Live Firmware Activation specification

Document number	DEN0147
Document quality	BET
Document version	1.0-bet0
Document confidentiality	Non-confidential

Quality Status: Beta (BET)
Beta quality status has a particular meaning to Arm of which the recipient must be aware. At this quality level the release is sufficiently stable and committed for initial product development. The recipient can expect some changes to the Beta quality released material.

Release information

Date	Version	Changes
2025/Mar/31	1.0BET0	• Update FIDs with official allocation. • Added call_again flag to LFA_ACTIVATE. • Added Device Tree enumeration method.
2024/Dec/10	1.0ALP4	• Added cpu_rendezvous_optional return flag to LFA_GET_INVENTORY. • Added skip_cpu_rendezvous caller flag to LFA_ACTIVATE. • Move no_cpu_reset_needed return flag from LFA_PRIME to LFA_GET_INVENTORY as may_reset_cpu (polarity inverted).
2024/Jul/31	1.0ALP3	• Change to rules-based writing style. • Remove version value from LFA_GET_INVENTORY
2024/Apr/30	1.0ALP2	• Add support for CPU Reset during activation.
2024/Mar/29	1.0ALP1	• Initial release of the specification. • Restrict to implicit CPU Rendezvous. • Terminology changes.
2024/Feb/01	1.0ALP0	• Migration from DEN0118. • Initial internal release.

Arm Non-Confidential Document License ("License")

This License is a legal agreement between you and Arm Limited ("**Arm**") for the use of Arm's intellectual property (including, without limitation, any copyright) embodied in the document accompanying this License ("**Document**"). Arm licenses its intellectual property in the Document to you on condition that you agree to the terms of this License. By using or copying the Document you indicate that you agree to be bound by the terms of this License.

"**Subsidiary**" means any company the majority of whose voting shares is now or hereafter owned or controlled, directly or indirectly, by you. A company shall be a Subsidiary only for the period during which such control exists.

This Document is **NON-CONFIDENTIAL** and any use by you and your Subsidiaries ("Licensee") is subject to the terms of this License between you and Arm.

Subject to the terms and conditions of this License, Arm hereby grants to Licensee under the intellectual property in the Document owned or controlled by Arm, a non-exclusive, non-transferable, non-sub-licensable, royalty-free, worldwide License to:

(i) use and copy the Document for the purpose of designing and having designed products that comply with the Document;

(ii) manufacture and have manufactured products which have been created under the License granted in (i) above; and

(iii) sell, supply and distribute products which have been created under the License granted in (i) above.

Licensee hereby agrees that the Licenses granted above shall not extend to any portion or function of a product that is not itself compliant with part of the Document.

Except as expressly licensed above, Licensee acquires no right, title or interest in any Arm technology or any intellectual property embodied therein.

The content of this document is informational only. Any solutions presented herein are subject to changing conditions, information, scope, and data. This document was produced using reasonable efforts based on information available as of the date of issue of this document. The scope of information in this document may exceed that which Arm is required to provide, and such additional information is merely intended to further assist the recipient and does not represent Arm's view of the scope of its obligations. You acknowledge and agree that you possess the necessary expertise in system security and functional safety and that you shall be solely responsible for compliance with all legal, regulatory, safety and security related requirements concerning your products, notwithstanding any information or support that may be provided by Arm herein. In addition, you are responsible for any applications which are used in conjunction with any Arm technology described in this document, and to minimize risks, adequate design and operating safeguards should be provided for by you.

Reference by Arm to any third party's products or services within this document is not an express or implied approval or endorsement of the use thereof.

THE DOCUMENT IS PROVIDED "AS IS". ARM PROVIDES NO REPRESENTATIONS AND NO WARRANTIES, EXPRESS, IMPLIED OR STATUTORY, INCLUDING, WITHOUT LIMITATION, THE IMPLIED WARRANTIES OF MERCHANTABILITY, SATISFACTORY QUALITY, NON-INFRINGEMENT OR FITNESS FOR A PARTICULAR PURPOSE WITH RESPECT TO THE DOCUMENT. Arm may make changes to the Document at any time and without notice. For the avoidance of doubt, Arm makes no representation with respect to, and has undertaken no analysis to identify or understand the scope and content of, third party patents, copyrights, trade secrets, or other rights.

NOTWITHSTANDING ANYTHING TO THE CONTRARY CONTAINED IN THIS LICENSE, TO THE FULLEST EXTENT PERMITTED BY LAW, IN NO EVENT WILL ARM BE LIABLE FOR ANY DAMAGES, IN CONTRACT, TORT OR OTHERWISE, IN CONNECTION WITH THE SUBJECT MATTER OF THIS LICENSE (INCLUDING WITHOUT LIMITATION) (I) LICENSEE'S USE OF THE DOCUMENT; AND (II) THE IMPLEMENTATION OF THE DOCUMENT IN ANY PRODUCT CREATED BY LICENSEE UNDER THIS LICENSE). THE EXISTENCE OF MORE THAN ONE CLAIM OR SUIT WILL NOT ENLARGE OR EXTEND THE LIMIT. LICENSEE RELEASES ARM FROM ALL OBLIGATIONS, LIABILITY, CLAIMS OR DEMANDS IN EXCESS OF THIS LIMITATION.

This License shall remain in force until terminated by Licensee or by Arm. Without prejudice to any of its other rights, if Licensee is in breach of any of the terms and conditions of this License then Arm may terminate this License immediately upon giving written notice to Licensee. Licensee may terminate this License at any time. Upon termination of this License by Licensee or by Arm, Licensee shall stop using the Document and destroy all copies of the Document in its possession. Upon termination of this License, all terms shall survive except for the License grants.

Arm Limited. Company 02557590 registered in England.

110 Fulbourn Road, Cambridge, England CB1 9NJ.

Arm document reference: PRE-21585

version 5.0, March 2024

Contents
Live Firmware Activation specification

Contents

Conventions

Typographical conventions

The typographical conventions are:

italic

Introduces special terminology, and denotes citations.

bold

Denotes signal names, and is used for terms in descriptive lists, where appropriate.

`monospace`

Used for assembler syntax descriptions, pseudocode, and source code examples.

Also used in the main text for instruction mnemonics and for references to other items appearing in assembler syntax descriptions, pseudocode, and source code examples.

SMALL CAPITALS

Used for some common terms such as IMPLEMENTATION DEFINED.

Used for a few terms that have specific technical meanings, and are included in the Glossary.

Red text

Indicates an open issue.

Blue text

Indicates a link. This can be

- A cross-reference to another location within the document
- A URL, for example http://developer.arm.com

Numbers

Numbers are normally written in decimal. Binary numbers are preceded by `0b`, and hexadecimal numbers by `0x`. In both cases, the prefix and the associated value are written in a monospace font, for example `0xFFFF0000`. To improve readability, long numbers can be written with an underscore separator between every four characters, for example `0xFFFF_0000_0000_0000`. Ignore any underscores when interpreting the value of a number.

Rules-based writing

This specification consists of a set of individual *content items*. A content item is classified as one of the following:

- Declaration
- Rule
- Goal
- Information
- Rationale
- Implementation note
- Software usage

Declarations and Rules are normative statements. An implementation that is compliant with this specification must conform to all Declarations and Rules in this specification that apply to that implementation.

Declarations and Rules must not be read in isolation. Where a particular feature is specified by multiple Declarations and Rules, these are generally grouped into sections and subsections that provide context. Where appropriate, these sections begin with a short introduction.

Arm strongly recommends that implementers read *all* chapters and sections of this document to ensure that an implementation is compliant.

Content items other than Declarations and Rules are informative statements. These are provided as an aid to understanding this specification.

Content item identifiers

A content item may have an associated identifier which is unique among content items in this specification.

After this specification reaches beta status, a given content item has the same identifier across subsequent versions of the specification.

Content item rendering

In this document, a content item is rendered with a token of the following format in the left margin: L_{iiiii}

- *L* is a label that indicates the content class of the content item.
- *iiiii* is the identifier of the content item.

Content item classes

Declaration

A Declaration is a statement that does one or more of the following:

- Introduces a concept
- Introduces a term
- Describes the structure of data
- Describes the encoding of data

A Declaration does not describe behaviour.

A Declaration is rendered with the label *D*.

Rule

A Rule is a statement that describes the behaviour of a compliant implementation.

A Rule explains what happens in a particular situation.

