Open Source Firmware Development: Reduce Firmware Support Package (FSP) boundary on Intel® SoC Platform

Author: Subrata Banik (subratabanik@google.com)

Introduction or Background

System Firmware Development on the latest SoC platforms has been limited to the proprietary firmware and Intel® SoC platform is not an exception. These proprietary firmwares are the essential slices for bringing silicon into life and defines the stability. Over the years, the state of closed-source blobs used for platform bring-up hasn't improved, in the sense of leading towards openness. Exhaustive Outline section highlights the IA-SoC based system firmware SPI flash layout (Figure 1-3) and Table 1-1 shows the challenges in the current approach.

TL;DR challenges with rapid growth in proprietary firmware from platform enabling standpoint are:

- Dependency on SoC vendors to own the bug and provide the fix, finally sharing the release cadence might not fit well with open-source project milestones.
- Limited engineering effort at silicon vendor side where else the broad open-source ecosystem is unable to contribute even if they have intention to contribute.
- One-binary-fits-all requirements makes it bloated and unnecessary feature inclusion without the particular product interest and might increase platform

security risk.

Objectives

This proposal is intended to discuss a path forward towards getting openness in system firmware development with Intel SoC (that primarily uses FSP blobs for Silicon and Platform Initialization).

Primary objectives

- Improve the platform enabling model better by balancing out the `binary blob model` (include if *mandatory* and can't be open-source) and focusing on bringing openness in silicon code by leveraging open-source boot firmware aka. coreboot.
- Reduce the boundaries of proprietary firmware running on Host CPU Firmware a.k.a Intel Firmware Support Package (FSP).

Secondary objectives

- Achieving the ambitious goal of fast booting (< 1 second boot time) up the system firmware.
- Drop unused FSP modules to reduce FSP boundaries to eventually optimize the SPI flash footprint.

Non goals

- This is moreover an initiative to overcome the problem that open-source firmware community is facing, additionally, hearing from ODM/OEM partners and representatives[1] working on Intel SoC platform etc.

Key features and/or Requirements

- Share a Platform Initialization (PI) vision that utilizes more openness.
- Consistently in the gen-over-gen SoC platform, be able to meet the system firmware boot time requirement of < 1 second.
- Define a fixed SPINOR size requirement that applies to all product guidelines.

Design ideas

This design document shares emerging ideas to solve this longstanding problem of proprietary firmware for Open Source Firmware (OSF) Development. This section is specifically written to meet the *primary objectives* as listed above. Additionally, the ideas are listed below in merit of exclusive study (on Intel platform, additionally, based on some comparative study on other SoC platforms) and in-house proof of concept

[1] Open Source Firmware Community is asking about Intel's commitment towards PI open-source https://mobile.twitter.com/_zaolin_/status/1497237365135491072

performed on the latest Intel platform.

Unless this design discussion materializes now, the current state of Open Source firmware development using proprietary firmware on IA SoC platforms won't improve in future.

An "Alternative Path" forward towards Open Source Firmware

This section highlights an alternative approach where SoC vendors are not committed towards open sourcing silicon reference code, and at times, it's critical for the open source community and products that derived out using OSF for their business commitment. Gaining more control over the platform initialization is the major theme for this proposal that empowers the open source firmware developers.

Due to lack in minimal FSP design guide, the roles and responsibility boundaries between FSP and bootloader (i.e. coreboot) are not very clear as FSP (PEI phase) is intended to do more things to supplement UEFI bootloader. For example: lock down configuration done as part of FSP, is ideally a primary bootloader's responsibility.

The Key Performance Indicator (KPI) defines critical criteria for product quality. FSP being used for production as silicon reference code doesn't have responsiveness KPI. As a result, while integrated with the bootloader it is difficult to debug any responsiveness issue in case final product responsiveness KPI is not met.

