BIOS Support for Intel® 2014(Broadwell) Platform

Major Project Report

Part-I & II

Submitted in partial fullfilment of the requirements

for the degree of

Master of Technology in Computer Science and Engineering

By

Adodariya Vishal P.

(11MCEC01)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

INSTITUTE OF TECHNOLOGY

NIRMA UNIVERSITY

AHMEDABAD

BIOS Support for Intel® 2014(Broadwell) Platform

Major Project Report

Phase-I & II

Submitted in partial fullfilment of the requirements

for the degree of

Master of Technology in Computer Science and Engineering

By

Adodariya Vishal P.

(11MCEC01)

External Project Guide:	Internal Project Guide:
Mr. Satya Yarlagadda	Prof. Vibha Patel
Project Lead,	Dept.of Computer
Intel India Technology Pvt. Ltd.	Science & Engineering,
Bangalore	Nirma University

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

INSTITUTE OF TECHNOLOGY

NIRMA UNIVERSITY

AHMEDABAD

DECLARATION

This is to certify that,

I, **Adodariya Vishal P.**, **11MCEC01**, a student of semester IV Master of Technology in Computer Science Engineering, Nirma University, Ahmedabad , hereby declare that the project work **BIOS Support for Intel® 2014(Broadwell) Platform** has been carried out by me under the guidance of Mr. Satya Yarlagadda and Mr. Piyush Sharma, Intel Technology India Private Limited, Bangalore and Prof, Vibha Patel Department of Computer Science and Engineering, Nirma University, Ahmedabad. This Project has been submitted in the fulfillment of the requirements for the award of degree Master of Technology (M.Tech.) in Computer Science and Engineering, Nirma University, Ahmedabad during the year 2012 - 2013.

I have not submitted this work in full or part to any other University or Institution for the award of any other degree.

Adodariya Vishal P. (11MCEC01)

CERTIFICATE

This is to certify that the Major Project entitled **BIOS Support for Intel® 2014(Broadwell) Platform** submitted by **Adodariya Vishal Parsottambhai (11MCEC01)**, towards the fulfillment of the requirements for the degree of Master of Technology in Computer Science Engineering of Nirma University of Science and Technology, Ahmedabad is the record of work carried out by her under my supervision and guidance. In my opinion, the submitted work has reached a level required for being accepted for examination. The results embodied in this major project, to the best of my knowledge, have not been submitted to any other university or institution for award of any degree or diploma.

Mr. Satya Yarlagadda	Prof, Vibha Patel
External Guide,	Internal Guide,
Intel Technology India Ltd.	Nirma University
Mr. Piyush Sharma	Prof. Vijay Ukani
Project Manager,	PG Coordinator - CSE,
Intel Technology India Ltd.	Nirma University
Dr. Ketan Kotecha	Dr. Sanjay Garg
Director,	HOD - CSE,
Nirma University	Nirma University

ACKNOWLEDGEMENT

First and foremost, sincere thanks to my Mentor, **Mr. Satya Yarlagadda**, Intel Technology India Private Limited, Bangalore for his valuable guidance. Throughout the training, he has given me much valuable advice on project work. Without him, this project work would never have been completed.

I would like to thank my Manager **Mr. Piyush Sharma** , Intel Technology India Private Limited, Bangalore. I enjoyed his vast knowledge and owe him lots of gratitude for having a profound impact on this report.

I would also like to thank my Internel guide **Prof, Vibha Patel**, Institute of Technology, Nirma University, Ahmedabad for her valuable guidance.

I would also like to thank **Dr. K.R.Kotecha**, Director, Institute of Technology, Nirma University, Ahmedabad for providing me an opportunity to get an internship at Intel Technology India Private Limited, Bangalore.

I would like to thank my all faculty members for providing encouragement, exchanging knowledge during my post-graduate program.

I also owe my colleagues my teammates specially Supriyo, Sudhakar, Satya, Bimod, Radhika, Rajeend in the Intel, special thanks for helping me on this path and for making project at Intel more enjoyable.

Adodariya Vishal P. (11MCEC01)

Contents

List of Figures

Abstract

Today, the complexity of the computer systems has grown, with processors and chipsets incorporating millions of the transistors and compatible with dozens of operating system,hundreds of platform components and thousands of hardware devices and software applications. Hence the complexity of BIOS source code is increased so it is hard to develop different BIOS for each type of platform with different flavor, so need arise of one BIOS source code that can support all types of different board . POST (Power-on self-test) activated by BIOS. It runs a series of checks and diagnostics on your motherboard. The POST runs very quickly, Its hard to finds out where actually execution is stopped in which phase or which hardware is faulty or missing in the system if its stops in between. Thats why POST status codes are required which display progress and Error codes while executing the POST. By using these status codes we can track the execution of POST if serial debug log is not available.

Chapter 1

Introduction

Today, the complexity of the computer systems has grown, with processors and chipsets incorporating millions of the transistors and compatible with dozens of operating system, hundreds of platform components and thousands of hardware devices and software applications. Hence the complexity of BIOS source code is increased so it is hard to develop different BIOS code for each type of platform with different flavor.

Platform is mainly divided in to two categories one is traditional (TRAD) and another is ultrathin (ULT). Traditional has two chip solution which means different die for CPU and PCH , while in ULT, CPU and PCH are in same die which called as single chip solution.Each type of platform has different flavors like mobile, desktop, unicore servers . Each type of flavor can support different types of board, they are different in terms of components on it. like some board has 4 Dimms while some has 2 Dimms.

In this thesis report, our proposal is to develop algorithm for detection of a board dynamically which support same bios available for specific platform. Also program various GPIOs depends upon which components is used, how usb port mapping is done Etc.., so that each board can support same BIOS.

Chapter 2

literature survey

2.1 Intel Architecture - Platform Overview

2.1.1 Introduction to platform

Platform encompasses all required ingredients, features, capabilities, initiatives and technologies.
The 4 major ingredients:

- Hardware.
  - Processors, chipsets, communications, memory, boards, and systems.
- Software.
  - Operating systems (OSs), applications, firmware, and compilers.
- Technologies.
  - Hyper-Threading Technology (HT Technology), Intel Virtualization Technology, and Intel Active Management Technology (Intel AMT).
- Standards and Initiatives.
  - Wi-Fi, WiMAX, the Wireless Verification Program, and so on.

The platform is complex with lots of components on it. Every component must work as designed and there shouldn't be any conflicts between the devices on it. The Figure below [1] shows the typical diagram of Intel Client platform 2014. It comes up with single chip solution which means CPU and PCH are in single die.

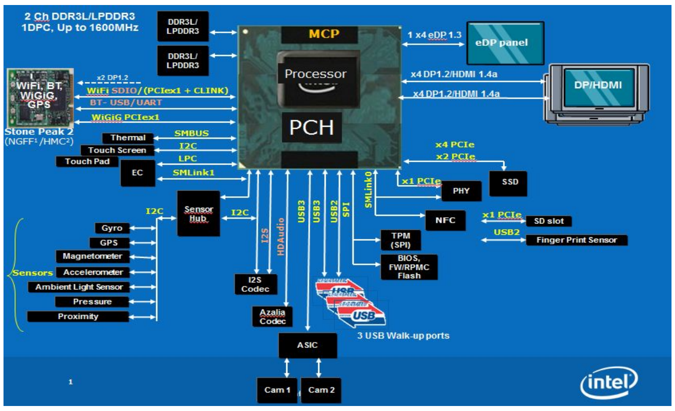


Figure 2.1: Intel platform architecture

2.1.2 Platform Controller Hub (PCH) Architecture Overview

Platform Controller Hub is a Intel microchips which is found in Intel Hub Architecture chipsets. PCHs are designed to remove problem of a bottleneck between processor and the motherboard. As cores or speed of processor is increasing, the data transfer rate between the CPU and the motherboard would achieve full bandwidth capacity hence bottleneck can remove. The speed would be limited by the Front Side Bus. As a solution, the new PCH chips are coms in to picture platform architecture transferred several functions, connections, and controllers belonging to previously northbridge and southbridge chip and insert chip called the PCH and the CPU. In summary, the PCH took over most of the tasks of the southbridge and the few remaining roles traditionally done by northbridge that have not been incorporated into the CPU package.[6]

Before the Platform Controller Hub, a motherboard would have a two piece chipset consisting of an northbridge chip called the MCH and a southbridge chip. The northbridge, also refer as an memory controller hub (MCH) later. The CPU, memory, PCI Express graphics slot(PEG), if present it would connect to it directly to platform. The northbridge connected with the CPU, with front-side bus (FSB), and RAM is connected through back-side bus, were each described by data transfer speeds. The southbridge chip now also called an I/O controller hub , connects northbridge chip to all other peripherals such as hard drives, CD and floppy drives, Ethernet, keyboard/mouse which are lower bandwidth h/w.bigger pipeline is required between the CPU and the northbridge chip would soon be unable to keep up with the CPUs speed hence system down.because CPUs gained more speed and more cores.

2.1.3 Platform Software Architecture Overview

Firmware

The definition of firmware on PC is instructions (with data) that are consumed by non-IA execution engines associated with a non-CPU hardware device. There are three main categories of firmware:

- Fixed embedded firmware: contained in ROM and hidden from platform view.
- Upgradeable embedded firmware: contained in built-in non-volatile memory with default image (code/data); upgradable during life cycle.
- Externally stored firmware: storage of the code/data is outside of the device package that executes the firmware. The external storage is likely in non-volatile memory form. Patches for embedded firmware falls into this category.

No action is required on platform SW to support fixed embedded firmware. Examples of embedded firmware components are:

- CPU microcode, uncore firmware.
- ME ROM code.

Upgradeable embedded firmware is presumed to be functional at platform build time. Discrete graphics card firmware is in this category. Upgrade tool is expected to be available for at least one of the user's SW environment. There is no known ingredient with firmware in this category.

2.2 Introduction to BIOS as ingredient

The basic input/output system (BIOS), also known as the System BIOS or ROM BIOS, is a de facto standard defining a firmware interface.[7]

The BIOS software is built into the PC, and is the first code run by a PC when powered on ('boot firmware'). The primary function of the BIOS is to set up the hardware and load and start a boot loader. When the Power button of PC is pressed, the BIOS checks and initialize system devices which can be potential boot device like the video display card, keyboard, mouse, hard disk drive, optical disc drive and other hardware. The BIOS after locates boot device in which OS is stored,like hard disk or a CD/DVD,and loads and executes that software, giving it control of the PC. This process is known as booting, also known as bootstrapping short for .

