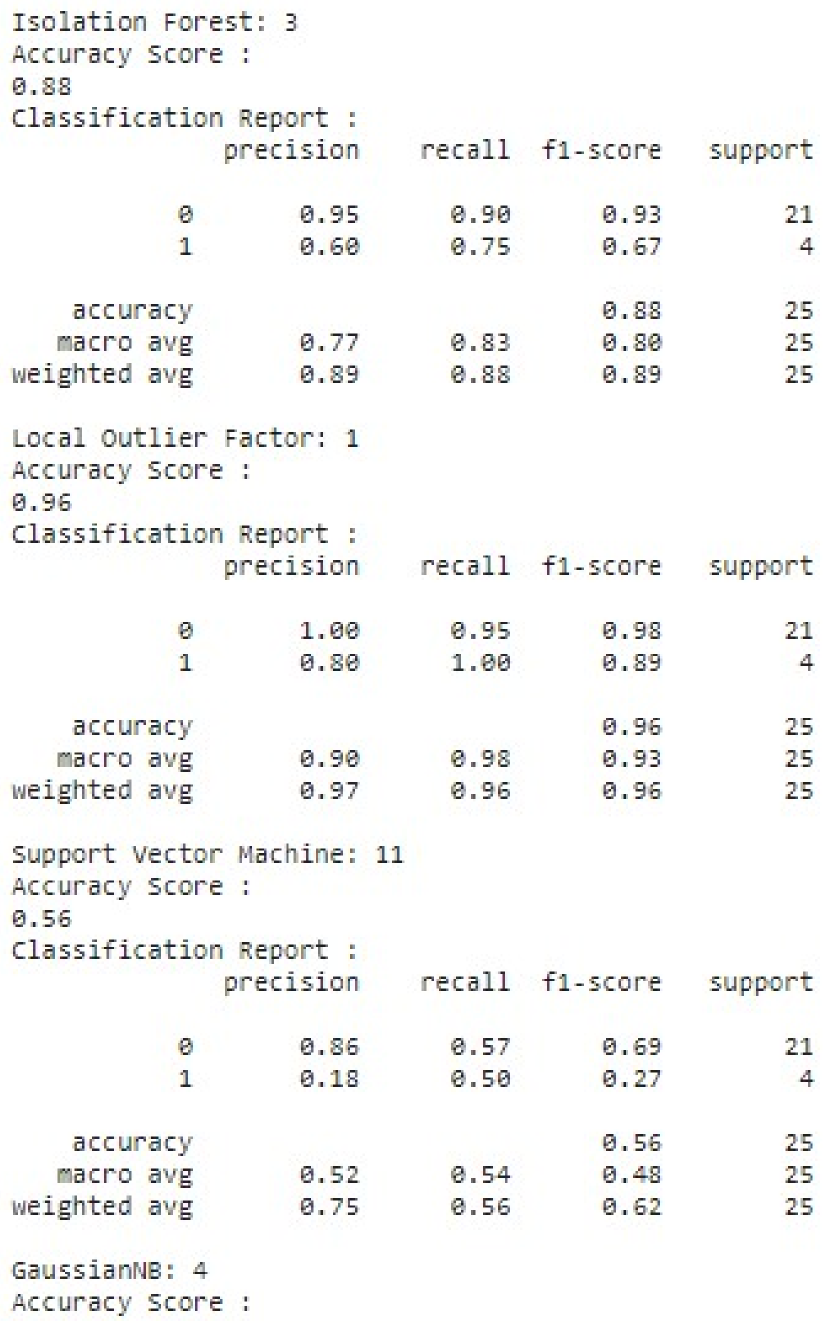
```
y_pred[y_pred == 1] = 0
y_pred[y_pred == -1] = 1
n_errors = (y_pred != Y).sum(
```

