Touchless HW/FW test automation, fault tolerant execution with system death level detection and recovery

Major Project Report

Submitted in partial fulfillment of the requirements

for the degree of

Master of Technology

in

Electronics & Communication Engineering

(Embedded Systems)

By

Sharad Thakur

(15MECE27)

Electronics & Communication Engineering Department

Institute of Technology-Nirma University

Ahmedabad-382 481

Dec 2016

Touchless HW/FW test automation, fault tolerant execution with system death level detection and recovery

Major Project Report

Submitted in partial fulfillment of the requirements

for the degree of

Master of Technology

in

Electronics & Communication Engineering

(Embedded Systems)

By

Sharad Thakur

(15mece27)

Under the guidance of

External Project Guide:

Mr. Kishore Mottadi

Software Engineering Manager,CSS-BDV,

Intel Technology India Pvt. Ltd.,

Bangalore.

Internal Project Guide:

Dr. N. P. Gajjar

Program coordinator, Embedded systems,

Institute of Technology,

Nirma University, Ahmedabad.

Electronics & Communication Engineering Department

Institute of Technology-Nirma University

Ahmedabad-382 481

Dec-2016

Declaration

This is to certify that

a. The thesis comprises my original work towards the degree of Master of Technology in Embedded Systems at Nirma University and has not been submitted elsewhere for a degree.

b. Due acknowledgment has been made in the text to all other material used.

- Sharad Thakur

15MECE27

Disclaimer

"The content of this paper does not represent the technology,opinions,beliefs, or positions of Intel Technology India Pvt. Ltd.,its employees,vendors, customers, or associates."

Certificate

This is to certify that the Major Project entitled **"Touchless HW/FW test automation, fault tolerant execution with system death level detection and recovery"** submitted by **Sharad Thakur (15mece27)**, towards the partial fulfillment of the requirements for the degree of Master of Technology in Embedded Systems, Nirma University, Ahmedabad is the record of work carried out by him under our supervision and guidance. In our opinion, the submitted work has reached a level required for being accepted for examination.The results embodied in this major project, to the best of our knowledge,haven't been submitted to any other university or institution for award of any degree or diploma.

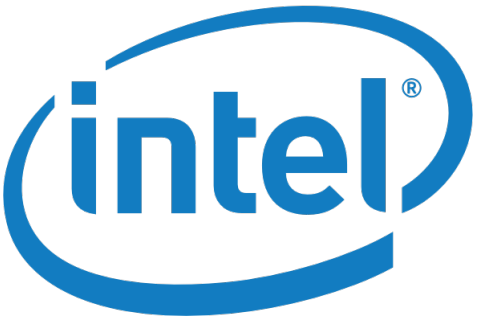
Date: Place: Ahmedabad

Dr. N. P. Gajjar
Internal Guide
Nirma university

Program Coordinator
Nirma university

Dr. D.K.Kothari
Head of department, EC
Nirma university

Dr. Alka Mahajan
Director, ITNU
Nirma university



Certificate

This is to certify that the Major Project entitled **"Touch less HW/FW test automation, fault tolerant execution with system death level detection and recovery"** submitted by **Sharad Thakur(15MECE27)**, towards the partial fulfillment of the requirements for the degree of Master of Technology in Embedded Systems, Nirma University, Ahmedabad is the record of work carried out by him under our supervision and guidance. In our opinion, the submitted work has reached a level required for being accepted for examination.

Mr. Kishore Mottadi
Software Engineering Manager,CSS-BDV
Intel Technology India Pvt. Ltd.
Bangalore

Ms. Sneha Pingle
Software Engineer,CSS-BDV
Intel Technology India Pvt. Ltd.
Bangalore

Acknowledgment

I would like to express my gratitude and sincere thanks to **Dr. D.K.kothari**, Head of Electronics Department, and **Dr. N.P.Gajjar**, PG Coordinator of M.Tech Embedded Systems program for allowing me to undertake this thesis work and for his guidelines during the review process.

I take this opportunity to express my profound gratitude and deep regards to **Dr. N. P. Gajjar**, guide of my major project for his exemplary guidance, monitoring and constant encouragement throughout the course of this thesis. The blessing, help and guidance given by him time to time shall carry me a long way in the journey of life on which I am about to embark.

I would take this opportunity to express a deep sense of gratitude to **Mr. Mottadi,Kishore**,Engineering Manager,Intel Technology India Pvt. Ltd. for his cordial support, constant supervision as well as for providing valuable information regarding the project and guidance, which helped me in completing this task through various stages.I would also thank to **Miss Pingle,Sneha**, my Project Mentor for always helping,giving me good suggestions, solving my doubts and guide me to complete my project in better way.

Lastly, I thank almighty, my parents, brother and friends for their constant encouragement without which this assignment would not be possible.

- Sharad Thakur
15MECE27

Abstract

Manual testing of a device under test are tedious, time consuming and prominent to errors. Industry is switching to test automation solutions as they are robust reliable and pocket friendly. Many Automation tools with script handling capability can perform various tasks but many teams dont have required resource and skills to establish that framework. This thesis focuses on how effective automation tools are with their correct implementation in end-to-end automation.

Automation tool kit is a hardware tool that is utilized to monitor and control the Device under test. It is a slave tool that has various control capabilities and is commanded by host system. Automation tool kit legacy has been replaced with Automation tool kit 2 as it went obsolete. Automation tool kit 2 has better software handling capabilities, how? Is discussed in thesis. Automation tool kit 2 have API written in modular technique which helps in easy implementation and easy debugging of bugs. Newer version being backward compatible provides support to older platforms also. End-To-End robust Automation framework system should be capable of handling failures too which means Automation framework should have Fault tolerance system and death level detection system. Automation tool kit provides such capability to Automation framework for implementing fault tolerance and death level detection modules. Postcode can be read which determines system death state were after correct fault tolerance technique can be implemented. Before implementing the new features thorough testing is required using Automation tool kit 2. The changes required in automation framework in order to implement automation tool kit 2, frameworks stability testing, fault tolerance and system death level detection, features as various section of this thesis.

The implementation of this project results in unblocking of test cases, time-saving, cost-saving for bios validation. Detailed quantitative numbers are mentioned in results chapter. It also showcases how paramount utilization of automation tools in test automation results in saving of time and cost.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Background

Automation: The act or process of converting the controlling of a machine from manual to more automatic manner. Hearing automation, one pictures enormous factories with big robotic arms manufacturing products round the clock. But automation is just not that big thing as big word it is. Automation can be even as simple as renaming a bunch of file or copying data from one location to another.
Automating a system is one time tedious job with long term benefits. It reduces manual efforts, increase efficiency and increased throughput or productivity. Improved quality or increased predictability of quality. Improved robustness (consistency), of processes or product. Automation has been achieved by various means including mechanical, hydraulic, pneumatic, electrical, electronic devices and computers, usually in combination. Complicated systems, such as modern factories, airplanes and ships typically use all these combined techniques.
Automation is second stage of testing. For making things automated manual test cases should be performed successfully. Bios has large number of features and functionality to be tested. Automation helps to reduce repetitive test cases to save time.

1.2 Motivation

Good perks of a project helps you to select it. But motivation is what keeps you going till the end.

Its good to do productive work. Work that benefits the team. Work with most learning outcome is much of value. Job of Automation in a validation and debugging team is a challenging one. one must have complete understanding of the system you work to implement your solutioning. With new platforms coming many major and minor revisions are required to migrate the system. This Porting process is also challenging.

Because of constant regressive work in the team there is less time to revisit the corned issues and fix them. One such main stream issue are migrating to new version of tools. it demands time and efficiency so that process can end with minimum errors. Automation tool kit has to be ported to it's new version and needed to be utilized maximum. The trickiness and amount of knowledge that i would gain was enough for me to motivate to choose this as a project work.

1.3 Problem statement

Automation tool kit is being widely used as a validation tool among many teams. It has a wide variety of hardware and daughter card support. One can over power source control, front panel control, clear the cmos, read post code status, reading back bios code as well as version and etc. It provides facility of handling its features using GUI and scripts both. The issue with legacy ATK was that that it was complex in implementation. Debugging and making changes with legacy was not easy. It missed modularity in its structure. So whole new implementation in respect with software was seen in ATK v2.0. It came up as ATK version2.0, also referred as ATK2. Greater effort was to port complete system to new version of ATK. Au-

tomation framework has to be restructured to adapt new version of ATK tool. Changes where required in every script that slightly even utilizes ATK in it. Pain increases because it has to be done for all the platform ATK supports. Though ATK is not a platform specific devices but there does remain minor difference in form of implementation. So, to migrate from ATK legacy to ATK2 automation framework has to go through all the major and minor changes necessary. To add up to the scoop of hurdles another was time as ATK legacy went END OF LIFE. New platforms only supported ATK2. This resulted into blockage in automation test cases for new platform.

Hence, faster and precise porting is required.

1.4 Gantt Chart

The time-line of project work from the start of the project is shown in below gantt chart.

ACTIVITY	PLAN START	PLAN DURATION	ACTUAL START	ACTUAL DURATION	PERCENT COMPLETE	Period july	august	september	october	november	december	january	february	march	april	may	june
Knowledge transfer of org	1	2	1	2	100%												
learning bios flashing and debugging methods	1	2	1	2	100%												
Post code tool development	2	2	2	2	100%												
Ramping up python	2	4	2	5	100%												
Working with ATK1	3	3	3	1	100%												
Syncing ATK2 with current platforms	4	5	4	2	100%												
Script based testing of ATK2 features	5	6	5	7	100%												
HW & SW based solution for conn & disconn onboard cards	6	7	6	7	100%												
Test scenario for on board cards	7	7	7	7	100%												
Recognising system fault level using ATK and post cord reader	8	9	8	9	100%												
Test scenario for code porting	10	11	10	11	100%												
Porting automation infrastructure to ATK2	11	12	11	12	70%												
Report Making	10	12	10	11	100%												

Figure 1.1: Gantt Chart

1.5 Thesis Organization

The thesis work carried out during the course of time has been presented here in four chapters.

Chapter 1: Introduction In this chapter importance of this thesis will be explained.Background study for thesis work to get going and is the motivation behind the selection of this topic for thesis. Gantt chart shows the time line pf the work to be done. Before that understanding the work is necessary, problem statement explains the base of this project, hurdles and glimpse of the path to overcome it.

