FF-A Memory Management Protocol

Document number	DEN0140
Document quality	BETA0
Document version	1.2
Document confidentiality	Non-confidential

FF-A Memory Management Protocol

Release information

Date	Version	Changes
2024/May/03	v1.2 BET0	• Renamed Bypass multi-borrower check to Bypass multiple borrower identification check.
2023/Nov/30	v1.2 ALP0	• Added memory management guidance from the FF-A v1.2 ALP0 Base specification. • Updated ABIs to reserve extended register contents and clarified Reserved register usage. • Clarified usage of the IMPDEF field in an EMAD. • Clarified a Borrower must specify memory permissions in a retrieve request. • Extended ABIs to define extended register usage. • Fixed memory transaction example diagrams.

Non-Confidential Proprietary Notice

Arm Limited. Company 02557590 registered in England.

110 Fulbourn Road, Cambridge, England CB1 9NJ.

LES-PRE-20349 version 21.0

Contents

FF-A Memory Management Protocol

Preface

This document describes the FF-A Memory Management protocol. Until FF-A v1.2 ALP0, this guidance was a part of the FF-A specification [1] henceforth called the *Base specification*. This guidance has been separated into this document for the sake of brevity of the Base specification. This change is applicable from FF-A v1.2 ALP1 of the Base specification.

The guidance is treated as an extension of the Base specification. It is not versioned independently. Instead, it adopts that version of the Base specification in which changes to this guidance are released. It also fulfils the same compatibility requirements as the Base specification.

The guidance in this document can be at a different ALPHA quality level as compared to the Base specification. E.g. this document could be at ALP1 while the Base specification is at ALP3. To achieve alignment with the Base specification, this document must be at the BETA quality level for the Base specification to qualify for the same quality level. This approach allows this document to evolve somewhat independently of the Base specification w.r.t quality levels but also provides a point of alignment at the BETA quality level.

The reader is expected to use the guidance in this document in conjunction with the Base specification.

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.

Additional reading

This section lists publications by Arm and by third parties.

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

[1] *Arm Firmware Framework for Arm A-profile version 1.2*. See https://developer.arm.com/documentation/den0077/g

Feedback

Arm welcomes feedback on its documentation.

If you have any comments or queries about our documentation, create a ticket at https://support.developer.arm.com/.

As part of the ticket, include:

- The title, FF-A Memory Management Protocol.
- The document ID and version, DEN0140 1.2.
- The section name to which your comments refer.
- The page number(s) to which your comments apply.
- The rule identifier(s) to which your comments apply, if applicable.
- A concise explanation of your comments.

Arm also welcomes general suggestions for additions and improvements.

Chapter 1
Memory Management

1.1 Overview

The Firmware Framework describes mechanisms and interfaces that enable FF-A components to manage access and ownership of memory regions in the physical address space to fulfill use cases such as:

- DRM protected video path.
- Communication with a VM with pre-configured machine learning frameworks,
- Biometric authentication and Secure payments.

FF-A components can use a combination of Framework and Partition messages to manage memory regions in the following ways.

1. The Owner of a memory region can transfer its ownership to another FF-A endpoint.
2. The Owner of a memory region can transfer its access to one or more FF-A endpoints.
3. The Owner of a memory region can share access to it with one or more FF-A endpoints.
4. The Owner can reclaim access to a memory region after the FF-A endpoints that were granted access to that memory region have relinquished their access.

All FF-A components except for EL3 LSPs that are co-resident with the EL3 SPMC can use the memory management interfaces described in this chapter.

1.2 Address translation regimes

Memory management relies on the two fundamental operations of mapping and un-mapping a memory region from the stage of a translation regime managed by a partition manager on behalf of a physical partition. This enables the Framework to enforce isolation of the physical address space through the use of access and ownership attributes described in 1.3 *Ownership and access attributes*. The translation regime and the stage depend on the type of physical partition as follows.

1. The Hypervisor creates and manages stage 2 translations on behalf of a EL1 PE endpoint, in the Non-secure EL1&0 translation regime, when EL2 is enabled.
2. The Hypervisor creates and manages stage 2 translations for a Non-secure Stream ID assigned to an independent or dependent peripheral device, in the Non-secure EL1&0 translation regime in the SMMU. A SEPID is used to identify the stage 2 translation tables (see the Base FF-A Specification [1]).
3. The SPMC creates and manages stage 2 translations on behalf of a S-EL1 PE endpoint in the Secure EL1&0 translation regime, when S-EL2 is enabled.
4. The SPMC creates and manages stage 1 translations on behalf of a S-EL0 PE endpoint in the Secure EL2&0 translation regime, when S-EL2 is enabled.
5. The SPMC creates and manages stage 1 translations on behalf of a S-EL0 PE endpoint in the Secure EL1&0 translation regime, when S-EL2 is disabled.
6. The SPMC creates and manages stage 2 translations for a Secure Stream ID assigned to an independent or dependent peripheral device in the Secure EL1&0 translation regime in the SMMU. A SEPID is used to identify the stage 2 translation tables (see the Base FF-A Specification [1]).

The role of the Hypervisor or SPMC in managing a stage of the translation regime of a logical partition that is not co-resident with them is IMPLEMENTATION DEFINED.

1.3 Ownership and access attributes

The Hypervisor, SPM, and all endpoints have *access* and *ownership* attributes associated with every memory region in the physical address space.

Access determines the level of visibility an FF-A component has of a physical memory region at a point in time. Table 1.1 describes the levels of access an FF-A component can have for a memory region.

Table 1.1: Access states

No.	Access state	Acronym	Description
1	No access	NA	A component has *no* access to a memory region. It is not mapped in the component's translation regime.
2	Lent access	LA	A component has *no* access to a memory region. It is not mapped in the component's translation regime but it is mapped in the translation regime of at least one other component.
3	Exclusive access	EA	A component has exclusive access to a memory region. It is mapped only in the component's translation regime.
4	Shared access	SA	A component has shared access to a memory. It is mapped in the component's translation regime and the translation regime of at least one other component.

A component has access to a memory region with one of the following combinations of data and instruction access permissions.

- Read-only, Execute-never.
- Read-only, Executable.
- Read/write, Execute-never.

Ownership is a software attribute that determines if a component can allocate and grant access to a memory region to another component. Table 1.2 describes the ownership states applicable to an FF-A component for a physical memory region.

Table 1.2: Ownership states

No.	Ownership state	Acronym	Description
1	Owner	Owner	Component owns the memory region.
2	Not Owner	!Owner	Component does not own the memory region.

A component that has access to a memory region without ownership is called the *Borrower*. A component that lends access to a memory region it owns is called the *Lender*. Only the *Owner* of a memory region can be its *Lender*.

A component can own a memory region without having access to it. An Owner without access can still allocate chunks from the memory region and request the partition manager to lend it to one or more Borrowers. An example of this memory usage model is when a pool of Secure memory is owned by an OS in the Normal world. It cannot access the memory but can lend it to a S-Endpoint to implement a use case like protected video path setup.

Access control must be enforced through an IMPLEMENTATION DEFINED mechanism and/or by encoding these

permissions in the translation regime of a physical endpoint managed by the partition manager (see 1.2 *Address translation regimes*).

Ownership of a memory region must be assigned and tracked by the Hypervisor and SPMC for both logical and physical partitions through an IMPLEMENTATION DEFINED mechanism.

An endpoint requests access to and/or ownership of a memory region through its partition manifest during boot time (see the Base FF-A Specification [1] for more information).

The Framework assumes that ownership and access attributes are,

1. Not enforceable by the Hypervisor or SPMC for a logical partition resident in a separate exception level.
2. Enforced by the Hypervisor or SPMC for a co-resident logical partition through an IMPLEMENTATION DEFINED mechanism.

The attributes can be still used to facilitate migration of the partition from being logically isolated to being physically isolated.

1.3.1 Ownership and access rules

The SPM and Hypervisor must enforce the following general ownership and access rules to memory regions.

1. The size of a memory region to which ownership and access rules apply must be a multiple of the smallest translation granule size supported on the system.
   - It is 4K in the AArch32 Execution state.
   - A EL1 or S-EL1 partition must discover this by reading the ID_AA64MMFR0_EL1 System register in the AArch64 Execution state.
   - A S-EL0 SP must determine this through an IMPLEMENTATION DEFINED discovery mechanism for example, DT or ACPI tables.
2. A normal memory region must be mapped with the Non-secure security attribute in any component that is granted access to it.
3. A Secure memory region must be mapped with the Secure security attribute in any component that is granted access to it.
4. Each memory region in the physical address space must have a single Owner.
5. Only the Owner of a memory region can grant access to it to one or more Borrowers in the system.
6. Only the Owner of a memory region can transfer its ownership to another endpoint in the system.
7. The Owner of a memory region must not be able to change its ownership or access attributes until all Borrowers have relinquished access to it.
8. The number of distinct components to whom an Owner can grant access to a memory region is IMPLEMENTATION DEFINED.
9. If an SP is terminated because of a fatal error condition, access to the memory regions of the SP is transferred to the SPM.
10. If a VM is terminated because of a fatal error condition, access to the memory regions of the VM and their ownership are transferred to the Hypervisor.
11. If the Hypervisor or OS kernel are terminated because of a fatal error condition, access to their memory regions and ownership are transferred to the SPM.

1.3.2 Ownership and access states

Table 1.3 describes valid combinations of access and ownership states applicable to an FF-A component for a memory region.

Table 1.3: Valid combinations of ownership and access states

No.	Ownership state	Access state	Acronym	Description
1	Not Owner	No access	!Owner-NA	Component has neither ownership nor access to the memory region. Another component owns it and might have access to it. Other components might have access to it too.
2	Not Owner	Exclusive access	!Owner-EA	Component has exclusive access without ownership of the memory region.
3	Not Owner	Shared access	!Owner-SA	Component has shared access with one or more components without ownership of the memory region.
4	Owner	No access	Owner-NA	Component owns the memory region and has no access to the memory region. No other component has access to the memory region either.
5	Owner	Exclusive access	Owner-EA	Component owns the memory region and has exclusive access to it.
6	Owner	Shared access	Owner-SA	Component owns the memory region and shares access to it with one or more components.
7	Owner	Lent access	Owner-LA	Component owns the memory region and has no access to the memory region. It has, • Either granted exclusive access to the memory region to another component. • Or shared access to the memory region among other components.

1.4 Memory management transactions

This version of the Framework describes transactions that enable endpoints to manage access and ownership of physical memory regions. These transactions are listed in Table 1.4.

- Each transaction involves a memory region, an Owner endpoint (Endpoint A) and at least one another endpoint (Endpoint B).
- Each transaction triggers a transition from one valid combination of ownership and access states listed in Table 1.5 to another.
- The transition to effect the transaction is performed in one or more steps described in 1.4.2 *Transaction life cycle*.
- Each step is performed by invoking an FF-A ABI listed in Chapter 2 *Memory management interfaces*.

Table 1.4: Memory region transactions

No.	Transaction	Description
1.	Donate	• Endpoint A transfers ownership of a memory region it owns to endpoint B. See 1.5 *Donate memory transaction*.
2.	Lend	• Endpoint A transfers access to a memory region to only endpoint B. Endpoint B gains exclusive access to the memory region. • Endpoint A transfers access to a memory region to endpoint B and at least one other endpoint simultaneously. Endpoint B gains shared access to the memory region. • See 1.6 *Lend memory transaction*.
3.	Share	• Endpoint A grants access to a memory region to endpoint B and optionally to other endpoints simultaneously. See 1.7 *Share memory transaction*.
4.	Reclaim	• Endpoint A reclaims exclusive access to a memory region it had lent or shared with Endpoint B.

Table 1.5: Valid combinations of start and end ownership and access states in a memory management transaction

No.	Endpoint A state	Endpoint B state	Description
1	Owner-EA	!Owner-NA	Endpoint A has exclusive access and ownership of the memory region that is inaccessible from endpoint B.
2	Owner-NA	!Owner-NA	Endpoint A has no access to the memory region it owns. The region is inaccessible from endpoint B.
3	!Owner-NA	Owner-EA	Endpoint B has exclusive access and ownership of the memory region that is inaccessible from endpoint A.
4	Owner-LA	!Owner-EA	Endpoint A owns the memory region and has granted exclusive access to the memory region to endpoint B.

No.	Endpoint A state	Endpoint B state	Description
5	Owner-LA	!Owner-SA	Endpoint A has transferred access to a memory region it owns. Access to the memory region is shared between endpoint B and at least one other endpoint.
6	Owner-SA	!Owner-SA	Endpoint A shares access to a region of memory it owns with endpoint B and possibly other endpoints.

U **Implementation Note**

FF-A components should track the state of a memory region during a memory management transaction as follows,

- An Owner tracks the level of access it has to a memory region.
- An Owner tracks the level of access that Borrowers have to a memory region along with the identity of the Borrowers.
- A Borrower tracks the level of access the Owner has to a memory region along with the identity of the Owner.
- A Borrower tracks the level of access it has to a memory region.
- A Borrower tracks the level of access that other Borrowers have to a memory region along with the identity of the Borrowers.
- For each memory region, the SPM and Hypervisor track the following.
  - The identity of each Borrower.
  - The identity of the Owner.
  - The level of access of each Borrower.
  - The level of access of the Owner.

1.4.1 Component roles

The Hypervisor and SPM are responsible for tracking and enforcing the ownership and access attributes of a memory region so that only valid transitions are permitted between states during a memory management transaction. This collective role is termed as a *Relayer*. When both the SPM and Hypervisor are involved in a memory transaction, the SPM must maintain sufficient information such that it does not have dependency on the Hypervisor.

In the absence of the Hypervisor, the OS Kernel subsumes its role as the Relayer.

The SPMD and SPMC collectively fulfill the role of a Relayer as follows.

- The SPMD forwards an FF-A ABI invocation at the Non-secure physical FF-A instance to complete a step in a memory management transaction to the SPMC.
- The SPMC handles FF-A ABI invocations at the Secure physical and virtual FF-A instances from S-Endpoints and those forwarded by the SPMD.

An endpoint can fulfill the role of an Owner, Lender or Borrower (see 1.3 *Ownership and access attributes*) in a memory management transaction. In all transactions, an endpoint is also a Sender or Receiver. This depends on the type of transaction as follows.

- In a transaction to donate ownership of a memory region, the Sender is the current Owner, and the Receiver is the new Owner.
- In a transaction to lend or share access to a memory region, the Sender is the Lender, and the Receiver is the Borrower.
- In a transaction to relinquish access to a memory region, the Sender is the Borrower, and the Receiver is the Lender.

In the absence of the Hypervisor, the OS Kernel's roles as the Relayer, Owner, Lender and Borrower are considered to be logically separate from each other. The interface used by internal components that implement these roles to

exchange memory management transactions is IMPLEMENTATION DEFINED.

Table 1.6 specifies the roles each FF-A component can play in a memory management transaction.

Table 1.6: FF-A component roles in a memory management transaction

Config No.	FF-A component	Owner	Lender	Borrower	Relayer
1.	NS-Endpoint	Yes	Yes	Yes	No
2.	S-Endpoint	Yes	Yes	Yes	No
3.	SEPID	Yes	Yes	Yes	No
4.	Hypervisor	No	No	No	Yes
5.	SPM	No	No	No	Yes

Valid combinations of component roles in a transaction to donate, lend or share memory are listed in Table 1.7. A FF-A component can use one or more combinations in a memory management transaction as the Sender.

Valid combinations of component roles in a transaction to relinquish memory are listed in Table 1.8.

Table 1.7: Valid role combinations in donate, lend or share memory transactions

Config No.	Sender	Receiver	Relayer
1.	VM	VM	Hypervisor
2.	VM	NS SEPID	Hypervisor
3.	NS-Endpoint	Secure SEPID	Hypervisor (if present) and SPM
4.	NS-Endpoint	SP	Hypervisor (if present) and SPM
5.	SP	Secure SEPID	SPM
6.	SP	SP	SPM

Table 1.8: Valid role combinations in relinquish memory transactions

Config No.	Sender	Receiver	Relayer
1.	VM	VM	Hypervisor
2.	NS-SEPID	VM	Hypervisor
3.	Secure SEPID	NS-Endpoint	Hypervisor (if present) and SPM
4.	SP	NS-Endpoint	Hypervisor (if present) and SPM
5.	Secure SEPID	SP	SPM
6.	SP	SP	SPM

1.4.2 Transaction life cycle

Each transaction described in Table 1.4 takes place in three steps as follows and illustrated in Figure 1.1.

1. The Sender sends a Framework message to the Relayer to start a transaction involving one or more Receivers.
2. The Sender sends a Partition message requesting each Receiver to complete the transaction.
3. Each Receiver sends a Framework message to the Relayer to complete the transaction.

A transaction could be targeted to a *dependent* peripheral device identified by a SEPID (see the Base FF-A Specification [1]). In this case, the partition message in *step 2* is sent to the *proxy* endpoint of the device. The proxy endpoint sends a Framework message in *step 3* to validate, authorize and complete the transaction of behalf of the device.

A transaction could be targeted to an *independent* peripheral device identified by a SEPID (see the Base FF-A Specification [1]). In this case, an IMPLEMENTATION DEFINED message is sent to this device in *step 2*. The device uses an IMPLEMENTATION DEFINED mechanism to communicate with the Relayer to complete the transaction in *step 3*.

In the SPM configuration where the SPMC is co-resident with a logical SP, the Relayer, Sender and Receiver components are implemented in the same Exception level and software image. The transaction life-cycle for this configuration is as follows.

- When the SP is the Sender, it must use an IMPLEMENTATION DEFINED interface to start the transaction with the SPMC in step 1.
- When the SP is the Receiver, it could use an IMPLEMENTATION DEFINED interface to complete the transaction with the SPMC in step 3.

  Alternatively, the SPMC could deliver the Framework message sent in step 1 to the logical SP through an IMPLEMENTATION DEFINED interface. Since the SP is aware of the transaction details, it could choose to accept or reject the transaction and inform the SPMC at this step. Successful completion of step 1 by the SPMC could be treated as a successful completion of the entire transaction. Steps 2 & 3 would not be required in this case.

The choice of mechanism used by the Sender and the Receiver SP to complete the memory management transaction is IMPLEMENTATION DEFINED.

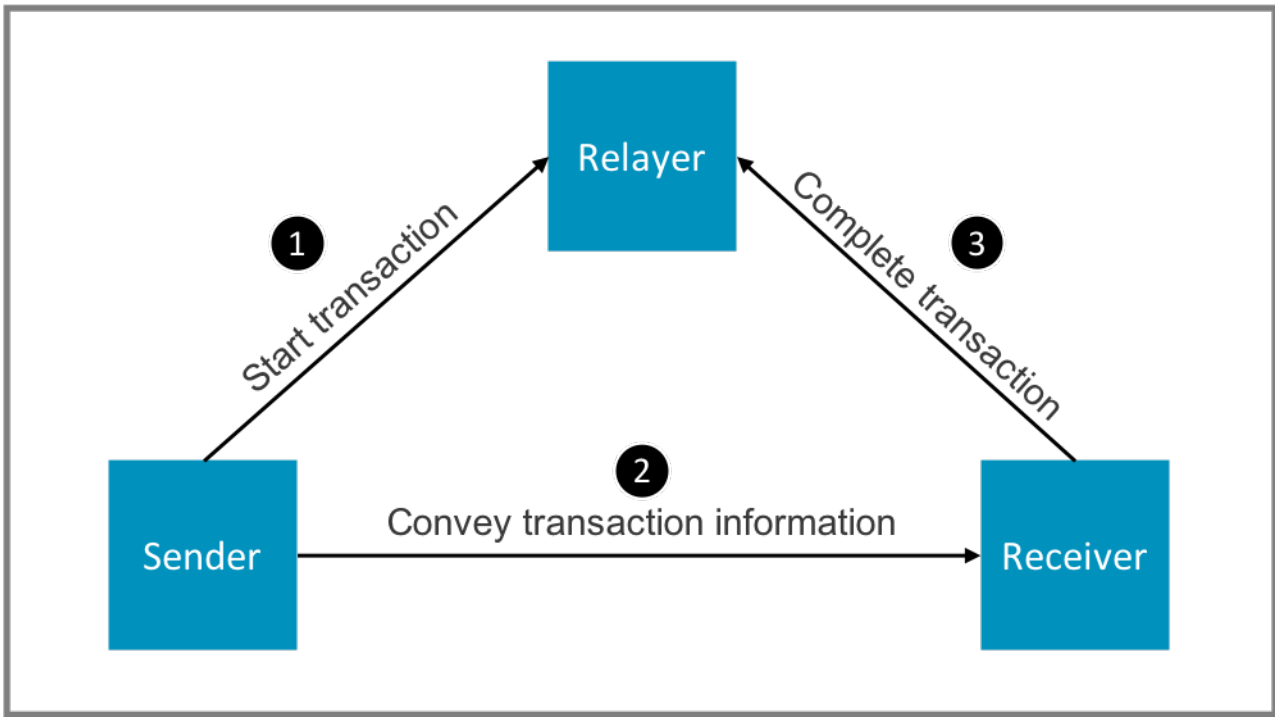

Figure 1.1: Memory management transaction lifecycle

1.5 Donate memory transaction

This transaction is used to transfer the ownership of a memory region from the endpoint that owns it to another endpoint. A list of valid combinations of roles played by various FF-A components in this transaction is specified in Table 1.7.

1.5.1 Donate memory state machine

Table 1.9 describes the state machine for donating a memory region from the perspective of two endpoints *A & B*. *A* owns the memory region. It attempts to donate the memory region to *B*. Valid and invalid state transitions in response to this transaction have been listed.

In each valid transition,

- *A* loses both ownership and access to the memory region and enters the *!Owner-NA* state.
- *B* gains ownership and exclusive access to the memory region and enters the *Owner-EA* state.

Table 1.9: Donate memory transaction state machine

No.	Current Endpoint A state	Current Endpoint B state	Next Endpoint A state	Next Endpoint B state	Description
1	Owner-EA	!Owner-NA	!Owner-NA	Owner-EA	• Owner has exclusive access to the memory region and transfers ownership to endpoint B.
2	Owner-NA	!Owner-NA	Error	–	• Owner does not have exclusive access to the memory region. It cannot transfer its ownership.
3	Owner-NA	!Owner-SA	Error	–	• Owner has lent access to the memory region to endpoint B and possibly other endpoints. It cannot transfer its ownership.
4	Owner-SA	!Owner-NA	Error	–	• Owner has shared access to the memory region with one or more endpoints. It cannot transfer its ownership.
5	Owner-SA	!Owner-SA	Error	–	• Owner has shared access to the memory region with endpoint B and possibly other endpoints. It cannot transfer its ownership.

1.5.2 Donate memory transaction lifecycle

This transaction takes place as follows (also see 1.4.2 *Transaction life cycle*).

1. The Owner uses the *FFA_MEM_DONATE* interface to describe the memory region and convey the identity of the Receiver to the Relayer as specified in Table 1.20. This interface is described in 2.1 *FFA_MEM_DONATE*.
2. If the Receiver is a *PE endpoint* or a *SEPID* associated with a dependent peripheral device,

1. The Owner uses a Partition message to request the Receiver to retrieve the donated memory region. This message contains a description of the memory region relevant to the Receiver in the form of a globally unique *Handle* (see 1.9.2 *Memory region handle*).
2. The Receiver uses the *FFA_MEM_RETRIEVE_REQ* and *FFA_MEM_RETRIEVE_RESP* interfaces to map the memory region in its translation regime and complete the transaction. These interfaces are described in 2.4 *FFA_MEM_RETRIEVE_REQ* & 2.5 *FFA_MEM_RETRIEVE_RESP* respectively.

   In case of an error, the Sender can abort the transaction before the Receiver retrieves the memory region by calling the *FFA_MEM_RECLAIM* ABI (see 2.7 *FFA_MEM_RECLAIM*).
3. If the Receiver is a *SEPID* associated with an independent peripheral device, an IMPLEMENTATION DEFINED mechanism is used by the Sender and Relayer to map and describe the memory region to the Receiver (see 2.1.1 *Component responsibilities for FFA_MEM_DONATE*).

1.5.3 Donate memory transaction lifecycle example

Figure 1.2 illustrates a transaction to transfer a memory region from one PE endpoint to another.

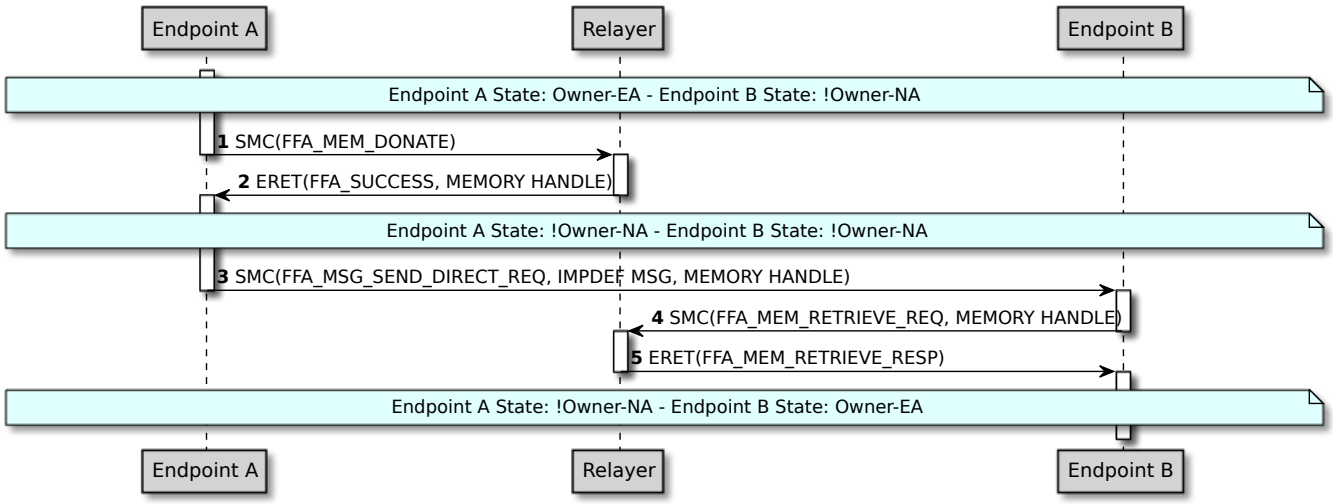

Figure 1.2: Donate Memory Transaction Lifecycle

1.6 Lend memory transaction

This transaction is used by an Owner to transfer its access to a memory region to one or more Borrowers.

