Uncore IP BIOS Development for Intel's Next Generation Processor

Submitted By

Priyanka Bhati

18MCEC02

DEPARTMENT OF COMPUTER ENGINEERING

INSTITUTE OF TECHNOLOGY

NIRMA UNIVERSITY

AHMEDABAD-382481

May 2020

Uncore IP BIOS Development for Intel's Next Generation Processor

Major Project

Submitted in partial fulfillment of the requirements

for the degree of

Master of Technology in Computer Science and Engineering

Submitted By

Priyanka Bhati

(18MCEC02)

Under the guidance of

External Project Guide:

Mr. Vishal Adodariya

BIOS Engineer,

Intel Technology India Pvt. Ltd.

Bangalore.

Internal Project Guide:

Dr.Priyanka Sharma

Professor, CSE Department

Institute of Technology,

Nirma University, Ahmedabad.

DEPARTMENT OF COMPUTER ENGINEERING

INSTITUTE OF TECHNOLOGY

NIRMA UNIVERSITY

AHMEDABAD-382481

May 2020

Certificate

This is to certify that the major project entitled **"Uncore IP BIOS Development for Intel's Next Generation Processor"** submitted by **Priyanka Bhati (18MCEC02)**, towards the partial fulfillment of the requirements for the award of degree of Master of Technology in Computer Science and Engineering of Nirma University, Ahmedabad, is the record of work carried out by him 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 part-I, to the best of my knowledge, haven't been submitted to any other university or institution for award of any degree or diploma.

Date: Place: Ahmedabad

Dr. Priyanka Sharma
Internal Guide & Professor,
Coordinator M.Tech - CSE,
Institute of Technology,
Nirma University, Ahmedabad.

Mr. Vishal Adodariya
External Guide,
BIOS Engineer,
Intel Technology India Pvt. Ltd.
Bangalore.

Dr. Madhuri Bhavsar
Professor and Head,
CE Department,
Institute of Technology,
Nirma University, Ahmedabad.

Dr Rajesh N Patel
I/C Director,
Institute of Technology,
Nirma University, Ahmedabad

Statement of Originality

I, **Priyanka Bhati**, **18MCEC02**, give undertaking that the Major Project entitled **"Uncore IP BIOS Development for Intel's Next Generation Processor"** submitted by me, towards the partial fulfillment of the requirements for the degree of Master of Technology in **Computer Science & Engineering** of Institute of Technology, Nirma University, Ahmedabad, contains no material that has been awarded for any degree or diploma in any university or school in any territory to the best of my knowledge. It is the original work carried out by me and I give assurance that no attempt of plagiarism has been made.It contains no material that is previously published or written, except where reference has been made. I understand that in the event of any similarity found subsequently with any published work or any dissertation work elsewhere; it will result in severe disciplinary action.

_____

Signature of Student

Date:

Place:

_____

Endorsed by

Dr. Priyanka Sharma

(Signature of Guide)

Acknowledgements

It gives me immense pleasure in expressing thanks and profound gratitude to **Dr. Priyanka Sharma**, Associate Professor, Computer Engineering Department, Institute of Technology, Nirma University, Ahmedabad for her valuable guidance and continual encouragement throughout this work. The appreciation and continual support she has imparted has been a great motivation to me in reaching a higher goal. Her guidance has triggered and nourished my intellectual maturity that I will benefit from, for a long time to come.

It gives me an immense pleasure to thank **Dr. Madhuri Bhavsar**, Hon'ble Head of Computer Engineering Department, Institute of Technology, Nirma University, Ahmedabad for her kind support and providing basic infrastructure and healthy research environment.

A special thank you is expressed wholeheartedly to **Dr. Alka Mahajan**, Hon'ble Director, Institute of Technology, Nirma University, Ahmedabad for the unmentionable motivation she has extended throughout course of this work.

I would also thank the Institution, all faculty members of Computer Engineering Department, Nirma University, Ahmedabad for their special attention and suggestions towards the project work.

- Priyanka Bhati
18MCEC02

Regd. Office:
Intel Technology India Private Limited
23-56P, Outer Ring Road,
Devarabeesanahalli, Varthur Hobli
Bellandur Post
Bangalore 560 103, India
CIN-U85110KA1997PTC021606

Tel: +91-80-2605 3000
Fax: +91-80-2605 6190
website: www.intel.in

To Whomsoever It May Concern

WWID: 11899005

Employee Name: Priyanka Bhati

Internship Dates: 6/3/2019 to 5/22/2020

The letter is to confirm the mentioned above has undergone internship at Intel Technology India Pvt. Ltd. Bangalore

We wish you all the best for your future assignments.

Yours Sincerely,

Authorized Signatory Vasanthi Naidu

Date: June 28, 2020

Place: **Bangalore**

Abstract

Intel's System on chip(SoC) features a new set of Intel Uncore Intellectual Property (IP) for every generation. With this enhancement's it becomes difficult to maintain the commonality between the different project running in parallel. This project involve working on development of new features for Intel's upcoming processor, focusing on BIOS development for IPs residing on North or Uncore part of SOC. The main purpose is to create a single silicon package that supports all the upcoming platforms having different generation of IP's. There is a need to identify the commonality between current generation with next generation, so that same IP code can be plug/unplug into next generation based on the requirement. The IP based silicon support architecture allows us to reuse firmware code for IP across dies. The idea of having convergence in IP code, it will reduce efforts while enabling IP on new platform/generation. The single silicon package is intended to provide chipset initialization code for an arbitrary number of platforms at once. Debugging is an important aspect for a developer. For every phase of software life-cycle, debug tool plays an important aspect. For the Unified Extensible Firmware Interface(UEFI) development there requires a need for developing debug tool which is platform independent and which cuts the cost of time and manual validation.

