Bachelor's Thesis (TUAS)

Degree Program in Information Technology

Specialisation: Information Technology

2013

Yu Zhang

IMPLEMENTATION OF EXTENSIBLE FIRMWARE INTERFACE

-PERFORMANCE TEST

BACHELOR'S THESIS | ABSTRACT

TURKU UNIVERSITY OF APPLIED SCIENCES

Information Technology

2013-05 | 46

Instructor: Patric Granholm

Yu Zhang

Implementation of Extensible Firmware Interface

During the last 30 years, the performance of the various components in PCs has been increased constantly following Moore's Law. As the most important part, BIOS is responsible for connecting the PC hardware and software, initializing the PC and system boot. Such a key technology, however, has not been improved with the rapid development of PC technology. BIOS's immutability and drawbacks has become an obstacle to the development of PC technology. In order to solve the limitations of the traditional BIOS, Intel developed a new generation of alternative to the traditional BIOS named Extensible Firmware Interface (EFI) in 2000. Compared with traditional BIOS, EFI has incomparable superiority. It not only solves most of the drawbacks of traditional BIOS but also provide users with a more powerful and perfect service with a new design concept.

The main purpose of this paper is to learn the theoretical of computer structure and elaborate the future theory related to EFI. The thesis is structured as follows. The first chapter contains a detailed description of the traditional BIOS in the computer, then an introduction of how the BIOS function in the computer's boot process and finally an analysis of the EFI advantages compared with the traditional BIOS. The second chapter introduces the basic principles of the EFI including EFI system boot process, boot manager, system tables, services, protocols, handles and the EFI Shell. In addition, a detailed instruction of seven stages is also suggested according to the Framework working based on EFI standard. Finally, a comparison test is conducted about start-up characteristics with EFI BIOS and the traditional BIOS platform as well as demonstrate how users can benefit from using EFI BIOS. As a principle, EFI also belongs to the BIOS. In order to facilitate the distinction, in this thesis the BIOS used by customers and outside the EFI standard are referred to as traditional BIOS.

KEYWORDS:

Framework, UEFI, BIOS, Intel

TURUN AMMATTIKORKEAKOULU
TURKU UNIVERSITY OF APPLIED SCIENCES

CONTENTS

FIGURES

TABLES

ACRONYMS, ABBREVIATIONS AND SYMBOLS

ENIAC	Electronic Numerical Integrator and Computer
BIOS	Basic Input / Output System
CMOS	Complementary metal–oxide–semiconductor
EPROM	Erasable programmable read only memory
ESCD	Extended System Configuration Data
UEFI	Unified Extensible Firmware Interface
GUID	Globally Unique Identifier
HANDLE	Abstract reference to a resource
FRAMEWORK	Intel Platform Innovation Framework

1 Introduction to microcomputer

1.1 Computer Architecture

The computer is regarded as an extension of the human brain which means it can think. It has the ability to carry out calculations, simulations, analyze problems, operate machines and process services (Stokes, 2007). The computer architecture theory proposed by the mathematician John von Neumann laid the foundation of the modern computer. It contains two points, one is that binary should be the computing infrastructure, the other one is that program and any data are both stored together (Stokes, 2007). The computer is composed of five parts: operator, controller, memory, input and output devices. Input information includes characters and letters. The output is the result of the processing such as numbers, letters, tables and graphics. The memory can be used to store programs and data and also exchange messages between input and output devices and the central processing unit (CPU). The CPU contained the operator and the controller parts in the early time but these two have been combined as the development of science and integrated circuit technology. Von Neumann's theory solved the problem of balancing the speed and operation automation. Till today, most of the computers still work according to this theory.

1.2 Computer development

The world's first electronic digital computer named ENIAC was officially put into use at the University of Pennsylvania in the United States on February 15, 1946. It had 17,468 vacuum tubes with a power consumption of 174 kW and could do 5000 addition operations per second (History of Computing, 2013). A few decades after ENIAC, the computer has been developed rapidly. The major electronic devices have been changed to transistors, smaller-scale integrated circuits and then large-scale VLSI. The performance of computer has been greatly enhanced with every upgrading and the size and power consumption has been greatly reduced. Especially the emergence of the microcomputer allowed computers to spread rapidly into offices and homes alike. In general, computers have gone through the following four iconic stages.

1. The vacuum tube computer (1945-1956)

   In order to meet the needs of the U.S. government and military during World War II, Howard Aiken developed a fully electronic calculator to draw a trajectory map for the U.S Navy in 1944. This machine is referred to as the Mark I which used electromagnetic signals to move mechanical parts. It is as big as half a football field containing 500 miles of wire. On February 14th 1946, ENIAC with the ability of parallel computing was born which represents a milestone in the history of computers (History of Computing, 2013).

2. The transistor computer (1956-1963)

   In 1938, the invention of the transistor greatly enhanced the development of computers. Instead of using big evacuated tube, the volume and power consumption of electronic devices was decreased and the speed was improved. The transistor technology is the first to use for early supercomputers which were used for a large number of atomic science data processing. These machines were expensive and in very few quantities (History of Computing, 2013).

3. The integrated circuit computer (1964-1971)

   With the invention of the integrated circuit (IC) in 1958, several kinds of electronic components could be combined into a small silicon chip. The greatest impact computer is the IMB360 (History of Computing, 2013) system, which is the first general-purpose computer with average speed of operation from thousands to a million times per second.

4. The LSI Computer (1971 - present)

   Till the 1980s, the very-large-scale integrated circuit (VLSI) could already accommodate hundreds of thousands of components on the chip and then millions of components by using ULSI. Based on the development of the semiconductor, the first real personal computer was born in 1927. Its microprocessor contained 2,300 transistors and is able to conduct 60,000 instructions per second. In 1981, IBM introduced the personal computer (PC) for home, office and school (History of Computing, 2013).

2 Basic Input/output System (BIOS)

2.1 Definition

In IBM PC-compatible computers, the Basic Input / Output System (BIOS), is also known as the system BIOS or ROM BIOS (Tim, 2005). It is a program on a ROM chip embedded in the motherboard. It holds the basic input output program, system setting information, power-on self-test and system startup self-check programs. As a bridge connected to software programs and hardware devices, the BIOS provide the most fundamental and direct control and support for the hardware.

- CMOS and BIOS

As the BIOS is a program, it is store in the CMOS components. CMOS is short for complementary metal-oxide semiconductor (Baker, 2008). It is a semiconductor technology which can integrate Mosefet on a silicon chip. This technique is often used in the production of RAM and switching applications system. Because of the low power consumption of CMOS RAM, it can be sustained by the motherboard backup battery after the system power has been turned off. Thus, the user-setting stored in CMOS will not lose. CMOS itself is only a memory chip with the data storage function so the setting parameters of the CMOS need a special program. The early computer's CMOS setup program was stored on a floppy disk which was not very convenient. Now the vast majority of manufacturers have set a CMOS setup program. The capacity of Flash EPROM is 1MB or 2MB.

- Types of BIOS

Although there are hundreds of brands of computers in the world, only Taiwanese and the United States companies have ability of BIOS development, design and production. Phoenix and AMI are the two BIOS development companies in the United States. In 1998, Taiwan's BIOS Design Manufacturers' Award was acquired by Phoenix. After that the BIOS development has been monopolized by U.S. companies.

Award BIOS

Award BIOS is the BIOS product developed by Award Software. It is currently the leading production of BIOS and is applied on many companies' motherboards. The Award BIOS has almost become the standard specifications of the Intel CPU series motherboards (Award Software at a Glance, 1998).

AMI BIOS

AMI BIOS has been developed by AMI. It was first developed in the mid-1980s and used for most of the 286 and 386 computers. A new version was introduced in the 1990s to adapt to the development of technology, but it seemed to lag somewhat behind as the green energy-saving system began to spread. The market share of Award BIOS greatly increased by this opportunity. During this period, AMI developed WIN BIOS with a windowing function and the markers of the main window were more intuitive which is easier for BIOS setup (Corporate Profile Overview, 2001).

2.2 Functions of BIOS

The BIOS is responsible for initializing the hardware, detection hardware capabilities and boot the operating system in the PC boot process as shown in Figure 1.

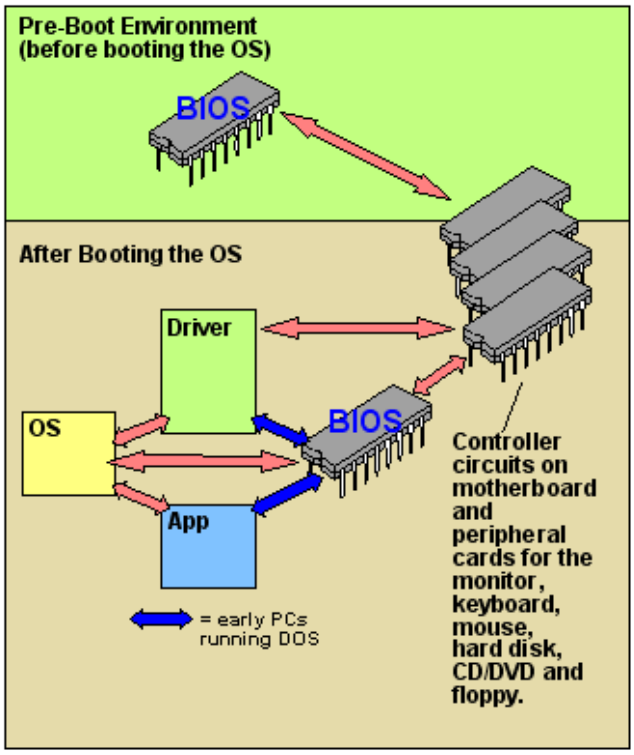

Figure 1. BIOS function in computer architecture (Wiley, 2010)

i. Self-test and initialization

After the computer power is turned on, the system will have an internal equipment inspection process which is called POST (Power on self-test) program. The complete test includes a test of the CPU, 640k basic memory, 1M or more extended memory, ROM, motherboard, CMOS, display card, hard disk and keyboard. If a problem is found in this process, the system will prompt or warn.

