Graphics Output Protocol(GOP) Validation Automation

Major Project Report

Submitted in partial fulfillment of the requirements

for the degree of

Master of Technology

in

Electronics & Communication Engineering

(Embedded Systems)

By

Ankurkumar Patel

(13MECE20)

Electronics & Communication Engineering Branch

Electrical Engineering Department

Institute of Technology

Nirma University

Ahmedabad-382 481

May 2015

Graphics Output Protocol(GOP) Validation Automation

Major Project Report

Submitted in partial fulfillment of the requirements

for the degree of

Master of Technology

in

Electronics & Communication Engineering

(Embedded Systems)

Ankurkumar Patel
(13MECE20)

Under the guidance of

External Project Guide:	Internal Project Guide:
Ramadeva, Dwarakanath	**Prof.Sachin Gajjar**
Engineering Manager,VPG,	Assistant Professor, EC Department,
Intel Technology India Pvt. Ltd.,	Institute of Technology,
Bangalore.	Nirma University, Ahmedabad.

Electronics & Communication Engineering Branch

Electrical Engineering Department

Institute of Technology

Nirma University

Ahmedabad-382 481

May 2015

Declaration

This is to certify that

a. The thesis comprises my original work towards the degree of Master of Technology in Embedded Systems at Nirma University and has not been submitted elsewhere for a degree.

b. Due acknowledgment has been made in the text to all other material used.

- Ankurkumar Patel
13MECE20

Disclaimer

NIRMA UNIVERSITY
INSTITUTE OF TECHNOLOGY

Certificate

This is to certify that the Major Project entitled **"Graphics Output Protocol(GOP) Validation Automation"** submitted by **Patel Ankurkumar G. (13MECE20)**, towards the partial fulfillment of the requirements for the degree of Master of Technology in Embedded Systems, Nirma University, Ahmedabad is the record of work carried out by him under our supervision and guidance. In our 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 our knowledge,haven't been submitted to any other university or institution for award of any degree or diploma.

Date: Place: Ahmedabad

Prof.Sachin Gajjar
Internal Guide

Dr. N.P. Gajjar
Program Coordinator

Dr. D.K.Kothari
Section Head, EC

Dr. P.N.Tekwani
Head of EE Dept.

Dr. K. Kotecha
Director, IT

Certificate

This is to certify that the Major Project (Phase- I) entitled **"Graphics Output Protocol Validation Automation"** submitted by **Patel Ankurkumar G.(13MECE20)**, towards the partial fulfillment of the requirements for the degree of Master of Technology in Embedded Systems, Nirma University, Ahmedabad is the record of work carried out by him under our supervision and guidance. In our opinion, the submitted work has reached a level required for being accepted for examination.

Ramadeva, Dwrakanath
Engineering Manager,VPG
Intel Technology India Pvt. Ltd.
Bangalore

Acknowledgements

I would like to express my gratitude and sincere thanks to **Dr. P.N.Tekwani**, Head of Electrical Engineering Department, and **Dr. N.P.Gajjar**, PG Coordinator of M.Tech Embedded Systems program for allowing me to undertake this thesis work and for his guidelines during the review process.

I take this opportunity to express my profound gratitude and deep regards to **Prof.Sachin Gajjar**, guide of my major project for his exemplary guidance, monitoring and constant encouragement throughout the course of this thesis. The blessing, help and guidance given by him time to time shall carry me a long way in the journey of life on which I am about to embark.

I would take this opportunity to express a deep sense of gratitude to **Ramadeva, Dwarakanath** ,Engineering Manager,Intel Technology India Pvt. Ltd. for his cordial support, constant supervision as well as for providing valuable information regarding the project and guidance, which helped me in completing this task through various stages.I would also thank to **T, Nithin**,my Project Mentor for always helping,giving me good suggestions, solving my doubts and guide me to complete my project in better way.

I am obliged to **P, Raghupathy** and **S, Vidhya K** staff members of Firmware validation team, Intel Technology India Pvt. Ltd. for the valuable information provided by them in their respective fields. I am grateful for their cooperation during the period of my assignment.

Lastly, I thank almighty, my parents, brother, sisters and friends for their constant encouragement without which this assignment would not be possible.

- Ankurkumar Patel
13MECE20

Abstract

Graphics output is important in the pre-boot space to support modern firmware features. These features include the display of logos, the localization of output to any language, setup and configuration screens.

Graphics output may also be required as part of the startup of an operating system. There are potentially times in modern operating systems prior to the loading of a high performance OS graphics driver where access to graphics output device is required. The Graphics Output Protocol supports this capability by providing the Extensible Firmware Interface(EFI) Operating system(OS) loader access to a hardware frame buffer and enough information to allow the OS to draw directly to the graphics output device.

Validation of such protocol will require large amount of tests to be executed on them. The automation of these tests will effectively reduce the time required in Validation phase of the Product Development Life-Cycle and man power.

In this Project efforts are made to reduce the resources requirement and human intervention.Reduction in human intervention will cause great reduction in execution time, because normal execution of such tests approximately takes a day while the automation can complete that execution in approximately 2 hours.In this case the whole execution will be managed and supervised by validation server so human Involvement is not at all required.

Contents

List of Figures

Abbreviation Notation and Nomenclature

GOP	Graphics Output Protocol
VBT	VBIOS Table
VBIOS	Video BIOS
BIOS	Basic Input Output System
DP	Display Port
eDP	Embedded Display Port
HDMI	High Definition Media Interface
CSM	Compatibility Support Module
EFI	Extensible Firmware Interface
UEFI	Unified Extensible Firmware Interfaces
VGA	Video Graphics Array
VBE	Video BIOS Enable
PCI	Peripheral Component Interconnect
EPL	Eclipse Public License
EDID	Extended Display Identification Data
IHV	Independent Hardware Vendors
GUID	Global Unique Identifier
GPT	Global Page Table
SMM	System Management Module
OS	Operating System
USB	Universal Serial Bus
IBM	International Business Machine
LCD	Liquid Crystal Display
IA	Intel Architecture
DVI	Digital Video Interface
DUT	Device Under test
ACPI	Advanced Configuration and Power Interface

ROM	Read Only Memory
OSPM	System-directed configuration and Power Management
POST	Power-On Self-Test

Chapter 1

Introduction

1.1 Background

For over 37 years, Intel Corporation has been in a unique position of designing and providing Graphics firmware that are optimized along the three axes of performance, cost, and compatibility. Successfully meeting these requirements across many generations has resulted in complex designs that impose severe validation requirements on the drivers.

Some of the important issues unique to Intel that necessitate rigorous validation are the high volumes of display firmware that are shipped and used in a very diverse set of applications. The high volume of firmware shipped implies that the cost of a bug escape is extremely high in loss of customer revenue and reputation of product reliability. Thus, Intel puts an extraordinary amount of effort in ensuring the firmware thoroughly validated.

Intel has tackled the validation challenge in ways that include expanding the scope of validation in pre-silicon and post silicon, making older techniques more efficient and validating across as many levels of abstraction as applicable to break the problems complexity and innovating better techniques to address complexity of new firmware features.

The validation process of firmware requires lots of time, considerable amount of efforts, large number of human resources and equipment. Requirement of reduction in time required to develop and ship the firmware leads us to automation of such complex validation process.

