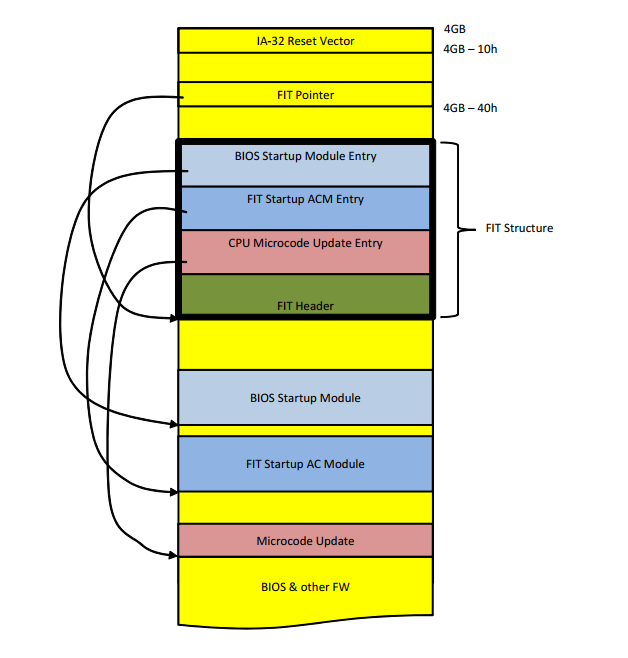
# Introduction

uCode is a boot critical firmware component that demands frequent update. This doc describes a resilient way to update uCode on Intel platform. With this method, uCode update become lightweight and secure. It will not impact other part of firmware. And customer won’t warry about bricking system.

The doc is composed of 5 major topics. It deals with how update information is carried and stored, which covers FMP capsule to carry uCode and Firmware image layout to store uCode. Following this, doc characterizes the update process with emphasis in NIST193 support. The last part discusses post-build scripts used in this solution.

# Firmware Interface Table

* Describe a data structure inside BIOS flash and consists of multiple entries
* Each entry defines the starting address and attributes of different components in the BIOS
* FIT resides in the BIOS Flash area and is located by a FIT pointer at physical address, 4GB - 40h
* This table is generated at build time, no modify for runtime
* The CPU processes the FIT before executing the first BIOS instruction located at the reset vector(0xFFFFFFF0)



# uCode region in System Firmware Image

Chasm Falls uCode update designs a **new firmware layout** tocarryuCode in the Firmware Device. Figure.6 below shows an example. Comparing with original design, new uCode Firmware Volume has 2 new features: 1) Version FFS to carry version info for entire uCode region. 2) uCode organized in Slot Mode to keep FIT content consistency. New uCode Firmware Volume comprise a Firmware Volume Header, a Version FFS file, and uCode FFS file. Version FFS carries global version info for entire uCode array, including 32-bit Version, 32-bit Lowest Supported Version, and a null-terminated Version String. Version FFS is followed by a PAD File to meet alignment request. Following PAD file is the uCode RAW FFS file to carry uCode array. All components follow Firmware File System described by PI spec

All the uCode patches are placed in slots, with 1KB alignment to comply with both early uCode patching and post-Reset Vector uCode patching. The uCode array is still described by PCD sets {PcdCpuMicrocodePatchAddress, PcdCpuMicrocodePatchRegionSize}.

The FIT table below the reset vector has uCode entries that point to the uCode slots in the uCode FFS File. **It is not allowed to change** when the uCode FV is updated using a UEFI Capsule.