The lower part of Figure 1 shows the initialization including setting registers, creating interrupt vectors and initialization and detection of some external devices. When the computer starts, these parameters will be read and may hinder the starting if it does not tally with the actual hardware setup. The function of boot loader is to boot operating system. The BIOS reads the boot record from the starting sector of the disk and transfers the control of the computer to the boot record. The boot record loads the computer operating system.

ii. Hardware interrupts handling

When the computer is turned on, the BIOS will tell the CPU and other hardware interrupt numbers. There are several groups of services and each group has a special interrupt. For example, the interrupt number of video services is 10H and the interrupt number for screen printing is 05H. When an operation command is input by hardware, BIOS would use the corresponding hardware interrupt number to complete the command. All the relational operations can be carried out via corresponding instruction descriptions and do not need direct control by users. For example, when a key is pressed on the keyboard, a signal will be sent to the keyboard interrupt service routine and the interrupt routine will inform the CPU of the name of the key and send the information to the operating system (System Boot Process Explained, 2010)

iii. Program service request

The program service request is for the application and operating system which is mainly related to the input and output devices, such as reading the disk and output file to the printer. In order to accomplish these operations, BIOS must directly communicate with the computer I/O device. By issuing the command to port and transmit-receive data with external devices, the program can work without operating the specific hardware.

2.3 The Boot Process

Almost all computer users turn on the power to start the computer and watch the startup screen every day. What has a computer to do between pressing the power button and displaying the operating system? In order to start the computer smoothly, every computer parts including BIOS, operating system, and hardware must be normally operating. Even a single error is likely to cause the starting failed. Although the entire start-up process may only be as little as 30 seconds, the BIOS program has

performed a large number of works to prepare the computer for running. Figure 2 shows the boot process in normal PC (System Boot Process Explained, 2010).

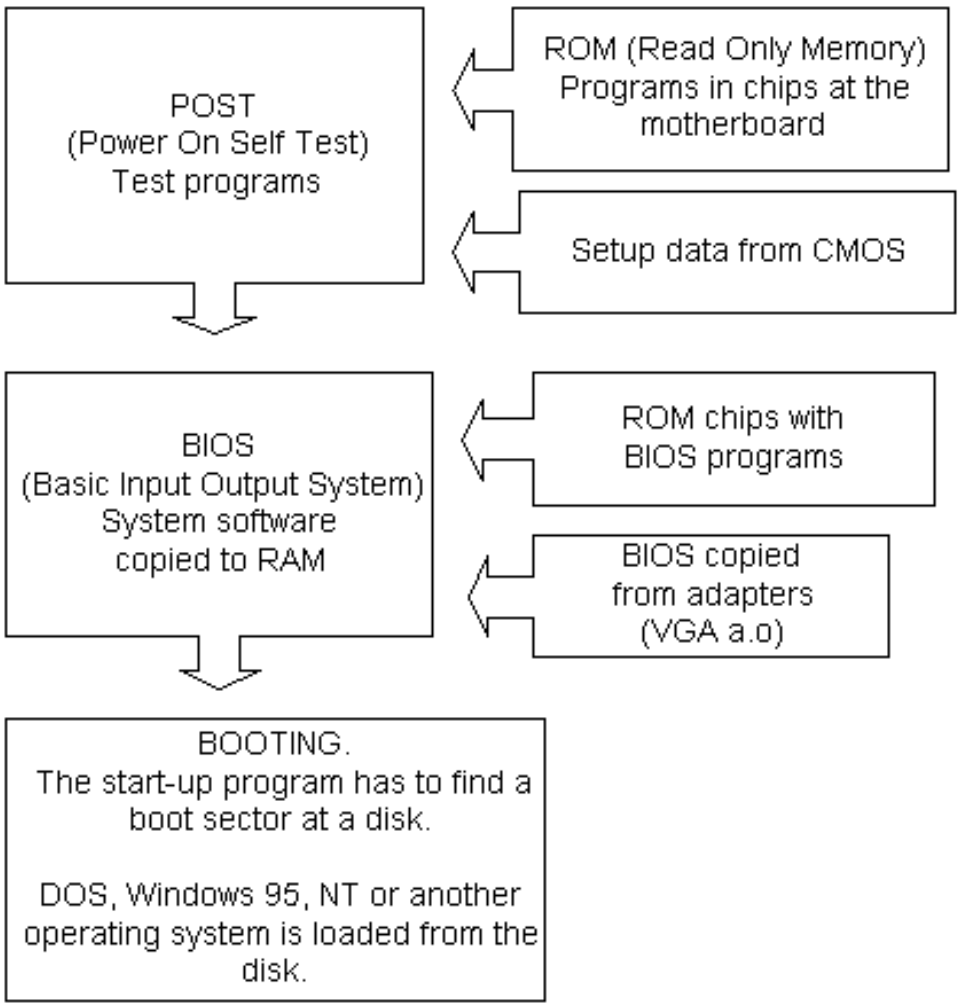

Figure 2. Booting process

a) Execute Jump instruction

After the power switch has been pressed, power starts to supply energy to the motherboard and other devices in the case. The control chipset on the motherboard will send a RESET signal to the CPU and let the CPU internal automatic recovery to the initial state. Yet, at the moment, the CPU does not immediately go to execute instructions. When the chipset detects the power has begun to stabilize with a voltage about 5V, the RESET signal is removed and the CPU starts executing instructions from address FFFF0H. This address is actually

within the address range of the system BIOS which can jump to the true boot code in the system BIOS. No matter if it is in the Award BIOS or AMI BIOS, this address is just a jump instruction.

b) Enter the POST

The system BIOS startup code first performs POST. Its main task is to check whether the key devices are working properly or not. The most common hardware error is memory problems, such as memory compatibility issues or memory type not matching. Since the graphics card is not yet initialized, if a fatal error is detected during POST, the alarm will be issued by the speaker. Each BIOS vendor will have its own alarm mode. Under normal circumstances, the POST self-test process is very fast within 2-4 seconds. After the POST, the other code would be called for more complete hardware detection (Power –on self-test, 2010).

c) Initialize the display card

System BIOS finds the starting position C0000H stored in the graphics ROM chip and calls initialization code to initialize the display card via video BIOS. After a very short time, most of the graphic cards will display on the screen some initialization information such as manufacturers, graphics chip type, memory size and so on.

d) Test Hardware

Now the system BIOS will detect and display the CPU type and operating frequency on the screen and then test all the RAMs. When memory test has been passed, BIOS starts to detect standard hardware devices including hard drives, parallel port, CD-ROM. The newer BIOS in this process will also simultaneously detect and set the timing parameters of the memory, hard disk parameters and access patterns. After the standard equipment detection, the BIOS will begin to detect Plug and Play devices. All the device information about found devices will be displayed on the screen concurrently assigning to these devices interrupts, and I/O resources.

e) Update ESCD

After all system testing and initialization have been completed, the BIOS will update ESCD (Extended System Configuration Data). ESCD is used to exchange configuration information between the system BIOS and the operating system. Generally speaking, if a re-test of every PNP device is executed in every system startup process, there should be two problems even if devices are never changed. The first is a waste of waiting time. The second is that BIOS does not assign the most reasonable resources every time. ESCD is used to solve the above two problems. In first system self-test, the entire resource allocation table will be written to ESCD. After that, the BIOS will only compare information with ones in ESCD, if there is no change of the system hardware, ESCD data are not rewritable (Power – on self-test, 2010).

f) Startup disks in the specified order

The last work for BIOS is to boot from a floppy disk, hard disk or USB Disk according to the boot sequence in the BIOS setup. To boot from the hard drive, for example, the system BIOS reads and executes the master boot record on the hard disk. The master boot record finds the first active partition from partition table and then reads and executes it. For a DOS system, the partition boot recorder is responsible for reading and execution files io.sys, config.sys, msdos.sys, command.sys, finally the autoexec.bat. After the prompt appears, a user can start to use the computer. For Windows systems, the computer will continue with the boot and initialization of the GUI.

3 Super BIOS - EFI

The BIOS obviously plays a significant role in the PC. However, the traditional BIOS have almost no technical improvements through the last 20 years shown as Table 1. The BIOS working in 16 bit environment is the most fatal flaw, the CPU manufacture has to consider the compatibility of the system when developing new processors.

Table 1. Development of computer technology

	Early stage (1980s)	Medium term (1990s)	Current (2010)
CPU	16bit internal, 8bit external	32-bit virtual memory	64-bit extension
System Bus	8bit ISA	XT,EISA,XVGA	SXVGA
Video	Monochrome text	Color VGA, SVGA	SXVGA
OS	DOS	Windows3.1, Windows95	Windows XP Windows Vista/7
BIOS	BIOS based on 16bit assembly language		

In order to overcome the shortcomings of the traditional BIOS, a new generation of BIOS standard Extensible Firmware Interface was launched by the Intel Company together with some other leading manufacturers. Although the underlying terms is very different, the basic function of EFI BIOS is the same as the traditional ones that is as a communication bridge between hardware and software. The improvement is that the new bridge is easier to manipulate as well as the function being more powerful (UEFI Specification Version 2.3.1, 2011).

