“Boot time Optimization for Single Socket Workstation”

Project Report

Submitted in Partial Fulfillment of the
Requirements for the Degree of

Master OF TECHNOLOGY

IN

Embedded Systems

By

Nayana Maheshbhai Patel
(19MECE12)

Department of Electronics and Communication Engineering,
Institute of Technology,
Nirma University,
Ahmedabad 382 481

May 2021

CERTIFICATE

This is to certify that the Major Project Report entitled "Boot time Optimization for Single Socket Workstation" submitted by Ms. Nayana Maheshbhai Patel (19MECE12) towards the partial fulfillment of the requirements for the award of degree in Master of Technology in the field of Embedded systems of Nirma University is the record of work carried out by him/her under our supervision and guidance. The work submitted has in our opinion reached a level required for being accepted for examination. The results embodied in this major project work to the best of our knowledge have not been submitted to any other University or Institution for award of any degree or diploma.

Date: 16th May 2021

Yasmin Tabassum

Manager
Datacenter Firmware Development (DFD)
Intel Corporation
India

Dr. Nagendra Gajjar

Professor,
Coordinator M.Tech EC
Institute of Technology
Nirma University

Dr. Dhaval Pujara

Professor & Head of Department
Department of Electronics & Communication Engg.
Institute of Technology
Nirma University
Ahmedabad

Dr. R. N. Patel

Director
Institute of Technology
Nirma University
Ahmedabad

Undertaking for Originality of the Work

I, Nayana Maheshbhai Patel, Roll No. 19MECE12 , give undertaking that the Major Project entitled "Boot time Optimization for Single Socket Workstation" submitted by me, towards the partial fulfillment of the requirements for the degree of Master of Technology in Embedded Systems of Nirma University, Ahmedabad, is the original work carried out by me and I give assurance that no attempt of plagiarism has been made. I understand that in the event of any similarity found subsequently with any other published work or any project report elsewhere; it will result in severe disciplinary action.

Nayana Patel

Date: 16th May 2021

Endorsed by: Dr. N. P. Gajjar

Acknowledgement

It gives me great delight in expressing thanks and gratitude to Prof. Nagendra P. Gajjar, Co-ordinator of Embedded systems, Electronics and Communication Engineering Department, Institute of Technology, Nirma University, Ahmedabad for his significant guidance and consistent support all through this work. The continual assistance and support he has given has been an incredible inspiration to me in achieving more significant goal.

It gives me an immense pleasure to thank Dr. Dhaval Pujara, Honorable Head of Electronics and Communication Engineering Department, Institute of Technology, Nirma University, Ahmedabad for his kind support and providing basic infrastructure and healthy research environment and guidance.

I would also like to thank my mentor Parin Chheda (BIOS Firmware Engineer) and manager Yasmin Tabassum (BIOS Firmware Engineering Manager) for their continuous support and guidance for my M.Tech thesis project at Intel Corporation.

It was a pleasure to be associated with the Intel corporation, and I would like to thank the entire staff.

Nayana Patel

19MECE12

ABSTRACT

Technology is changing day by day and with that performance and efficiency of computers is increasing at unbelievable rate. Performance of the computer is defined by speed and accuracy of the processors. BIOS is the crucial part of system initialization. It is responsible for executing all the processes like detecting and initializing memory, initializing different I/O devices, communication devices, enumerating devices on different buses and all the other drivers and then handing off the control to OS.

Nowadays UEFI has taken place of BIOS. Legacy BIOS is based upon Intel's original 16-bit architecture, BIOS was designed differently for different boards which in turn increases the cost of the new product. So, it is replaced by Unified Extensible Firmware Interface – UEFI which provides utility to use unified code structure which can be common for different platforms.

As we know that speed is the one of the most important factors that affect the performance of the system, we target minimum possible boot time for systems. Many efforts have made earlier for boot time optimization on different platforms and implemented successfully.

To reduce the boot time required by system, this project is divided in two phases: Analysis and implementation. As part of analysis phase the current boot time is taken and note the modules taking more time to execute. After analysis see that what is the purpose of the module and see if we can discard or reduce the module. The boot time is affected based on the type of boot flow of system and some system configurations also change boot time considerably.

INDEX

LIST OF FIGURES

LIST OF TABLES

NOMENCLATURE

Abbreviations

IAGS	Intel Architecture Graphics and Software
BIOS	Basic Input/Output System
UEFI	Unified Extensible Firmware Interface
SFP	System Firmware Products
DFD	Datacenter Firmware Development
SEC	Security Phase
PEI	Pre – EFI
FV	Firmware Volume
FFS	Firmware File System
FD	Firmware Device
HOB	Hand Off Block
DXE	Driver Execution Environment
BDS	Boot Device Selection
ACPI	Advanced Configuration and Power Interface
FPDT	Firmware Performance Data Table
MRC	Memory Reference Code
RMT	Rank Margin Test
OPROM	Option ROM

Chapter 1: Introduction

1.1 Company Profile

Intel is the U.S. based Multinational organization and head quarters are situated in Santa Clara, California. It is one of the most valued company in the semiconductor market based on the revenue. It is the developer of variety of microprocessors of x86 series, which was having enormous market in 1990s. Intel secured 12th rank in Forbes most valued 100 companies and secured 24th rank in Fortune 500 companies by revenue.

Major line of products of Intel are microprocessors, Integrated circuits, Memory chips, embedded processors, Network Interface controllers, System on chip (SoC), Integrated Graphics Processing Units (IGPU), solid state drives (SSD) and many devices associated with Computation and Networking. It provides solutions for mobile, computers and servers. Intel is now focusing on providing solutions for data center with the use of AI across smart devices.

On July 18, 1968 Intel was founded by Robert Noyce and Gordon Moore who were experts in semiconductor field. Intel was originally called N M Electronics. Intel was market leader for SRAM and DRAM throughout 1970s then they introduced x86 processor which was big success of the company.

Intel's vision is to become reliable performance groundbreaker by using the potential of data and mission is to provide solutions to customers with reliable, cloud to edge computing inspired by Moore's law. Intel values are One Intel, Inclusion, customer obsessed, transparency, quality and truth [2].

In PC microprocessors major competitors of the Intel are Nvidia, Advanced micro devices (AMD), Silicon Integrated systems and VIA Technologies. NXP Semiconductors, Infineon, Broadcom Limited, Marvell Technology Group are major competitors in Networking solutions.

1.2 Group Profile

Intel Architecture, Graphics and Software – IAGS

Intel Architecture, Graphics, and Software (IAGS) group is responsible for providing innovation in Architecture, software and Graphics solution by driving excellence in Memory, Processing power, connectivity, hardware and software optimization as well as security [3].

The vision of the IAGS group is to provide a combination of Vector, Scalar and Spatial architectures designed with modern process technology which are integrated into systems with cutting-edge packaging that can be deployed with high speed communication and can be executed with unified single software with high level security.