- If the region is lent to a single Borrower, it is granted exclusive access to it.
- If the region is lent to more than one Borrower, they are granted shared access to it.

A list of valid combinations of roles played by various FF-A components in this transaction is specified in Table 1.7.

1.6.1 Lend memory transaction state machine

Table 1.10 describes the state machine for lending a memory region from the perspective of two components *A* & *B*. *A* owns the memory region. It attempts to relinquish its access to the memory region and grant shared or exclusive access to it to *B*. Valid and invalid state transitions in response to this transaction have been listed.

Table 1.10: Lend memory transaction state machine

No.	Current Endpoint A state	Current Endpoint B state	Next Endpoint A state	Next Endpoint B state	Description
1	Owner-EA	!Owner-NA	Owner-LA	!Owner-EA or !Owner-SA	• Owner has exclusive access to the memory region and relinquishes access to it to one or more Borrowers including endpoint B.
2	Owner-NA	!Owner-NA	Owner-LA	!Owner-EA or !Owner-SA	• Owner has no access to the memory region and grants access to it to one or more Borrowers including endpoint B.
3	Owner-NA	!Owner-EA	Error	–	• Owner has already lent the memory region to endpoint B with exclusive access.
4	Owner-NA	!Owner-SA	Error	–	• Owner has already lent the memory region to endpoint B and other endpoints.
5	Owner-SA	!Owner-NA	Error	–	• Owner has already shared the memory region with one or more endpoints. It cannot lend it to endpoint B.
6	Owner-SA	!Owner-SA	Error	–	• Owner has already shared the memory region with endpoint B and possibly other endpoints.

1.6.2 Lend memory transaction lifecycle

This transaction takes place as follows (also see 1.4.2 *Transaction life cycle*).

1. The Lender uses the *FFA_MEM_LEND* interface to describe the memory region and convey the identities of the Borrowers to the Relayer as specified in Table 1.20. This interface is described in 2.2 *FFA_MEM_LEND*.
2. If a Borrower is a *PE endpoint* or a *SEPID* associated with a dependent peripheral device,
    1. The Lender uses a Partition message to request each Borrower to retrieve the lent memory region. This message contains a description of the memory region relevant to the Borrower in the form of a globally unique *Handle* (see 1.9.2 *Memory region handle*).
    2. Each Borrower uses the *FFA_MEM_RETRIEVE_REQ* and *FFA_MEM_RETRIEVE_RESP* interfaces to map the memory region in its translation regime and complete the transaction. These interfaces are described in 2.4 *FFA_MEM_RETRIEVE_REQ* & 2.5 *FFA_MEM_RETRIEVE_RESP* respectively.
3. If the Borrower is a *SEPID* associated with an independent peripheral device, an IMPLEMENTATION DEFINED mechanism is used by the Lender and Relayer to map and describe the memory region to the Borrower (see 2.2.1 *Component responsibilities for FFA_MEM_LEND*).
4. In case of an error, the Lender can abort the transaction before the Borrower retrieves the memory region by calling the *FFA_MEM_RECLAIM* ABI (see 2.7 *FFA_MEM_RECLAIM*).

1.6.3 Lend memory transaction lifecycle example

Figure 1.3 illustrates a transaction to lend a memory region from one PE endpoint to another.

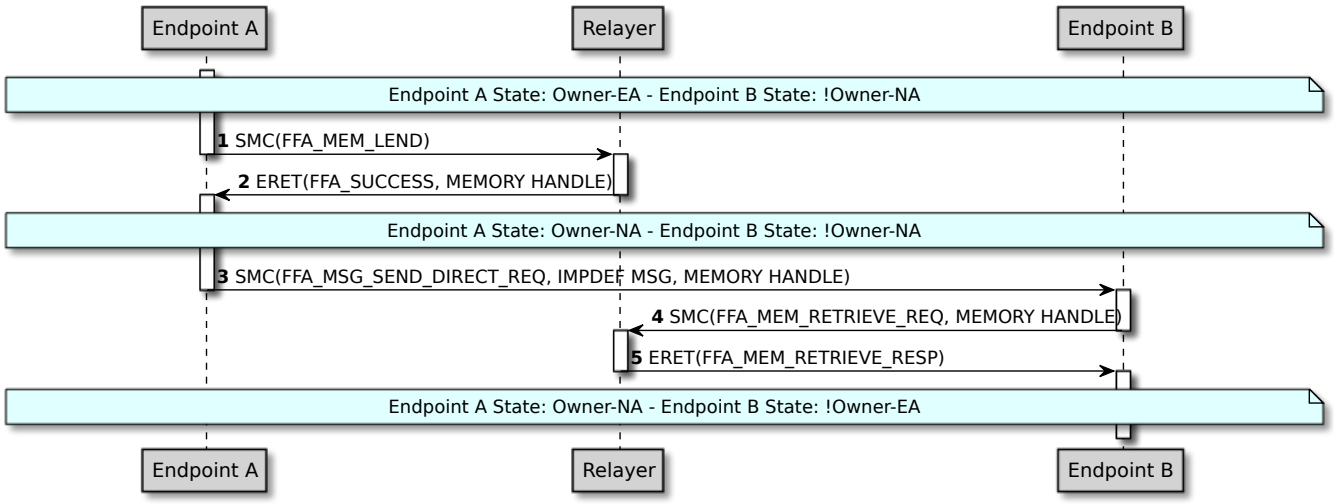

Figure 1.3: Lend Memory Transaction Lifecycle

1.7 Share memory transaction

This transaction is used by an Owner of a memory region to share access to it with one or more Borrowers.

A list of valid combinations of roles played by various FF-A components in this transaction is specified in Table 1.7.

1.7.1 Share memory transaction state machine

Table 1.11 describes the state machine for sharing a memory region from the perspective of two components *A* & *B*. *A* owns the memory region. It attempts to share the memory region with *B*. Valid and invalid state transitions in response to this transaction have been listed.

Table 1.11: Share memory transaction state machine

No.	Current Endpoint A state	Current Endpoint B state	Next Endpoint A state	Next Endpoint B state	Description
1	Owner-EA	!Owner-NA	Owner-SA	!Owner-SA	• Owner has exclusive access to the memory region and grants access to it to one or more Borrowers including endpoint B.
2	Owner-NA	!Owner-NA	Error	–	• Owner has already lent the memory region to one or more endpoints. It cannot share it with endpoint B.
3	Owner-NA	!Owner-EA	Error	–	• Owner has already lent the memory region to endpoint B with exclusive access.
4	Owner-NA	!Owner-SA	Error	–	• Owner has already lent the memory region to endpoint B and other endpoints.
5	Owner-SA	!Owner-NA	Error	–	• Owner has already shared the memory region with one or more endpoints. It cannot share it with endpoint B.
6	Owner-SA	!Owner-SA	Error	–	• Owner has already shared the memory region with endpoint B and possibly other endpoints.

1.7.2 Share memory transaction lifecycle

This transaction takes place as follows (also see 1.4.2 *Transaction life cycle*).

1. The Lender uses the *FFA_MEM_SHARE* interface to describe the memory region and convey the identities of the Borrowers to the Relayer as specified in Table 1.20. This interface is described in 2.3 *FFA_MEM_SHARE*.
2. If a Borrower is a *PE endpoint* or a *SEPID* associated with a dependent peripheral device,

1. The Lender uses a Partition message to request each Borrower to retrieve the shared memory region. This message contains a description of the memory region relevant to the Borrower in the form of a globally unique *Handle* (see 1.9.2 *Memory region handle*).
2. Each Borrower uses the *FFA_MEM_RETRIEVE_REQ* and *FFA_MEM_RETRIEVE_RESP* interfaces to map the memory region in its translation regime and complete the transaction. These interfaces are described in 2.4 *FFA_MEM_RETRIEVE_REQ* & 2.5 *FFA_MEM_RETRIEVE_RESP* respectively.

3. If the Borrower is a *SEPID* associated with an independent peripheral device, an IMPLEMENTATION DEFINED mechanism is used by the Lender and Relayer to map and describe the memory region to the Borrower (see 2.3.1 *Component responsibilities for FFA_MEM_SHARE*).
4. In case of an error, the Lender can abort the transaction before the Borrower retrieves the memory region by calling the *FFA_MEM_RECLAIM* ABI (see 2.7 *FFA_MEM_RECLAIM*).

1.7.3 Share memory transaction lifecycle example

Figure 1.4 illustrates a transaction to share a memory region between two PE endpoints.

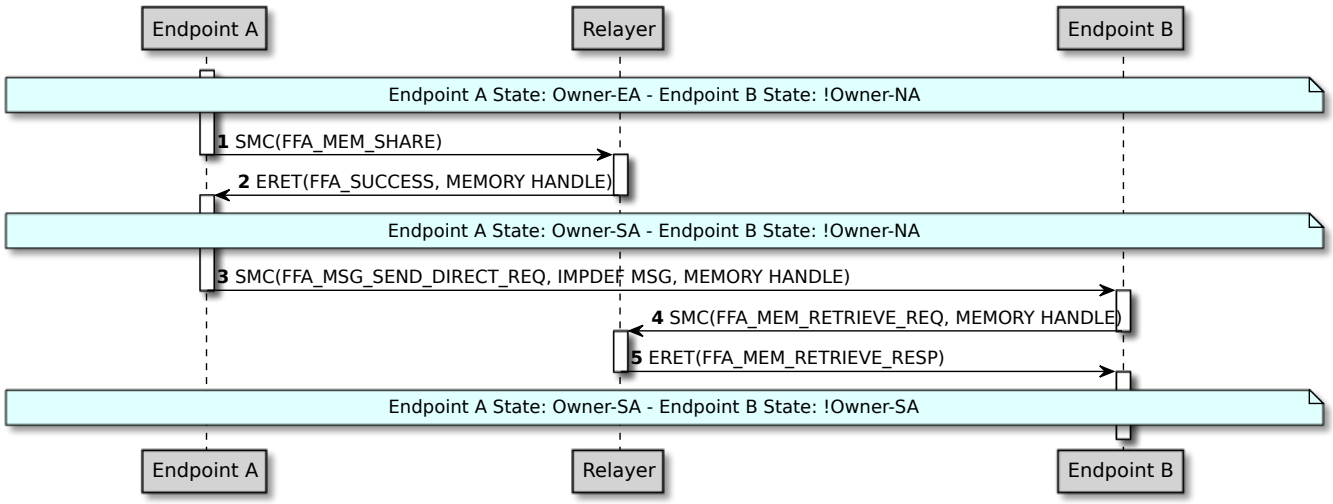

Figure 1.4: Share Memory Transaction Lifecycle

1.8 Reclaim memory transaction

This transaction is used by a Lender to reclaim exclusive access to a memory region that it had previously shared or lent to one or more Borrowers. The Lender starts this transaction by requesting each Borrower to relinquish access to the memory region e.g. by sending a partition message. The Lender reclaims access once all Borrowers have relinquished their access.

A list of valid combinations of roles played by various FF-A components in this transaction is specified in Table 1.8.

1.8.1 Reclaim memory access state machine

Table 1.12 describes the state machine for reclaiming a memory region from the perspective of two components *A* & *B*. *A* owns the memory region and could have lent or shared it with *B*. Alternatively, *B* might not have access to the memory region. *B* attempts to relinquish access to this memory region. Valid and invalid state transitions in response to this transaction have been listed.

Table 1.12: Reclaim memory state machine

No.	Current Endpoint A state	Current Endpoint B state	Next Endpoint A state	Next Endpoint B state	Description
1	Owner-EA	!Owner-NA	Error	–	• The Owner tries to reclaim access to a memory region that it already has exclusive access to.
2	Owner-NA	!Owner-NA	Error	–	• The Owner tries to reclaim access to a memory region that one or more other Borrower still have shared or exclusive access to.
3	Owner-LA	!Owner-EA	Owner-EA or Owner-NA	!Owner-NA	• The Owner reclaims the level of access it originally had on the memory region after Endpoint B relinquishes its exclusive access.
4	Owner-LA	!Owner-SA	Owner-EA or Owner-NA	!Owner-NA	• Endpoint B relinquishes access to the memory region that it shares with other Borrowers. The Owner reclaims the level of access it originally had on it once all Borrowers have relinquished access.
5	Owner-SA	!Owner-NA	Error	–	• The Owner tries to reclaim access to a memory region currently shared with one or more other Borrowers.

No.	Current Endpoint A state	Current Endpoint B state	Next Endpoint A state	Next Endpoint B state	Description
6	Owner-SA	!Owner-SA	Owner-EA	!Owner-NA	• Endpoint B relinquishes access to the memory region that it shares with the Owner and possibly other Borrowers. The Owner reclaims exclusive access once all Borrowers have relinquished access.

1.8.2 Reclaim memory transaction lifecycle

This transaction takes place as follows (also see 1.4.2 *Transaction life cycle*). It is assumed that the memory region was originally lent or shared by the Lender to the Borrowers. This transaction must not be used on a memory region owned by an endpoint.

1. If a Borrower is a *PE endpoint* or a *SEPID* associated with a dependent peripheral device,
    1. The Lender requests each Borrower to relinquish access to the memory region e.g. by specifying its *Handle* (see 1.9.2 *Memory region handle*).
    2. Each Borrower uses the *FFA_MEM_RELINQUISH* interface (see 2.6 *FFA_MEM_RELINQUISH*) to unmap the memory region from its translation regime by using the corresponding Handle. This could be done in response to the request from the Lender or independently.
    3. Each Borrower uses a Partition message to inform the Lender that it has relinquished access to the memory region.

    In case of an error, the Borrower can abort the transaction before the Lender reclaims the memory region by calling the *FFA_MEM_RETRIEVE_REQ* ABI (see 2.4 *FFA_MEM_RETRIEVE_REQ*).
2. If the Borrower is a *SEPID* associated with an independent peripheral device,
    1. The Lender could use an IMPLEMENTATION DEFINED mechanism to request each Borrower to relinquish access to the memory region.
    2. Each Borrower uses an IMPLEMENTATION DEFINED mechanism to request the Relayer to unmap the memory region from its translation regime (see 2.7.1 *Component responsibilities for FFA_MEM_RECLAIM*). This could be done in response to the message from the Lender or independently.
    3. Each Borrower uses an IMPLEMENTATION DEFINED mechanism to inform the Lender that it has relinquished access to the memory region.
3. Once all Borrowers have relinquished access to the memory region, the Lender uses the *FFA_MEM_RECLAIM* interface to reclaim the level of access it originally had on the memory region. This interface is described in 2.7 *FFA_MEM_RECLAIM*.

1.8.3 Reclaim memory transaction lifecycle example

Figure 1.5 illustrates a transaction to reclaim a memory region that is shared between two PE endpoints by requesting the borrower to relinquish its access.

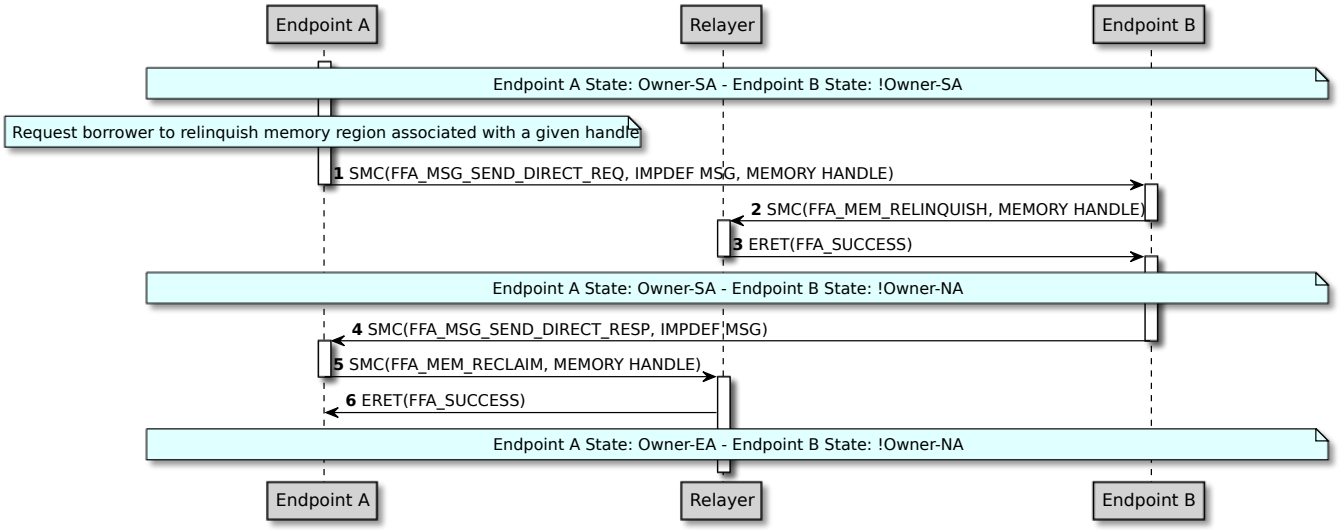

Figure 1.5: Reclaim Memory Transaction Lifecycle

1.9 Memory region description

A memory region is described in a memory management transaction either through a *Composite memory region descriptor* (see 1.9.1 *Composite memory region descriptor*) or a globally unique *Handle* (see 1.9.2 *Memory region handle*). One or both mechanisms could be used. This depends upon the step being performed in a transaction. For example,

- A Composite memory region descriptor is used to describe a memory region to a Relayer when a transaction to share, lend or donate memory is initiated by the Sender. It is also used by a Relayer to describe the memory region to a Receiver.
- A Handle is used by a Borrower to retrieve or relinquish a memory region. It is also used by an Owner to reclaim a memory region.

1.9.1 Composite memory region descriptor

A memory region is described in a memory management transaction by specifying the list and count of 4K sized pages that constitute it (see Table 1.13).

Table 1.13: Composite memory region descriptor

Field	Byte length	Byte offset	Description
Total page count	4	0	• Size of the memory region described as the count of 4K pages. • Must be equal to the sum of page counts specified in each constituent memory region descriptor. See Table 1.14.
Address range count	4	4	• Count of address ranges specified using constituent memory region descriptors.
Reserved	8	8	• Reserved (SBZ).
Address range array	–	16	• Array of address ranges specified using constituent memory region descriptors.

The list is specified by using one or more constituent memory region descriptors (see Table 1.14). Each descriptor specifies the base address and size of a virtually or physically contiguous memory region.

Table 1.14: Constituent memory region descriptor

Field	Byte length	Byte offset	Description
Address	8	–	• Base VA, PA or IPA of constituent memory region aligned to the page size (4K) granularity.
Page count	4	8	• Number of 4K pages in constituent memory region.

Field	Byte length	Byte offset	Description
Reserved	4	12	• Reserved (SBZ).

The pages are addressed using VAs, IPAs or PAs depending on the FF-A instance at which the transaction is taking place. This is as follows.

- VAs are used at a Secure virtual FF-A instance if the partition runs in Secure EL0 in either execution state.
- IPAs are used at a virtual FF-A instance if the partition runs in EL1 in either execution or security state.
- PAs are used at all physical FF-A instances.

Figure 1.6 describes a virtually contiguous memory region range *VA_0* of size 16K through its composite memory region descriptors at the virtual and physical FF-A instances. *VA_0* was allocated through a dynamic memory management mechanism inside an endpoint for example, malloc. It is composed of:

- Two constituent IPA regions *IPA_0* and *IPA_1* of size 8K each at the virtual FF-A instance.
- *IPA_0* is comprised of two PA regions *PA_0* and *PA_1* at the physical FF-A instance. Each PA region is of size 4K.
- *IPA_1* is comprised of two PA regions *PA_2* and *PA_3* at the physical FF-A instance. Each PA region is of size 4K.

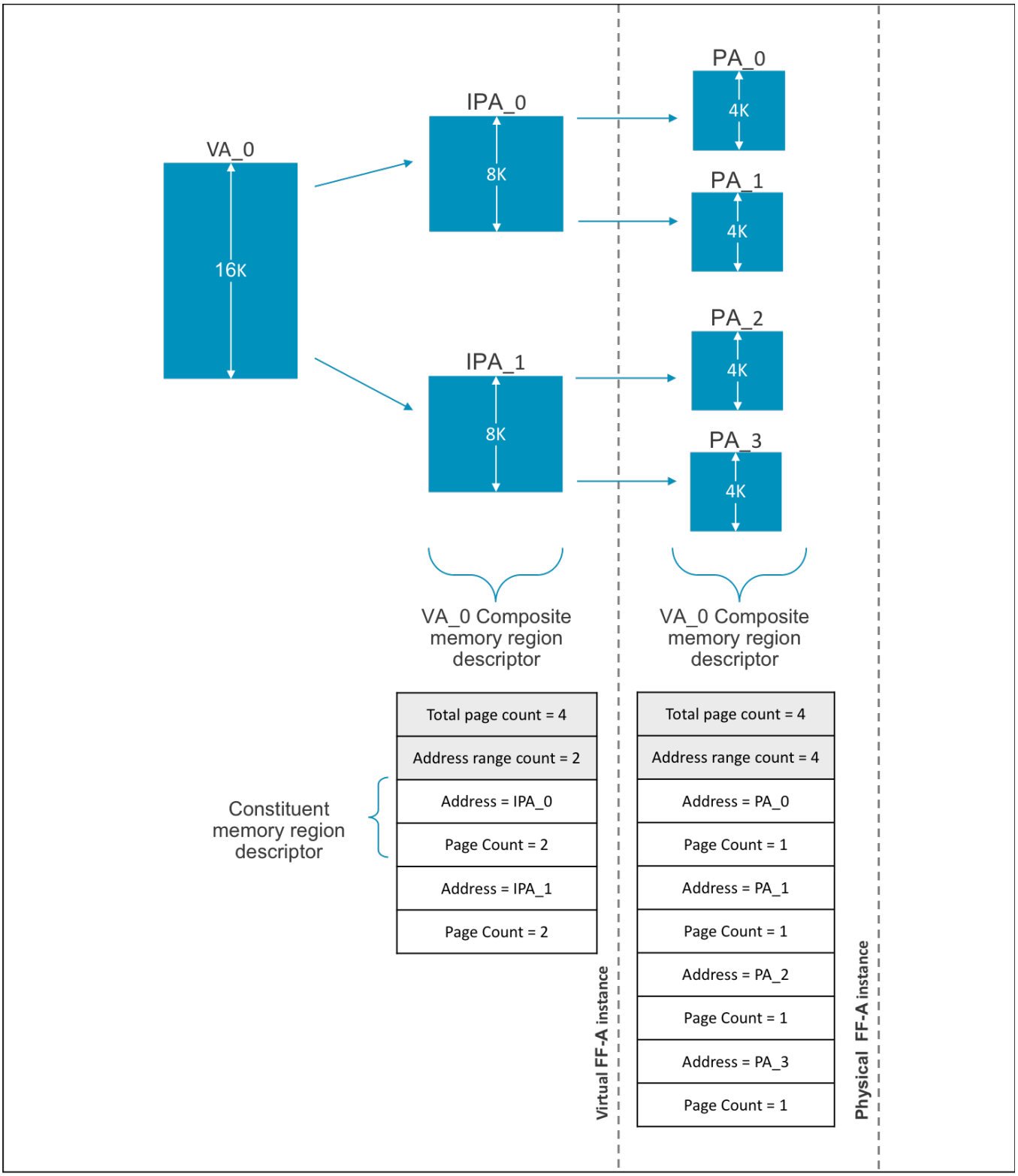

Figure 1.6: Example memory region description

1.9.2 Memory region handle

- A 64-bit *Handle* is used to identify a composite memory region description for example, *VA_0* described in Figure 1.6
- The *Handle* is allocated by the *Relayer* as follows.
  - The SPMC must allocate the Handle if every Receiver participating in the memory management transaction is an SP or SEPID associated with a Secure Stream ID in the SMMU.
  - The Hypervisor must allocate the Handle if every Receiver participating in the memory management transaction is a VM or SEPID associated with a Non-secure Stream ID in the SMMU.
  - Either the Hypervisor or the SPMC could allocate the Handle in all other cases (see 1.11.1 *Handle usage*).
- A *Handle* is allocated once a transaction to lend, share or donate memory is successfully initiated by the *Owner*.
- Each *Handle* identifies a single unique composite memory region description that is, there is a 1:1 mapping between the two.
- A *Handle* is freed by the *Relayer* as follows.
  - In a lend or share transaction the *handle* is freed after a successful reclaim transaction by its *Owner*.
  - In a donate transaction the *Handle* is freed at the end of a successful retrieval by a Receiver.
- Encoding of a *Handle* is as follows.
  - Bit[63]: Handle allocator.
    - b'0: Allocated by SPM.
    - b'1: Allocated by Hypervisor.
  - Bit[62:0]: IMPLEMENTATION DEFINED.
  - The value *0xFFFFFFFFFFFFFFFF i.e. each bit in Bit[63:0] set to b'1* is reserved as an invalid *Handle*.
- A *Handle* must be encoded as a register parameter in any ABI that requires it as follows.
  - Two 32-bit general-purpose registers must be used such that if *Rx* and *Ry* are used, such that $x < y$,
    - *Rx = Handle[31:0]*.
    - *Ry = Handle[63:32]*.

1.10 Memory region properties

The properties of a memory region are as follows.