1.2 Motivation

Validation techniques for Graphics firmware may be broadly classified as windows firmware validation and Android firmware validation .Windows firmware validation can be further classified into GOP validation,OS display driver validation and Media validation where GOP validation will have some predefined tests to check for functionality but executing these tests will require lot more methodology and lot more resources and it is becoming complex day by day because more and more new features are getting added in each and every release. Need for a powerful methodology capable of finding problems with firmware becomes more pronounced as more complex drivers are designed and deployed.

1.3 Problem Definition

GOP validation test cases are becoming complex and bigger as the GOP is becoming Complex due to addition of new features and availability of so many displays in market, apart from that display resolution are becoming bigger and new displays are coming in market with high definition capabilities. GOP validation is required to characterize the behavior of GOP and they should meet the specification. As the complexity of the firmware increases, the number of tests to be done also increases. So to carry out firmware validation efficiently we require content development as well as automation frame work.

1.4 Objective

One of the major goal of GOP validation is detecting bugs that have emerged during development of GOP. In addition, the goal includes to Validate specific features to ensure that the driver meets the functional requirements as defined in the functional specifications. GOP validation is required to face system and testing environment failures analysis and provide the root cause analysis at the fastest turn-around time, such that in case there is a real problem, it can be repaired immediately, enabling another Firmware iteration start in order to keep up with the time-to-production schedule.

1.5 Scope

The scope of the project is to give insight of firmware validation, automation used in firmware validation and debug techniques. Further description of how we can develop content and automation frame work used in firmware validation, which helps in reducing the time required for validation of Intel Firmware.

1.6 Requirements

For the project work carried out at Intel, needs an in-depth knowledge of the GOP firmware, Display technologies, understanding of the post silicon environment and in particularly the firmware validation environment like Hot plugging, Host validation server Interface, Platform Structure is required. For the coding purpose, in-depth knowledge of C language, EFI shell coding techniques, Visual Studio and other tools specific for C coding at Intel is needed.

1.7 Project Work Flow

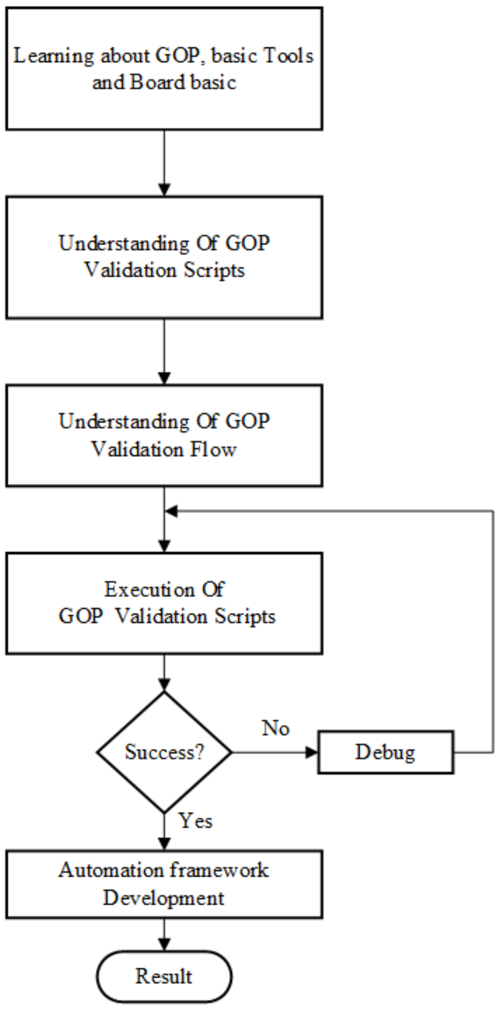

Learning about GOP, basic Tools and Board basic
Understanding Of GOP Validation Scripts
Understanding Of GOP Validation Flow
Execution Of GOP Validation Scripts
Success?
No
Debug
Yes
Automation framework Development
Result

Figure 1.1: Project Flow Chart

In this project we have to write the scripts in C language specific tool for validating GOP firmware, for that first we need to learn some basic tool which are involve in validation. To validate a firmware we should have clear idea about the execution flow of validation , for that we need to manually execute test scripts After having understanding of validation execution flow we need to divide the flow in manual and automatic functioning things . Then we need to develop code for the manual things then we need to validate our script and if in this process we hit some bug then we need to debug that. In the end we are trying to automate the whole process.

1.8 Gantt Chart

The timeline of project work from the start of the project is shown in below gantt chart.

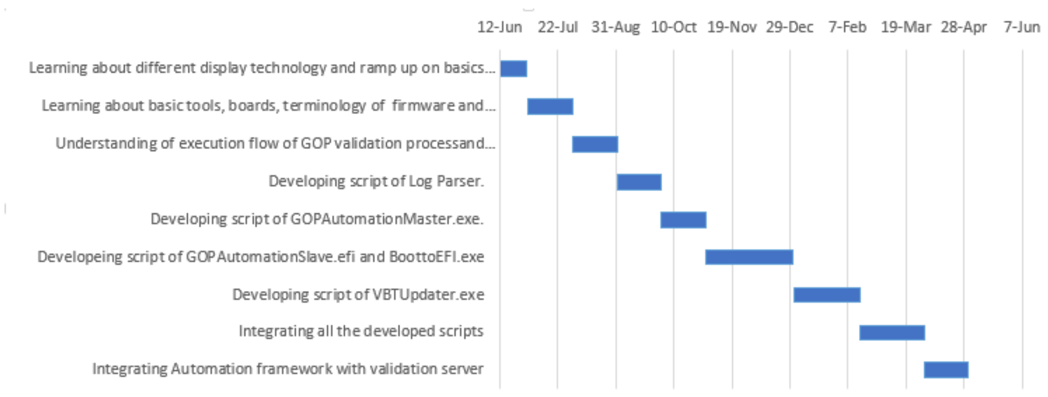


Figure 1.2: Gantt Chart

Chapter 2

Literature review

2.1 High Definition Media Interface(HDMI)

HDMI can deliver high quality sound or vision without the risk of quality loss due to the conversion or compression of a video or audio signal. HDMI pictures are smoother and sharp. Sound is also crisp and taut, without any distortion. And of course, using the single cable HDMI can get rid of a lot of messy cables snaking around your home theatre kit .Because of its digital nature, HDMI also works well with fixed-pixel displays such as LCD, plasma or DLP screens and projectors. A HDMI cable allows you to exactly match pixel-by pixel the native resolution of the screen with whatever source device you've got connected. HDMI systems will also automatically convert a picture into its most appropriate format, such as 16:9 or 4:3.HDMI signals are digital in nature while conventional TVs and radios operate on analog signals, on the contrary HDTVs works on digital signals. HDMI has some built-in smarts that allow you to control any device connected via HDMI through the one remote.[1]

2.1.1 Features of HDMI

- HDMI technology eliminates unnecessary signal conversions

- HDMI technology supports standard, enhanced, or high-definition video at 24 bits/pixel, 165MHz max clock frequency.
- HDMI technology supports up to 8 channel digital audio on a single cable eliminating costly A/D signal conversions.
- HDMI offers Bi-directional control signal transfer.
- HDMI offers 5 Gbps bandwidth, 55 percent spared for future expansion.
- HDMI offers 1 simple, user-friendly connector.
- HDMI technology is backward compatible to DVI hot plug enabled assemblies upto 5 meters in length