System Firmware Products - SFP

System Firmware Products group provide solutions for scalable system firmware for client and data center platforms. It also provides security tools to verify silicon IP and security settings. It is responsible for developing BIOS firmware and provide automation for unit testing and validation for BIOS, IFWI and security. It provides Continuous Integration (CI) infrastructure with review and post-merge checks. It aims to deliver qualified UEFI FW and IFWI releases for server & client products.

Datacenter Firmware Development - DFD

Datacenter Firmware Development group provide solutions particularly for servers. This group delivers solutions like BIOS, IFWI and security for server platforms. It is responsible to deliver UEFI FW and IFWI releases for server products.

1.3 Introduction

Technology is changing day by day and with that performance and efficiency of computers is increasing at unbelievable rate. Performance of the computer is defined by speed and accuracy of the processors. BIOS is the crucial part of system initialization. It is responsible for executing all the process like detecting and initializing memory, initializing different I/O devices, communication devices enumerating PCI devices and all the other drivers and then handing off the control to OS.

Nowadays UEFI has taken place of BIOS. Legacy BIOS is based upon Intel's original 16-bit architecture, BIOS was designed differently for different boards which in turn increases the cost of new product. So, it is replaced by Unified Extensible Firmware Interface – UEFI which provided utility to use some common features.

As we know that speed is the one of the most important factors that affect the performance of the system, we target minimum possible boot time for systems. Many efforts have made earlier for boot time optimization on different platforms and implemented successfully. We can optimize the code for each phase SEC, PEI, DXE and BDS of UEFI framework for single socket workstation.

1.3.1 Legacy BIOS

Basic Input/Output System (BIOS) is the dominant standard which defines a firmware interface. "Legacy" in the context of firmware specification refers to an older, widely used specification. The main responsibility of BIOS is to set up the hardware, load and start an operating system (OS). When the computer boots the BIOS initializes and identifies system devices including the video cards, keyboard, mouse, hard disk and other hardware. The BIOS then locates software held on a boot device for example a hard disk or CD/DVD – and loads and executes that software giving it control of the computer. This process is also called as "Booting" or "Boot strapping" [4].

Legacy BIOS is based upon Intel's original 16-bit architecture commonly referred to as 8086 architecture and as technology advanced Intel extended that 8086 architecture from 16 to 32 bit.

1.3.2 UEFI – Unified Extensible Firmware Interface

UEFI was developed as a replacement for Legacy BIOS to streamline the booting process and act as the interface between operating system and its platform firmware. It not only replaces most BIOS functions, but also offers a rich extensible pre-OS environment with advanced boot and runtime services. UEFI architecture allows users to execute applications on a command line interface. It has intrinsic networking capabilities and is designed to support multi-processors systems. The UEFI is responsible for setting up or producing the services and protocols that can be consumed by other software or firmware components.

The UEFI Forum board of directors consists of representatives from 11 industry leaders as described in Figure 1. These organizations work to ensure that the UEFI specifications meet industry needs [4].

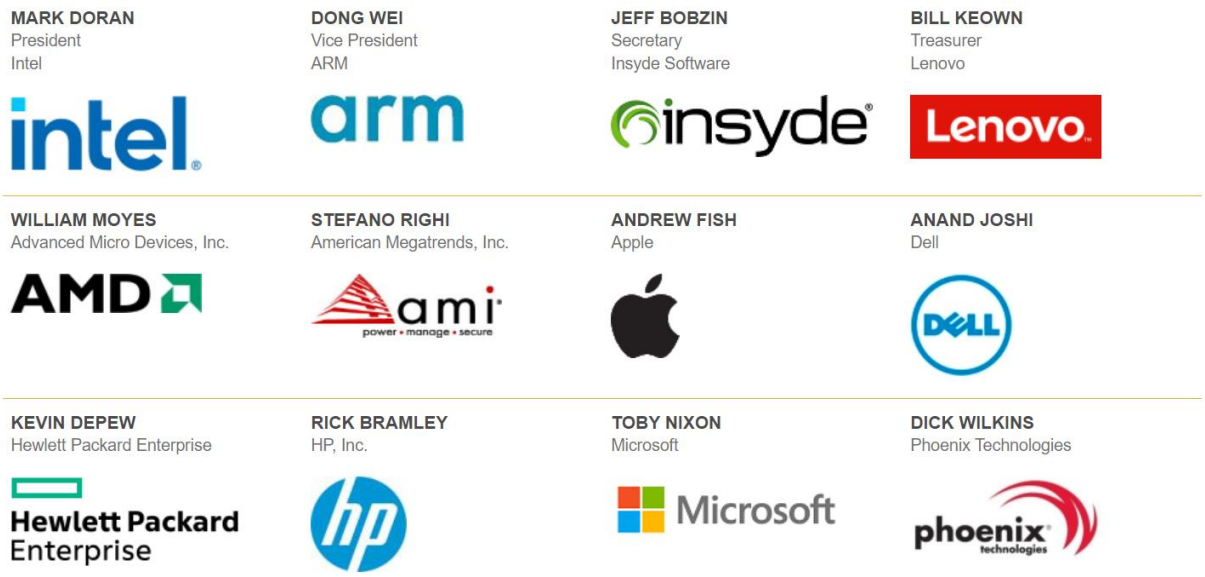

Figure 1: Board of Directors of UEFI Forum

UEFI uses a different interface for runtime services and boot services but UEFI does not specify how" Power on Self-Test" (POST) and Setup are implemented those are BIOS' primary functions.

One major problem with existing BIOS implementations is that since they are highly customized for a specific motherboard, there maintenance is difficult. A lot of effort is required in significant porting, integration, testing and debug work of changes in component modules. The UEFI architecture is designed to considering these limitations and to resolve them.

1.3.3 UEFI Driver Model Extension

Boot devices are accessible via a set of protocol interfaces. The UEFI Driver Model was not designed to replace the high-performance OS specific drivers but to access boot devices in the pre-boot environment, to support the execution of modular pieces of code, also known as drivers. These drivers control hardware buses and devices on the platform, and, they may provide some software-derived, platform specific service. The information

required by the driver developers for implementing combination of bus drivers which boot an UEFI compliant OS are included in the UEFI Driver Model. Thus, the UEFI Driver Model is designed to be generic.

1.3.4 Legacy BIOS vs UEFI (refer Table 1)

Legacy BIOS	UEFI
Written in Assembly language	Written in C language
Environment is not modular	Environment is modular
Uses 16-bit memory access	Allows memory access via 32-bit or 64-bit pointers
BIOS can only boot from drives having size 2TB or less --- mention reason or verify	UEFI can boot from drives having size less than 9.4 ZB
No built-in boot/test environment	Built-in boot/test via EFI shell
Non standardized implementation hence difficult to share	Standardized implementation hence easily sharable

Table 1: Legacy BIOS vs UEFI

1.4 Scope of Training

1) UEFI and EDK II Training

In this training we learned basic architecture and specifications of UEFI.