Additionally, any issue being fixed into the Intel FSP development trunk has several weeks latency (between 2-6 weeks in some cases) to make the fixed FSP externally available for consumption. Having only read access to FSP GitHub limits the external Intel FSP development community (i.e., open source firmware development engineers) to unable to continue into the code even the fix is known.

➢ Gain Control of Platform Initialization using native coreboot driver

This approach states the specific route that coreboot platform initialization implementation had explored in the past and would also like to explore, knowing the lack of open source silicon initialization commitment.

In the past with FSP 2.0 specification, coreboot decided to make FSP-T optional as it relies on open source cache-as-ram (CAR) implementation and updates periodically to add new SoC support.

This section highlights the short term and long term plan in this approach, along with few design assumptions:

\> Drop FSP APIs and use coreboot native driver with below characteristics:

- Chipset programming is documented as part of external datasheets (processor/pch).
- Implement only the Intel recommended chipset programming steps for ODM/OEMs, for example: Alder Lake FAS chapter 12 specifies the Security Configuration, ideally all host-firmware running on Alder Lake SoC should comply with this recommendation.
- Programming steps and outcome expectations are well captured in White Paper/BWG/FAS (non-confidential document).
- Only focus on IP programming that is applicable for targeted platforms (example: RAS, RST etc. are irrelevant for CrOS devices).

\> Few consideration while implementing chipset programming recommendation in coreboot:

- Design APIs based implementation based on the underlying IP.
- Implemented using a common code library and have hooks for SoC/mainboards(variants) to override if required. Ideally we would like to avoid overrides because it might make things harder while debugging. The idea is to have a more compatible design approach reflected in firmware and software, how it appears in hardware in general (like reusable IP across different SoC with up-rev IP version).

Figure 1-1 presents the current plan of action for implementing the gain control proposal in scope of coreboot, which eventually helps to reduce the dependency over FSP-APIs.

