Platform Fault Detection Interface specification

Document number	110468
Document quality	ALP0
Document version	1.0
Document confidentiality	Non-confidential

Platform Fault Detection Interface specification

Release information

Date	Version	Changes
2025/Jun/20	1.0 ALP0	• Initial version

Non-Confidential Proprietary Notice

Arm Limited. Company 02557590 registered in England.

110 Fulbourn Road, Cambridge, England CB1 9NJ.

PRE-20349

8 March 2024

Contents

Platform Fault Detection Interface 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.

Pseudocode descriptions

This book uses a form of pseudocode to provide precise descriptions of the specified functionality. This pseudocode is written in a monospace font. The pseudocode language is described in the Arm Architecture Reference Manual.

Assembler syntax descriptions

This book contains numerous syntax descriptions for assembler instructions and for components of assembler instructions. These are shown in a `monospace` font.

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 SMC Calling Convention (SMCCC)*. See https://developer.arm.com/documentation/den0028/latest

Feedback

Arm welcomes feedback on its documentation.

Feedback on this book

If you have comments on the content of this book, send an e-mail to errata@arm.com. Give:

- The title (Platform Fault Detection Interface specification).
- The number (110468 1.0).
- The page numbers to which your comments apply.
- The rule identifiers to which your comments apply, if applicable.
- A concise explanation of your comments.

Arm also welcomes general suggestions for additions and improvements.

Note

Arm tests PDFs only in Adobe Acrobat and Acrobat Reader, and cannot guarantee the appearance or behavior of any document when viewed with any other PDF reader.

Chapter 1
Introduction

I$_{0001}$ This specification is in Alpha state. The interface definitions, including function identifiers, parameter encodings, and return codes, are subject to change. Implementations and software relying on this specification must not assume stability or compatibility across revisions.

1.1 Scope

D$_{0002}$ This document defines the **Platform Fault Detection Interface (PFDI)**, a standard interface that enables *System Software* to request fault detection checks from *Platform Firmware*.

I$_{0003}$ PFDI allows firmware-resident mechanisms to perform platform or PE-level validation of hardware and firmware state.

I$_{0004}$ The interface is designed to be implemented by Platform Firmware and invoked from System Software using `SMC` calls in accordance with Arm SMCCC [1].

S$_{0005}$ The PFDI interface supports the following use cases:

- Power-on self-tests of Processing Elements (PEs)
- Retrieval of firmware-internal boot-time check results
- Runtime test execution
- Controlled fault injection for software validation

1.2 Context

I0006

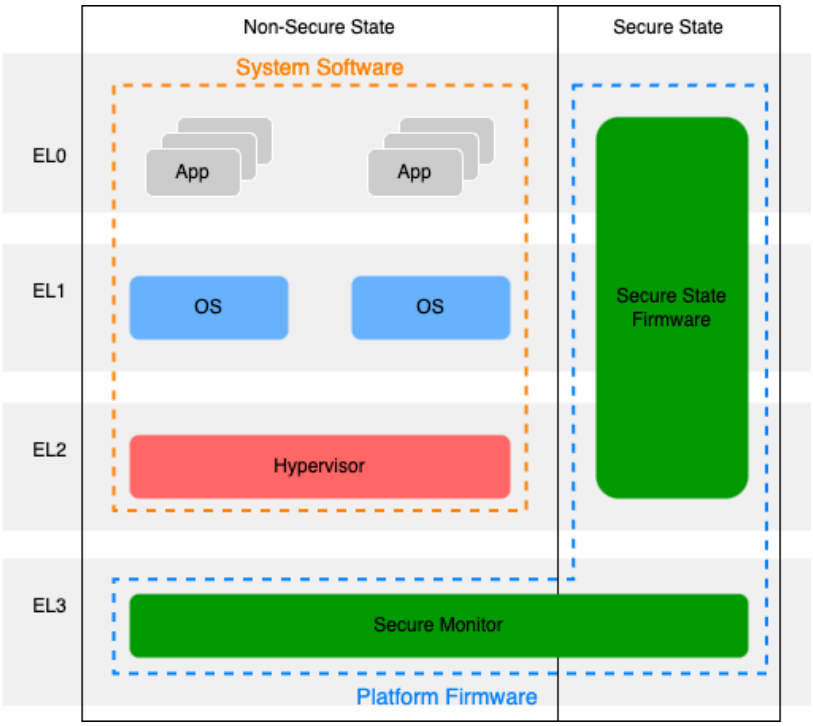

Figure 1.1: System Architecture

I0007 Platform Firmware is the Firmware that runs at Exception Level 3 and in the secure world.