Chapter 2: Literature survey It describes about bios its draw back and requirement for uefi bios.How validation testing is performed and its importance too. Brief study about automation and different framework and why python is important in automation. what features of python helps in implementation of automated test scripts.

Chapter 3: Pre-operating system firmware Bios works as interface between hardware and operating system. this chapter will well explain it. It also describes different modules stitched with bios and there importance. How bios differs with platforms can be learned from this chapter.

Chapter 4: Automation framework Validation testing is done in automated fashion. In this chapter knowledge about different automation framework is shared, how they are implemented and why they are necessary. Importance of there modularity can be understood in this chapter.

Chapter 5: Automation tool kit This chapter will describe about the tool being used for automation. ATK is a versatile tool capable of monitoring and controlling a validation platform. How to work with it using GUI and API's will be learned in this chapter.

Chapter 6: Fault Tolerance System This chapter discusses about implementation of Fault tolerance system in automation framework. It gives knowledge of various FTS implementation methods and in detail discussion of recovery block

mechanism used to implement FTS.

Chapter 7: System death level detection The greater the amount of information is more precise decision can be taken. How we can gain system death information and

Chapter 8: Results This chapter includes the implementation results of the project.

Chapter 9: conclusion This chapter contains the conclusion regarding the report and what can be implemented using the knowledge bases shared in this report. This is first part of project work and contains major of theory and study work. Conclusion defines how these information will be put in use in completion of project.

Chapter 10: Future scope Further work that can be done regarding this project is stated in this chapter. The scope of enhancement beyond this thesis are mentioned as a part of future work.

Chapter 2

Literature survey

2.1 BIOS:Pre-os Firmware

BIOS is acronym for BASIC INPUT OUTPUT SYSTEM. A BIOS functions as an insulator between the hardware on one hand, and the operating system or application software on other end. It hands over control of hardware to operating system. BIOS prepares your computer for correct functioning. It can also be stated as set of commands which run, check, performs several functionality within computer to meet basic requirement to be able to boot operating system. Bios is kept in a non-volatile memory location called ROM (Read only memory). Rom used to keep bios is EEPROM (Electrically erasable read only memory). It resides on motherboard of computer. [1]

Roms are not of as large in size as of RAM or storage devices. That means bios and other firmware which dwell in ROM are of small in size. Bios is of size of few MBs only.

Figure 2.1: Bios configuration [7]

There are few major tasked performed by BIOS

- **Detecting components and POST:** Bios program checks for the major devices connected with your computer. It checks if they are functioning properly or not. Bios performs POST (power on self-test). It checks CPU, RAM, interrupt and DMA controllers and other parts of the chip-set, video display card, keyboard, hard disk drive, optical disc drive and other basic hardware are working correctly or not. If any failure is scene and operating system cannot be booted it flashed error either in for of message or beep or hex code. Only after POST is successful computer can boot up. An error found in the POST is usually fatal (that is, it causes current program to stop running) and will halt the boot process, since the hardware checked is absolutely essential for the computer's functions.

- **Booting Up:** Once POST is done, computer is ready to boot. But before booting BIOS has to figure out from where to boot. It checks all storage devices connected for boot program. It has a specific order to check for called boot order, if there are more than one boot programs bios lists it or selects the default one. In the drive boot programs are kept at a separate section called master boot record (MBR). Once these boot loaders are triggered they

are ready to take command from BIOS to OS (MBR).

- **Gate keeping:** Once operating system is loaded BIOS still has work to do. Being in background it provides low level support for OS in handling major tasks. Consider if operating system needs to talk to the hard drive, for instance, it doesn't have to know how or where your hard drive is attached. It just sends the message to the BIOS, which takes care of passing it along to the correct drive on the correct socket.

- **Bios CMOS setup:** Although BIOS comes pre-installed and pre-configured still you can change bios setting. A button cell can be seen attached on motherboards. It helps keep the changed bios setting in a small CMOS RAM. If you clear the ram bios setting would go back to default. Changing boot order, cpu frequency, clock speed, password are few basic stuffs that can be altered in bios.

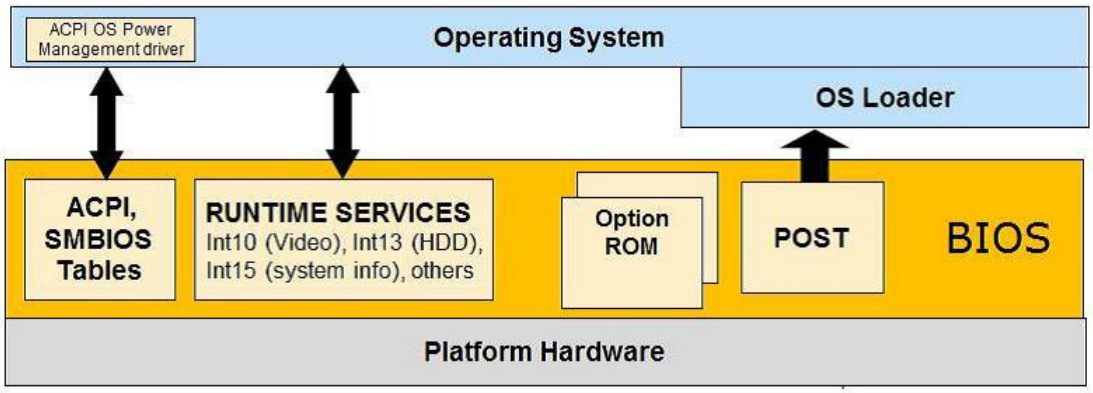


Figure 2.2: High level diagram of BIOS [7]

BIOS is medium to communicate between operating system and platform hardware. BIOS is the only software in the platform that knows all the details of the motherboard. POST is the responsible for testing system which will use OS loader to load

operating system whereas ACPI and SMBIOS tables are useful for power management and to control it. Run time services are services which will generate interrupt for particular service to OS then OS will give return back notification by executing that services.

2.1.1 UEFI and BIOS comparison

UEFI replaced bios legacy. Uefi supports bios legacy features.UEFI can support remote diagnostics and repair of computers, even with no operating system installed.Unified Extensible Firmware Interface(UEFI) is grounded in Intels initial Extensible Firmware Interface(EFI) specification, which defines a software interface between an operating system and platform firmware. The UEFI architecture allows users to execute applications on a command line interface. It has intrinsic networking capabilities and is designed to work with multi processors systems.

Legacy BIOS	UEFI BIOS
This is the traditional BIOS	New architecture based on EFI spec
Written in assembly code; initially designed for IBM PC-AT	C based; initially designed for Itanium server systems
Interface is per-BIOS "spaghetti" code, not modular	Well defined module environment and interface based on EFI specification
Lives within the first 1MB of system memory	Can live anywhere in the 4GB system memory space
Uses 16bit memory access, requires hacks to access above 1MB memory	Allows direct access of all memory via (32-bit and/or 64 bit) pointers
Supports 3rd party modules in the form of 16 bit Option ROMS	Supports 3rd party 32/64 bit drivers
No built in boot/test environment	Built-in boot/test via EFI - Shell
Only supports 16-bit runtime services such as INT10, INT13, etc	New runtime interfaces and supports legacy OSs and 16-bit legacy devices
Example of Legacy BIOS: AMI core 8, Phoenix legacy BIOS	Standardized implementations: Aptio (AMI), H2O (Insyde), Tiano (Intel)

Figure 2.3: comparison between uefi and bios [8]

2.2 Validation testing

To validate if results being achieved are being achieved in right manner. The process that determines if product satisfies specified business requirement or not is called validation testing. This evaluation can be done either during development period or at the end of the development process of the product. Products can be software as well as hardware.[6]

Most products go under validation testing in there development phase only. Because things are subjected to change easily and with cost efficient way while in development phase. Validation testing helps in producing accurate and stable product with less bugs and defects.

Validation is required because often it happens that what is verified on paper may not be achieved with the product, so validation is required to resolve anomaly. There is a difference between verification and validation where verification defines checking for if required outcome are obtained or not while validation defines that outcome have been achieved in required way.

Under validation of product there are many kinds of testing it under goes.

- **Unit testing:** In this type of testing smallest testable part of an application is tested for its functionality as an individual entity.
- **Acceptance testing:** This technique of testing defines if software has meet its requirement or not. This reflects the acceptance of the product.
- **Regression testing:** Which new changes in the software done older programming should still work. The changes should not affect passing functionality. So with every new fix other functionality should also work.
- **White box testing:**This is a method of testing of software where internal functionality is also tested with related outcome.
- **Black box testing:** In This method only result is monitored with respect to given input

Importance of validation

- Quantitatively determine the variability of a process and its control.
- Investigate deviations if any from established parameters.
- Cost improves quality of product.
- Greater scrutiny of the process performance for development and deployment of process controls.
  Test case: they are written for validation. Formal method of validation
  These test cases can be manual or automated. Automated test cases reduces validation time and human effort. Typical test case parameters that are required to run a test case.
- Test case ID
- Test case scenario
- Test description
- Test steps
- Prerequisites
- Expected outcome

Automation of test cases is biggest asset and requirement of validation industry. Test scripts are written to automate test cases. Scripting languages are base for writing these test scripts.
But not all functionality of product can be tested using automation so manual validation is required. For this, with constant human intervention test cases are to be followed

2.3 Automation

Software automation testing means testing the application using a special software which controls the tests that are to be runned on our product to validate it. Test that are generally automated are regressive testing. Test automation can be done for continuous testing purpose, repetitive test can be automated and kept for running. Automation reduces human error .A specific formal routine can be set for test cases to be runed by.[6]

In Regression testing due to a minor change in application every other functionality has to be checked again. This is necessary but is tiring job. Only way to accomplish it faster is with more skilled labor working manually and still living chances of error. This also increases the cost of product.

Test automation can be costly because of its software, tools being used and automation infrastructure need to be built for that. More flexibility we require in our automation framework more expensive its implementation becomes. Flexibility defines whether remote access feature are required or not, email acknowledgment is required or not, step by step log is required or not. This cost is although one time investment and can be calibrated with regular use of it in regression testing for different platforms.