Abbreviations

BIOS	Basic Input Output System.
SoC	System on Chip.
IP	Intellectual Property.
UEFI	Unified Extensible Firmware Interface.
PCIe	Peripheral Component Interface express.
GPE	Graphics Processing Engine.
DMI	Direct Media Interface.
DXE	Driver Execution Environment.
PEI	Pre-EFI Initialization.
EDK II	Extensible Firmware Interface Developer Kit II
EFI	Extensible Firmware Interface.
OS	Operating System.
GPE	Graphics Processing Engine.

Contents

List of Figures

Chapter 1

Introduction

1.1 Uncore Intellectual Properties

Intel System on a Chip (SoC) features a new set of Intel Uncore Intellectual Property (IP) for every generation. The term "Uncore" is defined as the part of the processor design which keeps the high-performance cores together with the platform to expand the scope of the product.The Uncore includes all the chip components outside the CPU core: multi-core memory coherence controller, system memory controllers, large shared cache, socket-to-socket interconnects, and the chip-level clocking, debug mechanisms and power delivery that tie the various on-die components together[4]. The Uncore encompasses system agent (SA), memory and Uncore agents such as graphics controller, display controller, memory controller and Input Output (IO). The Uncore IPs are Peripheral Component Interface Express (PCIe), Graphics Processing Engine (GPE), Thunderbolt, Imaging Processing Agent (IPU), North Peak (NPK), Virtualization Technology for directed-IO (Vt-d), Volume Management Device (VMD).

Data communication system has been evolved for the purpose of transferring data to and fro between host and peripheral devices via PCIe. PCIe is designed to replace the previous PCI standards. Thunderbolt is a hardware interface developed by Intel which provides connection of external peripherals to the system. It combines Display Port (DP) and PCIe into two serial signals, also providing DC power through all in one cable. Graphics Processing Engine (GPE), shared graphics solutions, Integrated graphics, integrated graphics processors (IGP) or unified memory architecture (UMA) uses the

section of a computer's system RAM instead of the dedicated graphics memory. GPEs can be integrated onto the motherboard as part of the chipset. Virtual Technology for Directed-IO (Vt-d) is an input/output memory management unit (IOMMU) which permits the guest virtual machines to utilize peripheral devices, such as accelerated graphics cards, Ethernet, and hard-drive controllers, with the help of and interrupt remapping.

1.2 Objective

The idea of creating a single silicon package which will increase the re-usability of common code across different platforms.While all the code needed for all supported platforms is present, only a subset of that code will be compiled at any one time. Depending on which platform the code is being compiled for, the subset that is compiled will differ. This package is intended to support multiple concurrently active projects. Hence it is expected that code changes supporting different projects will be constantly made in parallel.Also, to support the concept of platform independent creating a debug tool which can validate the registers programmed based on the platform which helps in reducing the debugging time.

Chapter 2

Literature Survey

2.1 History of Intel Architecture

Beginning with the eight bit, now the architecture of Intel contains of 32-bit and 64-bit microprocessors that are utilized for various variants of applications, various requirements on the basis of performance, cost, power levels. The success of Intel architecture is mainly because of its compatibility. Each generation of microprocessor provides backward compatibility with older chips and software's with addition of new features. With this compatibility development teams, programmers and engineers are able to reuse the software's and can focus more on the new features for better productivity.

In the span of more than 40+ years many changes have occurred in Intel architecture chips. Earlier the chip-sets got were given technical part numbers like 8086, 80386, or 80486 [1]. In reference to the last two digits of each chip's part number "x86 architecture" commonly name was given. In 1993, the "x86" naming convention gave the product names such as Intel® Pentium® processor, Intel® Celeron® processor, Intel® Core™ processor, and Intel® Atom™ processor [1].

Each new generation adds unique feature with the previous basic features. Consider as an example of the processor named Pentium. It supported the extensions for multimedia i.e. MMX technology which has enhanced the sound and video processing. Today on most of the of the Intel processors, multilevel caches, power-management features and encryption/decryption extensions are found on it as Floating-point units (FPUs) became

the standard for Intel Architecture[1].

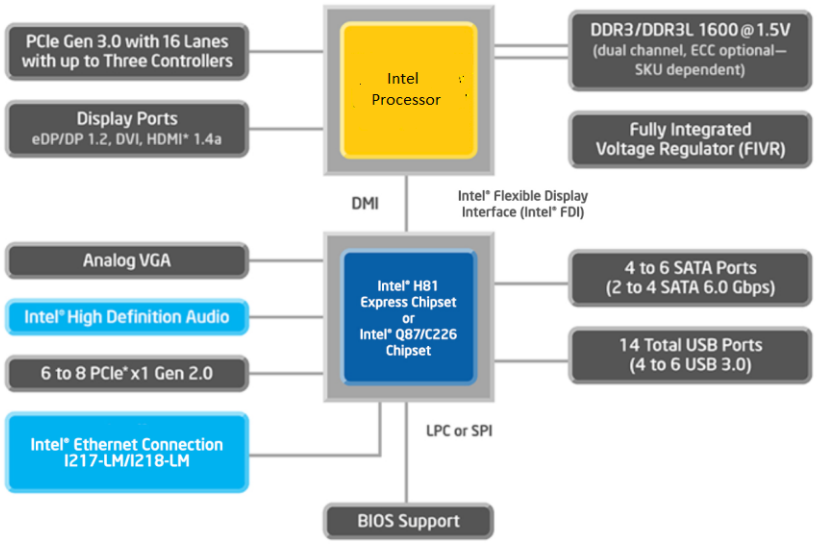

Figure 2.1: Intel System Architecture

Intel architecture includes 2 main components: the microprocessor chip and the companion chip referred as platform controller hub (PCH). Previously, Intel Processors has 2 supportive chips: the north bridge (MCH) and the south bridge (ICH). Presently, north bridge functionality are included within the processor and therfore the south bridge is replaced by the PCH which is more capable. In SoC, there's is no PCH rather all the functionality is included in the processor itself.