Figure 1 uCode Component System Firmware Region Layout

Version FFS File Payload Format is

1. **typedef** **struct** {
2. **UINT32**  Version;
3. **UINT32**  LowestSupportedVersion;
4. CHAR16  VersionString[0];
5. } INTEL\_MICROCODE\_VERSION\_FFS\_DATA;

## Impacts of uCode Flash Layout Change

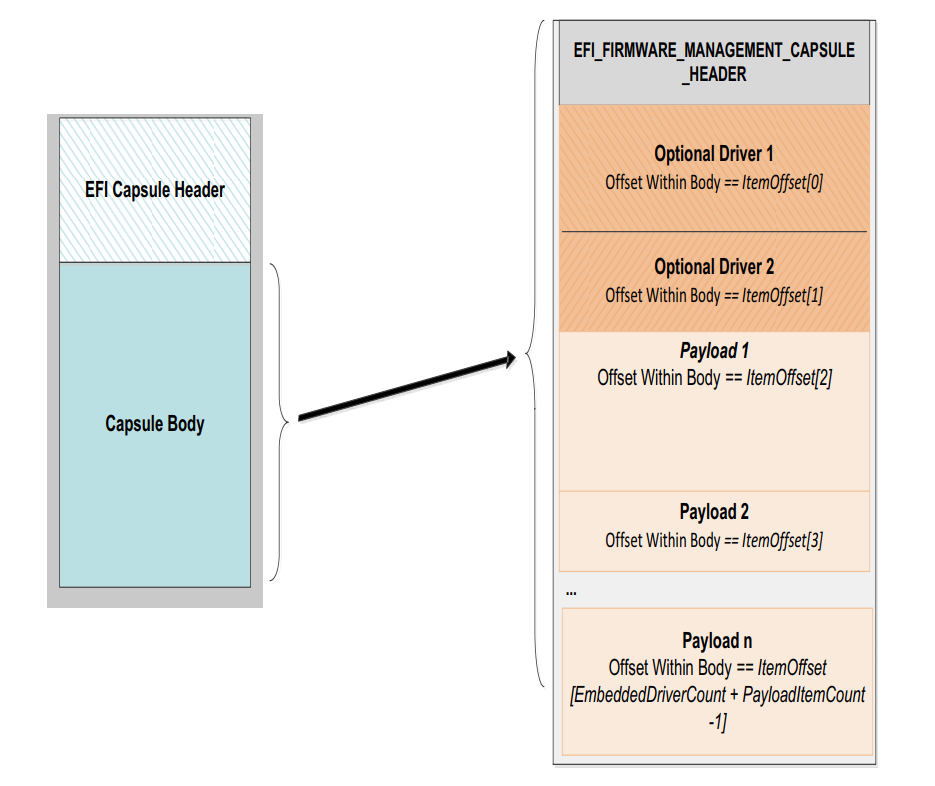
Everything in new uCode Flash Layout follows PI spec. FV, FFS and alignment are all standard firmware file system components. Flash layout has 4 consumers. 1) uCode early patching. 2) uCode Post-Reset Vector patching. 3) uCode FMP instance. 4) Firmware Binary Tools. There is no impact to the first 3 consumers. Because of the uCodes are still organized in array to be accessed by 1) and 2), the array is pointed by PCD settings. While inside array, uCode patches are still aligned at 1KB boundary. Besides that, uCode FMP instance is a new component designed to incorporate with this new layout. There could be some impact to 4th consumer, the Firmware Binary Tools. Some 3rd party tools like, FitGen, ACETool have hardcoded assumptions in allocating and parsing uCode FFS. The correct rule is to follow PI spec to parse firmware file system, skip Firmware Volume header, Version FFS, Pad FFS, allocate uCode FFS by standard File naming GUID {197DB236-F856-4924-90F8-CDF12FB875F3}. At last, retrieve each uCode entry at 1KB boundary in form of an array. Every 3rd party tools **shall follow this rule** to avoid the misconfiguration.

## Slot mode to avoid FIT update

Carried in BIOS region, uCode is also shared to other IFWI component though firmware interface table (FIT). Thus, consistency between FIT and uCode must be maintained.

In our design, FIT is **untouched** in update to make uCode an independent updatable resource. On the other hand, size and number of uCode could increase in future update. If all uCode are tightly packed, uCode information in FIT could be inconsistent after update. Slot mode technology is developed to address this problem. In slot mode, platform could pre-reserve some room in each uCode slot. Platform can also reserve extra empty slots for extension. The size and empty slot don’t break uCode scanning logic, because each uCode are still placed in an array and aligned at 1KB.