Some important features of automation framework

a. Data driven capabilities

b. Debugging and logging capabilities

c. Extensible and Customization

d. E-mail Notifications

e. Platform independence

f. Version control friendly

g. Support unattended test runs

If we talk about test automation approaches, there can be two kind of approaches

- Graphical user interface testing
- Api based testing

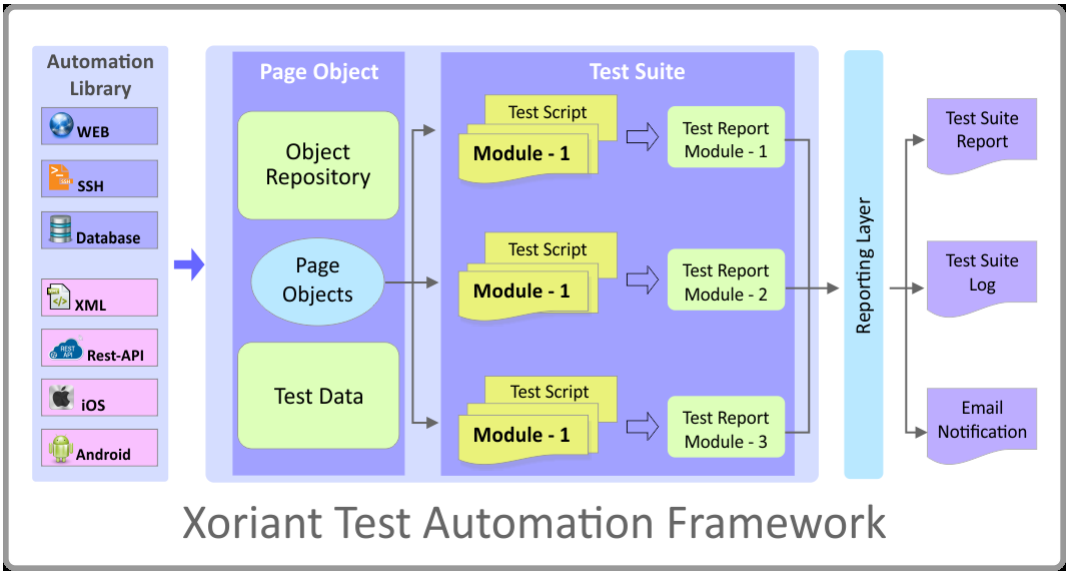


Figure 2.4: Example of xoriant automation framework [9]

2.3.1 Graphical user interface testing

Automation software creates a graphical interface where with mouse click and keystrokes execution, controlling of test scenario can be done. Results can be seen on the GUI itself. This gives handy atmosphere for the user to work on. We can record the events and play back later as many time as required. This is helpful in web related testing where we can monitor the screen.

List of test tool with gui interface

Table 2.1: List of test tool with gui interface

tool name	**Developer**	**License**
Essential test	Zeenyx Software, Inc.	Proprietary
HP WinRunner	HP	Proprietary
Linux Desktop Testing Project	Collaborative project	Gnu lgpl
Oracle Application Testing Suite	Oracle	Proprietary
Rational Functional Tester	IBM Rational	Proprietary
Selenium Testing Anywhere	Collaborative project Automation anywhere	Apache Proprietary
Visual Studio Coded UI Test	Microsoft	Proprietary

2.3.2 Api based testing

API testing is also being widely used by software testers due to the difficulty of creating and maintaining GUI-based automation testing. It involves directly testing APIs as part of integration testing, to determine if they meet expectations for functionality, reliability, performance, and security.[9] Since APIs lack a GUI, API testing is performed at the message layer. APi testing is faster comparative to GUI testing as it does not include overheads. API testing is considered critical when an API serves as the primary interface to application logic since GUI tests can be difficult to maintain with the short release cycles and frequent changes commonly used with agile software development.

2.3.3 Test Automation Framework

Automation framework is a well-defined structure to run the test cases in a specific manner to achieve results. Automation framework are different than test tools. There can be different types of framework depending upon application it need to test.

Various framework/scripting techniques are generally used:

a. Linear (procedural code, possibly generated by tools like those that use record and playback)

b. Structured (uses control structures - typically if-else, switch, for, while conditions/ statements)

c. Data-driven (data is persisted outside of tests in a database, spreadsheet, or other mechanism)

d. Keyword-driven

e. Hybrid (two or more of the patterns above are used)

2.4 Scripting language

It is a programming language generally for run time environment. It produces scripts with which we can execute tasks in automated fashion. Environments that can be automated through scripting include software applications, web pages within a web browser, the shells of operating systems (OS), embedded systems, as well as numerous games. Scripting language can be domain level specific language for a particular platform or high level programming language working at higher abstraction level in a mainframe. Examples of programming language are Python, Perl, tcl, ruby, lua. Type pf scripting language

- **Glue language:** Scripting for combining software components. Language specialized in gluing components together are termed as glue languages.
- **Job control language:** Languages specifically build to control the job of a system are listed in this shell language. It is done using controlling command line interpreters such as shell, ms-dos or apple shell.
- **Gui scripting:** With the advent of graphical user interfaces, a specialized kind of scripting language emerged for controlling a computer. These languages

could in principle be used to control any GUI application; but, in practice their use is limited because their use needs support from the application and from the operating system.

- **Application specific language** Many computer games get help of scripting language to define the action of non-player objects in the game. These generate specific patterns for them to be embedded in the program. These languages can be program specific but are general at higher level.

- **Embedded language:** These languages may be technically equivalent to an application-specific extension language but when an application embeds a "common" language, the user gets the advantage of being able to transfer skills from application to application. A more generic alternative is simply to provide a library (often a C library) that a general-purpose language can use to control the application, without modifying the language for the specific domain.

2.5 Python programming language

widely used high-level, general-purpose, interpreted, dynamic programming language

It is a very powerful object oriented programming language. It has high readability, what you write can be read for a meaning. Python is open source language which is compatible with many operating system. Lets go with some history of python.[10]

2.5.1 History of Python

Pythons began in December 1989[31] by Guido van Rossum at Centrum Wiskunde and Informatica (CWI) in the Netherlands as a successor to the ABC language. Later in 2000 python 2.0 was released with major up-gradation including a cycle-detecting garbage collector and support for Unicode. With this release the develop-

ment process was changed and became more transparent and community-backed. Python 3.0 also commonly referred as Python 3000 or py3k a major, backwards-incompatible release, was released on 3 December 2008 after a long period of testing. Latest released version is python 3.5.

2.5.2 Syntax and semantics

Arrangement of words and there logic is syntax and semantics. Python has more of English keywords for expression than punctuation. It also uses less of syntactic expression than c and pascal.[10]

Python uses space for indentation in comparison to brackets and curls in other languages. Increase in indentation after some statement defines start of a block while decrease in indentation show the end of the block.

Python has many modules for statement and flow control like if statement, for, while else-if while few more in it are like try, exception for better handling of statements. Implementation of these conditional statement are different than what we see in c, c++ and other language. It has more linguistic expression way of implementation. Python focus more on string operations. String variables does not have fixed size in python they can be a single character as well as size of paragraph.

Methods: Python is object oriented based language. It intensively uses benefits of classes, methods, instances type of stuffs from OOPS. Import function are used to call the different modules in a file. Once called we can use the methods from inherited class.

2.5.3 Libraries

Biggest asset of python is its large stock of libraries. It supports wide range of platforms because of its vast library collection. If not it is open source and one can create the library and add up to the collection. Installing python or running python does not needs all the package to be dumped together. We can get on the need

basis. There is a python package index, a full repository for support to third party software. It holds over **92,000 packages** offering wide range of functionality such as.[**?**]

- Graphical user interfaces, web frameworks, multimedia, databases, networking and communications
- Test frameworks, automation and web scraping, documentation tools, system administration
- Scientific computing, text processing, image processing

2.5.4 Development and implementation

Python is a run time language. It can be executed on a command line in a sequential runned fashion. Pythons command line is a basic one while some developer software provides additional features on command line too example Ipyhton IDLE. Editor can also be used for writing python scripts. Write using any editor, save the file using python extension .py and execute with python interpreter.
The first python implementation was cpython where its library was written in mixture of c and python. CPython was intended from almost its very conception to be cross-platform.
Pypy is a faster version of cpython. Its just in time compiler gives significant improvement over cpython. Stackless python is a fork of cpython which implements micro threads. Whereas micropython is faster version of python 3 for implementation in microcontrollers.
User mode debugging is a simplest form of debugging process which is capable of single target user mode process. In user mode, you can simply examine the program state and you can also modify the states. So it will be notified when special events happening in the target process. Debugger performs debugging process with target debugger till the process is running on target. This is also known as live debugging.

Python has wide usage. So we can see its wider implementation on different platforms with vivid implementations and compilers being developed.
There is another concept known as postmortem debugging in which debuggers examine a dump files containing a snapshot of a given process in user mode.
There are several compilers to high-level object languages, with either unrestricted Python, a restricted subset of Python, or a language similar to Python as the source language

Three built in user mode debuggers come with the windows debugging tools. Those are cdb.exe, ntsd.exe and windbg.exe. These all have same functionality but it works in different ways. These all can do console application debugging and graphical windows program too. If the sources are available than these all can perform source level debugging too. Straight machine level debugging is also done by them.

a. Jython, it can be executed with every Java virtual machine implementation. This also enables the use of Java class library functions from the Python program.

b. IronPython, runs Python programs on the .NET Common Language Runtime.

c. The RPython language can be compiled to C, Java bytecode, or Common Intermediate Language, and is used to build the PyPy interpreter of Python.

d. Pyjamas compiles Python to JavaScript.

e. Shed Skin compiles Python to C++.

f. Cython and Pyrex compile to C.

Chapter 3

Pre- operating system firmware

Pre os firmware sets up link between operating system and hardware. It works as interface between both. Bios stitched with other critical modules are called pre-os firmware. Some of these modules are mandatory while few are optional and are required only on specific requirement. Bios is most important part of pre os firmware. It is largest stack holder in pre-os. There are large variants of bios developed for a specific platform depending on the major features required. A single platform has large variants of SKUS which further required different bios builds. To understand better we will have a look at different Intel platforms architecture.