BIOS software is stored on a Flash chip on the motherboard. now a days the BIOS flash chip's contents can be flashed and upgradeable.

BIOS will also have a user interface (or UI for short). Typically this is a menu system accessed by pressing a certain key on the keyboard when the PC starts. In the BIOS UI, a user can:

- Hardware configuration.
- setting of the system clock.
- trigger system's components.
- select devices which doesn't through any Error and eligible for potential boot device.
- set various password,for example password for securing access to the BIOS UI functions itself and preventing malicious users from booting the system from unauthorized peripheral devices.

The BIOS provides a small library of basic input/output functions used to operate and control the peripherals such as the keyboard, text display functions and so forth, and these software library functions are callable by external software. In the IBM PC and AT, certain peripheral cards such as hard-drive controllers and video display adapters carried their own BIOS extension Option ROM, which provided additional functionality. Operating systems and executive software, designed to supersede this basic firmware functionality, will provide replacement software interfaces to applications.

previously BIOS works with 16-bit, 32-bit,architecture after that in 64-bit architecture eras, while new BIOS architecture comes up which is known as EFI is used for new 32-bit and 64-bit architectures. this era BIOS is used for booting up a system and for video initialization; but otherwise is not used during the ordinary running of a system, while in early systems (particularly in the 16-bit era), BIOS was used for hardware access - operating systems (notably MS-DOS) would call the BIOS rather than directly accessing the hardware. In the 32-bit era and later, operating systems instead generally directly accessed the hardware using their own device drivers.

The role of the BIOS has changed over time; some time ago BIOS is a legacy system,but now super seded by the more complex Extensible Firmware Interface (EFI),

2.2.1 Major features in the platform BIOS

Here is a (not exhaustive) list of the major features in the platform BIOS (not in execution order):

- Core initialization.
- CPU initialization (Multi-core, multi-threading).
- Memory initialization (DDR3).
- Chipset initialization (PCI, USB, etc.).
- BIOS setup and update facilities with protection.
- Various pre-OS devices.
- User Inputs (keyboard, mouse, PS/2, USB, ...).
- VBIOS (discrete, integrated, switchable).
- MEBx, AMT (Advanced Management Technology)
- Security .
- BIOS, HDD password.
- TXT.
- TPM measurement, including iTPM.

- Configuration, power and thermal management setup
- ACPI.
- APM.
- Wake from S-state.
- Hot keys.
- Boot select .
- Optical drive.
- USB.
- SATA,SSD.
- Boot manager, User OS, EFI Shell, recovery OS.

2.2.2 Power-on self-test

Power-On Self-Test (POST) refers to routines run immediately after power is applied, by nearly all electronic devices. Perhaps the most widely-known usage pertains to computing devices (personal computers, PDAs, networking devices such as routers, switches, intrusion detection systems and other monitoring devices). Other devices include kitchen appliances, avionics, medical equipment, laboratory test equipment all embedded devices. The routines are part of a device's pre-boot sequence. Once POST completes successfully, bootstrapping code is invoked.

POST includes routines to set an initial value for internal and output signals and to execute internal tests, as determined by the device manufacturer. These initial conditions are also referred to as the device's state. They may be stored in firmware or included as hardware, either as part of the design itself, or they may be part of semiconductor substrate either by virtue of being part of a device mask, or after being burned into a device such as a Programmable Logic Array (PLA).

Test results may be enunciated either on a panel that is part of the device, or output via bus to an external device. They may also be stored internally, or may exist only until the next power-down. In some cases, such as in aircraft and automobiles, only the fact that a failure occurred may be displayed (either visibly or to an on-board computer) but may also upload detail about the failure(s) when a diagnostic tool is connected. POST protects the bootstrapped code from being interrupted by faulty hardware. Diagnostic information provided by a device, for example when connected to an engine analyzer, depends on the proper function of the device's internal compo nents. In these cases, if the device is not capable of providing accurate information, subsequent code (such as bootstrapping code) may not be permitted to run. This is done to ensure that, if a device is not safe to run, it is not permitted to run.

2.2.3 EFI BIOS boot phases

BIOS boot phases are shown in fig below [3] [4]

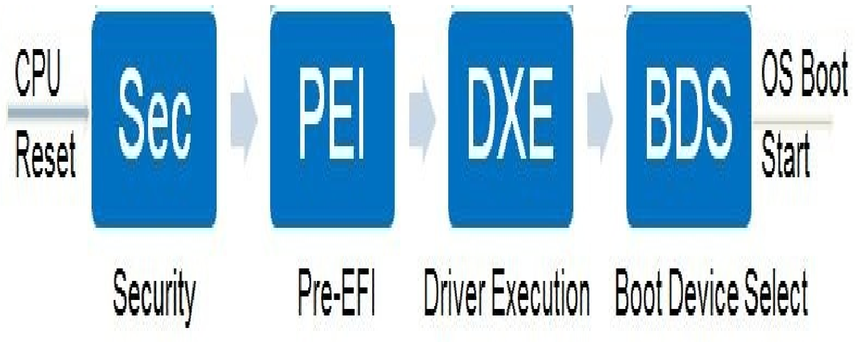


Figure 2.2: EFI BIOS boot phases

Security phase:

The Security (SEC) phase is the first phase in the PI Architecture architecture and is responsible for the following:

- Handling all platform restart events.
- Creating a temporary memory store.
- Serving as the root of trust in the system.
- Passing handoff information to the PEI Core.

In addition to the minimum architecturally required handoff information, the SEC phase can pass optional information to the PEI Core, such as the SEC Platform Information

PPI or information about the health of the processor.

Pre-EFI Initialization (PEI) Phase:

The Pre-EFI Initialization (PEI) phase of the PI Architecture specifications (hereafter referred to as the PI Architecture) is invoked quite early in the boot flow. Specifically, after some preliminary processing in the Security (SEC) phase, any machine restart event will invoke the PEI phase.The PEI phase will initially operate with the platform in a nascent state, leveraging only on processor resources, such as the processor cache as a call stack, to dispatch Pre-EFI Initialization Modules (PEIMs). These PEIMs are responsible for the following:

- Initializing some permanent memory complement.
- Describing the memory in Hand-Off Blocks (HOBs).
- Describing the firmware volume locations in HOBs.
- Passing control into the Driver Execution Environment (DXE) phase.

Philosophically, the PEI phase is intended to be the thinnest amount of code to achieve the ends listed above. As such, any more sophisticated algorithms or processing should be deferred to the DXE phase of execution.
The PEI phase is also responsible for crisis recovery and resuming from the S3 sleep state. For crisis recovery, the PEI phase should reside in some small, fault-tolerant block of the firmware store. As a result, it is imperative to keep the footprint of the PEI phase as small as possible. In addition, for a successful S3 resume, the speed of the resume is of utmost importance, so the code path through the firmware should be minimized. These two boot flows also speak to the need to keep the processing and code paths in the PEI phase to a minimum. The implementation of the PEI phase is more dependent on the processor architecture than any other phase. In particular, the more resources the processor provides at its initial or near initial state, the richer the interface between the PEI Foundation and PEIMs.

Driver execution Environment (DXE) Phase:

The Driver Execution Environment (DXE) phase is where most of the system initialization is performed. Pre-EFI Initialization (PEI), the phase prior to DXE, is responsible for initializing permanent memory in the platform so that the DXE phase can be loaded and executed. The state of the system at the end of the PEI phase is passed to the DXE phase through a list of position-independent data structures called Hand-Off Blocks (HOBs). HOBs are described in detail in the Platform Initialization Hand-Off Block Specification. There are several components in the DXE phase:

- DXE Foundation.
- DXE Dispatcher.
- A set of DXE Drivers.

The Dxe Core produces a set of Boot Services, Runtime Services, and DXE Services. The DXE Dispatcher is responsible for discovering and executing DXE drivers in the correct order. The DXE drivers are responsible for initializing the processor, chipset, and platform components as well as providing software abstractions for system services, console devices, and boot devices. These components work together to initialize the platform and provide the services required to boot an operating system. The DXE phase and Boot Device Selection (BDS) phases work together to establish consoles and attempt the booting of operating systems. The DXE phase is terminated when an operating system is successfully booted. The Dxe Core is composed of boot services code, so no code from the Dxe Core itself is allowed to persist into the OS runtime environment. Only the runtime data structures allocated by the Dxe Core and services and data structured produced by runtime DXE drivers are allowed to persist into the OS runtime environment.

Boot Device Selection (BDS) Phase:

The Boot Manager in DXE executes after all the DXE drivers whose dependencies have been satisfied have been dispatched by the DXE Dispatcher. At that time, control is handed to the Boot Device Selection (BDS) phase of execution. The BDS phase is responsible for implementing the platform boot policy.This boot policy provides flexibility that allows system vendors to customize the user experience during this phase of execution.

The Boot Manager must also support booting from a short-form device path that starts with the first node being a firmware volume device path. The boot manager must use the GUID in the firmware volume device node to match it to a firmware volume in the system. The GUID in the firmware volume device path is compared with the firmware volume name GUID. If a match is made, then the firmware volume device path can be appended to the device path of the matching firmware volume and normal boot behavior can then be used.

The BDS phase is implemented as part of the BDS Architectural Protocol. The DXE Foundation will hand control to the BDS Architectural Protocol after all of the DXE drivers whose dependencies have been satisfied have been loaded and executed by the DXE Dispatcher. The BDS phase is responsible for the following:

- Initializing console devices.
- Loading device drivers.
- Attempting to load and execute boot selections.

If the BDS phase cannot make forward progress, it will re-invoke the DXE Dispatcher to see if the dependencies of any additional DXE drivers have been satisfied since the last time the DXE Dispatcher was invoked.

2.2.4 Boot modules

Platform

- This module touches almost all components on the mother board including CPU and PCH.
- Board detection.
- Processor Power management including C-states, P-states, throttling.
- Thermal reporting.
- Implement some security features related to processor.