3.1 The limitations of traditional BIOS program

Microsoft's operating system had been built on the concept of interrupt starting from the DOS era. Regardless of the switch of program or the switch of system, the operating system is achieved by interrupts. Although major BIOS vendors had improved a lot by adding many new elements to the product, such as ACPI and USB in recent years, the backwardness of BIOS's underlying function still had not changed.

3.1.1 The deficiencies of the traditional BIOS

The traditional BIOS's weakness can be summarized in the following five aspects:

A. Lack of uniform standards

The traditional BIOS have no uniform standard for the connection between hardware and the operating system. Thus, the structure varies by different BIOS manufacturers. It leads to a great deal of obstacles for users in order to become familiar with the BIOS operating environment.

B. The 16-bit modes of operation

The BIOS run under 16-bit assembly code since the first BIOS program was introduced. Because of compatibility reasons, 80386 reserved the 16-bit mode of operation. Even in the popular era of 64-bit processors, the CPU will switch to 16-bit real mode after power on, which greatly reduces the performance of the system.

C. The programming difficulties

The BIOS can only be written in assembly language development while the operating system software already uses C, C++ or even .NET. When using a high-end workstation motherboard and the latest SCSI disk array cards, this array card does not work. The reason is that writing, debugging and maintaining of BIOS

code can only be implemented by very specific expertise programmers. This resulted in the development of BIOS functions being very slow and a BIOS update cycle often lasts for months or even longer.

D. Lack of network support

Nowadays the network has gained recognition. It is a limitation of network-based program development that BIOS is unable to achieve network before entering the operating system.

E. User interface unfriendly

Characters and single-language user interface lead to poor maneuverability and usability.

3.1.2 The advantages and characteristics of the EFI specification

After the development of the Itanium CPU, the Intel Company in conjunction with other major hardware vendors developed the new BIOS specification EFI standard in order to allow a variety of new technology quickly transferred to the application stage. It provides a uniform standard for BIOS developers and simplifies the Firmware development process. The main advantages of EFI are:

A. The interface between hardware and software is more standardized

The EFI specification uses a modular design which logically contains the hardware control part and the operating system management part. Each motherboard manufacturers' EFI BIOS have similar code in the hardware part and a development interface in the software part. Various functions of the plug-ins, UI and features can be developed through this interface. For example, a variety of backup and diagnostic functions are available through EFI. It is also a major selling point for the motherboard manufacturers (Intel, 2010).

B. Breakthrough in capacity limits

EFI runs in 32-bit protected mode which can make full use of the CPU and memory space (Intel 2010). It can divide an area of the hard disk and store its own file system in this space. In this way, it can not only ensure plenty of space but also perform frequently used programs such as the hard disk partition, multi-operating system boot, system backup and recovery. For example, Intel has developed a system backup and recovery tool, like Norton Ghost, for the EFI platform which enables ordinary users do disk recovery, backup operations in the EFI BIOS program instead of using professionals.

C. Provision of good support for new hardware

EFI defines the interface of the hardware platform including standard bus types, such as USB and SCSI. In order to support emerging types of bus, EFI defines the EFI Driver Model which directly uses protocol/device path instead of using interrupts as it would in the traditional BIOS. The protocol/device path is similar to the form of the driver in the operating system.

D. Powerful graphical user interface

The EFI BIOS internally installed a graphics driver which can provide users with a high-resolution graphical user interface. When the operating system has problems, the user can enter the EFI environment to modify the configuration. All configurations can be done by clicking the mouse. Everything is as simple as operating applications in the Windows system.

E. Good network support

The EFI standard supports the TCP/IP network protocol and enables the direct use of network resources in the EFI interface. Other authorized users can conduct

reliable remote fault diagnosis through the network on their host with no need of entering into the operating system.

F. Provision of a consistent Firmware interface for the operating system

EFI defines a set of platform abstraction interfaces which can be used for operating system calls. By calling the abstract interface, the OS loader of the operating system and interface hardware platform can be run independently.

G. Breakthrough in the limitation of 2TB disk

The Globally Unique Identifier Partition Table Format (GUID Partition Table, 2011) was introduced in the EFI specification. GPT can break through the traditional MBR partition disk partitioning structure which allows only four primary partitions. The new structure is not restricted and the partition type is also changed by the GUID.

3.2 EFI fundamental architecture

EFI is a new model defined between the operating system and platform firmware, which provides platform-related information and start of service and operating system runtime services. It defines the essential interface for the platform firmware as well as interface and data structures for the operating system. The EFI driver model ensures that all drivers designed according to this model will able to work together.

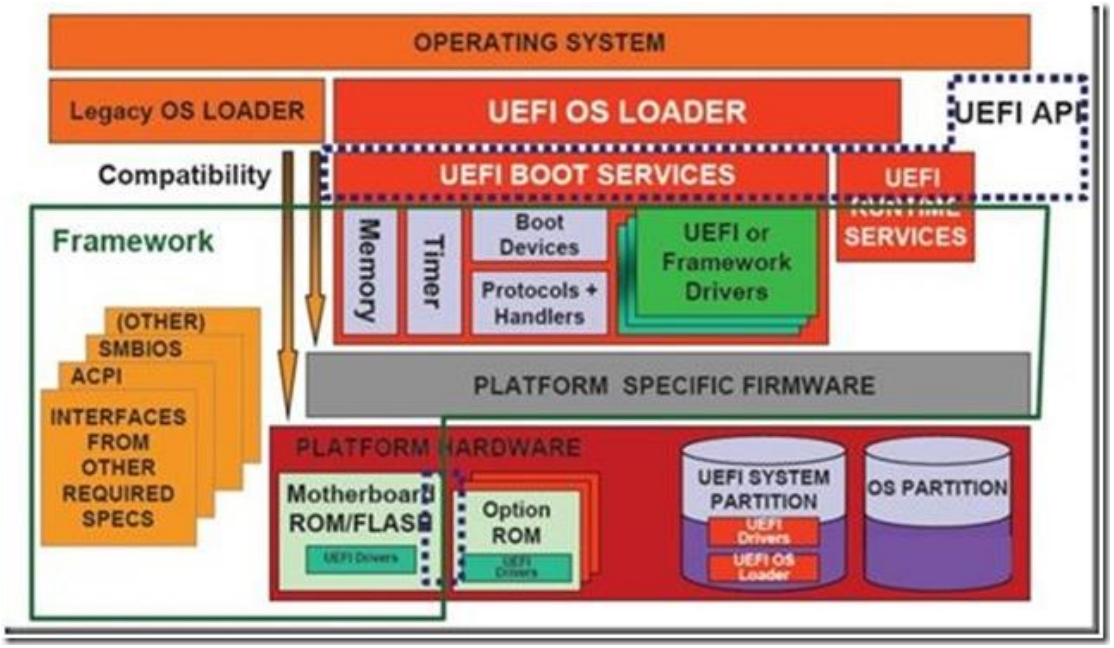

Figure 3. EFI basic architecture and relationship between platform hardware and OS (Introduction on UEFI, 2008)

EFI is divided into two parts as seen in Figure 3: EFI image and the platform initialization framework. There are three kinds of EFI image: EFI Application, OS Loaders and EFI drivers. The EFI Application is the core applications after hardware initialization and before the operating system, such as Startup Manager, BIOS settings and the EFI Shell. OS Loader is a special EFI Application and is mainly responsible for starting the operating system and turning off the EFI application. EFI Driver provides the interface protocol between devices. Each device has its own device version number and the corresponding drive parameters.

While the operating system is starting, EFI platform-specific firmware is used to transfer OS Loader mirror from the EFI system partition. EFI can boot not only from the disk but also from the storage media such as CD and DVD. After the OS loader is started, it will complete the entire operating system startup. Now the loader needs to call the UEFI Boot Services and related interface and initialize the components of the management module related to platform firmware.

3.3 The basic components of EFI architecture

EFI is composed of basic components including EFI Boot Manager, Firmware Core, EFI Protocol and EFI Driver Model (Intel, 2003).

3.3.1 EFI Boot manager

The start of the EFI BIOS platform is controlled by the boot manager. The first step is to initialize the platform firmware and then start to load EFI drivers and EFI application so as to further initialize to enrich the platform functions. The second step is to load a special EFI, Operating system loader.

The function of EFI Boot Manager is to use any file in the partition supported by EFI or use the load service of image file defined by EFI to load the EFI Application and EFI driver. The storage format of EFI image file is PE32 developed from the PE format which means a portable executable format. The image file can work on 32-bit or 64-bit systems at the same time without changing the code. The EFI image can be divided into two categories: EFI applications and EFI driver.

- EFI applications

  The EFI application is a kind of EFI image file which can be loaded and executed by the EFI Boot Manager or some other applications of EFI. To load an EFI application, firmware needs to be allocated enough memory space to copy the application and may also have to patch the address relocation. When the application returns the entry point or call the exit () parameters in boot service, the application will be unloaded from memory. The right to control will be returned to the EFI components which called this application (Intel, 2010).

- EFI driver

  The main role of the driver is access and management of hardware resources in the EFI environment. It will not replace the device driver in the operating system. The driver is also loaded by the EFI boot loader and the boot process is the same as the EFI applications. The greatest difference is the EFI application will be abandoned after running while the EFI driver will be retained in memory and referenced by other EFI core firmware. There are two kinds of EFI driver, Boot Service Driver and Runtime Driver. The difference between these two types of drive is that Runtime Drivers are still available after calling guide service function ExitBootService () but the Boot Service Driver will be unavailable.

