

Generic IP independent BIOS Signing and Parsing

Gahan Saraiya

Institute of Technology
Nirma University

March 17, 2020

Outline

1. About the Project
2. Motivation
3. BIOS: Basics
4. Requirement Specification
5. My Contribution
6. Future Scope
7. Phase 1: Study of existing architecture for hotspot
8. Phase 2: Analysis and gathering detailed information
9. Requirement Specification
10. Why JSON?
11. Implementation of Parsing



In general to generate BIOS image (*.rom file), compilation of XYZ.c (source code) has to be done, this compilation not only involves compilation of DXE driver, PEI driver, EFI Application but also includes pre-processing checks, compression of raw files which takes huge amount of time depending on the system configuration. Implementation of this project aids in reduction of this compilation time.



Motivation: Stakeholders

- ▶ BIOS development Team
- ▶ Automation Team
- ▶ Validation Team
- ▶ Other Development Team



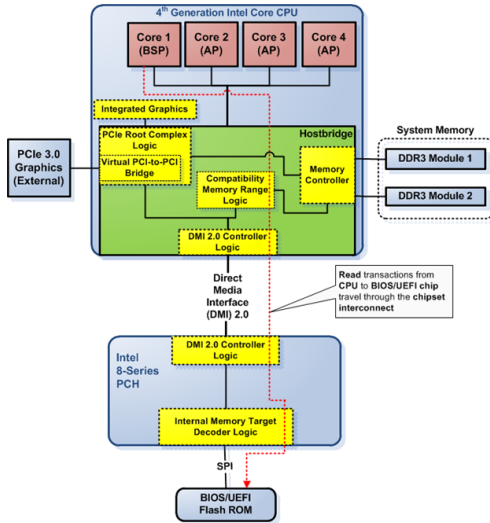
Motivation: Issues/Challenges to the industry (Towards my contribution)

- ▶ BIOS image generation: Compilation of whole source code
- ▶ More Time complexity: Compilation of source code to generate BIOS image



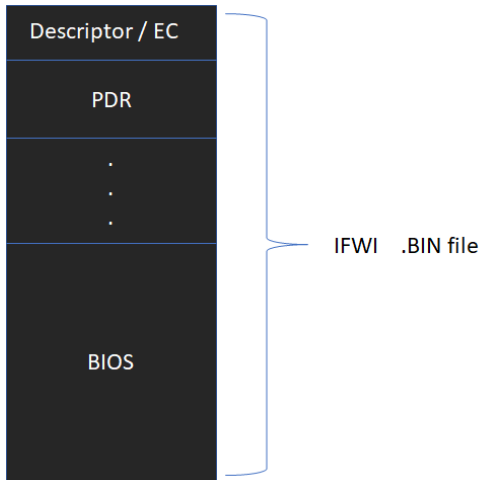
- ▶ Set of Software Routines
 - ▶ Initialize and test hardware on start
 - ▶ Provides the OS with a generic hardware abstraction
- ▶ the BIOS must do its job before your computer can load its operating system and applications





3. BIOS: Basics

BIOS: Firmware Image



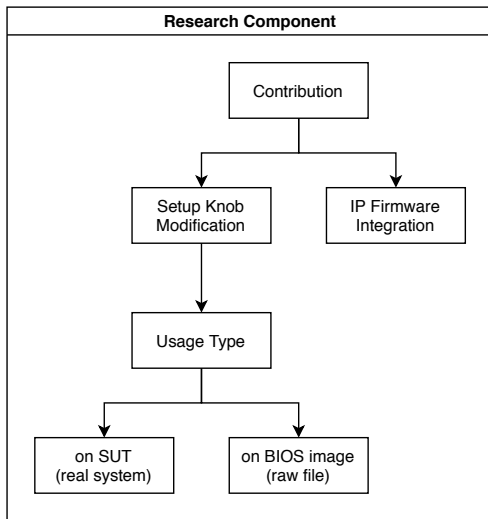
3. BIOS: Basics

Requirement Specification

- ▶ Visual Studio C/C++ IDE
- ▶ Python 3
- ▶ Visual Studio Code
- ▶ Memory Access Interfaces¹

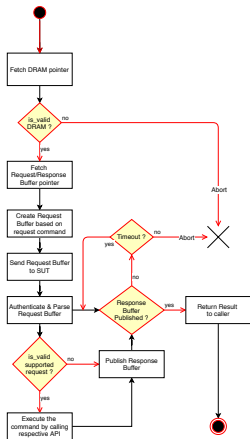
¹itp, itp2, windows, linux, ltb, dci, efi shell, etc.