PCH

PCH module is used to provide following services. Intel HD Audio, SMBUS, SPI, SATA, Legacy interrupt, System Management Interrupts, System reset, Timers, USB, Display Link Etc.

Memory

This module runs in pei phase in 32bit mode.It supports detection and initialization of memory modules and complies with the requirements in the BIOS specification.

ME

Me is management engine, it provides centralized administration, also responsible for vpro technology ,integrated clock control etc.

Security

It provides trusted execution environment also provide security related function create cryptographic keys, windows bitlocker.

System Agent

This is uncore part mainly responsible for graphics also initializing System Memory, initializing Power Management, internal Graphics, internal Audio, and PCI Express, modifying SA register default values for optimal performance, SMRAM initialization.

Chapter 3

Platform Detection

3.1 Platform directory

BIOS source code is now updated to EDKII fashion as EDK has some drawbacks.The EDK could not be compiled without the entire source tree. Furthermore, the unit of distribution required the EDK as the whole tree. To solve this issue, EDKII introduced the new concept, the "Package". Using this, a release and its work can be made with "Packages" as opposed to requiring the Whole Source Tree.
A package is the minimal unit of distribution as well as providing a natural split in a big projectwhich serves various purposes. For example, from the viewpoint of hardware a developer may divide CPU/chipset/platform related definitions and drivers into three individual packages that facilitate a user's distribution and reuse. Developers also can put all modules that are independent of various platforms into a single package. Therefore, developers only need focus on platform-specific code when porting to a new platform.

EDKII and refrence code package details

1	Package	Details
2	Basetool	Provides build related tools for both EDK and EDK2. Also contains miscellaneous tools such as ECC and EOT
3	EdkCompatibilityPkg	Several security features were introduced (e.g. Authenticated Variable Service, Driver Signing, etc.)
4	EdkShellBinPkg	Provides header files and libraries that enable you to build the EDK module in UEFI 2.0 mode with EDK II Build
5	EmulationIntelRestrictedPkg	Binary images of the EFI Shell 1.0 for IA32, X64 and IPF
6	FatBinPkg	This package provides binary device drivers to support the FAT32 file system
7	FatPkg	This Platform file is used to generate the Binary Fat Drivers
8	IA32FamilyCpuPkg	This package supports IA32 family processors, with CPU DXE module, CPU PEIM, CPU S3 module,SMM modules, related libraries, and corresponding definitions
9	IntelFrameworkModulePkg	Intel Framework Module Package contains the definitions and module implementation which follows Intel EFI Framework Specification.
10	IntelFrameworkPkg	This package provides definitions and libraries that comply to Intel Framework Specifications

11	MdeModulePkg	This package provides the modules that conform to UEFI Industry standards.
12	MdePkg	provides all definitions(including functions, MACROs, structures and library classes) and libraries instances, which are defined in MDE Specification.
13	NetworkPkg	NetworkPkg provides IPv6 network stack drivers, IPsec driver, PXE driver, iSCSI driver and necessary shell applications for network configuration.
14	PcAtChipsetPkg	This package is designed to public interfaces and implementation which follows PcAt defacto standard
15	PerformancePkg	Provide performance measurement capability.
16	SecurityPgkg	This package provides functionalities like TPM, User identification (UID), secure boot and authenticated variable.
17	ShellBinPkg	This package contains binary shell application that follows UEFI specification and UEFI Shell 2.0 specification.
18	ShellPkg	This Package provides all definitions for EFI and UEFI Shell
19	SourceLevelDebugPkg	This package provides target side modules to support source level debug
20	UefiCpuPkg	This Package provides UEFI compatible CPU modules and libraries

3.2 Pkg directory structure

Each pkg contains following type of directory structure,it has one,

- Package.dec Package declaration file
- Package.dsc Platform Package build description file
- Package.fdf platform flash map description file
- Include Public header files

- Protocol Public Protocol header files
- Guid Public GUID header files
- IndustryStandard Public Industry Standard header files
- Library Public Library class header files
- Application Uefi Applications
- NameOneDxe Dxe Driver NameOne source files and INF.
- NameTwoPei Pei Driver NameTwo source files and INF.

Package Declaration File (DEC)
Each package has a single package declaration file (DEC) to define the packages public interfaces. The public interfaces are the packages public header files, GUIDs, and PCDs. The DEC has Defines, Includes, LibraryClasses, Guids, Ppis, Protocols and Pcds sections.

- The [Defines] section defines the package name and package GUID.
- The [Includes] section must list the root directory of public header file directory.
- The [LibraryClasses] section contains every library class header file in the Package
  Include
  Library directory.
- The [Guids] section specifies the Guid value for each Guid in the Package
  Include
  Guid directory.
- The [Ppis] section specifies the Guid value for each PPI in the Package
  Include
  Ppi directory.
- The [Protocols] section specifies the Guid for each Protocol in the Package
  Include
  Protocol directory.

- The PCDs are declared in different PCD sections according to their type (FeatureFlag, FixedAtBuild, PatchableInModule, Dynamic, and DynamicEx). If a PCD supports multiple PCD types, it must be declared in all supported type sections. When a PCD is declared, its data type and default value must also be specified.

Package Declaration File (DSC)
Each package usually creates another build description file (DSC). All modules can be added into DSC to be compiled and verified. DSC has the following sections:

- Defines
- LibraryClass
- PCD
- Components
- The [Defines] section sets build related information, such as the build output directory, build target, Guid, and build ARCHs.
- The [Components] section lists all modules (Drivers, Application, and Library Instances) in the platform.
- The [LibraryClasses] section specifies the chosen library instance for every library class, which is consumed by the drivers and applications in the [Components] section.
- The [PCDs] section configures PCD type and value for those PCDs used by the modules in the [Components] section. If the PCD value is same as the default value in DEC, and the PCD type has no specific requirement, the PCD may not be configured in the DSC. Its value and type will be the default setting in DEC. If all PCDs are not required in the DSC file, the [PCDs] section may be not created.

Package flashmap description file
Flash Description File (FDF) used to create firmware images, Option ROM images or bootable images for removable media. For the purposes of this design discussion, the FDF file will be also referred to as FlashMap.fdf. it defines size and offsets of firmware volumes.

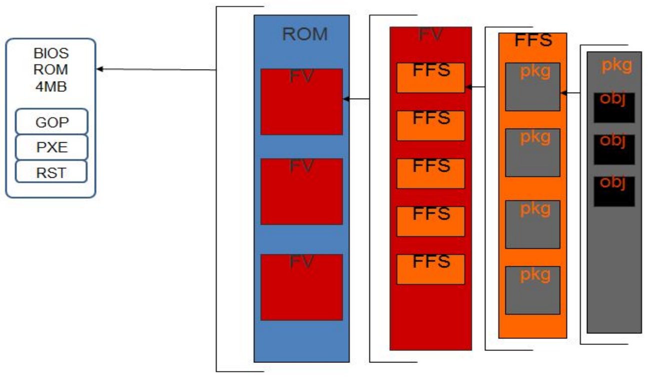


Figure 3.1: Firmware Storage

3.3 Firmware Storage

Each pkg contains one/many drivers, each driver creates object files. Many object files together create pkg.

Framework Firmware File System (FFS) is a binary layout of file storage for firmware volumes. It is a flat file system in that there is no provision for any directory hierarchy; rather, files all exist in the root directly. Files are stored end to end without any directory entry to describe which files are present.

A continues physical repository that contains firmware code/ firmware data is known as firmware device. A single physical firmware device may be divided into smaller pieces to form multiple logical firmware devices,A logical firmware device is called a firmware volume.

Rom file maps all FV files according to its address and offset value. This is final BIOS Rom image which is going to flash on platform. **Package SPI image**

Each platform has either one or two flash chips which connect through on SPI (serial peripheral interface).

SPI = BIOS + ME FW + Straps + blank space

Some platforms contain two different spi chips spi0 and spi1 each of 8mb. SPI0 has 2 main modules.

Descriptor/straps

Management Engine firmware (ME FW) .

Giga bit ethernet(Gbe)

In industry where more security requires there 5mb ME firmware used which includes

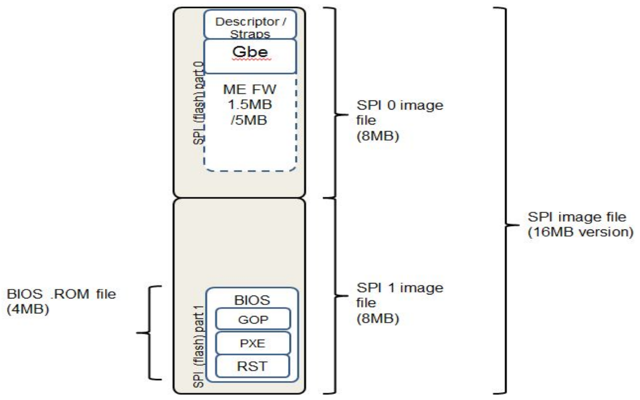


Figure 3.2: Two SPI parts

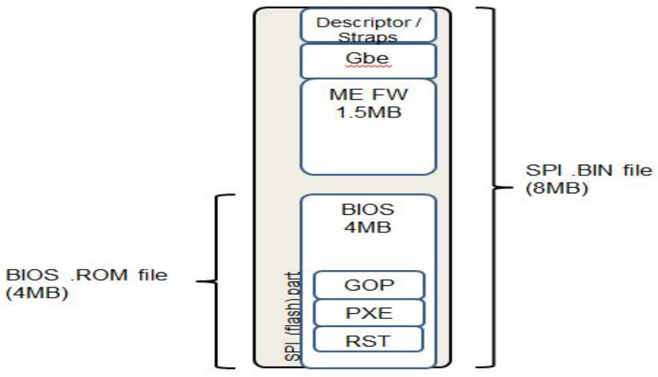


Figure 3.3: one SPI part

security feactures like Vpro and bitlocker Etc..
SPI1 has main bios image which has gop(Graphics output protocol), which enables display,makes configuration depends upon whatever display is connected to platform. pxe (pre execution environment) which is used to install operating system when platform doesnt have any operating system. It install os from server to harddrive. RST provides reset address from where cpu starts reading instructions.