The figure 2.2 shows the internal block diagram of the processor highlighting its four cores each having its own caches that are L1 and L2 and one shared caches which is L3. The cache coherence is done by the DRAM controller which is on chip. If the requested data is not present at the address of the cache or data is new then the memory controller will send the data at the address requested. The transfers of data are always 64 bit wide between the processor and the memory.

In the Intel architecture system, the PCI express has the bandwidth which is the highest I/O interface. Depending on the processor used number of PCIe lanes are decided, mostly it will be in x4. The PCIe width is mainly of 16 lanes because the maximum width for PCIe graphics card is the same. The very first PCIe specification has the data

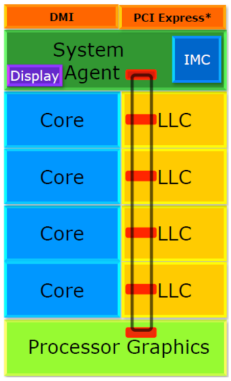

Figure 2.2: Intel processor internal block diagram [1]

rates of 2.5 GT/sec (250 MB/sec) per lane, which is an equivalent to DMI whereas the second generation of PCIe multiples by two the data rate to 5 GT/sec (500 MB/sec) [1].

The Direct Media Interface (DMI) bus which has very high-end speed and provides point to point connection link between the two chips as shown in figure 2.1. It supports rate od transfer rates as 2 GB/second over each of two unidirectional lanes [1]. It has 4 lanes that is Transmit and Receive (2) differential signaling (2) which is equivalent to 16 pins and also supports in signaling of 2.5 billion transfers/ sec. It further integrates advanced priority bases servicing. This confirms that the I/O subsystem acquires the specified bandwidth for peak performance.

2.2 Basic Input Output System

A boot ROM is required to boot the processor and then load the operating system in an Intel architecture system. BIOS: the basic input/output system is the boot ROM for the Intel architecture system. The functionality of the BIOS is to program the components and registers which further leads on to set the devices according to design of the hardware. When the system boots, the BIOS identifies and initializes the devices on the system including hard-disk drive, solid state drive, mouse, video display card, keyboard and various hardware followed by locating software held on a boot device. After the initial configuration is done and the execution of BIOS starts, it will find the amount and type of memory. After the hardware and devices are configured properly, the control goes to Operating system.

Mainly, the Intel architecture comes with the required boot firmware except for custom systems Intel provides Intel® Boot Loader Development Kit (Intel® BLDK), which may be used to create a UEFI-compliant boot loader which is compatible with different operating systems [1]. For minimum basic functionality boot-loader, Intel® Firmware Support Package (Intel® FSP) is provided. FSP includes most critical functions of Intel processor and chipsets.

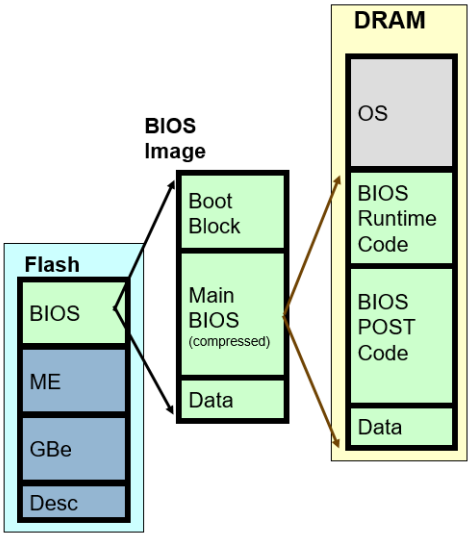

Figure 2.3: BIOS flashing mechanism

BIOS is taken as the most faithful computing base and hence it is very important[5]. The system BIOS is acquired in non-volatile memory called as boot firmware. The system BIOS also loads system management functions like thermal and power management. The BIOS is placed on electrically erasable programmable read-only memory (EEPROM) or other non-volatile storage. The BIOS are often modified by end users using a utility program. Frequent updates are provided for the system firmware to fix the bugs, for supporting new hardware and patch vulnerabilities.

2.3 Unified Extensible Firmware Interface

UEFI was developed for replacing the legacy BIOS to smooth the procedure for booting which behaves as interface for a operating system and its platform firmware. It has taken place for most of the BIOS capacities, yet in addition provides a good extensible pre-OS conditions with various new boot and run-time administrations. It is in Intel's underlying

Extensible Firmware Interface (EFI) determination 1.10. The design of UEFI prepares the customers to execute applications in a pre-defined interface. It has networking capacities and can work with multi core frameworks.

The interface is in the form of data tables which contains boot and run-time service calls, platform related information which are present with the OS loader. This information provides an environment for the booting of OS. UEFI specification need to be followed to provide an abstraction to OS by the firmware and the platform.