My Contribution towards issues/challenges



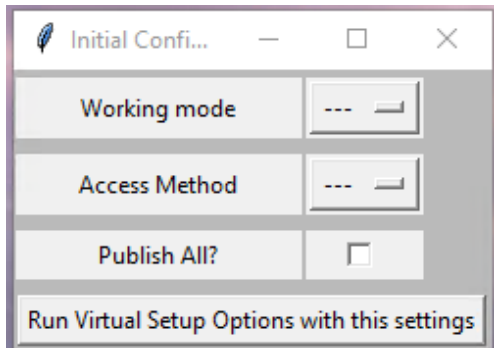
5. My Contribution

Setup Knobs Modification: Process Flow I



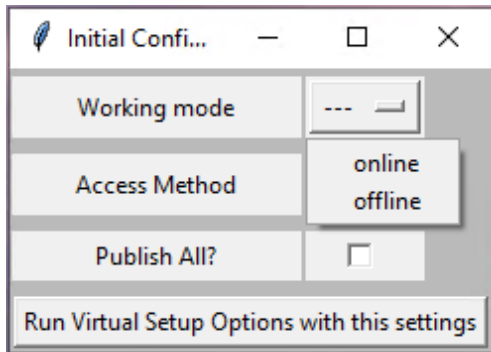
Setup Knobs Modification Flow on SUT

Setup Knobs Modification: Implementation Snaps I



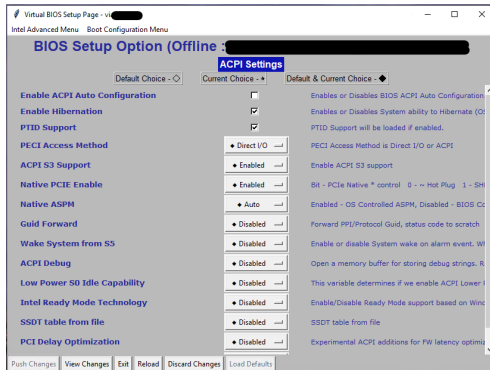
Menu to Select initial configuration for work

Setup Knobs Modification: Implementation Snaps II



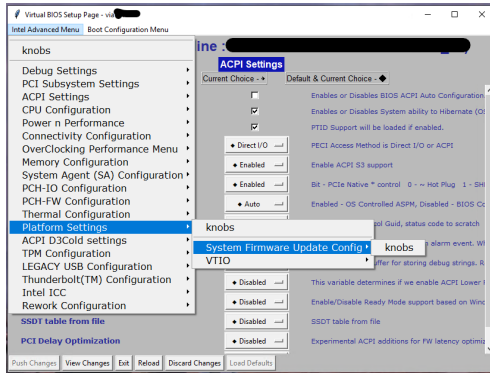
Available work mode for the system: Online and Offline

Setup Knobs Modification: Implementation Snaps III



Setup Options listed under ACPI Configurations

Setup Knobs Modification: Implementation Snaps IV



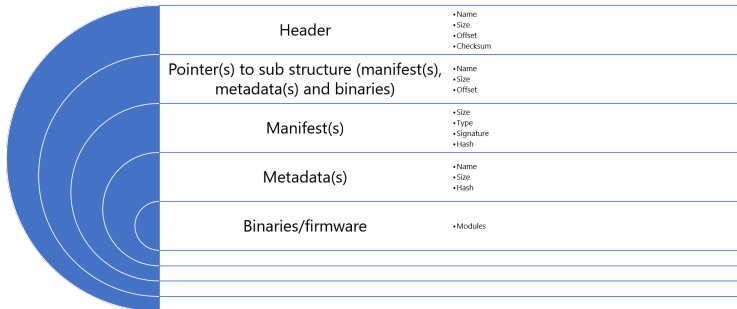
Navigating through BIOS setup page

Setup Knobs Modification: Outcome

- ▶ Cross platform usage
- ▶ API as a driver in BIOS Firmware
- ▶ Generic solution for usage types - on **SUT**, on **BIOS image**
- ▶ Information parsing and simulation
- ▶ Realtime sync for simulation changes
- ▶ Seamless Integration