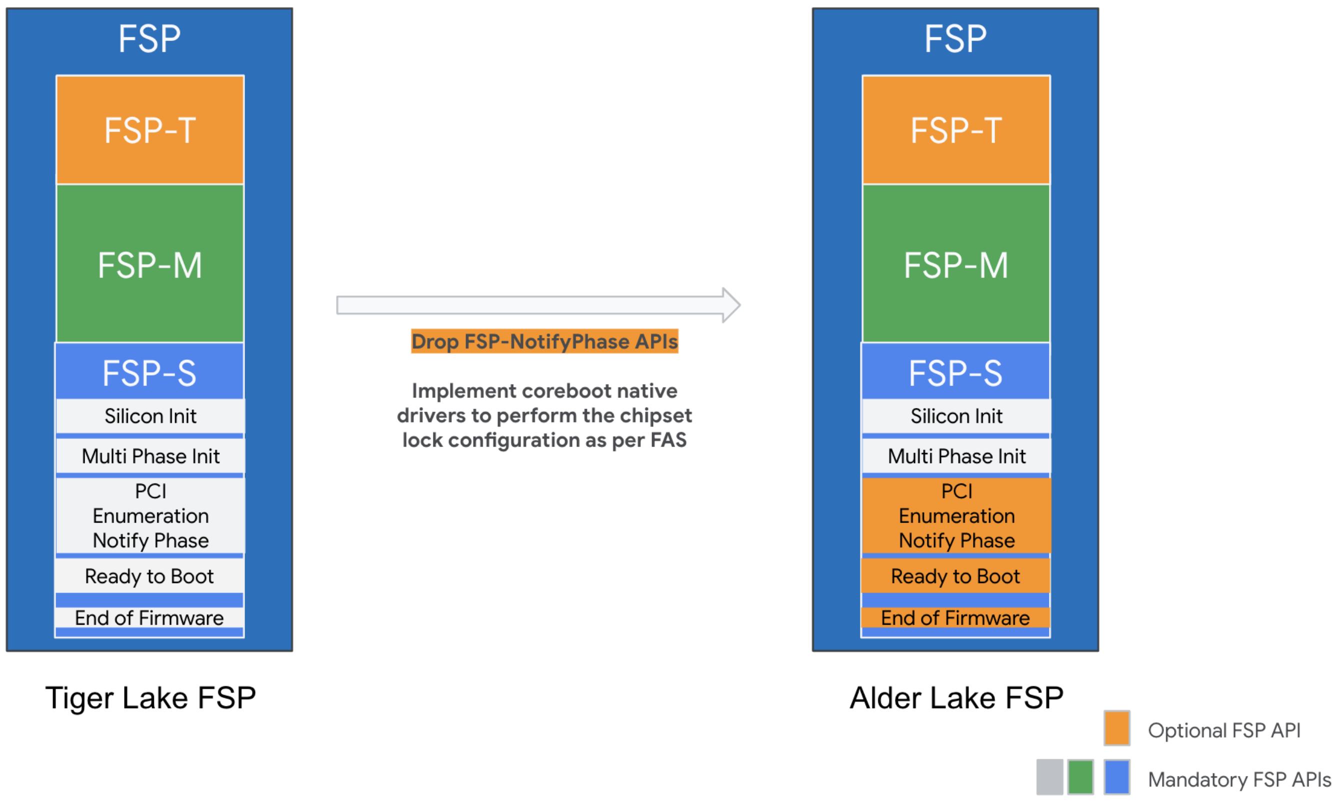

Figure 1-1. FSP footprint reduction between Tiger Lake to Alder Lake

Here are the benefits of this approach:

- Thinner FSP footprint with reduced FSP APIs reduce the baggage of proprietary firmware in Open Source System firmware stack.
- Feasible to open-source the majority of FSP code using coreboot native implementation.
- Improved flexibility while debugging and feature implementation without getting locked into Silicon program milestone and unable to support feature requests even with potential business reason[2].
- Efficient tooling can also help to reduce FSP binary size after removing unused FSP APIs (*example: Removal of FSP-S associate APIs would help to get any size savings by 3x times due to Chrome AP firmware SPI layout*).

Below code changes landed into upstream coreboot to get rid of FSP Notify Phase APIs [3] (as proposed in Figure 1-1) for Alder Lake SoC based platform.

[2] Need to compromise on Firmware Boot time due to Intel is unable to provide a bug fix on Alder Lake-P platform due to program milestone concerns https://review.coreboot.org/c/coreboot/+/61447

[3] Intel FSP 2.1 specification release email to coreboot mailing list highlights that FSP dispatch mode is also not using FSP-NotifyPhase APIs natively implemented in PEI phase. https://mail.coreboot.org/hyperkitty/list/coreboot@coreboot.org/thread/WCGVZABKXYYSWBSLORIKIHY4JE5VWCGM/

Description	**Code Changes**	**Comments**
Skip FSP-Notify Phase 1 APIs	https://review.coreboot.org/q/topic:DROP_FSP_NOTIFY1_API	FSP Notify Phase 1 API is designed to lock down chipset registers before executing 3rd party code during platform initialization.
Skip FSP-Notify Phase 2 APIs	https://review.coreboot.org/q/topic:b:211954778	FSP Notify Phase 2 APIs are a combination of two internal APIs to keep the hardware controllers into the specific mode designed for it and/or put into low-power mode prior to handing off to payload/OS.

Note: Reduction of FSP Notify Phase APIs is just a tiny step towards the right direction where the idea is to get rid of most of non-mandatory FSP APIs (FSP-Silicon Init aka. FSP-S) with native coreboot drivers. Figure 1-2 illustrates the proposed FSP-API view for future SoC platforms (which is possible to achieve with help of the open source community while co-working) that integrate with open-source coreboot boot firmware.

The proposed design solution and implementation is very much generic and can be applicable even for other SoC designs that inherits the FSP framework, for example: AMD latest SoC platform adopts FSP.