//Run Classification Metrics

// Generation of Classification Report

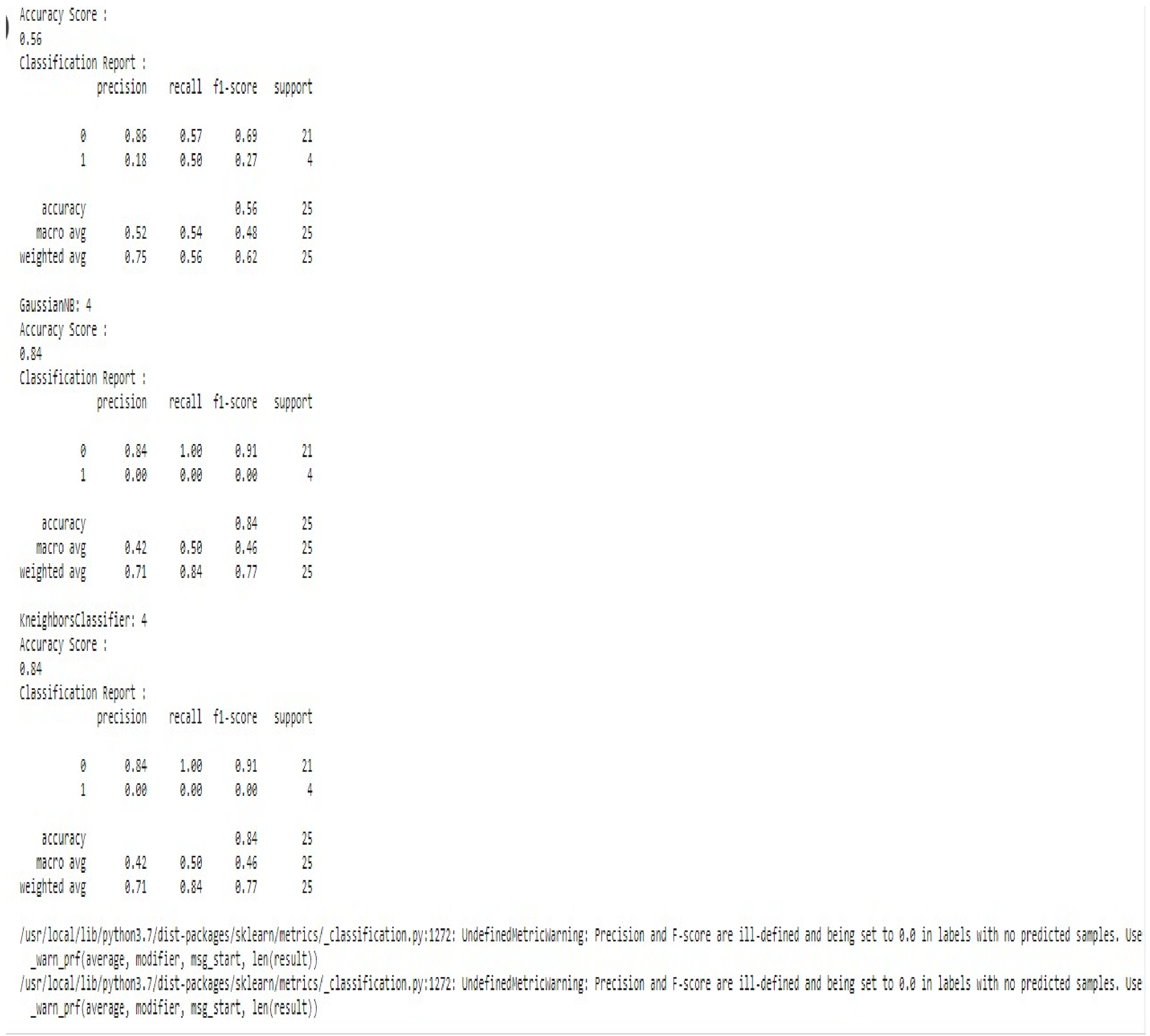
```
print("{}: {}".format(clf_name,n_errors))
print("Accuracy Score :")
print(accuracy_score(Y,y_pred))
print("Classification Report :")
print(classification_report(Y,y_pred))
```

Figure showing output of the classification metric report



```
Isolation Forest: 3
Accuracy Score :
0.88
Classification Report :
              precision    recall  f1-score   support

           0       0.95      0.90      0.93        21
           1       0.60      0.75      0.67         4

    accuracy                           0.88        25
   macro avg       0.77      0.83      0.80        25
weighted avg       0.89      0.88      0.89        25

Local Outlier Factor: 1
Accuracy Score :
0.96
Classification Report :
              precision    recall  f1-score   support

           0       1.00      0.95      0.98        21
           1       0.80      1.00      0.89         4

    accuracy                           0.96        25
   macro avg       0.90      0.98      0.93        25
weighted avg       0.97      0.96      0.96        25

Support Vector Machine: 11
Accuracy Score :
0.56
Classification Report :
              precision    recall  f1-score   support

           0       0.86      0.57      0.69        21
           1       0.18      0.50      0.27         4

    accuracy                           0.56        25
   macro avg       0.52      0.54      0.48        25
weighted avg       0.75      0.56      0.62        25

GaussianNB: 4
Accuracy Score :
```