A Rule does not define concepts or terminology.

A Rule is rendered with the label *R*.

Goal

A Goal is a statement about the purpose of a set of rules.

A Goal explains why a particular feature has been included in the specification.

A Goal is comparable to a “business requirement” or an “emergent property.”

A Goal is intended to be upheld by the logical conjunction of a set of rules.

A Goal is rendered with the label *G*.

Information

An Information statement provides information and guidance as an aid to understanding the specification.

An Information statement is rendered with the label *I*.

Rationale

A Rationale statement explains why the specification was specified in the way it was.

A Rationale statement is rendered with the label *X*.

Implementation note

An Implementation note provides guidance on implementation of the specification.

An Implementation note is rendered with the label *U*.

Software usage

A Software usage statement provides guidance on how software can make use of the features defined by the specification.

A Software usage statement is rendered with the label *S*.

References

This section lists publications by Arm and by third parties.

See Arm Developer (http://developer.arm.com) for access to Arm documentation.

[1] *Platform Security Firmware Update for the A-profile Arm Architecture*. (ARM DEN 0118 1) Arm.

[2] *SMC calling convention*. (ARM DEN 0028 1.2) Arm.

[3] *Arm Power State Coordination Interface*. (ARM DEN 0022 D.b) Arm.

Feedback

Arm welcomes feedback on its documentation.

Feedback on this book

If you have any comments or suggestions for additions and improvements create a ticket at https://support.developer.arm.com/. As part of the ticket include:

- The title (Live Firmware Activation specification).
- The number (DEN0147 1.0-bet0).
- The section name to which your comments refer.
- The page number(s) to which your comments apply.
- A concise explanation of your comments.

Arm also welcomes general suggestions for additions and improvements.

Inclusive terminology commitment

Arm values inclusive communities. Arm recognizes that we and our industry have used terms that can be offensive. Arm strives to lead the industry and create change. We believe that this document contains no offensive terms. If you find offensive terms in this document, please contact terms@arm.com.

Glossary

Activation Client

The entity in the non-secure state that calls the Live Activation ABI

BMC

Baseboard Management Controller

CPU rendezvous

Synchronizes all CPUs into a state compatible with Live Firmware Activation.

CPU reset

Architectural reset of all AP cores performed during Live Firmware Activation

FW

Firmware

Host Software

The host Operating System or Hypervisor running on the platform.

LFA

Live Firmware Activation

Live Activation Store

The abstract container for firmware images on which Live Activation can be performed. This part can be integrated with the Platform Firmware Store.

Live Firmware Activation

The procedure of activating a firmware image instance, replacing a previously running instance, while the system remains in execution.

Live Firmware Activation Agent

The entity responsible for performing Live Firmware Activation.

LSB

Least Significant Byte

MBZ

Must be zero.

Platform Firmware Store

The non-volatile memory containing the firmware images that are executed on the platform.

UUID

Universally Unique Identifier

Introduction

Live firmware activation is a technology that enables the activation of platform firmware in real-time, without the need for a system reboot. This approach reduces the impact of firmware updates or system configuration changes on service availability. Service availability is a key metric in deployments including data centers and cloud providers. By minimizing disruptions, this approach helps service providers maintain their Service Level Agreements (SLAs).

In some cases, the Host Software cannot recognize new hardware or configuration changes without a reboot, meaning that the new live firmware activation method might not address enabling of new features or hardware immediately.

Live firmware activation addresses various firmware maintenance challenges and supports use cases including the following:

1. **Security Updates:** In response to a newly discovered security vulnerability, a data center needs to update the firmware of its servers immediately. With live firmware activation, the provider can apply the security patch across all servers without any service interruption.
2. **Performance Optimization:** A data center wants to optimize the performance of its servers by updating the firmware to a version that offers better resource management. Using live firmware activation, the data center can implement the updates without any downtime, ensuring that the services running on these servers continue to operate at optimal levels.
3. **Hardware Compatibility:** A data center installs a new peripheral device firmware in its infrastructure and needs to update the firmware of its existing servers to ensure compatibility. With live firmware activation, the provider can update the firmware in real-time, allowing the new device firmware to integrate seamlessly into the existing infrastructure without any service disruptions.

In these scenarios, live firmware activation proves to be an important tool that enhances the efficiency, reliability, and security of data centers and cloud environments.

Chapter 1
Firmware activation concepts

Firmware components start executing once activated. The firmware components are activated at:

1. cold boot: during the regular platform boot, and
2. system runtime: by the runtime Live Firmware Activation Agent (LFA Agent), via a flow termed live firmware activation.

This section specifies a live firmware activation design and the supporting firmware activation ABI (Chapter 2 *Firmware activation ABI*) implemented by the LFA Agent. The ABI enables a caller to interact with the LFA Agent to:

- discover details of the platform firmware components.
- discover firmware component updates available for live activation.
- trigger a firmware activation procedure.

The activation ABI calls are routed by the LFA Agent at EL3. Depending on the platform Security Model, the LFA Agent can delegate aspects of activation to lower ELs or other system components, e.g. security processor. The LFA Agent is responsible for coordinating the activation of the different firmware components.

Aside from the firmware activation ABI implemented by the LFA Agent, the remaining aspects of the LFA implementation and its delegation logic are IMPLEMENTATION DEFINED.

The diagram in Figure 1.1 shows an example system design with a LFA Agent implemented at EL3. In this example, the LFA Agent has permanent read access to the Firmware Store. The nature of the Firmware Store is IMPLEMENTATION DEFINED. The Firmware store can be flash owned by a security processor, a Secure partition (e.g. the Update Agent as defined in the Firmware Update for A-profile specification [1]), or even an ephemeral Store implemented as a shared buffer with Non-Secure. The mechanism by which the LFA Agent reads from the Firmware Store is IMPLEMENTATION DEFINED.

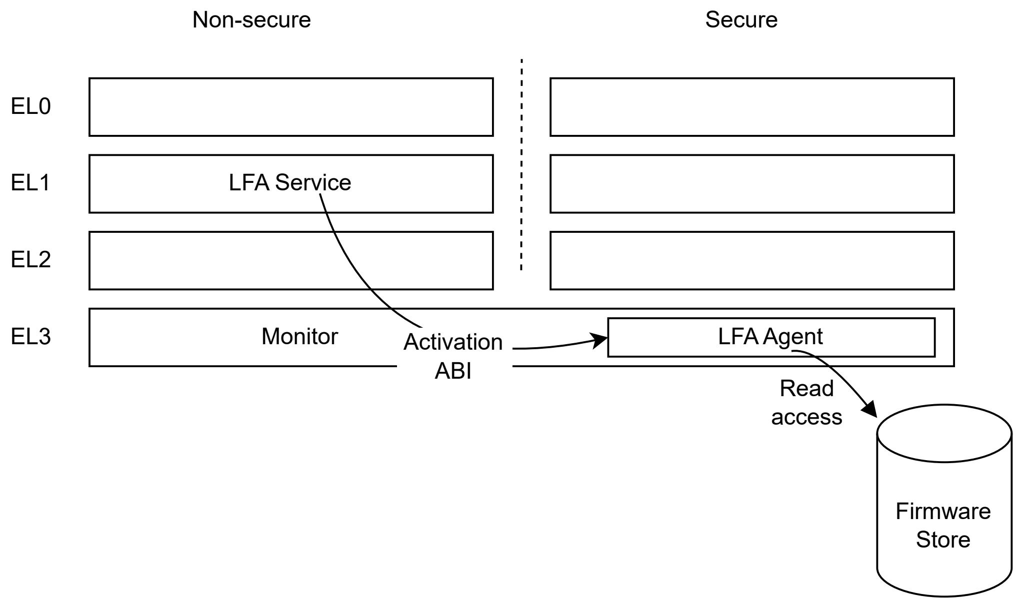

Figure 1.1: Example system diagram for activation

The LFA Agent is allowed to delegate the activation procedure. For instance, when the component being activated resides in S-EL1, the component residing in S-EL2 can carry that activation procedure.

Any firmware component that is live activated must ensure that the Host Software, in the Non-secure Security State, does not need to perform any firmware-specific discovery following a firmware live activation. That means that, from the point of view of the Host Software, all primitives and services, that an old instance of a firmware component exposed to the Host Software, must be present in the new instance of that component, and behave in an equivalent manner.