- *Instruction and data access permissions* describe the type of access permitted on the memory region.
- *One or more endpoint IDs* that have access to the memory region specified by a combination of access permissions and memory region attributes.
- *Memory region attributes* control the memory type, accesses to the caches, and whether the memory region is Shareable and therefore is coherent.

There is a 1:1 association between an endpoint and the permissions with which it can access a memory region. This is specified in Table 1.15.

Table 1.15: Memory access permissions descriptor

Field	Byte length	Byte offset	Description
Endpoint ID	2	–	• 16-bit ID of endpoint to which the memory access permissions apply.
Memory access permissions	1	2	• Permissions used to access a memory region. – bits[7:4]: Reserved (SBZ). – bits[3:2]: Instruction access permission. ∗ b'00: Not specified and must be ignored. ∗ b'01: Not executable. ∗ b'10: Executable. ∗ b'11: Reserved. Must not be used. – bits[1:0]: Data access permission. ∗ b'00: Not specified and must be ignored. ∗ b'01: Read-only. ∗ b'10: Read-write. ∗ b'11: Reserved. Must not be used.
Flags	1	3	• ABI specific flags as described in 1.10.1 *ABI-specific flags usage*.

Table 1.16 specifies the data structure that is used in memory management transactions to create an association between an endpoint, memory access permissions and a composite memory region description.

This data structure must be included in other data structures that are used in memory management transactions instead of being used as a stand alone data structure (see 1.11 *Memory transaction descriptor*). A composite memory region description is referenced by specifying an offset to it as described in Table 1.16. This enables one or more endpoints to be associated with the same memory region but with different memory access permissions for example, SP0 could have *RO* data access permission and SP1 could have *RW* data access permission to the same memory region.

Table 1.16: Endpoint memory access descriptor

Field	Byte length	Byte offset	Description
Memory access permissions descriptor	4	–	• Memory access permissions descriptor as specified in Table 1.15.
Composite memory region descriptor offset	4	4	• Offset to the composite memory region descriptor to which the endpoint access permissions apply (see Table 1.13). • Offset must be calculated from the base address of the data structure this descriptor is included in. • An offset value of 0 indicates that the endpoint access permissions apply to a memory region description identified by the *Handle* parameter specified in the data structure that includes this one.
IMPLEMENTATION DEFINED value	16	8	• IMPLEMENTATION DEFINED information.
Reserved	8	24	• Reserved (SBZ).

1.10.1 ABI-specific flags usage

An endpoint can specify properties specific to the memory management ABI being invoked through this field.

In this version of the Framework, the *Flags* field is Reserved (MBZ) in an invocation of the following ABIs.

- FFA_MEM_DONATE.
- FFA_MEM_SHARE.
- FFA_MEM_LEND.

The *Flags* field must be encoded by the Receiver and the Relayer as specified in Table 1.17 in an invocation of the following ABIs.

- FFA_MEM_RETRIEVE_REQ.
- FFA_MEM_RETRIEVE_RESP.

The Relayer must return *INVALID_PARAMETERS* if the *Flags* field has been incorrectly encoded.

Table 1.17: Flags usage in FFA_MEM_RETRIEVE_REQ and FFA_MEM_RETRIEVE_RESP ABIs

Field	Description
Bit[0]	• Non-retrieval Borrower flag. – In a memory management transaction with multiple Borrowers, during the retrieval of the memory region, this flag specifies if the memory region must be or was retrieved on behalf of this endpoint or if the endpoint is another Borrower. ∗ b'0: Memory region must be or was retrieved on behalf of this endpoint. ∗ b'1: Memory region must not be or was not retrieved on behalf of this endpoint. It is another Borrower of the memory region. – This field MBZ if this endpoint: ∗ Is the only PE endpoint Borrower/Receiver in the transaction. ∗ Is a *Stream endpoint* and the caller of the *FFA_MEM_RETRIEVE_REQ* ABI is its *proxy endpoint*.
Bit[7:1]	• Reserved (SBZ).

1.10.2 Data access permissions usage

An endpoint could have either *Read-only* or *Read-write* data access permission to a memory region from the highest Exception level it runs in.

- *Read-write* permission is more permissive than *Read-only* permission.
- Data access permission is specified by setting *Bits[1:0]* in Table 1.15 to the appropriate value.

This access control is used in memory management transactions as follows.

1. In a transaction to lend or share memory,
   - If the Lender's ownership and access state for the memory region is *Owner-EA*,
     – The Lender must specify the level of access that the Borrower is permitted to have on the memory region. This is done in the invocation of the *FFA_MEM_SHARE* or *FFA_MEM_LEND* ABIs.
     – The Relayer must validate the permission specified by the Lender as follows. This is done in response to an invocation of the *FFA_MEM_SHARE* or *FFA_MEM_LEND* ABIs. The Relayer must return the *DENIED* error code if the validation fails.
       ∗ At the Non-secure physical FF-A instance, an IMPLEMENTATION DEFINED mechanism is used to perform validation.
       ∗ At any virtual FF-A instance, if the endpoint is running in EL1 or S-EL1 in either Execution state, the permission specified by the Lender is considered valid only if it is the same or less permissive than the permission used by the Relayer in the *S2AP* field in the stage 2 translation table descriptors for the memory region in one of the following translation regimes:
         · Secure EL1&0 translation regime, when S-EL2 is enabled.
         · Non-secure EL1&0 translation regime, when EL2 is enabled.
       ∗ At the Secure virtual FF-A instance, if the endpoint is running in S-EL0 in either Execution state, the permission specified by the Lender is considered valid only if it is the same or less permissive than the permission used by the Relayer in the *AP[1]* field in the stage 1 translation table descriptors for the memory region in one of the following translation regimes:
         · Secure EL1&0 translation regime, when EL2 is disabled.
         · Secure PL1&0 translation regime, when EL2 is disabled.

        · Secure EL2&0 translation regime, when Armv8.1-VHE is enabled.

    - If the Lender's ownership and access state for the memory region is *Owner-NA* and the transaction is to lend memory,
        - The Lender must specify the level of access that the Borrower is permitted to have on the memory region. This is done in the invocation of the *FFA_MEM_LEND* ABI.
        - The Relayer must perform the following checks. The Relayer must return the *DENIED* error code if the validation fails.
            * The Hypervisor must use an IMPLEMENTATION DEFINED mechanism to validate the permissions specified by the Lender for a Borrower.
            * The Hypervisor must forward the ABI invocation to the SPMC. The SPMC must use an IMPLEMENTATION DEFINED mechanism to validate the permissions specified by the Lender for a Borrower SP.
    - If the Borrower is an independent peripheral device, then the validated permission is used to map the memory region into the address space of the device.
    - The Borrower (if a PE or Proxy endpoint) must specify the level of access that it would like to have on the memory region.

      In a transaction to share or lend memory with more than one Borrower, each Borrower (if a PE or Proxy endpoint) must specify the data access permission that each other Borrower has on the memory region. This must match the permission specified by the Lender to the Relayer for each other Borrower.

      This is done while invoking the *FFA_MEM_RETRIEVE_REQ* ABI.
    - The Relayer must validate the permissions, if specified by the Borrower (if a PE or Proxy endpoint) in response to an invocation of the *FFA_MEM_RETRIEVE_REQ* ABI.
        - If the Lender's ownership and access state for the memory region was *Owner-EA* at the start of a transaction to lend or share memory, the Relayer must ensure that the permission of the Borrower is the same or less permissive than the permission that was specified by the Lender and validated by the Relayer.
        - If the Lender's ownership and access state for the memory region was *Owner-NA* at the start of a transaction to lend memory, the Relayer must use an IMPLEMENTATION DEFINED mechanism to validate the permission specified by the Borrower and Lender.

      In a transaction to share or lend memory with more than one Borrower, the Relayer must ensure that each Borrower specifies the data access permission that every other Borrower participating in the transaction has on the memory region. The permission for each Borrower must match the permission that was specified by the Lender to the Relayer.

      The Relayer must return the *DENIED* error code if the validation fails.

2. In a transaction to donate memory,
    - Whether the Owner is allowed to specify the level of access that the Receiver is permitted to have on the memory region depends on the type of Receiver.
        - If the Receiver is a PE or Proxy endpoint, the Owner must not specify the level of access.
        - If the Receiver is an independent peripheral device, the Owner could specify the level of access.

      The Owner must specify its choice in an invocation of the *FFA_MEM_DONATE* ABI.
    - The value of data access permission field specified by the Owner must be interpreted by the Relayer as follows. This is done in response to an invocation of the *FFA_MEM_DONATE* ABI.
        - If the Receiver is a PE or Proxy endpoint, the Relayer must return *INVALID_PARAMETERS* if the value is not *b'00*.

  - If the Receiver is an independent peripheral device and the value is not *b'00*, the Relayer must take one of the following actions.
    * Return *DENIED* if the permission is determined to be invalid through an IMPLEMENTATION DEFINED mechanism.
    * Use the permission specified by the Owner to map the memory region into the address space of the device.
  - If the Receiver is an independent peripheral device and the value is *b'00*, the Relayer must determine the permission value through an IMPLEMENTATION DEFINED mechanism.

- The Receiver (if a PE or Proxy endpoint) should specify the level of access that it would like to have on the memory region. This is done while invoking the *FFA_MEM_RETRIEVE_REQ* ABI.
- The Relayer must validate the permission specified by the Receiver to ensure that it is the same or less permissive than the permission determined by the Relayer through an IMPLEMENTATION DEFINED mechanism. This is done in response to an invocation of the *FFA_MEM_RETRIEVE_REQ* ABI. The Relayer must return the *DENIED* error code if the validation fails.
  - For example, consider the following sequence,

    1. The manifest of an SP0 specifies RO permission for memory region *memA*.
    2. SP0 donates *memA* to SP1.
    3. SP1 donates *memA* back to SP0.
    4. SP0 retrieves *memA* with RW permission.

    In this sequence, SP0 is able to escalate its permission on *memA* above what was prescribed in its manifest. The SPMC could mitigate against this threat by,

    * Recording the permission of SP0 on *memA* at the time of its initialization.
    * Ensuring that the permission of SP0 on *memA* does not exceed the recorded permission in a subsequent transaction to donate *memA* back to SP0 as described in the above sequence.

3. The Relayer must specify the permission that was used to map the memory region in the translation regime of the Receiver or Borrower. This must be done in an invocation of the *FFA_MEM_RETRIEVE_RESP* ABI.
4. In a transaction to relinquish memory that was lent to one or more Borrowers,
   - If the ownership and access state of the memory region was *Owner-EA* when it was lent, it must be mapped back into the translation regime of the Lender with the same data access permission that was used when it was lent.
   - If the ownership and access state of the memory region was *Owner-NA* when it was lent, it must not be mapped back into the translation regime of the Lender.

   This must be done in response to an invocation of the *FFA_MEM_RECLAIM* ABI.

1.10.3 Instruction access permissions usage

An endpoint could have either *Execute (X)* or *Execute-never (XN)* instruction access permission to a memory region from the highest Exception level it runs in.

- *Execute* permission is more permissive than *Execute-never* permission.
- Instruction access permission is specified by setting *Bits[3:2]* in Table 1.15 to the appropriate value.

This access control is used in memory management transactions as follows.

1. Only XN permission must be used in the following transactions.
   - In a transaction to share memory with one or more Borrowers.
   - In a transaction to lend memory to more than one Borrower.

   Bits[3:2] in Table 1.15 must be set to *b'00* as follows.

- By the Lender in an invocation of *FFA_MEM_SHARE* or *FFA_MEM_LEND* ABIs.
- By the Borrower in an invocation of the *FFA_MEM_RETRIEVE_REQ* ABI.

The Relayer must return the *INVALID_PARAMETERS* error code if this field is incorrectly encoded. Else, the Relayer must set *Bits[3:2]* in Table 1.15 to *b'01* while invoking the *FFA_MEM_RETRIEVE_RESP* ABI.

2. In a transaction to donate memory or lend memory to a single Borrower,
   - Whether the Owner/Lender is allowed to specify the level of access that the Receiver is permitted to have on the memory region depends on the type of Receiver.
     - If the Receiver is a PE or Proxy endpoint, the Owner must not specify the level of access.
     - If the Receiver is an independent peripheral device, the Owner could specify the level of access.

     The Owner must specify its choice in an invocation of the *FFA_MEM_DONATE* or *FFA_MEM_LEND* ABIs.
   - The value of instruction access permission field specified by the Owner/Lender must be interpreted by the Relayer as follows. This is done in response to an invocation of the *FFA_MEM_DONATE* or *FFA_MEM_LEND* ABIs.
     - If the Receiver is a PE or Proxy endpoint, the Relayer must return *INVALID_PARAMETERS* if the value is not *b'00*.
     - If the Receiver is an independent peripheral device and the value is not *b'00*, the Relayer must take one of the following actions.
       * Return *DENIED* if the permission is determined to be invalid through an IMPLEMENTATION DEFINED mechanism.
       * Use the permission specified by the Owner to map the memory region into the address space of the device.
     - If the Receiver is an independent peripheral device and the value is *b'00*, the Relayer must determine the permission value through an IMPLEMENTATION DEFINED mechanism.
   - The Receiver (if a PE or Proxy endpoint) specifies the level of access that it would like to have on the memory region. This must be specified in an invocation of the *FFA_MEM_RETRIEVE_REQ* ABI. The Relayer must return *INVALID_PARAMETERS* in case of an error.
   - The Relayer must validate the permission specified by the Receiver (if a PE or Proxy endpoint) to ensure that it is the same or less permissive than the permission determined by the Relayer through an IMPLEMENTATION DEFINED mechanism.
     - For example, the Relayer could deny executable access to a Borrower on a memory region of Device memory type.

     This is done in response to an invocation of the *FFA_MEM_RETRIEVE_REQ* ABI. The Relayer must return the *DENIED* error code if the validation fails.

     If the invocation of *FFA_MEM_RETRIEVE_REQ* succeeds, the Relayer must set *Bits[3:2]* in Table 1.15 to either *b'01* or *b'10'* while invoking the *FFA_MEM_RETRIEVE_RESP* ABI.
3. In a transaction to relinquish memory that was lent to one or more Borrowers, the memory region must be mapped back into the translation regime of the Lender with the same instruction access permission that was used at the start of the transaction to lend the memory region. This is done in response to an invocation of the *FFA_MEM_RECLAIM* ABI.

1.10.4 Memory region attributes usage

An endpoint can access a memory region by specifying attributes as follows.

- *Memory security state*. This could be Secure or Non-secure.

- *Memory type*. This could be Device or Normal. Device memory type could be one of the following types.
  - Device-nGnRnE.
  - Device-nGnRE.
  - Device-nGRE.
  - Device-GRE.

  The precedence rules for memory types are as follows. < should be read as *is less permissive than*.
  - Device-nGnRnE < Device-nGnRE < Device-nGRE < Device-GRE < Normal.
- *Cacheability attribute*. This could be one of the following types.
  - Non-cacheable.
  - Write-Back Cacheable.

  These attributes are used to specify both inner and outer cacheability. The precedence rules are as follows.
  - Non-cacheable < Write-Back Cacheable.
- *Shareability attribute*. This could be one of the following types.
  - Non-shareable.
  - Outer Shareable.
  - Inner Shareable.

  The precedence rules are as follows.
  - Non-Shareable < Inner Shareable < Outer shareable.

The data structure to encode memory region attributes is specified in Table 1.18.

The security state of a memory region is specified by setting *Bit[6]* in Table 1.18 to an appropriate value. The usage is described in 1.10.4.1 *Usage of NS bit*.

Other memory region attributes are specified by an endpoint by setting *Bits[5:0]* in Table 1.18 to appropriate values. The usage is described in 1.10.4.2 *Usage of other memory region attributes*.

Table 1.18: Memory region attributes descriptor

Field	Byte length	Byte offset	Description
Memory region attributes	2	–	• Attributes used to access a memory region. – bits[15:7]: Reserved (SBZ). – bits[6]: NS-bit. ∗ b'0: Secure memory. ∗ b'1: Non-secure memory. – bits[5:4]: Memory type. ∗ b'00: Not specified and must be ignored. ∗ b'01: Device memory. ∗ b'10: Normal memory. ∗ b'11: Reserved. Must not be used. – bits[3:2]: ∗ Cacheability attribute if bit[5:4] = b'10. · b'00: Reserved. Must not be used. · b'01: Non-cacheable. · b'10: Reserved. Must not be used. · b'11: Write-Back. ∗ Device memory attributes if bit[5:4] = b'01. · b'00: Device-nGnRnE. · b'01: Device-nGnRE. · b'10: Device-nGRE. · b'11: Device-GRE. ∗ Reserved (MBZ) if bit[5:4] = b'00 or b'11. – bits[1:0]: ∗ Shareability attribute if bit[5:4] = b'10. · b'00: Non-shareable. · b'01: Reserved. Must not be used. · b'10: Outer Shareable. · b'11: Inner Shareable. ∗ Reserved (MBZ) if bit[5:4] = b'01 or b'00.

1.10.4.1 Usage of NS bit

The *NS bit* is used by the SPMC to specify the security state of a memory region retrieved by a SP. The following rules govern the usage of this bit.

1. The *NS bit* is Reserved (MBZ) in an invocation of the following ABIs.

    1. FFA_MEM_DONATE
    2. FFA_MEM_LEND
    3. FFA_MEM_SHARE
    4. FFA_MEM_RETRIEVE_REQ

    The callee at any FF-A instance must return INVALID_PARAMETERS if the bit is set by the caller.

2. The *NS bit* is set to *b'1* by the Hypervisor in an invocation of the FFA_MEM_RETRIEVE_RESP ABI at the Non-secure virtual FF-A instance.

3. A Hypervisor could invoke the FFA_MEM_RETRIEVE_REQ ABI at the Non-secure physical FF-A instance (also see 2.4.3 *Support for retrieval by the Hypervisor*).

    The *NS bit* is set by the SPMC in the corresponding invocation of the FFA_MEM_RETRIEVE_RESP ABI at the Non-secure physical FF-A instance as follows.

1. The bit is set to *b'0* if the version of the Framework implemented by the Hypervisor is v1.0.
2. The bit is set to *b'1* if the version of the Framework implemented by the Hypervisor is greater than v1.0.

4. The *NS bit* is used by the SPMC in an invocation of the FFA_MEM_RETRIEVE_RESP ABI at the Secure virtual FF-A instance as described below.
    1. In response to an invocation of the FFA_MEM_RETRIEVE_REQ ABI to retrieve shared memory,
        1. *NS bit = b'1* in the following scenarios.
            1. Owner of the memory region is a NS-Endpoint.
            2. Owner of the memory region is a SP, and the region is mapped as Non-secure in the Owner's translation regime.
        2. *NS bit = b'0* in the following scenario.
            1. Owner of the memory region is a SP, and the region is mapped as Secure in the Owner's translation regime.
    2. In response to an invocation of the FFA_MEM_RETRIEVE_REQ ABI to retrieve lent memory,
        1. *NS bit = b'1* in the following scenarios.
            1. At least one Borrower of the memory region is a NS-Endpoint.
            2. Owner of the memory region is a NS-Endpoint, no Borrower of the memory region is a NS-Endpoint but the SPMC cannot change the security state of the memory region.
            3. Owner of the memory region is a SP, and the memory region is mapped as Non-secure in its translation regime.
        2. *NS bit = b'0* in any scenario where *NS bit != b'1*.
    3. In response to an invocation of the FFA_MEM_RETRIEVE_REQ ABI to retrieve donated memory,
        1. *NS bit = b'1* in the following scenarios.
            1. Owner of the memory region is a NS-Endpoint, but the SPMC cannot change the security state of the memory region.
            2. Owner of the memory region is a SP, and the memory region is mapped as Non-secure in its translation regime.
        2. *NS bit = b'0* in any scenario where *NS bit != b'1*.

    The above usage of the *NS bit* also applies at the Secure physical FF-A instance with a logical S-EL1 SP if the SPMC at EL3 implements the capability to change the security state of a memory region in the physical address space.

5. If the SPMC changes the security state of the memory region, then it must restore it back to its original security state before allowing the Owner of the memory region to successfully reclaim the memory region through an invocation of the FFA_MEM_RECLAIM ABI.

1.10.4.1.1 Discovery of NS bit usage

The following rules and guidelines govern the implementation of the *NS bit* in an SP and SPMC.

1. A v1.1 SPMC always implements support for specifying the security state of a memory region through the *NS bit* in an invocation of the FFA_MEM_RETRIEVE_RESP ABI.
2. A v1.1 SP always implements support for interpreting the *NS bit* in an invocation of the FFA_MEM_RETRIEVE_RESP ABI by the SPMC.
3. Use of the *NS bit* could be back-ported to a v1.0 SP or SPMC. The FFA_VERSION and FFA_FEATURES ABIs are used by an SP and SPMC to determine if the *NS bit* can be used by them in a compatible manner. This is described below.

1. A v1.0 SP requests use of the *NS bit* in an invocation of the FFA_FEATURES ABI as follows,
    1. By specifying the Function ID of the FFA_MEM_RETRIEVE_REQ ABI and
    2. By setting or clearing *Bit[1]* in the *Input properties* parameter.

    See the *FFA_FEATURES* ABI in the Base FF-A specification [1] for details.

    A v1.1 SP must set *Bit[1]* in the *Input properties* parameter.
2. An SPMC specifies whether it uses the *NS bit* in an invocation of the FFA_FEATURES ABI with the Function ID of the FFA_MEM_RETRIEVE_REQ ABI, by setting or clearing *Bit[1]* in the *Interface properties* return parameter. See *FFA_FEATURES* ABI in the Base FF-A specification [1] for details.

    A v1.1 SPMC always sets *Bit[1]* in the *Interface properties* return parameter.
3. Table 1.19 specifies the valid combinations of SP and SPMC versions and their support for *NS bit* usage. It also specifies the combinations in which the *NS bit* is used by both the SP and SPMC in a compatible manner.

Table 1.19: Valid SP and SPMC combinations for NS bit usage

No.	SP version	NS bit usage requested	SPMC version	NS bit usage supported	NS bit used by both
1.	v1.0	No	v1.0	No	No
2.	v1.0	No	v1.0	Yes	No
3.	v1.0	Yes	v1.0	No	No
4.	v1.0	Yes	v1.0	Yes	Yes
5.	v1.0	No	v1.1	Yes	No
6.	v1.0	Yes	v1.1	Yes	Yes
7.	v1.1	NA	v1.1	Yes	Yes

1.10.4.2 Usage of other memory region attributes

Memory region attributes are used in memory management transactions as follows.

1. In a transaction to share memory with one or more Borrowers and to lend memory to more than one Borrower,
    - If the Lender's ownership and access state for the memory region is *Owner-EA*,
        - The Lender specifies the attributes that each Borrower must access the memory region with. This is done in the invocation of the *FFA_MEM_SHARE* or *FFA_MEM_LEND* ABIs. The same attributes are used for all Borrowers.
        - The Relayer validates the attributes specified by the Lender as follows. This is done in response to an invocation of the *FFA_MEM_SHARE* or *FFA_MEM_LEND* ABIs. The Relayer must return the

DENIED error code if the validation fails.

- At the Non-secure physical FF-A instance, an IMPLEMENTATION DEFINED mechanism is used.
- At any virtual FF-A instance, if the endpoint is running in EL1 or S-EL1 in either Execution state, the attributes specified by the Lender are considered valid only if they are the same or less permissive than the attributes used by the Relayer in the stage 2 translation table descriptors for the memory region in one of the following translation regimes:
  - Secure EL1&0 translation regime, when S-EL2 is enabled.
  - Non-secure EL1&0 translation regime, when EL2 is enabled.
- At the Secure virtual FF-A instance, if the endpoint is running in S-EL0 in either Execution state, the attributes specified by the Lender are considered valid only if they are either the same or less permissive than the attributes used by the Relayer in the stage 1 translation table descriptors for the memory region in one of the following translation regimes:
  - Secure EL1&0 translation regime, when EL2 is disabled.
  - Secure PL1&0 translation regime, when EL2 is disabled.
  - Secure EL2&0 translation regime, when Armv8.1-VHE is enabled.

- If the Lender's ownership and access state for the memory region is *Owner-NA* and the transaction is to lend memory,
  - The Lender specifies the attributes that each Borrower must access the memory region with. This is done in the invocation of the *FFA_MEM_LEND* ABI. The same attributes are used for all Borrowers.
  - The Relayer must perform the following checks. The Relayer must return the *DENIED* error code if the validation fails.
    - The Hypervisor must use an IMPLEMENTATION DEFINED mechanism to validate the attributes specified by the Lender for a Borrower.
    - The Hypervisor must forward the ABI invocation to the SPMC. The SPMC must use an IMPLEMENTATION DEFINED mechanism to validate the attributes specified by the Lender for a Borrower SP.
- If the Borrower is an independent peripheral device, then the validated attributes are used to map the memory region into the address space of the device.
- The Borrower (if a PE or Proxy endpoint) should specify the attributes that it would like to access the memory region with. This is done by invoking the *FFA_MEM_RETRIEVE_REQ* ABI.
- The Relayer must validate the attributes, if specified by the Borrower (if a PE or Proxy endpoint) in response to an invocation of the *FFA_MEM_RETRIEVE_REQ* ABI.
  - If the Lender's ownership and access state for the memory region was *Owner-EA* at the start of a transaction to lend or share memory, the Relayer must ensure that the attributes of the Borrower are the same or less permissive than the attributes that were specified by the Lender and validated by the Relayer.
  - If the Lender's ownership and access state for the memory region was *Owner-NA* at the start of a transaction to lend memory, the Relayer must use an IMPLEMENTATION DEFINED mechanism to validate the attributes specified by the Borrower and Lender.

  The Relayer must return the *DENIED* error code if the validation fails.

2. In a transaction to donate memory or lend memory to a single Borrower,
   - Whether the Owner/Lender is allowed to specify the memory region attributes that the Receiver must use to access the memory region depends on the type of Receiver.
     - If the Receiver is a PE or Proxy endpoint, the Owner must not specify the attributes.

