IMPLEMENTATION OF MODULAR TESTING IN BIOS DEVELOPMENT AND DEBUG

By

TEH HONG HOR

A Dissertation submitted for partial fulfillment of the requirement for the degree of Master of Science

July 2014

Acknowledgement

First of all, I would like to express my deepest gratitude to my supervisor, Dr Zaini for her enthusiastic supervision, her insight and her constant support for me in this Master Dissertation. Her guidance helped me in conducting the project and thesis writing.

Besides that, I would like thank to my sponsor, Talent Corp who supports me financially throughout the master degree program. I would like to extend my appreciation to person in charge of Talent Corp, Miss Lyne who deals with multiple parties for anything related to my master degree program.

Next, I would like to express thank Intel Microelectronic Malaysia who provides me the facility to conduct this project. Mr. Lee Kim Wah, my great colleague who is always willing to mentor me in BIOS knowledge, also share me valuable advices for me in conducting this project.

Last but not least, I would like to thank my parents for their unconditional support to me throughout the master degree study.

TABLES OF CONTENTS

List of Tables

Page

List of Figures

Page

List of Abbreviations

Abbreviation	Meaning
VLSI	Very Large Scale Integration
SOC	System-On-Chip
IP	Intellectual Property
TTM	Time to Market
BIOS	Basic Input Output System
UEFI	Unified Extensible Firmware Interface
LPSS	Low Power Sub-System
UAV	Unmanned Aerial Vehicle
PCI	Peripheral Component Interconnect
FPGA	Field Programmable Gate Array
API	Application Programming Interface
TAM	Test Access Mechanisms
ATPG	Automatic Test Pattern Generation
POST	Power-On Self-Test
HDD	Hard Disk Drive
SATA	Serial ATA
USB	Universal Serial Bus
GUID	Globally Unique Identifier
CAD	Computer-aided design
INF	Information File
DSC	Description File
DEC	Declaration File
EDK	Embedded Development Kit

IMPLEMENTASI BAGI UJIAN BERMODUL DALAM PEMBANGUNAN BIOS DAN DEBUG

ABSTRAK

Pojek ini membentangkan pendekatan bermodular dalam pembangunan BIOS dengan tujuan untuk menangani masalah masa pembangunan yang panjang melalui kaedah yang sedia ada iaitu pendekatan platform perkakasan dan pendekatan platform maya. Pendekatan yang dicadangkan ini terdiri daripada platform generasi terdahulu, kad FPGA and pemacu UEFI. FPGA dimuatkan dengan RTL bagi subsistem kuasa rendah (LPSS). Kemudian, kad FPGA dipasang ke dalam slot PCI yang berada di platform. Di samping itu, pemacu UEFI Konfigurasi and pemacu UEFI Reset dibina untuk mengkonfigurasi dan reset daftar LPSS. Kedua-dua pemacu UEFI ini disimpan ke dalam USB dan dipasang ke dalam USB port bagi platform tersebut. Ia akan dilaksanakan dalam persekitaran UEFI Shell. Dalam projek ini, masa pembangunan LPSS yang diperlukan oleh tiga kaedah telah dibandingkan. Keputusan menunjukkan bahawa cara bermodul mampu menjimatkan sehingga 90% masa pembangunan kalau dibandingkan dengan dua cara yang lain. Kedua-dua pemacu UEFI juga berfungsi dengan betul. Masa pemprosesan kedua-dua pemacu UEFI Konfigurasi and pemacu UEFI Reset adalah 1 hingga 2 saat sahaja. Kesimpulannya, BIOS boleh dibangunkan dengan cara bermodul, tanpa perlu membangunkannya secara keseluruhan. Oleh itu, ia dapat mengurangkan masa pembangunan BIOS dengan cekap.

IMPLEMENTATION OF MODULAR TESTING IN BIOS DEVELOPMENT AND DEBUG

ABSTRACT

This project presents a modular approach in BIOS development in purpose to tackle the problem of long development time with the existing methodologies which are hardware platform approach and virtual platform approach. The proposed approach consists of previous generation platform, a FPGA card and UEFI drivers. The FPGA is loaded with the RTL of one Intellectual Property (IP) from the current company project. The chosen IP is Low Power Subsystem (LPSS). The card is then plugged into the PCI slot of the platform. Besides, UEFI Configuration Driver and UEFI Reset Driver are built to configure and reset the LPSS registers respectively. Both of them are stored into a thumb drive and plugged into USB port of the platform. They are executed in the UEFI Shell environment. In this project, the development time of LPSS needed by the three methodologies which are hardware platform approach, virtual platform approach and modular approach are compared. The results indicate that modular approach is capable to save up to 90% of the development time in comparison with the other two approaches. At the same time, both of the UEFI drivers are functioning correctly. The processing time of both of the UEFI Configuration Driver and UEFI Reset Driver are about 1 to 2 seconds only. In conclusion, the novelty of the modular approach is that the BIOS can be developed in modular basis, without having to develop the BIOS as a whole. Therefore, it is able to cut down the BIOS development time efficiently.

Chapter 1

Introduction

1.1 The reason of the implementation of modular approach

Modern semiconductor process technologies have the capability to manufacture a complete system into single die, which is called system-on-chip or SOC. These chips consist of millions of transistors and various hardware modules. Therefore, design and verification techniques [1] face more challenges with the higher complexity of SOC. The increasing complexity of SOC also increases the complexity of BIOS code as there are more configurations needed for various modules. Due to the increasing complexity of SOC, it also significantly bring up the issue of long BIOS development time with the existing two BIOS development methodologies which are hardware platform approach and virtual platform approach.