The activation of firmware involves different steps, such as:

- loading images from the Live Activation Store onto the location they will execute from,
- authenticating/measuring these images,
- initializing, or reinitializing, the firmware components.

For all those activities, platform firmware requires CPU cycles.

The Host Software is commonly in control of providing cycles for the different firmware activities. The live firmware activation ABI primarily serves as a means for the Host Software to provide cycles for the live firmware activation to take place.

The live firmware activation procedure is broken up into 2 distinct phases:

1. Prime: Preparatory phase where any long-running operations, that do not impact component integrity, can be performed without reducing system availability.
2. Activation: Phase where components are reactivated. Any components being activated will be unavailable during this period.

During the Prime phase, the platform prepares the different components for activation. It is guaranteed that any firmware components remain in operation and able to service any requests. The Prime phase is a single-threaded operation but not pinned on a single physical CPU. The actual component activation happens in the Activation phase.

During the Activation phase the components that will undergo activation, and any dependents, will be unable to service requests from clients or interrupts.

1.1 Firmware activation schedule

The LFA Agent informs the Host Software about the coarse schedule required to successfully complete the firmware activation procedure.

The Host Software provides cycles to the LFA Agent by explicitly calling the following ABI primitives:

- LFA_PRIME: called during the Prime phase. Multiple calls can be required.
- LFA_ACTIVATE: called during the Activation phase by one or all active CPUs.

The activation schedule can only be called for one component at a time.

1.1.1 Live Firmware Activation Agent responsibilities

The commitment by the LFA to start an activation process requires a set of preconditions to be met. Some of those pre-conditions will be checked during the LFA_PRIME phase.

R_{MNDDX} The LFA_ACTIVATE phase can rely on a co-ordinated initialization of the firmware components on all CPUs. In order for firmware component initialisation to be correctly performed, LFA implementation must ensure that the set of active CPUs remain constant from the start and until the end of this phase.

1.1.2 CPU Rendezvous

In some cases of live firmware activation, the CPU Rendezvous process is required to ensure that firmware services are not being called by the Host OS or triggered by interrupts during the firmware update and activation. This guarantees that the firmware update can proceed without interference or conflicts, maintaining system stability and integrity.

All CPUs calling LFA_ACTIVATE enter a controlled and idle state. During this phase, the CPUs are effectively paused and prevented from executing any workload. This may involve disabling the interrupt handling for each CPU to ensure that they no longer process incoming interrupts. This allows the LFA Agent to ensure that each CPUs have successfully halted any operations.

Once the LFA Agent has ascertained that every CPU is synchronized and in a known, stable state, firmware update and activation operation can be performed safely. After those operations are completed, all CPUs resume normal operation with adjusted or restored configurations.

The CPU Rendezvous requirement for a specific firmware is provided as part of the return flags of the LFA_GET_INVENTORY call.

1.1.3 Host Software responsibilities

It is recommended that the Host Software quiesces activity and, if feasible, disables any interrupts prior to calling LFA_ACTIVATE, otherwise the CPU can be overwhelmed upon exit due to an excess of pending interrupts.

In cases where CPU Rendezvous is required, the Host Software should ensure that all active CPUs perform their LFA_ACTIVATE calls as close temporally together as possible in order to minimize activation time and interrupt blackout window. A recommended way to achieve this is to gather all CPUs on a software barrier prior to the calls to LFA_ACTIVATE.

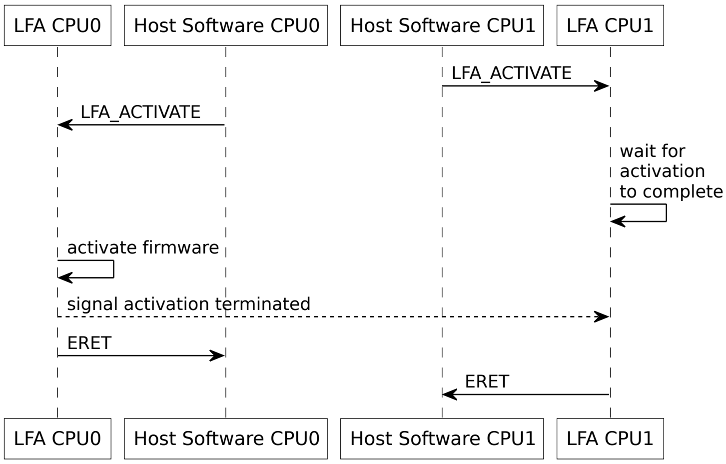

Figure 1.2: LFA_ACTIVATE example

1.2 Security considerations

1.2.1 Firmware authentication

The firmware authentication performed during platform boot is called Verified Boot or Secure Boot. This process ensures the integrity and the trustworthiness of a firmware image before loading and executing it. Verified Boot is based on the Root of Trust (RoT) of Verification, which is anchored to the Hardware Platform RoT or to the immutable platform firmware.

The authentication process is usually composed of two phases:

- Check that the image matches the authentication: e.g. verify that the signature of the image is correct.
- Verify the origin of the authentication: e.g. verify that the signer trust chain links to an approved trust anchor for the system.

R_{XXPCC} The firmware authentication process performed during Live Firmware Activation shall be based on the platform RoT of Verification. This means that for any given firmware image, the boot-time or run-time verification process is based on the same RoT.

1.2.2 Firmware component measurement log

The Host Software can maintain an event log holding a sequence of firmware component digests alongside other miscellaneous information. A new entry is added to the event log when a firmware image is activated.

The firmware component measurement must be performed in a secure manner (including preventing TOC/TOU vulnerabilities from untrusted entities). Measurement of the firmware must be recorded prior to the execution of any part of the new image.

The values resulting from the firmware component measurement encompass both code and initial data. The details of firmware measurement are out of scope of this document.

The firmware component measurements are recorded by a trusted entity. The trusted entity stores or extends a measurement register with new digests. The authenticity of the event log can be verified by the log consumer. The consumer obtains the log digest, that encompasses all the contents in the log – or a set of them – from a trusted party.

The log is deemed authentic if the consumer replays the extension process, using the digests from the log, and the result is the digest obtained from the trusted entity. The format of the log and the protocol to obtain the log measurement, in a trustworthy way, is out of the scope of this specification.

1.2.3 Permanent commitment to firmware images

Each firmware image has a security version.

R_{GPWNT} Prior to activation, the image security version is compared against a platform held security version number (SVN). The SVN is also commonly called an anti-rollback counter. The platform can have an SVN for each firmware image type. The number of SVN existing on a platform and the mapping to each image type is IMPLEMENTATION DEFINED.

R_{BPJVD} An image for which the security version is smaller than the SVN must not be activated, and must never execute on the platform.

R_{VBTLH} The SVN can be incremented to match the security version of the currently active image. This constitutes a permanent commitment to the currently active image, and to those of higher security version. The platform must have a mechanism to increment the SVN. The mechanism to increment the SVN is out of scope of the live activation ABI design.

I_{CMVGM} The SVN should only be updated once sufficient evidence on the stability of the image and correct system operation, following a live activation process, is verified. The procedure to be followed by the Host Software or system administrator, prior to updating the SVN of an image, is IMPLEMENTATION DEFINED.

$\mathrm{I}_{\mathrm{JDTWG}}$ The SVN requirement for image activation applies both to live and cold boot activation. When the SVN is incremented it is recommended that the image with a security version higher or equal exists in the boot Firmware Store.

The high-level procedure to ensure images are fit-for-purpose prior to permanent commitment to these has been referred to as *try-before-buy*.

1.3 Firmware image provisioning

In this document, Live Firmware Activation refers to the process of dynamically activating firmware images stored in the Live Activation Store, which provides access to firmware images that are compatible with Live Activation. Depending on the implementation, the Live Activation Store may be a separate firmware management instance or may be included as part of the Platform Firmware Store. It's important to note that making firmware images available in the Live Activation Store does not necessarily involve a permanent firmware update process that involves writing new firmware images to the Platform Firmware Store. The permanent firmware update process is managed by the Firmware Update (FWU) Agent through platform-specific interfaces. An example of the firmware update ABI is provided in the Firmware Update for the A-profile specification (see [1]).

In this document, provisioning refers to the process of adding a firmware image to the Live Activation Store. Provisioning is a necessary requirement before firmware images can be live activated. There are several ways a platform can provision firmware images. These are described in the following sections and include in-band and out-of-band provisioning.