3.1 Intel platform

Intel platforms have two main constituents CPU: Central processing unit and PCH: Peripheral control hub. Intel architecture beginning from 8-bit processor is now being implemented on 64 bit processors. Intel releases different processors under similar codename. Kabylake micro-architecture can be found with core I, atom and even xeon processors. So, the basic difference is created by the ingredient stuffing inside the processor. Intel platform has two important things CPU and PCH. Combination of these two decides the kind of SOC and functionality they can perform.

Different kinds of packages of soc intel platforms have

- Multichip package: In this kind of package CPU and PCH both are under one die. They make one SOC only. These kinds of packages can be seen in low powered devices like ultra books.
- Two chip package: in this package cpu and PCH reside in different sockets. There are two different silicon sockets on motherboard. Single SOC for both pch and cpu. These kind of package can be seen in high end laptops and desktops.
- Multi socket package: In multi socket. Soc of CPU will have more than one cpu inserted in it. There will be two silicon sockets one for multi cpu soc and other for pch soc. These type of packages are used for server machines.

Different segments are there depending upon the die package. Validation is divided into these segments as they are major division.

a. Ultra thin segment

b. Ultra light segment

c. Halo segment

d. Desktop segment

Every segment goes under validation. There are specially designed motherboards for all the segments called validation platforms. Where all the functionality of pre-os firmware can be tested. Only after these things are validated they are passed for OEM use.
Validation is a continuous process. Where with every testing and validation bugs errors fixes are done and new version is released with all fixes. Again the version has to go under similar scrutiny process until all the major issues are resolved and board becomes stable.

Bios basically checks for stability between motherboard and silicon. Because operating system commands to silicon and silicon is the one which has to use the resources available on the motherboard efficiently and effectively. This utilization does not only depend on single thing. Software and hardware are equally responsible for in sync and correct functioning of a feature.
Hardware dependency:

- Cpu and pch
- Motherboard
- Add on cards
- External segments

Software dependency:

- Bios
- Device drivers
- Operating system and drivers
- Utility tools and drivers

3.1.1 Pre-os Firmware components

Correct combination of both hardware and software dependency ensures the functioning of a feature. Establishing this combination is called best known configuration (BKC). In this pre-os firmware plays a crucial role because it has major dependency. Pre-os firmware came into existence after UEFI. Due to the limitations of bios uefi was developed. Many wrappers where build across bios and bios was made modular. This helped in better implementation and adaptability as per variants. Uefi is nothing but specifications that pre os firmware follows.

Figure 3.1: Intel validation motherboard [7]

Some Ingredients stitched with pre-os firmware:

- **Bios firmware:** Bios firmware handles the part of initialization of devices. POST operation. Boot loader operation, customization of bios configuration. In uefi environment it provides graphical interface for bios configuration. Bios flash chip also known as spi flash chip is directly linked with PCH. Serial peripheral Interface bus (SPI) is use to flash and communicate with BIOS program that resides under the flash chip. For every pot operation bios provides a post code. Post code is in hexadecimal format. This information of post code is given to bios from PCH.

- **Processor microcode:** Microcode resides inside cpu in a high speed memory chip. It initializes cpu. It is closely attached to machine level. But these microcode are hard coded. They cannot be altered once programmed. To remove this critical drawback microcode were attached with pre os firmware

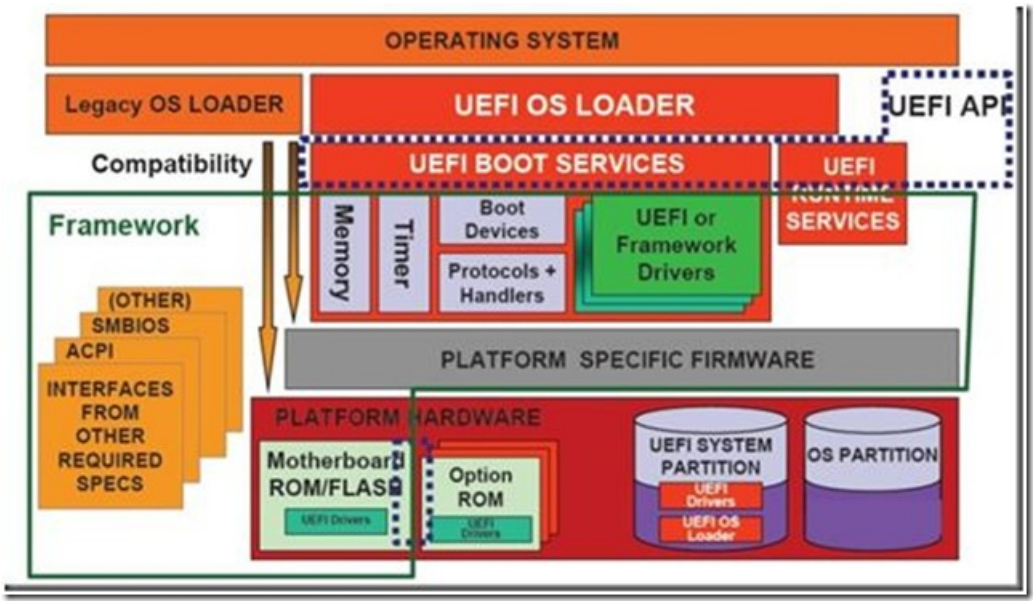


Figure 3.2: Uefi architecture [8]

as they can interact with cpu. If any update or patch is required in cpu microcode it can be fetched from pre os.

- **Pch patch:** Similar to microcode these patches are attached for pch. Not all functionality can be coded in a silicon. This helps to store the additional data regarding pch. They are mainly power management specific information.

- **Option rom:** Peripheral devices like ram, storage device, video cards can have there own rom chip. In that resides option rom or extension rom. They can be replaced with the module in bios. Option rom provides great flexibility in feature enabling. For example if a device is connected to 4k HD display but processor does not support than it will be useless to load drivers for 4k display instead processor can work with the installed firmware.

- **Security patches** Security is not only required form viruses but from malfunctioning also. Security patch helps in authentication of all the tasks and checks performed by bios. This Ensured any fraud device is not detected and recognizes only trusted devices and path. This keeps cpu safe from breaches

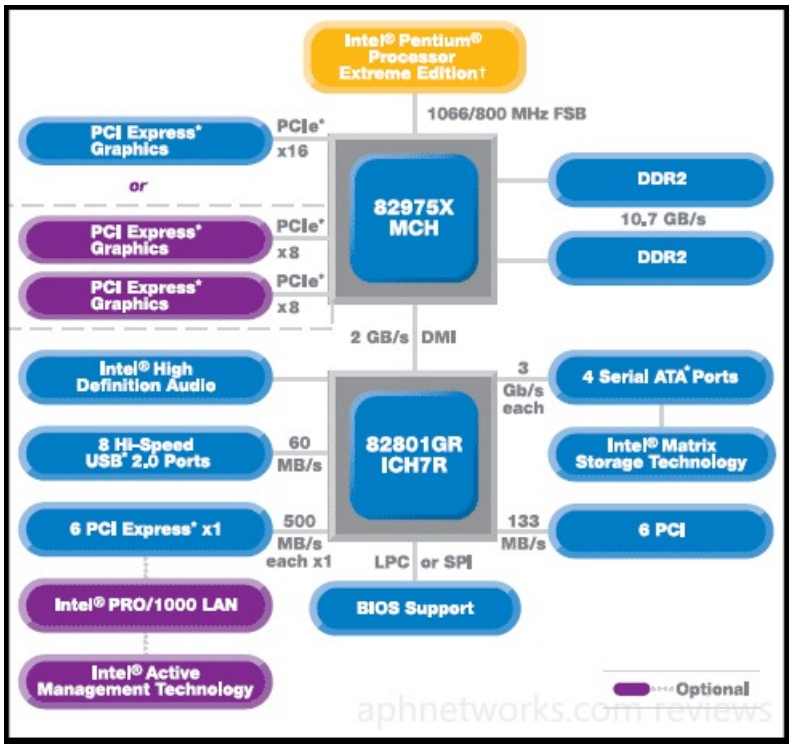


Figure 3.3: Bios and chip-set interface [7]

and missionaries. There are large number of security straps in cpu which can be enabled and disable via bios. These straps decide the level of security in system. Curtain straps can be customized are called soft straps while some are hard straps that cannot be altered.

- **Graphic drivers:** The GOP driver is a replacement for legacy video BIOS and enables the use of UEFI pre-boot firmware without CSM. The GOP driver can be 32-bit, 64-bit, or IA-64 with no binary compatibility. UEFI pre-boot firmware architecture (32-/64-bit) must match the GOP driver architecture (32-/64-bit). The Intel Embedded Graphics Drivers' GOP driver can either be fast boot (speed optimized and platform specific) or generic (platform agnostic for selective platforms). [3]

- **Thunderbolt / type c support:** For usb type c and thunderbolt support features are enabled in UEFI itself. Thunderbolt has to support many fea-

tures in pre-os environment so its support was required in pre os firmware. Thunderbolt features are optional in bios. We can enable and disable it as per need.

- **ME drivers:** These drivers provide the feature of hardware-based remote management, security, power management, and remote configuration features that enable independent remote access to Active management Technology-enabled PCs.

3.1.2 Testing of various Bios configuration

The presence of bios menus and BIOS settings are dependent on your intel platform, board model, hardware components installed, and the BIOS version. BIOS menu titles may differ as per them. Functioning of all the listing bios options are necessary. For this testing of each and every function are required. Verification and validation of these bios options using Automation framework will be mentioned in next chapter.

Chapter 4

Automation framework

4.0.1 What is a framework

A Test Automation Framework is a structure that is laid to provide an execution environment for the automation test scripts. The framework provides the user with various benefits that helps them to develop, execute and report the automation test scripts efficiently. It is specifically a structure that is created for automation running using scripts. [5]

There can be various advantages of framework like scalability modularity re usability, lesser cost and maintenance. SO to grab these benefits user are advised to work with one or other test framework. Moreover the need of single automation framework arises when we have bunch of developers working on different modules of a single project and you want them to work with single approach. If every developer starts implementing there own approach than it will be difficult to communicate and share results and knowledge.

4.0.2 Why Automation framework is required

Requirement of automation framework has been made pretty clear in above section. We will see the greater advantages of automation framework here and how to choose suitable framework. As defined by framework, automation framework helps keep the

work flow structured. These structure depends on the type of framework users is using.
We will list down few advantages of having a automation framework.