Modular concept can be implemented to solve the mentioned issue. Modularization in SOC means that SOC is partitioned into smaller blocks whereby engineers are able to make design, validation and development on targeting on single IP or module. This concept is also known as ‘divide-and-conquer’ style. The modular concept is not only applied in SOC, it is also implemented across different area of industries such as digital circuit [2], factory automation system [3], unmanned aerial vehicle autopilot systems [4] and automobile driveline [5].

The advantages of modularization concept are almost similar across different area of industries. One of the advantages of modularization is test reuse [6, 7]. For instance, the test is developed once for a target module and it can be reused again for it in different system chip. Thus, it optimizes the usage of engineering resource. Besides,

it also reduces test development time [7, 8] as the engineers are able to design, develop or validate the target IP or module as a stand-alone unit without having to test it together with whole complete system.

Modular concept can be materialized with the prototyping technique using FPGA especially in SOC. Prototyping SOC using FPGAs is an effective method for verifying and validating hardware designs [9]. Prototyping means that a portion of SOC design or entire design can be mapped onto the FPGA in order to gain performance acceleration [6] or to complete the hardware platform to enable software development [10]. In other words, one can design, develop or validate the portion as a stand-alone unit via FPGA prototyping.

In this project, the proposed modular approach implements prototyping method in BIOS development in order to materialize the concept of modularization. A particular IP is chosen to be mapped into a FPGA card. Then FPGA card is then plugged onto the previous generation platform. Besides, BIOS engineers have to develop UEFI driver which is targeting on the configuration of the chosen IP solely. In other words, BIOS engineers could now develop the BIOS code for a particular IP without having to develop the BIOS as a whole. The main contribution of this modular approach is to cut down the BIOS development time by allowing engineers to develop the BIOS code in modular basis.

1.2 Problem Statement on BIOS development & debug

The reason of the introduction of modular approach into BIOS development is mainly due to long BIOS development time with the use of the existing conventional methodologies which are hardware platform approach and virtual platform (Simics) approach.

Hardware platform approach has three steps of BIOS development process as shown in Figure 1.1. When the BIOS coding is ready, one can build the BIOS with the some scripting and tools in a computer. Then, the BIOS coding which are written in C language and assembly language will be converted to a binary file or BIN file. The next step is to store the BIOS BIN file into the flash memory of the platform. Once the platform is powered up, the BIOS will start booting. Then, BIOS engineers begin to verify their code by using debugging tools.

The BIOS development process with virtual platform approach is shown in Figure 1.2. Its only difference with the procedures of hardware platform approach is that it does not have the step of BIOS flash. Simics will look for the BIOS binary file through a script where the path of the binary file is written there. Theoretically, this approach will have shorter BIOS development time if compared with hardware platform approach. However, engineers have to always check the update for the scripts and virtual platform. Sometimes, it has some software problem too which causes time consuming issue. Besides, it is believed that it requires huge amount of money and workforce to develop this complex virtualized software system.

The common disadvantages of hardware platform approach and virtual platform approach is the long development time. This problem arises when one amends BIOS code of one IP or module. He has to undergo BIOS build process again to build a new version of BIOS which consists of the amended coding of the target IP and together with the unchanged coding of other irrelevant IPs as well. Therefore, the coding of irrelevant IP also contributes to the BIOS build time. This is very time consuming if one repeats the development cycle for many times.

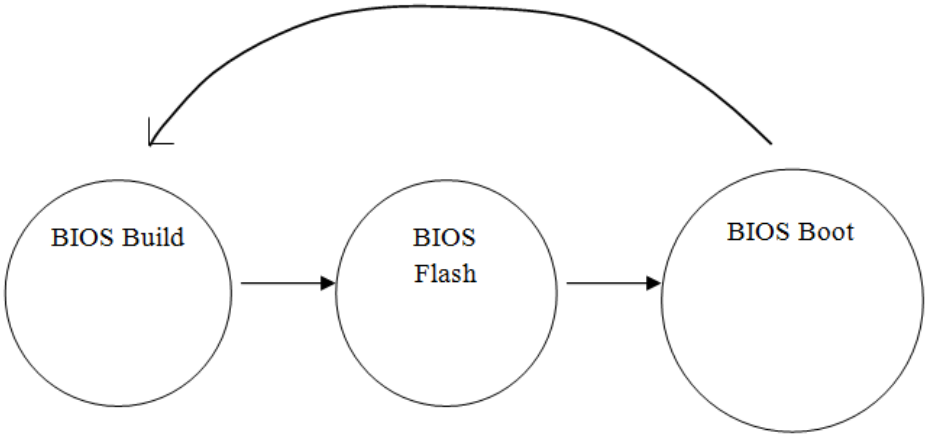

Figure 1.1 Procedures of BIOS development and debug using hardware platform.