The method used to provision firmware images depends on the platform. The firmware images must be provisioned before the Live Firmware Activation ABI is invoked. When the images are provisioned to the Live Activation Store, the Host Software can be notified through in-band, out-of-band channels, or the platform interrupt.

1.3.1 In-Band firmware provisioning

In-band firmware provisioning is a method of deploying firmware images in a system using the Host Software and hardware resources within the system. The implementation of the in-band provisioning services is implementation defined and depends on the platform architecture. For in-band provisioning, the platform may include the FWU Agent, which follows the Firmware Update for the A-profile ABI (see [1]).

If the shared buffer exists, then its base address and size are described in the HAFW ACPI table or Device Tree, see Chapter 2.9.3 *LFA Resources Enumeration with ACPI and Device Tree*. The Activation Client must place the firmware images in the shared buffer and call LFA_PRIME. The Activation Client should keep the contents of the shared buffer intact until after the LFA_ACTIVATE call returns successfully. If the Activation Client changes any of the data in the shared buffer between calling LFA_PRIME and the return of LFA_ACTIVATE, the LFA Agent is not guaranteed to perform the image activation and will return an error code. The format of the data placed in the shared buffer is IMPLEMENTATION DEFINED.

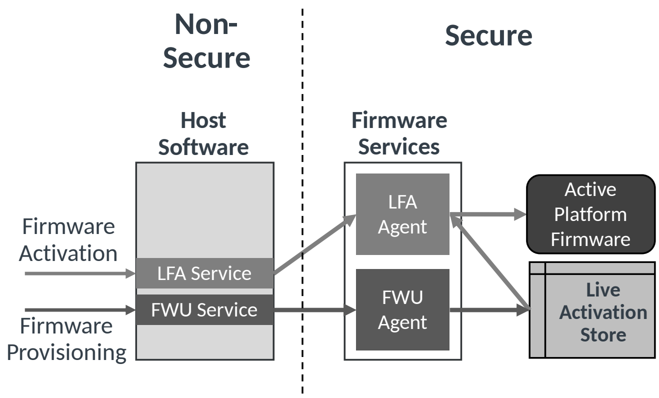

Figure 1.3: Example of in-band live activation

On platforms with the BMC, in-band provisioning can use an approach that redirects the provisioning operation to the Firmware Update Agent located in the BMC module. In this approach, the Host Software communicates with the BMC using a standardized protocol, such as IPMI, PLDM, RedFish, etc. The BMC then transfers the firmware images to the Live Firmware Store via the Firmware Update Agent. The BMC can also provide additional management and monitoring capabilities, such as the ability to verify the integrity of the firmware images before they are deployed.

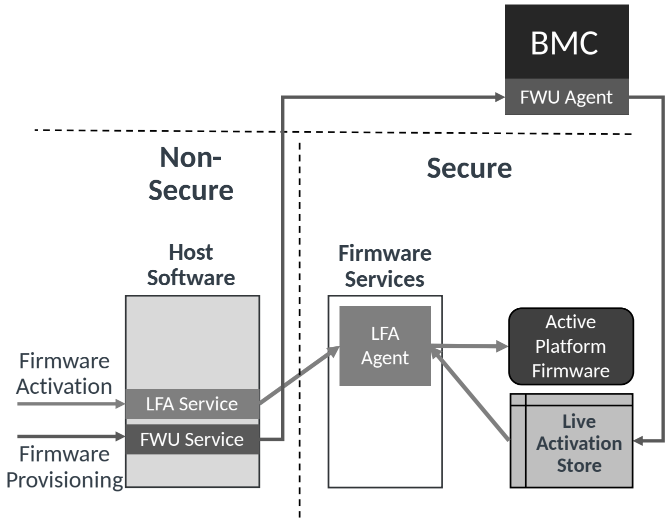

Figure 1.4: Example of in-band live activation with BMC

1.3.2 Out-of-band firmware provisioning

In this method, the BMC communicates directly with the Live Activation Store to transfer firmware images without involving the Host Software or other in-band components. The BMC can use standard out-of-band interfaces such as IPMI, RedFish, SSH, etc. to receive the firmware image. The interface between the BMC and the Live Activation Store is implementation defined. When the image is provisioned to the Live Activation Store, either the BMC can notify the Host Software through an internal platform-specific interface, or the live activation is scheduled using the Host Software's internal scheduler, or the Host Software is notified through online system services.

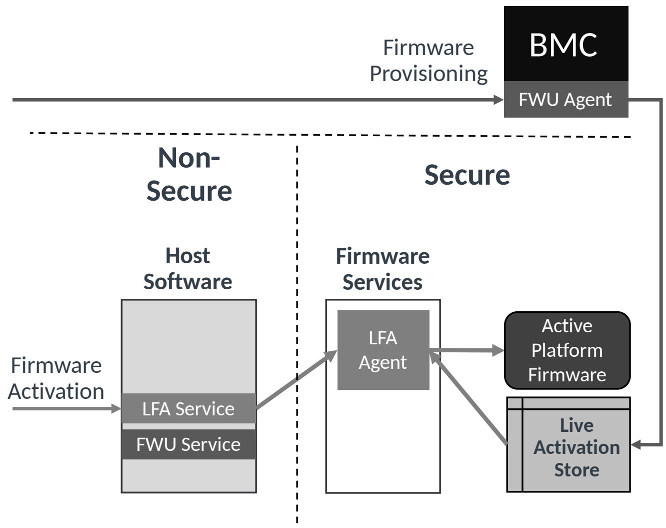

Figure 1.5: Example of out-of-band live activation

Chapter 2

Firmware activation ABI

The LFA ABI is present if the LFA_VERSION call is implemented.

The LFA ABI can only be present on SMCCC implementation that comply with SMCCC [2] version 1.2 or later. It uses SMC64/HVC64 convention and is only defined for AArch64 architecture state.

The caller must ensure all functions (except LFA_VERSION and LFA_FEATURES) are implemented before calling them. This is discoverable by calling LFA_FEATURES with lfa_func_fid set to the function identifier (FID).

2.1 LFA_VERSION

2.1.1 Usage

A caller uses this function to determine the version of the LFA ABI.

2.1.2 Caller responsibilities

A caller must determine that the SMCCC implementation version is greater than or equal to 1.2 before making this call.

2.1.3 Implementation responsibilities

Implementation compliant with this revision of the specification should return:

- *major_version* = 1
- *minor_version* = 0

2.1.4 Arguments

Register	Argument	Description
X0	FID = 0xC400_02E0	The function ID of the LFA_VERSION function.

2.1.5 Returns

Register	Return	Description
X0	version	If X0 >= 0: LFA_SUCCESS • X0[63:31]: MBZ • X0[30:16]: major_version • X0[15:0]: minor_version If X0 < 0: ERROR • X0 = -1: LFA_NOT_SUPPORTED

2.2 LFA_FEATURES

2.2.1 Usage

A caller uses this function to determine the existence of functions in the LFA ABI.

2.2.2 Implementation responsibilities

The implementation must return LFA_NOT_SUPPORTED if the function with FID=lfa_fid is not implemented.

2.2.3 Arguments

Register	Argument	Description
X0	FID = 0xC400_02E1	The function ID of the LFA_FEATURES function.
X1	lfa_fid	The function ID of the LFA ABI function to query features from.

2.2.4 Returns

Register	Return	Description
X0	features	• LFA_SUCCESS: function with FID=lfa_fid is implemented. • LFA_NOT_SUPPORTED: function with FID=lfa_fid is not implemented.

2.3 LFA_GET_INFO

2.3.1 Usage

A caller can obtain information about the LFA behaviour and current configuration. The information items that are obtainable via this call are:

- *lfa_info_selector*=0 (see Table 2.6):
  - the number of firmware components under supervision of the LFA.

2.3.2 Implementation responsibilities

The Implementation must:

- return LFA_INVALID_PARAMETERS if *lfa_info_selector* is a reserved value.

2.3.3 Arguments

Register	Argument	Description
X0	FID = 0xC400_02E2	The function ID of the LFA_GET_INFO function.
X1	lfa_info_selector	Selects the information returned. • lfa_info_selector = 0: (see Table 2.6) All other values of lfa_info_selector are reserved.

2.3.4 Returns

Table 2.6: Return for input lfa_info_selector = 0

Register	Return	Description
X0	status	LFA_SUCCESS
X1	lfa_num_components	The total number of platform firmware components.