The only difference is that here the ME firmware is of 1.5mb instead of 5mb. So straps,ME FW, BIOS image all resides in same SPI.

3.4 Platform detection

above figure explains the high level flow of board detection happens.

- CPU on Each flavour board has difference cpuid,so first check for valid cpuid and

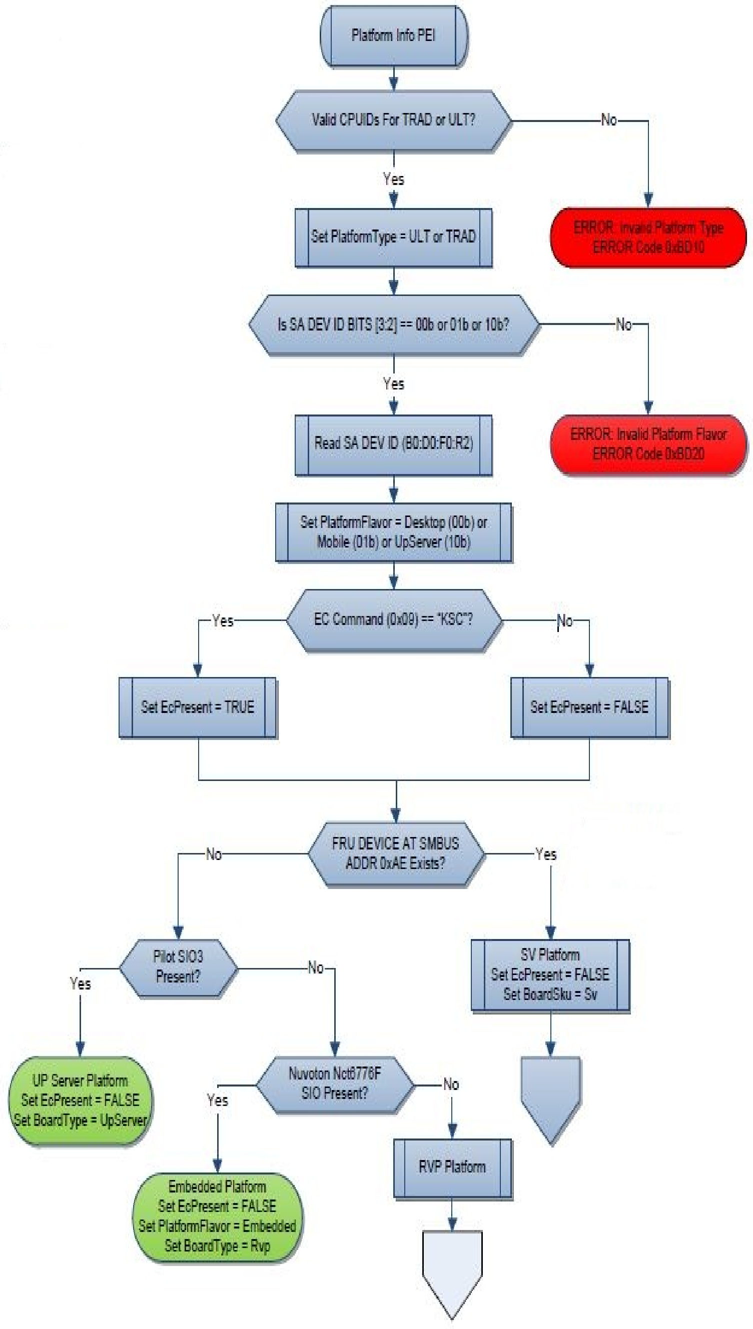
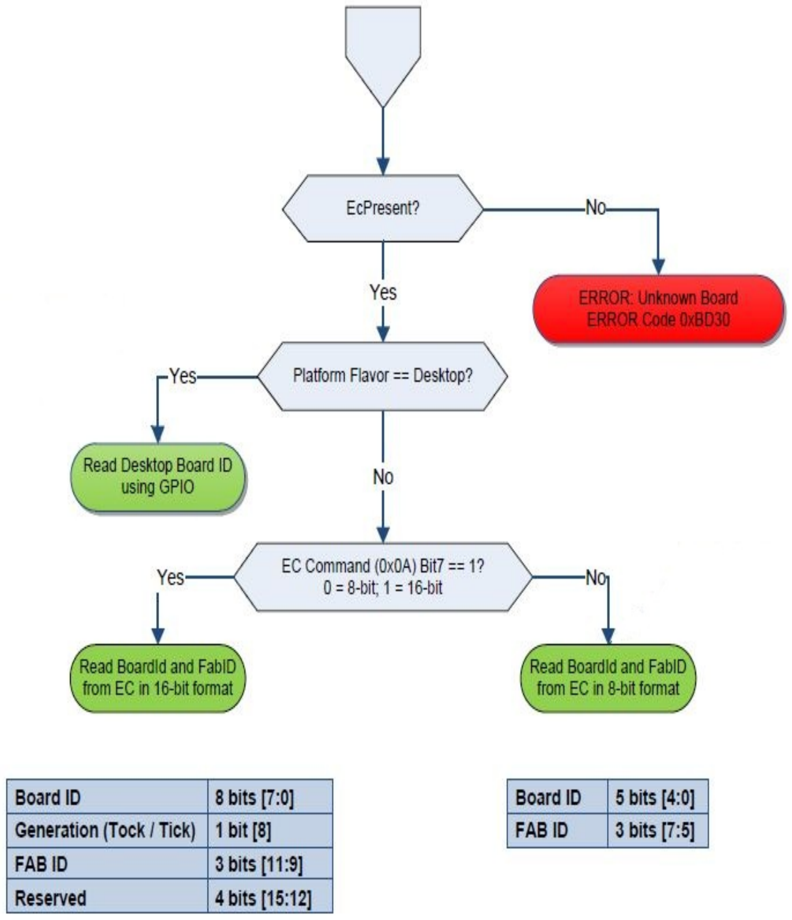

Platform Info PEI
Valid CPUIDs For TRAD or ULT?
No
ERROR: Invalid Platform Type ERROR Code 0xBD10
Yes
Set PlatformType = ULT or TRAD
Is SA DEV ID BITS [3:2] == 00b or 01b or 10b?
No
ERROR: Invalid Platform Flavor ERROR Code 0xBD20
Yes
Read SA DEV ID (B0:D0:F0:R2)
Set PlatformFlavor = Desktop (00b) or Mobile (01b) or UpServer (10b)
EC Command (0x09) == "KSC"?
Yes
No
Set EcPresent = TRUE
Set EcPresent = FALSE
FRU DEVICE AT SMBUS ADDR 0xAE Exists?
No
Yes
Pilot SIO3 Present?
SV Platform Set EcPresent = FALSE Set BoardSku = Sv
Yes
No
UP Server Platform Set EcPresent = FALSE Set BoardType = UpServer
Nuvoton Nct6776F SIO Present?
Yes
No
Embedded Platform Set EcPresent = FALSE Set PlatformFlavor = Embedded Set BoardType = Rvp
RVP Platform

Figure 3.4: high level flow of platform detection


EcPresent?
No
ERROR: Unknown Board
ERROR Code 0xBD30
Yes
Platform Flavor == Desktop?
Yes
Read Desktop Board ID using GPIO
No
EC Command (0x0A) Bit7 == 1?
0 = 8-bit; 1 = 16-bit
Yes
No
Read BoardId and FabID from EC in 16-bit format
Read BoardId and FabID from EC in 8-bit format

Board ID	8 bits [7:0]
Generation (Tock / Tick)	1 bit [8]
FAB ID	3 bits [11:9]
Reserved	4 bits [15:12]

Board ID	5 bits [4:0]
FAB ID	3 bits [7:5]

Figure 3.5: high level flow of platform detection

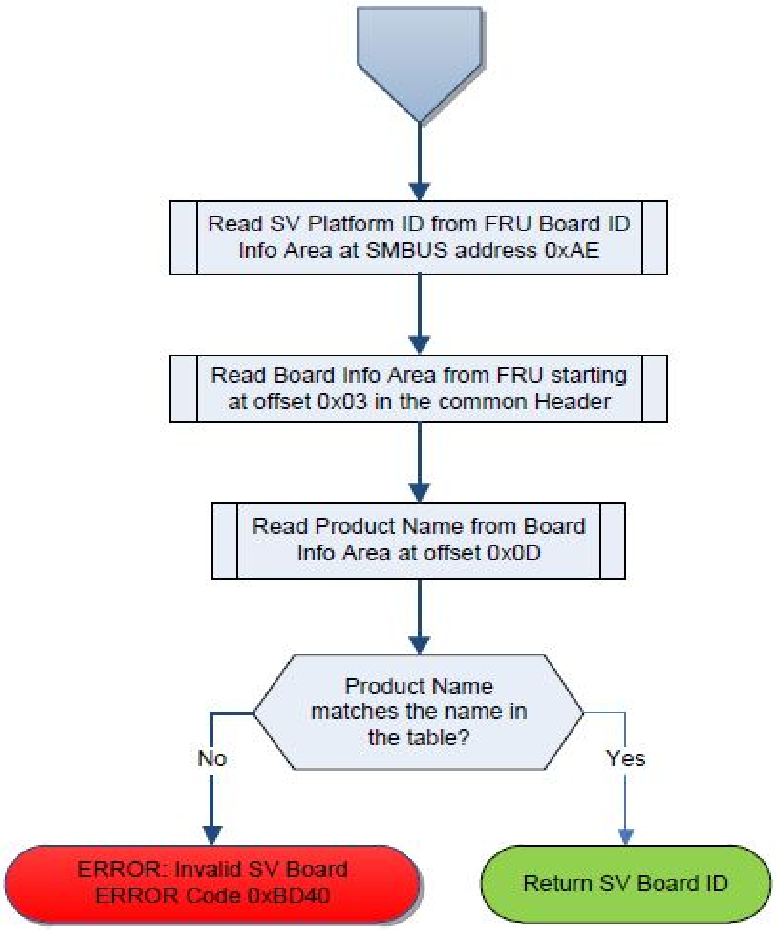

Read SV Platform ID from FRU Board ID Info Area at SMBUS address 0xAE
Read Board Info Area from FRU starting at offset 0x03 in the common Header
Read Product Name from Board Info Area at offset 0x0D
Product Name matches the name in the table?
No
Yes
ERROR: Invalid SV Board ERROR Code 0xBD40
Return SV Board ID

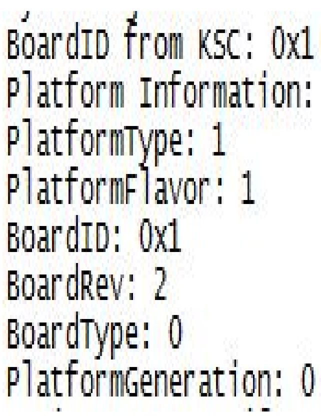
Figure 3.6: high level flow of platform detection

set platform type to ULT or TRAD.