- Re-usability of code: Framework maintains the code in a centralized library. if any new code is to be written with existing feature code can be used form library.

- Maximum coverage: Centralized track record helps to keep an eye on things. This records helps to maintain maximum coverage.

- Recovery scenario: Any crash on the local platform or host can ruin your test case but it can be regenerated from framework. because automation framework would work as a data backup no data loss will be there.

- Low cost maintenance: A centralized system helps in easy monitoring accessing and controlling of validation process. This saves time of developers in communicating within. Individual approach will increase the collaboration cost even.

- Minimal manual intervention: Automation framework provides end to end automation solution also. this helps in reducing manual intervention and errors.

- Easy Reporting: In a framework every test case runed will dump its logs and result at a same place or with same extraction process. this helps in easy reporting of bugs and issues.
  The image below we will see a example of automation framework. After setting up the framework engineer has to just give valid inputs in the system and rest are handled by the automation system. required inputs can be like.

  a. Host and device id, if there are more than one host and sut connected with framework.

b. test case parameters like what feature are to be tested, run duration, repeat cycles, any specific parameters.

c. Specific location for results to be saved.

d. Specific setting or function to be performed before or after the execution.

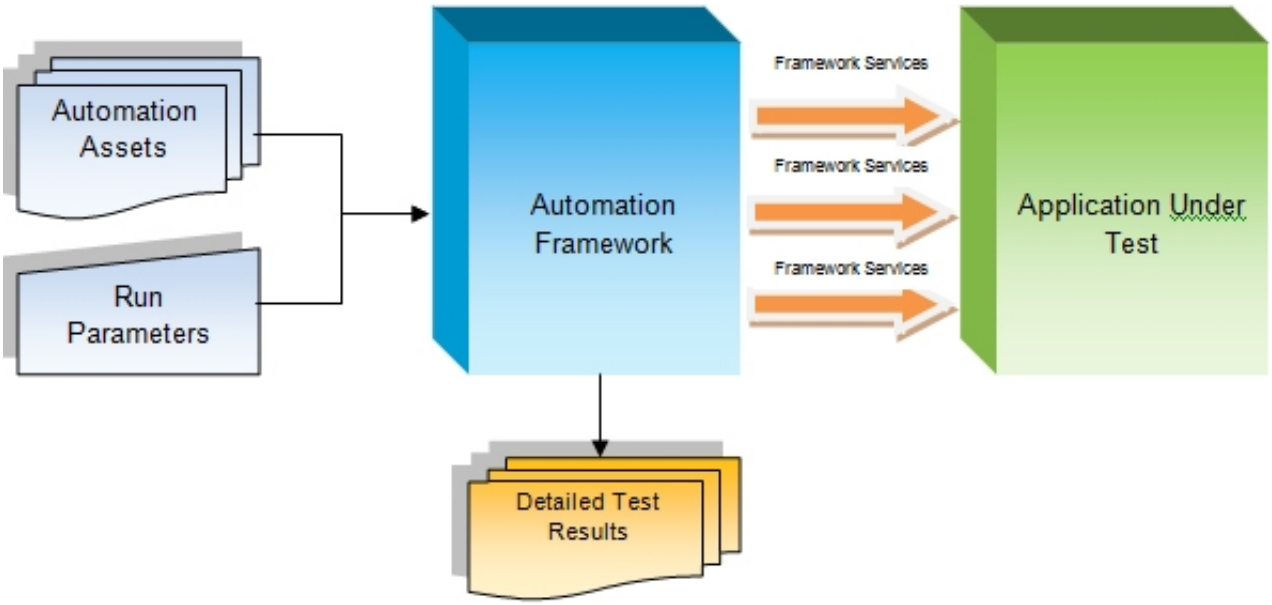


Figure 4.1: Example of automation architecture [6]

There are different kind of automation framework that can be setup. As we have basic knowledge about automation framework we will see different kinds of setups available in market. They can vary dependent on the advantage they provide.

a. Module Based Testing Framework

b. Library Architecture Testing Framework

c. Data Driven Testing Framework

d. Keyword Driven Testing Framework

e. Hybrid Testing Framework

f. Behavior Driven Development Framework

The automation framework which will be described in this complete thesis work is hybrid type of automation framework. Developed from library Architecture testing framework and data driven framework.

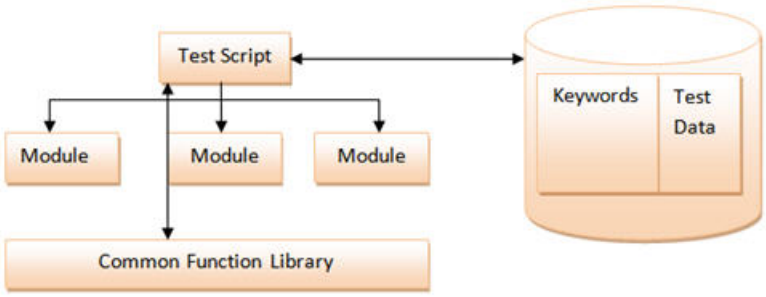


Figure 4.2: Hybrid automation architecture [5]

4.0.3 Hybrid automation framework for validation

Derived from library based architecture the first task to setup this network is to identify different functionality to make a modular model. In validation testing as we talked about there are large number of testing to be done. number of bios functionality are to be tested. But not all are unique many have repeated steps. For example

In 6th generation platform there are number of power options i.e Shutdown, sleep, connected standby, hibernate. Considering testing of each features are to be done form system under test's power down state, major steps would be common in these test cases.

Possible steps in test case of power options

- boot to operating system

- check if system is up?
- Put system into sleep/ shutdown/hibernate
- reboot the system back to os
- Check if system is in os or not?
- Results

Now from above test case steps maximum can be used for all other power option test cases. So instead of writing redundant script for all power option we can split it into functions and reuse it to great extend. A function can have its own script file. This can be done for all the functions regarding a set of ingredients. we can identify these functions with respect to bios and can prepare the scripts for them. This will create a library of it. This library can be centralized in the framework for all around access.

Automation can not be done using one implementation way and a single tool. There can be more than one tool or way of performing validation dependent on the scenario. this makes some test cases data dependent. With adapting data driven test framework we can make our framework more flexible. With correct run time data input correct library file can be chosen and a package can be generated for automated validation. The input data can also create hazards in testing. Input should be in a way that frameworks understand.

So to avoid parameter miss match error the input pattern are defined. Proper syntax have been developed for passing the parameter to the framework. It is called parameter syntax and it has to be followed by every developer who has to use this framework.This disciplines the process and increases the readability and eases the debugging process.

The major Advantage of this framework is that if any changes in functionality occurs it has to be changed at only one place for making it functioning for every test

scenarios.

4.0.4 BIOS Automation Framework

We discussed various automation framework and also categorized them. Intel's BIOS automation framework is build upon hybrid automation framework model. Modular representation of Automation framework is shown in the figure below.

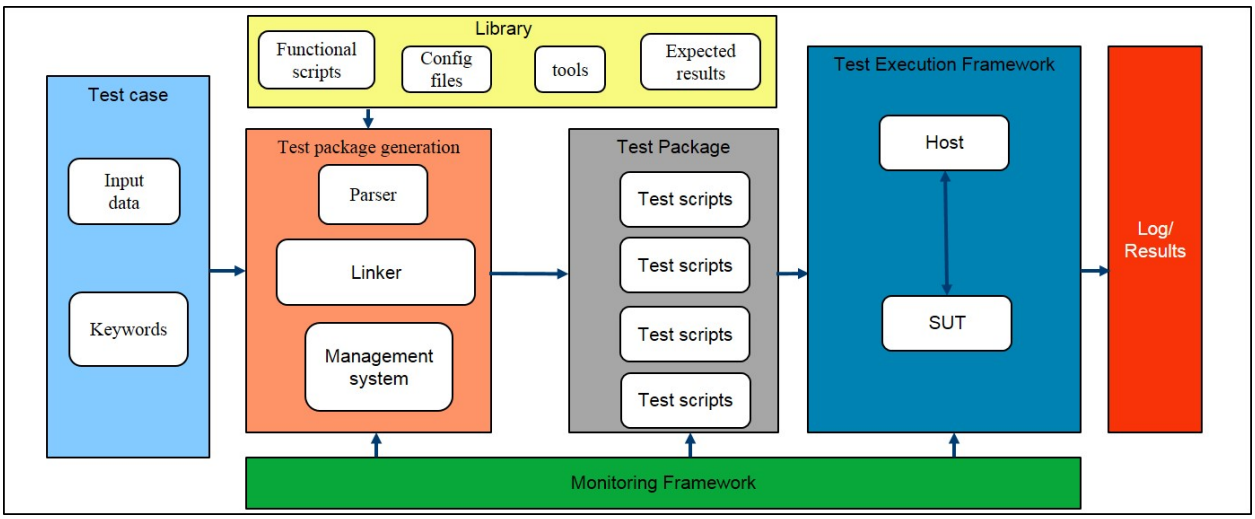


Figure 4.3: BIOS Automation Framework

Framework Explained: Test case contains the details of what has to be performed in a test case. There are input data and keywords associated with the test case which are passed down to test package generator block.

Test generator block gets specific test scripts dependent on the input values and keywords. Test package is generated and is passed to HOST and SUT.

Host runs the test package under a monitored environment. Logs and results are generated for every test case weather it passes or fails.

Chapter 5

Automation tool kit

Remote access capability is widely used in validation process so is it in intel. But There is no common solution for remote platform access. Various teams have developed personalized solutions to address this gap but it is typically platform and environment dependent. This complicates the debug process across the team. A common remote monitoring method was required to be developed.
ATK (Automation Tool Kit) is designed to solve this problem by standardizing these features and provides common software interface. It is able to cover a wide range of hardware requirements with minimal cost and high flexibility. The software component has been designed to be generic that can be integrated easily into other environments such as automation software and custom GUIs. A standard GUI is provided for debugging and remote execution monitoring. This tool is in use by several validation sites for multiple projects.
The operations that ATK can perform are listed below.

- Bios flashing
- Postcode reading
- Front panel control
- Clearing cmos

- AC connect/disconnect control
- Battery connect/disconnect control
- Sx , DSx state reading
- Relay control for any generic switching e.g. USB device plug unplug

5.0.1 Atk hardware and daughter cards

For accessing certain features of atk we require additional hardware. Atk device comes with large number of connecting pins with which we can connect additional hardwares. There are different kinds of hardware that are supported via atk software.