2.4 LFA_GET_INVENTORY

2.4.1 Usage

A caller uses this function to discover the firmware components that are managed by the LFA.

2.4.2 Implementation responsibilities

The LFA Implementation contains an IMPLEMENTATION DEFINED procedure that determines the firmware components that can be activated with a call to LFA_PRIME and LFA_ACTIVATE.

The policy adopted in this LFA procedure used to determine which firmware images can be activated (including dependencies between components) is IMPLEMENTATION DEFINED.

The mapping between *fw_seq_id* value and component is IMPLEMENTATION DEFINED and must be stable between activations. After an activation, the caller must re-enumerate the component states (using LFA_GET_INFO then LFA_GET_INVENTORY).

The implementation may only return flag value of cpu_rendezvous_optional=1 if may_reset_cpu=0. It is acceptable for an implementation to always return may_reset_cpu=1.

2.4.3 Preconditions

- The LFA_GET_INFO call shall be issued before this call, otherwise the call returns LFA_WRONG_STATE.
- The firmware component fw_seq_id shall be in the range of the components number lfa_info_selector returned by LFA_GET_INFO, otherwise the call returns LFA_INVALID_PARAMETERS.

2.4.4 Arguments

Register	Argument	Description
X0	FID = 0xC400_02E3	The function ID of the LFA_GET_INVENTORY function.
X1	fw_seq_id	The sequence ID of the firmware component. All firmware components managed by the LFA have a component sequence ID ranging from 0 up to *lfa_num_components* -1. The value of *lfa_num_components* can be obtained by calling LFA_GET_INFO.

2.4.5 Returns

Register	Return	Description
X0	status	• LFA_SUCCESS • LFA_INVALID_PARAMETERS • LFA_WRONG_STATE
X1	uuid_0	Only valid if X0=LFA_SUCCESS. Bytes 0 to 7 of the firmware component UUID. Byte 0 is located in the LSB of the UUID_0 field.
X2	uuid_1	Only valid if X0=LFA_SUCCESS. Bytes 8 to 15 of the firmware component UUID. Byte 8 is located in the LSB of the UUID_1 field.

Register	Return	Description
X3	flags	Only valid if X0=LFA_SUCCESS. • flags[0]: activation_capable – 1: component is capable of live activation. – 0: component cannot be live activated. • flags[1]: activation_pending – 1: activation of component is pending. – 0: activation of component is not pending. • flags[2]: may_reset_cpu – 1: LFA_ACTIVATE calls may reset CPUs. – 0: LFA_ACTIVATE calls will return to caller and will not reset CPUs. • flags[3]: cpu_rendezvous_optional – 1: component does not require CPU Rendezvous for activation. – 0: component does require CPU Rendezvous for activation. All other bits are reserved and MBZ.

2.5 LFA_PRIME

2.5.1 Usage

A caller uses this call to prepare the platform for a live firmware activation.

2.5.2 Caller responsibilities

A caller must invoke this function prior to invoking LFA_ACTIVATE.

Prior to invocation, the caller must ensure that a firmware for the component (identified by fw_seq_id) is available in the Live Activation Store.

Based on the value of the call_again flag returned by LFA_PRIME, additional calls might be required. Those calls can be issued from different CPUs, but only one CPU at a time should be making a call to LFA_PRIME.

2.5.3 Implementation responsibilities

The implementation will perform any preparatory actions for the live activation. These actions are IMPLEMENTATION DEFINED. Examples of the actions are:

- Loading firmware to be activated to a staging area and/or final execution location.
- Authenticating and/or measuring firmware image.

During this phase it is guaranteed that any firmware components remain in operation and able to service any requests. The implementation must not overwrite any of the currently active firmware instances during this call. The implementation must not initialise any of the new firmware components at this stage. The implementation can choose not to perform any actions, that is a legal platform-specific design choice.

The implementation must enforce that LFA_PRIME can only be called for one component at a time. Once a component is ready for activation, LFA_PRIME must not prepare live activation for another component until LFA_ACTIVATE or LFA_CANCEL have been finalized for the current live activation.

LFA_PRIME is a single-threaded operation that is not pinned to a specific CPU. The implementation must support calls being issued from difference CPUs, even for several calls to prime the same component. However, those calls must not happen concurrently.

2.5.4 Arguments

Register	Argument	Description
X0	FID = 0xC400_02E4	The function ID of the LFA_PRIME function.
X1	fw_seq_id	The sequence ID of the firmware component to prepare for activation.

2.5.5 Returns

Register	Return	Description
X0	status	• LFA_SUCCESS: The call completed successfully. • LFA_AUTH_ERROR: a firmware component failed to authenticate. • LFA_NO_MEMORY: insufficient memory to load firmware component. • LFA_DEVICE_ERROR: failed to load firmware component from the Live Activation Store. • LFA_WRONG_STATE: preconditions unmet. • LFA_BUSY: LFA_PRIME is executing concurrently on another CPU. • LFA_ACTIVATION_FAILED: operation failed, activation cancelled.
X1	flags	Only valid if X0=LFA_SUCCESS. • flags[0]: call_again – 1: LFA_PRIME is incomplete and must be called again. – 0: The LFA_PRIME procedure successfully completed. All other bits are reserved and MBZ.

2.6 LFA_ACTIVATE

2.6.1 Usage

A caller uses this call to request an immediate activation of the firmware component primed for activation.

2.6.2 Caller responsibilities

The caller responsibilities depend on the value of the may_reset_cpu flag returned by LFA_GET_INVENTORY for the component.

See also 1.1.3 *Host Software responsibilities*.

2.6.2.1 Possible CPU reset (may_reset_cpu=1)

- The caller must have saved all the state it requires to enable resumption of operation following a CPU reset. The context saved must reflect all caller visible state that is lost if a CPU reset has been performed during activation.
- The caller must not assume that the call will return using the specified entry point address. The implementation might return at the instruction following the SMC call, at the Exception level of the caller, instead of returning by the specified entry point address.
- The entry point address provided in the call must be a physical address from the point of view of the caller.
- The context identifier is meaningful only to the caller. The value is preserved by the implementation and presented to the CPU when it is started up at the return Exception level.
- The caller is not required to perform any cache or coherency management.

2.6.2.2 No CPU reset (may_reset_cpu=0)

The values of entry_point_address and context_id can be set to 0, and the call to LFA_ACTIVE will return at the instruction following the SMC call.

2.6.2.3 Skipping CPU Rendezvous (skip_cpu_rendezvous=1)

The caller takes responsibility for ensuring that other active cores will be able to handle the unavailability of the component during its live activation.

The option skip_cpu_rendezvous=1 is only valid if the flag value cpu_rendezvous_optional=1 was returned by LFA_GET_INVENTORY for the component.

When skip_cpu_rendezvous=1, additional LFA_ACTIVATE calls might be required. This will be signaled in the call_again flag returned. Those calls can be issued from different CPUs, but only one CPU at a time is permitted to make a call to LFA_ACTIVATE.

2.6.3 Implementation responsibilities

The implementation responsibilities depend on whether the firmware component requires a CPU reset operation during live firmware activation. This condition should have been reflect in the value of the may_reset_cpu flag returned from the LFA_GET_INVENTORY call for the component.

See also 1.1.1 *Live Firmware Activation Agent responsibilities*.

The implementation must be able to handle cancellation of live activation (through a call to LFA_CANCEL) even if some cores have called into LFA_ACTIVATE.

In case of non-critical errors during activation, the implementation should ensure that all CPUs return with the same error code.

2.6.3.1 Possible CPU reset (may_reset_cpu=1)

The semantics of LFA_ACTIVATE are similar to PSCI_CPU_SUSPEND (see [3]).

- The entry_point_address parameter is used by the caller to specify where code execution needs to resume after activation. The parameter must be a Physical Address (PA).
- The context_id parameter is only meaningful to the caller. The LFA implementation must preserve a copy of the value passed in this parameter. Following activation, the LFA implementation must place this value in X0, when it enters the first Non-secure Exception level. The context identifier can be used by the caller to point to the saved context that must be restored on a CPU when it starts up at the return Exception level. The caller can use other methods to implement this functionality.
- The LFA implementation must perform cache or coherency management.
- LFA_INVALID_ADDRESS is returned when the entry point address is known by the implementation to be invalid, because it is in a range that is known not to be available to the caller.

