Master *Radiation and its Effects on MicroElectronics and Photonics Technologies* (RADMEP)

VAL_AI:
AN INTEGRATED DEBUGGER TOOL FOR POST-SILICON VALIDATION

Master Thesis Report

Presented by
Cashmere Joy Dy Ramos

and defended at
University Jean Monnet
11-12 September 2023

Academic Supervisor: Prof. Frédéric Saigné, Université de Montpellier

Host Supervisor: Engr. Vishal Vishal, NXP Semiconductors
Engr. Eric Bourcier, NXP Semiconductors

Jury Committee: Prof. Sylvain Girard – Université Jean Monnet
Dr. Arto Javanainen – Jyväskylän Yliopisto
Prof. Paul Leroux – Katholieke Universiteit Leuven
Prof. Frédéric Saigné – Université de Montpellier

Abstract

As demands for more complex SoCs becomes high due to several technological advancements, the market for these chips is endless and companies are driven to keep up with the changing times by streamlining their top-down process for a more efficient flow. Post-silicon validation is one of the most important steps during System-on-Chip production process as this happens after fabrication to test functionalities of chips in both nominal conditions and varied process, voltage, and temperature conditions. Post-silicon validation debugging is the last line of defense for all possible attacks and bugs. The process involves rigorous amount of iterative and repeated process of opening several files and comparing data from previous tests. Due to the complicated nature of debugging, there is a need to effectively systematize it and automate the current process of digital validation. Moreover, there are many software tools in the development stages and there is also a need to integrate them altogether.

This work first starts with expounding on the current process and tools being used by NXP Semiconductors during post-silicon digital validation. The focus is then driven to automate merging files, visualization of data, and identification of possible causes of failure through a three-step solution: data collection, candidate identification, and problem analysis. The work begins with how the test runs are stored, queried, and shown. Two clustering techniques: kmeans and agglomerative clustering are also discussed. Some dimension reduction techniques like principal component analysis and uniform manifold and approximation projection were also elaborated. These techniques could help development and future work. Application integration is the last step to unify all infant software tools currently in the works under NXP Semiconductors.

Table of Contents

List of Figures

List of Tables

1 Introduction

Nowadays, as the consumer demand for digital devices and services have increased due to more technological advancements, the need for System-on-Chips (SoCs) is also exponentially increasing. As described in [1], these chips are integrated blocks containing several components for memory, timing, control, and interface among others. The market for these is endless -- from mobiles, automotives, IoT to even simple household machines. This further drives leading companies to develop more competitive features like using cloud computing, AI, better GPUs, more reliable security, and many others in a shorter amount of time to keep up with the dynamics of the changing times.

Post-silicon validation (PSV) happens after the System-on-Chips (SoCs) have been fabricated. Several tests are executed to check specification-recommended functionalities under nominal conditions and evaluate hardening for unusual use and fabrication under different process, voltage, and temperature dependent conditions. PSV debugging is one of the most painstaking steps during SoC release as this is the last line of defense for many attacks and bugs should be discovered at this stage. Usually, during verification, most of the functional problems need to be tested, detected, and corrected but some bugs and problems only arise after the silicon has been fabricated.

The process for PSV debugging involves a rigorous process of opening several files and comparing the data the engineer has in previous tests. These are then analyzed for further processing to determine the actual causes of failure. [2] discusses the typical faults being looked out for during PSV like escaped functional errors and electrical faults. Timing errors, speed (frequency), and delays are also tested to make sure the SoC is within the standard specifications as suggested by [3].

Due to the overly complicated iterative process of ensuring the chip is up to par and that it functions as it is supposed to, there is a need to standardize the flow of debugging to ease the constriction caused by more complex designs. This work focuses on providing a systemic approach to post-silicon validation using a web-based application for merging, viewing, filtering, and visualizing data and providing a possible causality of test results. The thesis first discusses the working environment and the current tools ready for use. This is then followed by the scope, purpose, and desired result. The previous works are talked through, and the methods used are expounded on the next part as well as the results followed by the conclusion and suggested future work.

1.1 NXP Semiconductors

NXP Semiconductors is a multi-national company headquartered in Austin, Texas and Eindhoven, Netherlands. It is one of the leading semiconductors companies providing silicon solutions for automotive, mobile, communication and infrastructure, and smart cities and homes. The company has sites all over the world in Americas, Asia-Pacific, and Europe/Middle East. The sites in France are in Caen, Paris-Saclay, Toulouse, and Mougins. The site in Sophia-Antipolis, Mougins found in Figure 1.1.1 is mostly where the research and development for the business line Secure Connected Edge is being housed.

Figure 1.1.1 NXP Semiconductors Mougins Site (Sophia-Antipolis)

Source: Adapted from NXP Semiconductors

The Secure Connected Edge is responsible for the complete top-down solution from analog and digital design, verification, to validation of System-on-Chips (SoCs) produced by the company. Verification is usually done to test the design capabilities and nominal functions before silicon manufacturing and assembly. After this, the designs are sent to fabrication laboratories for production. The validation team then tests the fabricated chip samples that arrive as the last line of defense against bugs and design problems. These are particularly important steps to make sure that the chip is meticulously designed, configured, fabricated, and is functional according to specifications and nominal values. This is also the step wherein the chips are being tested for their extreme capacities.

The digital validation team of the Secure Connected Edge (SCE) is responsible for the final line of post-silicon validation of secure elements, NFCs, and SoCs for distinguished clients. The teams are based in Mougins (France), Hamburg (Germany) and San Diego (US). The team has 25 engineers for all sites working together for final stage delivery.

1.2 Digital Validation Team (SCE)

In a fast-growing world of technology, many new interesting and complex features have popped here and there, and competition arises from different mind-boggling ideas that can automate and support a more efficient way of doing everything. However, with most technologies going digital and autonomous, security is now a big threat and opportunity for most companies and consumers. Nowadays, security plays a key element in determining the business trends in microelectronics and semiconductor markets. It is one of the main factors that most clients look for now in the industry. NXP Semiconductors prides itself on the security, quality, and efficiency of its products. Therefore, the company areas are finely divided into different color-coded physical areas describing the levels of access people have depending on their project involvement. This can also be briefly illustrated by Figure 1.2.1.

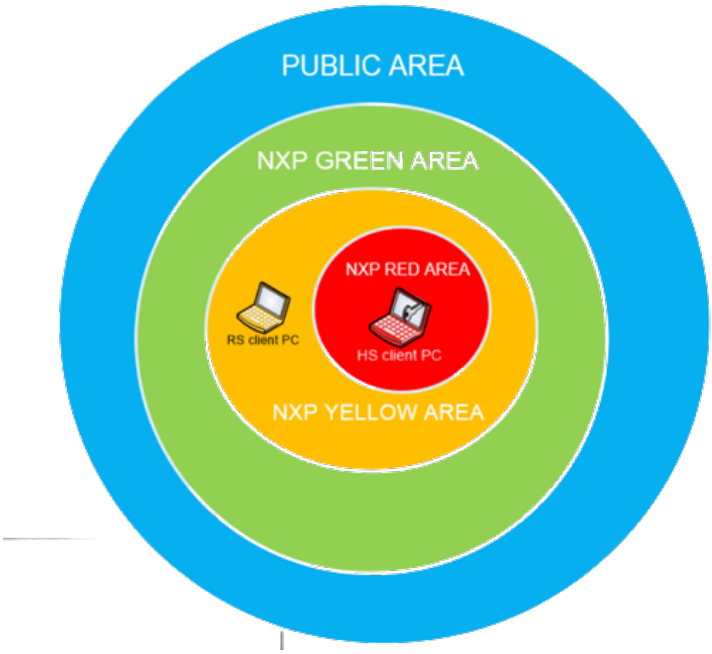

Figure 1.2.1 Physical Area Security Levels of NXP Semiconductors

Source: Adapted from NXP Semiconductors

1. Blue Area – Anywhere outside NXP certified premises.
2. Green Area – General NXP premises inside or outside the fence covered by NXP security.
3. Yellow Area – NXP Security Connected Edge area covered by intrusion protection which serves as detection level.
4. Red Area – Usually, NXP laboratories are located here, and this is surrounded by the yellow one. This serves as protection areas to both logical and physical assets NXP Semiconductors has.

NXP has various levels of logical security depending on the project and device as seen from Figure 1.2.2. Table 1.1 summarizes the different levels of logical security depending on the physical location of the device being used.