# UEFI Capsule Payload Format



### Capsule Header

1. ///
2. /// EFI Capsule Header.
3. ///
4. **typedef** **struct** {
5. ///
6. /// A GUID that defines the contents of a capsule.
7. ///
8. EFI\_GUID          CapsuleGuid;
9. ///
10. /// The size of the capsule header. This may be larger than the size of
11. /// the EFI\_CAPSULE\_HEADER since CapsuleGuid may imply
12. /// extended header entries
13. ///
14. **UINT32**            HeaderSize;
15. ///
16. /// Bit-mapped list describing the capsule attributes. The Flag values
17. /// of 0x0000 - 0xFFFF are defined by CapsuleGuid. Flag values
18. /// of 0x10000 - 0xFFFFFFFF are defined by this specification
19. ///
20. **UINT32**            Flags;
21. ///
22. /// Size in bytes of the capsule.
23. ///
24. **UINT32**            CapsuleImageSize;
25. } EFI\_CAPSULE\_HEADER;

### Capsule Body

This part will be processed by ProcessFmpCapsuleImage(DxeCapsuleLib.c) function. Call the function StartFmpImage to start all the drivers within capsule. Process all payloads with SetFmpImageData function in DxeCapsuleLib.c file.

#### EFI\_FIRMWARE\_MANAGEMENT\_CAPSULE\_HEADER

1. **typedef** **struct** {
2. **UINT32** Version;
4. ///
5. /// The number of drivers included in the capsule and the number of corresponding
6. /// offsets stored in ItemOffsetList array.
7. ///
8. UINT16 EmbeddedDriverCount;
10. ///
11. /// The number of payload items included in the capsule and the number of
12. /// corresponding offsets stored in the ItemOffsetList array.
13. ///
14. UINT16 PayloadItemCount;
16. ///
17. /// Variable length array of dimension [EmbeddedDriverCount + PayloadItemCount]
18. /// containing offsets of each of the drivers and payload items contained within the capsule
19. ///
20. // UINT64 ItemOffsetList[];
21. } EFI\_FIRMWARE\_MANAGEMENT\_CAPSULE\_HEADER;

#### Optional Driver

#### Payload (Processed in SetFmpImageData of DxeCapsuleLib.c)

Payload part processed in SetFmpImageData function of DxeCapsuleLib.c file. Locate protocol gEfifirmwareManagementProtocolGuid, call Fmp->SetImage enter SetTheImage function of FmpDxe driver. The Binary Update Image part include uCode patches content.



The Real Content part is described in “uCode Capsule Payload Format” section. This part is specified related on different platforms, for ChasmFalls Gen2 solution, we define this part as a Firmware Volume.

# uCode Capsule Payload Format

A standard UEFI Capsule format is used to distribute a uCode update to a target system. The portion of a UEFI Capsule that is customized for each update type is the Payload. A uCode update may contain one or more uCode patches. One goal is to use the same Payload layout for platforms that enable or disable the use of BIOS Guard. BIOS Guard updates require an additional signed BIOS Guard script that is linked to the uCode patches in the UEFI Capsule. This means there are at least two variable sized elements in the Payload. One for uCode patches and an optional element for the BIOS Guard script. One format that is simple to generate and parse is the Variable-Length Opaque Data format from *RFC 4506 External Data Representation Standard (XDR)*.

When BIOS Guard is enabled, the update image and BiosGuard update script must be fixed and signed at build time. The buffer that is updated by BIOS Guard cannot be modified before it is passed to the BIOS Guard interface to update the system firmware storage device. This means the layout of the uCode patches in the UEFI Capsule must be identical to the format in the system firmware image. In the previous section, a Firmware Volume layout is specified for the uCode region in the system firmware image.

The figure below shows the UEFI Capsule layout with a uCode Payload. The Payload is in XDR format and contains the uCode Firmware Volume image followed by an optional BIOS Guard script. The BIOS Guard is considered valid if it is present and has a non-zero size. The uCode Firmware Volume is identical in format to the uCode region in the system firmware image. If target system runs with FIT, all uCodes carried in uCode FV in capsule must be aligned at proper offset to make sure FIT setting are still consistent after update.