2.6.3.2 No CPU reset (may_reset_cpu=0)

The entry_point_address and context_id parameters should be ignored. LFA_ACTIVATE shall return at the instruction following the SMC call.

2.6.3.3 CPU Rendezvous

If LFA_GET_INVENTORY returns a flags value with cpu_rendezvous_optional=1, the host software can choose to set the flags value skip_cpu_rendezvous=1 in the LFA_ACTIVATE call parameters to signal that activation can proceed without CPU Rendezvous using the caller core. In this case the implementation must support calls being issued from difference CPUs, even for several calls to active the same component. However, those calls must not happen concurrently.

Otherwise the activation will proceed only when all active cores have called LFA_ACTIVATE.

2.6.4 Preconditions

- Firmware component preparation for live activation completed successfully with last call to LFA_PRIME returning flags[0] (call_again) equal to zero, otherwise this call returns LFA_WRONG_STATE.
- If skip_cpu_rendezvous=0, all active cores of the platform shall have a power-on state.

2.6.5 Postcondition

- Upon successful completion of the activation process, the firmware component prime status associated with the specified component fw_seq_id shall be reset.

2.6.6 Error conditions

- LFA_INVALID_PARAMETERS, LFA_INVALID_ADDRESS, and LFA_WRONG_STATE are caller error (wrong parameters or sequence), they do not alter the current activation state. LFA_ACTIVATE call can be performed again later with the correct parameters or after the correct sequence of calls.
- LFA_BUSY means live activation was postponed and should be attempted again later. The prime status of the firmware component has not been changed. The delay value for the next activation attempt is IMPLEMENTATION DEFINED.
- LFA_CRITICAL_ERROR means the activation of the component failed and the component is now in a non-recoverable state. The system should be restarted. Depending on the component (e.g. EL3 Runtime Firmware), LFA might need to initiate system reset. LFA shall rejected all subsequent LFA_PRIME and LFA_ACTIVATE calls and return LFA_WRONG_STATE.
- All other failures mean component live activation could not proceed and has been cancelled. This includes canceling the firmware component prime status associated with component fw_seq_id.

2.6.7 Arguments

Register	Argument	Description
X0	FID = 0xC400_02E5	The function ID of the LFA_ACTIVATE function.
X1	fw_seq_id	The sequence ID of the firmware component to activate.
X2	flags	• flags[0]: skip_cpu_rendezvous – 1: activation will proceed without CPU rendezvous. – 0: activation will proceed after CPU Rendezvous of all active cores. All other bits are reserved and MBZ.
X3	entry_point_address	Address at which the CPU must resume execution, when it enters the return Non-secure Exception level following the end of live activation.
X4	context_id	When the calling CPU first enters the return Non-secure Exception level, this value must be present in X0.

2.6.8 Returns

Register	Return	Description
X0	status	• LFA_SUCCESS: the call completed successfully. • LFA_INVALID_PARAMETERS: invalid calling parameters. • LFA_INVALID_ADDRESS: invalid address. • LFA_WRONG_STATE: preconditions unmet. • LFA_BUSY: system not ready for activation. • LFA_CRITICAL_ERROR: non-recoverable error during activation. • LFA_AUTH_ERROR: a firmware component failed to authenticate. • LFA_NO_MEMORY: insufficient memory to load firmware component. • LFA_DEVICE_ERROR: failed to load firmware component from the Live Activation Store. • LFA_COMPONENT_WRONG_STATE: a firmware component was not in the correct state to start the activation process. • LFA_ACTIVATION_FAILED: component activation failed.
X1	flags	Only valid if X0=LFA_SUCCESS. • flags[0]: call_again (only if skip_cpu_rendezvous=1, MBZ otherwise) – 1: LFA_ACTIVATE is incomplete and must be called again. – 0: The activate procedure successfully completed. All other bits are reserved and MBZ.

2.7 LFA_CANCEL

2.7.1 Usage

A caller uses this call to abort the firmware activation process during the prime or activate stages.

2.7.2 Caller responsibilities

- This call is allowed to be issued either during the preparation of the firmware component in the prime phase LFA_PRIME or after the prime phase LFA_PRIME has been completed.
- This call is allowed to be issued during the activation of the firmware in the activate phase LFA_ACTIVATE before all active cores of the platform have called LFA_ACTIVATE. If LFA_ACTIVATE is called with flag value of skip_cpu_rendezvous=1, the LFA_CANCEL can fail with a LFA_BUSY return code if activation cannot be canceled.

2.7.3 Implementation responsibilities

- This call shall cancel the firmware activation process and reset the firmware component prime status identified with firmware component fw_seq_id, allowing LFA_PRIME to restart the firmware component preparation process.
- This call shall return LFA_SUCCESS if no component is in the prime or activate phases.
- This call shall return LFA_BUSY if the LFA_ACTIVATE call had a flag value of skip_cpu_rendezvous=1 and the activation cannot be canceled.
- The firmware component fw_seq_id shall match the fw_seq_id used during the prime and activation phases, otherwise this call returns LFA_INVALID_PARAMETERS.

2.7.4 Arguments

Register	Argument	Description
X0	FID = 0xC400_02E6	The function ID of the LFA_CANCEL function.
X1	fw_seq_id	The sequence ID of the component for which activation will be canceled.

2.7.5 Returns

Register	Return	Description
X0	status	• LFA_SUCCESS: the call completed successfully. • LFA_BUSY: Live activation in progress. • LFA_INVALID_PARAMETERS: Invalid parameters.

2.8 Return status

Status	Value	Description
LFA_SUCCESS	0	The call completed successfully.
LFA_NOT_SUPPORTED	-1	Function not implemented.
LFA_BUSY	-2	Operation already in progress or system not ready.
LFA_AUTH_ERROR	-3	Firmware image authentication failed.
LFA_NO_MEMORY	-4	Insufficient memory to perform operation.
LFA_CRITICAL_ERROR	-5	Non-recoverable error.
LFA_DEVICE_ERROR	-6	Error while accessing Live Activation Store.
LFA_WRONG_STATE	-7	Preconditions are not met.
LFA_INVALID_PARAMETERS	-8	Invalid calling parameters.
LFA_COMPONENT_WRONG_STATE	-9	Firmware component not in the correct state.
LFA_INVALID_ADDRESS	-10	Address is not valid.
LFA_ACTIVATION_FAILED	-11	Component activation failed.

2.9 Live Firmware Activation example flows

This specification defines a set of SMC calls that enable the Host Software to trigger a live firmware activation.

The backend implementation to the SMC calls is platform-specific. This Section provides implementation guidance that platforms can adopt for the live activation flows of some firmware components.