2.3.1 UEFI Design Overview

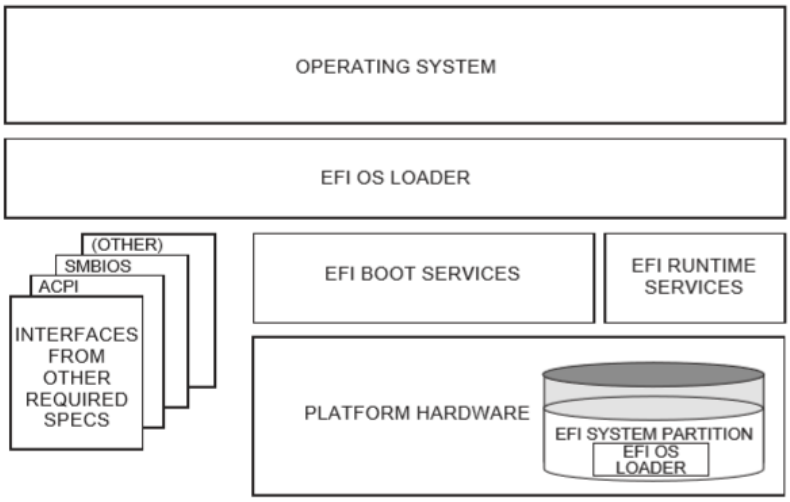

Figure 2.4: UEFI conceptual overview [2]

The UEFI design has different elements:

- **Reuse of existing table-based interfaces:**[2] To hold the services in present infrastructure code, available in OS and also firmware, some of present specs which are normally applied on platforms well matched with supported processor specifications must be applied on platforms to conform with the UEFI specification.
- **System Partition:**[2] The System partition defines a partition and record system which are designed for permitting the safe sharing among the multiple providers also for distinct functions.
- **Boot Services:**[2] It provides an interfaces for devices and system functionality

which can be utilized during the boot process. The Device access is hidden through "handles" and "protocols."

- **Run-time Services:**[2] A minimized set of run-time services are provided to make sure there is proper abstraction of base platform hardware sources that is needed by the OS throughout its normal functionality[2].

The figure 2.4 describes the interaction of various parts of UEFI specification and their interaction to platform hardware and OS software. The platform firmware retrieves the OS image from system partition. After starting, OS loader will still boot to complete operating system. For this, UEFI uses the interfaces and EFI boot services to initialize different platform components and OS software's. During the boot phase, EFI run-time services are also available.

Chapter 3

Design and Implementation

Intel Architecture system uses the EDK II build architecture to build the binary modules which boots to the OS. To generate the binary firmware images and UEFI applications EDK II build specification needs to be followed.

3.1 UEFI Firmware Images

UEFI specification defines the standard format for EFI firmware storage devices. The build systems should be such that it can process the files to create the file format specified by the UEFI specification. A "Firmware Volume" (FV) is a file level interface for firmware storage. In a single flash device, multiple FVs may be present. Firmware File System (FFS) is used as a file system. All the modules are stored as a file in FV. This module may be executed at a fixed address or may be relocated when loaded into memory. Some modules may run from memory while some from ROM if memory is not present. Each file has an internal binary format. This format helps in implementation of security, compression, signing etc.[3]

There are different layers of organization to a full UEFI firmware image. These layers are shown in figure 3.1. Each transition between layers implies a processing step that transforms or combines previously processed files into the next higher level [3]. The final outcome by the EDK build process is the creation of Binary modules. These binary modules can be distributed to different vendors as per requirements without disclosing the source code.

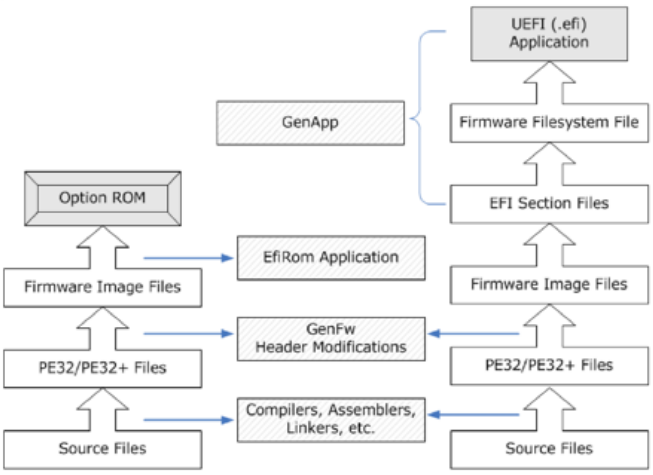

Figure 3.1: UEFI Firmware Image and UEFI application creation[3]

Additionally, this build process also support creation of stand-alone UEFI application. This application resides in the form of files placed onto the EFI System Partition that could be run from the UEFI command shell shown in figure3.2. UEFI applications could be developed and placed independently irrespective of the system manufacturer.

UEFI application are independent of the platform. It can be run on any platform. This applications are light-weight and can easily run on the shell.UEFI applications source code are written following the EDK II C Coding Standards Specifications[6].The UEFI application files (.efi files) built from application modules are put in the following directory: $(OUTPUT_DIRECTORY)/$(PLATFORM_NAME)/(BuildTarget)_(ToolChainTag)/$(ARCH) [3] The Sample UEFI application is shown in figure 3.3. It takes statement from the file and prints it on the shell. UEFI application are very well used for debugging.

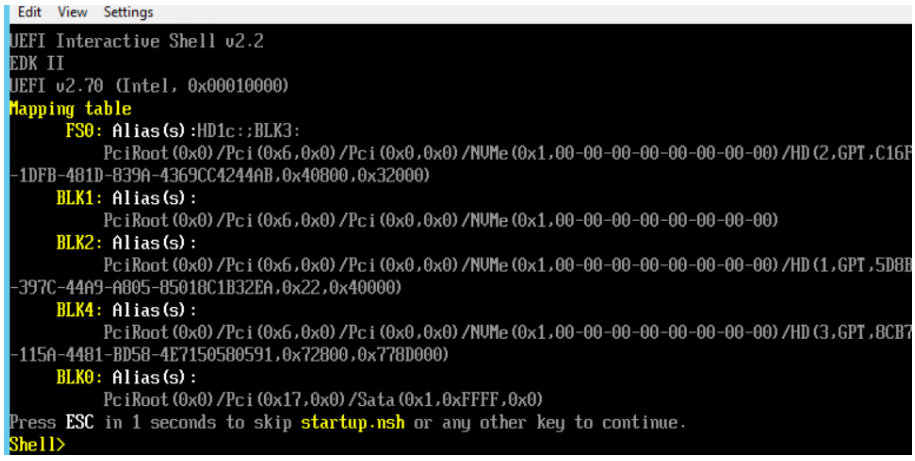

Figure 3.2: UEFI Shell

```
Shell> fs0:
FS0:\> first_appl.efi
FS0:\> a's First application
```