I0008 System Software is the operating system or hypervisor that runs at EL1 or EL2 and makes use of services provided by Platform Firmware via standard interfaces such as PFDI.

I0009 Platform Firmware is responsible for configuring the system and establishing a consistent view of platform state and resources before control is passed to System Software.

I0010 In addition to its initialization role, Platform Firmware remains resident at runtime and exposes service interfaces to System Software through the SMC instruction, following the Arm SMCCC.

I0011 The Platform Fault Detection Interface (PFDI) is one such runtime service. It enables System Software to interact with fault-detection mechanisms implemented in Platform Firmware, including:

- Retrieval of boot-time validation results
- Execution of integrity checks on hardware and firmware state
- Invocation of self-test mechanisms such as Software Test Library (STL) routines
- Controlled injection of faults for validation purposes

I0012 These mechanisms are firmware-resident and PE-local unless otherwise specified. The exact tests, conditions considered as faults, and fault coverage capabilities are IMPLEMENTATION DEFINED.

1.3 Intended Usage

S$_{0013}$ PFDI functions are intended to be called:

- Once during early initialization to trigger and read firmware software test results, including those run at boot
- Periodically or event-triggered to execute PE-local self-tests
- On demand during validation to inject test faults

I$_{0014}$ Behavior is PE-local unless otherwise specified.

1.4 Document Structure

I$_{0015}$ This document is structured as follows:

- **Execution Models and Integration Scenarios**: Describes how PFDI is used in non-virtualized and virtualized environments
- **Interface Summary**: Summarizes the function categories and ABI conventions
- **Functional Specification**: Defines each PFDI function
- **Compliance Requirements**: Specifies conformance rules for implementations

Chapter 2
Execution Models and Integration Scenarios

I0016 The Platform Fault Detection Interface (PFDI) is designed to support a range of platform configurations. The following subsections describe example execution models that influence how System Software issues PFDI calls.

I0017 This section is *informative* and does not impose requirements on platform topology or virtualization model.

2.1 Direct Invocation from a Non-Virtualized OS

I0018 In a non-virtualized system, a single operating system running at Exception Level 1 (EL1) issues SMC calls directly to Platform Firmware to access PFDI functionality. This is the simplest usage model.

I0019

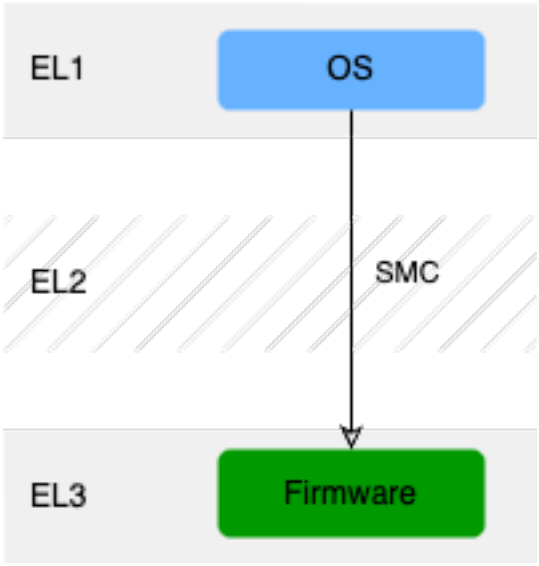

Figure 2.1: OS Invocation through SMC

2.2 Invocation from a Hypervisor

I0020 In virtualized systems, a hypervisor running at Exception Level 2 (EL2) may invoke PFDI functions on behalf of itself or its guest virtual machines.

I0021 In this model, the hypervisor issues SMCs directly to Platform Firmware. PFDI is treated as a firmware service used internally by the hypervisor.