EDK II is a modern, feature-rich, cross-platform firmware development environment for the UEFI and UEFI Platform Initialization (PI) specifications.

In this training we learned the following:

- Creating the PEIM module
- Creating the DXE-Driver module
- Adding and Removing the setup knob

- Building IFWI
- Creating simple EFI application

2) Git Training

Git is the most well-known implementation of a distributed version control system (VCS) framework. The Git core was initially written in the C programming language, yet Git has likewise been reimplemented in different programming languages, for example, Java, Ruby, and Python. Git provides us facility to clone any source code to which we have access. We can create local branch and do the additional changes on the source code cloned from master branch. We can do commit changes on the local branch.

The training included the basic following aspects:

- Clone the source code
- Commit the changes on local branch
- Merging local branch with master branch
- Create branches and resolve merge clashes
- Sending for code review
- Cherry pick the code
- Revert back changes done
- Git GUI usage

3) Simics Training

Simics is a full system simulator which is used to run the binaries for target hardware with high performance. Right now simics is marketed by Wind River Systems. Simics is used to test target IFWI. [1]

In simics following things are possible:

- We can debug the code
- Can connect the console to target and use the console as input or output device
- We can put break points and halt the system on particular point for debug purpose
- We can read and write register values

1.5 Organization of Report

Chapter 1 describes all the basic information required on BIOS history and UEFI. Chapter 2 gives overview of literature study for boot time optimization. Chapter 3 involves the design of the Basic Boot Flow of the Platfrom Initialization framework of Intel, Generic build process and architecture of UEFI. Chapter 4 gives the overview of efforts made in boot time reduction. Chapter 5 gives results and discussion about boot time reduction and Chapter 6 shows future scope of the project.

Chapter 2: Literature Review

Sources	Topics Learned
Beyond_BIOS_Second_Edition_Digital_Edition_(15-12-10)	BIOS Flow
edk2-UefiDriverWritersGuide-draft	Module structure
UEFI_Spec_2_3_1	UEFI specification
PI_Spec_1_6	Platform Initialization s[ecification
UEFI_Shell_2_2	UEFI shell specification

Table 2: Literature Review

2.1 Boot Optimization

Basic Input/Output system (BIOS) is the dominant standard which defines a firmware interface. "Legacy" in the context of firmware specification refers to an older, widely used specification. The main responsibility of BIOS is to set up the hardware, load and start an operating system (OS).

UEFI was developed as a replacement for Legacy BIOS to streamline the booting process and act as the interface between operating system and its platform firmware. It not only replaces most BIOS functions, but also offers a rich extensible pre-OS environment with advanced boot and runtime services. UEFI architecture allows users to execute applications on a command line interface.

The boot time depends on hardware and software specifications. Before doing any efforts for boot time optimization. We need to answer following questions [5]:

- What are the design goals?
- Pre-OS user interaction is needed?
- Can we use system configuration from last boot?
- What are the targeted operating system?
- Do we need Legacy BIOS support?
- Do we need to support Option ROMs?
- What is the BIOS recovery strategy?

Based on the Platform architecture and requirements we can reduce the boot time. EDK II provides a library PerformanceLib.h which is provided as part of MdePkg - edk2/MdePkg/Include/Library/PerformanceLib.h. This library provides some performance macros by which it can log the performance data of each module in the FPDT table of the ACPI data tables.

This library provides some performance macros by which it can log the performance data of each module in the FPDT table of the ACPI data tables.

Chapter 3: Design and Architecture

3.1 Design

3.1.1 Platform Initialization

The Intel Platform Innovation Framework for UEFI, also referred to as a "The Framework", was the authoritative firmware implementation that conformed to the UEFI specification. This framework is implemented in C.

The framework has been replaced with the more comprehensive UEFI Platform Initialization (PI) specification that complements the UEFI specification. The flow of PI can be seen in the Figure 2. UEFI specification is a firmware implementation designed to perform the full range of operations required to initialize the platform from power-on through the transfer of control to the operating system [6].

3.1.2 Platform Initialization (PI) Boot Sequence

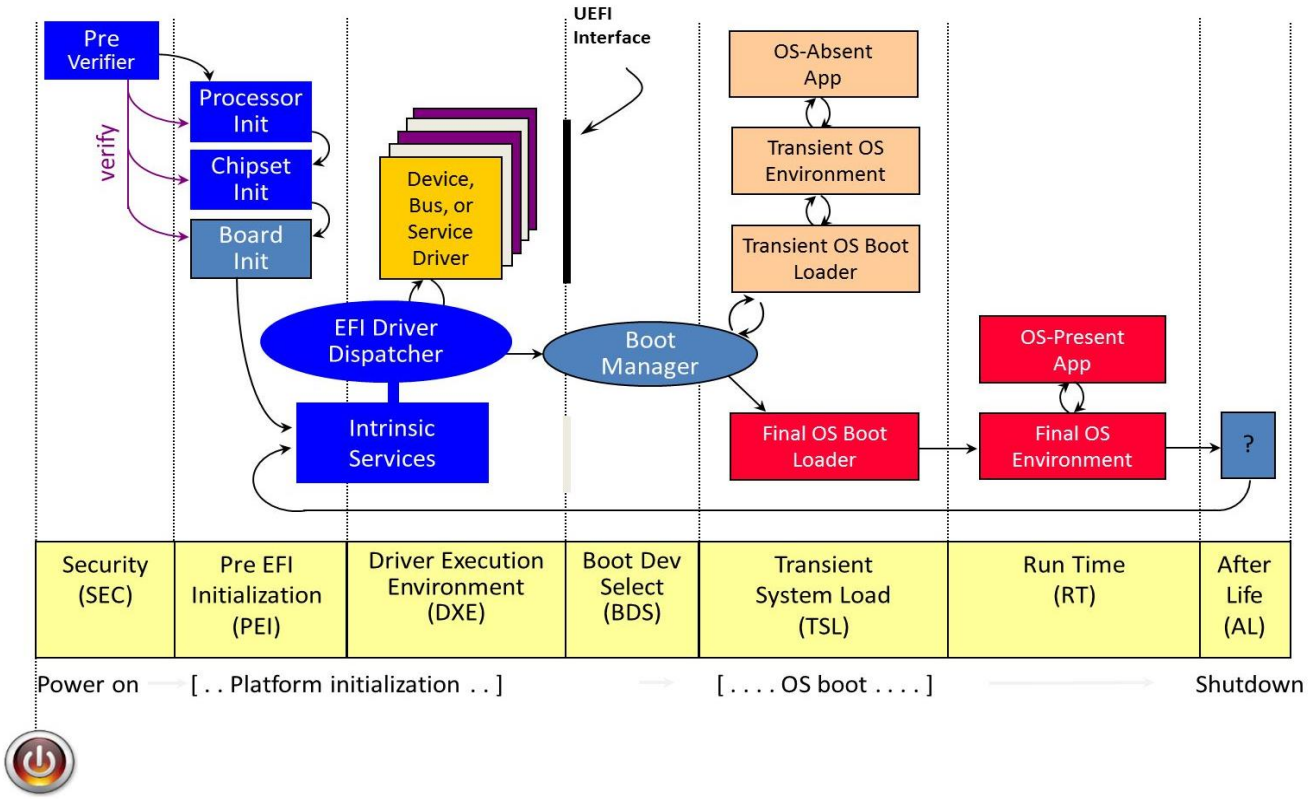

Figure 2: PI Boot Flow

Platform Initialization PI compliant system firmware must support the six phases:

1. "Security (SEC)" Phase
2. "Pre-EFI Initialization (PEI)" Phase
3. "Driver Execution Environment (DXE)" Phase
4. "Boot device selection (BDS)" Phase
5. "Run time (RT)" services
6. "After Life (AL)" of system.

3.1.2.1 Security - SEC Phase

Security (SEC) phase is the initial phase through which the boot flow begins. The SEC phase also passes optional information in addition to the minimum architectural information such as processor health or the SEC platform information [6].

This phase is responsible for the following:

- Handling restart events of every platform
- Initializing cache as RAM.

- Establish the trust root in the system
- Transit handoff content to next phase - the “PEI Foundation”

SEC phase is mainly written in assembly language because it is dependent on platform architecture.

3.1.2.2 Pre EFI - PEI Phase

Pre EFI-Phase is responsible for initializing permanent memory. After about preliminary processing of the Security (SEC) phase, any system restart event will bring up the PEI phase [6].

The PEI phase mainly consists of two parts:

- PEI Foundation (core code)
- Pre-EFI Initialization Modules - PEIMs

The PEI phase at first operates along the platform in a developing stage, holding only resources on processor, such as the cache for maintaining call stack and dispatching the “Pre-EFI Initialization Modules (PEIMs)”.

These PEIMs are responsible for:

- Initializing permanent memory
- Passing the information about platform and the main memory in the form of Hand-Off Blocks (HOBs)
- Passing the location of the firmware volume in HOBs
- Transit the control flow into next phase - the Driver Execution Environment (DXE) phase.

The PEI phase cannot presume the convenience of amount of main memory (RAM) as DXE and hence PEI phase support is limited to:

- Locates and validates PEIMs
- Dispatches of PEIMs
- Facilitates commuting of PEIMs
- Provides platform specification to later phases

3.1.2.3 Driver Execution Environment - DXE Phase

Most of the system initialization is performed in DXE phase. The DXE can use the permanent memory that is initialized by PEI phase. The DXE phase is the phase where the bulk of the booting occurs: devices are enumerated and initialized.

Following are DXE phase components [6]:

1. Drivers: Drivers are responsible for initializing the processor, chipset and platform components. More specifically, DXE drivers are responsible for providing Architecture Protocols (APs) that abstract the DXE core from the platform.
2. Foundation: The foundation is the main DXE executable binary responsible for dispatching drivers and producing a set of boot services, runtime services and DXE services.
3. Architectural Protocols (APs): DXE drivers produce APs to abstract DXE from hardware. Only the DXE foundation may call APs.
4. UEFI system table: The UEFI system table contains pointers to all the UEFI service tables, configuration tables, handle databases and console devices.
5. Dispatcher: The dispatcher is the part of the DXE foundation that searches for and executes the drivers in the correct order.

UEFI services are supported and the protocols and drivers are implemented. Also, the tables that create the UEFI interface are produced. The state of all the hardware and resources is passed to DXE phase through a list of Hand-off-blocks. The DXE foundation uses Architecture Protocols (APs), which are DXE drivers that abstract access to the hardware.

3.1.2.4 Boot Device Selection - BDS Phase

Boot Device Selection (BDS) is responsible for determining how and where you want to boot the operating system (OS) [6].

Primary steps of the BDS phase:

- Initialize language and string database
- Build device list
- Detect console devices
- Process boot options

3.1.2.5 Transient System Load (TSL) and Runtime (RT)

Primarily the OS vendor provides boot loader known as the “Transient System Load (TSL)”. TSL and Runtime Services (RT) phases may allow access to persistent content, via UEFI drivers and applications. Drivers in this category include PCI Option ROMs.

3.1.2.6 After Life (AL)

The After Life (AL) phase contains the persistent UEFI drivers used to store the state of the system during the OS systematically shutdown, sleep, hibernate or restart processes.

3.1.3 Generic Build Process

In UEFI all the source code is generally written in c source and header files. Some codes in SEC and PEI section are written in assembly code. According UEFI specification all the source code should be compiled and linked to generate PE32 and PE32+ images. As there are some codes in PEI modules that are executes from cache as RAM and some modules execute after memory initialization from RAM. So, these source codes are relocatable. So, some code can be executed from both modes. It will execute from cache if cache memory is enough otherwise it will execute from ROM. The relocation section may or may not be stripped and an appropriate header will replace the PE32/PE32+ header [7]. Non - essential information will be removed to generate a Terse image (TE). The binary executables are converted into EFI section consisting of a section header followed by the section data.

3.2 Architecture

3.2.1 Firmware Storage

BIOS image can be interpreted by the file system which is designed in a byte format which can be understood by low level programming language. Platform can be initialized by executing this BIOS image which is stored in flash device. Firmware storage provides basic infrastructure for accessing file.

Firmware storage is nonvolatile memory that stores the BIOS or firmware code. One of the advantages of UEFI is that the DXE and PEI core codes know how to extract code out of

firmware storage. The smallest unit of firmware storage is UEFI firmware files which is located within Firmware File system as you can see in the Figure 3, which is located in Firmware Volume which are stored in Firmware device [4].

- UEFI Firmware Files: These are smallest and least modular code and UEFI build tools create the firmware files.
- Firmware File Systems: Describes the organization of files and free space within the FV. Each FFS has a unique GUID used to associate a driver with a newly exposed FV.
- Firmware Volume: FV is logical firmware device. Each FV is organized into file system. FVs are used to describe a logical representation of a physical storage device. FVs are not device specific.
- Firmware Device - FD: Firmware device is non – volatile storage component and it is divided into various areas like firmware code, firmware data and fault tolerant. This include properties to program the device.