Figure 3.3: Sample UEFI application

3.2 Platform Initialization Boot Phases

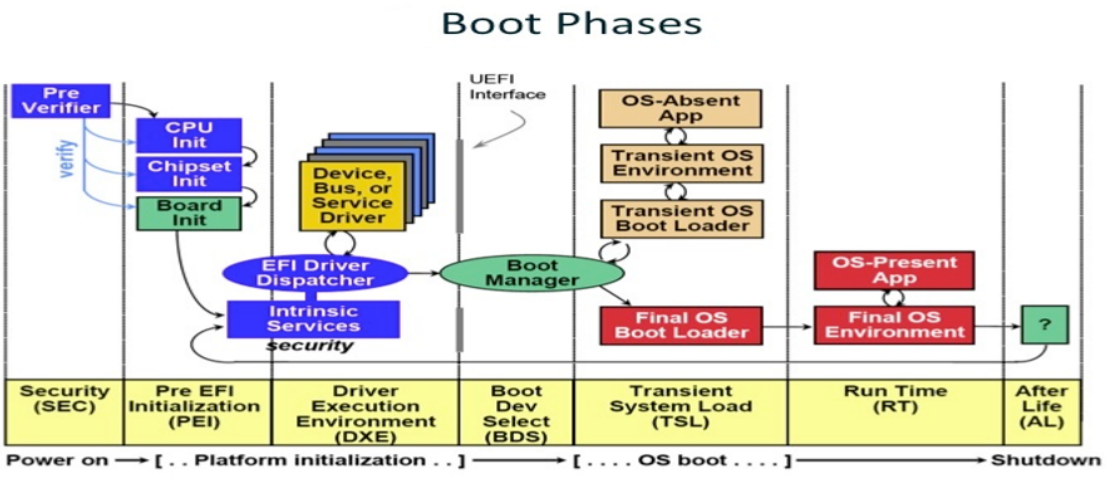

Figure 3.4: Platform Initialization Boot Phases

The system firmware should support this six phases: security (SEC), pre-EFI initialization (PEI), driver execution environment (DXE), boot device selection (BDS), run time (RT) services and After Life.

- Security Phase:

  The Security phase performs the following activities:

  - To handle all platform restart actions.
  - To create temporary memory storage.
  - To act like the root of trust.
  - To transfer hand-off information to PEI foundation.

- Pre-EFI Initialization (PEI):

  The PEI phase cannot assume the availability of amounts of memory (RAM). It operates on the on-processor resources, like the processor cache as a call stack. It performs following activities:

  - To locate and validate PEIMs
  - To dispatch PEIMs
  - To Initialize some permanent memory
  - To describe the firmware volume locations in Hand-off Blocks (HOBs).

- Drive Execution Environment (DXE):
  There are several components in the DXE phase:
  - Foundation of DXE
  - Dispatcher of DXE
  - DXE Drivers set

- BOOT Device Selection (BDS):
  The BDS phase performs the following:
  - Initializing the console devices
  - Loading the device drivers
  - Loading and executing the boot selections

- Transient System Load (TSL) and Runtime (RT):
  The Transient System Load (TSL) is mainly the OS vendor provided boot loader.

- After Life (AL):
  The After Life (AL) phase consists of UEFI drivers which are used to store the state of the system during the OS orderly shutdown, sleep, hibernate or restart processes.

3.3 Single Silicon Package Architecture

The IP based silicon support architecture allows us to reuse firmware code for IP across dies. The block diagram 3.6 shows the conceptual view of single silicon package. In the diagram IP is the logical unit of functionality in silicon. FRU (Field Replaceable Unit) is an Integration of IP into a chip and Product is an Integration of FRUs into a package. This architecture targets one silicon package to support multiple FRU with high IP leverage. This package is intended to support multiple concurrently active projects.

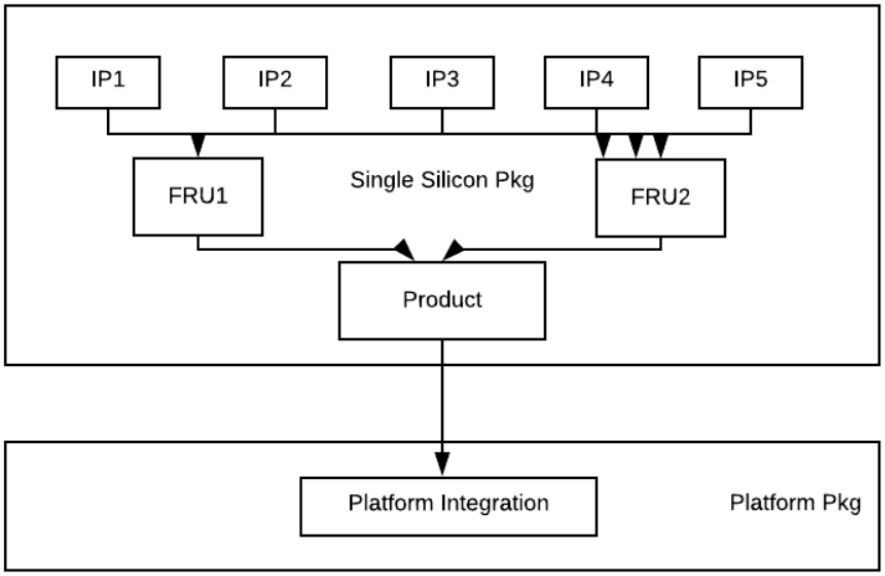

Figure 3.5: Single Silicon Package Architecture

Hence it is expected that code changes supporting different projects will be constantly made in parallel. The package layout is intended to facilitate parallel work.