Figure output of the classification metric report



```
Accuracy Score :
0.56
Classification Report :
              precision    recall  f1-score   support

           0       0.86      0.57      0.69        21
           1       0.18      0.50      0.27         4

    accuracy                           0.56        25
   macro avg       0.52      0.54      0.48        25
weighted avg       0.75      0.56      0.62        25

GaussianNB: 4
Accuracy Score :
0.84
Classification Report :
              precision    recall  f1-score   support

           0       0.84      1.00      0.91        21
           1       0.00      0.00      0.00         4

    accuracy                           0.84        25
   macro avg       0.42      0.50      0.46        25
weighted avg       0.71      0.84      0.77        25

KneighborsClassifier: 4
Accuracy Score :
0.84
Classification Report :
              precision    recall  f1-score   support

           0       0.84      1.00      0.91        21
           1       0.00      0.00      0.00         4

    accuracy                           0.84        25
   macro avg       0.42      0.50      0.46        25
weighted avg       0.71      0.84      0.77        25

/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:1272: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:1272: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use
  _warn_prf(average, modifier, msg_start, len(result))
```

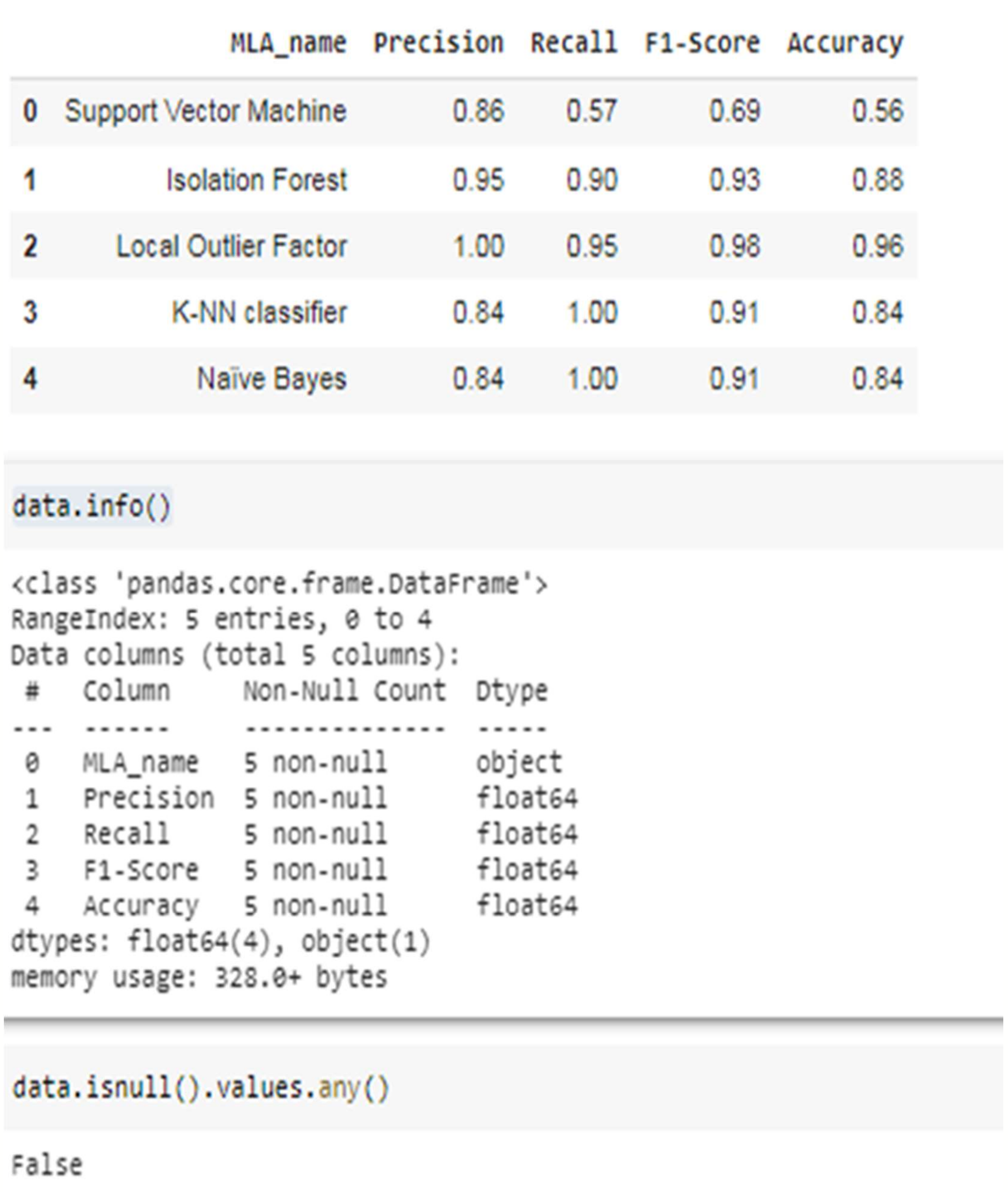
Figure output of the classification metric report

Another code run for Graphical comparison of Machine learning Algorithms.