2.1.2 Applications of HDMI

- For Connecting HD-TVs, Digital Flat-Panel Displays and Other Components with HDMI Connections to Digital DVD Player, Digital A/V Receiver and Other Equipment with HDMI Connections
- Entertainment Center: DVD Player, DTV, Camcorder, Computer Display Devices, especially LCD Panels/monitor
- Blue ray disc video and HD DVDs

HDMI interface is most useful when used with digital audio or video devices including gaming consoles such asPS3and Xbox 360 as well as Blue-Ray players and set top boxes. HDMI cable supports single cable interface as well as any major PC/TV video format. Best use of HDMI cable is observed with high-definition video with up to eight digital audio channels[1]

2.2 Display Port(DP)

It's the industry replacement for outmoded display technologies such as DVI, LVDS and VGA and it's currently being built into all new PC chipsets, GPU's and display controllers from major silicon manufacturers. Display Port utilizes a state-of-the-art digital protocol and provides an expandable foundation to enable amazing digital display experiences. Designed for low power implementation and high performance, Display Port enables the next generation display technology while providing compatibility with existing equipment. Designed specifically for usage in space-constrained applications like ultra-thin notebooks, netbooks and graphic cards where connector space is at a premium and where display performance really matters.[2]

Display Port is designed to be the future-ready, scalable solution for high performance digital display connectivity. It enables the highest resolutions, the fastest refresh rates and deepest color depths over standard cables.[2]

Display Port has unique features and capabilities that enable exciting new types of displays and display usages. And it doesn't require PC owners to replace all of their equipment because simple adaptors allow Display Port enabled devices to connect to monitors and projectors that use older technologies such as DVI, HDMI and VGA.[2]

2.2.1 Important features of Display Port 1.3

- Next Generation Revolutionary Display Interface Technology
  - Micro-packet architecture over 1- 4 lanes
  - Up to 8.1 Gbps per lane-32.4 Gbps over 4 lanes
  - Allowing for overhead - 25.92 Gbps over 4 lanes
  - Increased bandwidth allows 5K monitors (5120 x 2880) with a single cable

  and no compression
  - Multiple display support-two 4K UHD monitors (3840 x 2160) using VESA Coordinated Video Timing supported from a single connector
  - Continued support for video conversion to VGA, DVI and HDMI
  - Support for HDCP 2.2 and HDMI 2.0 with CEC
  - Support for 4:2:0 pixel structure which enables 8K x 4K displays
  - Ability to support UHD monitor @60Hz and 24-bit color over (2) lanes while assigning other 2 lanes for alternate data types such as USB as allowed in Dock Port

2.2.2 Important features of Display Port 1.2

- Revolutionary Display Interface Technology
  - Micro-packet architecture over 1- 4 lanes
  - Up to 5.4 Gbps per lane-21.6 Gbps over 4 lanes
  - Auxiliary channel for bidirectional data communications at 1 Mbps/ 720 Mbps Optional
  - Powered connectors (providing up to 1.5W)
  - Multiple display support-63 separate AV streams supported from a single connector
  - Scalable for large and small devices, displays, and cables
- Designed Specifically for High Performance Graphics and Flat Panel Displays
  - Low voltage compatibility for integration into the latest chipsets, GPU's and display controllers
  - Low power and scalable lanes for integration directly in LCD panels
  - Low EMI and RFI

  - Compact external connector with optional latching
  - Mini Display Port connector is space efficient for high performance for laptops, netbooks and multi-output graphics cards

- Long Cable Support
  - Up to 15 meters and beyond
  - Higher bandwidth active cables and hybrid cables also available (utilize DP power pin)
- High Performance is Standard
  - Beyond high definition; WQXGA at 10 bit color
  - Beyond Full HD 3D stereo support at 120hz
  - DP v1.2 enables High Color Range Quad Full HD delivered over a standard Display Port connector (up to 4K x 2K at 60 FPS and 24 bpp)
- Ultra-low Latency for super-fast response
- High performance over standard cables
- Enables Exciting New Display Designs
  - Unique direct drive monitor (DDM) capability enables ultra-thin monitors
  - Audio and HDCP enable high definition content playback and built-in speakers on multi-function displays
- Ubiquitous Connectivity to Any Display
  - Interoperability with DVI and HDMI signaling over Display Port Connector
  - Compatibility with existing displays via simple adapters

  - Dual-mode Display Port PCs can be connected to DVI, VGA and HDMI monitors and projectors.

- Reduces Wiring and enables Higher Performance in Laptops
  - Embedded Display Port (eDP) enables lighter weight internal cabling
  - Embedded Display Port reduces system power enabling longer notebook battery life
  - Enables higher-performance notebook LCDs for emerging applications such as 3D stereo, 120Hz refresh rates and the deepest color depths

2.3 Basic Input Output System(BIOS)

BIOS is the first code run by a PC when powered on. It acts as a layer between OS and Hardware. BIOS initialize the various platform components like CPU initialization, core initialization, memory and chipset initialization etc. The BIOS must do its job before your computer can load its operating system and applications.[6]
The basic input/output system (BIOS), also known as the System BIOS or ROM BIOS, is a de facto standard defining a firmware interface.

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 PC starts up, the first job for the BIOS is to initialize and identify system devices such as the video display card, keyboard and mouse, hard disk drive, optical disc drive and other hardware.
The BIOS then locates software held on a peripheral device (designated as a 'boot device'), such as a hard disk or a CD/DVD, and loads and executes that software, giving it control of the PC. This process is known as booting, or booting up, which is short for bootstrapping.[6]

BIOS software is stored on a non-volatile ROM chip built into the system on the motherboard. The BIOS software is specifically designed to work with the particular type of system in question, including having knowledge of the workings of various devices that make up the complementary chipset of the system. In modern computer systems, the BIOS chip's contents can be rewritten, allowing BIOS software to be upgraded.[6]

BIOS features are as follows:

- It acts as a layer between OS and Hardware
- It gets your computer up and running
- Initializes the hardware like Microprocessor, memory, chipset, devices, peripherals etc.
- Provides Power Management functionality through ACPI
- Loads and hands control over to the OS boot loader
- Provides a set of standardized routines for the OS to use
- Abstracts motherboard and silicon specifics from the OS
- Prepares system to run an OS
- Provides runtime services to the OS e.g. disk and video

The Advanced Configuration and Power Interface (ACPI) is a specification which was developed to establish industry common interfaces enabling robust operating system (OS)-directed motherboard device configuration and power management of both devices and entire systems. ACPI is the key element in Operating System-directed configuration and Power Management (OSPM).[6]