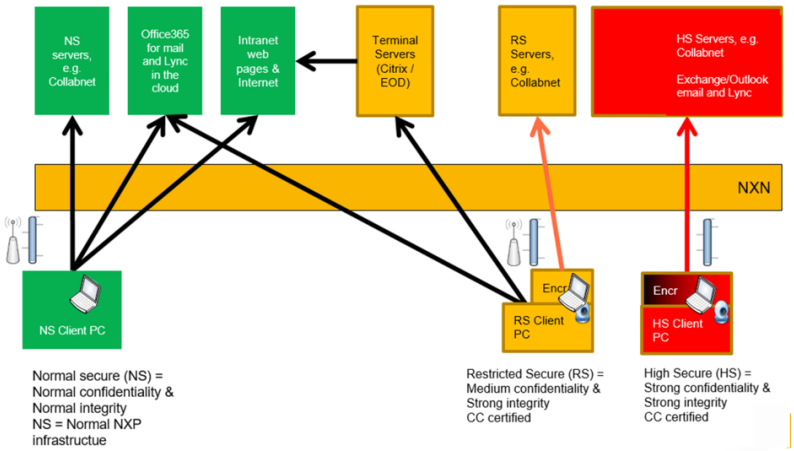

Figure 1.2.2 Logical Security of NXP Semiconductors *Source: Adapted from NXP Semiconductors*

1. Normal Secure – This is for devices working under normal NXP confidentiality and integrity criteria.
2. Restricted Secure – This is for devices working under normal NXP confidentiality and strong integrity criteria. The users should be BSI certified to use devices under restricted secure.
3. High Secure - This is for devices working under strong NXP confidentiality and strong integrity criteria. The users should be BSI certified to use devices under high secure. Usually, the lab machines can be found in these areas.

There are multiple labs located on each site. The labs are placed in red areas using RS domains. The main server of this RS network that the team is using is in Hamburg. In each laboratory for every site, there are RS devices configured to listen to the Cloud Validation server which schedules and automates tasks for each node or device. The RS devices are usually a client computer where the testbench hardware equipment is connected to. The RS server seen in Figure 1.2.3 can be accessed remotely by validation engineers using the personal RS computers to select and schedule tasks. This mechanism makes the tests more efficient in terms of workflow because overnight and weekend test runs can now be performed. Despite this, it is still a major advantage for validation engineers to be physically present in the laboratory for hardware and software debugging for a more effective process.

Nature of work	Green room	Yellow room	Red room
High Secure	✗	✗	✓
Restricted Secure	✗*	✗*	✓
Normal Secure	✓	✓	✓

Table 1.1 Logical vs Physical Security Levels Source: Adapted from NXP Semiconductors

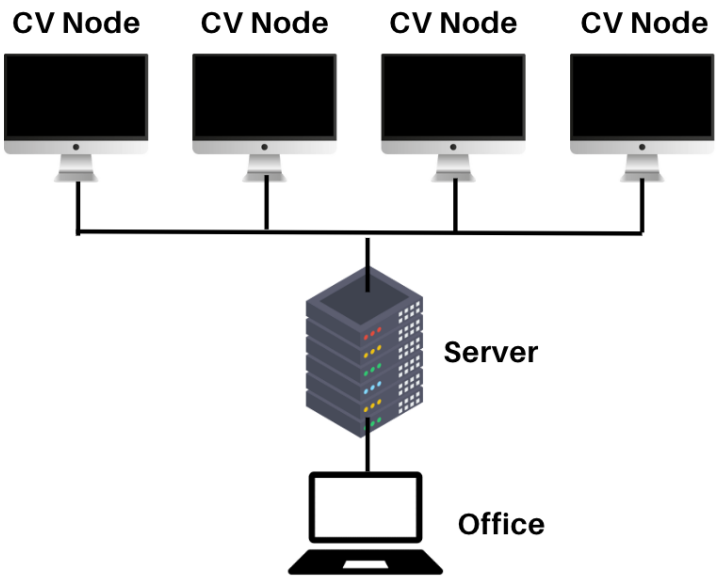

Figure 1.2.3: Laboratory Nodes Configuration

1.3 Validation Flow

The Digital Validation Team of the Secure Connected Edge of NXP semiconductors mainly handles the products contracted to a client. Figure 1.3.1 shows an iterative cycle which describes the flow of the digital validation team.

Figure 1.3.1: Iterative Cycle of Digital Validation Flow

The process starts during preparation wherein the chips have been designed but the silicon is still being fabricated in an outsourced company. This is when the test engineers prepare tests and testbenches using an FPGA to emulate them. At this stage, if the new chip is an evolved daughter chip, the tests can also be based on previous chips that are similar.

After preparation stage, the bring-up happens wherein the silicon arrives, and the initial tests are performed. The samples are first registered to the Validation Application and the default settings are loaded to make sure that any tests prior to this would not affect the current state of the chips. After this, usually a short doing-nothing test (a test which checks the most simplistic functionality whether the application is able to communicate to the target DuT) is executed just to test if the DuT is alive or not.

During evaluation phase, the chip is now being tested by its different basic functionalities without temperature dependence using nominal voltage conditions. The number of samples really depends on the actual project, but it is usually around 10 samples for the batch. The team undergoes test and hardware debugging during this stage.

Next is characterization wherein the chips are being subjected into different voltage and temperature dependent tests. New samples usually arrive with different corners varying the samples from S, N, and F for slow, typical, and fast corners, respectively.

Lastly in the figure, the hotspots are the phase wherein problems arise, and cross-functional teams work together to debug and solve the problems that come up. These hotspots can emerge anytime within the iterative cycle as the chip continues to be validated. Each phase takes a varied amount of time depending on the number of samples and the type of SoC that is being validated.

1.4 Tools Available for the Digital Validation Team

Several tools are available for the digital validation team in NXP Sophia-Antipolis, NXP Hamburg, and NXP San Diego. These tools help automate the validation flow for engineers and give a good overview of pending and executed tasks for the manager.

1.4.1 Validation Application

The validation application is a centralized tool for all validation engineers and engineering managers for various functions. The application written in Java covers eight main things as seen in Figure 1.4.1. This includes NFC Configuration, Test Automation, Equipment, eSE Trimming Control, Register View, eSE Control, and MIO Control. This acts as the main controller of all validation done remotely. While all of these are very useful, some examples of the software capabilities are as follows: Test Automation is used to schedule and execute tests; Equipment View is for making sure that the hardware configuration of the testbench is properly installed; Register View is an interface to dump and view all register values; etc. In general, the validation application is the central tool for validation engineers to begin their tests.

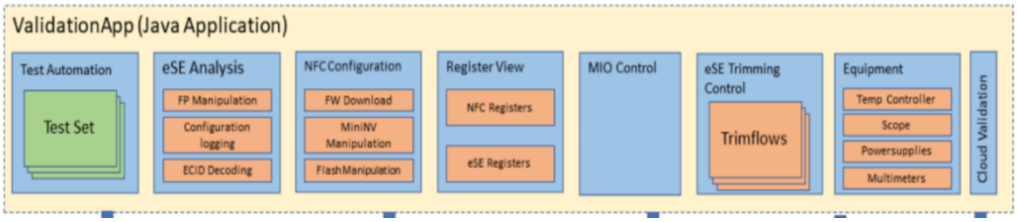

Figure 1.4.1: Validation Application Main Functions

Source: Adapted from NXP Semiconductors

1.4.2 Cloud Validation

This is integrated with the validation application to help schedule tests and tasks remotely using testbench equipment preconfigured and setup in the laboratories. This allows the engineers to preload the test beforehand and run everything from wherever they are.

1.4.3 CV Controller

This tool ensures all software versions of each CV (Cloud Validation) nodes are aligned. This improves the efficiency of the validation engineers by not having to manually log in to every single CV node and check each version one-by-one preventing setup errors.

1.4.4 MongoDB Server

This is the main repository for all of the data gathered after each and every test. It is a non-SQL server database which takes all testruns available for each database and correctly classifies them according to the testset collection they belong. Figure 1.4.2 shows the server schema and how the data is being organized per database. In this study, all data is extracted from the MongoDB server after the test has been executed. The entries are stores as JSON files and the test results for every chip follow a standard architecture for easier storage and access.

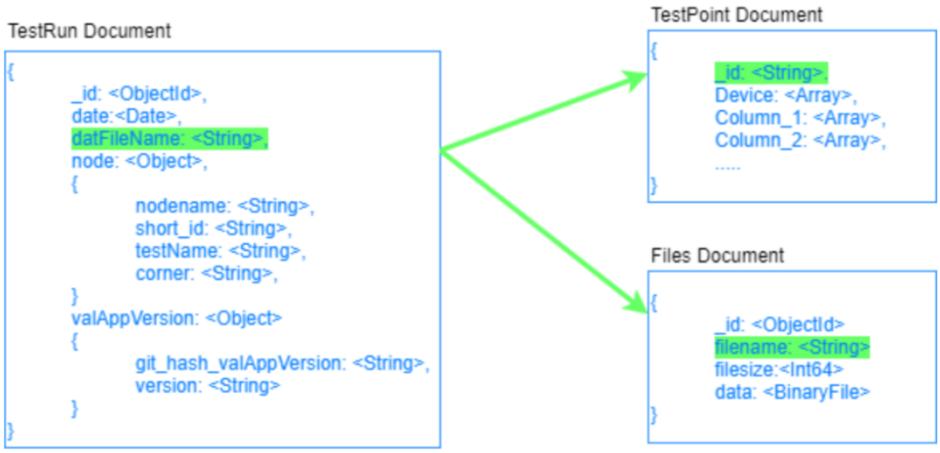

Figure 1.4.2: MongoDB Schema

Source: Adapted from NXP Semiconductors

2 Scope and Goals

NXP Semiconductors has many tools to increase the efficiency and productivity of validation engineers to streamline the process of closing in on SoCs. There are currently a lot of tools as discussed above to automate and schedule the tests remotely or on-site and store the data gathered. There is currently a limited number of tools to help validation engineers view the data to debug parameters more systematically. This work provides a systemic flow in viewing and debugging the SoC's validation results as its first functionality.