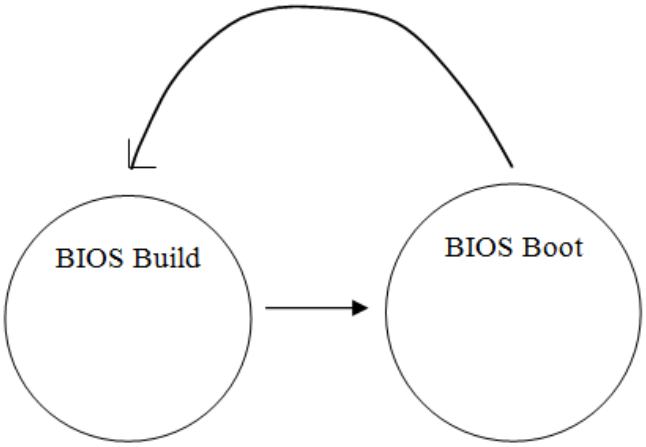

Figure 1.2 Procedures of BIOS development and debug using Simics.

Another common problem during BIOS development is the code dependency problem. BIOS is a complex piece of codes, it can be said as the integration of many modules such as library components, IP module drivers and etc. Since BIOS is coded in assembly and C language, it runs sequentially. For instance, there are two BIOS modules A and B, at which B will be executed after module. If a problem arises for module A, then it causes a hang issue and not able to execute module B. Then, BIOS

engineer who handles module B will have to talk to the BIOS engineer who handles module A to solve the issue. Or the BIOS engineer has to do some bypassing effort on module A so that he can test out his own module. From this scenario, it can be seen that this dependency issue creates some hassle which slows down the delivery efficiency. Therefore, it would be better off that there is new implementation for BIOS engineers to develop their parts independently.

There is a problem of insufficiency of silicon and hardware platform during post-Si. Therefore, engineers will have to share the usage of the platform and this will definitely slow down the debugging progress for the BIOS. Besides, it is not cheap to provide each of the engineers with one platform. Some more the platforms are no longer used after the project has been completed. Hence, it would be great that the past generation platform can be reused for the development of the current new projects.

In summary, a modular approach is proposed to tackle the problem of long BIOS development time with the current methodologies. Besides, the proposed modular approach aims to solve the dependency issue and the problem of inadequacy of silicon and hardware platform as it reuses previous generation platform.

1.3 Objective

The objectives of this project are to:

- Cut down the BIOS development time by allowing engineers to develop BIOS modularly.
- UEFI Configuration Driver and UEFI Reset Driver are expected to have correct functionality.
- The processing time of the UEFI Configuration Driver for IP registers configuration is expected to be comparable with the processing time required by hardware platform approach and virtual platform approach.

1.4 Scope of the project

Modular approach is introduced in the BIOS development in this project. The performance of this new approach will be compared with two existing methodologies which are hardware platform approach and virtual platform approach in terms of their BIOS development time for one IP module. The chosen IP of this project is low power subsystem (LPSS) which consists of 15 host controllers which are 8 I2C, 4 UART and 3SPI.

Xilinx Virtex-7 FPGA card is used in order to materialize the modular concept in BIOS development. It is loaded with the RTL of the new project LPSS. Then, it is plugged into the PCI slot of the previous generation board. Besides, UEFI Configuration Driver is built in order to configure the registers of LPSS, whereas UEFI Reset Driver is built to clear any possible configuration made by the UEFI Configuration Driver. Both of the drivers are written in C language. They are stored

into a thumb drive which is connected to the USB port on the previous generation platform. The drivers are loaded in the UEFI Shell environment.

The functionality and the processing time of both of the UEFI drivers are observed by capturing the time before and after the execution of the driver. Apart from this, the processing time of the UEFI Configuration Driver is compared with the processing time of the hardware platform approach and virtual platform approach for LPSS configuration.

1.5 Thesis Overview

The thesis is organized as follows.

Chapter 2 reviews the implementation of modular approach in various fields. BIOS and FPGA are briefed. Details on UEFI driver and PCI Express system are covered.

Chapter 3 mentions about of the proposal. The high level view of the proposed modular approach is presented. The BIOS development flow after the implementation of modular method is studied following by the programming flow for both of the UEFI drivers. The debugging tools and design parameters are mentioned as well.

Chapter 4 presents and evaluates the results for each of the design parameters.

Chapter 5 concludes the achievement of the objectives and summarizes the results obtained.

Chapter 2

Literature review

2.1 Introduction

This chapter covers variety of information. The existing methodologies for BIOS development are studied. Then, the past researches on the effort to cut down the BIOS development time will be reviewed as well. Besides, the details of FPGA-based prototyping, UEFI driver and PCI express system are covered as it is important to for the understanding on the proposed modular approach. BIOS and UEFI framework is discussed in order to get the knowledge of what BIOS is really doing. Moreover, the implementation of the modular concept across different area of industries is studied in order to expose the advantages of modularity.

2.2 Existing Methodologies

2.2.1 Hardware platform approach

Hardware platform approach is a common method which BIOS engineers use it during post-Si phase. It's the phase when there is a need for further development of BIOS and debugging purpose. Whenever there is a new project, there must be a new platform for the new project. The platform is no longer useful right after the project cycle is over.

There are certain repeated procedures for BIOS development and debug with the use of hardware platform approach. The procedures are illustrated in Figure 2.1. BIOS is a complex piece of code which consists of many modules at which each of them have specified role. The source files in C and assembly are built with the help of script and tools before they are converted to become a BIOS binary file. The binary file is then stored into the flash memory of the platform. Then, BIOS will start to run when the platform is powered up.