- once platform type is decided after that needs to check weather uncore part present or not, some platform flovor board doesn't has graphics or uncore part.
- if SA is present than either board is desktop board or mobile or sever board.
- next step is to check if EC component is present in platform, if yes than set ECpresent variable true.
- if FRU device present at address AE than set Ecpresent to false ane Boardsku to SV board, but if FRU device is not present than checks for pilot SIO3.
- if pilot SIO3 is present than the board type is Upserver.
- Try to open SIO configuration space by writing 0x87 twice to port 0x2e.then try to read from this port and after that close it if SIO doesn't close than its not present on platform, but if it close successfully than platform is Embedded platform.
- if SIO not present than board is RVP/CRB board.
- if SIO is not present and platform flavor is desktop than read board id using GPIO, if not than it sends a command to check bit 7,
- if bit7 is 0 than return 8-bit boardid, else return 16 bit boardid.
- to determine weather the board is SV board or not, Read SV Platform ID from FRU Board ID Info Area at SMBUS address 0xAE,PlatformRead Board Info Area from FRU starting at offset 0x03 in the common Header, Read Product Name from Board Info Area at offset 0x0D, if product name is matched with product name stored in table than return SV board ID,

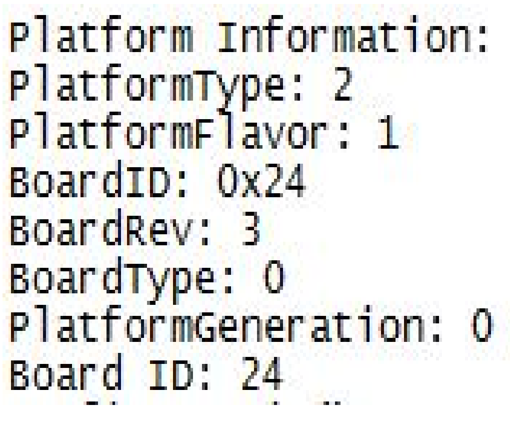
3.5 Result

board id is unique for each flavor board its either 8 bit or 16 bit.
platform type 1 for 2 chip solution and type 2 is for single chip solution.
platform flavor is either mobile,desktop,embedded, server.
boardRev is a Fab id and is of 3 bits.



```
BoardID from KSC: 0x1
Platform Information:
PlatformType: 1
PlatformFlavor: 1
BoardID: 0x1
BoardRev: 2
BoardType: 0
PlatformGeneration: 0
```

Figure 3.7: Board ID detection of TRAD board



```
Platform Information:
PlatformType: 2
PlatformFlavor: 1
BoardID: 0x24
BoardRev: 3
BoardType: 0
PlatformGeneration: 0
Board ID: 24
```

Figure 3.8: Board ID detection of ULT board

Chapter 4

POST status code

4.1 POST (Power-on self-test)

The Power On Self Test is activated by the BIOS. It runs a series of checks and diagnostics on your motherboard.
The principal duties of the main BIOS during POST are as follows.

- Verify CPU registers
- Verify the integrity of the BIOS code itself.
- Verify some basic components like DMA, timer, interrupt controller.
- Find and verify system main memory.
- Initialize BIOS.
- Identify, organize, and select which devices are available for booting

The BIOS begins its POST when the CPU is reset. The first memory location the CPU tries to execute is known as the reset vector. In the case of a hard reboot, the CPU will direct this code fetch (request) to the BIOS located on the system flash memory. For a warm boot, the BIOS will be located in the proper place in RAM and the CPU will direct the reset vector call to the RAM.

4.2 Why POST status codes are required ?

The POST runs very quickly, it hardly takes 2-3 mins to boot to OS and user will normally not even noticed that its happening unless it stops in between because presence of some faulty or some hardware is missing, When turned on the PC, it may happen system starts beeping sounds and then stopped without booting up. That is the POST telling something is wrong with the machine. Here the speaker is used because this test happens so early on, that the video isn't even activated yet!
Its hard to finds out where actually execution is stopped in which phase or which hardware is faulty or missing in the system,If execution is stops in between.
Some debug boards also doesn't contain serial interface which is used to take serial dumb of POST, In this situation also its really hard to find out where the execution is if its hangs in between, with help of last executed POST code its easy to debug execution is in which phase.

4.3 How POST status codes works ?

POST diagnostic information available by outputting a number to I/O port 80 (a screen display was not possible with some failure modes). Both progress indication and error codes were generated; in the case of a failure which did not generate a code, the code of the last successful operation was available to aid in diagnosing the problem. There are add-on cards also available that can be placed in to PCI slot and on which post codes can be seen else now a days with this comes inbuilt with debug board.

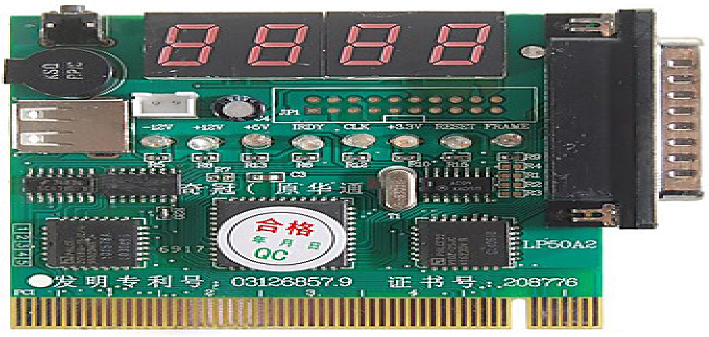


Figure 4.1: BIOS POST card for PCI slot.

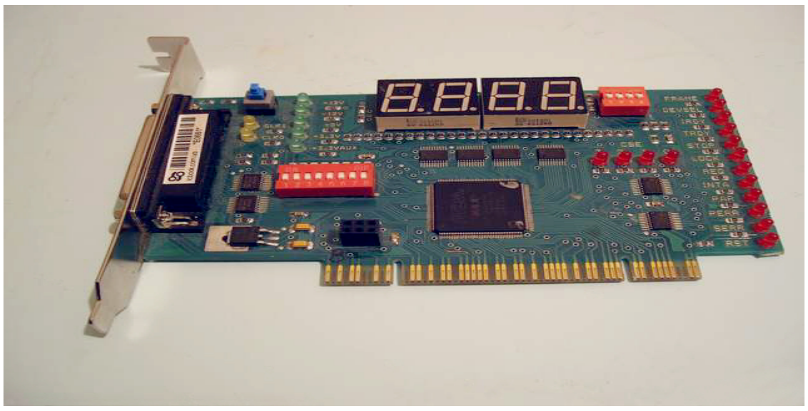


Figure 4.2: BIOS POST card for PCI slot.

4.4 Overview of POST status codes

POST status codes are mainly divided in to following categories.

- Progress codes
- Error codes
- Debug codes

For progress codes, operations correspond to activities related to the component classification. For error codes, operations correspond to exception conditions (errors). For debug codes, operations correspond to the basic nature of the debug information.
The values 0x000x0FFF are common operations that are shared by all subclasses in a class. There are also subclass-specific operations/error codes. Out of the subclass-specific operations, the values 0x10000x7FFF are reserved by this specification. The remaining

values (0x80000xFFFF) are not defined by this specification and OEMs can assign meaning to values in this range. The combination of class and subclass operations provides the complete set of operations that may be reported by an entity. The figure below demonstrates the hierarchy of class and subclass and progress, error, and debug operations.

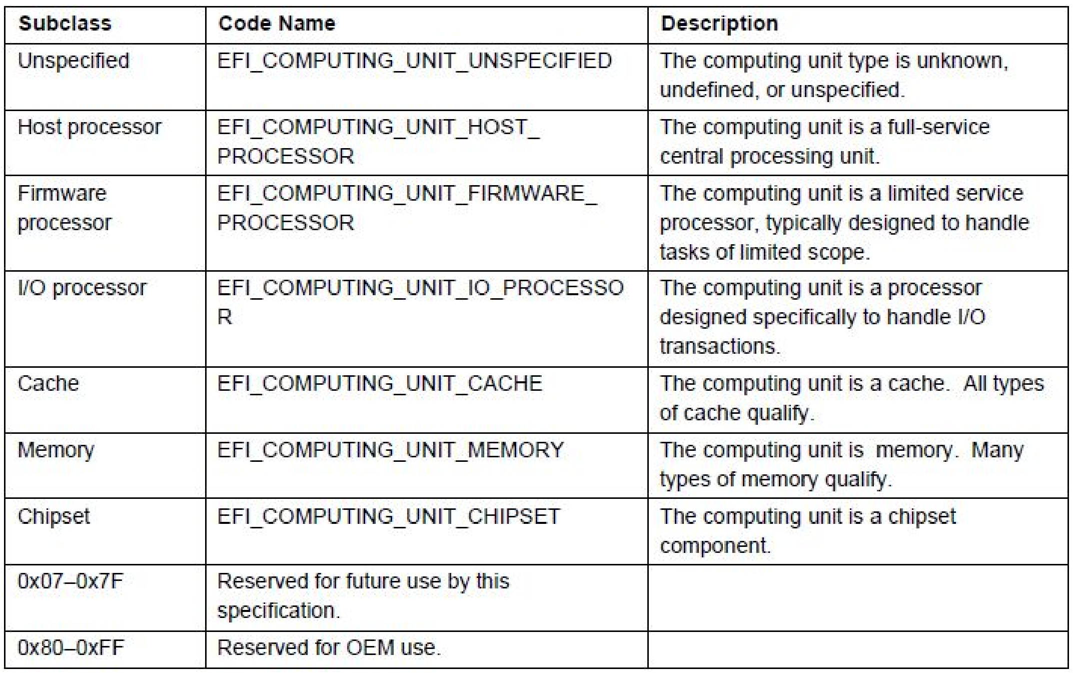
Hardware	Computing Unit	EFI_COMPUTING_UNIT
	User-Accessible Peripheral	EFI_PERIPHERAL
	I/O Bus	EFI_IO_BUS
Software	Host Software	EFI_SOFTWARE

. Each class also divided into subclasses:

4.5 Computing Unit Class

The Computing Unit class covers components directly related to system computational capabilities. Subclasses correspond to types of computational devices and resources.