There are also a lot of tools currently being developed as other projects using machine learning to increase efficiency. These help the engineers automatically classify if the parameter is essential or not during validation. This work also includes integration using front-end of different microservices currently also being developed within the team. These tools are still undergoing changes and debugging but they are currently being integrated into the software application.

The long-term intent of this project is as follows:

1. To be able to streamline the efficiency of post silicon validation in terms of workflow;
2. To be able to integrate the tools for data analysis that validation team is currently developing

As post-silicon validation becomes more rigorous, engineers usually open and view multiple files at one time to compare data and analyze results. Multiple datfiles stored as excel files from different test runs collated are checked against the log files to better understand the bugs present. For context, one test set could contain several datfiles which contain every test run and one datfile could contain thousands of rows and numerous columns. The specific goals of the project involve the following:

1. To merge and view data from different datfiles in the same test set and database;
2. To properly filter data according to what the user needs and visualize them;
3. To systematically provide context on the possible causalities of failure;
4. To integrate the application with other existing tools and provide the correct inputs and routes to the other microservices that are being constructed;
5. To provide possible ways to further identify the problem which could be implemented in the future;
6. To be code flexible for future development

3 Methodology and Related Work

3.1 Post Silicon Validation

Post-silicon validation is the last step before a System-on-Chip (SoC) goes out for consumer-use in the market. This is highly one of the most important steps in making sure that all of the chips are ready for integration into mobiles, automotives, etc. In this step, several tests are being made to make sure that the product is within the specifications and to quantify the capacity of the chip. [4] discusses the different types of post silicon validation. Functional bug hunting is about testing an SoC in its nominal conditions and checking the functionality of the chip to recover from different attacks. Random instruction testing is usually executed when time is heavily constrained, and few tests could be performed therefore only selecting a specific set of tests to be done. The process of making sure that the memory storage is at good health is called memory subsystem validation, and the input/output synchronism tests are called i/o concurrency.

3.2 Post Silicon Validation Debugger Tools

Many debugging techniques have been suggested to mitigate the bottleneck caused by PSV. [5] mentions the usual approach to PSV and the challenges that may be faced by engineers during this phase. The general process starts by detecting the problem through a series of tests and localizing them. The problem is then analyzed further to identify the root cause. Finally, steps are taken to fix and bypass the bugs and to rectify problems detected. [6] suggested a technique called Symbolic Quick Error Detection (QED) that talks more about short error detection latencies with high coverage and formal analysis to localize the problem area, however, this only focuses on logical bugs.

Several research papers tried to focus on debugger tools that could be useful for post-silicon validation. One of the references this paper was based on was [7] which focuses more visual representation of available data to debug chips during this phase. Several engineers and visual researchers concluded that slicing the study into four distinct stages would provide engineers with insightful data analysis. The study was divided into four parts: Data Collection, Candidate Identification, Problem Identification, and Problem Analysis. They started with making sure to query only the data needed to be selected. The user is also provided with options to filter and only take the specific attributes and values the user needs. For candidate identification, the application attempts to use clustering as a method to determine the correlation of each test case scenario there is and provide visual support to the validation engineers. For problem refinement, there is a user-defined set of inputs, outputs, and targets of which the computation begins but still uses clustering as a method for visual representation. The fourth stage now determines if the input affects the symptoms, so the data classification is now promoted to a target. This way, the debugging becomes more systematic, objective, and visually represented.

As discussed in previous sections, the goal of the tool is to aid in post silicon validation. [5] suggested that it could be more efficient to divide this kind of tool into distinct phases for better efficiency and to create a boundary for each stage of the project by detection, localization, identification, and rectification. For this purpose, the main project is divided into four parts: data collection and merging, candidate identification, problem analysis, and integration. This is seen in Figure 3.2.1.

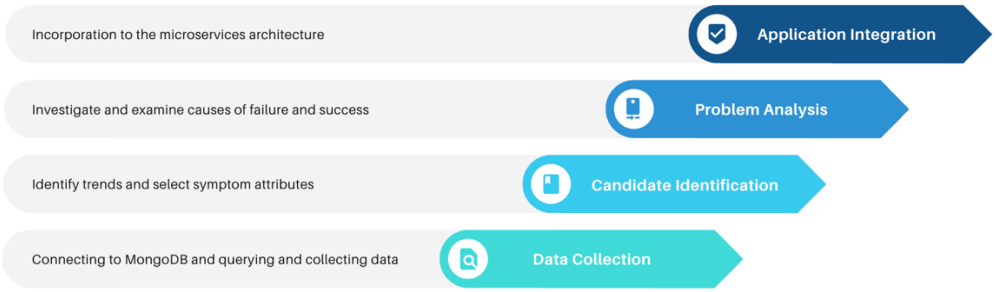

Figure 3.2.1: Iterative Stages of Suggested PSV Debugging

3.3 Python flask

The application is using a web-based application for easier implementation. The application is written in Python for better optimization and computational capabilities. Of the frameworks available: Django, Flask, Pyramid, and Tornado. Flask was chosen because of the wide range of documentation available and ease of use as seen in [8]. HTML and Javascript were used for the front-end part of the code for more convenient handling. The idea is to be able to display and run the entire application using the web browser and handle all computations at the backend Python side whether it is handled on the server or client side. The overall application would be called val_ai as this act as the central front-end tool to all microservices integrated within the application.

Flask is a web framework designed to provide features and tools to create applications using Python [9]. This allows programmers to split the parts into different directories and routes to arrange the code more systematically and efficiently. To use Python flask in this project, the groundwork has been laid on the following prerequisites:

- Python 3 programming environment;
- Understanding of routes, functions, and templates;
- Understanding of HTML concepts with CSS and JavaScript included

3.3.1 Django vs Flask

In this project, two main web frameworks were considered: Flask and Django. Flask uses Model View Controller (MVC) architecture while Django uses Model View Template (MVT) [10]. Figures 3.3.1 and 3.3.2 show the difference between their architectures.

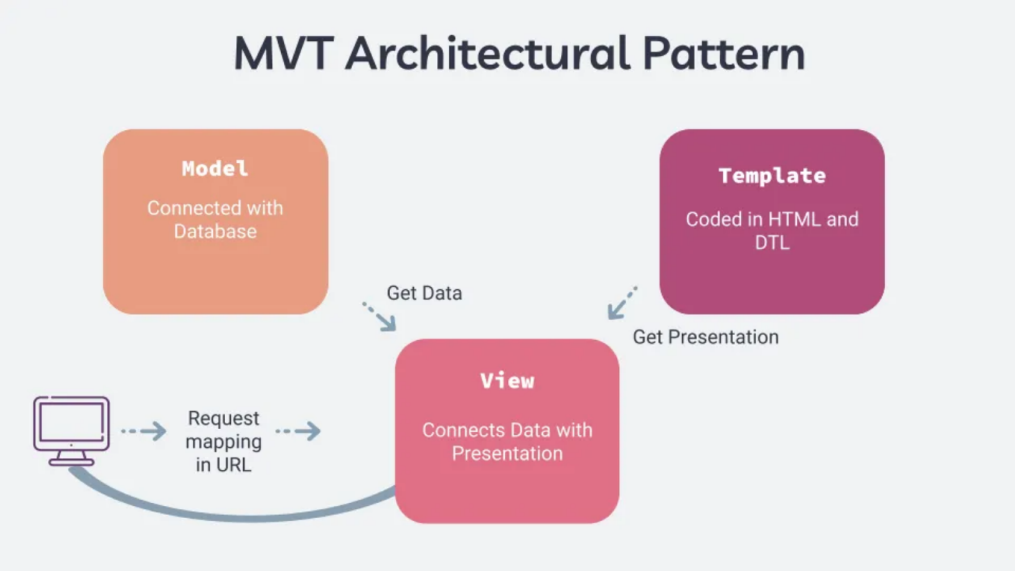

Figure 3.3.1: Model View Template (MVT) architecture

Source: Adapted from [11]