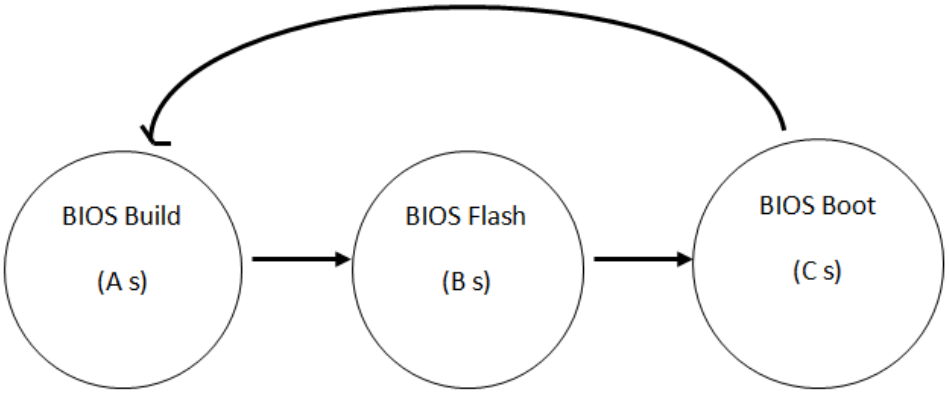

Figure 2.1 Time taken for each of the procedures of BIOS development using hardware platform.

The disadvantage of this approach is the long development time issue. This is mainly due to the overall procedures are required to repeat when there is only minor changes on the BIOS source files. For example, one would like to modify the BIOS code for LPSS; however, he has to build the BIOS again together with other irrelevant IP as well. BIOS build takes most of the time among the three steps of the BIOS development cycle of the hardware platform approach.

The time needed by the BIOS build process varies with the size of the BIOS source files. Besides that, there are other factors which affect the BIOS build time which are the speed of the processor or computer and also the number of processes running in the computer. The BIOS boot time is also subjective to the algorithm or program of the BIOS used.

‘A’, ‘B’ and ‘C’ represent the time taken in seconds for each of the procedures and ‘X’ is the number of repeating BIOS development cycle. A mathematical formula can be deduced for BIOS development time with the use of hardware platform approach, which is:

Time in seconds = (A + B + C)*X ---------- (2.1)

In summary, the time taken for BIOS development with hardware platform approach is increasing proportionaly with the number of repeating cycle.

2.2.2 Virtual platform approach

Virtual platform approach or Simics is another conventional method which is utilized during pre-Si phase for BIOS development and debug. It is a pure software approach for BIOS development. The advantage of this approach is the BIOS development can be done before the arrival of new project silicon and platform. The BIOS engineers can develop BIOS code as if they are having a real platform. The procedures of the BIOS development with the virtual platform approach are illustrated in Figure 2.2.

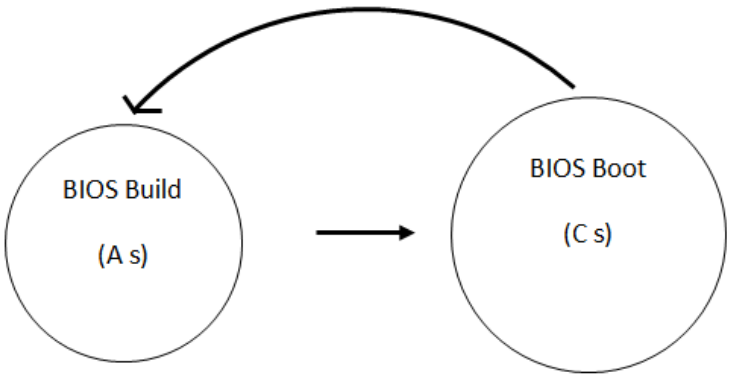

Figure 2.2 Time taken for each of the procedures of BIOS development using virtual platform.

Theoretically, the virtual platform approach will have shorter BIOS development time if compared with hardware platform approach as the effort of storing BIOS into

the flash memory is not necessary in virtual platform. There will be a script from the virtual platform to look for the BIOS binary and execute it.

The disadvantages of the virtual platform is that sometimes engineers have to check the script and virtual platform versioning in often. Sometimes, software issue from the virtual platform may need some time to solve. Therefore, it is time consuming. Other than that, it has the same disadvantage as hardware platform approach as well. If there is a small amendment on the BIOS source file, the BIOS will have to be built together with other irrelevant or unchanged IP. The factors affecting the time taken for BIOS build is also similar with hardware platform approach.

‘A’ and ‘C’ represent the time taken in seconds for each of the procedures and ‘X’ is the number of repeating BIOS development cycle. A mathematical formula can be deduced for BIOS development time with the use of virtual platform approach, which is:

Time in seconds = (A + C)*X ---------- (2.2)

In summary, the time taken for BIOS development with virtual platform approach is increasing proportionaly with the number of repeating cycle, however, it is theretically less time consuming as hardware platform approach since it has one step lesser.

2.3 Past researches on the effort to shorter BIOS development time

2.3.1 Updating BIOS in a flexible BIOS storage media

Jerry Jex from Intel Corporation [11] presents the advantages of updating BIOS revisions on flash memory in comparison with other types of storage media such as ROM (Read Only Memory), EPROM (UV Erasable PROM), and EEPROM (electrically erasable PROM).

One of the storage media for BIOS is EPROM/ROM. It has high density so that more information can be stored. Besides, ROM is not power consuming. However, the BIOS stored in EPROM or ROM requires chip replacement. The memory will have to be removed, and then the BIOS EPROM/ROM is inserted in the computer. The total computer down time could be over two hours. Therefore, this method is inflexible and time consuming for BIOS development.