2.9.1 Inventory sequence flow

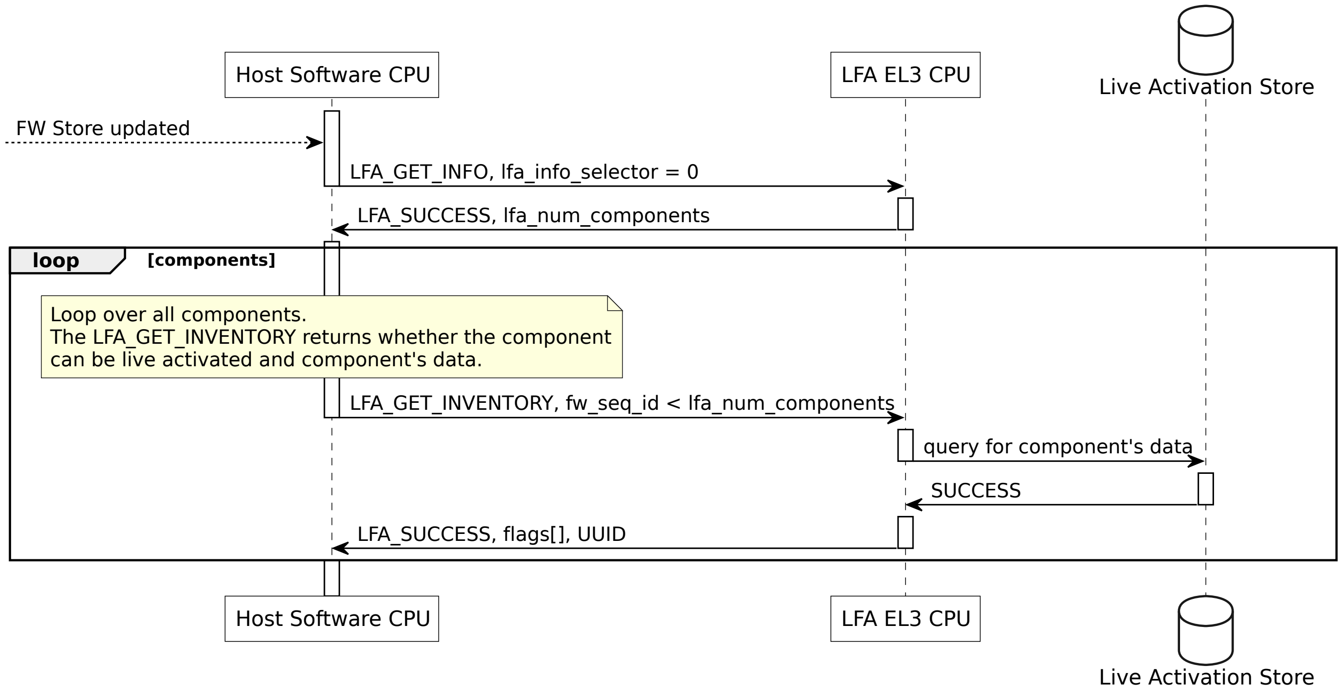

Figure 2.1: Inventory sequence flow

2.9.2 PRIME sequence flow

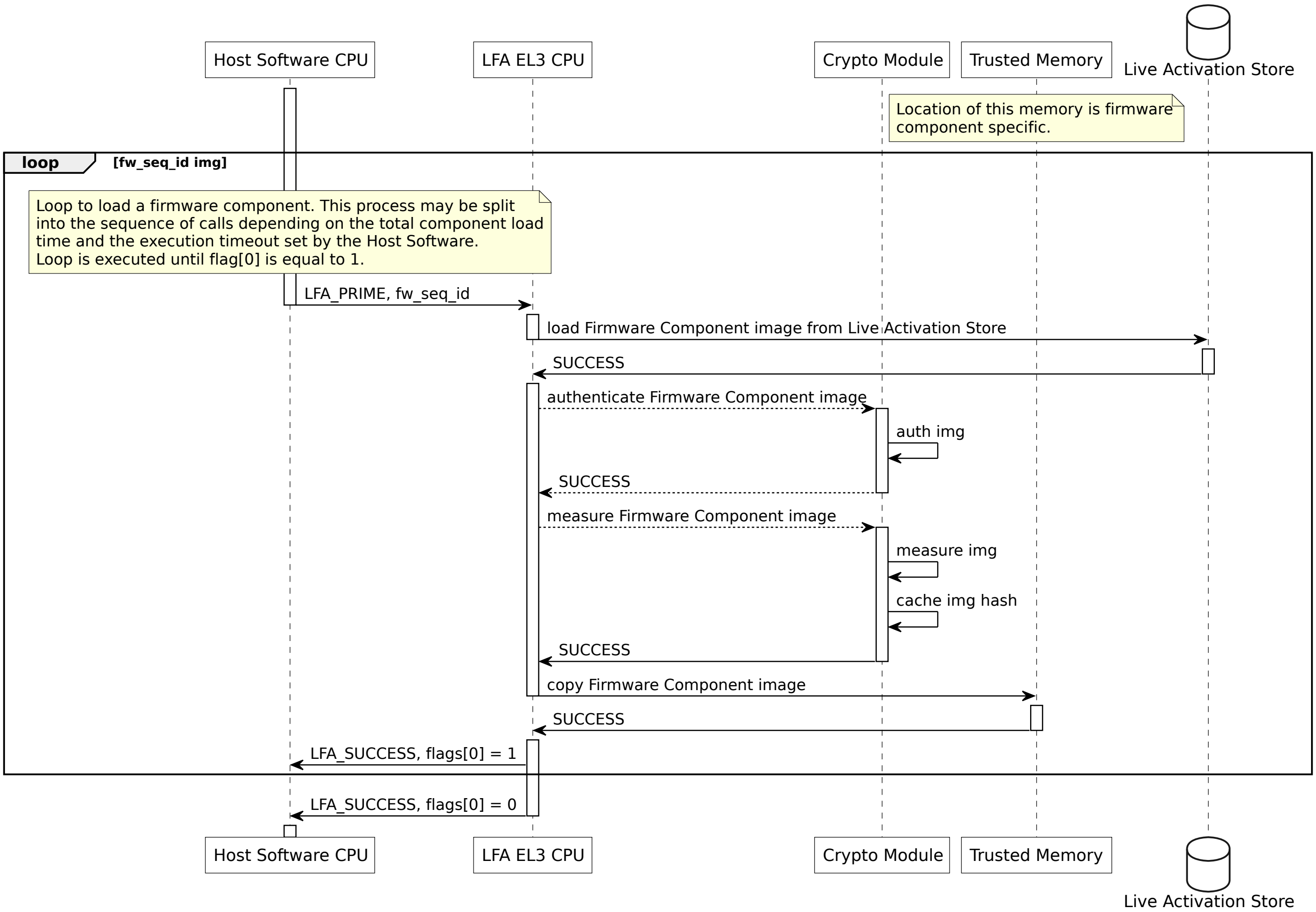

Figure 2.2: PRIME sequence flow

2.9.3 ACTIVATE sequence flow

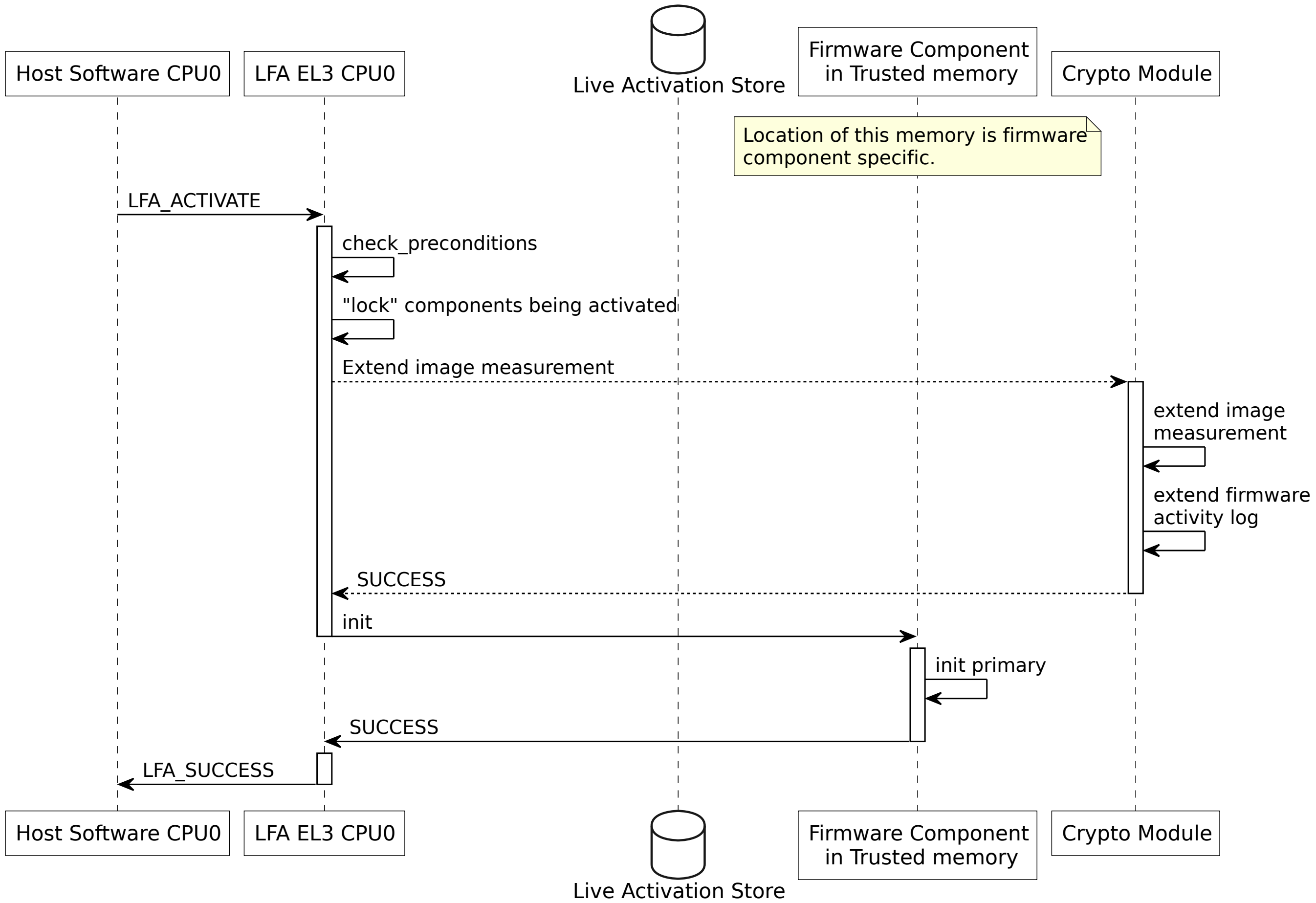

Figure 2.3: ACTIVATE sequence flow (skip_cpu_rendezvous=1)