  - If the Receiver is an independent peripheral device, the Owner could specify the attributes.

  The Owner must specify its choice in an invocation of the *FFA_MEM_DONATE* or *FFA_MEM_LEND* ABIs.

- The values in the memory region attributes field specified by the Owner/Lender must be interpreted by the Relayer as follows. This is done in response to an invocation of the *FFA_MEM_DONATE* or *FFA_MEM_LEND* ABIs.
  - If the Receiver is a PE or Proxy endpoint, the Relayer must return *INVALID_PARAMETERS* if the value in *bits[5:4]* != *b'00*.
  - If the Receiver is an independent peripheral device and the value is not *b'00*, the Relayer must take one of the following actions.
    - Return *DENIED* if the attributes are determined to be invalid through an IMPLEMENTATION DEFINED mechanism.
    - Use the attributes specified by the Owner to map the memory region into the address space of the device.
  - If the Receiver is an independent peripheral device and the value is *b'00*, the Relayer must determine the attributes through an IMPLEMENTATION DEFINED mechanism.
- The Receiver (if a PE or Proxy endpoint) should specify the memory region attributes it would like to use to access the memory region. This is done while invoking the *FFA_MEM_RETRIEVE_REQ* ABI.
- The Relayer must validate the attributes specified by the Receiver (if a PE or Proxy endpoint) to ensure that they are the same or less permissive than the attributes determined by the Relayer through an IMPLEMENTATION DEFINED mechanism.

  This is done in response to an invocation of the *FFA_MEM_RETRIEVE_REQ* ABI. The Relayer must return the *DENIED* error code if the validation fails.

3. The Relayer must specify the memory region attributes that were used to map the memory region in the translation regime of the Receiver or Borrower. This must be done while invoking the *FFA_MEM_RETRIEVE_RESP* ABI.
4. In a transaction to relinquish memory that was lent to one or more Borrowers, the memory region must be mapped back into the translation regime of the Lender with the same attributes that were used at the start of the transaction to lend the memory region. This is done in response to an invocation of the *FFA_MEM_RECLAIM* ABI.
5. An Owner/Lender or a Receiver/Borrower could specify a valid combination of memory region attributes that are less permissive than the attributes used by the Relayer in the translation regime it manages for that endpoint at a virtual FF-A instance.

   Alternatively, the Hypervisor could specify a valid combination of memory region attributes to the SPMC at the Non-secure physical FF-A instance.

   The Relayer could still reject the memory region attributes in accordance to an IMPLEMENTATION DEFINED policy e.g. the SPMC could prevent a DRAM memory region from being mapped as the device memory type. The Relayer must return *INVALID_PARAMETERS* in these scenarios.

1.11 Memory transaction descriptor

Table 1.20 specifies the data structure that must be used by the Owner/Lender and a Borrower/Receiver in a transaction to donate, lend or share a memory region. It specifies the memory region description (see 1.9 *Memory region description*), properties (see 1.10 *Memory region properties*) and other transaction attributes in an invocation of the following ABIs.

- FFA_MEM_DONATE.
- FFA_MEM_LEND.
- FFA_MEM_SHARE.
- FFA_MEM_RETRIEVE_REQ.
- FFA_MEM_RETRIEVE_RESP.

The format of this data structure changed between Framework versions 1.0 and 1.1. The changes to the v1.0 memory transaction descriptor (see Table 4.17) and implementation responsibilities to maintain backward compatibility are specified in 4.2 *Changes to FF-A v1.0 data structures for forward compatibility*.

Table 1.20: Memory transaction descriptor

Field	Byte length	Byte offset	Description
Sender endpoint ID	2	0	• ID of the Owner endpoint.
Memory region attributes	2	2	• Attributes must be encoded as specified in 1.10.4 *Memory region attributes usage*. • Attribute usage is subject to validation at the virtual and physical FF-A instances as specified in 1.10.4 *Memory region attributes usage*.
Flags	4	4	• Flags must be encoded as specified in 1.11.4 *Flags usage*.
Handle	8	8	• Memory region handle in ABI invocations specified in 1.11.1 *Handle usage*.
Tag	8	16	• This field must be encoded as specified in 1.11.2 *Tag usage*.
Endpoint memory access descriptor size	4	24	• Size of each endpoint memory access descriptor in the array. See Table 1.16.
Endpoint memory access descriptor count	4	28	• Count of endpoint memory access descriptors.
Endpoint memory access descriptor array offset	4	32	• 16-byte aligned offset from the base address of this descriptor to the first element of the *Endpoint memory access descriptor array*.
Reserved	12	36	• Reserved (SBZ).

Field	Byte length	Byte offset	Description
Endpoint memory access descriptor array	–	–	• Each entry in the array must be encoded as specified in 1.11.3 *Endpoint memory access descriptor array usage*. See Table 1.16 for the encoding of the endpoint memory access descriptor.

1.11.1 Handle usage

- This field must be zero (MBZ) in an invocation of the following ABIs at a virtual FF-A instance.
  - FFA_MEM_DONATE.
  - FFA_MEM_LEND.
  - FFA_MEM_SHARE.
- The Hypervisor could allocate the *Handle* and populate it in this field in the scenarios described in 1.9.2 *Memory region handle*. This is applicable in an invocation of the following ABIs at a Non-secure physical FF-A instance.
  - FFA_MEM_DONATE.
  - FFA_MEM_LEND.
  - FFA_MEM_SHARE.

  If the SPM cannot use the *Handle* allocated by the Hypervisor, it must return INVALID_PARAMETERS.
- A successful invocation of each of the preceding ABIs returns a *Handle* (see 1.9.2 *Memory region handle*) to identify the memory region in the transaction.

  This is also applicable to an invocation of these ABIs at the Non-secure physical FF-A instance where the Hypervisor allocates the *Handle*. The SPMC must return the allocated *Handle* if it can use it.
- The Sender must convey the *Handle* to the Receiver through a Partition message.
- This field must be used by the Receiver to encode this *Handle* in an invocation of the FFA_MEM_RETRIEVE_REQ ABI.
- A Relayer must validate this field in an invocation of the FFA_MEM_RETRIEVE_REQ ABI as follows.
  - Ensure that it holds a *Handle* value that was previously allocated and has not been reclaimed by the Owner.
  - Ensure that the *Handle* identifies a memory region that was shared, lent or donated to the Receiver.
  - Ensure that the *Handle* was allocated to the Owner specified in the *Sender endpoint ID* field of the transaction descriptor.

  It must return *INVALID_PARAMETERS* if the validation fails.
- This field must be used by the Relayer to encode the *Handle* in an invocation of the FFA_MEM_RETRIEVE_RESP ABI.

1.11.2 Tag usage

- This 64-bit field must be used to specify an IMPLEMENTATION DEFINED value associated with the transaction and known to participating endpoints.
- The Sender must specify this field to the Relayer in an invocation of the following ABIs.
  - FFA_MEM_DONATE.
  - FFA_MEM_LEND.

  - FFA_MEM_SHARE.

- The Sender must convey the *Tag* to the Receiver through a Partition message.
- This field must be used by the Receiver to encode the *Tag* in an invocation of the FFA_MEM_RETRIEVE_REQ ABI.
- The Relayer must ensure the *Tag* value specified by the Receiver is equal to the value that was specified by the Sender. It must return *INVALID_PARAMETERS* if the validation fails.
- This field must be used by the Relayer to encode the *Tag* value in an invocation of the FFA_MEM_RETRIEVE_RESP ABI.

1.11.3 Endpoint memory access descriptor array usage

1.11.3.1 Sender usage

A Sender must use this field to specify one or more Receivers and the access permissions each should have on the memory region it is donating, lending or sharing through an invocation of one of the following ABIs.

- FFA_MEM_DONATE.
- FFA_MEM_LEND.
- FFA_MEM_SHARE.

The access permissions and flags are subject to validation at the virtual and physical FF-A instances as specified in 1.10.3 *Instruction access permissions usage*, 1.10.2 *Data access permissions usage* and 1.10.1 *ABI-specific flags usage*.

In an FFA_MEM_SHARE ABI invocation, the Sender could request the memory region to be mapped with different data access permission in its own translation regime. It must specify these permissions and its endpoint ID in a separate Endpoint memory access descriptor.

A Sender must describe the memory region in a composite memory region descriptor (see Table 1.13) with the following non-exhaustive list of checks.

- Ensure that the address ranges specified in the composite memory region descriptor do not overlap each other.
- *Total page count* is equal to the sum of the *Page count* fields in each *Constituent memory region descriptor*.

The offset to this descriptor from the base of Table 1.20 must be specified in the *Offset* field of the Endpoint memory access descriptor as follows.

- In an FFA_MEM_DONATE ABI invocation,
  - The *Endpoint memory access descriptor count* field in the transaction descriptor must be set to 1. This implies that the Owner must specify a single Receiver endpoint in a transaction to donate memory.
  - The *Offset* field of the Endpoint memory access descriptor must be set to the offset of the composite memory region descriptor
- In an FFA_MEM_LEND and FFA_MEM_SHARE ABI invocation,
  - The *Endpoint memory access descriptor count* field in the transaction descriptor must be set to a non-zero value. This implies that the Owner must specify at least a single Borrower endpoint in a transaction to lend or share memory.
  - The *Offset* field in the Endpoint memory access descriptor of each Borrower must be set to the offset of the composite memory region descriptor. This implies that all values of the *Offset* field must be equal.

A Sender could specify an IMPLEMENTATION DEFINED value in the IMPLEMENTATION DEFINED *value* field for each Receiver. This value must be specified by the Receiver when retrieving the memory region. The Sender must ensure it informs the Receiver of this value via an IMPLEMENTATION DEFINED mechanism such as a partition message. If this field is unused it is MBZ.

1.11.3.2 Receiver usage

A Receiver must use this field to specify the access permissions it should have on the memory region being donated, lent or shared in an invocation of the FFA_MEM_RETRIEVE_REQ ABI. This is specified in 1.10.3 *Instruction access permissions usage* and 1.10.2 *Data access permissions usage*.

- A Receiver could do this on its behalf if it is a PE endpoint.
- A Receiver could do this on the behalf of its dependent peripheral devices if it is a proxy endpoint.

A Receiver could specify the address ranges that must be used to map the memory region in its translation regime by describing them in a composite memory region descriptor. The Receiver must perform the same checks as a Sender. These checks are described in 1.11.3.1 *Sender usage*.

The offset to this descriptor from the base of Table 1.20 must be specified in the *Offset* field of the corresponding endpoint memory access descriptor in the array. This implies that all values of the *Offset* field could be different from each other.

A Receiver could let the Relayer allocate the address ranges that must be used to map the memory region in its translation regime and optionally provide an alignment hint (see *Address range alignment hint* in Table 1.22). The value 0 must be specified in the *Offset* field of the corresponding endpoint memory access descriptor in the array. This implies that the *Handle* specified in Table 1.20 must be used to identify the memory region (see 1.11.1 *Handle usage*).

A memory management transaction could be to lend or share memory with multiple Borrowers. The Receiver must use this field to specify:

- The SEPIDs and data access permissions of any dependent peripheral devices (if any) that the Receiver is a *proxy endpoint* for.

  If the Relayer must allocate the address ranges to map the memory region in the *Stream endpoints*, the value 0 must be specified in the *Offset* field of the corresponding endpoint memory access descriptor in the array.

  If the Receiver specifies the address ranges to map the memory region in the *Stream endpoints*, then it must follow the preceding guidance to specify the address ranges that must be used to map the memory region in its translation regime.

- The identity of any other Borrowers and their data access permissions on the memory region (see 1.10.2 *Data access permissions usage* and 1.10.1 *ABI-specific flags usage*), unless the *Bypass multiple borrower identification check flag* is set to 1 and the partition manager supports this functionality (see 1.11.4.2 *Bypass multiple borrower identification check*).

  The value 0 must be specified in the *Offset* field of the corresponding endpoint memory access descriptor in the array.

The Receiver must populate the IMPLEMENTATION DEFINED *value* field with the value that was specified by the Sender. If this field is unused it is MBZ.

1.11.3.3 Relayer usage

A Relayer must validate the *Endpoint memory access descriptor count* and each entry in the *Endpoint memory access descriptor array* as follows.

- The Relayer could support memory management transactions targeted to only a single Receiver endpoint.

  It must return *INVALID_PARAMETERS* if the Sender or Receiver specifies an *Endpoint memory access descriptor count != 1*.

- It must ensure that these fields have been populated by the Sender as specified in 1.11.3.1 *Sender usage* in an invocation of any of the following ABIs.
  - FFA_MEM_DONATE.
  - FFA_MEM_LEND.
  - FFA_MEM_SHARE.

The Relayer must return *INVALID_PARAMETERS* in case of an error.

- In an invocation of the FFA_MEM_LEND ABI, a memory region in the *Owner-NA* state can be lent only if the following conditions are true.

  1. The Owner and Lender of the memory region is an NS-Endpoint.
  2. Each Borrower of the memory region is an S-Endpoint.

  The Relayer must return *DENIED* if these conditions are not fulfilled.

- It must ensure that the *Endpoint ID* field in each Memory access permissions descriptor specifies a valid endpoint that it manages. The Relayer must return *INVALID_PARAMETERS* in case of an error.

  In a transaction that includes at least one VM and S-Endpoint, the role of the Relayer is collectively fulfilled by the Hypervisor and SPM. They must ensure that each endpoint specified in the Memory access permissions descriptor is managed by either one or the other.

- In an invocation of the FFA_MEM_RETRIEVE_REQ ABI,

  – It must ensure that these fields have been populated by the Receiver as specified in 1.11.3.2 *Receiver usage*.

  – If the memory region has been lent or shared with multiple Borrowers, and the *Bypass borrower identification check flag* (see 1.11.4.2 *Bypass multiple borrower identification check*) is:

    * b'0': The Relayer must ensure that the Receiver specifies the identity of each other Borrower participating in the transaction. The list of Borrowers must match the list that was specified by the Lender to the Relayer.
    * b'1': The Relayer must take IMPLEMENTATION DEFINED actions.

  – If one or more Borrowers are dependent peripheral devices, the Relayer must ensure that the Receiver is their proxy endpoint.

  – If the Receiver specifies the address ranges that must be used to map the memory region in its translation regime, the Relayer must ensure that the size of the memory region is equal to that specified by the Sender.

  The Relayer must return *INVALID_PARAMETERS* in case of an error.

- It must validate the access permissions in the *Memory access permissions descriptor* in each *Endpoint memory access descriptor* as specified in 1.10.2 *Data access permissions usage* and 1.10.3 *Instruction access permissions usage*.

- The Relayer must ensure the IMPLEMENTATION DEFINED *value* field specified by the Receiver is equal to the value that was specified by the Sender for that Borrower. It must return INVALID_PARAMETERS if the validation fails.

A Relayer must use this field in an invocation of the FFA_MEM_RETRIEVE_RESP ABI in response to successful validation of an FFA_MEM_RETRIEVE_REQ ABI invocation as follows.

- To specify the access permissions with which the memory region has been mapped in the translation regime of the Receiver.

- A Receiver could let the Relayer allocate the address ranges to map the memory region. In this case, the Relayer must describe the address ranges in a composite memory region descriptor. The Relayer must perform the same checks as a Sender. These checks are described in 1.11.3.1 *Sender usage*.

  The offset to this descriptor from the base of Table 1.20 must be specified in the *Offset* field of the corresponding endpoint memory access descriptor in the array. This implies that all values of the *Offset* field could be different from each other.

- A Receiver could specify the address ranges that must be used to map the memory region in its translation regime. The Relayer must specify the value 0 in the *Offset* field of the corresponding endpoint memory access descriptor in the array.

- To populate the IMPLEMENTATION DEFINED *value* field with the value specified by the Sender for the Receiver.

1.11.4 Flags usage

- Flags are used to govern the behavior of a memory management transaction.
- Usage of the Flags field in an invocation of the following ABIs is specified in Table 1.21.
  - FFA_MEM_DONATE.
  - FFA_MEM_LEND.
  - FFA_MEM_SHARE.
- Usage of the Flags field in an invocation of the FFA_MEM_RETRIEVE_REQ ABI is specified in Table 1.22.
- Usage of the Flags field in an invocation of the FFA_MEM_RETRIEVE_RESP ABI is specified in Table 1.23.

1.11.4.1 Zero memory flag

In some ABI invocations, the caller could set a flag to request the Relayer to zero a memory region. To do this, the Relayer must:

- Map the memory region in its translation regime once it is not mapped in the translation regime of any other FF-A component.

  The caller must ensure that the memory region fulfills the size and alignment requirements listed in the Base FF-A Specification [1] to allow the Relayer to map this memory region. It must discover these requirements by invoking the *FFA_FEATURES* interface with the function ID of the *FFA_RXTX_MAP* interface (see the Base FF-A Specification [1]).

  The Relayer must return *INVALID_PARAMETERS* if the memory region does not meet these requirements.
- Zero the memory region and perform cache maintenance such that the memory regions contents are coherent between any PE caches, system caches and system memory.
- Unmap the memory region from its translation regime before it is mapped in the translation regime of any other FF-A component.

1.11.4.2 Bypass multiple borrower identification check

When participating in a memory transaction with multiple borrowers, a Receiver could set a flag in an invocation of the *FFA_MEM_RETRIEVE_REQ* ABI to indicate to the Relayer that it does not want the identities of the Borrower IDs to be validated due to an IMPLEMENTATION DEFINED policy. Support for this feature is optional and if implemented, the action taken by a Relayer in response to the flag being set is IMPLEMENTATION DEFINED.

The Relayer must return *INVALID_PARAMETERS* if the flag is set and the Receiver is the only Borrower in the associated memory transaction, or the feature is not supported.

If this flag is not set for a given invocation, the Relayer must validate the identities of the Borrowers participating in the memory transaction specified in the memory transaction descriptor (see 1.11.3.3 *Relayer usage*).

Table 1.21: Flags usage in FFA_MEM_DONATE, FFA_MEM_LEND and FFA_MEM_SHARE ABIs

Field	Description
Bit[0]	• Zero memory flag. – In an invocation of FFA_MEM_DONATE or FFA_MEM_LEND, this flag specifies if the memory region contents must be zeroed by the Relayer after the memory region has been unmapped from the translation regime of the Owner. ∗ b'0: Relayer must not zero the memory region contents. ∗ b'1: Relayer must zero the memory region contents. – MBZ in an invocation of FFA_MEM_SHARE, else the Relayer must return *INVALID_PARAMETERS*. – MBZ if the Owner has Read-only access to the memory region , else the Relayer must return *DENIED*.
Bit[1]	• Operation time slicing flag. – In an invocation of FFA_MEM_DONATE, FFA_MEM_LEND or FFA_MEM_SHARE, this flag specifies if the Relayer can time slice this operation. ∗ b'0: Relayer must not time slice this operation. ∗ b'1: Relayer must time slice this operation. – MBZ if the Relayer does not support time slicing of memory management operations (see 4.1.3 *Time slicing of memory management operations*), else the Relayer must return *INVALID_PARAMETERS*.
Bit[31:2]	• Reserved (SBZ).

Table 1.22: Flags usage in FFA_MEM_RETRIEVE_REQ ABI

Field	Description
Bit[0]	• Zero memory before retrieval flag. – In an invocation of FFA_MEM_RETRIEVE_REQ, during a transaction to lend or donate memory, this flag is used by the Receiver to specify whether the memory region must be retrieved with or without zeroing its contents first. ∗ b'0: Retrieve the memory region irrespective of whether the Sender requested the Relayer to zero its contents prior to retrieval. ∗ b'1: Retrieve the memory region only if the Sender requested the Relayer to zero its contents prior to retrieval by setting the *Bit[0]* in Table 1.21). – MBZ in a transaction to share a memory region, else the Relayer must return *INVALID_PARAMETERS*. – If the Sender has Read-only access to the memory region and the Receiver sets Bit[0], the Relayer must return *DENIED*. – MBZ if the Receiver has previously retrieved this memory region, else the Relayer must return *INVALID_PARAMETERS* (see 2.4.2 *Support for multiple retrievals by a Borrower*).

Field	Description
Bit[1]	• Operation time slicing flag. – In an invocation of FFA_MEM_RETRIEVE_REQ, this flag specifies if the Relayer can time slice this operation. ∗ b'0: Relayer must not time slice this operation. ∗ b'1: Relayer must time slice this operation. – MBZ if the Relayer does not support time slicing of memory management operations (see 4.1.3 *Time slicing of memory management operations*), else the Relayer must return *INVALID_PARAMETERS*.
Bit[2]	• Zero memory after relinquish flag. – In an invocation of FFA_MEM_RETRIEVE_REQ, this flag specifies whether the Relayer must zero the memory region contents after unmapping it from the translation regime of the Borrower or if the Borrower crashes. ∗ b'0: Relayer must not zero the memory region contents. ∗ b'1: Relayer must zero the memory region contents. – If the memory region is lent to multiple Borrowers, the Relayer must clear memory region contents after unmapping it from the translation regime of each Borrower, if any Borrower including the caller sets this flag. – This flag could be overridden by the Receiver in an invocation of FFA_MEM_RELINQUISH (see *Flags* field in Table 2.25). – MBZ if the Receiver has Read-only access to the memory region, else the Relayer must return *DENIED*. The Receiver could be a PE endpoint or a dependent peripheral device. – MBZ in a transaction to share a memory region, else the Relayer must return *INVALID_PARAMETERS*.
Bit[4:3]	• Memory management transaction type flag. – In an invocation of FFA_MEM_RETRIEVE_REQ, this flag is used by the Receiver to either specify the memory management transaction it is participating in or indicate that it will discover this information in the invocation of FFA_MEM_RETRIEVE_RESP corresponding to this request. ∗ b'00: Relayer must specify the transaction type in FFA_MEM_RETRIEVE_RESP. ∗ b'01: Share memory transaction. ∗ b'10: Lend memory transaction. ∗ b'11: Donate memory transaction. – Relayer must return *INVALID_PARAMETERS* if the transaction type specified by the Receiver is not the same as that specified by the Sender for the memory region identified by the *Handle* value specified in the transaction descriptor.

Field	Description
Bit[9:5]	• Address range alignment hint. – In an invocation of FFA_MEM_RETRIEVE_REQ, this flag is used by the Receiver to specify the boundary, expressed as multiples of 4KB, to which the address ranges allocated by the Relayer to map the memory region must be aligned. – Bit[9]: Hint valid flag. ∗ b'0: Relayer must choose the alignment boundary. Bits[8:5] are Reserved (MBZ). ∗ b'1: Relayer must use the alignment boundary specified in Bits[8:5]. – Bit[8:5]: Alignment hint. ∗ If the value in this field is *n*, then the address ranges must be aligned to the *2*n x 4KB* boundary. – MBZ if the Receiver specifies the IPA or VA address ranges that must be used by the Relayer to map the memory region, else the Relayer must return *INVALID_PARAMETERS*. – Relayer must return *DENIED* if it is not possible to allocate the address ranges at the alignment boundary specified by the Receiver. – Relayer must return *INVALID_PARAMETERS* if a reserved value is specified by the Receiver.
Bit[10]	• Bypass multiple borrower identification check. – In an invocation of FFA_MEM_RETRIEVE_REQ, this flag is used by the Receiver to specify whether it wants the Relayer to validate the partition IDs of other Borrowers provided in the endpoint memory access descriptors. See 1.11.4.2 *Bypass multiple borrower identification check*. ∗ b'0: The Relayer must ensure the Receiver has specified the identity of all Borrowers participating in the transaction. ∗ b'1: The Relayer must take IMPLEMENTATION DEFINED actions.
Bit[31:11]	• Reserved (SBZ).

Table 1.23: Flags usage in FFA_MEM_RETRIEVE_RESP ABI

Field	Description
Bit[0]	• Zero memory before retrieval flag. – In an invocation of FFA_MEM_RETRIEVE_RESP during a transaction to lend or donate memory, this flag is used by the Relayer to specify whether the memory region was retrieved with or without zeroing its contents first. ∗ b'0: Memory region was retrieved without zeroing its contents. ∗ b'1: Memory region was retrieved after zeroing its contents. – MBZ in a transaction to share a memory region. – MBZ if the Sender has Read-only access to the memory region.
Bit[2:1]	• Reserved (SBZ).

Field	Description
Bit[4:3]	• Memory management transaction type flag. – In an invocation of FFA_MEM_RETRIEVE_RESP, this flag is used by the Relayer to specify the memory management transaction the Receiver is participating in. ∗ b'00: Reserved. ∗ b'01: Share memory transaction. ∗ b'10: Lend memory transaction. ∗ b'11: Donate memory transaction.
Bit[31:5]	• Reserved (SBZ).

1.12 Compliance requirements

This section describes the compliance requirements that must be met by an implementation of an FF-A memory management feature. These requirements specify,

1. How the presence of a feature can be discovered. e.g. support for transmitting the memory transaction descriptor in fragments in conjunction with the FFA_MEM_SHARE ABI.
2. ABIs that must be implemented at a relevant FF-A instance to correctly support a memory management feature e.g. the set of ABIs that must be implemented to enable an Owner to lend, donate or share memory.

The list of features and their compliance requirements is given below.

1. The compliance requirements for lending, sharing and donating memory are described in Table 1.24. The table columns should be read as: For *feature* and *caller role*, at the specified FF-A *instance*, *interface* is implemented with the specified *conduits*. Presence of the *interface* must be ascertained through the FFA_FEATURES ABI.
2. Support for transmission of a memory transaction descriptor in dynamically allocated buffers is discovered through the FFA_FEATURES ABI as described in 4.1.1 *Transmission of transaction descriptor in dynamically allocated buffers*.
3. Support for transmission of a memory transaction descriptor in fragments is discovered through the FFA_FEATURES ABI as described in 4.1.2 *Transmission of transaction descriptor in fragments*.
4. Support for time slicing of memory management operations is discovered through the FFA_FEATURES ABI as described in 4.1.3 *Time slicing of memory management operations*.
5. Support for NS bit usage is discovered through the FFA_FEATURES ABI as described in 1.10.4.1.1 *Discovery of NS bit usage*.
6. Support for multiple retrievals of a memory region is discovered through the FFA_FEATURES ABI as described in 2.4.2 *Support for multiple retrievals by a Borrower*.
7. Support for ABIs to get and set memory permissions (see 2.8 *FFA_MEM_PERM_GET* and 2.9 *FFA_MEM_PERM_SET*) is discovered through the FFA_FEATURES ABI.