Another storage media for BIOS is EEPROM (electrically erasable PROM). This approach requires the erasing and programming of full sections of BIOS code at the same time. The advantage of the updateability of EEPROM eases the user to eliminate the need to replace chip from time to time. However, it’s not flexible enough for the user to choose the BIOS parts he would like to keep and modify. Besides, EEPROM has low density and it is more costly than flash memory.

Then, a better alternative over the mentioned storage media is to store BIOS in bulk erasable flash memory. Flash memory has the density of EPROM, low power of ROM, and updateability of EEPROM. The computer down time could be from seven seconds to ten minutes. This shows that it has greatly reduced the development time in comparison with EPROM/ROM. The 8K boot code or recovery code as shown in Figure 2.3 is important to make sure the computer start-up is complete so that the

computers will at least be able to read a disk or receive keyboard commands, initialize a small amount of RAM, a floppy disk, and a chip set that supports the floppy disk and display prompt on the screen.

This 8K is normally interspersed throughout the 64K in BIOS location. Therefore, problem occurs when the power goes down during the BIOS upgrade on the flash memory. Recovery code might be corrupted and it causes computer to be unworkable.

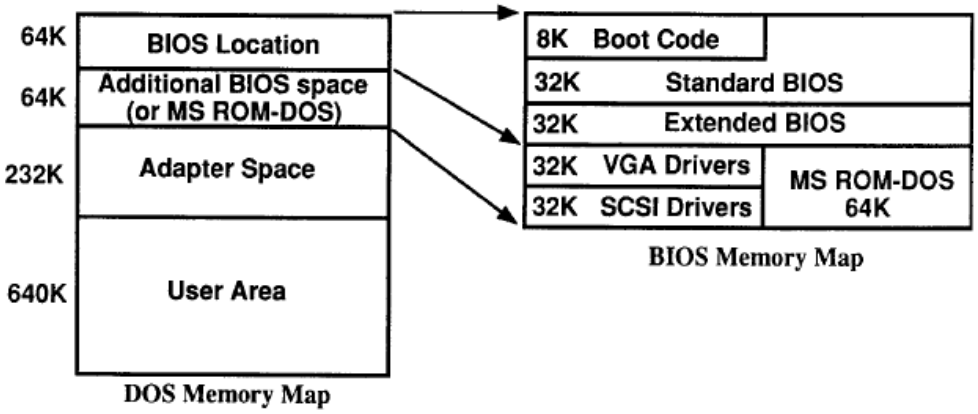

Figure 2.3 1MByte Dos Memory Map [11].

In order to cope the mentioned issue, block erasable flash memory is introduced to prevent the corruption of the recovery code. Blocked flash memry has a segmented array and hence each block can be erased seperately. It allows the recovery code to remain unchanged during the BIOS upgrade.

In summary, flash memory allows BIOS to get updated rapidly. With the use of flash memory, it is confident that BIOS development cost and time can be reduced significantly.

2.3.2 USB-BIOS Emulator

USB-BIOS emulator [12] is designed to help to reduce the BIOS development and debug time for BIOS engineers. When BIOS engineers are developing BIOS, they always have to check the data like PCI Configuration Space data, memory data, and the register data. Checking data is important for BIOS engineers to verify whether the BIOS code is functioning correctly or not, and also to examine whether the hardware responds as expected to the BIOS code or not.

There is a USB-BIOS emulation card is plugged onto the PCI slot of the hardware platform as shown in Figure 2.4. This card has a 256-bytes built-in FIFO buffer where the data will be held in. Then these data can be captured by the BIOS engineer to their computers via USB port.

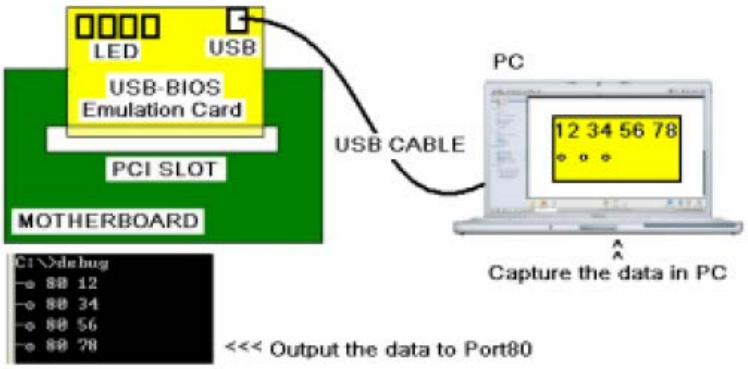

Figure 2.4 Hardware setup of USB-BIOS Emulator [12].

In summary, the USB-BIOS emulator enables BIOS engineers to retrieve the intended data efficiently and quickly, hence, the BIOS development time can be reduced. However, Intel BIOS engineers have in-house debugger software and ITP (In-Target Probe) to retrieve the data such as PCI Configuration Space data, memory data, and the register data.

2.4 FPGA-based Prototyping

FPGAs (Field-Programmable Gate Arrays) were invented about 30 years ago. Their influence in engineering is growing rapidly due to the fact that FPGAs have inherent configurability and relatively cheap development cost [13]. FPGAs were firstly proposed to be used for implementing simple glue logic [14]. They can now be constructed as a complex digital devices which are used in broad range of applications such as aerospace and defense, consumer electronics, medical, security, video and image processing, wired and wireless communications and so on.