# NIST 800-193 Considerations

## Protect

The uCode region of the system FLASH must be protected so it is only updatable during a UEFI Capsule update of the uCode region. First, the uCode capsule is authenticated the same way as other UEFI Capsules. Second, by default, this region of the system FLASH must be locked to prevent writes. If BIOS Guard is enabled, then BIOS Guard protects the uCode region. If BIOS Guard is not enabled, the uCode region of the system firmware device must be locked as early as possible in the boot, and no later than the End Of DXE. The End of DXE lock point is only allowed when the boot mode is BOOT\_ON\_FLASH\_UPDATE.

## Detect

A method of detecting if all uCode patches are valid is required. uCode patches use a standard format that includes a 32-bit checksum. A uCode patch that has become corrupted will be rejected by the CPU, so there is no risk of a corrupted uCode patch being used. When patch at reset is used, a corrupted uCode patch that is rejected by the CPU may trigger recovery where the uCode patches from the recovery image are used. Not all uCode patches may be used every boot, so both normal boot and recovery boot must verify the checksum of all uCode slots in both the primary and recovery images. If any uCode patch fails its checksum, then recovery must be triggered to restore the corrupted uCode patch. The Firmware Volume Header and each FFS File also support checksum fields that can be verified to detect corruption.

**NOTE**: For GEN2, only recovery from an interrupted update is required. The discussion above is to make sure the design of the uCode storage format in GEN2 supports detection of corruption for all data in the uCode region.