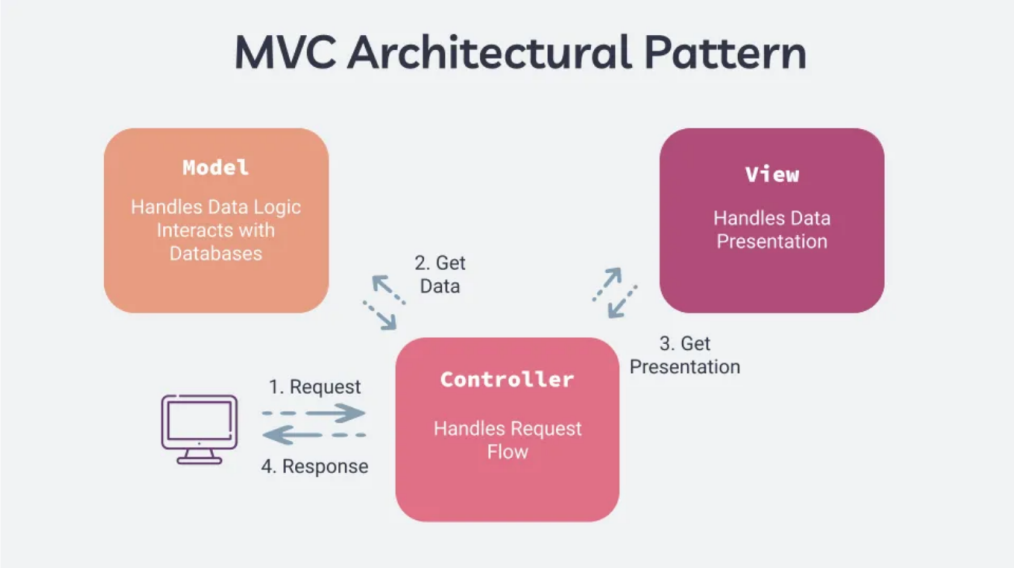

Figure 3.3.2: Model View Controller (MVC) architecture

Source: Adapted from [11]

The MVT architecture used by Django has three components according to [11]:

- Model – This is where the data is stored and logic is executed. This is also connected to the database.
- View – This accepts and sends request depending on what is needed.
- Template – This is where the data is shown.

The MVC architecture used by Flask has a different way of structuring requests and data handling. The three components used are:

- Model – This is also where the data is being stored and logic is executed and performed.
- View – This is where the requested data are being shown and presented.
- Controller – This is the main handler of all flows from and to the model and the view and is responsible for accepting and sending requests and responses.

Python flask was used instead of Django because of its capabilities to distribute the code into different parts for better understanding and further developments. Since building the application is currently in its premature stages, it would be more ideal to use a framework that could further be developed and understood by several people. This web framework also has minimalistic approach that would be better for beginners in building a web application. Table 3.1 summarizes the differences in the implementation of both frameworks.

	Flask	Django
Architecture	Model View Controller	Model View Template
Framework	WSGI Framework	Full-stack web framework
Flexibility	Full flexibility	Feature-packed
ORM Usage	SQLAlchemy	Built-in ORM
Design	Minimalistic	Batteries-included
Working Style	Diversified	Monolithic

Table 3.1: Comparison between Flask and Django

3.3.2 Target application structure

The target application structure can be seen in Figure 3.3.3 The app starts with the main app called val_ai.py containing all possible routes. The Templates folder contain all relevant pages that would be used to present the data requested. The modules folder is a collection of all functions made to be used in each route. The static folder has all the layout information for all of the pages. The downloads and files folders are the repositories for all the cache information and logs downloaded from the server or MongoDB.

```
.
└── flask_app
    ├── app
        ├──main
            ├──val_ai.py
        ├──Templates
```

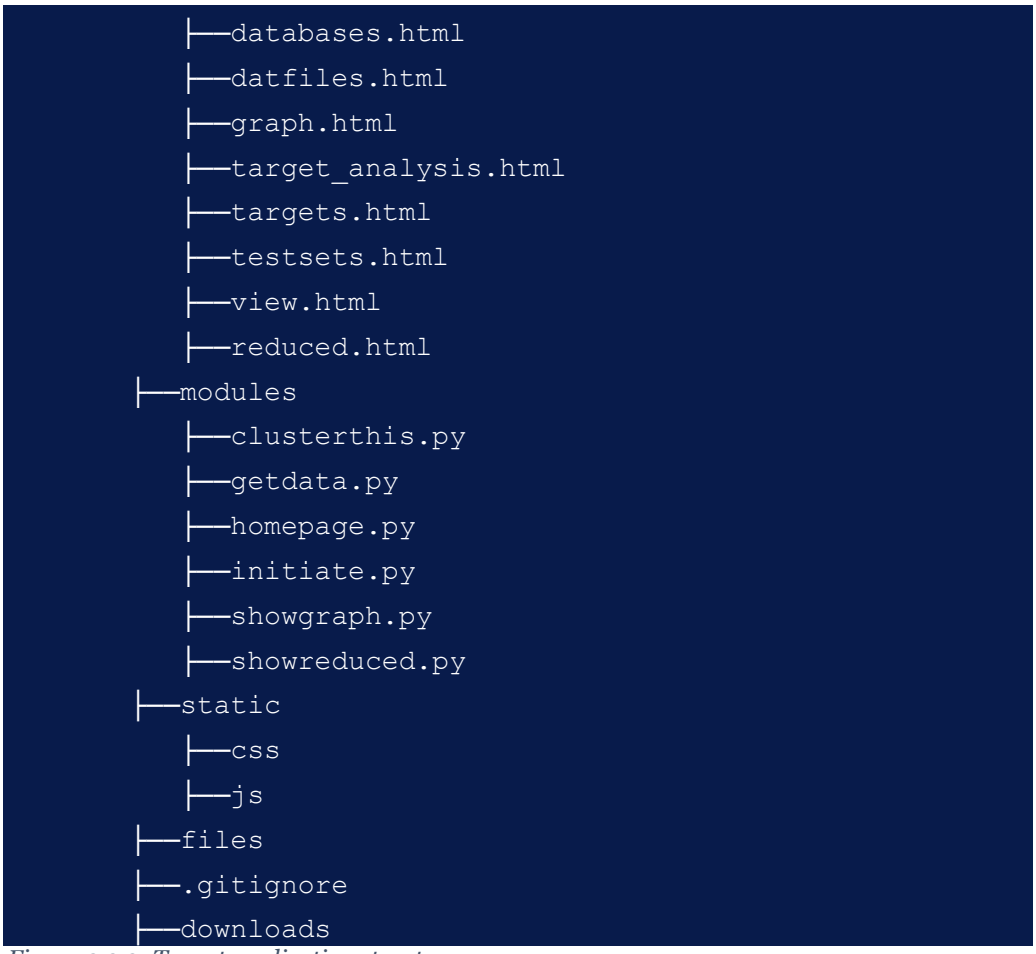

Figure 3.3.3: Target application structure

3.4 Parameter Classification

For easier identification throughout this work, the attributes and parameters are classified into three distinct kinds: input, symptom, and target, as taken from [7].

1. Input Attributes: These are defined as the parameter values being modified by the engineer to check the effect on the SoCs. These are also called shmoo parameters. Common things to vary (but are not limited to) are voltage, temperature, and input pulse widths.
2. Symptom Attributes: These are parameters that are affected as the input changes. Since the goal is to find the exact parameter to look out for and ignore all values that does not have a direct effect on the result of the test and the functionality of the SoC, the symptoms attributes are being considered as an act to localize the problem before finding its root cause.
3. Target Attributes: The parameters responsible for displaying the functionality and the effect of the change in input are classified to this.

3.5 Data Collection

During post-silicon validation, as discussed in other parts, all test results are stored in the MongoDB server. From here, the data is being accessed using a server certificate request and a connection port to the MongoDB server using a general client ID. The data are stored as JSON/BSON, but they are being collected as pandas DataFrames.

The server works by having a different database for each project and project version. In every database, there is a set of TestSets within that project. Contained within each TestSet collection are all of the test runs of that category. These test runs are the swept shmoo parameters for each sample. Figure 3.5.1 shows a diagram of how the database is structured for better querying. Each database has a "TestRuns" folder which contains all of the tests performed within that project with different varied parameters and results. This also contains details for each test that includes the test set name it belongs to. Another folder in the database is "dat_chunks" which contains all of the excel files stored in every run. The "log chunks" folder works the same way but contains all of the text files on which the logs are being stored. Each test set folder now contains all of the information from the runs under that test set.

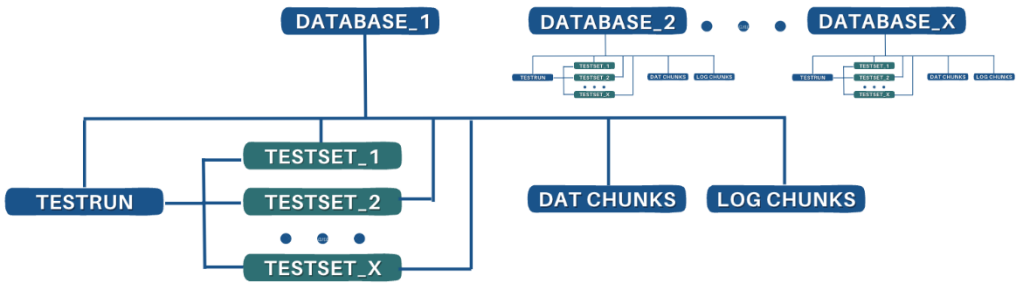

Figure 3.5.1: MongoDB Database Structure

Users are provided with several forms of access to the data depending on which would be the most convenient at any given time. First, all the database collection is shown on the first page with the corresponding list of test sets available per database. Once a particular test set has been chosen by the user, the datfiles available in that test set would be displayed, allowing the user to open which datfiles he or she would need. The users can also choose to filter the exact values for each attribute he would need. The datfiles of which these values can be found are then automatically merged in one window for easier viewing, as integrated from another microservice. The last form of access is when the link of the datfile to be viewed is accessed, if the database, test set, and the datfile name are known.