As SOCs are becoming extensively used nowadays, the usage of FPGAs are getting ubiquitous in the development of embedded systems in order to achieve shorter time-to-market and update capabilities after deployment are important[15]. There is a SOC development and verification method with the use of FPGA which is called FPGA-based prototyping. It is the approach to prototype SOC on FPGA for hardware verification and pre-Si validation of the software development [14]. In another word, this means that a soft IP core can be embedded inside an FPGA for SOC development and verification. Derek Chiou and his team [6] implemented FPGA-based prototpying concept into their fast, full system, and cycle-accurate simulators design. They presents FPGA-Accelerated Simulation Technologies (FAST) by mapping a modern processor into the FPGA which is able to achieve average simulation speed and it is expected to improve it over time.

There are a number of advantages of FPGA prototyping which are the early detection of software bugs, portability, low-cost replication, uses same RTL as SOC, the benefits of customization, design reuse, and shortened design cycle [14, 16]. Thus, it will improve overall development efficiency.

In this project, modular approach implements the concept of FPGA prototyping by loading the RTL of LPSS into the FPGA. Then, BIOS engineers are believed to be able to develop the BIOS code targeting on the particular IP only.

2.5 UEFI Driver

There are two UEFI drivers are built for this project which are UEFI Configuration Driver and UEFI Reset Driver. The importance of this section is to know about the build process flow of the UEFI drivers and also the required ingredients for UEFI driver build.

UEFI stands for Unified Extensible Firmware Interface. UEFI driver is a modular firmware code that supports chipset or platform features and is reusable in multiple system contexts [17] The development environment for creating an UEFI driver includes Windows, Linux and OS/X development workstations. Development workstations must be running an IA32 or X64 operating system. The UEFI driver can be executed or loaded in UEFI Shell environment. UEFI Shell is a simple, interactive “command interpreter” that allows EFI device drivers to be loaded, EFI applications to be launched, and operating systems to be selected for boot [17]. The build process of UEFI driver will be studied in section 2.5.1. In the following section, part of the recipes for driver build will be discussed.

2.5.1 Build Process Flow of UEFI Driver

In fact, there are several layers of organization to build an UEFI driver which are pre-build stage following by build stage. Post stage will be applied if the driver is intended to be continually processed as a part of firmware image, UEFI application or BIOS.

The ingredients to build EFI driver are C source files, header files, INF and DSC file. There are several tools to convert them to become a driver such as parsing tools and NMAKE. EFI driver is built via EDK II Build Process which is as shown in Figure 2.5. There are three stages to the build process:

1. Pre-Build or AutoGen stage: INF files, DEC files, DSC file and Meta-Data files are parsed by parsing tool in order to generate some makefiles.
2. Build or $(MAKE) stage: Source code files together with makefiles are processed to create PE32/PE32+/COFF images that eventually are processed to EFI format using NMAKE(Microsoft operating system development platforms) or MAKE(UNIX style operating system development platforms).
3. Post build or ImageGen stage: Binary (.bin) and EFI format files (.efi) are further processed to create FLASH image, UEFI applications or PCI Option ROMs. In this project, EFI format file is not further processed as it is already enough to play its role as driver to configure the registers of LPSS in FPGA card.

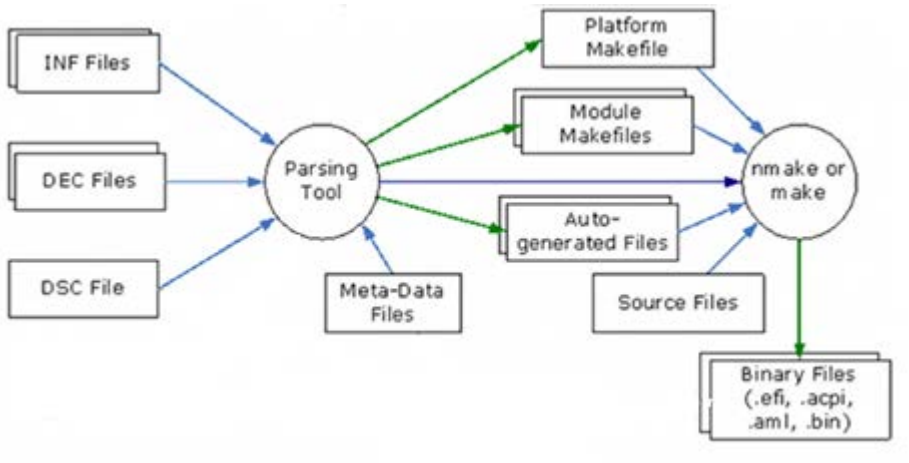

Figure 2.5 Build Process of EDK II Platform.

2.5.2 Role of INF and DSC file in building process of UEFI driver

DSC stands for platform description file. This file describes what and how modules, libraries and components are to be built, as well as defining library instances which will be used when linking EDK II modules. The path of the driver INF is included in DSC.

INF stands for information file [18]. As its name implies, this file describes how the module is coded in EDK specification as well as providing some basic build information. INF provides information such as driver name, GUID and driver type which are described under [Defines] category as shown in Figure 2.6. The main recipes of the driver which are source files (.c) and header files (.h) are defined under category [Source]. Besides, the necessary libraries are imported under [LibraryClasses] category; whereas required protocols are stated in [Protocols] section.

```
[Defines]
  INF_VERSION                    = 0x00010016
  BASE_NAME                      = DiskIoDxe
  MODULE_UNI_FILE                = DiskIoDxe.uni
  FILE_GUID                      = 6B38F7B4-AD98-40e9-9093-ACA2B5A253C4
  MODULE_TYPE                    = UEFI_DRIVER
  VERSION_STRING                 = 1.0
  ENTRY_POINT                    = InitializeDiskIo


  [Sources]
    ComponentName.c
    DiskIo.h
    DiskIo.c


  [Packages]
    MdePkg/MdePkg.dec


  [LibraryClasses]
    UefiBootServicesTableLib
    MemoryAllocationLib
    BaseMemoryLib
    BaseLib
    UefiLib
    UefiDriverEntryPoint
    DebugLib


  [Protocols]
    gEfiDiskIoProtocolGuid                ## BY_START
    gEfiBlockIoProtocolGuid               ## TO_START
```

Figure 2.6 UEFI driver declaration in INF file. [18].