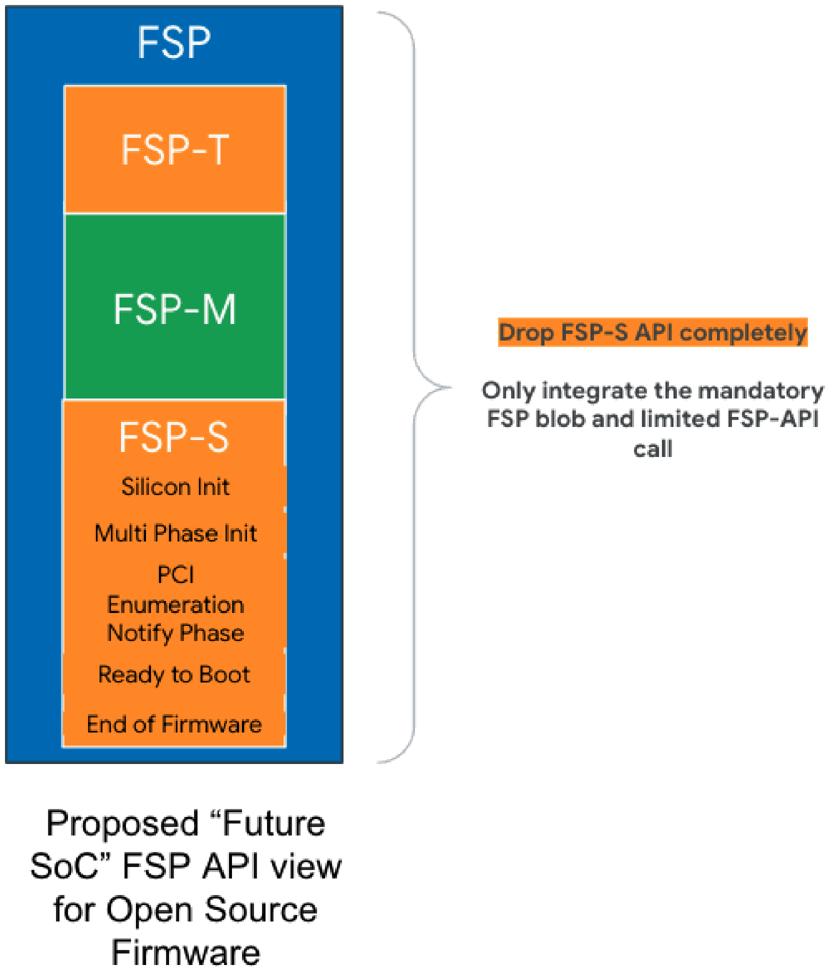
FSP
FSP-T
FSP-M
FSP-S
Silicon Init
Multi Phase Init
PCI Enumeration Notify Phase
Ready to Boot
End of Firmware
Drop FSP-S API completely
Only integrate the mandatory FSP blob and limited FSP-API call
Proposed "Future SoC" FSP API view for Open Source Firmware

Figure 1-2. Proposed FSP footprint with "*only*" mandatory FSP APIs for Intel SoC platform