//Loading libraries

```
import numpy as np

import pandas as pd
import sklearn

import scipy

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.metrics import classification_report,accuracy_score

from sklearn.ensemble import IsolationForest

from sklearn.neighbors import LocalOutlierFactor

from sklearn.naive_bayes import GaussianNB

from sklearn.neighbors import KNeighborsClassifier

from sklearn.svm import OneClassSVM

from pylab import rcParams

rcParams['figure.figsize'] =12,8

RANDOM_SEED = 42

data = pd.read_csv('MLA.csv',sep=',', encoding='latin-1')

data.head(n=10)

data.info()
```

Following Figure shows the dataset for Five machine learning algorithms.



	MLA_name	Precision	Recall	F1-Score	Accuracy
0	Support Vector Machine	0.86	0.57	0.69	0.56
1	Isolation Forest	0.95	0.90	0.93	0.88
2	Local Outlier Factor	1.00	0.95	0.98	0.96
3	K-NN classifier	0.84	1.00	0.91	0.84
4	Naïve Bayes	0.84	1.00	0.91	0.84

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5 entries, 0 to 4
Data columns (total 5 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   MLA_name   5 non-null      object
 1   Precision  5 non-null      float64
 2   Recall     5 non-null      float64
 3   F1-Score   5 non-null      float64
 4   Accuracy   5 non-null      float64
dtypes: float64(4), object(1)
memory usage: 328.0+ bytes
```

```
data.isnull().values.any()
```

```
False
```

Figure dataset for Five machine learning algorithms.

// Plotting the comparison of machine learning algorithm on the basis of their performance metrics

```
data.plot(x ='MLA_name',kind = 'bar', rot=0)

plt.title("comparison")
```

plt.xlabel("MLA_name")

plt.ylabel("Performance metrics")

Following figure showing the output of above snippet code

```
Text(0, 0.5, 'Performance metrics')
```

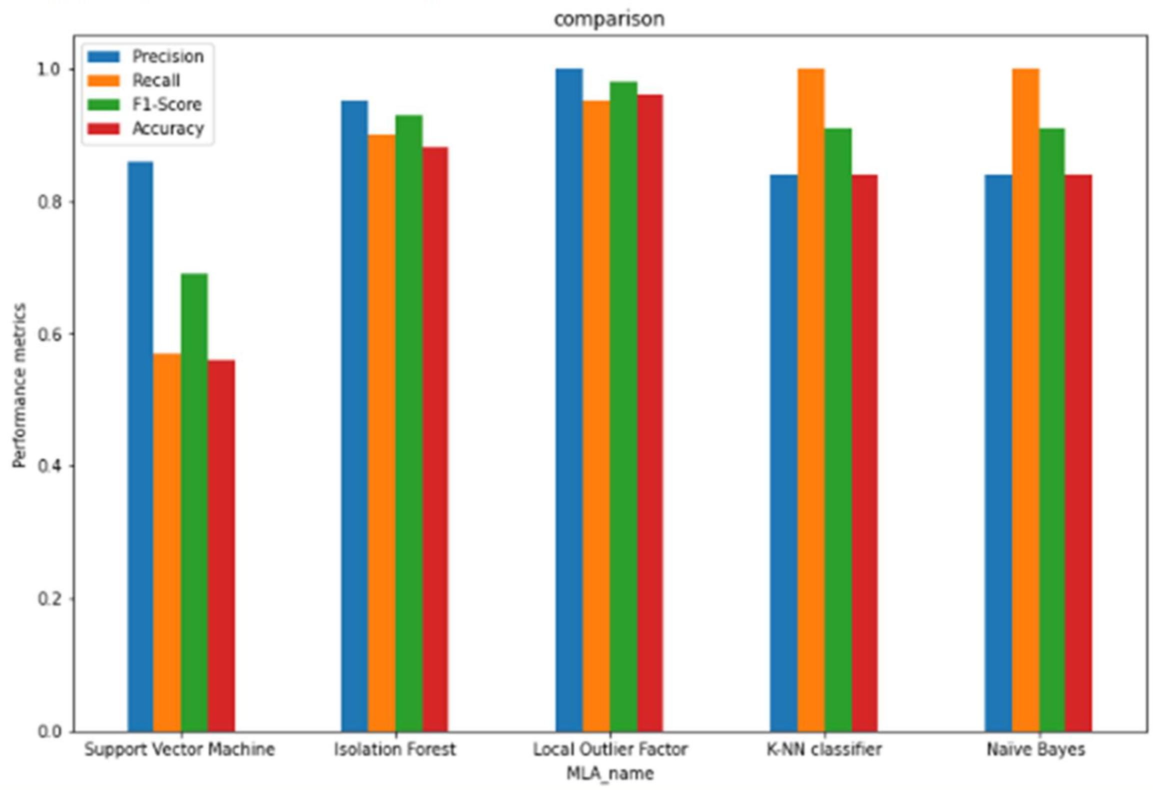


Figure Comparison of Different machine learning algorithms on the basis of their performance metrics.