Subclass Code Name Description

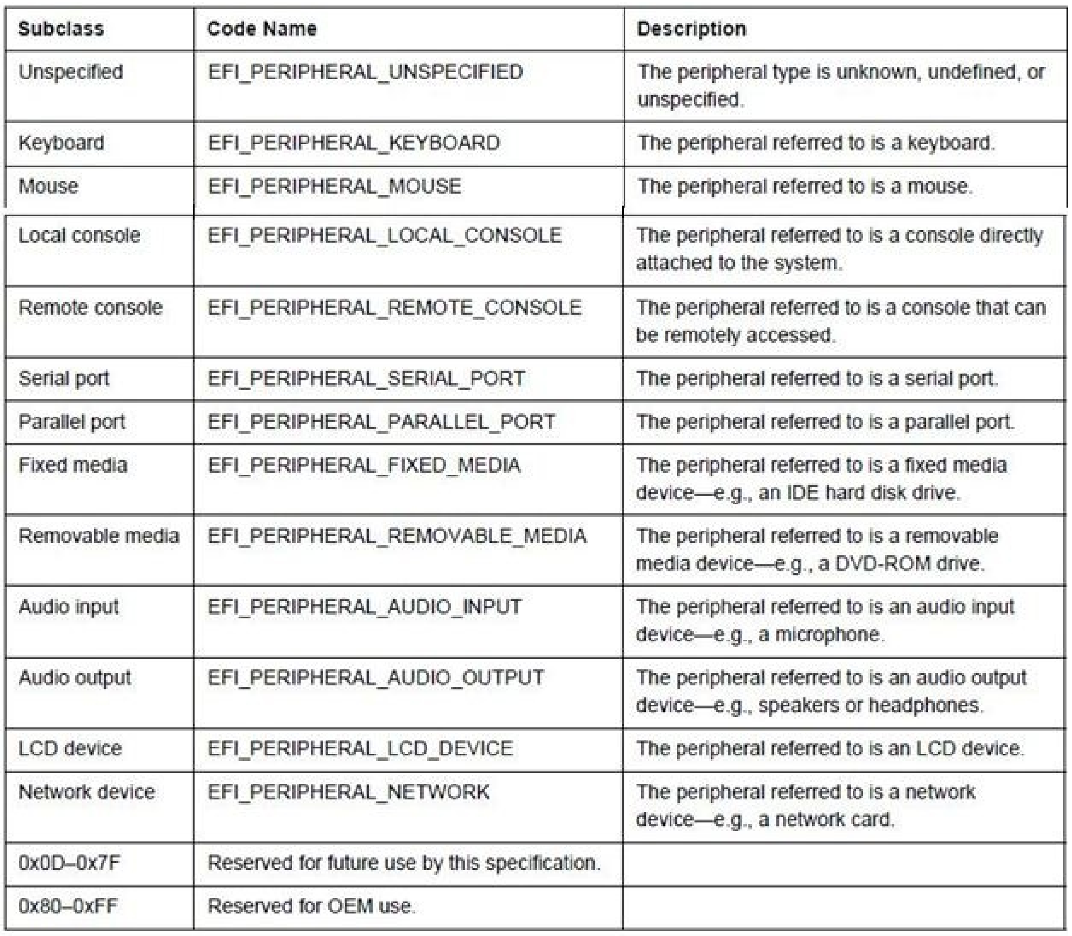


Subclass	Code Name	Description
Unspecified	EFI_COMPUTING_UNIT_UNSPECIFIED	The computing unit type is unknown, undefined, or unspecified.
Host processor	EFI_COMPUTING_UNIT_HOST_ PROCESSOR	The computing unit is a full-service central processing unit.
Firmware processor	EFI_COMPUTING_UNIT_FIRMWARE_ PROCESSOR	The computing unit is a limited service processor, typically designed to handle tasks of limited scope.
I/O processor	EFI_COMPUTING_UNIT_IO_PROCESSO R	The computing unit is a processor designed specifically to handle I/O transactions.
Cache	EFI_COMPUTING_UNIT_CACHE	The computing unit is a cache. All types of cache qualify.
Memory	EFI_COMPUTING_UNIT_MEMORY	The computing unit is memory. Many types of memory qualify.
Chipset	EFI_COMPUTING_UNIT_CHIPSET	The computing unit is a chipset component.
0x07–0x7F	Reserved for future use by this specification.	
0x80–0xFF	Reserved for OEM use.	

4.6 User-Accessible Peripheral Class

The User-Accessible Peripheral class refers to any peripheral with which the user interacts. Subclass elements correspond to general classes of peripherals.

Subclass Code Name Description

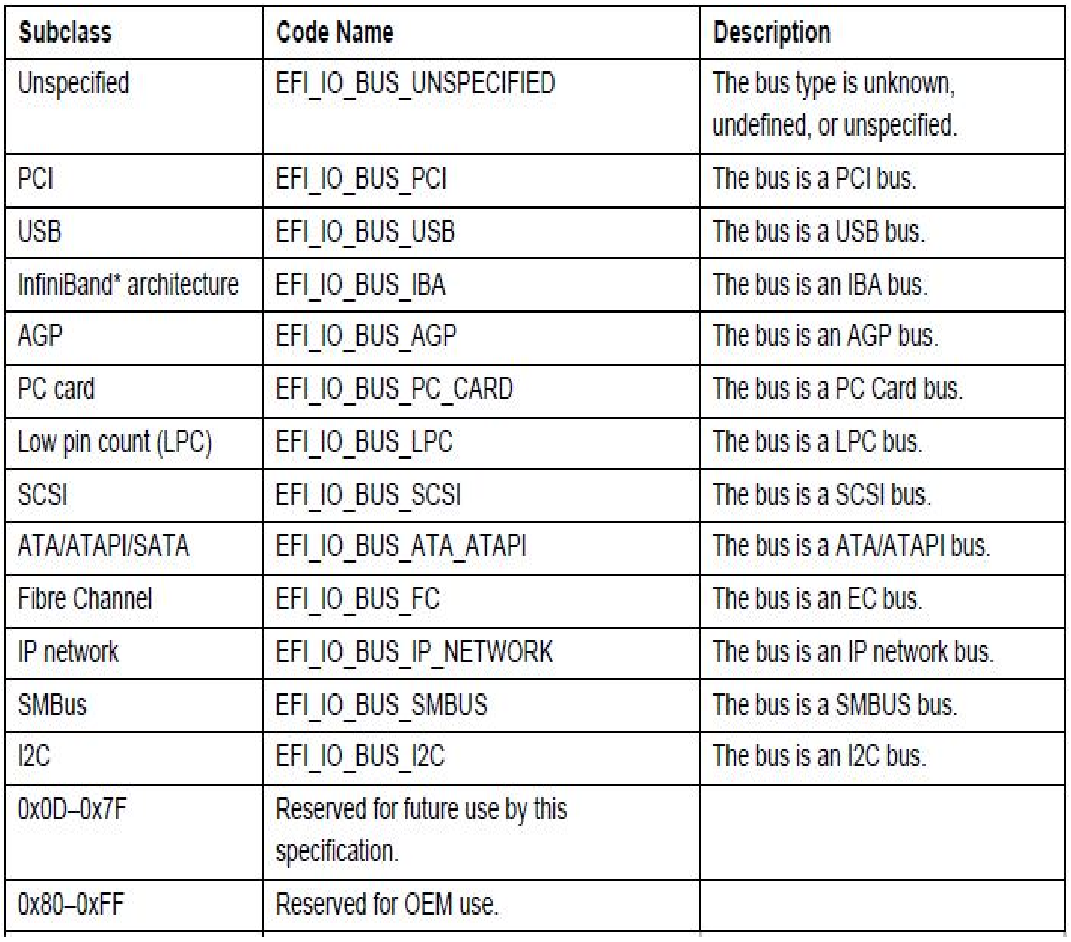


Subclass	Code Name	Description
Unspecified	EFI_PERIPHERAL_UNSPECIFIED	The peripheral type is unknown, undefined, or unspecified.
Keyboard	EFI_PERIPHERAL_KEYBOARD	The peripheral referred to is a keyboard.
Mouse	EFI_PERIPHERAL_MOUSE	The peripheral referred to is a mouse.
Local console	EFI_PERIPHERAL_LOCAL_CONSOLE	The peripheral referred to is a console directly attached to the system.
Remote console	EFI_PERIPHERAL_REMOTE_CONSOLE	The peripheral referred to is a console that can be remotely accessed.
Serial port	EFI_PERIPHERAL_SERIAL_PORT	The peripheral referred to is a serial port.
Parallel port	EFI_PERIPHERAL_PARALLEL_PORT	The peripheral referred to is a parallel port.
Fixed media	EFI_PERIPHERAL_FIXED_MEDIA	The peripheral referred to is a fixed media device—e.g., an IDE hard disk drive.
Removable media	EFI_PERIPHERAL_REMOVABLE_MEDIA	The peripheral referred to is a removable media device—e.g., a DVD-ROM drive.
Audio input	EFI_PERIPHERAL_AUDIO_INPUT	The peripheral referred to is an audio input device—e.g., a microphone.
Audio output	EFI_PERIPHERAL_AUDIO_OUTPUT	The peripheral referred to is an audio output device—e.g., speakers or headphones.
LCD device	EFI_PERIPHERAL_LCD_DEVICE	The peripheral referred to is an LCD device.
Network device	EFI_PERIPHERAL_NETWORK	The peripheral referred to is a network device—e.g., a network card.
0x0D–0x7F	Reserved for future use by this specification.	
0x80–0xFF	Reserved for OEM use.	