- For reading post code we have to connect **Lpc glider card.**
- For controlling power adapter **power splitter** is used.
- **DC Relay board** is used to control various buttons on platform.
- **Usb splitter** is used to control usb devices.
- **Paddle board** for flashing bios.

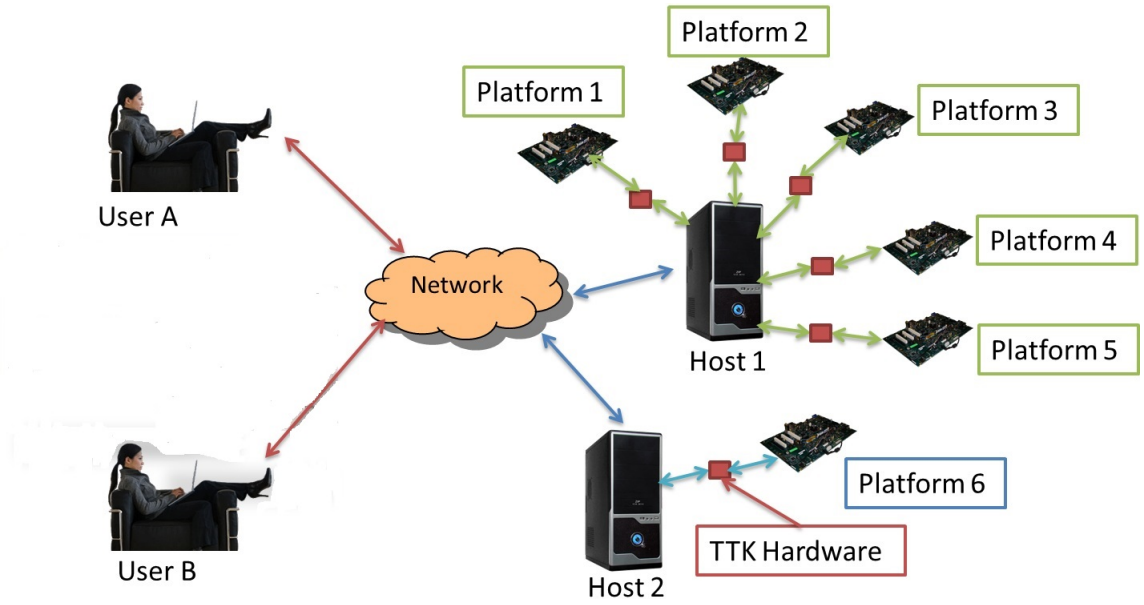


Figure 5.1: Atk highlevel overview [11]

Figure above is an example of ATK configuration where the user is connected to computer network and performs above capabilities remotely. In this example, multiple platforms are connected to the host computer through ATK hardware. The user is equip with software (in this context, we will refer this as client software) as interface to monitor, control and program the platform.
With one ATK server more than one platform can be controlled using client software. There are two GUIs that can be used depended on the operation to be performed.

a. Bios programmer: for performing bios operations like flashing bios, clearing cmos, reading bios version.

b. Client software: For controlling front panel, reading post code, voltage and led status.

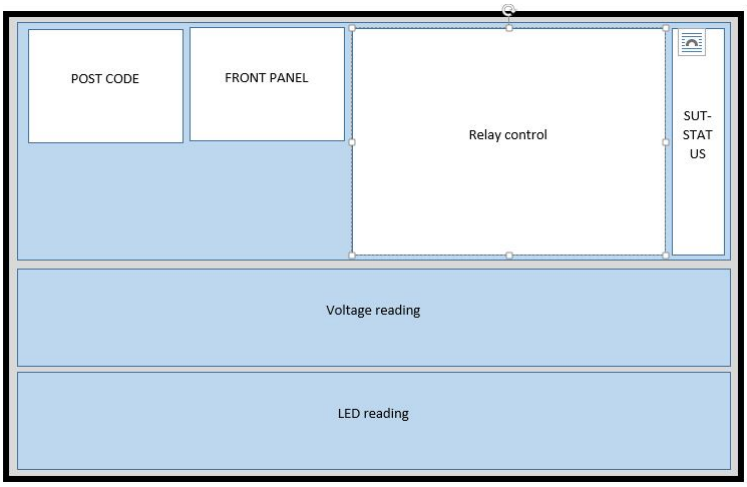


Figure 5.2: ATK client software gui

Using ATK client software facilities like post code reading gpio pin read write, AC control, front panel control can be accessed.

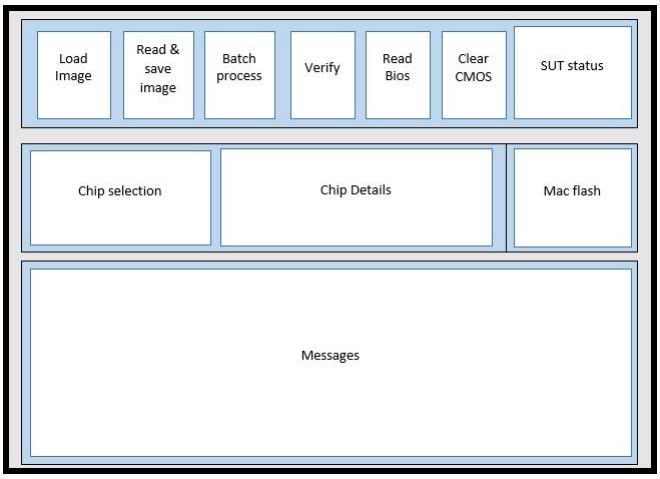


Figure 5.3: ATK bios programmer

Bios programmer is mainly used for bios options like reading and flashing of bios changing mac address inside bios.We can customize batch options also.

5.0.2 Difference between ATK legacy and ATK2

There is no difference between atk legacy and atk2 in terms of hardware or features. The gui and software implementation of atk 2 has been redefined. Atk can be used form gui and api both. In atk we can build our api using Python and c-sharp. Gui of atk2 is much simpler to use and better implementation technique. But major difference is in using apis.

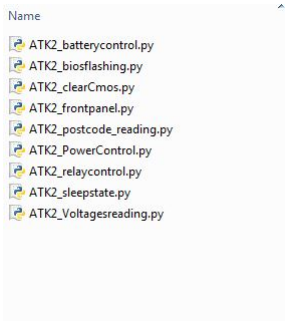


Figure 5.4: ATK v.2 API library files

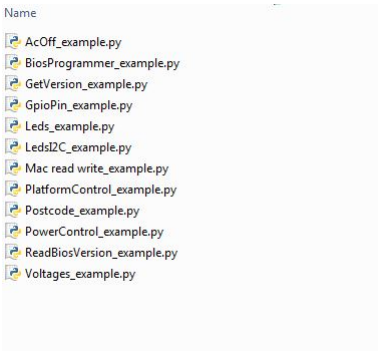


Figure 5.5: ATK v.2 API examples

Api are implemented in modular form. Every feature has its own API file and DLL file. For every major feature there is a separate library. So, to make a api we have

to call related library only. If any change are required in any feature related library can be changed, this eases the work of migration in future.
Features on different platforms can be same or can also differ. For example size of Bios image various as per platform. so flashing of Bios should be handled differently for each platform. Here modularity plays its role. We need to make changes only in Flashing APi for implement differently.
Atk software is built on c-# platform. For implementation of python api it uses .net wrapper which interlinks both c-sharp and python.
Advantages of ATK2 over ATK legacy

- It has modularity
- Debugging is easy in ATK2
- Api can be created easily
- Api can be created in c-# also
- Better informative and interactive GUI.

So, those were the difference between ATK legacy and ATK V.2. Further we will see where to apply these changes and what has to be modified in BIOS automation framework for implementing of the ATK v.2.
In the last chapter we saw Bios automation framework block diagram. Library contains all the scripts as per the keywords, functional keywords, config files. Those are the scripts that are specifically needed to be modified as per new api's. In all the library that has bits and pieces of older ATK scripts has be modified with the new api's of ATK V.2.

Chapter 6

Fault Tolerance System

6.0.1 Introduction

Time is money. Execution speed can help reduce time. And processing speed can only be increased in two ways parallel processing or distributed system. Parallel processing being an expensive option industry utilizes the true benefits of distributed system. Distributed system can be understood a resource sharing. While the number of dependent hardware will increase possibility of failure will also increase. This chapter will unveil the importance of fault tolerance system in distributed system. Starting with some basic difference in fault, failure and error. We will see fault classification and phases of fault tolerance system.

Distributed system is a model in which computing components are connected over network or a bus. In this kind of model each processor has its own memory device. If other node has to access data they have to talk through the processor. Distributed system provides stability in system failure. On failure of one node other node can carry on the task. In distributed system one host can maintain several clients.

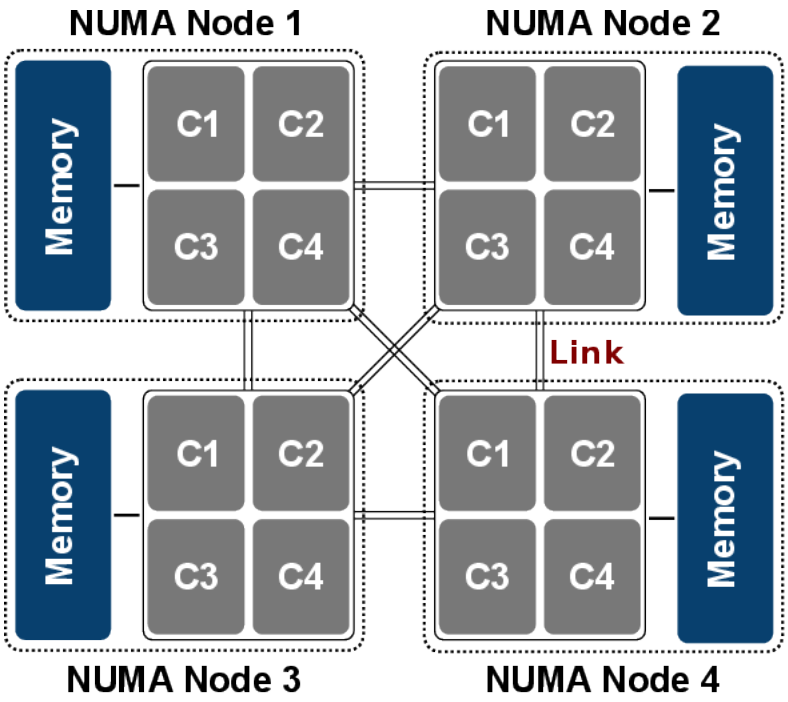


Figure 6.1: Fault tolerance in NUMA systems[12]

Numa is an example of distributed system architecture. Here each node has its own memory and controller and works as a separate entity. Major issues faced in establishing distributed system are

a. Fault tolerance

b. Communication primitive

c. Flexibility

d. Transparency

e. Stability

f. Scalability

The topic of interest for us would be fault tolerance.