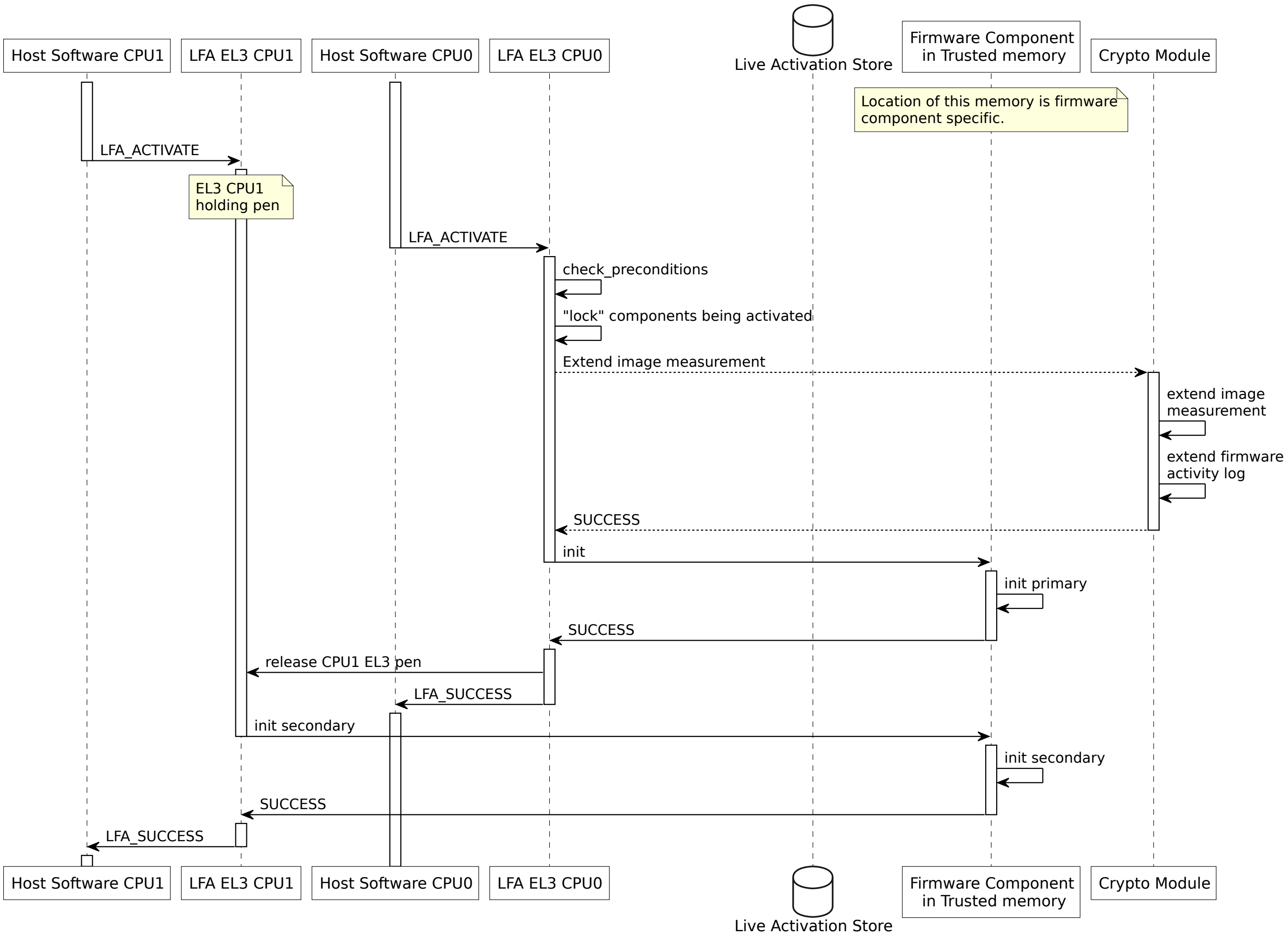

Figure 2.4: ACTIVATE sequence flow (skip_cpu_rendezvous=0 with 2 active cores)

Appendix

LFA Resources Enumeration with ACPI and Device Tree

The resource tables and structures presented in this section provide the Host Software with information regarding:

- The payload buffer used for in-band image provisioning necessary for Live Firmware Activation.
- The interrupt that signals when the Live Activation Store has been updated.

Host Software can use common standards such as Advanced Configuration and Power Interface (ACPI) or Device Tree (DT) to enumerate and access Live Firmware Activation resources. ACPI provides a structured method using firmware tables defined in ASL (ACPI Source Language), whereas Device Tree utilizes a structured data format describing hardware resources explicitly for platforms without ACPI support.

Note

The adresses and interrupt values in the following sections are provided as examples. Depending on the platform design, some fields may not be relevant. The absence of a specific field implies that the associated functionality or flow it enables is not supported by the platform. For instance, if the Payload buffer is absent from the structure, the image provisioning for live firmware activation must be carried out via alternate mechanisms which may include out-of-band provisioning of firmware images via BMC, using peripheral devices capable of Direct Memory Access (DMA), or transferring data through external storage interfaces or network-based file transfers.

ACPI Table

```
DefinitionBlock ("", "SSDT", 2, "XXXXXX", "XXXXXXXX", 1) {

    Scope (\_SB)
    {
        Device (FWU0) {

            Name (_HID, "ARMHF000") // Arm HAFW DSDT table identifier.

            Name (_CRS, ResourceTemplate () {

                // Payload buffer description.
                QWordMemory (,
                      ,
                      MinFixed,
                      MaxFixed,
                      NonCacheable,
                      ReadWrite,
                      0x0,
                      0x2000, // base PA of payload buffer
                      0x2FFF, // top PA of payload buffer
                      0,
                      0x1000, // payload buffer length
                      ,
                      ,
                )

                // Interrupt signaling the updated Live Activation Store.
                Interrupt(,
                    Edge,  // Interrupt type (or interrupt mode)
                    ActiveHigh, // Interrupt level
                    ,
                    ,
                    ,
                ) {100} // Note that this is an example interrupt.
                        // The interrupt ID is defined per-platform.
            })

            // _DSD -- Holds map between field names and objects within the _CRS
            Name (_DSD, Package () {
                ToUUID ("daffd814-6eba-4d8c-8a91-bc9bbf4aa30") ,
                Package() {

                    // Contains a set of packages of 2 entries each,
                    // the second entry is an index into the _CRS.
                    Package (2) {"payload-buffer",   1},
                    Package (2) {"fw-store-updated-interrupt", 2},
                },
            })
        }
    }
}
```

Device Tree structure

```
/dts-v1/;

/ {
    #address-cells = <2>;
    #size-cells = <2>;

    ic: interrupt-controller@2f000000 {
        compatible = "arm,gic-v3";
        #interrupt-cells = <3>;
        #address-cells = <2>;
        #size-cells = <2>;
        interrupt-controller;
        reg = <0x0 0x2f000000 0x0 0x10000>,
              <0x0 0x2f100000 0x0 0x200000>;
    };

    reserved-memory {
        #address-cells = <2>;
        #size-cells = <2>;
        ranges;

        fwu_payload: fwu_payload@2000 {
            reg = <0x0 0x2000 0x0 0x1000>;
            no-map;
        };
    };

    soc {
        #address-cells = <2>;
        #size-cells = <2>;
        ranges;

        fwu0 {
            compatible = "arm,armhf000";
            memory-region = <&fwu_payload>;
            interrupt-parent = <&ic>;
            interrupts = <0 100 1>; // SPI, Interrupt #100, Edge Rising
        };
    };
};
```