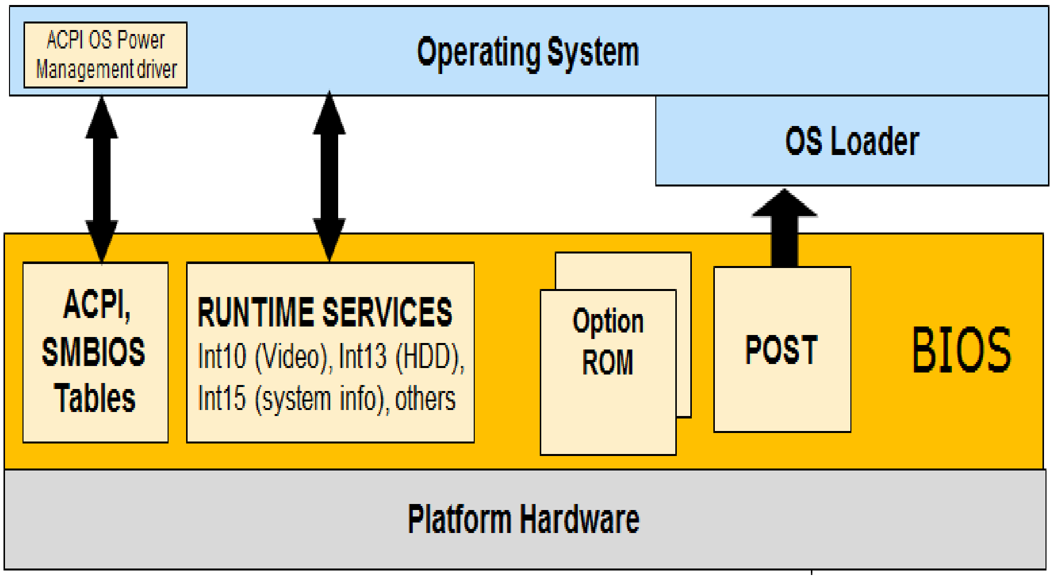
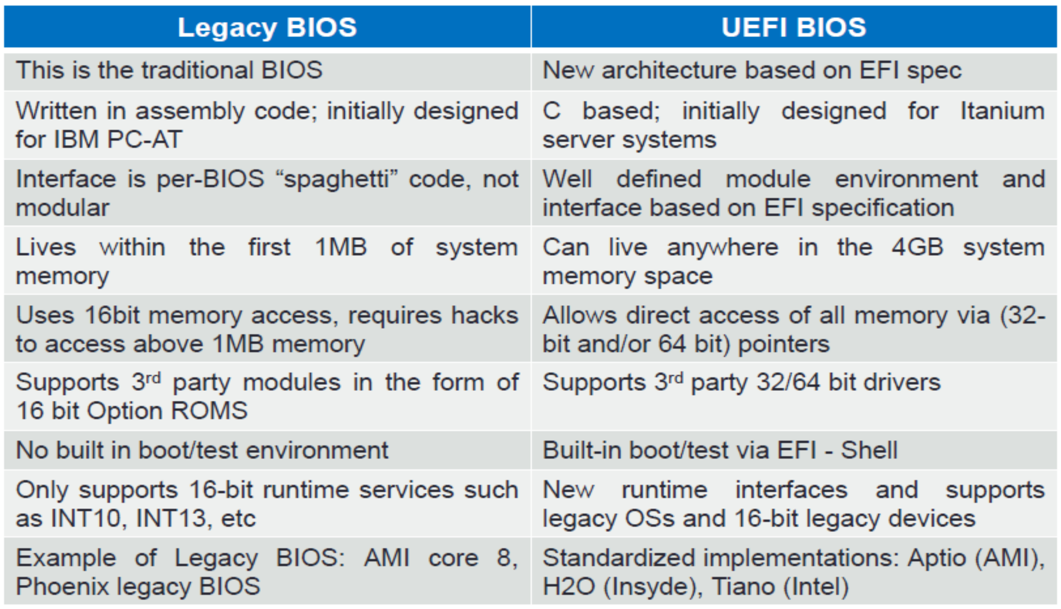

ACPI OS Power Management driver
Operating System
OS Loader
ACPI, SMBIOS Tables
RUNTIME SERVICES Int10 (Video), Int13 (HDD), Int15 (system info), others
Option ROM
POST
BIOS
Platform Hardware

Figure 2.1: BIOS Overview[6]

Power-On Self-Test (POST) refers to routines run immediately after power is applied,by nearly all electronic devices.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.

Currently, Industry has migrated from Legacy BIOS to a standard and modular EFI BIOS. EFI BIOS offers new & improved features and flexibility for code developers. The difference between Legacy BIOS and EFI BIOS is shown in fig. 2.2.

BIOS code used today supports legacy BIOS as well as EFI BIOS by means of a CSM module. All the OS are not compatible with the EFI BIOS; CSM module is used to run appropriate BIOS code. When CSM mode is ON, system will boot to legacy BIOS and if CSM mode is OFF, System will boot to native EFI BIOS.



Legacy BIOS	UEFI BIOS
This is the traditional BIOS	New architecture based on EFI spec
Written in assembly code; initially designed for IBM PC-AT	C based; initially designed for Itanium server systems
Interface is per-BIOS “spaghetti” code, not modular	Well defined module environment and interface based on EFI specification
Lives within the first 1MB of system memory	Can live anywhere in the 4GB system memory space
Uses 16bit memory access, requires hacks to access above 1MB memory	Allows direct access of all memory via (32-bit and/or 64 bit) pointers
Supports 3rd party modules in the form of 16 bit Option ROMS	Supports 3rd party 32/64 bit drivers
No built in boot/test environment	Built-in boot/test via EFI - Shell
Only supports 16-bit runtime services such as INT10, INT13, etc	New runtime interfaces and supports legacy OSs and 16-bit legacy devices
Example of Legacy BIOS: AMI core 8, Phoenix legacy BIOS	Standardized implementations: Aptio (AMI), H2O (Insyde), Tiano (Intel)

Figure 2.2: Legacy BIOS vs. EFI BIOS[6]

The CSM translates the information generated under the EFI environment into the information required by the legacy environment and makes the legacy BIOS services available for booting to the operating system and for use in runtime.[6]

Chapter 3

Unified Extensible Firmware Interface (UEFI)

3.1 Introduction

In computing, the Unified Extensible Firmware Interface (UEFI) is a specification that defines a software interface between an operating system and platform firmware. UEFI is meant to replace the Basic Input Output System (BIOS) firmware interface, originally present in all IBM PC-compatible personal computers. In practice, most UEFI firmware images provide legacy support for BIOS services. UEFI can support remote diagnostics and repair of computers, even without another operating system.[3]

Intel developed the original EFI (Extensible Firmware Interface) specification. Some of the EFI's practices and data formats mirror those from Microsoft Windows In 2005, UEFI deprecated EFI 1.10 (the final release of EFI). The Unified EFI Forum manages the UEFI specification.