6.0.2 Fault,Error and Failure

Fault tolerance is a property of a system which helps a system keep going after any event of failure too. Degradation in operating quality may be observed but system halt will not be seen. Failures can be of software or hardware type. Fault tolerance also ensures graceful degradation of the system in spite of sudden death.

Implicit is the systems specific behavior that is constituted as correct behavior, which becomes a reference for checking the faults in the system. A failure occurs when an actual running system deviates from this specified behavior and that cause of a failure is called an error. An error can be defined as an invalid system state,the state that is not allowed by the system behavior specification. whereas a fault is the root cause of a failure. That means that an error is merely the symptom of a fault. So a fault may not necessarily result in an error, but the same fault may result in multiple errors. Similarly, a single error may end up producing multiple failures in the system.[13]

For example, consider a software system in which an incorrectly written instruction in a program decrements a value of a variable instead of incrementing it. if this statement is executed, it will result in the incorrect value being written. If other functions calls this value, the whole system will deviate from its desired behavior. Here the incorrect statement is the fault, the invalid value is the error, and the failure is the behavior that results from the error. Note that if the variable is never read no failure will occur thought fault will be there. Or, if the invalid statement is never executed, the fault will not lead to an error. Thus, the mere presence of errors or faults does not necessarily imply system failure.[13]

The heart of all fault tolerance techniques is some form of masking redundancy to the components that have probability to fail. This means we replicate the components and algorithm which are prone to defects in such a way that if a component fails, the redundant non-failed replicas will continue to provide service with no appreciable disruption. There are many variations on this basic theme.

6.0.3 Implementation of Fault tolerance System

Fault tolerance can be differentiated as

- Software fault tolerance
- Hardware fault tolerance

Software fault-tolerance:

software fault tolerance is an ability of a software to handle the fault occurrence in the software or the hardware on which the software is performing. We will see the degradation in the ability and performance of the system. Primarily fault tolerance is implemented using redundancy. This will remain effective only when design defects are there due to redundancy. N-1 version, rollback and checkpoint are some example of software fault tolerance.

Hardware fault-tolerance:

Hardware fault tolerance is the ability of a hardware to keep on running even when a failure is scene. This is achieved by implementing additional redundant hardware which can take care of processing if one fails. Load sharing of work among n processor is also an option where a backup hardware is not required instead jobs are divided between them and on failure of one processor execution can still run on others

Hardware Fault tolerance are costly as they require additional hardware for implementation of this technique. So software fault tolerance mechanism are majorly used. The fault tolerance technique used in this Automation framework is software fault tolerance with checkpoint. We will talk in detail about this. But before lets see various software fault tolerance techniques

Recovery block:

Recovery block method is a simple method for fault tolerance. We can have more than one ways to achieve a required result. The execution is done using the primary method with less overheads to keep up the execution speed. At a point of failure

system is rollback and secondary alternate is applied. If any alternate is not effective enough to overcome the failure exception handler is called indicating required operation cannot be performed. Rolling back is also a complex process, hardware assistance may also be required at certain times. Recovery block uses checkpoint and recovery mechanism.

N-version software:

N-version software parallelizes n-version redundant hardware technique to implement software tolerance. There are N- different ways of implementation of a module. Each variant completes the same task but in a different manner. Each variant submits there result were decider decides for the correct result and submits it a result of that module. The n-version proves successful only when we have diversity in the task performance. Increase in the version increases the true outcome of the system. N-version may also have hardware dependency to get the task done.

Recovery block software fault tolerance is practical solution for less complex framework to tackle fault tolerance. In a test execution process various checkpoints are needed to be defined for the rollback to occur. We will further see different types of recovery blocks being used in Bios Automation framework. As we know these recovery blocks are run in sequential fashion each recovery block comes with a priority level that user can set. Priority once given becomes static throughout the test package generation. It can only be changed for new set of packages to be generated and tested.

The recovery blocks used in Automation framework.

- Cold reboot: System power is cut and it starts from initial boot sequence. In this system starts from power off state. Initialization of devices POST check is done. Memory reallocation is also done as part.

- Warm boot: Warm boot also known as restart is performed. Where system is just restarted and power remains connected to the machine. Running software are closed in this sequence. Warm reboots are required mostly when new

software is installed or any software stops responding

- Flash bios: On many occasions Bios (Basic Input output) firmware crashes and are needed to be flashed again. They have their dedicated flash chip in which BIOS resided. We can flash bios from the test system itself also but major test system have external port for bios to be flashed using external device
- Reset bios settings: Resetting the bios sets the default bios options. It is equivalent to doing a factory resetting of BIOS options. This is done by removing CMOS battery which saves settings in an external chip.
- Install/uninstall software: sometimes a software doesnt functions correctly, this requires uninstalling of a software are then installing it in a safe environment.

These are 5 recovery block which are implemented at time of fault occurrence. Priority can be set by user and as previously mentioned once priority set it becomes static. Priority goes from f1-f5. We will see a scenario in which on failure various recovery blocks are applied. In this report we go with one priority sequence

- T1- Warm Reset
- T2- Cold reset
- T3- Reset Bios Options
- T4- Flash Bios
- T5- Install/ Uninstall software

We see an example test case. Test case contains several steps. It fails at flashing BIOS. Failure scenario is the system ain't booting after that. which leads to the failure of further test steps. FTS is applied from the inception of the failure till the system gets alive again. If FTS is not able to fix the failure it throws exception.

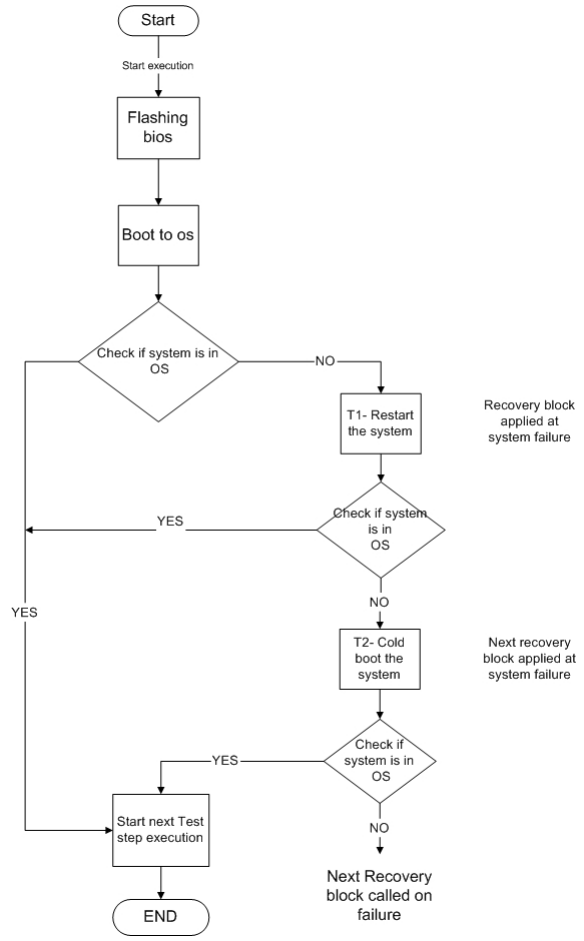


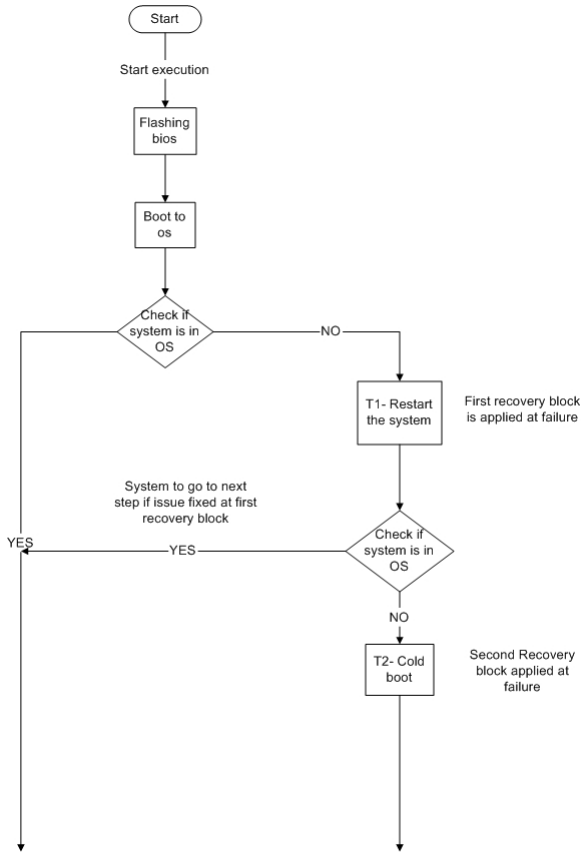
Figure 6.2: Fault tolerance algorithm example 1

In the above test case example, system failed after flashing BIOS. but applying T2 recovery block system booted up and further execution proceeded.
Touchless toolkit is capable of monitoring and control of a device under test. In implementing fault tolerance system in Bios automation framework touchless toolkit plays a very important role. From monitoring the Postcode to implementing the recovery blocks ATK is used. Let us see the role of ATK 2 in implementing of recovery blocks.

- Warm reboot: ATK 2 front panel header is used in restarting the system.

- Cold reboot- Ac connect/disconnect cable is used to perform cold reboot on system.
- Reset Bios options: Clear CMOS jumper pin is used to set default bios options.
- Flash bios: Paddle board connector for flashing bios.