2.6 PCI Express System

The understanding on PCI Express system is extremely important to comprehend the algorithm of the UEFI drivers to search the FPGA card which is plugged on the PCI slot of the platform before they start to execute their main function such as registers configuration and registers reset. As mentioned in section 1.3, the FPGA card is programmed with the RTL of LPSS. LPSS has 15 host controllers which are 8 I2C, 4 UART and 3 SPI. FPGA card is connected to an Express PCI bridge as shown in Figure 2.15 and the 15 host controllers will be enumerated as PCI devices.

In PCI Express System, it is common to mention about bus, device and function number (BDF number). It serves as a unique number for each of the PCI devices on the system. Bus can be imagined as many paths/roads; whereas device can be treated as a house on the road; function is the rooms of the house. In fact, device resides on a bus. It has one or more than one function; maximum is eight depending on the device is multifunctional device or single function. The FPGA card and 15 host controllers will have their own unique BDF number.

When the system is powered up, the configuration software only knows the existence of the host/PCI bridge within the root complex only. Then it will scan the PCI Express fabric in order to discover the topology. This process is referred as PCI Enumeration process [19]. An example of the topology of the system is shown in Figure 2.7. It can be observed that device on buses 4, 7, and 10 are single-function device; whereas device on bus 3 is multi-functional device.

Each of the functions will have configuration space at size of 4KB or 0x1000. The PCI configuration space composes of 3 sections such as PCI Header (16 Double Words), PCI-Compatible Space (48 Double Words) and PCI Express Extended

Configuration Space (960 Double Words) which are shown in Figure 2.8. FPGA card and 15 host controllers will have their own PCI configuration space. This PCI configuration space allows BIOS to change properties of the PCI device through memory cycle instead of I/O cycle. In this project, UEFI drivers perform some of the registers configuration of the LPSS host controllers in this PCI configuration space.

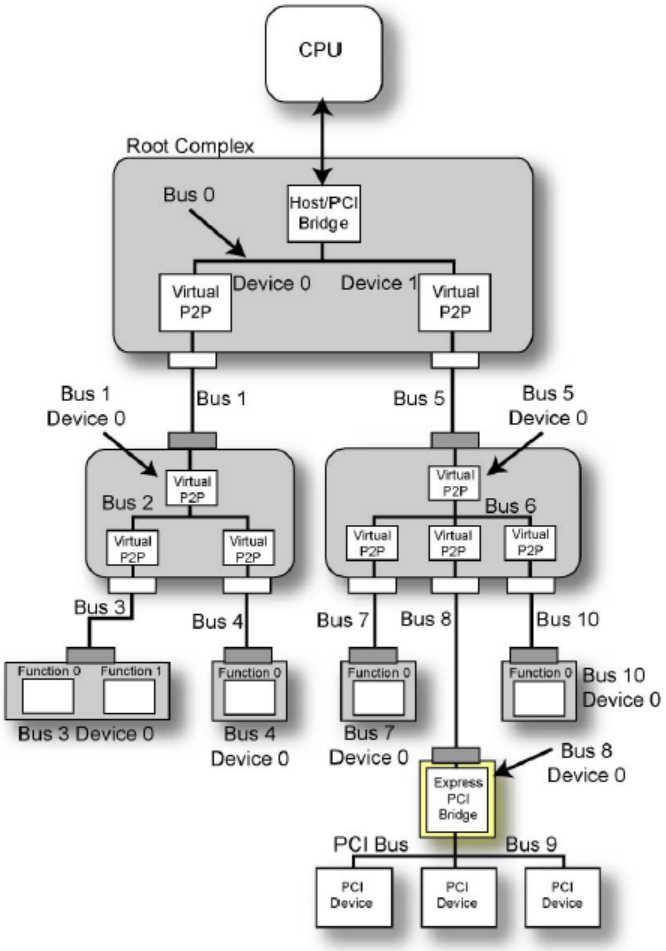

Figure 2.7 Topology of the PCI Express System [19].