- EFI boot process

  EFI allows extending the firmware architecture by loading EFI drivers and EFI applications. Figure 4 shows the EFI boot process. After power on, firstly the firmware platform is initialized, and then the EFI driver loaded. The next step is to select the system to be entered in the EFI system boot menu and submit the startup code to EFI. If everything is fine, access to the system occurs normally, otherwise the start service will be terminated and return to the EFI boot menu. Besides, EFI boot manager can combine the start menu of the OS Loader and EFI platform firmware to generate a unified boot menu which allows the user to select the operating system from any partition supported by the EFI boot.

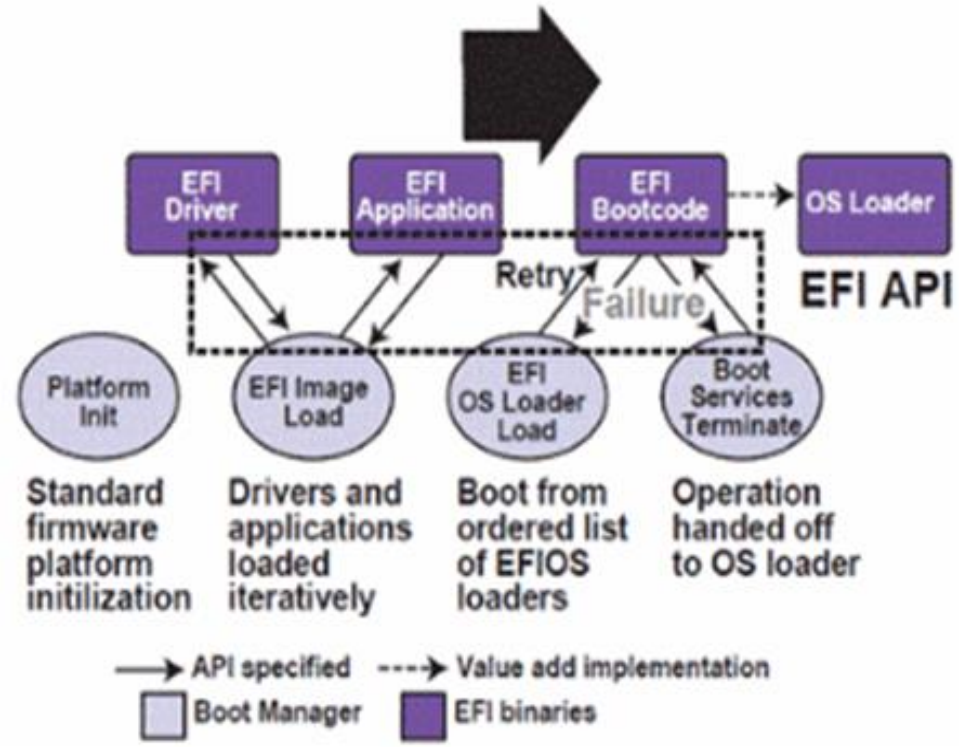

Figure 4. EFI boot process (Michael, 2012)

3.3.2 EFI Protocols and Handles

Protocols are vital in the definition and design in EFI and almost all implemented of functions in the EFI are based on the usage of protocols. The protocol in the EFI specification is a data structure identified by a globally unique identifier (GUID) which may be empty, may contain data fields or service entry address.

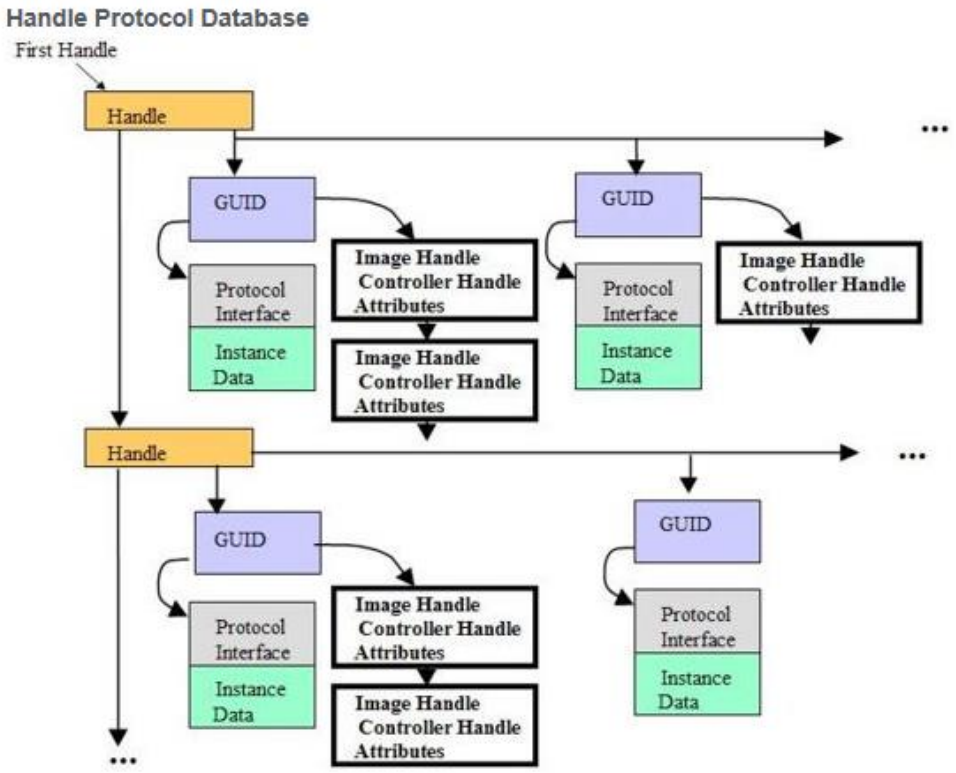

Figure 5. Handling database and protocol (Wolfgang, 2011)

The EFI driver can implement one or a plurality of protocols. The structure of protocol is shown in Figure 5. A protocol has a GUID (GUID Partition Table, 2011), protocol architecture and protocol services. GUID is the number of protocol containing 128 bits. Each protocol must contain a GUID. The protocol interface architecture is data structure or a pointer to performance functions. EFI can only define GUID and leave other parts blank. EFI provides several protocols for the firmware management or user industry standard such as USB, 1394 and PCI.

The handle is a collection of one or several protocols as shown in Figure 5. Each EFI driver and EFI Application may generate a handle at the initialization phase of the system firmware. A handle of the protocols constitutes a handle database which is global and can be reached by all EFI images.

3.3.3 EFI Service

The purpose of the EFI interface is to set a public boot environment abstraction layer and enable the usage of EFI image. Functions are called with a 64-bit interface which provides a space for future expansion. By this abstract call, the hardware platform and operating system can develop separately and maintain the consistency of the interface. The EFI kernel mainly provides two types of services. The EFI Boot Services is only available before the operating system loads and the Runtime Services is available even during the operating system running.

3.3.3.1 EFI Boot Services

EFI Boot Services is an interface function which can be called by the EFI Image in the pre-boot environment. The EFI Boot service's function is to control and manage hardware resources. These interface functions are divided into two kinds: global and handle-based. Global functions can access all system services and are available on all platforms. Handle-based functions can only access specific devices or functions (UEFI Specification, 2011). The EFI boot service provides the following services:

A. Timer, Event, and Task Priority Services. EFI does not support interrupt so its drivers and applications adopt a poll working method which is implemented by Timer.
B. Memory Allocation Services: allocating and free memory as well as acquiring system memory mapping table.
C. Protocol Handler Services: providing for EFI Application in order to reduce the code in bus driver and device driver.
D. Imaging: primarily used to manage the EFI Image file (Intel, 2003).

3.3.3.2 EFI runtime services

The interface functions of EFI runtime services can be invoked in the pre-start process and running process. In any case, the memory used by the EFI runtime services should be reserved and not used by the operating system. The common runtime service interface functions are mainly divided into four categories:

A. Variable Services: identify hardware information between the management of the operating system and the EFI platform.
B. Time Services: an abstract interface for the operating system to operating the hardware clock device mainly used for getting system clock and wake the computer.
C. Virtual Memory Service: provides optional virtual memory addressing schemes for the operating system. If the loader uses SetVirtualAddressMap () function, the operating system will call EFI runtime services automatically in the virtual address mode. All run-time services are non-blocking and can be called even in the case of interrupt off.
D. Some other services including ResetSystem which can reset the entire system, SetWatchDogTimer, Stall, and so on (Intel, 2003).

3.3.4 EFI driving model

The role of the EFI Driver Model is to provide a framework for the development of the driver. The design and development of the driven equipment can be simplified and executable file image can be smaller. The development of the EFI driver must comply with this model. The features of the EFI model are simplicity, multi-applicability, flexibility and scalability. An EFI device driver needs to be loaded to generate the EFI Driver binding protocol on the image handle. After binding, the device driver is responsible for the protocol implementation on the device controller in order to complete all abstract functions supported by this controller.

- Driver initialization

A drive image file must be loaded from some kind of media such as USB-disk, hard disk or floppy disk. Once the drive image is found, it can be loaded by the start service (LoadImage ()) into system memory. When a driver is loaded by LoadImage (), it must be implemented by StartImage (). This is a startup rule for all EFI applications and EFI drivers. An Entry Point fit for the EFI driver model must obey strict rules. First, Entry Point is not allowed to contact any hardware which means the driver can only put the protocol into its own image handle. Besides, if a drive is hoped to be uninstalled after run, it can update the image loading protocol in order to achieve the Unload () function (Intel, 2003).