The original motivation for EFI came during early development of the first Intel-

HP Itanium systems in the mid-1990s. BIOS limitations (such as 16-bit processor mode, 1 MB addressable space and PC AT hardware) were unacceptable for the larger server platforms Itanium was targeting. The effort to address these concerns began in 1998 and was initially called Intel Boot Initiative; it was later renamed to EFI.[3]

In July 2005, Intel ceased development of the EFI specification at version 1.10, and contributed it to the Unified EFI Forum, which has evolved the specification as the Unified Extensible Firmware Interface (UEFI). The original EFI specification remains owned by Intel, which exclusively provides licenses for EFI-based products, but the UEFI specification is owned by the Forum.[3]

Version 2.1 of the UEFI (Unified Extensible Firmware Interface) specification was released on 7 January 2007. It added cryptography, network authentication and the User Interface Architecture (Human Interface Infrastructure in UEFI). The current UEFI specification, version 2.4, was approved in July 2013.[3]

The interface defined by the EFI specification includes data tables that contain platform information, and boot and runtime services that are available to the OS loader and OS. UEFI firmware provides several technical advantages over a traditional BIOS system:

- ability to boot from large disks (over 2 TB) with a GUID Partition Table (GPT)
- CPU-independent architecture
- CPU-independent drivers
- flexible pre-OS environment, including network capability
- modular design

3.2 Compatibility Support Module

The Compatibility Support Module (CSM) is a component of the UEFI firmware that provides legacy BIOS compatibility by emulating a BIOS environment, allowing legacy operating systems and some option ROMs that do not support UEFI to still be used. CSM also provides required legacy System Management Mode (SMM) functionality as an addition to features provided by the UEFI SMM. This is optional, and highly chipset and platform specific. An example of such a legacy SMM functionality is providing USB legacy support for keyboard and mouse, by emulating their classic PS/2 counterparts.[3]

3.3 Intel EFI

Intel's implementation of EFI is the Intel Platform Innovation Framework, codenamed "Tiano." Tiano runs on Intel's XScale, Itanium and IA-32 processors, and is proprietary software, although a portion of the code has been released under the BSD license or Eclipse Public License (EPL) as TianoCore. TianoCore can be used as a payload for coreboot.[3]

Phoenix Technologies' implementations of UEFI include its SecureCore and SecureCore Tiano products. American Megatrends offers its own UEFI firmware implementation known as Aptio, while Insyde Software offers InsydeH2O, its own implementation of Tiano.[3]

3.4 UEFI Driver Model Goals

- Compatible
  Drivers conforming to this specification must maintain compatibility with the previous EFI and UEFI Specification.

- Simple

  Drivers that conform to this specification must be simple to implement and simple to maintain. The UEFI Driver Model must allow a driver writer to concentrate on the specific device for which the driver is being developed. A driver should not be concerned with platform policy or platform management issues. These considerations should be left to the system firmware.

- Scalable

  The UEFI Driver Model must be able to adapt to all types of platforms. These platforms include embedded systems, mobile, and desktop systems, as well as workstations and servers.

- Flexible

  The UEFI Driver Model must support the ability to enumerate all the devices, or to enumerate only those devices required to boot the required OS. The minimum device enumeration provides support for more rapid boot capability, and the full device enumeration provides the ability to perform OS installations, system maintenance, or system diagnostics on any boot device present in the system.

- Extensible

  The UEFI Driver Model must be able to extend to future bus types as they are defined.

- Portable
  Drivers written to the UEFI Driver Model must be portable between platforms and between supported processor architectures.

- Interoperable

  Drivers must coexist with other drivers and system firmware and must do so without generating resource conflicts.

- Describe complex bus hierarchies

  The UEFI Driver Model must be able to describe a variety of bus topologies from very simple single bus platforms to very complex platforms containing many buses of various types.

- Small driver footprint

  The size of executables produced by the UEFI Driver Model must be minimized to reduce the overall platform cost. While flexibility and extensibility are goals, the additional overhead required to support these must be kept to a minimum to prevent the size of firmware components from becoming unmanageable.

- Address legacy option rom issues

  The UEFI Driver Model must directly address and solve the constraints and limitations of legacy option ROMs. Specifically, it must be possible to build add-in cards that support both UEFI drivers and legacy option ROMs, where such cards can execute in both legacy BIOS systems and UEFI-conforming platforms, without modifications to the code carried on the card. The solution must provide an evolutionary path to migrate from legacy option ROMs driver to UEFI drivers.

3.5 Requirements of UEFI

A UEFI driver is required for any PC hardware device needed for the boot process to complete.Hardware devices can be categorized into the following:

- Graphic output devices: Simple text, graphics output
- Console devices: Simple input provider, simple input ex, simple pointer - mice, serial I/O protocol (remote consoles)

Note that independent hardware vendors (IHVs) can choose not to implement all of the required elements of the UEFI specification. For example all elements might not be implemented on a specialized system configuration that does not support all the services and functionality implied by the required elements. Also, some elements are required depending on a specific platform's features. Some elements are required depending on the features that a specific driver requires. Other elements are recommended based on coding experience, for reasons of portability, and/or for other considerations. It is recommended that you implement all required and recommended elements in your drivers.[3]

Chapter 4

GOP Automation Framework

4.1 GOP

The Graphics Output Protocol (GOP) is enabled by UEFI driver to support graphic console output in the pre-OS phase.The ultimate goal of GOP is to replace legacy VGA BIOS and eliminate VGA HW functionality.[4]

4.1.1 Advantages of GOP

- Easier portability and fewer resource constraints
- All GPUs within a platform become ”equal”; no more unique ”VGA enabled” GPU
- No more messy and hard-to-maintain proprietary INT15 handshaking between platform and GPU

4.1.2 Major differences between GOP drivers and legacy VBIOS

- Accessed through UEFI protocols vs. Interrupts and VGA/VBE interface

- Boot only services vs. both boot and OS run-time services
- Written in C instead of x86 assembler (not a spec requirement)

4.1.3 Major differences between GOP drivers and legacy VBIOS

- Accessed through UEFI protocols vs. Interrupts and VGA/VBE interface
- Boot only services vs. both boot and OS run-time services
- Written in C instead of x86 assembler (not a spec requirement)

4.2 GOP Automation

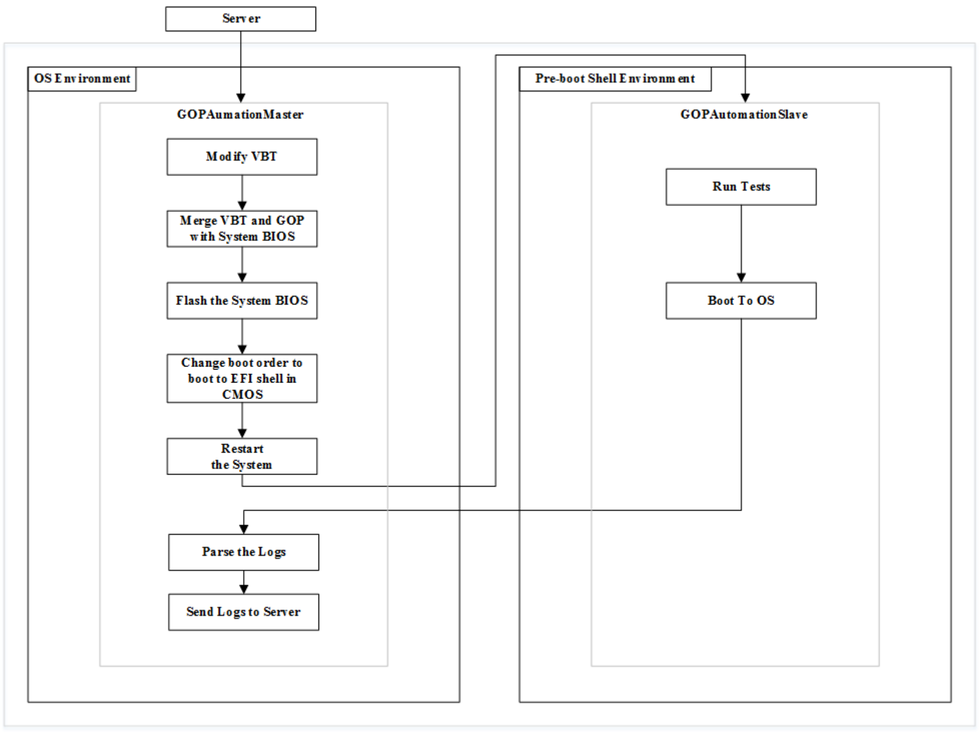


Figure 4.1: GOP Automation[7]

GOP Automation flow is shown in fig. 4.1.Here GOPAutomationMaster will run in windows environmet and GOPAutomationSlave will run in EFI environment.For different tests VBT settings are different so GOPAutomationMaster will modify the VBT as per test need then it will merge the modified VBT and GOP driver with system BIOS.then it will flash the modified BIOS to the system and will change the CMOS settings to always off to select GOP.then it will make the system to boot to EFI where GOPAutomationSlave will take charge and GOPAutomationSlave will run all the tests and will make the system to boot to windows.then GOPAtomation-Master will parse the logs and will send the final log to server.