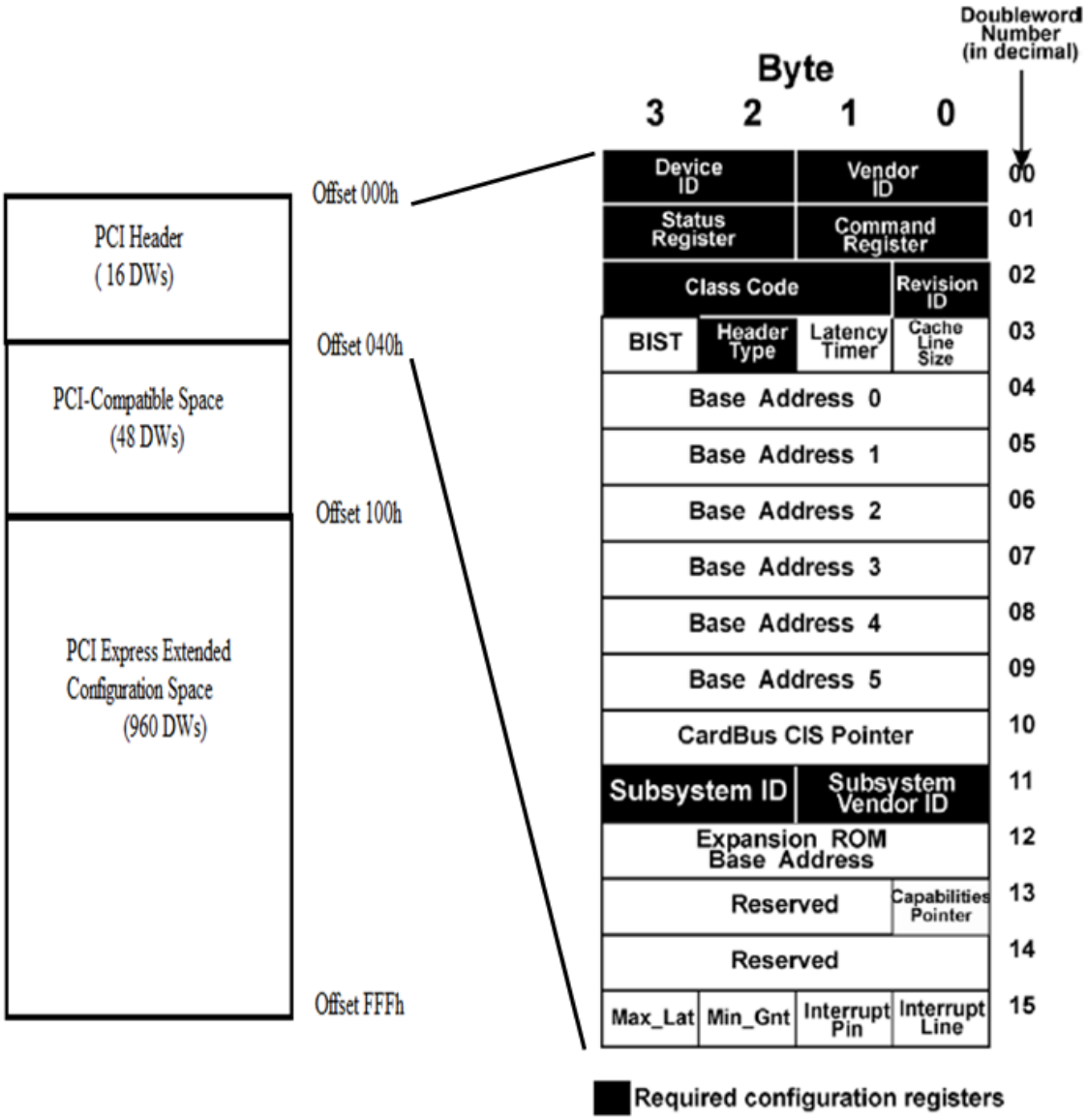

Figure 2.8 PCI Configuration Space and PCI header [19].

There are several steps in PCI Express enumeration. The brief flow of the enumeration manner is explained as below [19]:

a. Firstly, enumeration will start at bus 0, device 0. If FFFFh vendor ID is returned, the software will continue the enumeration process on the next device on the current bus.
b. If a valid vendor ID is obtained, the Header Type and capability registers of the device will be checked.
c. Software will write/update the bridge's bus number registers (primary, secondary, and subordinate).
d. Software will update the bridge's (or all the bridges which have been identified on the similar branch) subordinate number.
e. Before moving to next device on the same bus, software performs depth-first search. It will check for vendor ID of [(bus = secondary), (device = 0), (function = 0)]. If it is valid, then the steps from c to e are repeated.
f. If Header Type is an end port, software will stop performing depth-first search for that particular branch. Then new branch will be started for depth-first search.

2.7 BIOS and UEFI framework

BIOS stands for basic input/output system. It is built-in software which allows CPU to communicate with other devices. It consists of a number of functions such as control of keyboard, display screen, disk drives, and serial communications and so on. BIOS is a set of instructions to the CPU, they are stored in ROM (Read Only Memory), EPROM (UV Erasable PROM), bulk erasable flash memory, or block erasable flash memory [20]. Therefore, it is called firmware as it appears as both hardware and software. In order to speed up the booting process, BIOS can be copied from ROM to RAM for computer boot. This action is known as shadowing [20].

The usual sequence of the BIOS [20] when the computer is switched on is CMOS setup checking, following by interrupt handlers and device drivers loading, registers and power management initialization, power-on self-test (POST) execution, system settings display, device boot and lastly bootstrap sequence initiation. POST aims to ensure the system devices are functioning well. POST consists of system hardware and chipset registers initialization, power management initialization, RAM testing, keyboard enabling, serial and parallel port testing, floppy disk driver and hard disk drive controllers initialization, and system summary information display.

There are some limitations of legacy BIOS such as maximum access to 1MB for programs, maximum size of 2TB for HDD, limitation of BIOS size is 1MB and limitation to get extended, difficult manageability of the source code, and etc. Therefore, Unified Extensible Firmware Interface (UEFI) is introduced as a new industry standard for an extra interface layer sitting on top of the traditional BIOS which is in between of hardware and operating system as shown in Figure 2.9.