- Device driver binding protocol

A device driver cannot generate any new device handle but only can install other protocol interface in the existing device handle. The device driver connected to device handle must install driver binding protocol EFI_DRIVE RBINGDING_ PROTOCOL in its image handle. The protocol includes three functions: Supported (), Start () and Stop ().

A. The Supported () function is used to test whether a driver supports a given controller. The function will be called many times in the platform initialization phase. In order to reduce the start-up time, the scope of the test function should be small and the execution time should be short.
B. The Start () function is used to start driver to activate the device corresponding to a given controller.
C. The Stop () function is used to forcibly stop the management of drive handle to the device and is written in all protocol interfaces installed by Start (). If the parameters of the function NumberOfChilden equals to 0, it is a device driver. If NumberOfChilden equals to 1, it is a bus driver.

3.4 EFI shell

As a special EFI application, the EFI shell is a console interface to start the application, load EFI protocols and device drivers. It is very similar to the CMD window in the Windows or Shell in Linux. In fact, the shell is an operating environment. It needs to receive and explain user input and tell the kernel to execute, and finally output the results of implementation (Brian Richardson, 2010).

Here are some basic commands used in EFI shell (Shell Command Reference Manual, 2007).

Shell Boot Commands

autoboot -- View or set autoboot timeout variable

mount -- Mounts a file system on a block device

Shell Configuration Commands

cpuconfig -- Deconfigure or reconfigure cpus

date -- Displays the current date or sets the systemdate

Shell Device Commands

default -- Sets, Resets, or Clears default NVM values

mm -- Displays or modifies MEM/IO/PCI

Generic Shell Commands

cd -- Displays or changes the current directory

mv -- Moves one or more files/directories to destination

rm -- Deletes one or more files or directories

4 EFI Framework

4.1 Introduction to Framework

EFI is a pure interface specification, not a set of software. It describes an interface between the operating system and platform firmware and its concrete realization varies with the design of motherboard manufacturers. INTEL specifies the implementation of EFI specification which is the Intel Platform Innovation Framework for EFI, referred to as the Framework. It fully complies with the EFI specification and takes advantage of the inherent characteristics of the EFI. Besides, Framework introduces many of the design ideas of modern computer science and programs in structured high-level languages. Different from the traditional BIOS, Framework initializes the platform by stages. Each stage is achieved by the different modules and has well-defined interfaces between the stages.

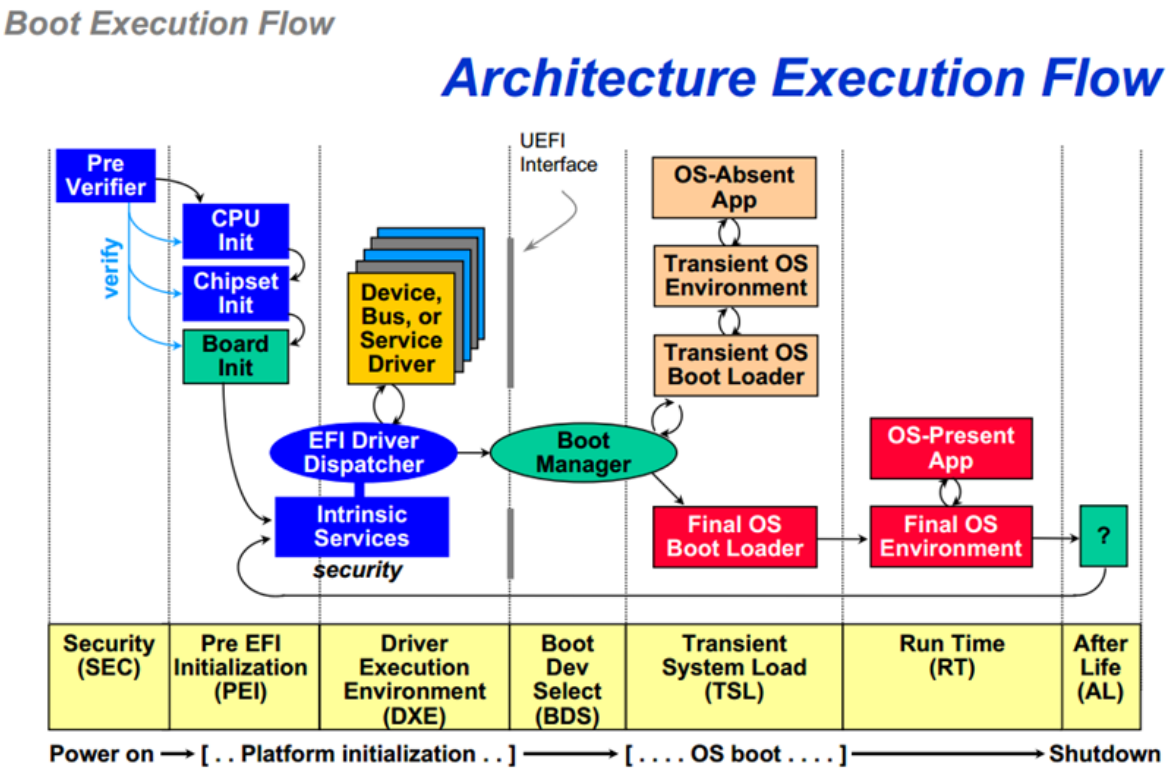

Figure 6. EFI Framework Execution Flow (Intel, 2003)

As can be seen from Figure 6, the first code to execute belongs to the Security (SEC) phase. The main task of this phase is to initialize the temporary memory area and verify early platform initialization code. When the SEC is finished, the initial memory description and validation module entrance will be passed to the PEI entrance in order to maintain the continuation of the trust chain. The PEI and DXE phases are the core of the EFI Framework. The PEI phase is mainly responsible for initializing the CPU and chipset and the DXE phase is responsible for most of the hardware initialization work. Interfaces between the modules and modules and system are named by the GUID. The advantage of using GUID is that it can extend services unlimited without considering conflict of names (Intel, 2010).

4.1.1 SEC stage

Security is the first code that is executed after power on and is the credible starting point of the Framework architecture. The main objective of SEC stage is to ensure the security of the initial code executed by the processor and the follow-up code passed to the next module. The SEC stage needs a processor to access the primary information of the platform and needs to support the lowest level of hardware. It needs to allocate a certain amount of temporary space to run this code with the help of the implementation capacity of the CPU. The mapping address of temporary space is specified by the SEC table from CPU cache or RAM.

4.1.2 PEI stage

The main purpose of this stage is to ensure that the DXE phase has sufficient operating environment. Since the framework architecture is written in C, the memory should be supported for stack and other resources for establishing the execution environment of the C code. The PEI phase is responsible for at least finding the system startup path, initializing the minimum system RAM and triggering DEX loader. The implementation of PEI depends more on the processor architecture than any other stage does. The more resources provided by processor in the initial stages, the more

interfaces will allocated between the PEI Foundation and PEIM.The following define the role of PEI Foundation and PEIM.

A. The PEI Foundation: It is an executable binary file responsible for distribution of PEIMs and providing a core set of services to be called by PEIMs. The PEI Foundation can be transplanted in a given instruction set on all platforms.

B. PEIM (Pre-EFI Initialization module): Executable binary files loaded by PEI basis code can perform different tasks and initialization. The PEIM can call another via PEIM PPI (PEIM-TO-PEIM interface).

There are three conditions that must be met in the PEI phase:

A. Temporary RAM: PEI Foundation needs to initialize a small amount of RAM in the SEC stage so that PEI has enough space to store data before the system memory is fully initialized (Intel, 2003).
B. Boot Firmware Volume: BFV contains PEI Foundation and PEIM and stores in the system memory address space (Intel, 2003).
C. Security: The SEC stage provides an interface to the PEI Foundation for the implementation of the security authentication operation. In order to extend the root of trust, the PEI Foundation will use the security mechanism to check each PEIM (Intel, 2003).

PEI's main task is to initialize the permanent memory which is the system's main memory. The main memory is initialized using a specific PEIM which is the PEI services InstalPeiMemory (). After installing the main memory, the PEI Foundation will temporary copy the memory stack to a segment of the main memory and switch to the main memory to continue the PEI stage. The last action of the PEI Foundation is to transfer control right to the DXE IPL PPI so that when the DXE has the control rights, the DEX Foundation will also obtain a precise memory usage records. Figure 7 describes the transferring right of control from PEI to DXE. In short, the PEI initial system meets the minimum requirements of the starting of DXE. The PEI Foundation is

able to locate and transfer rights of control to the DXE Foundation and to transfer system status information to the DXE Foundation.