IP firmware Integration: Structure of Module



Proposed Structure for firmware signing

IP firmware Integration: Outcome

- ▶ Removal of IP dependency during firmware loading
- ▶ IP Subsystem :
 - ▶ Loader and Verifier
 - ▶ IP is always consumer
- ▶ Signature verification using SHA hash algorithm
- ▶ Seamless Integration of any other hash algorithm for verification
- ▶ Hardware based and Software based verification support
- ▶ Prevent common security threats
- ▶ Allow easier OEM adoption and modification based on the respective design
- ▶ Reusability/Portability of design across many IPs
- ▶ Generic design which supports any new IP integration

- ▶ Study of existing hotspots for automation
- ▶ Analyzing and gathering requirements of automation to hotspot
- ▶ Implementing and managing platform to keep up-to-date the user base of the framework



Phase 1: Study of existing architecture for hotspot

- ▶ Identifying changes of two different BIOS Image of different check-ins
- ▶ Lookup of module by GUID and vice-versa
- ▶ Exposed source code support for OEM information
- ▶ Runtime BIOS Support for temporary UEFI variable creation

7. Phase 1: Study of existing architecture for hotspot

Phase 2: Analysis and gathering detailed information of the problem

- ▶ Debugging via comparing Setup Knobs
- ▶ Lookup of order of the module in BIOS Image as file system
- ▶ Verification of module integration via GUID
- ▶ OEM needs to fill information which need not to compulsorily expose the source code
- ▶ Automation and Testing for **non-BIOS driver** require BIOS Support for creation of temporary UEFI variables

8. Phase 2: Analysis and gathering detailed information of the problem

Requirement Specification

- ▶ Visual Studio C/C++
- ▶ Visual Studio Code
- ▶ Python
- ▶ JSON
- ▶ Memory Access Interfaces²

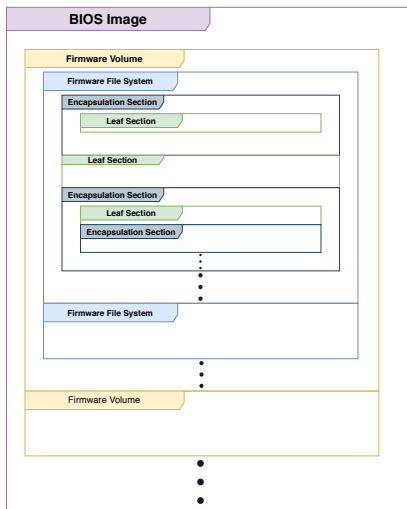
²itp, itp2, windows, linux, ltb, dci, efi shell, etc.

Why JSON?

- ▶ Light-weight structured-database!
- ▶ Minimal in terms of space complexity
- ▶ Easier to parse and interpret
- ▶ Portable

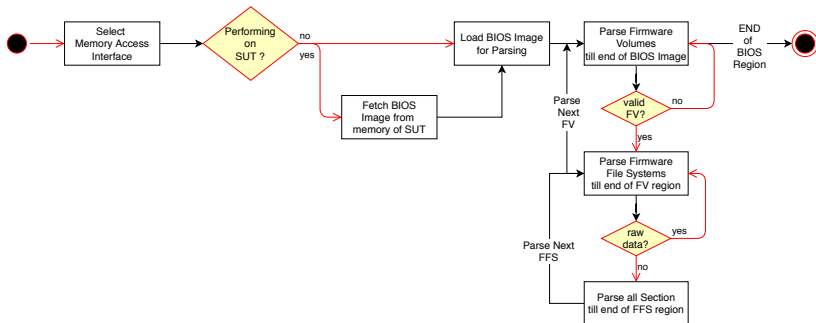


Implementation of Parsing: Format of BIOS Image



11. Implementation of Parsing

Implementation of Parsing: Work Flow



- ▶ Human Readable interpretation of BIOS Image
- ▶ GUIDs Lookup
- ▶ Verification of existence of module by GUID
- ▶ Storing the image file system content by GUID
- ▶ Summarizing changes of two BIOS image



Thank You