After defining the datfiles needed by the user, the program automatically downloads the corresponding log files and saves them in a /files folder located in the same repository. This is then called and shown whenever a specific test run or a row from the table is clicked. The program also immediately allows the user to skip to the actual part of the log wherein the test run can be found.

More features were added at this stage for better usage like saving the current table with its current settings. The user can define which rows or columns he or she would only like to see depending on the parameter value and the bin result for each testrun. Here, the user can also find the information regarding the testset and testruns he or she has selected. Figure 3.5.2 shows a flowchart that describes how the algorithm is being implemented.

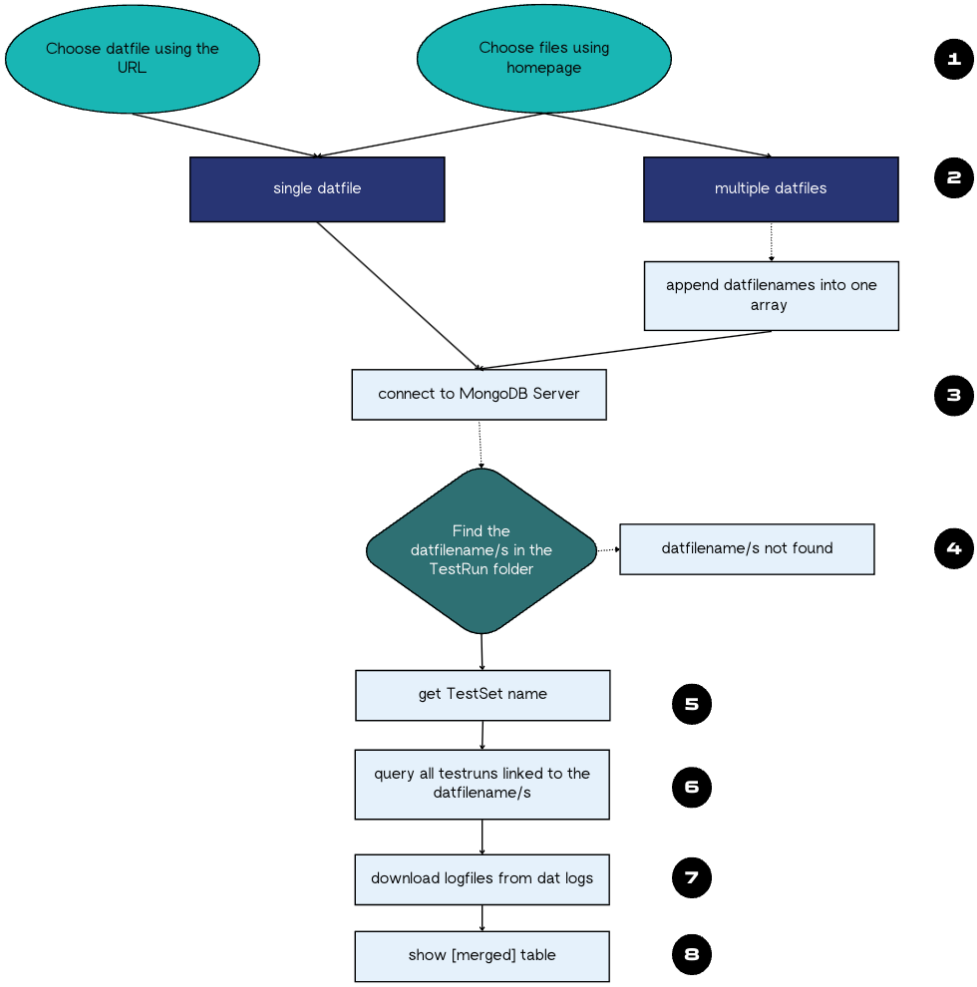

Figure 3.5.2: Flowchart of data collection

3.6 Candidate Identification

During this phase, the data needed had already been collected and merged into one callable variable which was already cached for easier access. The tool takes the user to another page where the test results are displayed as bar graphs representing each attribute against their bin result. This stage aims to create a better visual for engineers to process their data. This is also the stage when the trends for each attribute are being shown and the ones that do not change are not initially considered and are greyed out. The users, however, can still select these attributes if they would like to include them in further reports and analysis. These attributes that could affect the outcome of the tests are input attributes. The ones included in the list of parameters affected by the change in the input are the symptoms. Being included in the list of attributes selected means that the parameter is a symptom attribute. These attributes selected are returned as JSON data back to python for further computation. Figure 3.6.1 shows a rough summary of how the data is being analyzed step-by-step.

Several methods have been used to compute the relational relevance of each test and display them as clusters. There are multiple techniques used for data clustering to provide the user with several ways of choosing which test runs are significant.

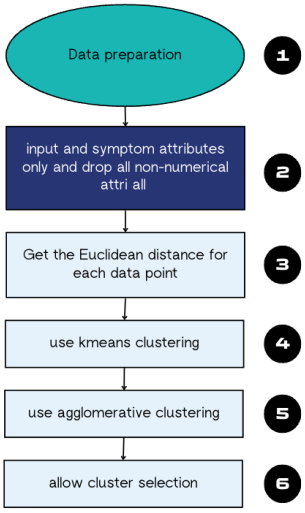

Figure 3.6.1 Candidate Identification Step-by-Step

3.6.1 Data preparation and Euclidean distance

For data preparation, the data points are prepared by getting only the numeric part of each entry. At this stage, only the numerical values are considered to ease the testing for easier computation and data handling. Only the input and symptom attributes are selected to be part of the analysis and all the non-numerical fields are dropped from the table. The goal is to be able to perform quantitative analysis on each test run.

The Euclidean distance is then taken for all remaining attributes for each data point. This acts like data preparation for further clustering methods. The equation below describes how the Euclidean distance was taken for each data point. This would be the main distance metric to determine the similarity and dissimilarity of two data points for further processing. [12] used the same equation below to compute for the Euclidean distance for various data points with multiple components. This is also the equation used in this work using python NumPy library.

$$\text{Euclidean distance(p, q)} = \sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2 + \cdots + (p_x - q_x)^2}$$

Equation 3.6.1 Euclidean distance Source: Adapted from [12]

3.6.2 K-Means Clustering

After considering the only the relevant input and symptom attributes and getting the euclidean distance between each test run, clustering is now performed to further visualize how similar and dissimilar each test run is to another.

At this stage, a clustering is considered due to the capability to allow fast computations given that the post-silicon validation team has an immense amount of test runs to process. This is usually used when the data needs to be processed fast. [13] described the process below:

1. Initialization: This type of clustering starts by randomly selecting centroids depending on the set number of clusters.
2. Assignment: Using these centroids, the test runs are assigned to a centroid whose Euclidean distance is the nearest to them.
3. Update: These are then computed iteratively to find the best centroids for all the clusters formed as optimization by calculation for the mean for each data entry assigned to one cluster.

The algorithm only stops when either (1) the maximum number of iterations has been reached or (2) the centroids are not changing no matter how many times the iterative computations are being executed, whichever comes first [14].

The main problem with this is that not all datasets are suited for this kind of clustering as the data must be spherical or equally sized, as discussed in [15]. Since the test runs in post-silicon validation do not promise results being instantly ready for kmeans clustering, another clustering method is introduced in the next section. For the purpose of testing and the software currently in its premature stages, this clustering method was still implemented side-by-side with the next one.

3.6.3 Agglomerative Clustering

Another type of popular clustering method provides the user the ability to perform clustering without having to initially define the number of clusters needed. This was chosen as the next clustering method because [15] describes this method as more computationally complex than that of kmeans but it requires less inputs from the user and is better in data handling of most datasets especially ones for further analysis. The idea of this method is to initialize each data point as its own cluster and actively iterate them such that they gradually form larger ones. The step-by-step can be seen below as described in [16]:

1. Initialization: The Euclidean distance or the pairwise distance is computed for each test run and each of them is initialized as its own cluster.
2. Choice of linkage: This defines how the distance is being measured from cluster to cluster. The options are as follows:
    a. Single linkage – This defines the distance between the closest points of two clusters. This is the type of linkage used in this work to simplify relational computations and improve latency since there are a lot of data being processed.
    b. Complete linkage – This defines the distance between the farthest points of two clusters.
    c. Average linkage – This defines the average distance between two points in two clusters.
    d. Ward's linkage – This method makes sure that the variance is minimum during merging.
3. Merging: This stage merges the clusters depending on their distances and the choice of linkage. In this stage, the matrix containing the distances is also updated.
4. Dendrogram Construction: This is a tree-like structure of which height defines the similarity and dissimilarity of two data points. The iterative process only stops when either of the two conditions have been met: (1) the pre-defined number of clusters or height of dendrogram have been reached; and (2) when all of the data points are in one single cluster.