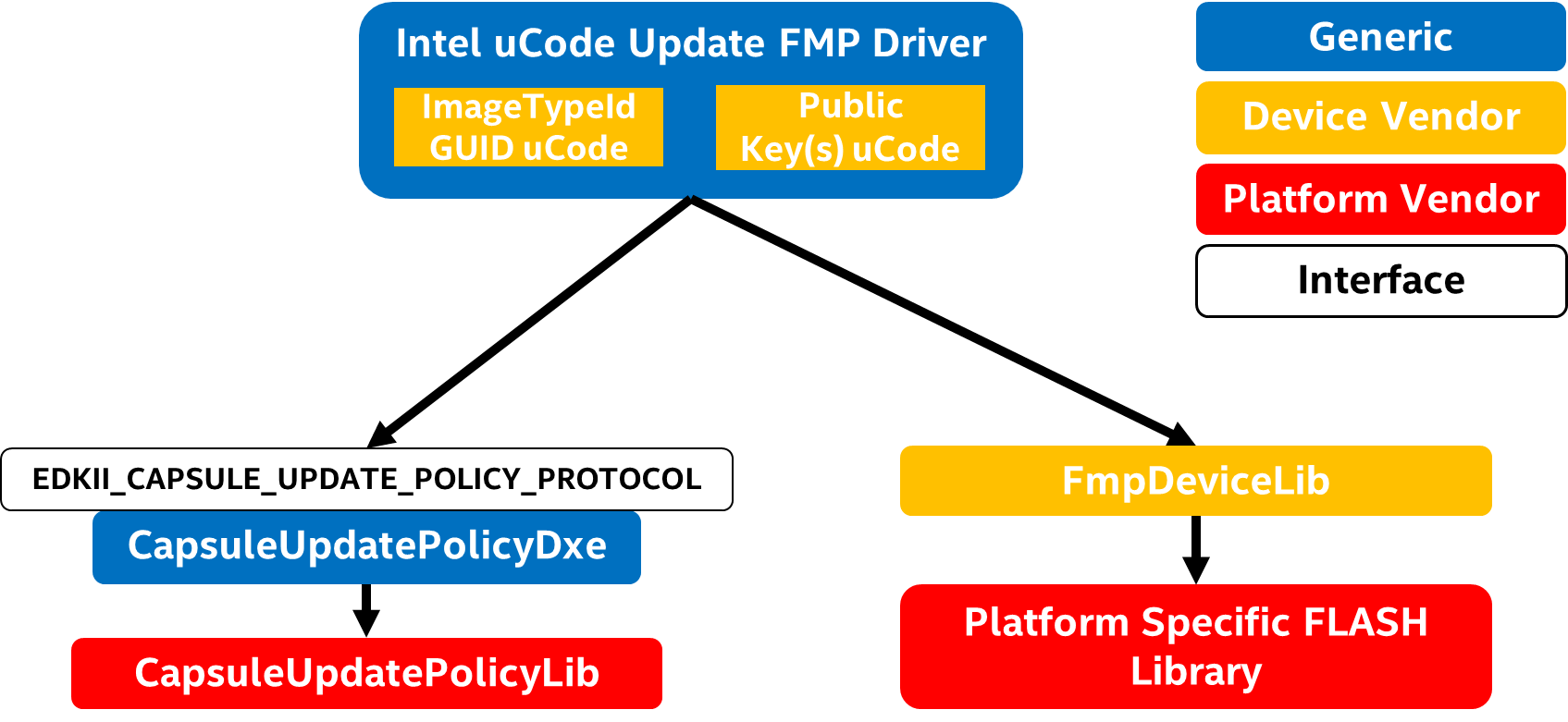
## Recovery

uCode is stored in a region of system firmware that is covered by the Top Swap feature, so the primary and recovery images each have their own copy of uCode patches. When a UEFI Capsule update is used to update the uCode, TopSwap ensures there is at least one good image (either primary or recovery) can be used for recovery.

# uCode FmpDxe Module

The figure below shows the block diagram for the **FmpDxe** module for uCode updates. This design takes advantage of the Capsule Update Policy Protocol to remove the static link of the platform specific **CapsuleUpdatePolicyLib** from the **FmpDxe** module. The component that need to be implemented is the **FmpDeviceLib** for uCode updates, and that library may require services from a platform specific library to update the uCode FV image in the system firmware storage device. This **FmpDeviceLib** should support both BIOS Guard enabled and BISO Guard disabled configurations.