Page Break

Exhaustive Outline

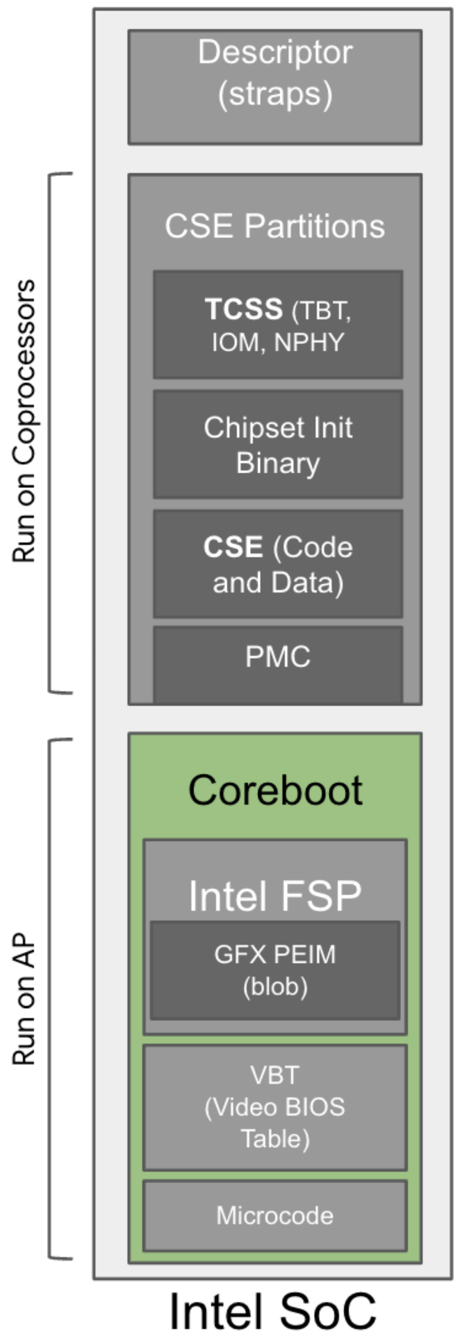

Figure 1-3. IA-SoC based System Firmware SPI Flash Layout

Firmware components used on IA SoC based design can typically divide into three categories as:

1. Proprietary/Closed Source Firmware running on Coprocessors:

All firmware components that reside outside the "*coreboot*" region belong to this category, for example: CSE, PMC, TCSS etc.

2. Proprietary/Closed Source Firmware running on Host CPU:

Firmware components are part of "*coreboot*" region and colored GRAY belong to this category, for example: Intel FSP and associated binaries into it.

3. Open Source Firmware running on Host CPU:

The only open source block in this entire Firmware stack is "*coreboot*" but unfortunately it relies on proprietary blobs ABI (Application Binary Interface) to perform the platform initialization (PI).

In the matrix of open-source readiness on system firmware with IA-SoC, the score is **< 50%**[4] (incorporating #1 and #2 from above, ~13MB/24MB SPI layout is occupied by closed-source firmware).

[4] Source: Brya ChromeOS.fmd
https://github.com/coreboot/coreboot/blob/master/src/mainboard/google/brya/chromeos.fmd

Table 1-1, describes the challenges due to proprietary firmware during platform enabling.

Security	Less Code Audit Opportunities and no traceability about incorporating security audit concerns back into the firmware. Current industry trend is about 1 defect/1kLOC, hence, unaudited closed source code is always a risk.
Platform Enabling	Platform enabling also demands SoC vendor support due to multiple binary blob dependencies even during bringup. Recent ODM summit Q&A session also brought such concern about the validation aspect of closed source binaries prior release externally.
Limited Open Initiatives	Lesser engagement from the open source community and ODM engineers due to not having required documentation access, code visibility etc. Community engineers also brought such concern, how to get more visibility into those required documents early for development and debugging.
Debugging	Debugging and bug fixes are difficult for closed source binary without having access to the source code. Limited debug functionality due to non-uniform debug, postcode, timestamp etc. libraries between open and closed source firmware components. For example: Intel FSP development is an unique scenario where FSP debug libraries have standalone implementation without leverage from the boot firmware, results into \`*cbmem -c*\` doesn't include FSP debug along with coreboot serial log, furthermore, \`*cbmem -t*\` doesn't include FSP module timestamp while debugging boot time issues.
Ungoverned Growth in FW blobs	Unaccountable growth in SPINOR size and boot time impact in Year over Year (YoY) SoC platforms. Between Skylake to Tiger Lake the FSP binary size has almost increased by 2x times[5].

[5] Intel FSP GitHub: A comparison between Fsp binary size between Skylake and latest SoC platform, Tiger Lake, https://github.com/intel/FSP