4.3 Automation framework

4.3.1 Tools required

1. EFIFlash: Tool to flash BIOS in EFI shell
2. EPCS Util: Tool to read CMOS settings
3. EasyUEFI: Tool to change boot order from UEFI to Windows

4.3.2 Tools developed

1. GOPAutomationMaster.exe to RUN in windows environment
2. GOPAutomationSlave.efi to RUN in UEFI environment
3. VBTUpdater.exe to run in Windows environment
4. BoottoEFI.exe - Program to use EasyUEFI for changing boot sequence
5. ChangeBIOSConfig.exe - Tool to change CMOS Settings.
6. Test Selector - Tool for selecting testcases
7. Log Parser - executable to summarize logs.

4.3.3 Prerequisites

1. Dedicated client machine (pool) for GOP automation
2. USB drive must be connected to the system
3. Pre-loaded image of WIN 8 (bypassing login screen)
4. Pre-installed EasyUEFI application for boot order changing
5. Server environment should dump all utils/tools and tsel.csv file in a pre-determined location

4.4 Validation Server Setup

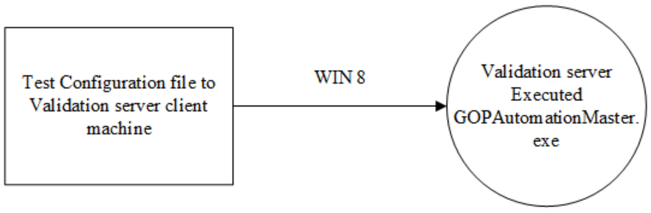

Test Configuration file to Validation server client machine
WIN 8
Validation server Executed GOPAutomationMaster. exe

Figure 4.2: Validation Server Execution flow[7]

validation server execution flow is shown in fig. 4.2. Validation Server is required to eliminate human intervention in validation procedure, it will trigger the things which is supposed to be automated in client machine using a specific framework which runs on server. The server provides connections with different DUTs which is required for test execution. Which will ultimately introduces reusability of resources in validation.

4.5 Master Program Flow in Windows

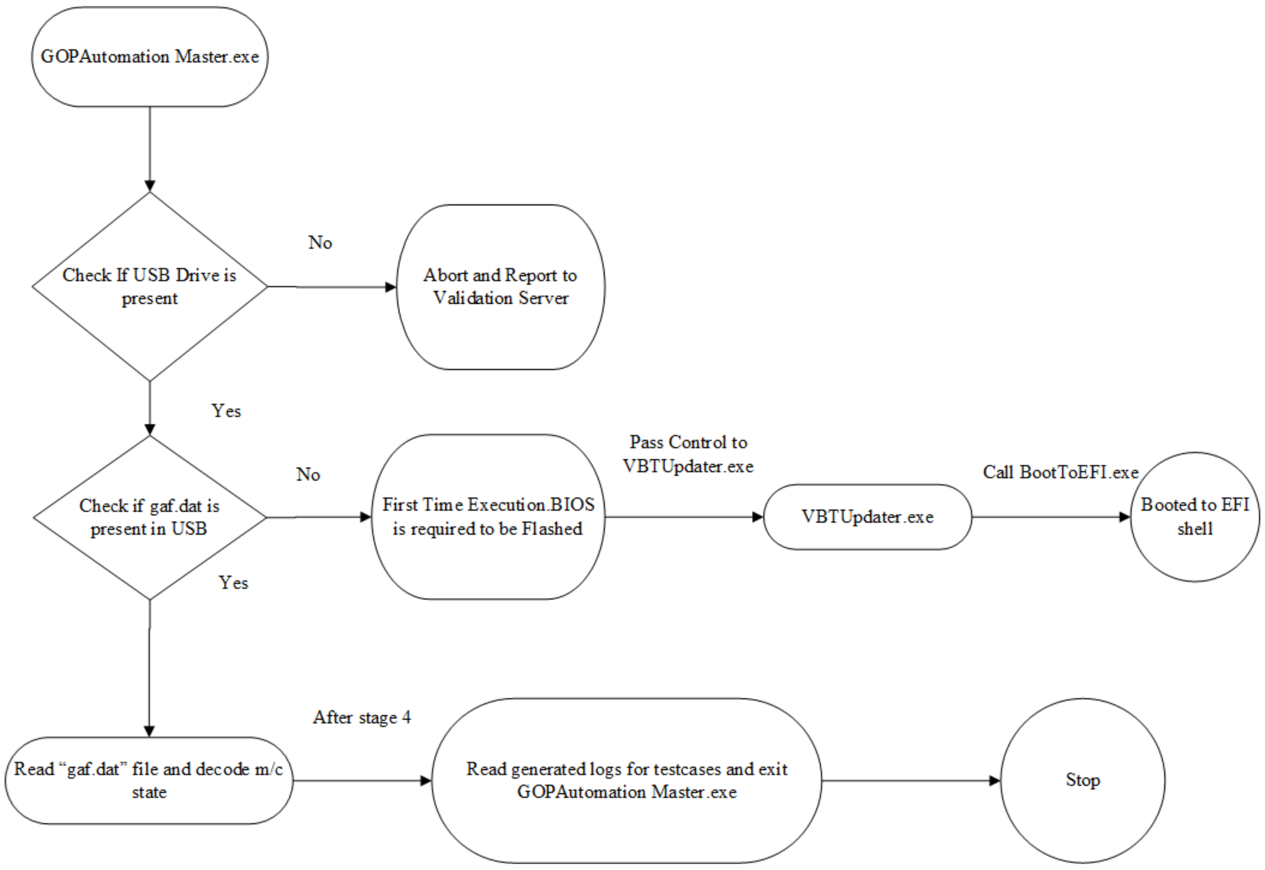

GOPAutomation Master.exe
Check If USB Drive is present
No
Abort and Report to Validation Server
Yes
Check if gaf.dat is present in USB
No
First Time Execution.BIOS is required to be Flashed
Pass Control to VBTUpdater.exe
VBTUpdater.exe
Call BootToEFI.exe
Booted to EFI shell
Yes
After stage 4
Read "gaf.dat" file and decode m/c state
Read generated logs for testcases and exit GOPAutomation Master.exe
Stop

Figure 4.3: GOPAutomationMaster.exe Execution Flow [7]