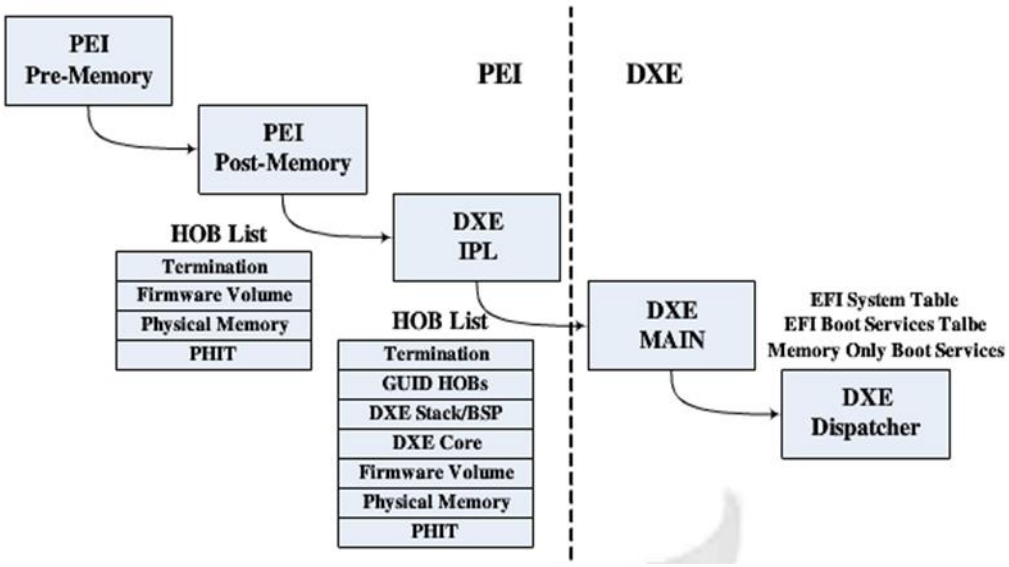

Figure 7. The control right transferred from PEI to DXE stage

4.1.3 Driver execution environment (DXE)

DXE performs most of system initialization. After the PEI stage, DXE gets permanent memory for loading and execution. The state of the system will be sent by a series of position-independent data structure HOB (Hand-off Block) to DXE. [13] The only requirement to perform DXE is an effective HOB list describing information about permanent memory and other system status. There are many different ways to produce an effective HOB list and PEI is just one of them. As shown in Figure 8, the DXE phase has three components: DXE Foundation, DXE Dispatcher and DXE Drivers.

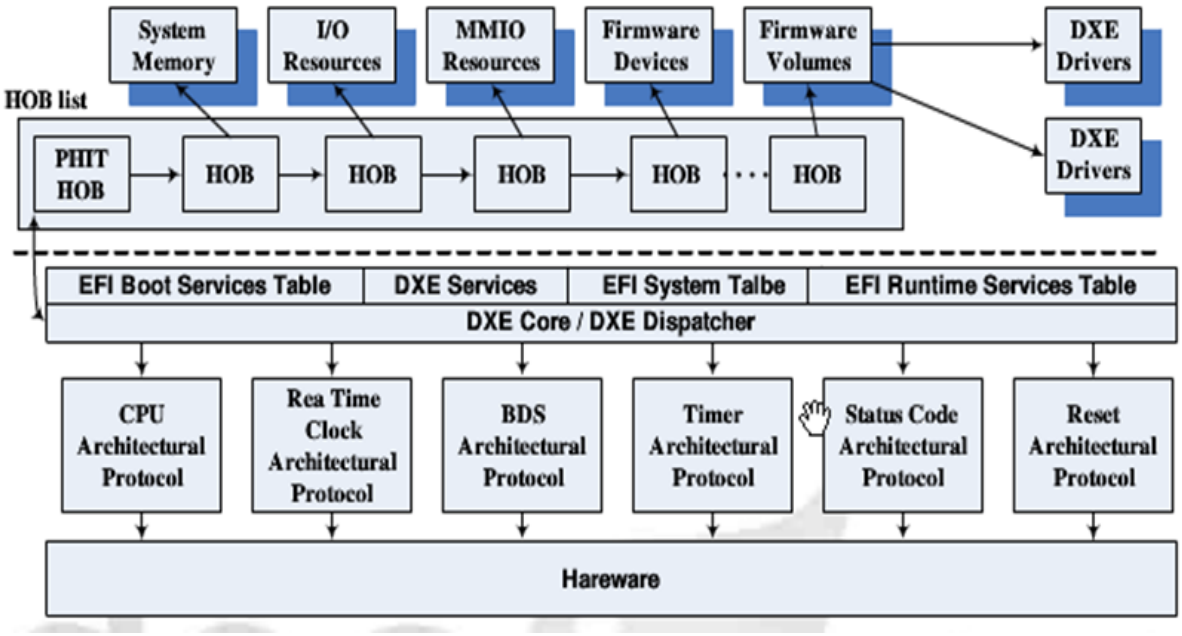

Figure 8. DXE architecture (Intel, 2003).

The DXE structure includes the following sections:

Entrance: HOB list from PEI.

Lower level: DXE Foundation schedules the hardware abstraction.

Upper level: data structures used in DXE including EFI Boot services table, DXE services and Runtime services Table.

The DXE Foundation cannot enter the operation system runtime environment because the DXE Foundation belongs to the starting service code. Only runtime data structures and runtime service distributed by the DXE Foundation can enter the operating system runtime environment.

- DXE Foundation

  DXE Foundation serves a boot image stored in the BFV. As the first implemented components of the DXE phase, it prepares the boot environment for the components of the later DXE stages. The HOB list is one of the prerequisites

including all the information required by the DXE Foundation, such as available system memory range, firmware volume position, I/O resources and hardware platform. According to the HOB list, the DXE Foundation generates the EFI boot service, runtime services, and DXE Services.

- DXE dispatcher

  DXE Dispatcher is to find the firmware volume of the system according to the HOB list and execute the DXE driver ordered in accordance with certain rules. The allocation sequence is determined by a priority document in optional storage FV which stringently defines the execution order of the firmware volume driver. If there is no priority file in driver firmware volume, a dependency expression can be found and determine the fulfillment of the driver performance. Whenever the DXE Dispatcher finds a firmware volume, it will find the priority document and load driver according to the document then the next firmware volume. After all the firmware volumes have been executed, the Dispatcher will transfer control to the BDS framework protocol which implements the start of the platform strategy.

4.1.4 Boot Device Selection (BDS)

The BDS stage starts after all the DXE drivers have been assigned. It is the final stage before the EFI transfers system control to the OS where users can perform many operations. This stage has two requirements before executed. The first is the DXE Architectural Protocols have been registered to handle database. The second is BDS has checked all the required drivers have been installed otherwise it returns to the DXE distributor to proceed with the installation of the driver. Figure 9 shows the execute flow.

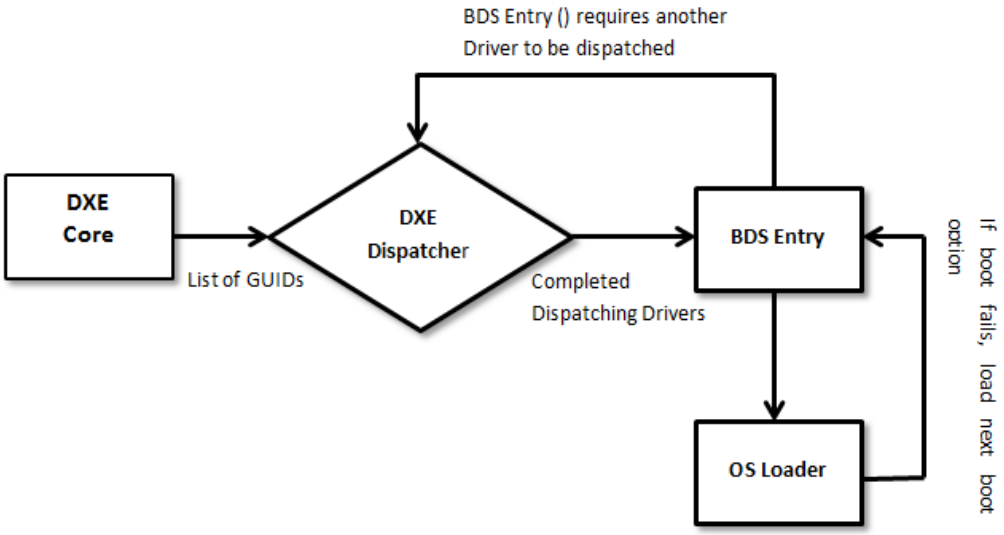

Figure 9.BDS execute flow

4.1.5 Transient System Load (TSL)

The tasks in this stage will determine the result of the selection in the BDS stage. If the choice is to start the operating system, control will be handed over to the final operating system loader. If the choice is to choose the EFI application, control will be transferred to the temporary operating system loader in order to establish the temporary operating environment for running EFI application.

4.1.6 Runtime (RT)

This is the OS running stage. At this time, the main works of the EFI BIOS have been completed. In this stage, the EFI system table and EFI runtime service is still working while other services are ineffective including all the DXE basis code and startup service code.

4.1.7 Afterlife (AL)

AL stage is the follow-up phase of the RT stage. The control right of the platform is handed over from the operating system to the firmware platform before shutting down the system.

5 Performance test between EFI and legacy BIOS

The emergence of the EFI standard injected new vitality to the computer industry. For the users, EFI brought a beautiful and friendly BIOS operating environment. Moreover, EFI led to a fast start and support over 2TB partition which is one of the reasons of its popularity. Microsoft's Surface Pro is able to start so fast because of using EFI boot and SSD technology. The experiment undertaken for this Bachelor's thesis is designed to study the benefits brought by EFI Fast boot and the differences compared with traditional BIOS.