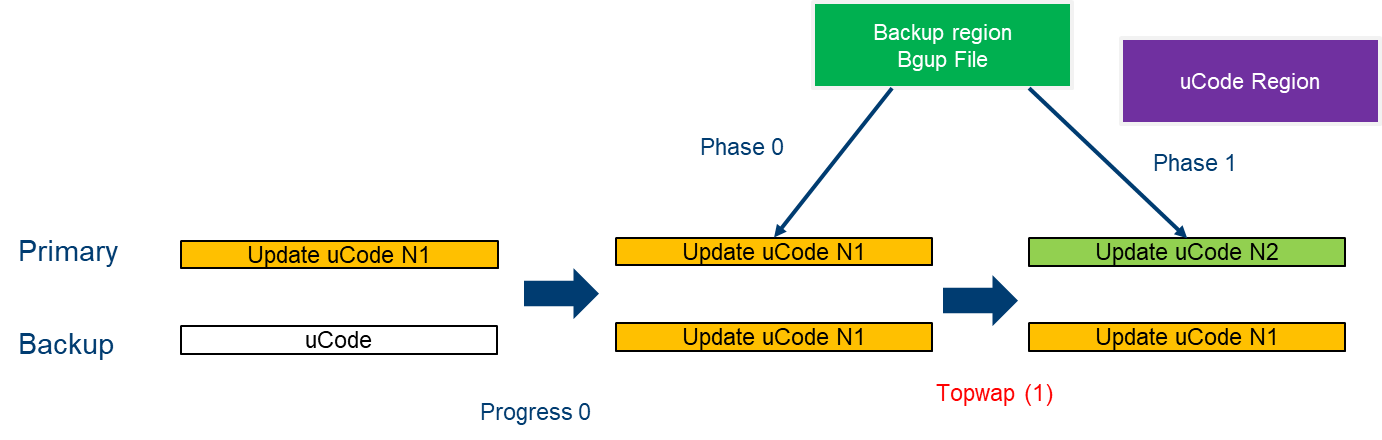
uCode Component Update FMP DXE Module



## Primary and Recovery Image Update with TopSwap

The figure below shows the sequence of updates to the Primary and Recovery images so that platform can always recover from interrupted update. This example shows the BIOS Guard enabled sequence, but the same sequence applies when BIOS Guard is disabled. The first step is to update the Recovery image to make sure it contains the same uCode FV contents that are in the Primary Image. Once that step is completed and verified, the uCode region in the Primary Image is updated with the new uCode FV contents. Once that step is completed and verified, the update is complete, and the next boot of the Primary Image uses the new uCode patches. If recovery is triggered after the update, then the known good uCode region from the Recovery Image is used.

This sequence is also resilient to interrupted updates. TopSwap setting is always configured to a good uCode image. If the update to the Recovery Image is interrupted, then the system will boot from the known good Primary Image and can continue or restart the update to the Recovery Image. If the update of the Primary Image is interrupted, then the system will automatically boot into the Recovery Image and the update of the Primary Image can be continued or restarted.



# Building Scripts for uCode Capsule

Post build scripts are developed to support Slot Mode and Version FFS generation. The following flow chat is used to illustrate Capsule building process.

usage: python NewGenCap.py --mode/-m <ucodebgup | ucodefull | ucodeslot>

[--fw-version <UINT32>] [--lsv <UINT32>]

[--fw-version-string <CHAR16 String>]



The microcode\_padding.py script is used to generating “Version.FFS” and do padding data for each uCode patches which is defined in [FV.FvMicrocode] of BoardPkg.fdf. This script already integrated in building process. For generates “Version.FFS” binary content, building tool also defined some default value in prep.bat script. Such as on TGL platform, SLOT\_SIZE=0x1C000, FW\_VERSION=0x0001, LSV=0x0001, FW\_VERSION\_STRING=”Version 0.0.0.1”. The platform owner could change these environment variables before building image. When change the SLOT\_SIZE variable, please make sure the value is at least equal or greater than the size of each uCode patches.

The NewGenCap.py tool generates the UEFI Capsule for uCode update. This script tool reuses the microcode\_padding.py script to generates “Version.FFS” and padding data for uCode patches. Note that, please run “prep.bat c” script before using this script.

## Microcode FV FDF File

The following FDF file is used to generating Firmware Volume for UEFI Capsule. The Firmware Volume size should be the same as uCode region on target platform. The BlockSize and NumBlocks are depends on gSiPkgTokenSpaceGuid.PcdFlashMicrocodeFvSize in FlashMapInclude.fdf file. MicrocodeVersion.data carries global version info to manage entire microcode region. If set the starting address of uCode patches at 4KB alignment, the PcdFlashMicrocodeOffset must also set ti 4KB accordingly. UEFI Capsule could carries more than one uCode patches by adding uCode patch in Microcode FFS.

[FV.MICROCODE\_FV]

BlockSize = 0x10000

NumBlocks = 0x9

#

# Microcode Version FFS with Version, LSV and Version String.

# Raw data file was created in pre-build step

#

FILE RAW = $(gIntelMicrocodeVersionFfsFileGuid) {

TigerLakeBoardPkg\Binaries\Microcode\MicrocodeVersion.data

}

#

# Microcode FFS file with aligned microcode patches

#

FILE RAW = $(gIntelMicrocodeArrayFfsFileGuid) {

Align=4K TigerLakeBoardPkg\Binaries\Microcode\m\_80\_806c0\_0000005a.pdb

Align=XX ...\xxx.pdb

}

# Related Definitions

gIntelMicrocodeVersionFfsFileGuid = {0xE0562501, 0xB41B, 0x4566, {0xAC, 0x0F, 0x7E, 0xA8, 0xEE, 0x81, 0x7F, 0x20}}

gIntelMicrocodeArrayFfsFileGuid = {0x197DB236, 0xF856, 0x4924, {0x90, 0xF8, 0xCD, 0xF1, 0x2F, 0xB8, 0x75, 0xF3}}