The data are then visualized using these two methods of clustering. This way, the validation engineer would have a better relational understanding of the tests executed and results. Both were implemented using the Python sci-kit learn package.

3.7 Problem Analysis

During this stage, the different methods of clustering have already been presented for all test runs. The user should now select which of the clusters are more relevant for further analysis. This cluster number is sent back for further analysis. The software checks which of the symptoms contribute the most to the test result on it having a pass or fail. Several tests have been considered to detect the target parameter that resulted to a failure and the most relevant input parameter that could cause it. These are discussed further below. The user can go back and forth to the main page to make sure that the selected parameters are the most probable. The user can opt to not choose anything and let the default settings detect and predict the problem area.

The Kolmogorov-Smirnov test was first considered as suggested by [7] but this was not implemented anymore due to the capability selecting a particular cluster. This could further be added into future work if during the developmental stage, it would be more beneficial to compare clusters. This section is still added for related work.

Two dimension-reduction techniques are discussed below to process the data points, the use of Uniform Manifold and Approximation Projection (UMAP) and Principal Component Analysis (PCA). These techniques further process the given set of data so that the attributes could be localized into determining if they are the parameter that could be held accountable for the failure. Both still preserve the original structure that the data points have. In this case, PCA was chosen because of its availability in terms of libraries and packages. This method is also more efficient in computing, and it preserves the variance between test run executions. UMAPs would first be discussed followed by PCA.

3.7.1 Kolmogorov-Smirnov test

In [7], the Kolmogorov-Smirnov test was used to detect the correlation between two clusters. This test is a non-parametric statistical test that defines if two samples come from one distribution or if one sample comes from a specific known distribution. This is usually used to define intra-cluster relations and analysis. The empirical cumulative distribution function (ECDF) of the sample data is compared to the cumulative distribution function (CDF) of the function being considered to check the goodness-of-fit [17]. The test works using the basic steps below:

1. Hypothesis synthesis: The sample either follows the reference distribution or not.
2. Test statistic computation: The vertical distance between the ECDF of the sample data and the CDF of the known distribution is calculated.
3. Calculation of critical value and decision making: The p-value is computed, which represents the probability of observing the test statistic when the null hypothesis is true. Depending on the chosen critical value, the test either passes or fails.

In this work, this was not implemented anymore because the user can choose a specific cluster that could be sent for further analysis.

3.7.2 Uniform Manifold and Approximation Projection (UMAP)

Uniform Manifold and Approximation Projection for dimension reduction is a reduction algorithm used by [7] to perform localization of the detected bug. Since the test runs have multiple attributes linking to the result of the test (pass/fail), one workable solution to make everything easier to visualize and process is reducing the dimensions or the number of dependencies the results have to the parameters.

According to [18], this technique can be used to reduce the dimensions if the data has three properties: (1) distribution is uniform in Riemannian manifold, (2) Riemannian metric is constant at least in a specific local area, and (3) the whole manifold is connected locally. This technique was considered because of its effectiveness in dealing with complex and nonlinear structures. It is designed to keep the manifold structure of datasets with high dimensionality. A manifold is a lower-dimensional space that could essentially capture the properties of its high-dimensional counterpart while still preserving the local distances each point has from each other. To implement dimension reduction using UMAPs, the following process is usually followed [19, 20]:

1. Preparation: The test runs are prepared beforehand by eliminating all non-numeric entries for easier computations. The values are also normalized when using UMAPs.
2. Hyperparameter setting: Some of the include the following:
    a. n_neighbors – the number of neighbors considered
    b. min_dist – the minimum distance of points in the lower dimension so that crowding is controlled
    c. n_components – target number of dimensions
3. Nearest neighbor and optimization: Based on the parameter of n_neighbors, a graph is created which captures the local structure of the high dimensions of the data. During optimization, UMAPs keep a cost function that generates the deviation of the pairwise distance between points from higher dimension and lower dimensions.
4. Visualization or Analysis: Depending on the need for using UMAPs, this stage can either be used for further machine learning analysis or to plot the reduced data.
5. Iterations: The entire process can be iterative if the desired embedding has not yet been achieved yet. This also applies if some of the data points are changed throughout the course of using the dimension-reduction technique.

Since UMAPs can be computationally expensive due to computing for the nearest neighbor and creating the cost function, another dimension-reduction technique is discussed in the next section. Other factors considered are also discussed.

3.7.3 Principal Component Analysis (PCA)

The Principal Component Analysis (PCA) is another dimension-reduction technique used to make sure that the maximum variance is attained in the orthogonal axes to create a lower-dimension set of data points that have the same information. This technique is usually used for visualization and feature selection. The process for PCA can be seen below [21]:

1. Preparation: The data points are standardized and made sure that only the numerical values are considered.

2. Mean computation: The mean of all data points is computed to find the center data point.
3. Centering the data: This part subtracts the feature values from the mean so the center data point is at the origin.
4. Covariance matrix: The covariance matrix is computed using the equation below:

$$\mathrm{Cov}(\vec{X},\vec{Y}) \equiv \sigma^2_{\vec{X}\vec{Y}} \equiv \left\langle (\vec{X} - <\vec{X}>)(\vec{Y} - <\vec{Y}>)^T \right\rangle$$

Equation 3.7.1 Covariance Matrix Source: Adapted from [22]

5. Compute and sort eigenvalues and eigenvectors: The eigenvalues represent the variance there is for each eigenvector. These are then sorted in a decreasing order.

6. Principal components and projection: The dot product of the selected eigenvalues and new data matrix is computed and is projected as a lower dimension set of data points.

Overall, PCA was chosen instead of UMAPs because of its computational efficiency. The data being processed by the validation team are standardized and do not require that much power in nonlinear computations. PCA also has a more straightforward approach that would be suitable for early developmental stages. Aside from these, accessibility was also considered since PCA can be implemented using sci-kit learn Python package and UMAPs would require more installation.

The test runs are iteratively processed, and the reduced data would be able to show the possible causes of failure the test runs have in common. The similarities and dissimilarities of the data points are taken into consideration when suggesting a possible attribute to check. In the same way, the cause of success or setup fails could also be checked.

3.8 Application Integration

The program was also integrated into other ongoing tools developed using machine learning techniques to serve as the front-end side of all of the applications being created. The goal is to be the main user interface for all the tools. These are the scope shot analyzer, log summarizer, and log analyzer. All of these are planned to be integrated with val_ai as the intended central node.

3.8.1 Monolithic Architecture

One traditional model to consider is the use of monolithic architecture. Here, the software application is treated as one big service which is independent and self-contained [23]. The idea is to access and deploy the whole tool as one big release. This is beneficial during the development phase as it releases the application from being dependent on other repositories with different ongoing developments. This makes debugging and testing more convenient for the programmers as well as users. The problem with using this architecture in the development of val_ai is that it lacks scalability and flexibility. One change could refresh the whole stack of code already pushed and could affect other services present in the tool. Once the application becomes more functional and complex, this architecture would be inferior in terms of reliability.

3.8.2 Microservices Architecture

In simple terms, this architecture works by coupling the different tools together to make it more seamless. Rather than putting all the tools into one deployable unit, the services are just accessed on one node, but they keep several repositories. Figure X shows the differences of both. [24] describes the services are treated as a collection wherein they are independent of each other even on the database level and the way they communicate with the general API (Application Programming Interface). The benefits of this include continuous deployment and development for each service, highly maintainable and testable, flexible with new technology, and reliability [25]. However, some factors should also be considered like organizational overhead, standardization, and infrastructure costs.

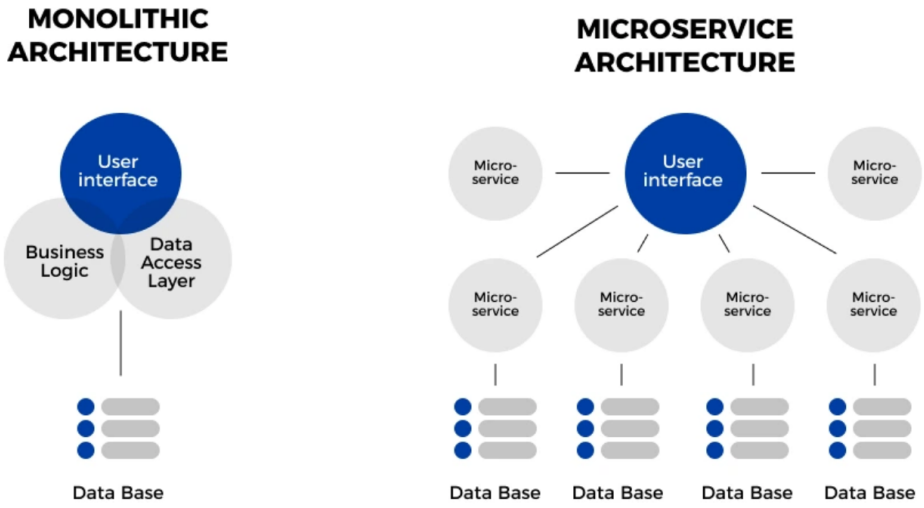

Figure 3.8.1 Monolithic vs Microservices Architecture

Source: Adapted from [26]

Since these application tools are developed by different engineers on its early stages, it has been decided to use microservices architecture for parallel deployment and testing. The standardization and infrastructure costs should not impose a problem early in the timeline as these services offered by the application are still in their infancy stages.