4.7 I/O Bus Class

The I/O bus class covers hardware buses irrespective of any software protocols that are used. At a broad level, everything that connects the computing unit to the user peripheral can be covered by this class. Subclass elements correspond to industry-standard hardware buses.

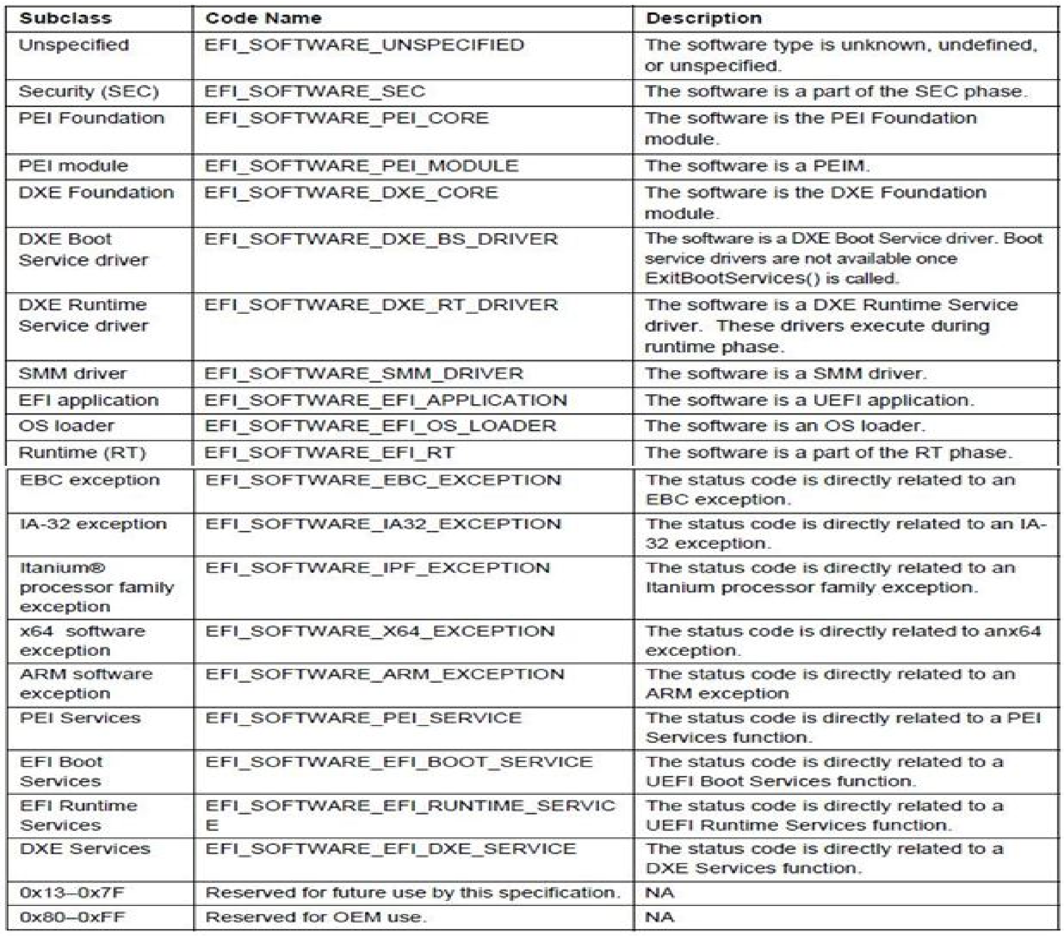
Subclass Code Name Description



Subclass	Code Name	Description
Unspecified	EFI_IO_BUS_UNSPECIFIED	The bus type is unknown, undefined, or unspecified.
PCI	EFI_IO_BUS_PCI	The bus is a PCI bus.
USB	EFI_IO_BUS_USB	The bus is a USB bus.
InfiniBand* architecture	EFI_IO_BUS_IBA	The bus is an IBA bus.
AGP	EFI_IO_BUS_AGP	The bus is an AGP bus.
PC card	EFI_IO_BUS_PC_CARD	The bus is a PC Card bus.
Low pin count (LPC)	EFI_IO_BUS_LPC	The bus is a LPC bus.
SCSI	EFI_IO_BUS_SCSI	The bus is a SCSI bus.
ATA/ATAPI/SATA	EFI_IO_BUS_ATA_ATAPI	The bus is a ATA/ATAPI bus.
Fibre Channel	EFI_IO_BUS_FC	The bus is an EC bus.
IP network	EFI_IO_BUS_IP_NETWORK	The bus is an IP network bus.
SMBus	EFI_IO_BUS_SMBUS	The bus is a SMBUS bus.
I2C	EFI_IO_BUS_I2C	The bus is an I2C bus.
0x0D–0x7F	Reserved for future use by this specification.	
0x80–0xFF	Reserved for OEM use.	

4.8 Software Classes

The Software class covers any software-generated codes.



Subclass	Code Name	Description
Unspecified	EFI_SOFTWARE_UNSPECIFIED	The software type is unknown, undefined, or unspecified.
Security (SEC)	EFI_SOFTWARE_SEC	The software is a part of the SEC phase.
PEI Foundation	EFI_SOFTWARE_PEI_CORE	The software is the PEI Foundation module.
PEI module	EFI_SOFTWARE_PEI_MODULE	The software is a PEIM.
DXE Foundation	EFI_SOFTWARE_DXE_CORE	The software is the DXE Foundation module.
DXE Boot Service driver	EFI_SOFTWARE_DXE_BS_DRIVER	The software is a DXE Boot Service driver. Boot service drivers are not available once ExitBootServices() is called.
DXE Runtime Service driver	EFI_SOFTWARE_DXE_RT_DRIVER	The software is a DXE Runtime Service driver. These drivers execute during runtime phase.
SMM driver	EFI_SOFTWARE_SMM_DRIVER	The software is a SMM driver.
EFI application	EFI_SOFTWARE_EFI_APPLICATION	The software is a UEFI application.
OS loader	EFI_SOFTWARE_EFI_OS_LOADER	The software is an OS loader.
Runtime (RT)	EFI_SOFTWARE_EFI_RT	The software is a part of the RT phase.
EBC exception	EFI_SOFTWARE_EBC_EXCEPTION	The status code is directly related to an EBC exception.
IA-32 exception	EFI_SOFTWARE_IA32_EXCEPTION	The status code is directly related to an IA-32 exception.
Itanium® processor family exception	EFI_SOFTWARE_IPF_EXCEPTION	The status code is directly related to an Itanium processor family exception.
x64 software exception	EFI_SOFTWARE_X64_EXCEPTION	The status code is directly related to anx64 exception.
ARM software exception	EFI_SOFTWARE_ARM_EXCEPTION	The status code is directly related to an ARM exception
PEI Services	EFI_SOFTWARE_PEI_SERVICE	The status code is directly related to a PEI Services function.
EFI Boot Services	EFI_SOFTWARE_EFI_BOOT_SERVICE	The status code is directly related to a UEFI Boot Services function.
EFI Runtime Services	EFI_SOFTWARE_EFI_RUNTIME_SERVIC E	The status code is directly related to a UEFI Runtime Services function.
DXE Services	EFI_SOFTWARE_EFI_DXE_SERVICE	The status code is directly related to a DXE Services function.
0x13–0x7F	Reserved for future use by this specification.	NA
0x80–0xFF	Reserved for OEM use.	NA

4.9 Implementation

Each postcode is falls into specific subclass and class, each class has its unique value which is predefined.
There are four main classes.

- Computing Unit.
- User-Accessible Peripherals.
- I/O Bus.
- Software Host Software.

#define EFI_COMPUTING_UNIT 0x00000000
#define EFI_PERIPHERAL 0x01000000
#define EFI_IO_BUS 0x02000000
#define EFI_SOFTWARE 0x03000000.
This is the way how computing unit classs subclasses are defined:
#define EFI_COMPUTING_UNIT_UNSPECIFIED
(EFI_COMPUTING_UNIT — 0x00000000)
#define EFI_COMPUTING_UNIT_HOST_PROCESSOR
(EFI_COMPUTING_UNIT — 0x00010000)
#define EFI_COMPUTING_UNIT_FIRMWARE_PROCESSOR
(EFI_COMPUTING_UNIT — 0x00020000)
#define EFI_COMPUTING_UNIT_IO_PROCESSOR
(EFI_COMPUTING_UNIT — 0x00030000)
#define EFI_COMPUTING_UNIT_CACHE
(EFI_COMPUTING_UNIT — 0x00040000)
#define EFI_COMPUTING_UNIT_MEMORY
(EFI_COMPUTING_UNIT — 0x00050000)
#define EFI_COMPUTING_UNIT_CHIPSET
(EFI_COMPUTING_UNIT — 0x00060000)
To implement the POST status codes, array of structure have been created. for each process codes, and error codes. (STATUS_CODE_TO_DATA_MAP mPostCodeProgressMap[]).
..

Report_status_code(Type,Value), this is the function which is used to write on port80 .

Here type is weather status code is process code or error code.

And value is hex number which is displayed

Example:

Progress code:

Progress code for execution reached to DXE phase.

{ PEI_CPU_AP_INIT, 0x35 },

This status falls into compting unit class and host processor subclass.

#define PEI_CPU_AP_INIT (EFI_COMPUTING_UNIT_HOST_PROCESSOR —

EFI_CU_HP_PC_AP_INIT)

For this status code function call is.

REPORT_STATUS_CODE (

EFI_PROGRESS_CODE,

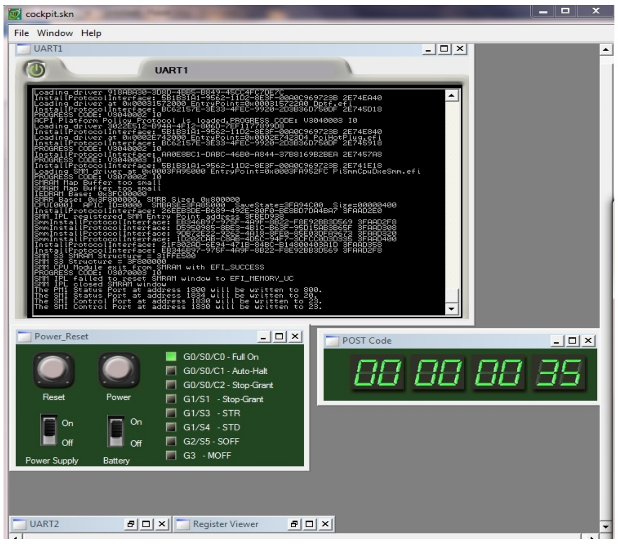
EFI_COMPUTING_UNIT_HOST_PROCESSOR — EFI_CU_HP_PC_AP_INIT

)

Which will show value 35 on the status codes display.