3.3.1 Package structure

The package is organized with the top level directories - Include, IncludePrivate, FRU, Library, Product, and IpBlocks. All other top level directories represent shared code that does not fit within the context of a specific IP Block. IP Blocks are specific silicon features which are found in many generations of silicon and are expected to evolve and improve as silicon development progresses. Within the product folder will be a subfolder for each supported product. Code which is specific to a single product should be placed inside the respective product folder. Code which is useful for more than one product (aka "shared code") should be placed in the appropriate IP block folder. Shared code does not need to be universally used on all products, it only needs to be useful for more than one product. For libraries, drivers and PEIMs that are used by more than one platform, shared DSC files may be defined as appropriate. For example, a shared DSC can be defined that contains libraries, drivers, and PEIMs that are used by multiple platforms.

The IP block consists of individual IP initialization code that are modular and common across generations. Each IP Block directory consists of its own include files, libraries,

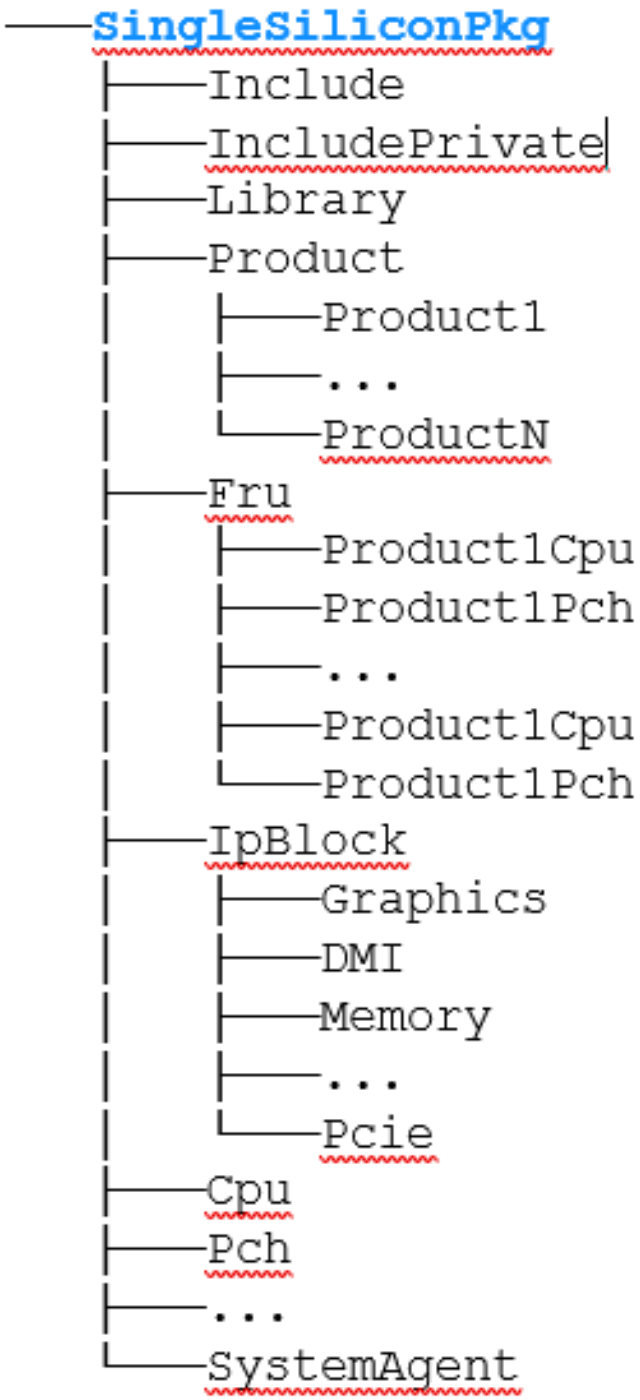

Figure 3.6: Package Structure

PEIMs, and DXE drivers. An IP Block must not contain any silicon policy structure definitions, or any public API definitions.

3.4 Register Debug Tool

For various IP's, different registers and their flow are programmed based on the specification defined by the architecture. There are various debugging methods used in the firmware development.

- Based on Hardware:

  JTAG(Joint Test Action Group) interface allows complete inspection of the hardware and software execution flow. This is used for pre silicon debugging when the stability of the system is not so good. But the JTAG interface is not available on the production hardware which limits its usage. Also, the cost of JTAG is not worth when only minor issue are to be fixed.

- Checkpoints:

  Checkpoints is a hexadecimal value sent to I/O port 0x80. These are used to point out the task in which phase PEI, DXE and BDS the BIOS is executing in the system.These are useful during the pre-boot phase to find out in which phase the system fails. But these checkpoints card cannot store and hence it should be manually noted.

- Debug Strings:

  Debug strings provides the detailed information of each driver entry points, the protocols, the strings added to identifies the faulty points. But on the production firmware, debug strings are disabled.

For debugging the flow of the system and to find the root cause in case of failure, developing the debug tool which will allow debugging the IP in case of when there is no other debugging options available(e.g Customer System) where there will be no JTAG/DCI interface present.

Register Debug Tool is an UEFI application which is platform independent and can be run on any phase of the system whether it is pre-boot or production. This tool will be more faster for debugging and will reduce the time cost. This tool will help in verifying the register programmed of various IP's like DMI and PCIe.