I0022

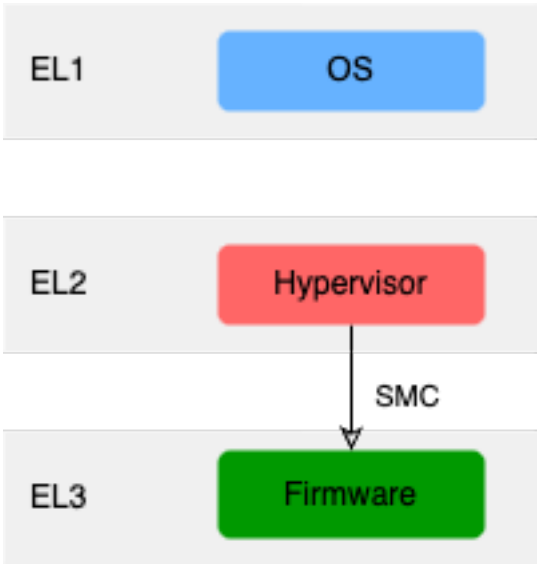

Figure 2.2: Hypervisor Invocation through SMC

2.3 Indirect Invocation from a Guest VM via HVC

I0023 In some virtualized systems, a guest virtual machine (VM) running at EL1 may issue PFDI calls indirectly via the hypervisor, using the HVC instruction.

I0024 A guest VM may execute a PFDI call using either the HVC or SMC instruction. If the VM is configured to access PFDI, the hypervisor traps the call and forwards it to Platform Firmware using the SMCCC interface.

I0025

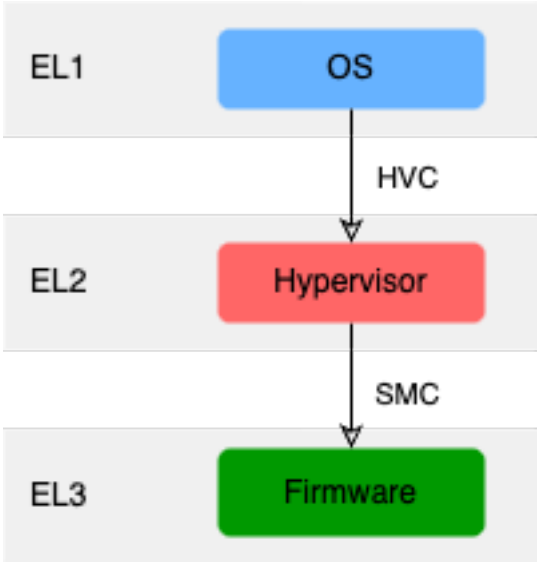

Figure 2.3: OS Invocation through HVC

R0026 The PFDI interface exposed to a guest VM via HVC shall be identical to the native SMC interface defined by this specification.

R0027 If forwarding is enabled, the hypervisor shall forward registers X0 to X4 unmodified as inputs to the SMC instruction issued on behalf of the calling VM.

I0028 Defining which VM is authorized or not to access PFDI functions is up to the hypervisor.

U0029 A hypervisor implementation should return SMCCC_RET_NOT_SUPPORTED to any guest VM that attempts to invoke a PFDI function but is not authorized to access the interface.

Chapter 3
Interface Summary

I$_{0030}$ This section provides an overview of the PFDI interface functions, calling conventions, return values, and PE-specific behavior.

3.1 Function Groups

I$_{0031}$ The functions defined in this specification are categorized as follows:

3.1.1 Capability Discovery

D$_{0032}$ These functions allow System Software to determine which PFDI features are implemented and supported on each PE.

- `PFDI_VERSION`
- `PFDI_FEATURES`
- `PFDI_PE_TEST_ID`
- `PFDI_PE_TEST_PART_COUNT`

3.1.2 Self-Test Execution

D$_{0033}$ These functions control the execution of test parts and platform integrity checks, and allow execution of and access to firmware and PE-level test results:

- `PFDI_PE_TEST_RUN`
- `PFDI_PE_TEST_RESULT`
- `PFDI_FW_CHECK`

3.1.3 Software Validation Support

D$_{0034}$ This function supports controlled error injection to allow software validation of error handling paths:

- `PFDI_FORCE_ERROR`

3.2 Calling Convention

D_{0035} All PFDI functions use the Arm SMCCC [1] with the SMC64 calling convention.

D_{0036} The function identifier is passed in register `W0`. All other parameters and results are carried in 64-bit registers.

D_{0037} The following registers are used:

- `X0`: Function identifier and return value
- `X1-X4`: Function parameters and return extensions (function-specific)
- `X5-X17`: Reserved by SMCCC; must be preserved across the call

R_{0038} System Software shall populate registers X0 to X4 with valid input values for the requested PFDI function, in accordance with the parameter definitions in Chapter 4 *Functional Specification*.