5.1 Platform demand analysis

The Quick Start function is only supported by EFI boot thus a latest motherboard supporting UEFI BIOS was needed. In general, the post-2010 motherboards support the latest EFI standard. Since the EFI standard was founded by INTEL, an INTEL chipset was the best choice. Another point was that the disk should support GPT partition format. Besides, the system had to use a 64-bit operating system no matter if it is Linux or Windows. Although both the 64-bit Windows 7 and Windows 8 system can support UEFI boot, Windows 8 supports additional Quick Start function which can greatly reduce the loading time after the start. Obviously it was a better choice for cooperating with the EFI motherboard. In accordance with the above requirements, the test system hardware and software environment were as follows:

	Hardware Platform
Laptop Model	Fujitsu Lifebook UH572
CPU	I5 3317U Ivy Bridge
Motherboard	FUJITSU FJNBB31
Memory	DDR3-1600 4G
Hard disk	HITACHI 320GB M4 64G SSD
Graphic Card	HD 4000
	Software Platform
OS platform	Win8 64bit Pro
	Win7 64bit Pro
BIOS version	Phoniex Technologies UEFI Version 2.07

5.1.1 GPT partition

From the Vista and Win7 era, the GPT format was designed to solve the problem caused by the limitation of the hard disk. The GPT is short of Globally Unique Identifier Partition Table Format. It also has another name, GUID partition table format, which could be seen in much disk management software. GPT is the disk partition format used by UEFI. One of the GPT partition advantages is that is has different partitions for different data and different permissions for different partitions. GPT can ensure the uniqueness of GUID so copying the entire hard disk is not allowed in GPT. In this way, GPT can ensure the security of the data within the disk. Compared with the MBR partition, the GPT partition table uses 8 bytes equally in 64bit to store the number of sectors so it can support up to 264 sectors. Suppose 512 byte per sector capacity then the maximum capacity of each partition is 9.4ZB (9.4 billion terabytes) which is far more than 2.19TB's limitation of the MBR. The following Figure 10 shows the details of the operating system supporting the GPT partition. All the 64-bit Windows system can support a GPT-formatted hard disk as a data disk for the read and write operations. However, to support system boot feature, UEFI BIOS is needed. On the other hand, Linux and Mac systems can give a comprehensive support for the GTP partition format. Basically, systems after Fedora 8 or 10.4.0 Mac can directly support GPT partition (GUID Partition Table, 2011).

OS family	Version or edition	Platform	Read and Write Support	Boot Support
FreeBSD	Since 7.0	IA-32, x64	Yes	Yes
Linux	Fedora 8+ and Ubuntu 8.04+	IA-32, x64	Yes	Yes
Mac OS x	Since 10.4.0	IA-32, x64	Yes	Yes
Mircosoft	Windows XP	IA-32	No	No
Mircosoft	Windows Server 200	IA-32	No	No
Mircosoft	Windows Vista	IA-32	Yes	No
Mircosoft	Windows 7/8	IA-32	Yes	No
Mircosoft	Windows Xp	X64	Yes	No
Mircosoft	Windows Server 200	IA 64	Yes	Yes
Mircosoft	Windows Vista	x64	Yes	Requries UEFI
Mircosoft	Windows 7/8	X64	Yes	Requries UEFI

Figure 10. GPT support on operation systems (GUID partition Table, 2011)

Creation or change of the GPT partition is not difficult. Since converting from the MBR partition to the GPT partition will lose all the data from the hard disk, we need to backup before changing the hard disk partition format and then use the Windows built-in disk management function or the disk management software such as DiskGenius to easily convert hard disk to GPT (GUID) format conversion. After finishing, we can start the installation process.

5.2 (EFI vs. BIOS) step-by-step test instructions

1. Prepare a data clean USB disk downloaded Windows 8 64-bit installation files from the MSDN and copy into the U disk.

2. Set the motherboard to fast startup mode, install Windows 8 with USB disk.

3. Boot the system with USB disk in UEFI mode

4. Partition the new hard disk with GPT and install the operating system according to the instructions.

5. Use the timer to record the time from full boot to entering the system. Repeat six times and take the average.

6. Go to Control Panel and close Quick Start option. Restart and press F2 to enter the BIOS. Shut off the Quick Start and open the CSM compatibility mode.

7. Format the hard disk partition, insert Windows 8 64-bit installer disk, and install again.

8. Partition the hard disk in the MBR format and complete the system installation following the instructions.

9. Use the timer to record the time from full boot to entering the system. Repeat six times and take the average. (Repeat step 5)

5.2.1 Specific steps

1. Prepare an USB stick with at least 8GB capacity and format as FAT32 form. NTFS is not supported in UEFI Start format. Download the Windows 8 installation image file named en_windows_8_x64_dvd_91 5440.iso on MSDN. Check the mirror before production of the system installation U disk and check the SHA1 detection code provided by MSDN in order to ensure that the installation file has not been modified. Production of the installation U disk is the same as ones for other systems. Write the image file to the USB Disk with dedicated ISO making software which chose the UltraISO software here. First insert the U disk into the USB 2.0 port. Open the installation file in UltraISO. In the software menu, select "Start - Select the hard disk image" and set the write mode to USB-HDD +. Click OK and wait about 10 minutes depending on the system configuration, an USB disk system is produced. Open the U disk and double check the "EFI" file folder which is the key point to boot via UEFI, as shown in Figure 11.

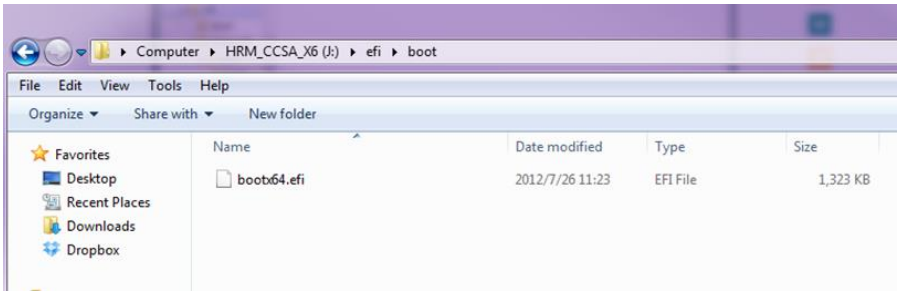

Figure 11. EFI image in Boot folder

Bootx64.efi is the boot image which can boot manually through the EFI SHELL in the motherboard or by choosing startup sequence UEFI: USB Disk in EFI BIOS.

2. Restart the computer and enter the BIOS setup program according to the different vendors set up. The test platform UH572 uses F2 to enter. BIOS initial screen is shown in Figure 12.

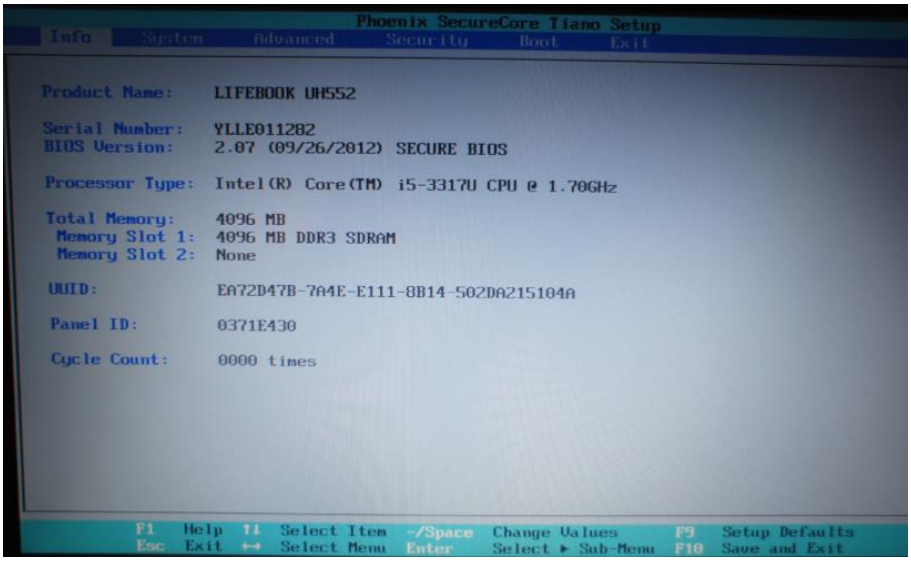

Figure12. Phoenix SecureCore Tiano Info Screen

The Info screen shows basic information about the platform. The product name is Lifebook UH552 and the BIOS version number 2.07. BIOS was generated on September 26, 2012. CPU is I5 3317U with frequency 1.7GHz. Other information is the memory capacity, equipment labels and battery usage time.

In the advanced interface, select Fast Boot to enable CSM compatibility mode to disable state. In this way, UEFI can open the quick start mode. Fast boot is not a necessary option in the UEFI boot. However, in order to better play the advantages of Windows 8 Quick Start, select Open. The first stage of the boot is BIOS self-test which is mainly for initializing devices and verifying components. The self-test of CPU, memory and display output part is necessary and does not relatively change much. However, the self-test of some I/O interfaces is not so necessary especially the USB interface because they will not affect the success of start which means they may be skipped. In short, the Fast Boot skips some non-essential parts of the self-test process in order to speed up the boot. Part of the desktop motherboards can also select Ultra-Fast mode. The importance is that if the user wants to use Ultra-fast mode, the user must select the GPT partition and install the UEFI boot when installing the Windows 8.

3. Restart the system and enter the boot sequence menu, select UEFI: TOSHIBA TransMemory 1.00. In order to make windows automatically, install drivers required by

UEFI, choose the startup options with UEFI when selecting the disk as shown in Figure 13.