3.4.1 EDK II Build Process

For processing the EDK II meta-data files, binary files and some EDK libraries EDK II build system is used. The EDK II build system provides the UEFI Distribution Packaging Tool (UEFIPT) that can be used to create, install or remove UEFI distribution packages[3].EdkCompatibilityPkg provides the backward compatibility for EDK platforms and components.

Before building the source, system environment variables such as WORKSPACE, EDK_TOOLS_PATH are initialized. This is done by the EDK setup scripts. The Build process is divided into three stages:

1. Pre-Build: In this stage, meta-data files, encoded files and Make files are parsed.
2. Build: In this stage, source code files proceeds to generate PE32/PE32+ images which are further processed to generate EFI format files using MAKE(for UNIX style operating system development platforms) or NMAKE(for Microsoft style operating system platforms).
3. Post-Build: In this stage, collects the EFI files and binary to create EFI FLASH images, UEFI applications or EFI update Capsules.

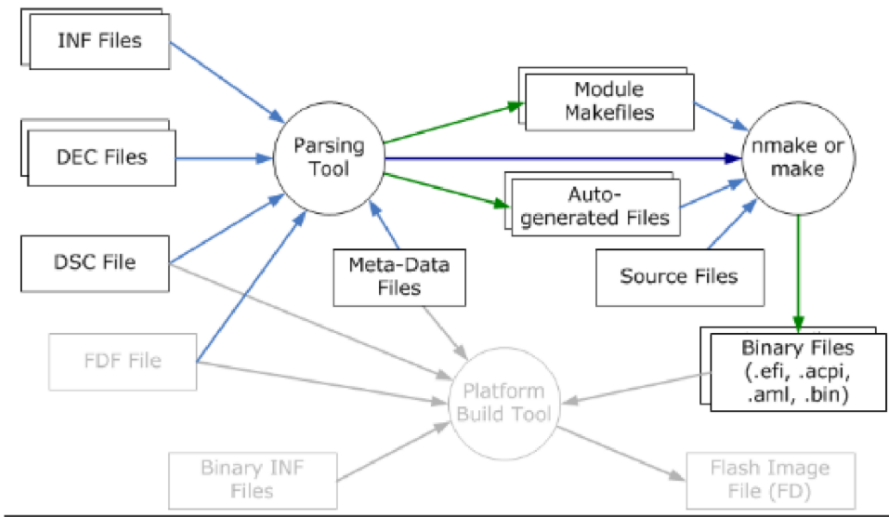

Figure 3.7: EDK II Build Process Flow [3]

3.4.2 Flow of the tool

UEFI application runs on the UEFI shell in the pre-boot environment.Below is the flow of the register debug tool:

1. Program the register bits to the register address based on the Firmware architecture specification defined for each IpBlock.

2. For specific IP, fetch the value programmed based on the policies and setup options to get the run-time values using the different UEFI protocols and API.

3. Build the module package based on EDKII build process to generate the Register-DebugTool.efi file.

4. Boot to UEFI shell and run the RegisterDebugTool.efi file from the shell as shown in figure 3.8. As UEFI application are platform independent this .efi file can be copied into other platforms and run from the shell.

5. The output obtained can be matched with the original values specified. Based on that the root cause of failing to boot because of certain IP would be identified.

Figure 3.8: Register Debug Tool run from shell

Boot manager loads the UEFI applications. For loading the UEFI application, firmware allocates memory to carry the image and copies the portion to the allocated memory. The control is then passed to the UEFI application's entry point. When the application returns the status EFI_SUCCESS or adds the EFI_BOOT_SERVICES.Exit(), the application is taken out from the memory.

The UEFI application's entry point is defined in EFI_IMAGE_ENTRY_POINT() as shown in figure3.9. The parameters passed are ImageHandle which allocates the firmware for UEFI image and SystemHandle is the pointer to the EFI System Table which provided pointer to console devices, Runtime Services Table and Boot Services Table.

```
Prototype
  typedef
  EFI_STATUS
  (EFIAPI *EFI_IMAGE_ENTRY_POINT) (
   IN EFI_HANDLE     ImageHandle,
   IN EFI_SYSTEM_TABLE  *SystemTable
   );
```

Figure 3.9: UEFI Entry Point [2]

The register values programmed are always 32 bit aligned. For each bit the programming is done based on the specification defined. Figure 3.10 shows the register values of overall DMI controller. Each line shows the 16 Byte of register values. This values are matched with the specification to find out the faults in programming done.

Figure 3.11 shows the list of boot critical register values which are fetched from the setup options.

Figure 3.12 shows the register values programmed for the PCIe controller.

```
86 80 36 9A 06 01 90 00-04 00 04 06 00 00 01 00
00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 20
00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
00 00 00 00 01 00 00 00-2F 00 11 00 81 48 72 00
43 00 81 38 00 00 00 00-00 00 00 00 00 00 00 00
00 00 00 00 37 08 08 00-00 00 00 00 02 00 00 00
01 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
05 90 00 00 00 00 00 00-00 00 00 00 00 00 00 00
00 A0 00 00 00 00 00 00-00 00 00 00 00 00 00 00
01 00 03 C8 00 00 00 00-00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
77 80 00 0E 42 18 00 00-08 20 12 00 00 00 00 00
00 77 10 00 00 00 00 00-04 80 02 00 00 80 00 00
50 07 00 00 00 03 00 40-00 0F 00 00 00 40 01 40
01 00 01 00 00 00 00 00-00 00 00 00 10 00 06 00
00 10 00 00 00 20 00 00-00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
03 00 00 00 01 00 00 00-00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
```