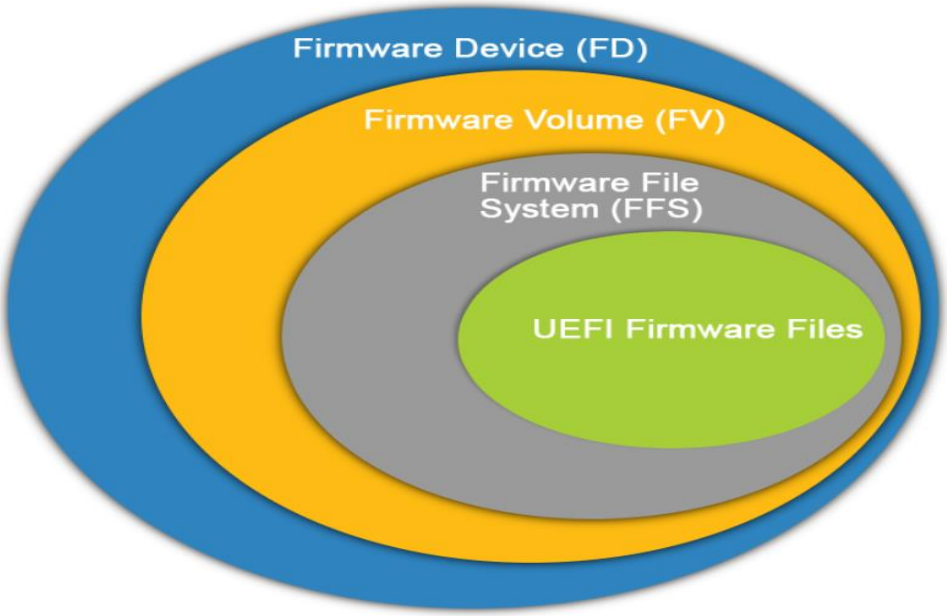

Figure 3: Firmware storage system

3.2.2 EDK II Packages

EDK II provides package utility that is a directory that arranges a collection of units with a single package declaration file. Each package is having directory structure in unified way

that separate the different source files. The package contains .dec, .dsc, .inf and .vfr files. EDK II provides common packages like MdePkg, ShellPkg, MdeModulePkg, etc.

Each package defines public interfaces. The DEC files has Defines, Includes, Library Classes, Guids, Ppis, Protocols and Pcds sections.

Each package in EDK II usually have description file (DSC). Any modules that are added in DSC files are compiled and verified. Only active platform DSC file is built [8].

EDK II packages also provide some common libraries like PrintLib, BaseMemoryLib, DebugLib, PciExpressLib, TimerLib and many other which provide basic utility for creating and debugging module.

Chapter 4: Implementation

4.1 Proposed Work

BIOS image is stitched with other ingredients which generate IFWI (Integrated Firmware Image) file. This IFWI file which is binary file is flashed to the target device. This binary file is executed from the SPI flash when we power on the system. Each phase is executed in sequence like SEC, PEI, DXE and BDS respectively and different drivers are loaded like DXE driver, PEI driver and other EFI applications. These phases take considerable time to complete the tasks. Time taken by thess phases can be reduced by some amount by removing or skipping the execution of some modules. To achive this goal the work is divided in two phases: Analysis and Implementation. To reduce boot time of any system we need to see current boot time and areas where we can do modifications depending on platform to reduce that time. In implementation we can change some flow of module or optimize code in such a way that can give us reduction in boot time.

4.2 Analysis of Boot Time

4.2.1 Getting Boot Time Data

To reduce the boot time required by system, we need to get current boot time of the system. To get the performance of the system EDK II provides some libraries which can give us the time taken by each module.

EDK II provides a library PerformanceLib.h which is provided as part of MdePkg – EDKII. This library provides some performance macros by which it can log the performance data of each module in the FPDT table of the ACPI data tables [9]. To record the time taken by each

The ACPI – Advanced Configuration and Power Interface is the specification which comes under UEFI specification to develop common industry interface to enable the operating system to do the power management of the entire system and devices connected with the system. ACPI system description tables contains many tables from which FPDT is one of the system description table [10].

The FPDT – Firmware Performance Data Table contains information about performance records of platform initialization process [10]. This information includes the data of the performance for specific tasks in platform initialization boot process. This FPDT table only provides information that is common in every platform initialization boot process. The 64 bit performance counter is used to note the start and end time of the modules and that data is recorded in FPDT table format. This table contains performance data starting from reset vector to handoff to the OS loader. We can track the performance of the system resulting from changes in software or hardware configuration for each phase of UEFI which is very helpful to study the system performance.

EDK II provides some common performance library instances which are used to log the performance data. Following are performance library instances provided by EDK II [9]:

- PeiPerformanceLib: Provides infrastructure for PEIMs to log performance data.
- SmmPerformanceLib: Provides infrastructure for SMM drivers to log performance data.
- SmmCorePerformanceLib: This is used by SMM core to log performance data.
- DxePerformanceLib: Provides infrastructure for DXE drivers to log performance data.
- DxeCorePerformanceLib: This is used by DXE core to log performance data.

EDK II provides shell application utility which can be executed from shell command interface. To retrieve the performance data, UEFI shell command – "DP" is used which is provided by EDK II platform. This command can be implemented by including appropriate .inf files from shellPkg into active platform .dsc files and .fdf files and set PcdPerformanceLibraryPropertyMask = 1.

4.3 Implementation

4.3.1 BIOS image build

There are two types of BIOS images:

- Debug BIOS
- Release BIOS

Debug BIOS is used by developers to debug the module. Debug BIOS will print the debug messages written in code for debug purpose. We can also give print error level specification according to our requirement.

Release BIOS is having minimal debug prints which reduces the time required by BIOS to boot to OS or shell significantly.

Standard boot time of the bios is considered as boot time of release bios.

4.3.2 Coldboot and warmboot

Coldboot – System starting from powerless state or shutdown. This follows the whole path of system initialization.

Warmboot – System start when reset is given from the system. This skips some modules compared to coldboot [11].

Starting from When BIOS starts executing, memory is not initialized in the SEC phase. In SEC phase the cache is used as main memory from where BIOS starts execution this concept is called “Cache as RAM”. In Pre-EFI phase, memory is initialized and other phases implemented from RAM. In the PEI phase, MRC - Memory Reference code is executed which initializes the memory. In MRC code, memory initialization and training is carried out and then memory testing is carried out. We can skip the testing of memory to reduce the boot time in warm boot flow. So, we can skip some part of MRC in warm boot path or use the parameters from previous boot for MRC, as at the time of first boot – cold boot the MRC is executed fully. In cold boot path we need to implement whole MRC process because at that time no memory is initialized, in warm boot we can skip the MRC because it is initiated in cold boot already.