4 Results and Work

The overall application was implemented as planned. In this section, the code snippets will be shown and the functions in these routes will be defined. Some screenshots would be shown to present the data query and presentation.

The application starts off by connecting to the MongoDB server using the code in Annex A. Here, a specific set of username and password is used to log in and access the data of the test results from different projects stored in different database. A randomly-generated secret key is also used to hold sessions using the framework.

4.1 Data Collection

The application starts off with the homepage selector that would display all available databases through get_databases(). Once a specific database is chosen, the corresponding testsets would show using get_test_sets(). The user should choose a specific test set to evaluate and the and the datfiles contained in that test set would be presented in a checklist in get_dat_files(). These are defined by the following application routes as displayed in Figure 4.1.1:

```
@app.route('/')
def get_databases():
    database_details = displaydata()
    return render_template('databases.html',
databases=database_details)

@app.route('/testsets/<database_name>')
def get_test_sets(database_name):
    mongo = MongoDBBaseConnector()
    mongo.set_db(database_name)
    testsets = mongo.test_set_names
    return render_template('testsets.html',
database=database_name, test_sets=testsets)

@app.route("/datfiles/<database>/<test_set>", methods=['GET'])
def get_dat_files(database, test_set):
    session['current_test_set'] = test_set
    session['database'] = database
    dat_files = displaydat(database, test_set)

    return render_template('datfiles.html', database=database,
test_set=test_set, dat_files=dat_files)
```

Figure 4.1.1 Application routes for data

If the user opts to use the URL using the format database/testsetname/datfilename format, the next part would be the first application route.

The next page would then take the user to a table containing the merged testruns of all selected datfiles using these app routes. For this part, two app routes have been created depending on how many datfiles are currently being handled. The URL changes also depending on the specific route selected. The merged testruns being passed from python to the HTML side are dataframes.

```
@app.route("/<database>/<current_test_set>/<datfilename>", methods=['GET',
'POST'])
```

```
def single(database, current_test_set, datfilename):

    global mongoitems
    mongoitems, file_contents, binresult = showtable(database, datfilename)
    print(mongoitems.columns)
    session['binresult'] = binresult

    return render_template('view.html',
row_data=list(mongoitems.values.tolist()), datfilename=datfilename,
items=enumerate(mongoitems.columns),
                           file_cont=file_contents,
                           column_names=mongoitems.columns.values,
stringnames=current_test_set, zip=zip)


@app.route("/<database>/<current_test_set>", methods=['GET', 'POST'])
def multiple(database, current_test_set):

    datfilename = session.get('datfilename')
    query, txtfilename, binresult = getparam(database, datfilename,
current_test_set)
    stringname = current_test_set
    global mongoitems
    mongoitems, file_contents = mergetable(database, txtfilename, stringname,
query)
    session['binresult'] = binresult
    session['current_test_set'] = stringname
    session['database'] = database

    return render_template('view.html',
row_data=list(mongoitems.values.tolist()), datfilename=datfilename,
items=enumerate(mongoitems.columns), file_cont=file_contents,
                           column_names=mongoitems.columns.values,
stringnames=current_test_set, zip=zip)
```

Figure 4.1.2 Routing Depending on the Datfiles

The functions showtable() and getparam() are used to access the MongoDB server and get all necessary data to be able to show the testruns. In addition to that, the function mergetable() appends all selected datfiles and logfiles into a variable for easier merging.

4.2 Candidate Identification

The next page would lead to the candidate identification wherein the attributes are shown as bar graphs and histograms to show trends and bins. The selected attributes are now sent back to the next app route for further processing.

```
@app.route("/<database>/<current_test_set>/<datfilename>/analyze",
methods=['GET', 'POST'])
def analyze(database, current_test_set, datfilename):

    binresult = session.get('binresult')
    image_data = showtotalbin(binresult)
    global mongoitems
    histograms, merged, data_combined, min_values, max_values, std_values =
showinputbin(mongoitems)
    data_img, data_img_e, columns, unique_values_dict = showhist(merged,
data_combined)

    return render_template('graph.html', image_data=image_data,
columns=columns, data_img=data_img,
histograms=enumerate(histograms),data_img_e=data_img_e, min_values=min_values,
max_values=max_values, std_values=std_values,
unique_values_dict=unique_values_dict)
```

Figure 4.2.1 Analysis Route

This app route shows several clusters of data depending on the chosen attributes of the user. The clusters are shown in agglomerative, kmeans, and pca.

```
@app.route("/target_clustered", methods=['GET', 'POST'])
def clustering():
    clusters_dict = session.get('clusters')
    clusters = pd.DataFrame(clusters_dict['data'],
columns=clusters_dict['columns'])
    clustered = showeuclidean(clusters)
    kmeans_clusters = showkmeans(clusters)
    agg_clusters = showagglo(clusters)
    pca_clusters = showpca(clusters)

    return render_template('target_analysis.html', clustered=clustered,
kmeans_clusters=kmeans_clusters, agg_clusters=agg_clusters,
pca_clusters=pca_clusters)
```

Figure 4.2.2 Clustering Route

4.3 Problem Analysis

During this part, the cluster selected by the user would be sent back to Python. The user would be able to go back and forth to change this parameter. The function is used as a dimension reduction technique to further process test runs and data points. This is shown in Figure 4.3.1

```
@app.route("/reduced_dimension", methods=['POST'])
def selected_cluster_data():
   clusters_dict = session.get('clusters')
   clusters = pd.DataFrame(clusters_dict['data'],
columns=clusters_dict['columns'])
   cluster_label = int(request.json['cluster_label'])
   reduced = showreduced(clusters, cluster_label)

   return render_template('reduced.html', reduced=reduced)
```

Figure 4.3.1 Problem analysis route

4.4 Application Integration

The val_ai application was integrated with the validation analyzer tool currently being developed under NXP Semiconductors. Microservices was used as architecture during integration as discussed in previous sections. The machine learning services integrated to the application are: log analyzer, scope shot analyzer, and many more services currently being developed.

5 Conclusion

The val_ai application was implemented according to the specifications within the defined goals. The data can be collected and merged with the filtering option for engineers to take advantage of. The visuals of the data are also improved with more interactive features that will allow users to select only the tests. The application also checks the parameters which could most likely affect the SoCs.

The frameworks and algorithms chosen and used supported the necessary functionalities of the application. The libraries which were initially chosen were carefully considered depending on the use of the actual processing, documentation available, and accessibility.

The goals and specifications were achieved within a brief period. Val_ai was successfully integrated into the validation analyzer application and acts as a general interface for all microservices currently being developed. The application has several routes that would point to various machine learning programs.

Although the application testing was cut short due to the limited time and other microservices are still in the works of being developed and integrated to achieve full serviceability, the debugger tool was able to help systematize and automate some repetitive tasks to ease process flow.

Some minor areas for improvement include caching of data for ease of access and server-side computing to improve delay time and latency. The data handling and uniformity of all entries are also one of the few things that could make the processing better for both client and server sides.

6 Future Work

The tool developed is currently being integrated into the validation-analyzer application. The val_ai application would be helpful in providing a front-end solution for the software services using machine learning during debugging. This also would provide more visualization and ease of access. The possibilities for further development include other tools that could provide a one-stop-shop for all available tools within the digital validation team.

Since the whole integrated application is still in its infancy and development stages, the next steps would be to reproduce the data from previous tests and find out if the same failures and causes would be realized. This step is vital to future tests to ensure the application is useful in succeeding projects and tests.

Since machine learning models are currently being constructed to process the results and data from several sources like log files, datfiles, and scopeshots, it would also be interesting to invest in cloud computing to make everything -- from storage, tools, server, etc. -- faster and more accessible as seen in Figure 4.4.1.

Further front-end improvements could also be very helpful in making the application easier to use and learn for even new engineers joining the validation industry. More features could also be added to automate the tasks from the tests itself to the result requiring minimal effort from the engineers. The SoCs could then be checked by the engineers only as a precaution and as a last line of defense, but the software could handle more computing power and automation.

The future is endless for machine control especially now that remote work is available and widespread. Nothing can ever replace in-person debugging but implementing these things could ease up the ever-changing and fast-growing industry of System-on-Chips especially with the market constantly getting more demand for seamless integration of device parts.