In one more example we will see the application of FTS system where the failure is same, system is not booting after flashing the bios.



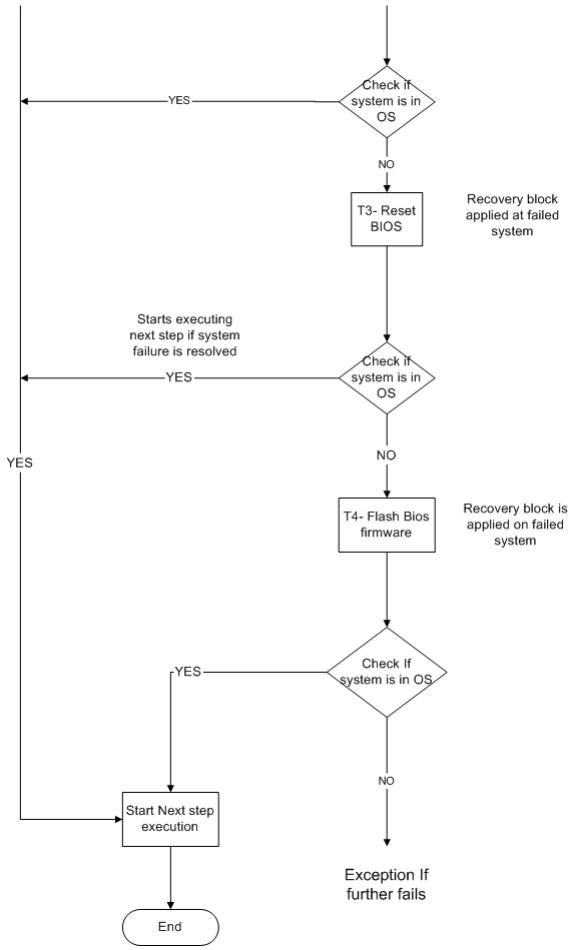


Figure 6.3: Fault tolerance algorithm example 2

We can see that after applying fourth recovery block system failure is handled. This surely takes time and processing. many times processing is not a questionable parameter but time certainly is. If one recovery block takes roughly 5 minute to execute, as the effort increases time utilized also increase. In second scenario FTS doesn't proves much effective. It requires enhancement in implementation technique to reduce time. this will be covered in next chapter system death level detection.

Chapter 7

System Death Level Detection

7.0.1 Introduction

Fault tolerance helps system function after failure also. This reduces the execution time due to any failure. Implementation of fault tolerance system can have advantages and disadvantages as well. Though it helps system fix the faults on its own but in a diverse system which can have varieties of failure can increase number of recovery blocks as well. This adds heavy overheads in the fault recovery scenario. It may successfully diagnose the fault and overcome it but the time taken would be far greater than required. In the end it may even not be successful in analyzing the correct fault and all the time utilized result in zero output.

This situation forces diverse systems to adopt smart fault tolerance system which can reduce the implementation of time on recovery block. Intelligence of FTS system can be increased by detecting the system death reason and then applying recovery block rather than implementing all the recovery blocks sequentially in a predefined order. As, not always we will face the same scenario and same sequence will prove beneficial.

7.0.2 Algorithm

The recovery blocks will be the same as we discussed in the last chapter just there sequence will change. How detection of the system death level takes place using POSTCODE will be discussed further in this chapter. After reading postcode, sequence of recovery block is decided which helps overcome failure at minimum iteration. It is as per the user how intelligently they club the postcode to define a sequence of recovery block to make work done with minimum efforts.

Postcode reader tool reads the postcode at the death time of the system. This works as an input for the sequence decider which gives sequence of recovery block as output.

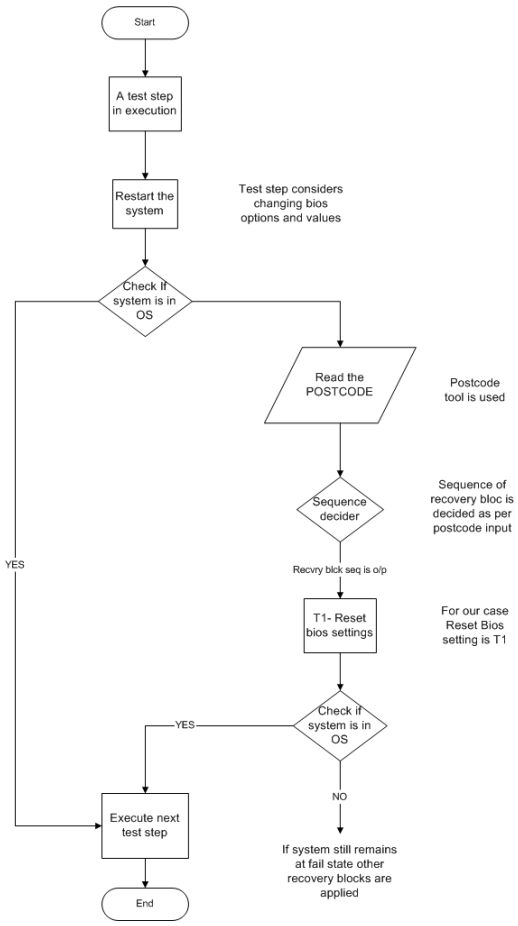


Figure 7.1: Death level detection algorithm

Recovery blocks are applied as per the run time defined sequence. The probability of removal of fault using system death level detection is much higher than implementing only FTS system.

7.0.3 Postcode Tool

POST (power-on-self test) is a diagnostic test that BIOS runs to check its hardware functionality. In proper operation and failure of the hardware bios returns a 4 digit Hex code i.e. POSTCODE. If self-diagnosis completes without any error system boots to operating system else specific error code is thrown. This gives in-depth information about system and system failure. So reading Post code of a system helps in diagnosis and implementation of the fault tolerance system.
Postcode tool helps read the postcode and decide the recovery block sequence. Postcode tool provides the sequence of recovery block as an output. Postcode tool has been written in C.Below is a code snippet of POSTCODE tool.

```
main(arg c, arg[v])
{
// Search for postcode in Databae//
postcode = search();

if(postcode ==true)
{
postcode_detail = describe(postcode);
printf(postcode_detail);
}

else
{
printf("Invalid postcode");
}

//decide sequence of recovery block

seq = seq_blck_(postcode);
f= fopen("seq.txt","a")
f.printf(seq)

// sequence is written in an outputfile
```

Figure 7.2: Postcode tool snippet

Below figure shows the white box representation of Postcode tool. The sequence of recovery block depends upon the past provided data to the tool using which it decides the sequence. Tool has a database which stores the postcodes information and what sequence has to be applied with respect to following achieved postcode.

Figure 7.3: Postcode tool white box

Currently system death level is implemented with basic understanding only. Further in-depth analysis can be done to increase the accuracy of recovery block sequence to decrease the fault run time. This requires more test case run and there failure analysis. This will eventually help in increasing the accuracy of the FTS system.

Chapter 8

Results

An every undergoing project is expected to give an outcome either it be success or failure. We can present result in qualitative as well as quantitative measure. This section will discuss the quantitative results that are and will be achieved by this project.
Number of test cases unblocked.

a. Bios flashing : 67 test cases

b. Relay control : 134 test cases

c. Postcode Reading : 48 test cases

d. Mac update : Enabled

Further test cases related to MAC update can be written.

Implementation of fault tolerance system will result in decrease in fault analysis and recovery time. With implementation of system death level detection in fault tolerance idle improvement in time is 80% but in practical we see improvement of 25% to 35% only.
Improvement in Automation framework with fault tolerance system.

a. It increases the probability of any system being available at any given time.

b. Extendible to handle 'N' number of crash scenarios.

c. Analysis done easy, reports and logs help debugging the failure.

Chapter 9

Conclusion

In this thesis work we had detailed study about the tool to be used and automation framework being used. For migrating from one version to another in depth analysis is required, of areas being effected and parts to be altered. This clears the cloud for porting process.

Different automation framework, importance of choosing correct automation framework as per requirement and how important can automation tools be, can be well understood from this thesis.Automation Tool kit v.2 has been implemented after doing the required script modification and unit testing in the automation framework. Enhancing the fault tolerance system using Automation tool kit was a challenge. Testing of system death level detection in bios automation framework with basic functionality has to be well structured and performed to to performance analysis of enhancement system.

Chapter 10

Future Scope

Automation tool kit is a controlling and monitoring purpose tool with wide utility. But as in result we saw not all features are being used currently after migrating from ATK legacy to ATK v.2.

The future work for this project can be implementing of the blocked features once they are compatible with the platform. Enhancement of the system death level detection can also be done to increase its database for accurate detection of the fault in the system.

Bibliography

[1] "What is bios"[Online], website, December,2016
$http://www.computerhope.com/jargon/b/bios.htm$

[2] "INTEL architecture white paper ",December 2016,
$IA-introduction-basics-paper.pdf$

[3] "What is gop"[online], Website, December 2016
$http://www.intel.com/content/www/us/en/intelligent-systems/intel-embedded-graphics-drivers/faq-bios-firmware.html$

[4] "Different Bios components"
$http://www.intel.com/content/dam/support/us/en/documents/motherboards/desktop/sb/b$

[5] "Automation Definition"
$http://www.oracle.com/technetwork/articles/entarch/shrivastava-automated-frameworks-1692936.html$

[6] "Types of automation Framework"
$http://www.softwaretestinghelp.com/test-automation-frameworks-selenium-tutorial-20/$

[7] "Intel advanced bios options" [Image][online], December 2016
$https://docs.oracle.com/cd/E19269-01/820-5830-13/app_bios.html$

[8] "Comparison between uefi and legacy"[online][Image],December 2016 $http://www.keywordsking.com/dWVmaSB2cyBiaW9z/$

[9] "Xiorant automation Framework"[online][Image], November 2016
$http : //www.xoriant.com/brochures/xoriant - test - automation - framework - xtaf$

[10] "Python"[Online], December 2016
$https : //en.wikipedia.org/wiki/Python_{(programming_language)}$

[11] "Intel documents"

[12] "Distributed system architecture"[Online][Image],
$http : //www.iue.tuwien.ac.at/phd/weinbub/dissertationsu16.html$

[13] "@article aksu 2005 fault, title=Fault Tolerance in Distributed Systems, author=Aksu, Naima, journal=Term Project, year=2005"