GOPAutomationMaster.exe flow is shown in fig.4.3. GOPAutomationMaster.exe is supposed to be executed on DUT in OS environment.it will check if USB is present or not. If it found that USB containing test cases is not connected than it will inform validation server that USB is not present.gaf.dat is the file which contains the information about the state of the automation. If it founds that USB is present than it will check for gaf.dat file and if it founds gaf.dat is not present which means it is the first execution than it knows that BIOS is required to be created and flashed, so it will go to VBTUpdater.exe followed by BootToEFI.exe, if it founds that that the gaf.dat is present than it will decode the state if it is state 4 than it runs the Log Parser to get summarized log and stops the execution of GOPAutomationMaster.exe.[5]

4.6 VBTUpdater Program Flows

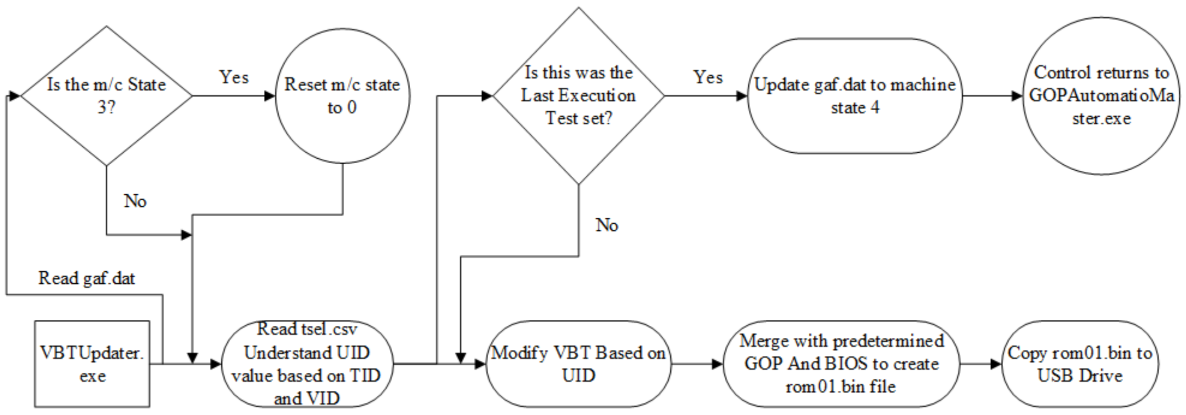

Is the m/c State 3?
Yes
Reset m/c state to 0
No
Read gaf.dat
VBTUpdater.exe
Read tsel.csv Understand UID value based on TID and VID
Is this was the Last Execution Test set?
Yes
No
Update gaf.dat to machine state 4
Control returns to GOPAutomatioMaster.exe
Modify VBT Based on UID
Merge with predetermined GOP And BIOS to create rom01.bin file
Copy rom01.bin to USB Drive

Figure 4.4: VBTUpdater.exe Execution Flow[7]

VBTUpdater.exe flow is shown in fig. 4.4. VBTUpdater.exe changes the configuration settings and then merges the BIOS with VBT and GOP and copy it to the USB.VBTUpdater.exe will read tsel.csv which contains information about different test cases and will modify VBT for each test cases merge with GOP and BIOS and creates modified BIOS to be flashed and will copy that file to USB drive.it will read the gaf.dat file and will check if it is state 3 than it will reset it to state 0 and will run all tests, if it founds that it is the last test execution than it will update state to 4 in gaf.dat file and will return the control to GOPAutomationMaster.exe

4.7 UEFI Shell Execution Flow

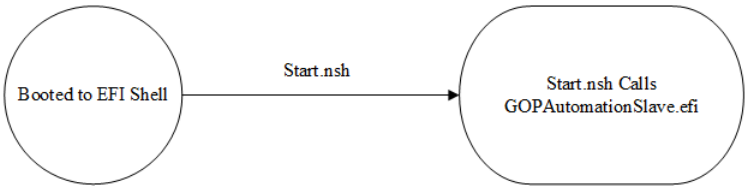


Figure 4.5: EFI shell Execution Flow [7]

The flow of BootToEFI.exe is shown in fig. 4.5. BootToEFI.exe will reboot the system to EFI shell and will load and run the Startup.nsh, here Startup.nsh will call the GOPAutomationSlave.efi

4.8 Operation States

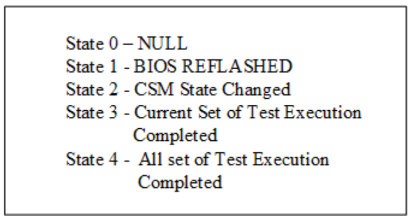

State 0 – NULL
State 1 - BIOS REFLASHED
State 2 - CSM State Changed
State 3 - Current Set of Test Execution Completed
State 4 - All set of Test Execution Completed

Figure 4.6: Operation States[7]

Different states of operations for GOP automation are as shown in fig. 4.6.

4.9 GOPAutomationSlave.efi Program Flow in UEFI Shell

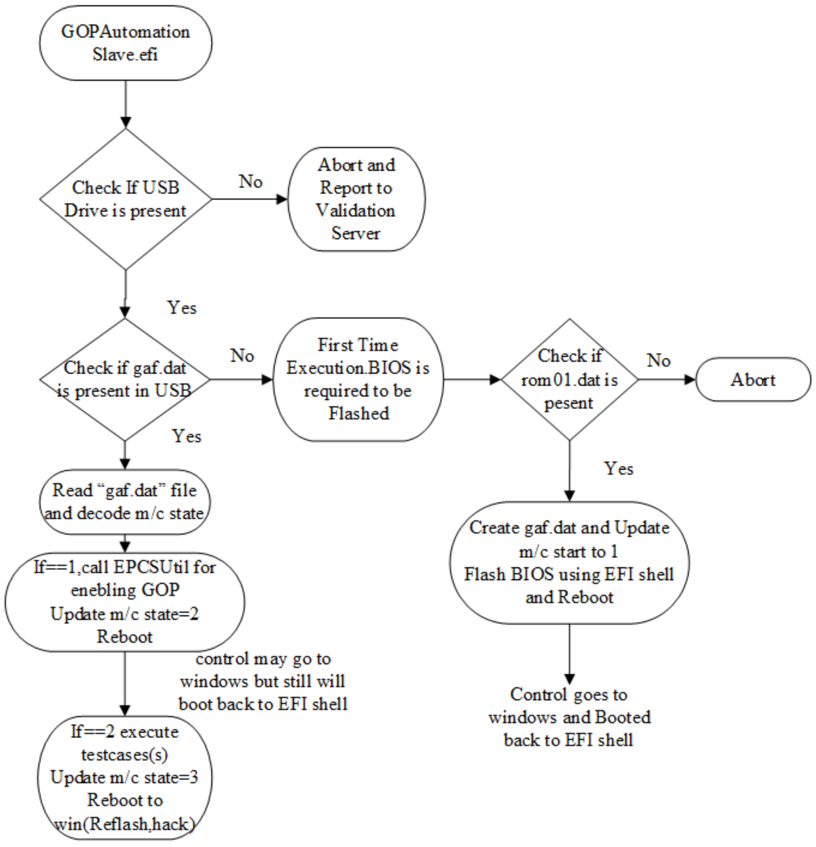

GOPAutomation Slave.efi
Check If USB Drive is present
No
Abort and Report to Validation Server
Yes
Check if gaf.dat is present in USB
No
First Time Execution.BIOS is required to be Flashed
Check if rom01.dat is pesent
No
Abort
Yes
Read "gaf.dat" file and decode m/c state
If==1,call EPCSUtil for enebling GOP Update m/c state=2 Reboot
control may go to windows but still will boot back to EFI shell
If==2 execute testcases(s) Update m/c state=3 Reboot to win(Reflash,hack)
Yes
Create gaf.dat and Update m/c start to 1 Flash BIOS using EFI shell and Reboot
Control goes to windows and Booted back to EFI shell