4.3.3 Support for OPROMs

Option ROM is firmware which generally resides in BIOS or on an expansion card.
Video cards, storage drivers and network adaptors for RAID modules generally requires OPROMs. These OPROMs provide driver services to initialize drivers for this devices. We can choose to load OPROMs depending on the need of the device at boot time. If we want to boot from the device whose firmware driver is loaded in OPROMs, they only we have to initialize the OPROMs at boot time otherwise we can load that OPROM after OS takes control [12].
Based on the platform need we can either optimize the module or remove the module. By removing some unnecessary modules, we can save boot-time.

- Removed some OPROMs from IFWI depending on the platform need, this gives some reduction in boot time.

4.3.4 Disabling Rank Margin Test – RMT

Intel has developed a tool called the DDR Rank Margining Tool (RMT) which is used to identify DDR margins at the rank level and offers automated margin testing of the DDR memories. The RMT is part of BIOS - Memory Reference Code(MRC) and can be enabled or disabled according to need. It is embedded in BIOS through the MRC and executes during startup of the system. It offers automated margining of Timing parameters and DDR Vref at the CPU and DIMMs with applied stress patterns. For single socket system this test is not necessary as there is only one socket which will be communicating with memory, So, for that parameters would be fixed no need to invoke RMT for calibration.

4.3.5 Fast boot enable

Fast boot is the feature adopted by the Intel which can saves time by adopting optimized path to boot. Fast boot can be enabled in warmboot path as well as coldboot path.
If fast boot is enabled:

- Boot from Network, Optical and removable devices are disabled
- Configuration data is retrieved from last boot which resides on the NVRAM
- Memory training is skipped as part of MRC for fast warm boot

- In Fast cold boot path if memory population is changed the system will retrain the memory and will save the data for next boot
- New consoles can't be added and initialized

4.3.6 Dual and Quad SPI mode

The Binary image of IFWI is loaded on the SPI flash which is connected to south bridge. From SPI flash the boot process starts and fetches the initial data.

SPI can support three modes as following [13]:

1. Normal mode
2. Dual I/O mode
3. Quad I/O mode

1) Normal mode (refer Figure 5):
   - Normal mode works in full duplex mode
   - Only one bit is transferred with each clock cycle as seen in Figure 4: Simple SPI Time DiagramFigure 4

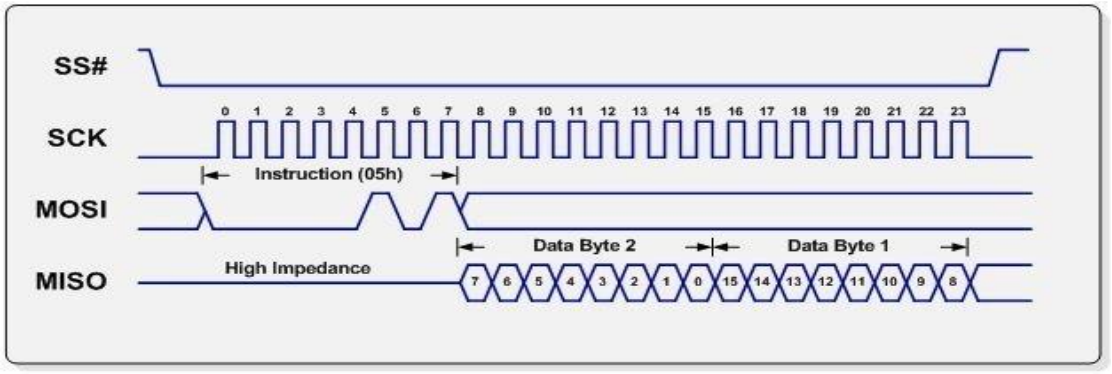

Figure 4: Simple SPI Time Diagram

2) Dual I/O read enable:
   - Dual mode works in half duplex mode with two data lines SIO0 and SIO1.
   - Two bits are transferred with each clock cycle.
   - Address will also be transferred two bits at a time on both lines.

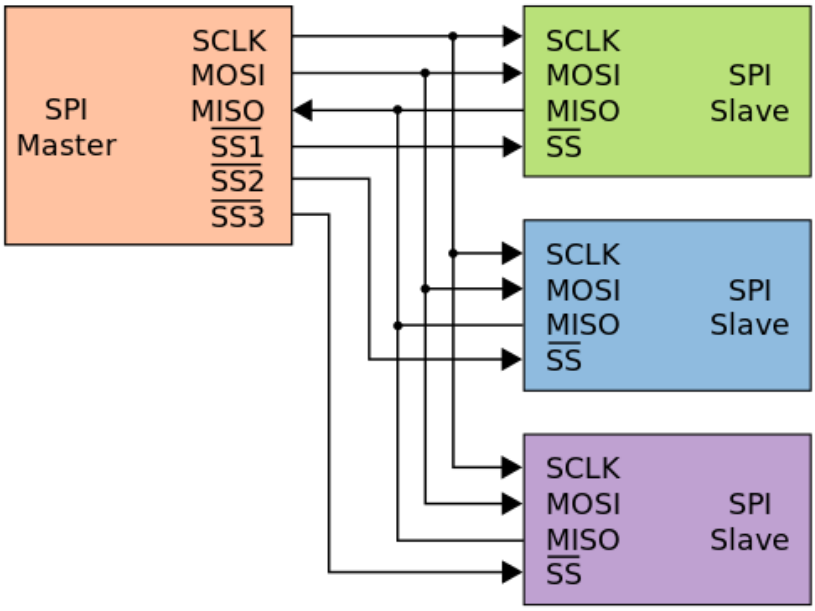

Figure 5: SPI connection diagram for normal mode

3) Quad I/O read enable:

- Quad mode uses 4-data pins SIO0, SIO1, SIO2, SIO3
- Four bits are transferred with each clock cycle in half duplex mode.
- Address will also be transferred four bits at a time on four data lines.

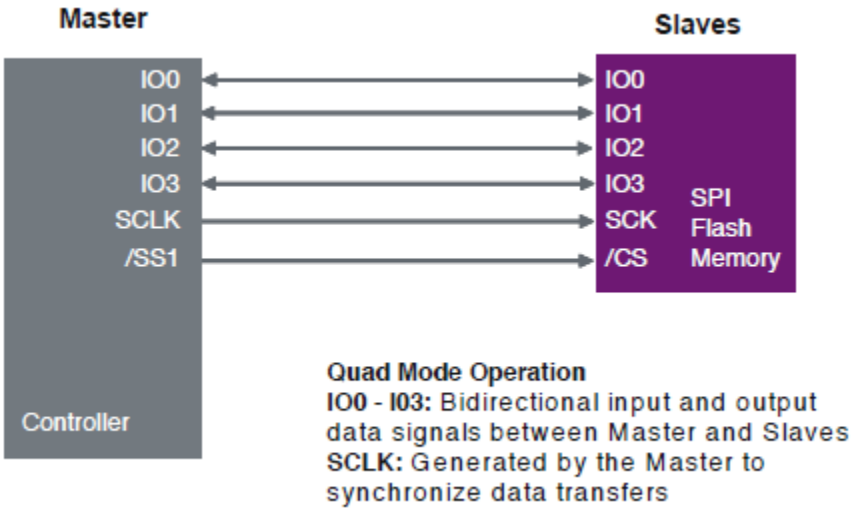

Figure 6: Quad mode SPI timing diagram

This Flash configuration settings can be changed by software used to generate IFWI combining all the components as seen in Figure 7.