Figure 3.10: DMI Register value

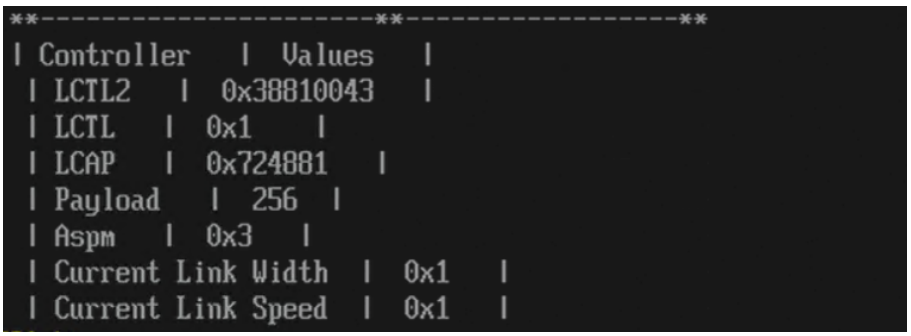

Figure 3.11: List of DMI critical register values

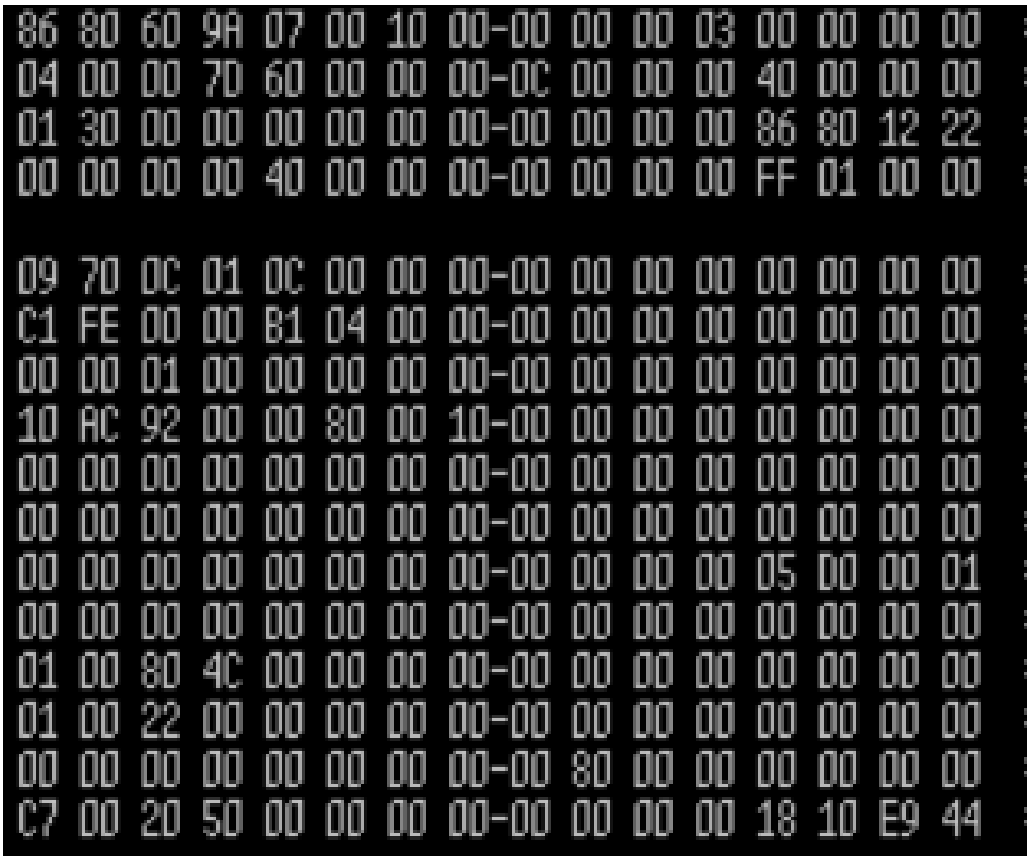

Figure 3.12: PCIe Register value

Chapter 4

Conclusion

Single Silicon Pkg will reduce development effort to work on new features and will make it easier to maintain multiple projects in parallel. Thus, this will provide effortless re-usability of code. Register Debug Tool debugs faster and is independent of platform which can even run on the production systems. This is very helpful in validation.

4.1 Future Scope

In the upcoming time-frame, will support in Intel's vision of IP model approach. The IP code should be plug and play where project architecture can decide which all IP's are supported and necessary to build project and using single switch or changes it should be able to serve the purpose. Developing debug tool for other IP's along with identifying the boot critical register offset and listing out during run-time to get the status of the IP mentioned. Also, will focus on doing more optimization where required along with unit testing.

Bibliography

[1] J. Turley, "Introduction to intel architecture," *Intel Corp., Santa Clara, CA, USA. White Paper*, 2014.

[2] "Unified extensible firrmware interface specifaction, version 2.8." https://uefi.org/sites/default/files/resources/UEFI_Spec_2_8_final.pdf, 2019.

[3] "Edk2 build specification." https://edk2-docs.gitbooks.io/edk-ii-build-specification/content/, 2019.

[4] D. L. Hill, D. Bachand, S. Bilgin, R. Greiner, P. Hammarlund, T. Huff, S. Kulick, and R. Safranek, "The uncore: A modular approach to feeding the high-performance cores.," *Intel Technology Journal*, vol. 14, no. 3, 2010.

[5] M. I. A. Butt, "Bios integrity an advanced persistent threat," in *2014 Conference on Information Assurance and Cyber Security (CIACS)*, pp. 47–50, IEEE, 2014.

[6] "Edk2 c coding standards specification." https://edk2-docs.gitbooks.io/edk-ii-c-coding-standards-specification/content/, 2019.