Table 1.24: Compliance requirements for sharing, lending and donating memory

Feature	Caller role	Instance	Mandatory Interface	Conduit
Memory sharing or lending	Owner	• Secure or NS virtual • NS physical[1]	• FFA_MEM_SHARE • FFA_MEM_LEND • FFA_MEM_RECLAIM	SMC, HVC, SVC
	Borrower	• Secure or NS virtual	• FFA_MEM_RETRIEVE_REQ • FFA_MEM_RELINQUISH	SMC, HVC, SVC
			• FFA_MEM_RETRIEVE_RESP	ERET
		• Secure physical[2]	• FFA_MEM_RETRIEVE_REQ • FFA_MEM_RELINQUISH	SMC
			• FFA_MEM_RETRIEVE_RESP	ERET

[1] Interfaces are mandatory at this instance in the absence of an Hypervisor.
[2] Interfaces are mandatory at this instance between the SPMC and a logical S-EL1 SP.

Feature	Caller role	Instance	Mandatory Interface	Conduit
	Relayer	• NS physical	• FFA_MEM_SHARE • FFA_MEM_LEND • FFA_MEM_RECLAIM	SMC
		• Secure physical[3]	• FFA_MEM_SHARE • FFA_MEM_LEND • FFA_MEM_RECLAIM	ERET
Memory donation	Owner	• Secure or NS virtual	• FFA_MEM_DONATE • FFA_MEM_RECLAIM	SMC, HVC, SVC
		• Secure physical[4] • NS physical[5]	• FFA_MEM_DONATE • FFA_MEM_RECLAIM	SMC
	Receiver	• Secure or NS virtual	• FFA_MEM_RETRIEVE_REQ	SMC, HVC, SVC
			• FFA_MEM_RETRIEVE_RESP	ERET
		• Secure physical[6] • NS physical[7]	• FFA_MEM_RETRIEVE_REQ	SMC
			• FFA_MEM_RETRIEVE_RESP	ERET
	Relayer	• Non-secure physical	• FFA_MEM_DONATE • FFA_MEM_RECLAIM • FFA_MEM_RETRIEVE_REQ	SMC
			• FFA_MEM_RETRIEVE_RESP	ERET
		• Secure physical[8]	• FFA_MEM_DONATE • FFA_MEM_RECLAIM • FFA_MEM_RETRIEVE_REQ	ERET
			• FFA_MEM_RETRIEVE_RESP	SMC

[3] Interfaces are mandatory at this instance between the SPMD and a S-EL2 or S-EL1 SPMC.
[4] Interfaces are mandatory at this instance between the SPMC and a logical S-EL1 SP.
[5] Interfaces are mandatory at this instance in the absence of an Hypervisor.
[6] Interfaces are mandatory at this instance between the SPMC and a logical S-EL1 SP.
[7] Interfaces are mandatory at this instance in the absence of an Hypervisor.
[8] Interfaces are mandatory at this instance between the SPMD and a S-EL2 or S-EL1 SPMC.

Chapter 2
Memory management interfaces

2.1 FFA_MEM_DONATE

Description

- Starts a transaction to transfer of ownership of a memory region from a Sender endpoint to a Receiver endpoint.
- Transaction details are described in a memory transaction descriptor (see Table 1.20).
- Descriptor is populated in the TX buffer of the Owner by default.
- Valid FF-A instances and conduits are listed in Table 2.2.
- Syntax of this function is described in Table 2.3.
- Encoding of result parameters in the FFA_SUCCESS function is described in Table 2.4.
- Encoding of error code in the FFA_ERROR function is described in Table 2.5.

Table 2.2: FFA_MEM_DONATE instances and conduits

Config No.	FF-A instance	Valid conduits
1	Secure and Non-secure physical	SMC, ERET
2	Secure and Non-secure virtual	SMC, HVC, SVC

Table 2.3: FFA_MEM_DONATE function syntax

Parameter	Register	Value
uint32 Function ID	w0	• 0x84000071. • 0xC4000071.
uint32 Total length	w1	• Total length of the memory transaction descriptor in bytes.
uint32 Fragment length	w2	• Length in bytes of the memory transaction descriptor passed in this ABI invocation. • *Fragment length* must be <= *Total length*. • If *Fragment length* < *Total length* then see 4.1.2 *Transmission of transaction descriptor in fragments* about how the remainder of the descriptor will be transmitted.
uint32/uint64 Address	w3/x3	• Base address of a buffer allocated by the Owner and distinct from the TX buffer. See 4.1.1 *Transmission of transaction descriptor in dynamically allocated buffers*. • MBZ if the TX buffer is used.

Parameter	Register	Value
uint32 Page count	w4	• Number of 4K pages in the buffer allocated by the Owner and distinct from the TX buffer. See 4.1.1 *Transmission of transaction descriptor in dynamically allocated buffers*. • MBZ if the TX buffer is used.
Other Parameter registers	w5-w7 x5-x17	• Reserved (SBZ).

Table 2.4: FFA_SUCCESS encoding

Parameter	Register	Value
uint64 Handle	w2/w3	• Globally unique Handle to identify the memory region on successful transmission of the transaction descriptor.
Other Result registers	w4-w7 x4-x17	• Reserved (MBZ).

Table 2.5: FFA_ERROR encoding

Parameter	Register	Value
int32 Error code	w2	• INVALID_PARAMETERS. • DENIED. • NO_MEMORY. • BUSY. • ABORTED.

2.1.1 Component responsibilities for FFA_MEM_DONATE

This interface is used to initiate a transaction to donate a memory region to a single Receiver endpoint (also see 1.5.2 *Donate memory transaction lifecycle*). Only the Owner and Relayer participate in this stage of the transaction. Responsibilities of the:

- Owner are listed in 2.1.1.1 *Owner responsibilities*.
- Relayer are listed in 2.1.1.2 *Relayer responsibilities*.

The transaction descriptor could be populated in a buffer dynamically allocated by the Owner as specified in 4.1.1 *Transmission of transaction descriptor in dynamically allocated buffers*.

Transmission of the transaction descriptor in fragments must be implemented by the Owner and Relayer as specified in 4.1.2 *Transmission of transaction descriptor in fragments*.

Time slicing of this ABI invocation must be implemented by the Owner and Relayer as specified in 4.1.3 *Time slicing of memory management operations*.

2.1.1.1 Owner responsibilities

1. Must ensure it is a *PE endpoint* and Owner of the memory region.
2. Must ensure the memory region is in an access state suitable for donation (see Table 1.9).
3. Must ensure the memory region fulfills the applicable rules stated in 1.3.1 *Ownership and access rules*.
4. Must describe memory region in the descriptor specified in Table 1.20 with a single endpoint memory access descriptor (also see 1.11.3.1 *Sender usage*).
5. Must implement support for handling all error status codes that can be returned on completion of this interface.
6. If the invocation of this interface completes successfully, then must send at least the following information to the Receiver in a Partition message:
    1. Globally unique Handle returned by the Relayer.
    2. Owner endpoint ID.

    If the Receiver specified in the memory transaction descriptor is a SEPID, then the message must be sent to:

    - Either the *proxy endpoint* for the SEPID (see the Base FF-A Specification [1]) through a Partition message.
    - Or the *independent* peripheral device associated with the SEPID through an IMPLEMENTATION DEFINED mechanism.

    Provision of any other information from the transaction descriptor is IMPLEMENTATION DEFINED.

    If the Receiver rejects the request to donate memory, the Sender should use the *FFA_MEM_RECLAIM* interface with the Handle returned by the Relayer to reclaim ownership of the memory region. It must treat the memory region as being inaccessible until the *FFA_MEM_RECLAIM* invocation completes.

2.1.1.2 Relayer responsibilities

1. Must check that the RX/TX buffer pair is mapped, if the transaction descriptor is not passed in a dynamically allocated buffer as specified in 4.1.1 *Transmission of transaction descriptor in dynamically allocated buffers*, and return *INVALID_PARAMETERS* if the RX/TX buffer pair is not mapped.
2. Must validate the *Total length* input parameter to ensure that the length of the transaction descriptor does not exceed the size of the buffer it has been populated in. Must return *INVALID_PARAMETERS* in case of an error.
3. Must validate the *Sender endpoint ID* field in the transaction descriptor to ensure that the Sender is the Owner of the memory region and a *PE endpoint*. Must return *DENIED* in case of an error.
4. Must ensure that a request by an SP to donate Secure memory to a NS-Endpoint is rejected by returning the *DENIED* error code.
5. Must ensure that the memory region is in the *Owner-EA* state for the Owner (see Table 1.9). It must return *DENIED* in case of an error.
6. Must validate that the *Endpoint memory access descriptor count* & *Endpoint memory access descriptor array* fields in the transaction descriptor as specified in 1.11.3.3 *Relayer usage*.
7. Must validate the *Memory region attributes* field in the transaction descriptor as specified in 1.10.4 *Memory region attributes usage*.
8. Must validate the *Flags* field specified in the transaction descriptor as specified in 1.11.4 *Flags usage*.
9. Must validate the *Handle* field specified in the transaction descriptor as specified in 1.11.1 *Handle usage*.
10. Unmap the memory region from the translation regime of the Owner, if managed by the Relayer as specified in 1.2 *Address translation regimes*. This includes removing access to the memory region from any DMA capable devices assigned to the Owner.

11. If the Receiver is a *PE endpoint* or a *Stream endpoint* with a *proxy endpoint* managed by the Relayer, then the Relayer must:
    1. Save the transaction descriptor information so that it can be validated when retrieved through invocations of the *FFA_MEM_RETRIEVE_REQ* & *FFA_MEM_RETRIEVE_RESP* interfaces.
    2. Return *NO_MEMORY* if there is not enough memory to complete this operation.
12. If the Receiver is a *Stream endpoint* associated with an *independent* device managed by the Relayer, then the Relayer must:
    1. Allocate an IPA range and map the memory region in the translation regime of the Receiver managed by the Relayer as specified in 1.2 *Address translation regimes*.

       The mapping must be created with the memory region attributes and permissions specified in the transaction descriptor.
    2. Describe the memory region to the device using the SEPID through an IMPLEMENTATION DEFINED mechanism.
13. If the call executes successfully, the Relayer must:
    1. Ensure that the state of the memory region in the participating FF-A components is observed as follows:
       1. If the Receiver is a *PE endpoint* or a *SEPID* associated with a dependent peripheral device, then:
          - *Owner-NA* for the Owner.
          - *!Owner-NA* for the Receiver.
       2. If the Receiver is a *SEPID* associated with an independent peripheral device, then:
          - *!Owner-NA* for the Owner.
          - *Owner-EA* for the Receiver.
    2. Allocate and return a *Handle* as described in 1.9.2 *Memory region handle*.
14. If the Owner is a VM and the Receiver is an SP or SEPID associated with a Secure Stream ID, the Hypervisor must forward the memory transaction descriptor to the SPM. This must be done by invoking this interface at the Non-secure physical FF-A instance as follows.
    1. The fields of the transaction descriptor must be unchanged apart from the following exception.
       1. The memory region must be described as composed of physically addressed constituent 4K pages in one or more *Constituent memory region descriptors*.

          This must be done by performing the VA or IPA to PA translation of the memory region described by the Owner at the non-secure virtual FF-A instance.

          The order in which the address ranges are specified by the Owner must be preserved by the Hypervisor.
       2. The *Constituent memory region descriptors* must be described in a *Composite memory region descriptor* which must be referenced by the *Endpoint memory access descriptor* included in the transaction descriptor.
    2. The updated transaction descriptor must be copied into the TX buffer shared between the Hypervisor and SPM.

       If the TX buffer is busy, the Hypervisor must return *BUSY*.

       If the TX buffer is too small and it is not possible to use the optional features to transmit the descriptor listed in 4.1.2 *Transmission of transaction descriptor in fragments* and 4.1.1 *Transmission of transaction descriptor in dynamically allocated buffers*, the Hypervisor must return *NO_MEMORY*

2.2 FFA_MEM_LEND

Description

- Starts a transaction to transfer access to a memory region from its Owner to one or more Borrowers.
- Transaction details are described in a memory transaction descriptor (see Table 1.20).
- Descriptor is populated in the TX buffer of the Owner by default.
- Valid FF-A instances and conduits are listed in Table 2.7.
- Syntax of this function is described in Table 2.8.
- Encoding of result parameters in the FFA_SUCCESS function is described in Table 2.9.
- Encoding of error code in the FFA_ERROR function is described in Table 2.10.

Table 2.7: FFA_MEM_LEND instances and conduits

Config No.	FF-A instance	Valid conduits
1	Secure and Non-secure physical	SMC, ERET
2	Secure and Non-secure virtual	SMC, HVC, SVC

Table 2.8: FFA_MEM_LEND function syntax

Parameter	Register	Value
uint32 Function ID	w0	• 0x84000072. • 0xC4000072.
uint32 Total length	w1	• Total length of the memory transaction descriptor in bytes.
uint32 Fragment length	w2	• Length in bytes of the memory transaction descriptor passed in this ABI invocation. • *Fragment length* must be <= *Total length*. • If *Fragment length* < *Total length* then see 4.1.2 *Transmission of transaction descriptor in fragments* about how the remainder of the descriptor will be transmitted.
uint32/uint64 Address	w3/x3	• Base address of a buffer allocated by the Owner and distinct from the TX buffer. See 4.1.1 *Transmission of transaction descriptor in dynamically allocated buffers*. • MBZ if the TX buffer is used.

Parameter	Register	Value
uint32 Page count	w4	• Number of 4K pages in the buffer allocated by the Owner and distinct from the TX buffer. See 4.1.1 *Transmission of transaction descriptor in dynamically allocated buffers*. • MBZ if the TX buffer is used.
Other Parameter registers	w5-w7 x5-x17	• Reserved (SBZ).

Table 2.9: FFA_SUCCESS encoding

Parameter	Register	Value
uint64 Handle	w2/w3	• Globally unique Handle to identify the memory region on successful transmission of the transaction descriptor. MBZ otherwise (see 1.9.2 *Memory region handle*).
Other Result registers	w4-w7 x4-x17	• Reserved (MBZ).

Table 2.10: FFA_ERROR encoding

Parameter	Register	Value
int32 Error code	w2	• INVALID_PARAMETERS. • DENIED. • NO_MEMORY. • BUSY. • ABORTED.

2.2.1 Component responsibilities for FFA_MEM_LEND

This interface is used to initiate a transaction to lend a memory region to one or more Borrower endpoints (also see 1.6.2 *Lend memory transaction lifecycle*). Only the Lender and Relayer participate in this stage of the transaction. Responsibilities of the:

- Lender are listed in 2.2.1.1 *Lender responsibilities*.
- Relayer are listed in 2.2.1.2 *Relayer responsibilities*.

The transaction descriptor could be populated in a buffer dynamically allocated by the Lender as specified in 4.1.1 *Transmission of transaction descriptor in dynamically allocated buffers*.

Transmission of the transaction descriptor in fragments must be implemented by the Lender and Relayer as specified in 4.1.2 *Transmission of transaction descriptor in fragments*.

Time slicing of this ABI invocation must be implemented by the Lender and Relayer as specified in 4.1.3 *Time slicing of memory management operations*.

2.2.1.1 Lender responsibilities

1. Must ensure it is a *PE endpoint* and Owner of the memory region.
2. Must ensure the memory region is in an access state suitable for lending (see Table 1.10).
3. Must ensure the memory region fulfills the applicable rules stated in 1.3.1 *Ownership and access rules*.
4. Must describe memory region in the descriptor specified in Table 1.20 with an endpoint memory access descriptor for each Borrower (also see 1.11.3.1 *Sender usage*).
5. Must implement support for handling all error status codes that can be returned on completion of this interface.
6. If the invocation of this interface completes successfully, then must send at least the following information to each Borrower in a Partition message:
    1. Globally unique Handle returned by the Relayer.
    2. Lender endpoint ID.

    If the Borrower specified in the memory transaction descriptor is a SEPID, then the message must be sent to:

    - Either the *proxy endpoint* for the SEPID (see the Base FF-A Specification [1]) through a Partition message.
    - Or the *independent* peripheral device associated with the SEPID through an IMPLEMENTATION DEFINED mechanism.

    Provision of any other information from the transaction descriptor is IMPLEMENTATION DEFINED.

    If the Borrower rejects the request to lend memory, the Lender should use the *FFA_MEM_RECLAIM* interface with the Handle returned by the Relayer to revert to the level of access (either exclusive or no access) it originally had on the memory region. It must treat the memory region as being inaccessible until the *FFA_MEM_RECLAIM* invocation completes.

2.2.1.2 Relayer responsibilities

1. Must check that the RX/TX buffer pair is mapped, if the transaction descriptor is not passed in a dynamically allocated buffer as specified in 4.1.1 *Transmission of transaction descriptor in dynamically allocated buffers*, and return *INVALID_PARAMETERS* if the RX/TX buffer pair is not mapped.
2. Must validate the *Total length* input parameter to ensure that the length of the transaction descriptor does not exceed the size of the buffer it has been populated in. Must return *INVALID_PARAMETERS* in case of an error.
3. Must validate the *Sender endpoint ID* field in the transaction descriptor to ensure that the Lender is the Owner of the memory region and a *PE endpoint*. Must return *DENIED* in case of an error.
4. Must ensure that a request by an SP to lend Secure memory to a NS-Endpoint is rejected by returning the *DENIED* error code.
5. Must validate that the memory region is either in the *Owner-EA* or *Owner-NA* state for the Lender (see Table 1.10). It must return *DENIED* in case of an error.
6. Must validate that the *Endpoint memory access descriptor count* & *Endpoint memory access descriptor array* fields in the transaction descriptor as specified in 1.11.3.3 *Relayer usage*.
7. Must validate the *Memory region attributes* field in the transaction descriptor as specified in 1.10.4 *Memory region attributes usage*.
8. Must validate the *Flags* field specified in the transaction descriptor as specified in 1.11.4 *Flags usage*.
9. Must validate the *Handle* field specified in the transaction descriptor as specified in 1.11.1 *Handle usage*.
10. Unmap the memory region from the translation regime of the Lender, if managed by the Relayer as specified in 1.2 *Address translation regimes*. This must be done only if the memory region is in the *Owner-EA* state. This includes removing access to the memory region from any DMA capable devices assigned to the Lender.

11. If the Borrower is a *PE endpoint* or a *Stream endpoint* with a *proxy endpoint* managed by the Relayer, then the Relayer must:
    1. Save the transaction descriptor information so that it can be validated when retrieved through invocations of the *FFA_MEM_RETRIEVE_REQ* & *FFA_MEM_RETRIEVE_RESP* interfaces.
    2. Return *NO_MEMORY* if there is not enough memory to complete this operation.
12. If the Borrower is a *Stream endpoint* associated with an *independent* device managed by the Relayer, then the Relayer must:
    1. Allocate an IPA range and map the memory region in the translation regime of the Borrower managed by the Relayer as specified in 1.2 *Address translation regimes*.

       The mapping must be done with the memory region attributes and permissions specified in the transaction descriptor.
    2. Describe the memory region to the device using the SEPID through an IMPLEMENTATION DEFINED mechanism.
13. If the call executes successfully, the Relayer must:
    1. Ensure that the state of the memory region in the participating FF-A components is observed as follows:
       1. If a Borrower is a *PE endpoint* or a *SEPID* associated with a dependent peripheral device, then:
          - *Owner-LA* for the Lender.
          - *!Owner-NA* for the Borrower.
       2. If a Borrower is a *SEPID* associated with an independent peripheral device, then:
          - *Owner-LA* for the Lender.
          - *!Owner-EA* for the Borrower, if the count of Borrowers in the transaction is = *1*.
          - *Owner-SA* for the Borrower, if the count of Borrowers in the transaction is > *1*.
    2. Allocate and return a *Handle* as described in 1.9.2 *Memory region handle*.
14. If the Lender is a VM and the Borrower is an SP or SEPID associated with a Secure Stream ID, the Hypervisor must forward the memory transaction descriptor to the SPM. This must be done by invoking this interface at the Non-secure physical FF-A instance as follows.
    1. The fields of the transaction descriptor must be unchanged apart from the following exception.
       1. The memory region must be described as composed of physically addressed constituent 4K pages in one or more *Constituent memory region descriptors*.

          This must be done by performing the VA or IPA to PA translation of the memory region described by the Owner at the non-secure virtual FF-A instance.

          The order in which the address ranges are specified by the Lender must be preserved by the Hypervisor.
       2. The *Constituent memory region descriptors* must be described in a *Composite memory region descriptor* which must be referenced by the *Endpoint memory access descriptor* included in the transaction descriptor.
    2. The updated transaction descriptor must be copied into the TX buffer shared between the Hypervisor and SPM.

       If the TX buffer is busy, the Hypervisor must return *BUSY*.

       If the TX buffer is too small and it is not possible to use the optional features to transmit the descriptor listed in 4.1.2 *Transmission of transaction descriptor in fragments* and 4.1.1 *Transmission of transaction descriptor in dynamically allocated buffers*, the Hypervisor must return *NO_MEMORY*

2.3 FFA_MEM_SHARE

Description

- Starts a transaction to grant access to a memory region to one or more Borrowers.
- Transaction details are described in a memory transaction descriptor (see Table 1.20).
- Descriptor is populated in the TX buffer of the Owner by default.
- Valid FF-A instances and conduits are listed in Table 2.12.
- Syntax of this function is described in Table 2.13.
- Encoding of result parameters in the FFA_SUCCESS function is described in Table 2.14.
- Encoding of error code in the FFA_ERROR function is described in Table 2.15.

Table 2.12: FFA_MEM_SHARE instances and conduits

Config No.	FF-A instance	Valid conduits
1	Secure and Non-secure physical	SMC, ERET
2	Secure and Non-secure virtual	SMC, HVC, SVC

Table 2.13: FFA_MEM_SHARE function syntax

Parameter	Register	Value
uint32 Function ID	w0	• 0x84000073. • 0xC4000073.
uint32 Total length	w1	• Total length of the memory transaction descriptor in bytes.
uint32 Fragment length	w2	• Length in bytes of the memory transaction descriptor passed in this ABI invocation. • *Fragment length* must be <= *Total length*. • If *Fragment length* < *Total length* then see 4.1.2 *Transmission of transaction descriptor in fragments* about how the remainder of the descriptor will be transmitted.
uint32/uint64 Address	w3/x3	• Base address of a buffer allocated by the Owner and distinct from the TX buffer. See 4.1.1 *Transmission of transaction descriptor in dynamically allocated buffers*. • MBZ if the TX buffer is used.

Parameter	Register	Value
uint32 Page count	w4	• Number of 4K pages in the buffer allocated by the Owner and distinct from the TX buffer. See 4.1.1 *Transmission of transaction descriptor in dynamically allocated buffers*. • MBZ if the TX buffer is used.
Other Parameter registers	w5-w7 x5-x17	• Reserved (SBZ).

Table 2.14: FFA_SUCCESS encoding

Parameter	Register	Value
uint64 Handle	w2/w3	• Globally unique Handle to identify the memory region on successful transmission of the transaction descriptor. MBZ otherwise (see 1.9.2 *Memory region handle*).
Other Result registers	w4-w7 x4-x17	• Reserved (MBZ).

Table 2.15: FFA_ERROR encoding

Parameter	Register	Value
int32 Error code	w2	• INVALID_PARAMETERS. • DENIED. • NO_MEMORY. • BUSY. • ABORTED.

2.3.1 Component responsibilities for FFA_MEM_SHARE

This interface is used to initiate a transaction to share a memory region with one or more Receiver endpoints (also see 1.7.2 *Share memory transaction lifecycle*). Only the Owner and Relayer participate in this stage of the transaction. Responsibilities of the:

- Owner are listed in 2.3.1.1 *Owner responsibilities*.
- Relayer are listed in 2.3.1.2 *Relayer responsibilities*.

The transaction descriptor could be populated in a buffer dynamically allocated by the Lender as specified in 4.1.1 *Transmission of transaction descriptor in dynamically allocated buffers*.

Transmission of the transaction descriptor in fragments must be implemented by the Lender and Relayer as specified in 4.1.2 *Transmission of transaction descriptor in fragments*.

Time slicing of this ABI invocation must be implemented by the Lender and Relayer as specified in 4.1.3 *Time slicing of memory management operations*.

2.3.1.1 Owner responsibilities

1. Must ensure it is a *PE endpoint* and Owner of the memory region.
2. Must ensure the memory region is in an access state suitable for sharing (see Table 1.11).
3. Must ensure the memory region fulfills the applicable rules stated in 1.3.1 *Ownership and access rules*.
4. Must describe memory region in the descriptor specified in Table 1.20 with an endpoint memory access descriptor for each Borrower (also see 1.11.3.1 *Sender usage*).
5. Must implement support for handling all error status codes that can be returned on completion of this interface.
6. If the invocation of this interface completes successfully, then must send at least the following information to each Borrower in a Partition message:
    1. Globally unique Handle returned by the Relayer.
    2. Owner endpoint ID.

    If the Borrower specified in the memory transaction descriptor is a SEPID, then the request must be sent to:

    - Either the *proxy endpoint* for the SEPID (see the Base FF-A Specification [1]) through a Partition message.
    - Or the *independent* peripheral device associated with the SEPID through an IMPLEMENTATION DEFINED mechanism.

    Provision of any other information from the transaction descriptor is IMPLEMENTATION DEFINED.