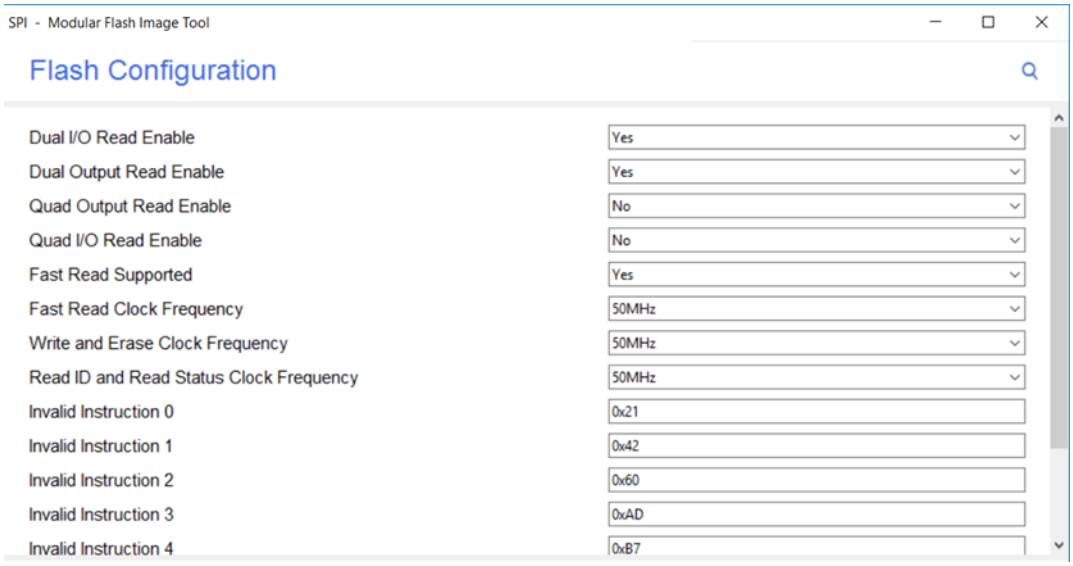

Figure 7: Flash configuration setting

4.3.6 Disable multi socket support

As name suggests single socket work station is having only one socket. In multi socket system more than one sockets are connected with each other with KTI links which are responsible for inter socket communication. For single socket communication no need to establish this inter socket communication links. To establish the proper communication links it takes time to train these links to give best performance and speed. So, for single socket thes time can be saved as there is no need to setup links.

Chapter 5: Results and Discussion

5.1 Results

The boot time can be reduced based on the platform and on use case of the platform. Before implementing anything for boot time reduction we must analyze the boot time for BIOS on platform and need to optimize the code and module whenever it is possible.

- Implemented shell command in UEFI shell to get performance data as can be seen from the Figure 8.

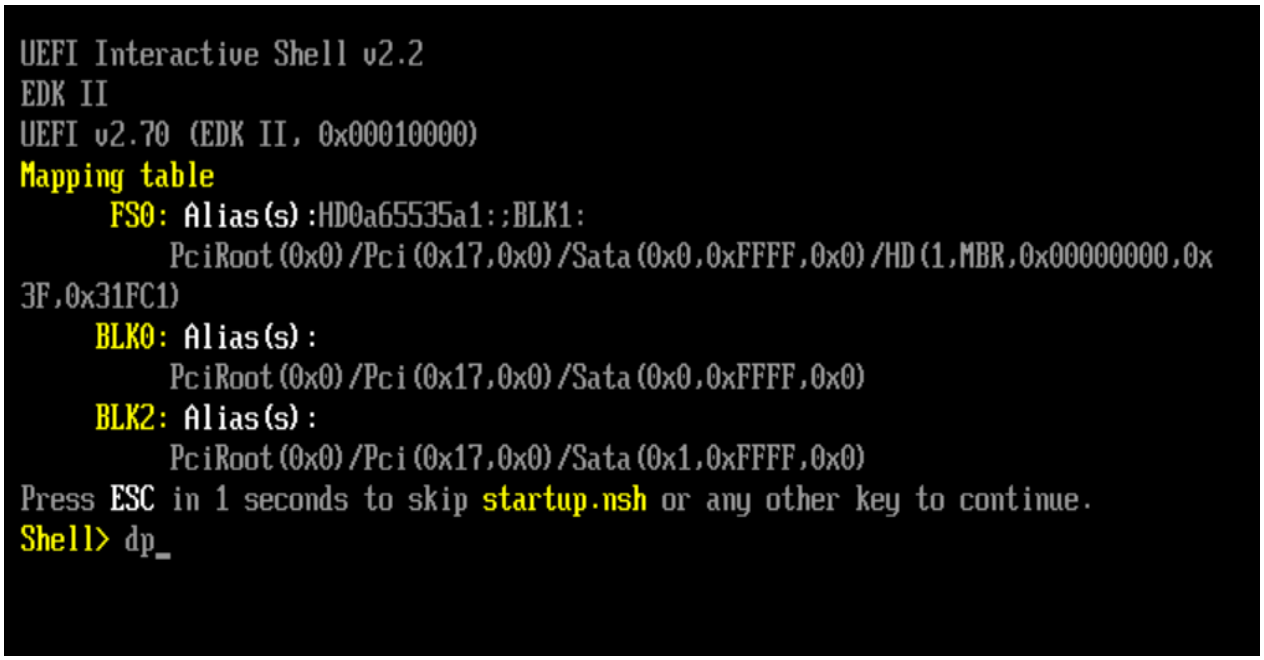

Figure 8: UEFI shell command – DP

- Extracted the boot time for release bios and debug bios for cold boot and warm boot. Time taken by particular module can be known by using micros defined in How much time is taken by particular module can be taken by using macros which are defined in performanceLib.h as seen in
- Extracted the boot time for release bios and debug bios for cold boot and warm boot. We got the ~20% reduction in boot time with release bios compared to debug bios.
  In release IFWI, time is reduced because no need to print any debug statements which reduces the write to uart console.