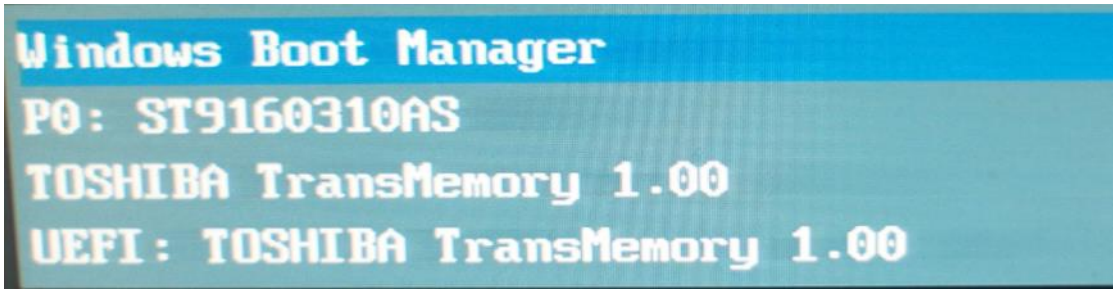

Figure 13. Screen shot of Disk Boot select menu

4. Install the system following the prompts. Installing Windows 8 with UEFI boot does not have much difference with the traditional installation, so we still install the operating system step-by-step via the Installation Wizard. The only difference is during disk partition. If the target disk is blank, directly select the disk and the installation program will help build the EFI partition. If Windows cannot be installed to the disk, the disk needs to be converted to GPT format since the target is creating an MBR disk. Here we use the Diskpart tool to manually create a GPT partition. Press SHIFT+F10 in the interface of partition selection and callout the Command Line.

5. After the system has been installed, wait for the windows to complete the final configuration, install and update the latest drivers. Restart and then enter the Driver - Power Management. Confirm that the fast boot option is open and shut down to test the system boot time. Take down the time and repeat six times. Ensure that in each test computers are turned off and then started. In this test, the start time is calculated from the time when the start button is pressed to the first time the UI interface is shown. Strictly speaking, the system has not been fully loaded at that time. However, the focus of this test is hardware self-test time. Since the time should be certain from showing UI interface to the completed state, it is not included in the scope of the test time.

6 After completing the test 6 times, enter the control panel - Power Management to turn off fast boot option. Configure fast boot to disable state in the UEFI BIOS settings as well. After system reboot, turn off the power and test 6 times again.

7 So far the UEFI test section has been completed. Set CSM mode to Enable and Boot Mode is Legacy Bios in the UEFI. At this time, the UEFI will work in compatibility mode which is the traditional BIOS mode. After restarting the computer, enter the boot menu and select Normal Mode to boot the installation

system TOSHIBA Transfer memory1.0 as shown in figure 13. Select the mode without UEFI prefix.

8 Follow the prompts to complete the installation of the operating system. Delete the GPT partition in the selected partition interface to blank the hard disk. Recreate the disk in the MBR partition table format.

9 After the installation and update has been completed, install the same version of the driver as in the fifth step and keep the same settings. Restart the system and conduct the start-up time test in the MBR partition format. Take the average score of 6 tests. This time the Fastboot is in unusable state which means the Fastboot function of the Windows 8 only supports in the GPT partition format.

5.3 Test Results

UH572 Fast Boot Test (HDD)			
	Disabled(s)	Fast Boot(s)	Legacy Mode(s)
Round 1	48.6	22.8	50.8
Round 2	50.2	20.5	49.6
Round 3	47.2	24.3	47.2
Round 4	44.1	23	46.4
Round 5	44.3	26	45.5
Round 6	43.8	25.3	49.3
Avg(s)	46.36666667	23.65	48.13333333

Figure 14. Fast Boot test result

Test Summary

Testing the boot time is not same as testing the Windows system start-up time. The latter can be draw accurately by installing software. However, this test mainly focuses on comparing boot time in the UEFI mode and normal mode so the only method is manual testing via a stopwatch.

The actual test boot time has shown relatively large errors which deviate relatively large. Therefore, the test sample was increased in testing and selected 6 similar times. The specific reason for the deviation may be related to the order of driver loading and is not in the scope of this test.

As can be seen in Figure14, in the condition of the disabled fastboot which is conventional AHCI start, the average time is 46.37 seconds. In the FAST mode, the average time is approximately 23.65 seconds which decreases about 22.8 seconds. In Legacy Mode which is simulating the traditional BIOS, the average time is 48.13 seconds which is 1.9 seconds more than in the Disable state. This shows even in the case of closing the Quick Start support, the start of system supporting UEFI is faster than ordinary BIOS about 1.3 seconds. When opening fastboot in Windows 8, the start-up time is shortened by nearly 24 seconds. Starting in the condition of opening the UEFI and the fast mode in Windows 8 is really short in hardware self-test and the self-test screen displayed on the screen is almost fleeting. As far as we are concerned, the latest Windows 8 is a perfect match with UEFI. This test is based on notebook platform and we supposed the effect would be more obvious when replaced by mainstream performance desktop platform.

6 Summary and future work

This thesis is mainly a study concerning theoretical knowledge and improving the understanding of the theory via a practical test. EFI BIOS, a new pre-boot platform, is proposed to address the limitations of the traditional BIOS. The study conducted on this platform is an innovative, creative, and challenging work. In this thesis, the EFI specification and the latest Framework working is researched and analyzed starting with the limitations and problems of traditional BIOS. Then EFI is introduced including the EFI system boot process, the boot manager, system tables, the protocols and the handles. Secondly, the working mechanism of the Framework in Intel specification is described elaborated parted by seven stages including SEC, PEI, DXE, BDS, TSL, RT and AL as well as their own functions and cooperation. These are a very important part of the EFI BIOS and we hope it can contribute to the future development on the EFI. In the last chapter, a detailed comparison test is conducted from the user perspective. The Quick Start of the EFI BIOS is also the most attractive feature for the user. Through the different performances comparing to the traditional BIOS, people can achieve a more perceptual understanding of the actual performance of the EFI.

The quick start not only has outstanding performance at the user level but also suggests a broader space for developers. However, as application development on the EFI system is different from other operating systems, people do not have much experience in researching the EFI system and only a few people develop applications based on EFI.

With the passage of time and the accumulation of technology, EFI-based applications will become more and more abundant. Motherboard manufacturers can also develop different types of applications for various target user groups. Combined with the network skills, we hope to develop a remote small terminal based on the feature that EFI supports TCP / IP so that the terminal can be run in the EFI platform before the start of the system. This technology can solve technical problems encountered by many company users before the system starts and also can save labor costs for IT technicians. Remote assistance for locating the problems of underlying hardware is unthinkable in the era of traditional BIOS, not to mention the networking capabilities.

REFERENCES

Stokes, Jon (2007). Inside the Machine: An Illustrated Introduction to Microprocessors and Computer Architecture. San Francisco: No Starch Press.

History of Computing (2013). History of Computing available at

http://en.wikipedia.org/wiki/Computer (Accessed 10 April 2013)

Baker, R. Jacob (2008). CMOS: circuit design, layout, and simulation (Second ed.). Wiley-IEEE. p. xxix.

Tim F. (2005) BIOS (Basic Input Output System) Available at:

http://pcsupport.about.com/od/termsb/p/bios.htm

INTEL (2003). Platform Innovation Framework for UEFI specification v0.90. Available at:

http://www.intel.com/content/www/us/en/architecture-and-technology/unified-extensible-firmware-interface/efi-specifications-general-technology.html

Brian Richardson (2010). "Ask a BIOS Guy: "Why UEFI"". Intel Architecture Blog. Retrieved 18 June 2010.Available at:

http://embedded.communities.intel.com/community/en/embassadors/blog/2010/05/10/ask-a-bios-guy-why-uefi (Accessed 20 April 2013)

Intel (2010). Platform Innovation Framework for UEFI Overview. Available at:

http://www.intel.com/content/www/us/en/architecture-and-technology/unified-extensible-firmware-interface/efi-homepage-general-technology.html

Windows 8 Secure Boot (2012). The Controversy Continues. PC World.

The System Boot Process Explained (2010). Available at

http://www.webopedia.com/DidYouKnow/Hardware_Software/2004/BootProcess.asp

UEFI Specification Version 2.3.1 (2011). UEFI Specifications and Tools. Available at:

http://www.uefi.org/specs/

Power –on self-test (2010). Available at

http://en.wikipedia.org/wiki/Power-on_self-test

GUID Partition Table (2011). Available at

http://en.wikipedia.org/wiki/GUID_Partition_Table

Intel (2003). Hand-Off Block (HOB) Specification v0.9 Available at:

http://www.intel.com/content/www/us/en/architecture-and-technology/unified-extensible-firmware-interface/efi-hand-off-block-specification.html

Award Software at a Glance (1998) Available at

http://web.archive.org/web/19980216055216/www.award.com/corporat/frabout.htm

Corporate Profile Overview (2001) Available at:

http://www.ami.com/About/Corporate/

Intel (2007). Shell command Reference Manual. Revision 1.1. Available at:

ftp://ftp.heanet.ie/mirrors/sourceforge/e/ef/efi-shell/documents/ShellCommandManual.pdf

Figure 1. Wiley (2010) BIOS - technical definition. Available at

http://computer.yourdictionary.com/bios

Figure 3. Introduction to UEFI (2008). Available at

http://www.bios.net.cn/BIOSJS/UEFI/5872.html

Figure 4. Michael L. (2012). UEFI on Linux Is like A Pathogen. Available at

http://www.phoronix.com/scan.php?page=news_item&px=MTA4NDA

Figure 5. Wolfgang R. (2011). Intel UEFI Architecture and Technical Overview. Available at

http://software.intel.com/en-us/articles/uefi-architecture-and-technical-overview

Figure 6. INTEL (2003). PlatForm Innovation Framework for UEFI specification v0.90. Available at: http://www.intel.com/content/www/us/en/architecture-and-technology/unified-extensible-firmware-interface/efi-specifications-general-technology.html

Figure 8. INTEL (2003). PlatForm Innovation Framework for UEFI specification v0.90. Available at: http://www.intel.com/content/www/us/en/architecture-and-technology/unified-extensible-firmware-interface/efi-specifications-general-technology.html