    If the Borrower rejects the request to share memory, the Sender should use the *FFA_MEM_RECLAIM* interface with the Handle returned by the Relayer to reclaim exclusive access to the memory region.

2.3.1.2 Relayer responsibilities

1. Must check that the RX/TX buffer pair is mapped, if the transaction descriptor is not passed in a dynamically allocated buffer as specified in 4.1.1 *Transmission of transaction descriptor in dynamically allocated buffers*, and return *INVALID_PARAMETERS* if the RX/TX buffer pair is not mapped.
2. Must validate the *Total length* input parameter to ensure that the length of the transaction descriptor does not exceed the size of the buffer it has been populated in. Must return *INVALID_PARAMETERS* in case of an error.
3. Must validate the *Sender endpoint ID* field in the transaction descriptor to ensure that the Lender is the Owner of the memory region and a *PE endpoint*. Must return *DENIED* in case of an error.
4. Must ensure that a request by an SP to share Secure memory to a NS-Endpoint is rejected by returning the *DENIED* error code.
5. Must validate that the memory region is in an access state suitable for sharing (see Table 1.11) and return *DENIED* in case of an error.
6. Must validate that the *Endpoint memory access descriptor count* & *Endpoint memory access descriptor array* fields in the transaction descriptor as specified in 1.11.3.3 *Relayer usage*.
7. Must validate the *Memory region attributes* field in the transaction descriptor as specified in 1.10.4 *Memory region attributes usage*.
8. Must validate the *Flags* field specified in the transaction descriptor as specified in 1.11.4 *Flags usage*.
9. Must validate the *Handle* field specified in the transaction descriptor as specified in 1.11.1 *Handle usage*.
10. If the Lender has specified a different data access permission to access the memory region in its translation regime, then the Relayer must validate the permission as specified in 1.10.2 *Data access permissions usage*, save the current permission and update the translation tables to reflect the new permission.

11. If the Borrower is a *PE endpoint* or a *Stream endpoint* with a *proxy endpoint* managed by the Relayer, then the Relayer must:
    1. Save the transaction descriptor information so that it can be validated when retrieved through invocations of the *FFA_MEM_RETRIEVE_REQ* & *FFA_MEM_RETRIEVE_RESP* interfaces.
    2. Return *NO_MEMORY* if there is not enough memory to complete this operation.
12. If the Borrower is a *Stream endpoint* associated with an *independent* device managed by the Relayer, then the Relayer must:
    1. Allocate an IPA range and map the memory region in the translation regime of the Borrower managed by the Relayer as specified in 1.2 *Address translation regimes*.

       The mapping must be done with the memory region attributes and permissions specified in the transaction descriptor.
    2. Describe the memory region to the device using the SEPID through an IMPLEMENTATION DEFINED mechanism.
13. If the call executes successfully, the Relayer must:
    1. Ensure that the state of the memory region in the participating FF-A components is observed as follows:
       1. If a Borrower is a *PE endpoint* or a *SEPID* associated with a dependent peripheral device, then:
          - *Owner-SA* for the Lender.
          - *!Owner-NA* for the Borrower.
       2. If a Borrower is a *SEPID* associated with an independent peripheral device, then:
          - *Owner-SA* for the Lender.
          - *!Owner-SA* for the Borrower.
    2. Allocate and return a *Handle* as described in 1.9.2 *Memory region handle*.
14. If the Lender is a VM and the Borrower is an SP or SEPID associated with a Secure Stream ID, the Hypervisor must forward the memory transaction descriptor to the SPM. This must be done by invoking this interface at the Non-secure physical FF-A instance as follows.
    1. The fields of the transaction descriptor must be unchanged apart from the following exception.
       1. The memory region must be described as composed of physically addressed constituent 4K pages in one or more *Constituent memory region descriptors*.

          This must be done by performing the VA or IPA to PA translation of the memory region described by the Owner at the non-secure virtual FF-A instance.

          The order in which the address ranges are specified by the Lender must be preserved by the Hypervisor.
       2. The *Constituent memory region descriptors* must be described in a *Composite memory region descriptor* which must be referenced by the *Endpoint memory access descriptor* included in the transaction descriptor.
    2. The updated transaction descriptor must be copied into the TX buffer shared between the Hypervisor and SPM.

       If the TX buffer is busy, the Hypervisor must return *BUSY*.

       If the TX buffer is too small and it is not possible to use the optional features to transmit the descriptor listed in 4.1.2 *Transmission of transaction descriptor in fragments* and 4.1.1 *Transmission of transaction descriptor in dynamically allocated buffers*, the Hypervisor must return *NO_MEMORY*.

2.4 FFA_MEM_RETRIEVE_REQ

Description

- Requests completion of a donate, lend or share memory management transaction.
- Transaction details are described in a memory transaction descriptor (see Table 1.20).
- Descriptor is populated in the TX buffer of the Receiver by default.
- Valid FF-A instances and conduits are listed in Table 2.17.
- Syntax of this function is described in Table 2.18.
- Encoding of error code in the FFA_ERROR function is described in Table 2.19.
- Successful transmission of the transaction descriptor is indicated by an invocation of the *FFA_MEM_RETRIEVE_RESP* function (see 2.5 *FFA_MEM_RETRIEVE_RESP*).

Table 2.17: FFA_MEM_RETRIEVE_REQ instances and conduits

Config No.	FF-A instance	Valid conduits
1	Secure and Non-secure physical	SMC, ERET
2	Secure and Non-secure virtual	SMC, HVC, SVC

Table 2.18: FFA_MEM_RETRIEVE_REQ function syntax

Parameter	Register	Value
uint32 Function ID	w0	• 0x84000074. • 0xC4000074.
uint32 Total length	w1	• Total length of the memory transaction descriptor in bytes.
uint32 Fragment length	w2	• Length in bytes of the memory transaction descriptor passed in this ABI invocation. • *Fragment length* must be <= *Total length*. • If *Fragment length* < *Total length* then see 4.1.2 *Transmission of transaction descriptor in fragments* about how the remainder of the descriptor will be transmitted.
uint32/uint64 Address	w3/x3	• Base address of a buffer allocated by the Owner and distinct from the TX buffer. See 4.1.1 *Transmission of transaction descriptor in dynamically allocated buffers*. • MBZ if the TX buffer is used.

Parameter	Register	Value
uint32 Page count	w4	• Number of 4K pages in the buffer allocated by the Owner and distinct from the TX buffer. See 4.1.1 *Transmission of transaction descriptor in dynamically allocated buffers*. • MBZ if the TX buffer is used.
Other Parameter registers	w5-w7 x5-x17	• Reserved (SBZ).

Table 2.19: FFA_ERROR encoding

Parameter	Register	Value
int32 Error code	w2	• INVALID_PARAMETERS. • DENIED. • NO_MEMORY. • BUSY. • ABORTED.

2.4.1 Component responsibilities for FFA_MEM_RETRIEVE_REQ

This ABI is used by a Receiver to retrieve a memory region. Retrieval implies a request to the Relayer to map the memory region in the translation regime of the Receiver. The Receiver must use the transaction descriptor (see Table 1.20) to identify the memory region and specify its properties. The Receiver could:

1. Retrieve a memory region that was shared, lent or donated by an Owner. For this scenario, responsibilities of the:
   - Receiver are listed in 2.4.1.1 *Receiver responsibilities*.
   - Relayer are listed in 2.4.1.2 *Relayer responsibilities*.
2. Retrieve a memory region that it had relinquished but has not been reclaimed by the Owner yet (see 2.4.2 *Support for multiple retrievals by a Borrower*).

It is also possible for a Hypervisor to use this interface to retrieve a memory region description on its behalf. This scenario is described in 2.4.3 *Support for retrieval by the Hypervisor*.

In all cases, a successful retrieval is indicated by an invocation of the *FFA_MEM_RETRIEVE_RESP* ABI by the Relayer.

The transaction descriptor could be populated in a buffer dynamically allocated by the Receiver as specified in 4.1.1 *Transmission of transaction descriptor in dynamically allocated buffers*.

Transmission of the transaction descriptor in fragments must be implemented by the caller and Relayer as specified in 4.1.2 *Transmission of transaction descriptor in fragments*.

Time slicing of this ABI invocation must be implemented by the Receiver and Relayer as specified in 4.1.3 *Time slicing of memory management operations*.

2.4.1.1 Receiver responsibilities

1. Must populate a transaction descriptor with the *Handle* (see 1.11.1 *Handle usage*) that identifies the memory region, the *Endpoint ID* that identifies the Owner and the *Tag* (see 1.11.2 *Tag usage*) associated with the

transaction.

Could populate other fields of the transaction descriptor as follows. This depends on how much information the Sender shares with the Receiver about the memory management transaction.

- See 1.11.3.2 *Receiver usage* for usage of the *Endpoint memory access descriptor array* field.
- See 1.11.4 *Flags usage* for usage of the *Flags* field.
- See 1.10.4 *Memory region attributes usage* for usage of the *Memory region attributes* field.

The Relayer must validate the information provided by the Sender. It must reject the transaction by sending a Partition message to the Sender if the validation fails.

The protocol between the Sender and Receiver to convey transaction information and to reject the transaction is IMPLEMENTATION DEFINED.

2. Must implement support for handling all error status codes that can be returned on completion of this interface.

2.4.1.2 Relayer responsibilities

1. Must check that the RX/TX buffer pair is mapped, if the transaction descriptor is not passed in a dynamically allocated buffer as specified in 4.1.1 *Transmission of transaction descriptor in dynamically allocated buffers*, and return *INVALID_PARAMETERS* if the RX/TX buffer pair is not mapped.
2. Must validate the *Total length* input parameter to ensure that the length of the transaction descriptor does not exceed the size of the buffer it has been populated in. Must return *INVALID_PARAMETERS* in case of an error.
3. Must validate the Sender endpoint ID field in the transaction descriptor to ensure that the Sender is the Owner of the memory region or the proxy endpoint acting on behalf of a Stream endpoint. Must return *DENIED* in case of an error.
4. Must validate the *Memory region attributes* field in the transaction descriptor as specified in 1.10.4 *Memory region attributes usage*.
5. Must validate the *Flags* field specified in the transaction descriptor as specified in 1.11.4 *Flags usage*.
6. Must validate the *Handle* field specified in the transaction descriptor as specified in 1.11.1 *Handle usage*.
7. Must validate the *Tag* field specified in the transaction descriptor as specified in 1.11.2 *Tag usage*.
8. Must validate that the *Endpoint memory access descriptor count* & *Endpoint memory access descriptor array* fields in the transaction descriptor as specified in 1.11.3.3 *Relayer usage*.
9. Must map the memory region in the translation regime of the Receiver managed by the Relayer as specified in 1.2 *Address translation regimes*.

   If the Receiver is a proxy endpoint for one or more Stream endpoints then the memory region must be mapped in the stage 2 translation tables corresponding to each SEPID. The memory region must not be mapped in the translation regime of the proxy endpoint.

   The order in which the address ranges are specified by the Lender must be preserved by the Hypervisor.

   The Relayer must return *NO_MEMORY* if there is not enough memory to map the memory region.
10. Must return *BUSY*, if *FFA_MEM_RETRIEVE_RESP* cannot be invoked because the Receiver RX buffer is busy.
11. Must return *NO_MEMORY* if *FFA_MEM_RETRIEVE_RESP* cannot be invoked because there is not enough memory to allocate a transaction descriptor to describe the memory region.
12. If the call executes successfully, the Relayer must ensure that the state of the memory region in the participating FF-A components is observed as follows:
    - If the transaction type is FFA_MEM_DONATE,

  - *!Owner-NA* for the Owner.
  - *Owner-EA* for the Receiver.

- If the transaction type is FFA_MEM_LEND, and the count of Borrowers in the transaction is = *1*,
  - *Owner-LA* for the Lender.
  - *!Owner-EA* for the Borrower.
- If the transaction type is FFA_MEM_LEND, and the count of Borrowers in the transaction is > *1*,
  - *Owner-LA* for the Lender.
  - *!Owner-SA* for the Borrower.
- If the transaction type is FFA_MEM_SHARE,
  - *Owner-SA* for the Lender.
  - *!Owner-SA* for the Borrower.

2.4.2 Support for multiple retrievals by a Borrower

After a Receiver relinquishes access to a memory region (see 2.6 *FFA_MEM_RELINQUISH*) that was lent or shared, a Relayer must allow the Receiver to retrieve the memory region again as long as it has not been reclaimed by its Owner. To support this mechanism, it must:

1. Allow the Owner to reclaim the memory region only if all Borrowers have relinquished it as many times as they have retrieved it.
2. Unmap the memory region from the translation regime of the Borrower only after it has been relinquished as many times as it was retrieved.
3. Ensure that the address ranges used to describe the memory region on each retrieval are the same if the memory region is already mapped in the translation regime of the Receiver.

The number of times a Receiver is allowed to retrieve a memory region without relinquishing it first is *1* by default. A Receiver must use the FFA_FEATURES ABI (see Table 3.1 and the Base FF-A Specification [1]) to determine the number of outstanding retrievals supported by the Relayer. The Relayer must return *DENIED* if a Receiver exceeds the retrieval count.

2.4.3 Support for retrieval by the Hypervisor

In a transaction to donate, share or lend a memory region between an Owner VM and a Receiver SP, the SPM is responsible for allocating the *Handle* to identify the memory region (see 1.9.2 *Memory region handle*).

The Hypervisor implementation could maintain an association between the Handle and the memory region. For example, to map the memory region back into the translation regime of the Owner in response to an FFA_MEM_RECLAIM ABI.

A Hypervisor implementation could choose to rely on the SPM to manage the association between the Handle and the memory region. For example, to avoid memory costs associated with tracking this state over a period of time.

In this case, the Hypervisor could use the FFA_MEM_RETRIEVE_REQ ABI to obtain the memory region description by specifying its Handle. It would use this description to map or unmap the memory region depending on the operation requested by a VM. For example, an operation to reclaim a memory region would follow these steps.

1. Lender VM calls FFA_MEM_RECLAIM.
2. Hypervisor uses the Handle to call FFA_MEM_RETRIEVE_REQ and obtain the memory region description.
3. Hypervisor forwards FFA_MEM_RECLAIM to the SPM to ensure all Borrowers have stopped using the memory region.
4. On a successful return from the SPM, the Hypervisor uses the memory region description to map the region in the translation regime of the Lender VM.

5. Hypervisor completes the invocation of FFA_MEM_RECLAIM from the Lender VM successfully.

If it chooses to use this mechanism, the Hypervisor must populate the transaction descriptor as follows.

1. It must specify the Handle specified by the VM in the *Handle* field.
2. It must ensure that all other fields in the transaction descriptor are zeroed.

From the perspective of an SPM, an invocation of the FFA_MEM_RETRIEVE_REQ ABI at the Non-secure physical FF-A instance could,

- Either originate from the Hypervisor as described above.
- Or originate from a Borrower VM. It was forwarded by the Hypervisor.

In the former case, the SPM must not update the ownership and access state associated with the memory region as it would do in the latter case (see 2.4.1.2 *Relayer responsibilities*). To do this, the SPM must distinguish between the two types of invocation as follows.

- In the former case, the *Endpoint memory access descriptor count* in the transaction descriptor must be *0*.
- In the latter case, the *Endpoint memory access descriptor count* in the transaction descriptor must be >= *1*.

In the former case, the SPM must also validate the *Handle* field specified in the transaction descriptor as follows.

- Ensure that it identifies a memory region that was either shared or lent to at least a single VM or is owned by a VM.
- Ensure that it was previously allocated and has not been reclaimed by the Owner.

The SPM must provide the memory region description to the Hypervisor through an invocation of the FFA_MEM_RETRIEVE_RESP ABI as follows.

- The memory region must be described as composed of physically addressed constituent 4K pages in one or more *Constituent memory region descriptors*.
- The *Constituent memory region descriptors* must be described in a *Composite memory region descriptor*.
- The *Composite memory region descriptor* must be referenced by a single *Endpoint memory access descriptor* included in the transaction descriptor.
- The *Sender endpoint ID* field must be set to the Lender or Owner VM ID in the transaction descriptor.
- The *Handle* field must be set to the input *Handle*.

The SPM must return *INVALID_PARAMETERS* if it does not support this feature. Also see Table 3.1 and the *FFA_FEATURES* ABI in the Base FF-A Specification [1].

U **Implementation Note**

This feature allows the Hypervisor to retrieve the physical address ranges of a memory region that must be either mapped or unmapped from the stage 2 translation descriptors of a VM.

It is possible that the Hypervisor implementation maintains mappings in the stage 2 translation descriptors for a VM such that a *IPA != PA*. In this case, it must track the original IPA ranges through an IMPLEMENTATION DEFINED mechanism to be able to correctly map or unmap the retrieved memory region.

Furthermore, in the case where the Hypervisor must map the memory region in the stage 2 translation descriptors for a VM, it must track the original memory access permissions and attributes of the memory region through an IMPLEMENTATION DEFINED mechanism.

2.5 FFA_MEM_RETRIEVE_RESP

Description

- A Relayer uses this interface to describe a memory region and its properties in response to the latest successful invocation of the *FFA_MEM_RETRIEVE_REQ* interface by an endpoint or Hypervisor.
- Transaction details are described in a transaction descriptor specified in Table 1.20.
- Descriptor is populated in the RX buffer of the Receiver by default.
- Valid FF-A instances and conduits are listed in Table 2.21.
- Syntax of this function is described in Table 2.22.
- Encoding of error code in the FFA_ERROR function is described in Table 2.23.
- Successful transmission of the transaction descriptor is indicated by an invocation of any FF-A function by the Receiver.

Table 2.21: FFA_MEM_RETRIEVE_RESP instances and conduits

Config No.	FF-A instance	Valid conduits
1	Secure and Non-secure physical	SMC, ERET
2	Secure and Non-secure virtual	ERET

Table 2.22: FFA_MEM_RETRIEVE_RESP function syntax

Parameter	Register	Value
uint32 Function ID	w0	• 0x84000075.
uint32 Total length	w1	• Total length of the memory transaction descriptor in bytes.
uint32 Fragment length	w2	• Length in bytes of the memory transaction descriptor passed in this ABI invocation. • *Fragment length* must be <= *Total length*. • If *Fragment length* < *Total length* then see 4.1.2 *Transmission of transaction descriptor in fragments* about how the remainder of the descriptor will be transmitted.
uint32/uint64 Parameter	w3/x3	• Reserved (MBZ).
uint32/uint64 Parameter	w4/x4	• Reserved (MBZ).
Other Parameter registers	w5-w7 x5-x17	• Reserved (MBZ).

Table 2.23: FFA_ERROR encoding

Parameter	Register	Value
int32 Error code	w2	• INVALID_PARAMETERS. • NO_MEMORY.

2.5.1 Component responsibilities for FFA_MEM_RETRIEVE_RESP

A Relayer invokes this interface as the caller with an endpoint as the callee in the following scenarios. It must fulfill the responsibilities listed in 2.5.1.1 *Relayer responsibilities*.

- The endpoint calls *FFA_MEM_RETRIEVE_REQ* to retrieve a memory region that was donated, lent or shared with it by the Owner. The Relayer completes the transaction by invoking the *FFA_MEM_RETRIEVE_RESP* interface.
- The endpoint calls *FFA_MEM_RETRIEVE_REQ* to retrieve a memory region it had relinquished earlier. The Relayer fulfills the request by invoking the *FFA_MEM_RETRIEVE_RESP* interface (see 2.4.2 *Support for multiple retrievals by a Borrower*).

The SPM invokes this interface as the caller with the Hypervisor as the callee in the scenario described in 2.4.3 *Support for retrieval by the Hypervisor*.

In all scenarios, the Relayer must populate a transaction descriptor specified in Table 1.20 to describe the memory region and its properties. This must be done in one of the following buffers.

- The RX buffer of the callee must be used if the callee used its TX buffer in the counterpart invocation of the *FFA_MEM_RETRIEVE_REQ* ABI earlier.
- If the callee used a dynamically allocated buffer in the counterpart invocation of the *FFA_MEM_RETRIEVE_REQ* ABI earlier, then the same buffer must be used (see 4.1.1 *Transmission of transaction descriptor in dynamically allocated buffers*).

Transmission of the transaction descriptor in fragments in all scenarios must be implemented by the Relayer and callee as specified in 4.1.2 *Transmission of transaction descriptor in fragments*.

In all scenarios, the callee (endpoint or Hypervisor) must fulfill the responsibilities listed in 2.5.1.2 *Callee responsibilities*. It must use the error codes listed in Table 2.23 to report an error back to the Relayer.

2.5.1.1 Relayer responsibilities

1. Must populate the Sender endpoint ID field in the transaction descriptor with the endpoint ID of the Owner.
2. Must populate the *Memory region attributes* field in the transaction descriptor as specified in 1.10.4 *Memory region attributes usage*.
3. Must populate the *Flags* field specified in the transaction descriptor as specified in 1.11.4 *Flags usage*.
4. Must populate the *Handle* field specified in the transaction descriptor as specified in 1.11.1 *Handle usage*.
5. Must populate the *Tag* field specified in the transaction descriptor as specified in 1.11.2 *Tag usage*.
6. Must populate the *Endpoint memory access descriptor count* & *Endpoint memory access descriptor array* fields in the transaction descriptor as specified in 1.11.3.3 *Relayer usage*.

2.5.1.2 Callee responsibilities

1. Must return *INVALID_PARAMETERS* if any field in the transaction descriptor has been incorrectly encoded.
2. Must return *NO_MEMORY* if there is not enough memory to use the memory region description provided by the Relayer.

3. Must transfer ownership of the RX buffer back to the producer if it was used to transmit the transaction descriptor by the Relayer. (See the Base FF-A Specification [1]).

2.6 FFA_MEM_RELINQUISH

Description

- Starts a transaction to transfer access to a shared or lent memory region from a Borrower back to its Owner.
- Valid FF-A instances and conduits are listed in Table 2.26.
- Syntax of this function is described in Table 2.27.
- Successful completion of this function is indicated through the invocation of the FFA_SUCCESS function by the callee without any further parameters.
- Encoding of error code in the FFA_ERROR function is described in Table 2.28.

Table 2.25: Descriptor to relinquish a memory region

Field	Byte length	Byte offset	Description
Handle	8	0	• Globally unique Handle to identify a memory region.
Flags	4	8	• Bit[0]: Zero memory after relinquish flag. – This flag specifies if the Relayer must clear memory region contents after unmapping it from the translation regime of the Borrower. ∗ b'0: Relayer must not zero the memory region contents. ∗ b'1: Relayer must zero the memory region contents. – If the memory region was lent to multiple Borrowers, the Relayer must clear memory region contents after unmapping it from the translation regime of each Borrower, if any Borrower including the caller sets this flag. – MBZ if the memory region was shared, else the Relayer must return *INVALID_PARAMETERS*. – MBZ if the Borrower has Read-only access to the memory region, else the Relayer must return *DENIED*. – Relayer must fulfill memory zeroing requirements listed in 1.11.4 *Flags usage*.

Field	Byte length	Byte offset	Description
			• Bit[1]: Operation time slicing flag. – This flag specifies if the Relayer can time slice this operation. ∗ b'0: Relayer must not time slice this operation . ∗ b'1: Relayer must time slice this operation. • MBZ if the Relayer does not support time slicing of memory management operations (see 4.1.3 *Time slicing of memory management operations*).
			• Bit[31:2]: Reserved (SBZ).
Endpoint count	4	12	• Count of endpoint ID entries in the Endpoint array.
Endpoint array	–	16	• Array of endpoint IDs. Each entry contains a 16-bit ID.

Table 2.26: FFA_MEM_RELINQUISH instances and conduits

Config No.	FF-A instance	Valid conduits
1	Secure and Non-secure physical	SMC, ERET
2	Secure and Non-secure virtual	SMC, HVC, SVC

Table 2.27: FFA_MEM_RELINQUISH function syntax

Parameter	Register	Value
uint32 Function ID	w0	• 0x84000076.
Other Parameter registers	w1-w7 x1-x17	• Reserved (SBZ).

Table 2.28: FFA_ERROR encoding

Parameter	Register	Value
int32 Error code	w2	• INVALID_PARAMETERS. • DENIED. • NO_MEMORY. • ABORTED.

2.6.1 Component responsibilities for FFA_MEM_RELINQUISH

This interface is used by a Borrower endpoint to inform the Relayer that it is relinquishing access to a memory region that was lent or shared with it earlier. The memory region is identified by its *Handle*.

Transaction details are populated in the descriptor specified in Table 2.25 as follows.

- The Handle and list of Borrower endpoints is populated in the descriptor described in Table 2.25 in the TX buffer of the caller.

  If the caller is a *proxy endpoint*, then the identity and count of the *Stream* endpoints on whose behalf it is relinquishing the memory region must be specified in the *Endpoint count* and *Endpoint array* fields in the descriptor.

  If the caller is a *PE endpoint* Borrower, then *Endpoint count* must equal 1 and it must specify its ID in the *Endpoint array* field in the descriptor.

- The caller could use the *Flags* field to request the Relayer to zero the memory region after it has been unmapped from its translation regime or time slice the unmapping operation.

Responsibilities of the:

- Borrower are listed in 2.6.1.1 *Borrower responsibilities*.
- Relayer are listed in 2.6.1.2 *Relayer responsibilities*.

Time slicing of this ABI invocation must be implemented by the Borrower and Relayer as specified in 4.1.3 *Time slicing of memory management operations*.

2.6.1.1 Borrower responsibilities

1. Must ensure it has access to the memory region identified by the *Handle* parameter.
2. Must ensure it is either the Borrower of the memory region or the proxy endpoint acting on behalf of one or more Stream endpoints who are the Borrowers instead.
3. Must implement support for handling all error status codes that can be returned on completion of this interface.