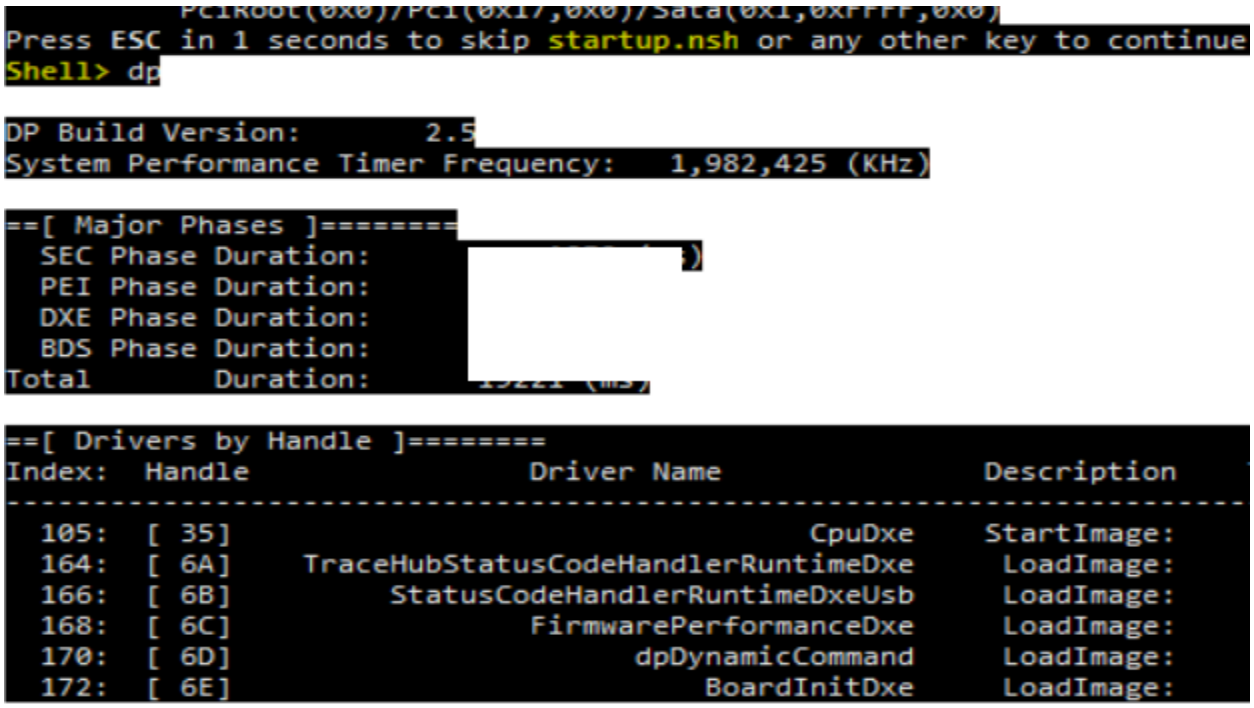

Figure 9: DP command output

- Disabling Rank Margin Test (RMT) will reduce ~6.50% in boot time with Release IFWI. This change saves time in PEI mode significantly because RMT is part of MRC – Memory Reference Code which runs in pre EFI phase.
- Removed some OPROMs which were not used, this gave ~9% reduction in boot time. This time is reduced in PEI and BDS phase with release IFWI.
- Enabling fast warm boot gave ~22% reduction in boot time in warm boot timing because in this path it uses previous configuration data from cold boot path.
- Dual mode SPI – Reduction in time ~13% on Release IFWI

5.2 Discussion

As we have seen in the result we can reduce the boot time for different modules by removing them or modifying in a way that reduces the boot time significantly. Release bios reduces the boot time significantly because of minimal debug logs. We can also skip the memory testing part of the MRC depending on the platform by enabling the fast boot flow. Based on the need and functionalities of the platform we can remove some option ROMs and CSM support for Legacy BIOS. As, this is the single socket system, multi socket communication support can be redused or skipped from the software perspective like link training, which in turn saves significant time.

Chapter 6: Conclusion and Future Scope

From these experiments we can conclude that by removing some code or modules we can save some booting time; it may be significant or not. It can be seen from the results that if we remove debug prints from the code which are added for debugging purpose the boot time reduces significantly. MRC code also takes considerable time in memory training and initialization. We must follow the whole MRC boot flow in cold boot, but we can skip memory testing part of the MRC. In case of warm boot, we can skip the MRC training completely by enabling fast boot because in warm boot memory read-write policies cannot be changed. Depending on the platform we can remove some OPROMs and CSM support for BIOS. Parallel execution of the code can also be implemented by modifying code in significant way, this will also reduce some boot time. We cannot implement same things on other platform without studying about the architecture of the platform.

References

[1] windriver, "Simics," [Online]. Available: https://www.windriver.com/products/simics.

[2] Intel Corporation, "Company profile," Intel, [Online]. Available: https://www.intel.in/content/www/in/en/company-overview/company-overview.html.

[3] Intel Corporation, "IAGS," Intel, [Online]. Available: https://intel.sharepoint.com/sites/IAGS.

[4] Tianocore Community, "Tianocore-training," [Online]. Available: https://github.com/tianocore-training/Tianocore_Training_Contents/wiki.

[5] G. X. Y. W. J. G. Michael Rothman, "Reducing Platfrom Boot Time," Intel - White Paper , 2011.

[6] Tianocore Community, "PI-Boot-Flow," Tianocore, [Online]. Available: https://github.com/tianocore/tianocore.github.io/wiki/PI-Boot-Flow.

[7] E. 2, "Creating efi images," [Online]. Available: https://edk2-docs.gitbook.io/edk-ii-build-specification/2_design_discussion/26_creating_efi_images.

[8] EDK 2, "Platform_dsc_defination," [Online]. Available: https://edk2-docs.gitbook.io/edk-ii-dsc-specification/3_edk_ii_dsc_file_format/33_platform_dsc_definition.

[9] Tianocore Community, "Edk2-Performance-infrastructure," [Online]. Available: https://github.com/tianocore/tianocore.github.io/wiki/Edk2-Performance-Infrastructure.

[10] UEFI.org, "resources," [Online]. Available: https://www.uefi.org/sites/default/files/resources/ACPI_6_2.pdf.

[11] differencebetween.net, [Online]. Available: http://www.differencebetween.net/technology/difference-between-cold-and-warm-booting/.

[12] AMI, "OPROM," [Online]. Available: https://www.ami.com/tech-blog/acronym-soup-what-is-oprom/.

[13] TotalPhase, "What are the Differences of Single vs Dual vs Quad SPI?," [Online]. Available: https://www.totalphase.com/blog/2020/05/what-are-the-differences-of-single-vs-dual-vs-quad-spi/#:~:text=Quad%20SPI%20is%20similar%20to,rates%20reach%20around%2040%20Mbps..

[14] EDK 2, "EDK 2 packages," [Online]. Available: https://edk2-docs.gitbook.io/edk-ii-module-writer-s-guide/2_an_edk_ii_package/21_introduction.