Result:

Error code:



```
Loading driver 918ABA30-3D8D-4BB5-B849-45CC4FC7DE7C
InstallProtocolInterface: 5B1B31A1-9562-11D2-8E3F-00A0C969723B 2E74EA40
Loading driver at 0x00031572000 EntryPoint=0x000315722A0 Dptf.efi
InstallProtocolInterface: BC62157E-3E33-4FEC-9920-2D3B36D750DF 2E745D18
PROGRESS CODE: V3040002 I0
ACPI Platform Policy Protocol is loaded.PROGRESS CODE: V3040003 I0
Loading driver 3022E512-B94A-4F12-806D-7EF1177899D8
InstallProtocolInterface: 5B1B31A1-9562-11D2-8E3F-00A0C969723B 2E74E840
Loading driver at 0x0002E742000 EntryPoint=0x0002E7423D4 PciHotPlug.efi
InstallProtocolInterface: BC62157E-3E33-4FEC-9920-2D3B36D750DF 2E745918
PROGRESS CODE: V3040002 I0
InstallProtocolInterface: AA0E8BC1-DABC-46B0-A844-37B8169B2BEA 2E7457A8
PROGRESS CODE: V3040003 I0
InstallProtocolInterface: 5B1B31A1-9562-11D2-8E3F-00A0C969723B 2E741E18
Loading SMM driver at 0x0003FA95000 EntryPoint=0x0003FA952FC PiSmmCpuDxeSmm.efi
PROGRESS CODE: V3070002 I0
SMRAM Map Buffer too small
SMRAM Map Buffer too small
IEDRAM Base: 0x3FC00000
SMRR Base: 0x3F800000, SMRR Size: 0x800000
CPU[000]  APIC ID=0000  SMBASE=3FA85000  SaveState=3FA94C00  Size=00000400
InstallProtocolInterface: 26EEB3DE-B689-492E-80F0-BE8BD7DA4BA7 3FAAD2E0
SMM IPL registered SMM Entry Point address 3FBED938
SmmInstallProtocolInterface: EB346B97-975F-4A9F-8B22-F8E92BB3D569 3FAAD2F8
SmmInstallProtocolInterface: D5950985-8BE3-4B1C-B63F-95D15AB3B65F 3FAAD308
SmmInstallProtocolInterface: 9DB72E22-9262-4A18-8FE0-85E03DFA9673 3FAAD320
SmmInstallProtocolInterface: 1D202CAB-C8AB-4D5C-94F7-3CFCC0D3D335 3FAAD400
InstallProtocolInterface: 21F302AD-6E94-471B-84BC-B14800403A1D 3FAAD358
InstallProtocolInterface: EB346B97-975F-4A9F-8B22-F8E92BB3D569 3FAAD2F8
SMM S3 SMRAM Structure = 31FFE500
SMM S3 Structure = 3F800000
SMM CPU Module exit from SMRAM with EFI_SUCCESS
PROGRESS CODE: V3070003 I0
SMM IPL failed to reset SMRAM window to EFI_MEMORY_UC
SMM IPL closed SMRAM window
The PM1 Status Port at address 1800 will be written to 800.
The SMI Status Port at address 1834 will be written to 20.
The SMI Control Port at address 1830 will be written to 23.
The SMI Control Port at address 1830 will be written to 23.
```

Figure 4.3: Progress code for execution reached to DXE phase

Error code for memory is installed or not:

PEI_MEMORY_NOT_INSTALLED, 0x55 ,

This status code is falls into software class and PEI foundation subclass.

#define PEI_MEMORY_NOT_INSTALLED (EFI_SOFTWARE_PEI_SERVICE —
EFI_SW_PEI_CORE_EC_MEMORY_NOT_INSTALLED)

For this status code function call is.

REPORT_STATUS_CODE (

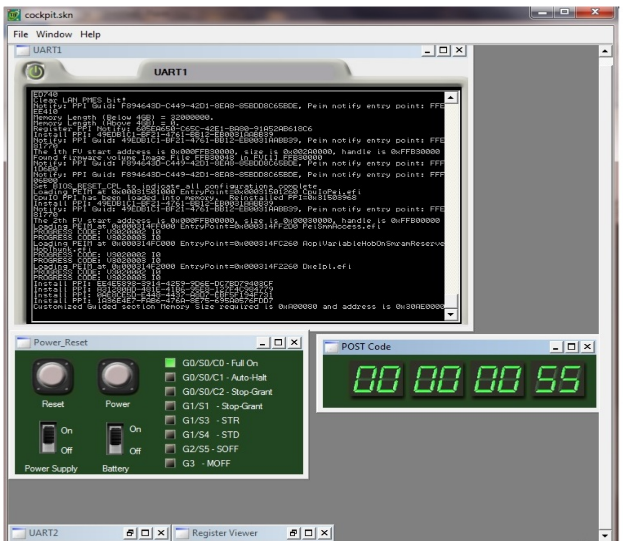
EFI_ERROR_CODE,

EFI_SOFTWARE_PEI_SERVICE — EFI_SW_PEI_CORE_EC_MEMORY_NOT_INSTALLED

)

Which will show value 55 on the status codes display.

Result:



```
ED740
Clear LAN PMES bit!
Notify: PPI Guid: F894643D-C449-42D1-8EA8-85BDD8C65BDE, Peim notify entry point: FFE
EE410
Memory Length (Below 4GB) = 32000000.
Memory Length (Above 4GB) = 0.
Register PPI Notify: 605EA650-C65C-42E1-BA80-91A52AB618C6
Install PPI: 49EDB1C1-BF21-4761-BB12-EB0031AABB39
Notify: PPI Guid: 49EDB1C1-BF21-4761-BB12-EB0031AABB39, Peim notify entry point: FFE
81770
The 1th FV start address is 0x000FFB30000, size is 0x002A0000, handle is 0xFFB30000
Found firmware volume Image File FFB30048 in FV[1] FFB30000
Notify: PPI Guid: F894643D-C449-42D1-8EA8-85BDD8C65BDE, Peim notify entry point: FFF
1D6B0
Notify: PPI Guid: F894643D-C449-42D1-8EA8-85BDD8C65BDE, Peim notify entry point: FFF
06B00
Set BIOS_RESET_CPL to indicate all configurations complete
Loading PEIM at 0x00031501000 EntryPoint=0x00031501260 CpuIoPei.efi
CpuIO PPI has been loaded into memory.  Reinstalled PPI=0x31503968
Install PPI: 49EDB1C1-BF21-4761-BB12-EB0031AABB39
Notify: PPI Guid: 49EDB1C1-BF21-4761-BB12-EB0031AABB39, Peim notify entry point: FFE
81770
The 2th FV start address is 0x000FFB00000, size is 0x00030000, handle is 0xFFB00000
Loading PEIM at 0x000314FF000 EntryPoint=0x000314FF2D0 PeiSmmAccess.efi
PROGRESS CODE: V3020002 I0
PROGRESS CODE: V3020003 I0
Loading PEIM at 0x000314FC000 EntryPoint=0x000314FC260 AcpiVariableHobOnSmramReserve
HobThunk.efi
PROGRESS CODE: V3020002 I0
PROGRESS CODE: V3020003 I0
Loading PEIM at 0x000314F2000 EntryPoint=0x000314F2260 DxeIpl.efi
PROGRESS CODE: V3020002 I0
PROGRESS CODE: V3020003 I0
Install PPI: EE4E5898-3914-4259-9D6E-DC7BD79403CF
Install PPI: A31280AD-481E-41B6-95E8-127F4C984779
Install PPI: 0AE8CE5D-E448-4437-A8D7-EBF5F194F731
Install PPI: 1A36E4E7-FAB6-476A-8E75-695A0576FDD7
Customized Guided section Memory Size required is 0xA00080 and address is 0x30AE0000
```

Figure 4.4: Error code for memory is installed or not

Chapter 5

Conclusion and Future Work

This report includes overview of BIOS, phases of BIOS , BIOS code tree , Pkg directory structure, detection of platform for various available flavors of boards of each platform as all the boards are going to use same BIOS image.
With usage of POST status codes its easy to track execution status of POST process also when serial debug log is not available or execution hangs in between or some hardware is faulty/missing.

Up till now, we have implement platform detection for Intels 2014 (Broadwell) platform, also implement POST status codes for some major critical events of BIOS, we will implement POST status codes for Intels refcodepkg, By doing this we can track execution is exactly in which module (Like CPU,PCH, Etc..)

Bibliography

[1] Intel Developers,2014 client Platform Architecture Document(PAD),Intel Technology India Pvt. Ltd.

[2] Intel Developers,2014 client Product Requirement Document(PRD),Intel Technology India Pvt. Ltd.

[3] Michael othman,Tim Lewis,Vincent Zimmer, Robert Hale, Harnessing the UEFI Shell, Intel press ,March 2010.

[4] Unified Extensible firmware Interface Specifaction, version 2.3,May 2009

[5] Advanced Configuration And Power Interface Specification, Revision 5.0, December 2011.

[6] Intel 64 and IA-32 Architectures Software Developers Manual volume1:Basic Architecture.

[7] EDKII Module Writers Guide,Revision 7.0,March 2011

[8] EDKII Information file specification,Revision 1.22,June 2012

[9] EDKII Description file specification,Revision 1.22,June 2012

[10] Intel Atom Processor Z670 with Intel SM35 Express Chipset, User Guide,july 2011 Development Kit

[11] EDKII Firmware File system specification,Revision 1.22,June 2012

[12] Platform Initialization Shared Architectural Elements,Vol 3,June 2012 Development Kit