2.6.1.2 Relayer responsibilities

1. Must ensure that the *Handle* provided by the Borrower is valid and associated with a memory region it can access. Must return *INVALID_PARAMETERS* in case of an error.
2. Must ensure that the *Flags* parameter is correctly encoded in the descriptor and the identities of Borrower endpoints are valid. Must return *INVALID_PARAMETERS* in case of an error.
3. Must ensure that the *Endpoint count* field has the value *1* for a *PE endpoint* and *> 0* for a *proxy endpoint*. Must return *INVALID_PARAMETERS* in case of an error.
4. Must ensure that the memory region is in the *!Owner-SA* or *!Owner-EA* state (see Table 1.5) for all Borrower endpoints specified by the caller. Must return *DENIED* in case of an error.

5. Must ensure that the memory region is unmapped from the translation regime of the Borrower (that is, it enters the *!Owner-NA* state (see Table 1.5)) only if it has been relinquished as many times as it has been retrieved by the Borrower.

   The memory region must be unmapped from the translation regime of the Borrower managed by the Relayer as specified in 1.2 *Address translation regimes*.

   If the caller is a proxy endpoint for a Stream endpoint then the memory region must be unmapped from the stage 2 translation tables corresponding to the SEPID.

   The Relayer must update internal state of the Borrower associated with the memory region to *!Owner-NA*.

   The Relayer must return *NO_MEMORY* if there is not enough memory to unmap the memory region.

6. Must clear the contents of the memory region after unmapping it if *bit[0]* is set in the *Flags* parameter.

2.7 FFA_MEM_RECLAIM

Description

- Restores exclusive access to a memory region back to its Owner.
- Valid FF-A instances and conduits are listed in Table 2.30.
- Syntax of this function is described in Table 2.31.
- Successful completion of this function is indicated through the invocation of the FFA_SUCCESS function by the callee.
- Encoding of error code in the FFA_ERROR function is described in Table 2.32.

Table 2.30: FFA_MEM_RECLAIM instances and conduits

Config No.	FF-A instance	Valid conduits
1	Secure and Non-secure physical	SMC, ERET
2	Secure and Non-secure virtual	SMC, HVC, SVC

Table 2.31: FFA_MEM_RECLAIM function syntax

Parameter	Register	Value
uint32 Function ID	w0	• 0x84000077.
uint64 Handle	w1/w2	• Globally unique Handle to identify the memory region (see 1.9.2 *Memory region handle*).
uint32 Flags	w3	• Bit[0]: Zero memory before reclaim flag. – This flag specifies if the Relayer must clear memory region contents before mapping it in the Owner translation regime. ∗ b'0: Relayer must not zero the memory region contents. ∗ b'1: Relayer must zero the memory region contents. – Relayer must fulfill memory zeroing requirements listed in in 1.11.4 *Flags usage*. – MBZ if the Owner has Read-only access to the memory region, else the Relayer must return *DENIED*.

Parameter	Register	Value
		• Bit[1]: Operation time slicing flag. – This flag specifies if the Relayer can time slice this operation. ∗ b'0: Relayer must not time slice this operation. ∗ b'1: Relayer must time slice this operation. • MBZ if the Relayer does not support time slicing of memory management operations (see 4.1.3 *Time slicing of memory management operations*) .
		• Bit[31:2]: Reserved (SBZ).
Other Parameter registers	w4-w7 x4-x17	• Reserved (SBZ).

Table 2.32: FFA_ERROR encoding

Parameter	Register	Value
int32 Error code	w2	• INVALID_PARAMETERS. • DENIED. • NO_MEMORY. • ABORTED.

2.7.1 Component responsibilities for FFA_MEM_RECLAIM

This interface is used in the following ways.

1. To complete a transaction to relinquish a memory region owned by the caller endpoint. Borrowers use the *FFA_MEM_RELINQUISH* interface to relinquish access to the memory region. The Owner uses this interface to reclaim exclusive access to the memory region.
2. To abort an in-progress transaction to donate, lend or share a memory region owned by the caller endpoint. If any Receiver endpoint is unable to accept the transaction and the memory region is not mapped into the translation regime of any other Receiver endpoint, the Owner can use this transaction to reclaim exclusive access to the memory region.

Responsibilities of the:

- Owner are listed in 2.7.1.1 *Owner responsibilities*.
- Relayer are listed in 2.7.1.2 *Relayer responsibilities*.

Time slicing of this ABI invocation must be implemented by the Owner and Relayer as specified in 4.1.3 *Time slicing of memory management operations*.

2.7.1.1 Owner responsibilities

1. Must ensure it is the Owner of the memory region identified by the *Handle* parameter.
2. Must ensure that access to the memory region has been relinquished by all Borrowers.

3. Must implement support for handling all error status codes that can be returned on completion of this interface.

2.7.1.2 Relayer responsibilities

1. Must ensure that the *Handle* provided by the Owner is valid and associated with a memory region it owns. Must return *INVALID_PARAMETERS* in case of an error.
2. Must ensure that the *Flags* parameter is correctly encoded. Must return *INVALID_PARAMETERS* in case of an error.
3. Must ensure that the memory region is in the *!Owner-NA* state (see Table 1.5) for all the Receiver endpoints managed by the Relayer and associated with the memory region.

   If one or more Borrowers are Stream endpoints associated with an *independent* peripheral device then in this case:

   1. Each Borrower must relinquish access to the memory region through an IMPLEMENTATION DEFINED mechanism.
   2. The Relayer must unmap the memory region from the stage 2 translation tables identified by the SEPID.

   Must return *DENIED* in case of an error.
4. Must clear the contents of the memory region if *bit[0]* is set in the *Flags* parameter.
5. If the state of the memory region for the Owner is *Owner-LA*, this implies that the region was lent.

   If the state of the memory region when it was lent was *Owner-EA*, the Relayer must map the memory region in the translation regime of the Owner as specified in 1.2 *Address translation regimes*.

   The mapping must be created at the same address range and with the same memory region properties as those when the *FFA_MEM_LEND* interface was invoked.

   Must return *NO_MEMORY* in case there is not enough memory to create the mapping in the Owner translation regime.

   If the state of the memory region when it was lent was *Owner-NA*, the Relayer must not map the memory region in the translation regime of the Owner.
6. If the state of the memory region for the Owner is *Owner-SA* this implies that the region was shared. The Relayer must map the memory region in the translation regime of the Owner as specified in 1.2 *Address translation regimes*.

   The mapping must be created at the same address range and with the same memory region properties as those when the *FFA_MEM_SHARE* interface was invoked.

   Must return *NO_MEMORY* in case there is not enough memory to change the mapping in the Owner translation regime.
7. If a VM is the Owner and the Borrower is an SP or SEPID associated with a Secure Stream ID, the Hypervisor must forward an invocation of this interface to the SPM.

   This must be done by invoking this interface at the Non-secure physical FF-A instance with the same parameter values specified by the Owner at the Non-secure virtual FF-A instance.
8. If the call executes successfully, the state of the memory region for the Owner must transition to *Owner-EA* or *Owner-NA*.

2.8 FFA_MEM_PERM_GET

Description

- This ABI is invoked at the Secure virtual FF-A instance with the SVC conduit to determine the permission attributes for a memory region accessible by the caller. Also see 2.8.0.0.1 *Overview*.
- Valid FF-A instances and conduits are listed in Table 2.34.
- Syntax of this function is described in Table 2.35.
- Encoding of result parameters in the FFA_SUCCESS function is described in Table 2.36.
- Encoding of error code in the FFA_ERROR function is described in Table 2.37.

Table 2.34: FFA_MEM_PERM_GET instances and conduits

Config No.	FF-A instance	Valid conduits
1	Secure virtual	SVC

Table 2.35: FFA_MEM_PERM_GET function syntax

Parameter	Register	Value
uint32 Function ID	w0	• 0x84000088. • 0xC4000088.
uint32/uint64 Base Address	w1/x1	• Base VA of a translation granule whose permission attributes must be returned.
Other parameter registers	w2-w7 x2-x17	• Reserved (SBZ).

Table 2.36: FFA_SUCCESS encoding

Parameter	Register	Value
uint32 Memory permissions	w2	• Bit[1:0]: Data access permission. – b'00: No access. – b'01: Read-write access. – b'10: Reserved. – b'11: Read-only access. • Bit[2]: Instruction access permission. – b'0: Executable. – b'1: Non-executable. • Bit[31:3]: Reserved (SBZ).

Parameter	Register	Value
Other Result registers	w3-w7 x3-x17	• Reserved (SBZ).

Table 2.37: FFA_ERROR encoding

Parameter	Register	Value
int32 Error code	w2	• INVALID_PARAMETERS: – Caller is not allowed to access the memory region the address lies in. – Base address is not correctly aligned. • DENIED: This function was invoked when the caller is not in the runtime model for SP initialization. • NOT_SUPPORTED: This function is not implemented at this FF-A instance.

2.8.0.0.1 Overview

FFA_MEM_PERM_GET is used to request the permission attributes of a memory region mapped in the following translation regimes of a S-EL0 SP.

- Stage 1 of the Secure EL1&0 translation regime.
- Single stage in the Secure EL2&0 translation regime.

The size of the memory region for which permission attributes are returned is equal to the translation granule size used in the applicable translation regime.

The VA specified in the *Base address* parameter is aligned to the size of the translation granule used in the translation regime. The permission attributes for this translation granule are returned by the callee. The caller determines the translation granule size through an IMPLEMENTATION DEFINED mechanism.

The returned permission attributes are those that will be observed by the caller and not the exact attributes programmed in the translation tables. For example, the instruction access permission could programmed as *executable* and data access permissions as *writable* in the stage 1 translation table of the Secure EL1&0 translation regime but SCTLR_EL1.WXN could be set to 1. In this case, the returned instruction access permission is non-executable.

This ABI can be invoked by an SP only during the initialization of its execution context on the primary PE (see the Base FF-A Specification [1]). An invocation of this ABI post-initialization on the primary PE or at any time on a secondary PE is completed by the callee by invoking the FFA_ERROR function with the *DENIED* error code.

2.9 FFA_MEM_PERM_SET

Description

- This ABI is invoked at the Secure virtual FF-A instance with the SVC conduit to set the permission attributes for a memory region accessible by the caller. Also see 2.9.0.0.1 *Overview*.
- Valid FF-A instances and conduits are listed in Table 2.39.
- Syntax of this function is described in Table 2.40.
- Returns FFA_SUCCESS without any further parameters on successful completion.
- Encoding of error code in the FFA_ERROR function is described in Table 2.41.

Table 2.39: FFA_MEM_PERM_SET instances and conduits

Config No.	FF-A instance	Valid conduits
1	Secure virtual	SVC

Table 2.40: FFA_MEM_PERM_SET function syntax

Parameter	Register	Value
uint32 Function ID	w0	• 0x84000089. • 0xC4000089.
uint32/uint64 Base Address	w1/x1	• Base VA of a memory region whose permission attributes must be set.
uint32 Page count	w2	• Number of translation granule size pages starting from the *Base address* whose permissions must be set.
uint32 Memory permissions	w3	• Bit[1:0]: Data access permission. – b'00: No access. – b'01: Read-write access. – b'10: Reserved. – b'11: Read-only access. • Bit[2]: Instruction access permission. – b'0: Executable. – b'1: Non-executable. • Bit[31:3]: Reserved (SBZ).
Other parameter registers	w4-w7 x4-x17	• Reserved (SBZ).

Table 2.41: FFA_ERROR encoding

Parameter	Register	Value
int32 Error code	w2	• INVALID_PARAMETERS: – Caller is not allowed to access the memory region the address lies in. – Memory permissions are incorrectly encoded. – Base address is not correctly aligned. – Page count is 0. • DENIED: This function was invoked when the caller is not in the runtime model for SP initialization. • NOT_SUPPORTED: This function is not implemented at this FF-A instance.

2.9.0.0.1 Overview

FFA_MEM_PERM_SET is used to set the permission attributes of a memory region mapped in the following translation regimes of a S-EL0 SP.

- Stage 1 of the Secure EL1&0 translation regime.
- Single stage in the Secure EL2&0 translation regime.

The VA of the memory region specified in the *Base address* parameter is aligned to the size of the translation granule used in the translation regime.

The size of the memory region for which permission attributes are set is expressed as a count of pages. The size of each page is equal to the translation granule size in the applicable translation regime.

The *Memory permissions* parameter must be encoded as per the following rules.

1. A combination of attributes that mark the region with RW and Executable permissions is prohibited.
2. A request to mark a device memory region with Executable permissions is prohibited.

In case of an error, the callee preserves the original permissions of the memory regions.

This ABI can be invoked by an SP only during the initialization of its execution context on the primary PE (see the Base FF-A Specification [1]). An invocation of this ABI post-initialization on the primary PE or at any time on a secondary PE is completed by the callee by invoking the FFA_ERROR function with the *DENIED* error code.

Chapter 3
Feature discovery

The *FFA_FEATURES* ABI (see [1]) is used to discover the presence of other FF-A ABIs and features implemented by an FF-A ABI. The presence of FF-A memory management ABIs described in this document can be determined through the *FFA_FEATURES* ABI as specified in the Base specification. Table 3.1 describes the discoverable features implemented by individual FF-A memory management ABIs.

Table 3.1: Encoding of interface properties parameters

FF-A Function ID (w1)	Input properties (w2)	Return parameters (w2-w3)
FFA_MEM_DONATE	Reserved (SBZ).	• *w2* : Bits[31:2] are reserved (MBZ) . – Bit[0]: Dynamically allocated buffer support. See 4.1.1 *Transmission of transaction descriptor in dynamically allocated buffers*. ∗ b'0: Partition manager does not support transmission of a memory transaction descriptor in a buffer dynamically allocated by the endpoint. ∗ b'1: Partition manager supports transmission of a memory transaction descriptor in a buffer dynamically allocated by the endpoint. • *w3/x3* : Reserved (MBZ).
FFA_MEM_LEND	Reserved (SBZ).	• Same as FFA_MEM_DONATE.
FFA_MEM_SHARE	Reserved (SBZ).	• Same as FFA_MEM_DONATE.

FF-A Function ID (w1)	Input properties (w2)	Return parameters (w2-w3)
FFA_MEM_RETRIEVE_REQ	• Bits[31:2] are Reserved (SBZ) • Bit[0] is Reserved (SBZ) • Bit[1]: Request callee to specify security state of a memory region if retrieved successfully. See 1.10.4.1.1 *Discovery of NS bit usage*. – b'0: Do not specify security state of the memory region. – b'1: Specify the security state of the memory region.	• *w2* : Bits[31:3] are Reserved (MBZ). – Bit[0]: Dynamically allocated buffer support. See 4.1.1 *Transmission of transaction descriptor in dynamically allocated buffers*. ∗ b'0: Partition manager does not support transmission of a memory transaction descriptor in a buffer dynamically allocated by the endpoint. ∗ b'1: Partition manager supports transmission of a memory transaction descriptor in a buffer dynamically allocated by the endpoint. – Bit[1]: Inform caller if the security state of a memory region is specified during a successful retrieval. See 1.10.4.1.1 *Discovery of NS bit usage*. ∗ b'0: Security state of the memory region is not specified. ∗ b'1: Security state of the memory region is specified. – Bit[2]: Support for retrieval by the Hypervisor. See 2.4.3 *Support for retrieval by the Hypervisor*. ∗ b'0: SPMC does not support this feature at the Non-secure physical FF-A instance. · This value is also returned by a partition manager if this feature is queried at any other FF-A instance. ∗ b'1: SPMC supports this feature at the Non-secure physical FF-A instance. • *w3* : Outstanding retrievals field. – Bit[31:8]: Reserved MBZ. – Bit[7:0]: Number of times a Receiver is allowed to retrieve a memory region before relinquishing it. The value specified is interpreted as *((1U << (value + 1)) - 1*.

Chapter 4
Appendix

4.1 Additional memory management features

4.1.1 Transmission of transaction descriptor in dynamically allocated buffers

4.1.1.1 Rationale

The transaction descriptor (see Table 1.20) is transmitted from the caller to the callee in an invocation of the following ABIs.

- *FFA_MEM_DONATE*. See 2.1 *FFA_MEM_DONATE*.
- *FFA_MEM_LEND*. See 2.2 *FFA_MEM_LEND*.
- *FFA_MEM_SHARE*. See 2.3 *FFA_MEM_SHARE*.
- *FFA_MEM_RETRIEVE_REQ*. See 2.4 *FFA_MEM_RETRIEVE_REQ*.
- *FFA_MEM_RETRIEVE_RESP*. See 2.5 *FFA_MEM_RETRIEVE_RESP*.

This version of the Framework assumes that by default, the transaction descriptor is populated in the RX/TX buffers of an endpoint, Hypervisor or SPM as follows.

- The TX buffer is used to transmit the descriptor from an endpoint to the Hypervisor or SPM.
- The TX buffer is used to transmit the descriptor from the Hypervisor to the SPM.
- The RX buffer is used to transmit the descriptor from the Hypervisor or SPM to an endpoint.
- The RX buffer is used to transmit the descriptor from the SPM to the Hypervisor.

It is possible that the size of the descriptor is larger than the RX or TX buffer. For example, an endpoint memory access descriptor entry (see Table 1.16) in the transaction descriptor could reference one or more composite memory region descriptors (see Table 1.13). The total size of the composite memory region descriptors could be larger than the RX or TX buffer.

Each FF-A component is allowed to share only a single RX/TX pair with another FF-A component (see the Base FF-A Specification [1] for more information). It is possible that an endpoint or a partition manager cannot tolerate the latency in acquiring access to these buffers for a memory management operation on a busy system. It is also possible that other users cannot tolerate the latency to acquire access to these buffers due to an ongoing memory management operation.

4.1.1.2 Overview

This version of the Framework supports an optional feature that allows an endpoint to:

1. Dynamically allocate a separate buffer, instead of using the TX buffer, to transmit the transaction descriptor in an invocation of the following ABIs.
   - *FFA_MEM_DONATE*.
   - *FFA_MEM_LEND*.
   - *FFA_MEM_SHARE*.
   - *FFA_MEM_RETRIEVE_REQ*.
2. Use this buffer instead of the RX buffer to transmit the transaction descriptor in an invocation of the *FFA_MEM_RETRIEVE_RESP* ABI.

4.1.1.3 Description

The ability of an endpoint to use this feature depends on whether its partition manager implements support to map the dynamically allocated buffer into its translation regime. An endpoint can discover the availability of this support through the *FFA_FEATURES* interface (see Table 3.1 and the Base FF-A Specification [1]) for more information).

An endpoint must follow these rules while allocating a buffer dynamically.

- The dynamically allocated buffer must use the same attributes as RX/TX buffers that are specified in the Base FF-A Specification [1].
- The dynamically allocated buffer must be contiguous in the address space where it is allocated.

- The dynamically allocated buffer must fulfill the size and alignment requirements listed in the Base FF-A Specification [1] to allow the partition manager to map it. The endpoint must discover these requirements by invoking the *FFA_FEATURES* interface with the function ID of the *FFA_RXTX_MAP* interface (see the Base FF-A Specification [1]).

The address and size of a dynamically allocated buffer must be specified in an invocation of the following ABIs.

- *FFA_MEM_DONATE*.
- *FFA_MEM_LEND*.
- *FFA_MEM_SHARE*.
- *FFA_MEM_RETRIEVE_REQ*.

The syntax for specifying the address and size is as follows.

- The *w3/x3* register must be used to specify the VA, IPA or PA of the dynamically allocated buffer.

  A value of 0 must be specified to indicate that the TX buffer is being used.

- The *w4* register must be used to specify the size of the dynamically allocated buffer as a count of the contiguous 4K pages that constitute it.

  A value of 0 must be specified if the TX buffer is being used.

In an invocation of the *FFA_MEM_RETRIEVE_RESP* ABI:

- The partition manager must use the same buffer that was used in the counterpart *FFA_MEM_RETRIEVE_REQ* ABI invocation.
- A value of 0 must be specified in the *w3/x3* register since there is no need to specify which buffer is being used.
- A value of 0 must be specified in the *w4* register since there is no need to specify the size of the buffer is being used.

If dynamically allocated buffers are supported, a partition manager must map the dynamically allocated buffer in its translation regime on invocation, and unmap it on completion of the following ABIs.

- *FFA_MEM_DONATE*.
- *FFA_MEM_LEND*.
- *FFA_MEM_SHARE*.

The buffer must be mapped by the partition manager on invocation of the *FFA_MEM_RETRIEVE_REQ* ABI. It must be unmapped after the complete transaction descriptor has been transmitted through the invocation of the counterpart *FFA_MEM_RETRIEVE_RESP* ABI.

A partition manager must return:

- *INVALID_PARAMETERS* in the following scenarios:
  - It does not support this feature and an endpoint attempts to use it as described above.
  - The address or size of the dynamically allocated buffer is invalid.
- *NO_MEMORY* if it does not have enough memory to map the dynamically allocated buffer in its translation regime.

4.1.2 Transmission of transaction descriptor in fragments

4.1.2.1 Rationale

The size of a memory transaction descriptor (see Table 1.20) could exceed the size of the buffer used by an endpoint to transmit it. This is possible in the following scenarios.

1. An endpoint or partition manager does not implement support for dynamically allocated buffers (see 4.1.1 *Transmission of transaction descriptor in dynamically allocated buffers*). The RX/TX buffers must be used instead and cannot always accommodate the memory transaction descriptor.
2. An endpoint or partition manager do implement support for dynamically allocated buffers. In some memory management operations, the size of the memory transaction descriptor exceeds the size of the dynamically allocated buffer.

4.1.2.2 Overview

This version of the Framework supports an optional feature that:

- Allows the Sender of the transaction descriptor, to break the descriptor into equal or variable sized *fragments*, such that each fragment fits into the RX, TX or a dynamically allocated buffer.
- Adds support to transmit the first fragment of a transaction descriptor instead of the entire descriptor to the following ABIs.
  - *FFA_MEM_DONATE*. See 2.1 *FFA_MEM_DONATE*.
  - *FFA_MEM_LEND*. See 2.2 *FFA_MEM_LEND*.
  - *FFA_MEM_SHARE*. See 2.3 *FFA_MEM_SHARE*.
  - *FFA_MEM_RETRIEVE_REQ*. See 2.4 *FFA_MEM_RETRIEVE_REQ*.
  - *FFA_MEM_RETRIEVE_RESP*. See 2.5 *FFA_MEM_RETRIEVE_RESP*.
- Defines the following ABIs to transmit the remaining fragments from the Sender to the Receiver.
  - *FFA_MEM_FRAG_RX*. See 4.1.2.4 *FFA_MEM_FRAG_RX*.
  - *FFA_MEM_FRAG_TX*. See 4.1.2.5 *FFA_MEM_FRAG_TX*.

  A Sender can invoke these interfaces as many times as there are fragments to transmit the complete descriptor to the Receiver.

4.1.2.3 Description

The ability of an endpoint to use this feature depends on whether its partition manager implements support for receipt and transmission of fragments of the memory transaction descriptor. An endpoint can discover the availability of this support by invoking the *FFA_FEATURES* interface (see Table 3.1 and the Base FF-A Specification [1]) with the FID of the FFA_MEM_FRAG_TX and FFA_MEM_FRAG_RX ABIs described below. An endpoint must support this feature if its partition manager supports it.

It is strongly recommended that endpoint and partition manager implementations include support for this feature.

An endpoint and partition manager must implement the following protocol to use this feature.

1. An endpoint is the *Sender* and the partition manager is the *Receiver* of fragments in an invocation of the following ABIs.
   - *FFA_MEM_DONATE*.
   - *FFA_MEM_LEND*.
   - *FFA_MEM_SHARE*.
   - *FFA_MEM_RETRIEVE_REQ*.
2. The partition manager is the *Sender* and endpoint is the *Receiver* of fragments in an invocation of the *FFA_MEM_RETRIEVE_RESP* ABI.
3. A *Sender* must use these ABIs to transmit the first fragment of the memory transaction descriptor as follows.
   - The *w2* register must be used to specify the length the first fragment.

- The buffer used to transmit the first fragment depends on the ABI being invoked as follows.
  - The *Sender* must either use its TX buffer or a dynamically allocated buffer (if supported by the *Receiver*) in an invocation of the following ABIs.
    - *FFA_MEM_DONATE*.
    - *FFA_MEM_LEND*.
    - *FFA_MEM_SHARE*.
    - *FFA_MEM_RETRIEVE_REQ*.
  - The buffer used by the *Sender* in an invocation of the *FFA_MEM_RETRIEVE_RESP* ABI must be one of the following.
    - The RX buffer of the *Receiver* if it used its TX buffer in the earlier counterpart invocation of the *FFA_MEM_RETRIEVE_REQ* ABI.
    - The dynamically allocated buffer that was used by the *Receiver* in the earlier counterpart invocation of the *FFA_MEM_RETRIEVE_REQ* ABI.
- A partition manager as the *Receiver* must return *INVALID_PARAMETERS* if it does not support this feature or the length of the fragment is invalid.

4. After receiving the first fragment, a *Receiver* must allocate a *Handle* (see 1.9.2 *Memory region handle*) and use it to associate the remaining fragments with the current instance of the ABI invocation.

   The same *Handle* must be used to identify the memory region description once all the fragments have been received.

5. A *Receiver* must request the *Sender* to transmit the next fragment through an invocation of the *FFA_MEM_FRAG_RX* ABI. See 4.1.2.4 *FFA_MEM_FRAG_RX* for a description of this ABI and its parameters.

   The *Receiver* must use this interface to request re-transmission of a fragment as well. This could happen if it was unable to receive the previous fragment due to an IMPLEMENTATION DEFINED reason.

   The *Receiver* must populate the *w4* parameter register at a physical FF-A instance as follows.