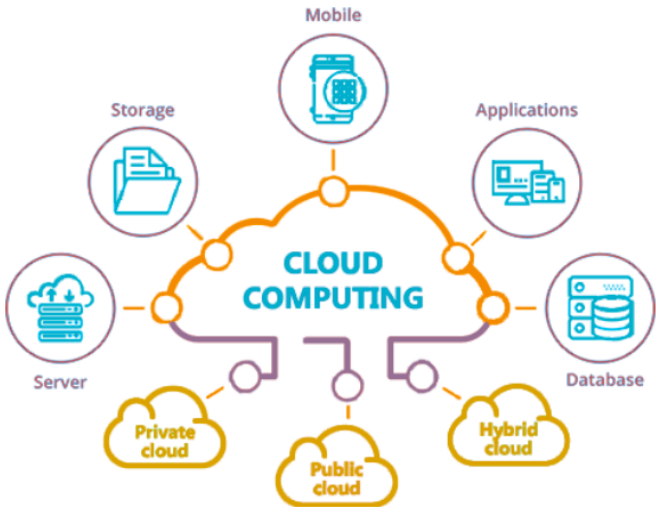

Figure 4.4.1 Cloud Computing

Source: Adapted from [18]

7 References

[1] "System on Chip Market: Global Industry Analysis and Forecast (2023-2029)," *MAXIMIZE MARKET RESEARCH*. Available: https://www.maximizemarketresearch.com/market-report/global-system-on-chip-soc-market/52751/?fbclid=IwAR111WwZwvL6hZa3cKBxOHiU-kAgTy8_0_N3R-hG-DUxdEdeJ5xnoP-tafE. [Accessed: Aug. 21, 2023]

[2] P. Mishra, R. Morad, A. Ziv, and S. Ray, "Post-Silicon Validation in the SoC Era: A Tutorial Introduction," *IEEE Design & Test*, vol. 34, no. 3, pp. 68–92, Jun. 2017, doi: https://doi.org/10.1109/mdat.2017.2691348

[3] S. Ray and A. Sinha, "Synergies Between Delay Test and Post-silicon Speed Path Validation: A Tutorial Introduction." Available: http://sandip.ece.ufl.edu/publications/ets21.pdf. [Accessed: May 21, 2023]

[4] Electronic Specifier, "5 types of post-silicon validation and why they matter," *www.electronicspecifier.com*. Available: https://www.electronicspecifier.com/products/test-and-measurement/5-types-of-post-silicon-validation-and-why-they-matter. [Accessed: Jul. 21, 2023]

[5] S. Mitra, S. Seshia, and N. Nicolici, "Post-Silicon Validation Opportunities, Challenges and Recent Advances." Available: https://people.eecs.berkeley.edu/~sseshia/pubdir/postSi-dac10.pdf. [Accessed: Jun. 18, 2023]

[6] D. Lin, E. Singh, C. Barrett, and S. Mitra, "A Structured Approach to Post-Silicon Validation and Debug Using Symbolic Quick Error Detection." Available: http://theory.stanford.edu/~barrett/pubs/LSB+.pdf. [Accessed: Jun. 08, 2023]

[7] A. Lalama *et al.*, "Interactive Analysis of Post-Silicon Validation Data," *IEEE Xplore*, Oct. 01, 2022. doi: https://doi.org/10.1109/TestVis57757.2022.00007. Available: https://ieeexplore.ieee.org/document/9978548. [Accessed: Apr. 12, 2023]

[8] Flask, "Welcome to Flask — Flask Documentation (2.3.x)," *flask.palletsprojects.com*. Available: https://flask.palletsprojects.com/en/2.3.x/. [Accessed: Apr. 03, 2023]

[9] "How To Structure a Large Flask Application with Flask Blueprints and Flask-SQLAlchemy | DigitalOcean," *www.digitalocean.com*. Available: https://www.digitalocean.com/community/tutorials/how-to-structure-a-large-flask-application-with-flask-blueprints-and-flask-sqlalchemy. [Accessed: Apr. 15, 2023]

[10] DataFlair Team, "Flask vs Django- The Hot Debate of Python Development Section - DataFlair," *DataFlair*, Sep. 07, 2019. Available: https://data-flair.training/blogs/flask-vs-django/. [Accessed: May 08, 2023]

[11] T. Chaudhari, "MVC vs MVT Architectural Pattern," *GDSC UMIT*, Sep. 28, 2021. Available: https://medium.com/dsc-umit/mvc-vs-mvt-architectural-pattern-d306a56dce55. [Accessed: Jun. 27, 2023]

[12] Cloudflare, "Computing Euclidean distance on 144 dimensions," *The Cloudflare Blog*, Dec. 18, 2020. Available: https://blog.cloudflare.com/computing-euclidean-distance-on-144-dimensions/. [Accessed: Jul. 23, 2023]

[13] A. Jhunjhunwala, “A continuously updating k-means algorithm,” *Medium*, Feb. 28, 2019. Available: https://towardsdatascience.com/a-continuously-updating-k-means-algorithm-89584ca7ee63. [Accessed: Jun. 18, 2023]

[14] P. Sharma, “The Ultimate Guide to K-Means Clustering: Definition, Methods and Applications,” *Analytics Vidhya*, Aug. 19, 2019. Available: https://www.analyticsvidhya.com/blog/2019/08/comprehensive-guide-k-means-clustering/#:~:text=We%20can%20stop%20the%20algorithm. [Accessed: Jun. 28, 2023]

[15] “A Comparison of KMeans and Agglomerative Clustering Algorithms for Data Analysis and Pattern Recognition,” *www.linkedin.com*. Available: https://www.linkedin.com/pulse/comparison-kmeans-agglomerative-clustering-algorithms-jagarlapoodi/. [Accessed: Apr. 30, 2023]

[16] C. Y. Wijaya, “Breaking down the agglomerative clustering process,” *Medium*, Feb. 14, 2021. Available: https://towardsdatascience.com/breaking-down-the-agglomerative-clustering-process-1c367f74c7c2. [Accessed: Jul. 01, 2023]

[17] V. Trevisan, “Comparing sample distributions with the Kolmogorov-Smirnov (KS) test,” *Medium*, Mar. 24, 2022. Available: https://towardsdatascience.com/comparing-sample-distributions-with-the-kolmogorov-smirnov-ks-test-a2292ad6fee5. [Accessed: Jul. 01, 2023]

[18] “UMAP: Uniform Manifold Approximation and Projection for Dimension Reduction — umap 0.5 documentation,” *umap-learn.readthedocs.io*. Available: https://umap-learn.readthedocs.io/en/latest/. [Accessed: Jul. 02, 2023]

[19] T. Sainburg, L. McInnes, and T. Q. Gentner, “Parametric UMAP Embeddings for Representation and Semisupervised Learning,” *Neural Computation*, pp. 1–27, Aug. 2021, doi: https://doi.org/10.1162/neco_a_01434

[20] “How UMAP Works — umap 0.5 documentation,” *umap-learn.readthedocs.io*. Available: https://umap-learn.readthedocs.io/en/latest/how_umap_works.html. [Accessed: Jul. 01, 2023]

[21] A. Dutt, “A Step By Step Implementation of Principal Component Analysis,” *Medium*, Oct. 18, 2021. Available: https://towardsdatascience.com/a-step-by-step-implementation-of-principal-component-analysis-5520cc6cd598. [Accessed: Jun. 28, 2023]

[22] “Ph 21.5: Covariance and Principal Component Analysis (PCA),” Aug. 2023. Available: http://pmaweb.caltech.edu/~physlab/lab_21_current/Ph21_5_Covariance_PCA.pdf

[23] Atlassian, “Microservices vs. monolithic architecture,” *Atlassian*. Available: https://www.atlassian.com/microservices/microservices-architecture/microservices-vs-monolith#:~:text=A%20monolithic%20architecture%20is%20a. [Accessed: Aug. 01, 2023]

[24] Google Cloud, “What Is Microservices Architecture?,” *Google Cloud*. Available: https://cloud.google.com/learn/what-is-microservices-architecture#section-1. [Accessed: Jul. 24, 2023]

[25] F. Omar, “Microservices vs Monolith: A Detailed Comparison (Prime Video as an example),” *Medium*, May 13, 2023. Available:

https://medium.com/@fariss.omar.ensi/microservices-vs-monolith-a-detailed-comparison-prime-video-as-an-example-acf58ad30aa6. [Accessed: Jun. 25, 2023]

[26] A. Davis, “The Pros and Cons of a Monolithic Application Vs. Microservices,” *www.openlegacy.com*, Oct. 05, 2022. Available: https://www.openlegacy.com/blog/monolithic-application. [Accessed: Aug. 17, 2023]

[27] A. Narayan, “Know about Cloud Computing Architecture,” *Knoldus Blogs*, Jul. 05, 2021. Available: https://blog.knoldus.com/know-about-cloud-computing-architecture/. [Accessed: Aug. 10, 2023]