Figure 4.7: GOPAutomationSlave.efi Execution Flow[7]

The flow of GOPAutomationSlave.efi is shown in fig. 4.7. GOPAutomationSlave.efi is supposed to run on EFI shell environment.it will check for USB if USB is not present than it will abort the execution and report it to Validation server, if it founds that USB is present than check for gaf.dat if it is not present than it is first time execution, BIOS is required to be flashed for that modified BIOS is required so it will check for BIOS if it is not present it will abort the execution and if it founds BIOS present than it will create gaf.dat, set state as state 1,flash the BIOS using

EFI shell and reboot, and on reboot control goes to windows and it will booted back to EFI shell, if it founds USB present than it will read and decode gaf.dat file if state is 1 than EPCSUtils enable GOP in CSM module ,update state 1 to state 2 and reboot the system then control goes to windows and it will make system boot back to EFI shell, if it founds state as state 2 in gaf.dat than it will execute all test cases, set state as state3 and reboot the system to windows.[8]

4.10 Action as per gaf.dat files machine states

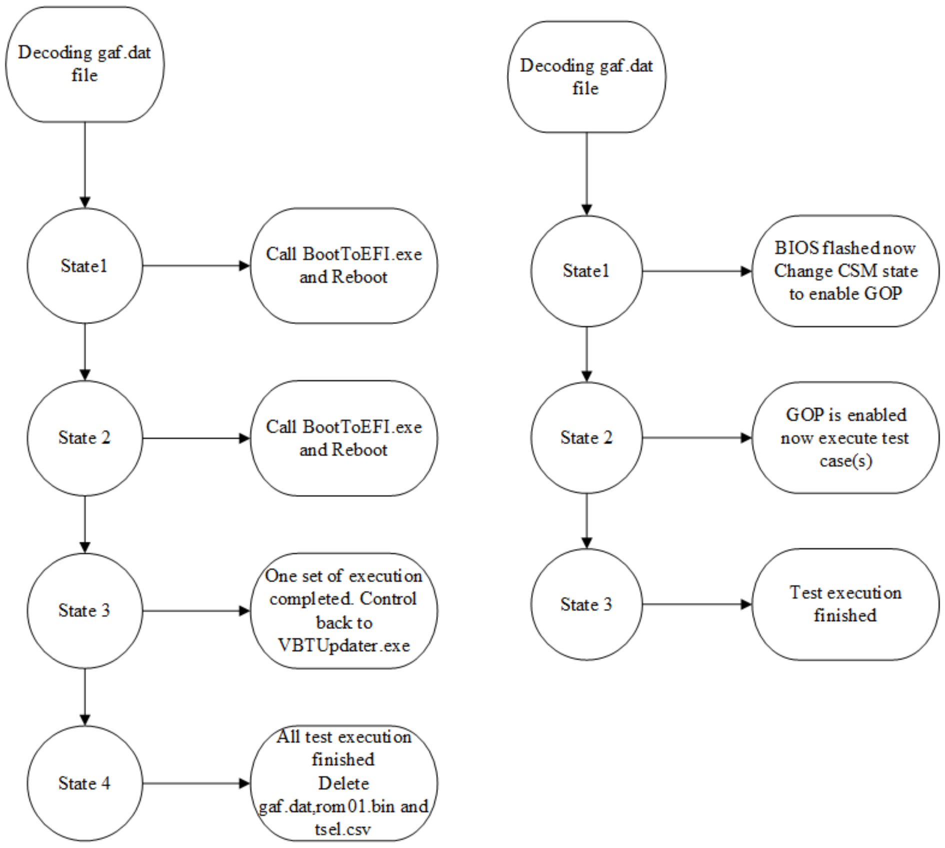

Decoding gaf.dat file
State1
Call BootToEFI.exe and Reboot
State 2
Call BootToEFI.exe and Reboot
State 3
One set of execution completed. Control back to VBTUpdater.exe
State 4
All test execution finished Delete gaf.dat,rom01.bin and tsel.csv
Decoding gaf.dat file
State1
BIOS flashed now Change CSM state to enable GOP
State 2
GOP is enabled now execute test case(s)
State 3
Test execution finished

Figure 4.8: Decoding gaf.dat with machine states[7]

Decoding of gaf.dat file and action as per decoding is described in fig. 4.8.If it is state 1 than action required will be calling BootToEFI.exe. if it is state 2 than call BootToEFI.exe and reboot ,if it is state 3 than one set of execution completed control back to VBTUpdater.exe, if it is state 4 than all execution is completed so delete gaf.dat,rom01.bin and tsel.csv.

Chapter 5

Conclusion

GOP validation process is the most exciting and challenging stage of display driver development process. The purpose of debug is to identify and resolve any bugs in firmware to ensure that firmware operates correctly for customers over the specification range. Firmware validation has become increasingly important to qualify products because of the following reasons: development alone does not fully account for many parameters impacting the firmware performance, efficient validation can lead us to achieve power/performance goals and quality measures.

We have expanded the automation to a level where we can submit the test content from validation server. This will help in doing work fast and more efficiently as we can control all the validation stuff remotely.

Chapter 6

Future Work

The server execution team is responsible for execution of test suite on server and for providing logs of the execution. For that they have to flash BIOS manually on each machine which is consuming significant time and resources. In this project we have implemented the method to flashing BIOS without human intervention.The server execution people will use our method to flash the BIOS and will be able to reduce their efforts and they will also integrate it with the existing server framework.

Some peer teams at Intel are also trying to automate their validation process.our framework will help them in automating their process by using some part of our code and some of the tools developed by us.

Bibliography

[1] $''HDMI\ Specifications'', [Online], Website, November$ 2014,
$http://www.microprocessor.org/HDMISpecification13a.pdf$

[2] $VESA, ''Display\ Port'', [Online], Website, November$ 2014,
$http://www.vesa.org/displayport-developer/why-displayport/$

[3] $UEFI, ''UEFI\ and\ GOP'', [Online], Website, November$ 2014,
$http://www.uefi.org/sites/default/files/resources/UPFS11\ P4\ UEFI\ GOP\ AMD.pdf$

[4] $''Graphics\ Output\ protocol'', [Online], Website, November$ 2014,
$https://coderwall.com/p/0xcxtq/uefi-shell-graphics-output-protocol$

[5] $Intel, ''UEFI\ Guide'', [online], Website, November$ 2014,
$http://www.intel.com/content/dam/doc/guide/uefi-driver-graphics-controller-guide.pdf$

[6] $Intel, Advanced\ Configuration\ and\ Power\ Interface\ Specification,$
$Revision5.0, December\ 6, 2011$

[7] $Intel, GOP\ Automation\ Framework\ Reference\ Manual, June$ 2014

[8] $Intel, EDK\ II\ BuildSpecification, June$ 2014.