   1. With the endpoint ID of the *Owner* of the memory region, if the fragment is being transmitted in response to the following ABI invocations.
      - *FFA_MEM_DONATE*.
      - *FFA_MEM_LEND*.
      - *FFA_MEM_SHARE*.
      - *FFA_MEM_RETRIEVE_REQ* on behalf of the Hypervisor see 2.4.3 *Support for retrieval by the Hypervisor*.
      - *FFA_MEM_RETRIEVE_RESP* on behalf of the Hypervisor see 2.4.3 *Support for retrieval by the Hypervisor*.
   2. With the endpoint ID of the *Borrower* of the memory region, if the fragment is being transmitted in response to the following ABI invocations.
      - *FFA_MEM_RETRIEVE_REQ* by the Hypervisor on behalf of a Borrower VM.
      - *FFA_MEM_RETRIEVE_RESP* by the Hypervisor on behalf of a Borrower VM.

6. A *Sender* must transmit the next fragment to the *Receiver* through an invocation of the *FFA_MEM_FRAG_TX* ABI. See 4.1.2.5 *FFA_MEM_FRAG_TX* for a description of this ABI and its parameters.

   The buffer used to transmit the fragment must be the same as the one used to transmit the first fragment.

   The *Sender* must populate the *w4* parameter register at a physical FF-A instance with the endpoint ID that was populated in the same register in the counterpart invocation of *FFA_MEM_FRAG_RX* by the *Receiver*.

7. A *Receiver* must acknowledge receipt of the final fragment. It must do this by completing the invocation of the ABI that was invoked to transmit the first fragment. For example, *FFA_MEM_SHARE* must be completed with the *FFA_SUCCESS* function as described in 2.3 *FFA_MEM_SHARE*.

8. A *Receiver* could abort the memory management operation while transmission of fragments is in-progress due to IMPLEMENTATION DEFINED reasons. The operation is identified by the ABI used to transmit the first fragment. The invocation of this ABI must be completed to signal to the *Sender* that the operation has been aborted.

   The mechanism to do this depends on the type of *Receiver* and the FF-A instance it resides at as follows.

   - The *Receiver* is a partition manager at a virtual FF-A instance. It must invoke the *FFA_ERROR* function with the *ABORTED* error code.
   - The *Receiver* is a partition manager at a physical FF-A instance. It must invoke the *FFA_ERROR* function with the *ABORTED* error code.
   - The *Receiver* is an endpoint at a virtual FF-A instance. In this version of the Framework, this scenario is possible only during the invocation of the *FFA_MEM_RETRIEVE_RESP* ABI. The *Receiver* must invoke the *FFA_MEM_RELINQUISH* ABI (see 2.6 *FFA_MEM_RELINQUISH*) to abort the operation.

   In all cases, the *Receiver* must restore any globally observable state associated with the memory region described by the transaction descriptor to what it was prior to receipt of the first fragment.

   In all cases, the *Sender* must restore any globally observable state associated with the memory region described by the transaction descriptor to what it was prior to transmission of the first fragment.

9. A *Sender* at a virtual FF-A instance must not abort the memory management operation while transmission of fragments is in-progress.

   The Hypervisor could abort an operation as the *Sender* at the Non-secure physical FF-A instance. It must invoke the *FFA_ERROR* function with the *ABORTED* error code to do this.

Figure 4.1 illustrates an example where the *FFA_MEM_RETRIEVE_REQ* and *FFA_MEM_RETRIEVE_RESP* interfaces are used to retrieve a transaction descriptor in fragments at a virtual FF-A instance. The following assumptions have been made.

- The memory region is shared with only a single Borrower.
- The RX/TX buffers of the Borrower are used by these interfaces.
- In invocations of both *FFA_MEM_RETRIEVE_REQ* and *FFA_MEM_RETRIEVE_RESP*, the descriptor in Table 1.20 is split into two fragments to be delivered to the Relayer and Borrower respectively.
- In invocations of both *FFA_MEM_RETRIEVE_REQ* and *FFA_MEM_RETRIEVE_RESP*, only parameters relevant to fragment transmission have been illustrated.

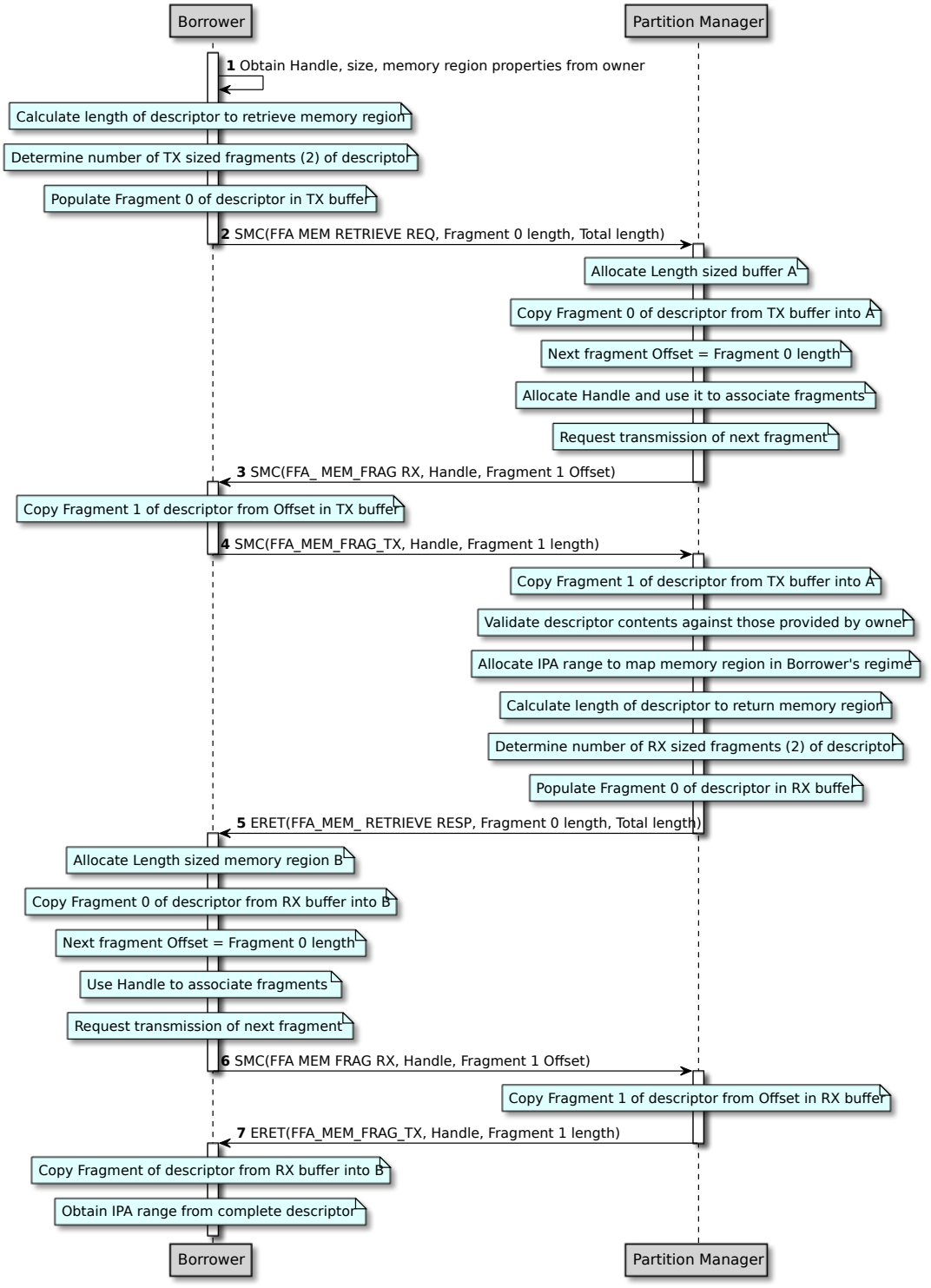

Figure 4.1: Example of fragment transmission while retrieving memory

4.1.2.4 FFA_MEM_FRAG_RX

Description

- A caller uses this interface to request the callee to transmit the next fragment of the memory transaction descriptor.
- Valid FF-A instances and conduits are listed in Table 4.2.
- Syntax of this function is described in Table 4.3.
- Successful completion of this function is indicated by an invocation of the *FFA_MEM_FRAG_TX* function (see 4.1.2.5 *FFA_MEM_FRAG_TX*).
- Encoding of error code in the FFA_ERROR function is described in Table 4.4.

Table 4.2: FFA_MEM_FRAG_RX instances and conduits

Config No.	FF-A instance	Valid conduits
1	Secure and Non-secure physical	SMC, ERET
2	Secure and Non-secure virtual	SMC, HVC, SVC, ERET

Table 4.3: FFA_MEM_FRAG_RX function syntax

Parameter	Register	Value
uint32 Function ID	w0	• 0x8400007A.
uint64 Handle	w1/w2	• *Handle* value to associate the fragment with the transaction descriptor of the ongoing memory management transaction with the callee.
uint32 Fragment offset	w3	• Byte offset from where the next fragment to be transmitted must start. • Offset must be calculated from the base of the transaction descriptor being transmitted. • Offset must be equal to one of the following: – The number of bytes of the transaction descriptor transmitted prior to the invocation of this interface. – The offset used in the previous invocation of this interface. This allows the Sender to re-transmit the previous fragment if the Receiver could not receive it due to an IMPLEMENTATION DEFINED reason.
uint32 Endpoint ID	w4	• ID of the Owner or Borrower endpoint. – Bit[31:16]: Endpoint ID. – Bit[15:0]: Reserved (SBZ). • Reserved (MBZ) at any virtual FF-A instance.

Parameter	Register	Value
Other parameter registers	w5-w7 w5-x17	• Reserved (SBZ).

Table 4.4: FFA_ERROR encoding

Parameter	Register	Value
int32 Error code	w2	• INVALID_PARAMETERS: Invalid Handle, fragment offset or endpoint ID value. • NOT_SUPPORTED: This function is not implemented at this FF-A instance. • ABORTED. Sender aborted transmission of fragments.

4.1.2.5 FFA_MEM_FRAG_TX

Description

- A caller uses this interface to transmit the next fragment of the transaction descriptor to the callee.
- Valid FF-A instances and conduits are listed in Table 4.6.
- Syntax of this function is described in Table 4.7.
- Successful completion of this function is indicated by an invocation of the *FFA_MEM_FRAG_RX* function (see 4.1.2.4 *FFA_MEM_FRAG_RX*).
- Encoding of error code in the FFA_ERROR function is described in Table 4.8.

Table 4.6: FFA_MEM_FRAG_TX instances and conduits

Config No.	FF-A instance	Valid conduits
1	Secure and Non-secure physical	SMC, ERET
2	Secure and Non-secure virtual	SMC, HVC, SVC, ERET

Table 4.7: FFA_MEM_FRAG_TX function syntax

Parameter	Register	Value
uint32 Function ID	w0	• 0x8400007B.
uint64 Handle	w1/w2	• *Handle* value to associate the fragment with the transaction descriptor of the ongoing memory management transaction with the callee.
uint32 Fragment length	w3	• Length of the fragment being transmitted.
uint32 Endpoint ID	w4	• ID of the Owner or Borrower endpoint. – Bit[31:16]: Endpoint ID. – Bit[15:0]: Reserved (SBZ). • Reserved (MBZ) at any virtual FF-A instance.
Other parameter registers	w5-w7 w5-x17	• Reserved (SBZ).

Table 4.8: FFA_ERROR encoding

Parameter	Register	Value
int32 Error code	w2	• INVALID_PARAMETERS: Invalid Handle, fragment length or endpoint ID value. • NOT_SUPPORTED: This function is not implemented at this FF-A instance. • ABORTED. Receiver aborted transmission of fragments.

4.1.3 Time slicing of memory management operations

4.1.3.1 Rationale

In each FF-A memory management ABI as follows, the partition manager is responsible for mapping or unmapping a memory region from the translation regime of an endpoint that invokes the ABI.

- *FFA_MEM_DONATE*. This interface is described in 2.1 *FFA_MEM_DONATE*.
- *FFA_MEM_LEND*. This interface is described in 2.2 *FFA_MEM_LEND*.
- *FFA_MEM_SHARE*. This interface is described in 2.3 *FFA_MEM_SHARE*.
- *FFA_MEM_RETRIEVE_REQ*. This interface is described in 2.4 *FFA_MEM_RETRIEVE_REQ*.
- *FFA_MEM_RELINQUISH*. This interface is described in 2.6 *FFA_MEM_RELINQUISH*.
- *FFA_MEM_RECLAIM*. This interface is described in 2.7 *FFA_MEM_RECLAIM*.

The duration of a mapping or unmapping operation on a set of translation tables depends on factors such as the size of the memory region, number of translation table entries it requires, number of cache and TLB maintenance operations etc. The operation runs to completion in the partition manager. This could prevent progress of the endpoint that requested the operation. In some scenarios, an endpoint might not be able to tolerate this delay for example, if it is prevented from processing its pending interrupts.

4.1.3.2 Overview

This version of the Framework supports an optional feature that allows the partition manager to divide the translation table operations into *time slices*.

An operation runs for the duration of a time slice. Once the time slice is over, the partition manager relinquishes control back to the endpoint. The endpoint resumes the operation later. On resumption the partition manager runs the operation for another time slice. The process repeats itself until the operation completes. The duration of a time slice and its discovery by a partition manager is IMPLEMENTATION DEFINED.

This optional feature enables both the endpoint and the partition manager to make progress during a long running memory management ABI invocation.

4.1.3.3 Description

The ability of an endpoint to use this feature depends on whether its partition manager implements support for time-slicing memory management ABI invocations. An endpoint can discover the availability of this support by invoking the *FFA_FEATURES* interface (see Table 3.1 and the Base FF-A Specification [1]) with the FID of the FFA_MEM_OP_PAUSE and FFA_MEM_OP_RESUME ABIs described below. This feature is only available to EL1 and S-EL1 endpoints.

An endpoint and its partition manager must implement the following protocol to use this feature.

1. An endpoint must request its partition manager to use time-slicing by setting the *Operation time slicing* flag in the *Flags* field as follows:
   - In the transaction descriptor (see 1.11.4 *Flags usage*) in an invocation of the following ABIs.
     - FFA_MEM_DONATE.
     - FFA_MEM_LEND.
     - FFA_MEM_SHARE.
     - FFA_MEM_RETRIEVE_REQ.
   - In Table 2.25 in an invocation of the FFA_MEM_RELINQUISH ABI.
   - In *w3* in an invocation of the FFA_MEM_RECLAIM ABI.
   - A partition manager must return *INVALID_PARAMETERS* if an endpoint sets the *Operation time slicing* flag and it does not support this feature.
2. A partition manager must divide the translation table operations required by the invoked ABI, if their duration is expected to exceed the time slice duration.

This must be done only after the partition manager has received the entire transaction descriptor in an invocation of the following ABIs.

- FFA_MEM_DONATE.
- FFA_MEM_LEND.
- FFA_MEM_SHARE.
- FFA_MEM_RETRIEVE_REQ.

Once the time slice duration expires, the partition manager must:

- Save enough state to resume the ABI invocation at a later point of time and not prevent progress of other FF-A functions before the paused ABI invocation is resumed.
- Cater for the scenario where the ABI invocation is paused on one PE and resumed by the endpoint on another PE.
- Use the *Handle* (see 1.9.2 *Memory region handle*) to identify the current instance of the ABI invocation when it is resumed later. A new *Handle* must be allocated if this was not done previously.
- Invoke the *FFA_MEM_OP_PAUSE* interface (see 4.1.3.4 *FFA_MEM_OP_PAUSE*) to inform the endpoint that the current ABI invocation has been paused and must be resumed later.

3. An endpoint must use the *FFA_MEM_OP_RESUME* interface (see 4.1.3.5 *FFA_MEM_OP_RESUME*) to resume the paused ABI invocation identified by the *Handle*.

   The endpoint could invoke other FF-A functions before resuming the paused ABI invocation.

4. A partition manager could abort the ABI invocation when it is resumed later due to IMPLEMENTATION DEFINED reasons. It must signal to the endpoint that the ABI invocation has been aborted by invoking the *FFA_ERROR* function with the *ABORTED* error code.

Figure 4.2 illustrates an example where the *FFA_MEM_RETRIEVE_REQ* and *FFA_MEM_RETRIEVE_RESP* interfaces are used by an endpoint to retrieve a transaction descriptor at a virtual FF-A instance. The following assumptions have been made.

- The operation to map the memory region in the translation regime of the endpoint is expected to take longer than the time slice value known to the partition manager.
- The endpoint has requested time slicing by setting the *Operation time slicing* flag.
- The partition manager divides the *FFA_MEM_RETRIEVE_REQ* function invocation into 3 time slices through the *FFA_MEM_OP_PAUSE* and *FFA_MEM_OP_RESUME* interfaces.

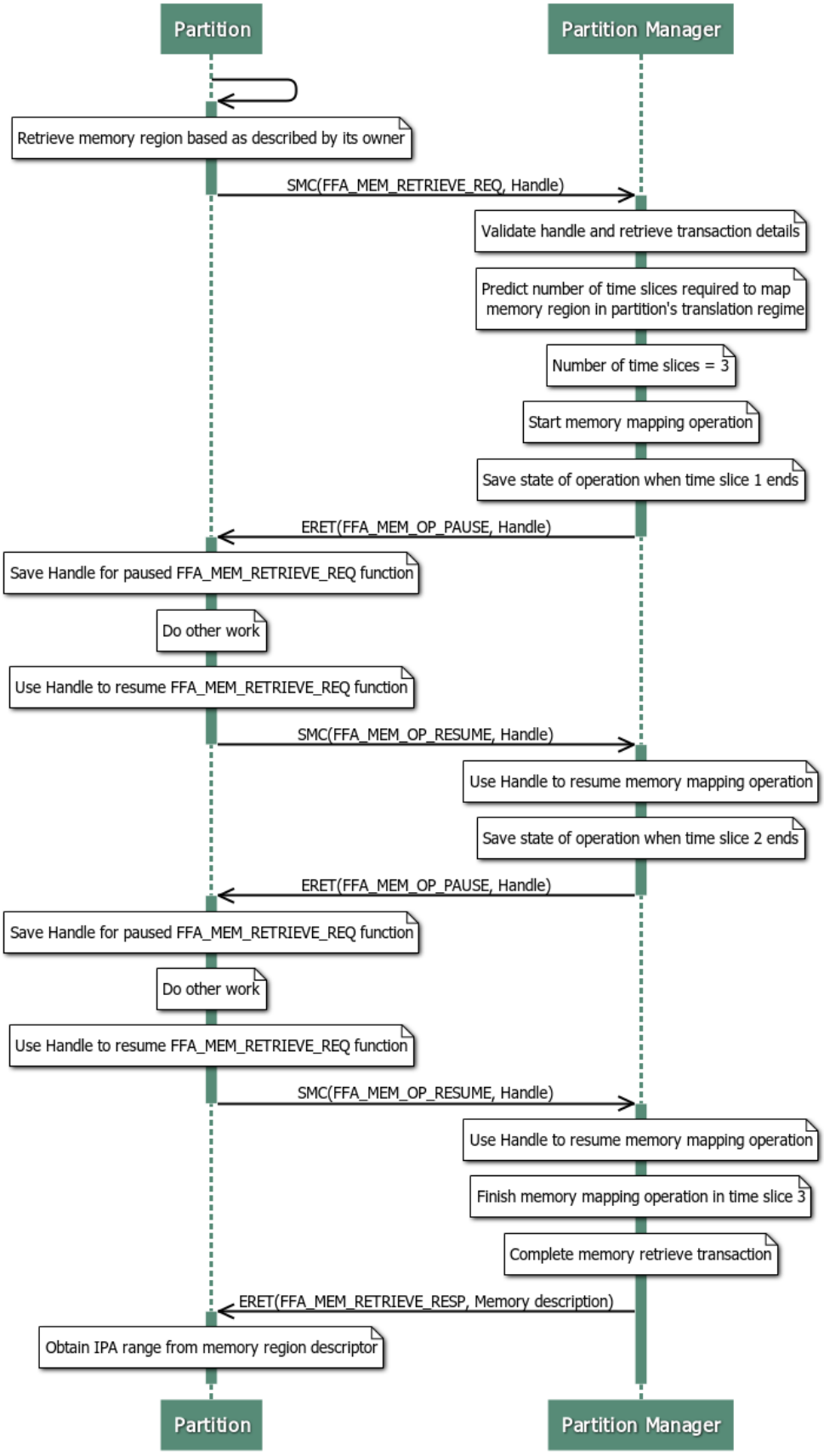

Figure 4.2: Example of time slicing during FFA_MEM_RETRIEVE_REQ

4.1.3.4 FFA_MEM_OP_PAUSE

Description

- A partition manager uses this interface to pause the execution of a memory management ABI invoked by an endpoint. Execution is returned to the endpoint.
- Valid FF-A instances and conduits are listed in Table 4.10.
- Syntax of this function is described in Table 4.11.
- Encoding of error code in the FFA_ERROR function is described in Table 4.12.

Table 4.10: FFA_MEM_OP_PAUSE instances and conduits

Config No.	FF-A instance	Valid conduits
1	Non-secure physical	ERET
2	Secure physical	SMC
3	Secure and Non-secure virtual	ERET

Table 4.11: FFA_MEM_OP_PAUSE function syntax

Parameter	Register	Value
uint32 Function ID	w0	• 0x84000078.
uint64 Handle	w1/w2	• *Handle* value to identify the paused memory management operation.
Other parameter registers	w3-w7 x3-x17	• Reserved (SBZ).

Table 4.12: FFA_ERROR encoding

Parameter	Register	Value
int32 Error code	w2	• INVALID_PARAMETERS: Invalid handle value. • NOT_SUPPORTED: This function is not implemented at this FF-A instance.

4.1.3.5 FFA_MEM_OP_RESUME

Description

- An endpoint uses this interface to request the partition manager to resume execution of a paused memory management ABI. The paused operation is identified by the supplied *Handle*.
- Valid FF-A instances and conduits are listed in Table 4.14.
- Syntax of this function is described in Table 4.15.
- Encoding of error code in the FFA_ERROR function is described in Table 4.16.

Table 4.14: FFA_MEM_OP_RESUME instances and conduits

Config No.	FF-A instance	Valid conduits
1	Non-secure physical	SMC
2	Secure physical	ERET
3	Secure and Non-secure virtual	SMC, HVC, SVC

Table 4.15: FFA_MEM_OP_RESUME function syntax

Parameter	Register	Value
uint32 Function ID	w0	• 0x84000079.
uint64 Handle	w1/w2	• *Handle* value to identify the paused memory management operation.
Other parameter registers	w3-w7 x3-x17	• Reserved (SBZ).

Table 4.16: FFA_ERROR encoding

Parameter	Register	Value
int32 Error code	w2	• INVALID_PARAMETERS: Invalid Handle value. • NOT_SUPPORTED: This function is not implemented at this FF-A instance.

4.2 Changes to FF-A v1.0 data structures for forward compatibility

Version 1.1 of the Framework specifies changes to make the following data structures defined in version 1.0 of the Framework forwards compatible.

1. Memory transaction descriptor in Table 1.20.
2. Endpoint memory access descriptor in Table 1.16.

These changes enable forward compatibility as described below.

1. A new field is always added at the end of these data structures.
2. A producer of this data structure specifies its size corresponding to the FF-A version it implements to a consumer.
3. A consumer of this data structure uses this size to correctly read the version of the data structure implemented by the producer.
4. A consumer of this data structure uses the size corresponding to the Framework version it implements to consume only fields defined in its version. Additional fields in the producer's version of this data structure are safely ignored enabling forward compatibility.

4.2.1 Changes to Memory transaction descriptor

The changes to this data structures (see Table 1.20 are listed below.

1. The *Endpoint memory access descriptor size* field has been added at the 24-byte offset in Table 1.20. This enables a consumer of this data structure to determine the size of this data structure used by the producer as described above.
2. The *Endpoint memory access descriptor array* at the 32-byte offset in the FF-A v1.0 descriptor (see Table 4.17) has been replaced by an offset to the array at the same offset in Table 1.20. This enables fields to be appended to this data structure independently from the location and size of the *Endpoint memory access descriptor array*.
3. A *Reserved* field of 12-bytes has been added at the 36-bytes offset to preserve the 16-byte alignment of the size of the FF-A v1.0 descriptor (see Table 4.17).
4. The *Memory region attributes* field size has been been expanded into the following Reserved field to increase its size from 1 byte to 2 bytes. These are reserved for future use.

Table 4.17: FF-A v1.0 memory transaction descriptor

Field	Byte length	Byte offset	Description
Sender endpoint ID	2	0	• ID of the Owner endpoint.
Memory region attributes	1	2	• Attributes must be encoded as specified in 1.10.4 *Memory region attributes usage*. • Attribute usage is subject to validation at the virtual and physical FF-A instances as specified in 1.10.4 *Memory region attributes usage*.
Reserved	1	3	• Reserved (MBZ).
Flags	4	4	• Flags must be encoded as specified in in 1.11.4 *Flags usage*.

Field	Byte length	Byte offset	Description
Handle	8	8	• Memory region handle in ABI invocations specified in 1.11.1 *Handle usage*.
Tag	8	16	• This field must be encoded as specified in 1.11.2 *Tag usage*.
Reserved	4	24	• Reserved (MBZ).
Endpoint memory access descriptor count	4	28	• Count of endpoint memory access descriptors.
Endpoint memory access descriptor array	–	32	• Each entry in the array must be encoded as specified in 1.11.3 *Endpoint memory access descriptor array usage*. See Table 1.16 for the encoding of the endpoint memory access descriptor.

Glossary

ABI

Application Binary Interface

DMA

Direct Memory Access

FF-A

Firmware Framework for A-profile

HVC

Hypervisor Call

MBP

Must be preserved

MBZ

Must be zero

MMIO

Memory Mapped Input Output

OS

Operating System

PE

Processing Element

PSA

Platform Security Architecture

SBZ

Should be zero

SMC

Secure Monitor Call

SMCCC

SMC Calling Convention

SMMU

System Memory Management Unit

SP

Secure Partition

SPM

Secure Partition Manager

SVC

Supervisor Call

VHE

Virtualization Host Extensions

VM

Virtual Machine