R_{0039} Platform Firmware shall return results using X0 to X4 with values that comply with the result encoding defined in Chapter 4 *Functional Specification* for the corresponding PFDI function.

R_{0040} Registers X5 to X17 shall be preserved by the Platform Firmware, in accordance with the SMCCC.

R_{0041} When a hypervisor relays a PFDI function call from a virtual machine, it shall issue the SMC64 instruction using the original values of X0 to X4 from the VM.

3.3 Return Codes

D0042 The Table 3.1 defines the values for error return codes used with the interface functions. The error return type is a 64-bit signed integer. Zero and positive values denote success and negative values indicate error.

D0043

Table 3.1: Return Codes

Name	Description	Value
SMCCC_RET_SUCCESS	The call completed successfully.	0
SMCCC_RET_NOT_SUPPORTED	PFDI is not supported on this platform.	-1
PFDI_RET_INVALID_PARAM	Some or all of the parameters passed to the call are invalid.	-3
PFDI_RET_FAULT_FOUND	A test has detected a fault.	-4
PFDI_RET_ERROR	The function call failed to complete for another reason.	-5
PFDI_RET_NOT_RUN	No test parts have executed.	-6
PFDI_RET_UNKNOWN	Value is not known.	-7
PFDI_RET_TEST_COUNT_ZERO	Zero tests are implemented.	-8

3.4 PE-local behavior

U0044 All PFDI function behavior is PE-local unless explicitly stated otherwise. Platform-specific implementations may support differing function sets or test mechanisms per PE.

S0045 System Software must probe each PE individually using `PFDI_FEATURES` and must not assume symmetric support across PEs.

Chapter 4
Functional Specification

I0046 This chapter defines each PFDI function, including its purpose, parameters, return values, and normative requirements.

4.1 Function Identifier Summary

D0047 Function identifiers are allocated in the SMCCC vendor-defined space and use the SMC64 calling convention.

D0048

Table 4.1: PFDI Function Identifiers

Function	ID	Description
PFDI_VERSION	0xC400_02D0	Report version
PFDI_FEATURES	0xC400_02D1	Discover supported functions
PFDI_PE_TEST_ID	0xC400_02D2	Get test engine metadata
PFDI_PE_TEST_PART_COUNT	0xC400_02D3	Report number of test parts
PFDI_PE_TEST_RUN	0xC400_02D4	Execute a test part range
PFDI_PE_TEST_RESULT	0xC400_02D5	Return boot-time test result
PFDI_FW_CHECK	0xC400_02D6	Report firmware check status
PFDI_FORCE_ERROR	0xC400_02D7	Inject an error for validation

4.2 PFDI_VERSION

4.2.1 Usage

D0049 Allows a client to retrieve the PFDI version of the firmware.

S0050 System software must check that it supports the version of the PFDI provided by the implementation. Currently a single version is defined. Future revisions of this specification will increment the major or minor version when changes to PFDI functions are introduced.

4.2.2 Format

D0051

Table 4.2: PFDI_VERSION Request

Type	Register	Content
uint32	w0	• Function ID: 0xC400_02D0
uint64	x1-x4	• Must be Zero (Reserved).

D0052

Table 4.3: PFDI_VERSION Response

Type	Register	Content
int64	x0	• On success: – Bits[63:31]: Must be Zero. – Bits[30:16]: PFDI Major Version. ∗ Must be 1 for this revision of PFDI. – Bits[15:0]: PFDI Minor Version. ∗ Must be 0 for this revision of PFDI. • On error: – SMCCC_RET_NOT_SUPPORTED: PFDI not supported.
uint64	x1-x4	• Must be Zero (Reserved).

4.2.3 Requirements

R0053 A PFDI implementation shall handle calls to the PFDI_VERSION function by returning a success value indicating the PFDI version that is supported.

R0054 Calls to other PFDI functions shall be supported in accordance with the version information returned by PFDI_VERSION.

R0055 All PEs in a PFDI instance shall implement the same version of the PFDI.

4.3 PFDI_FEATURES

4.3.1 Usage

D0056 Allows a client to query the PFDI implementation to determine the supported capabilities of the current PE.

S0057 System Software may use this function to discover which PFDI functions are available.

4.3.2 Format

D0058

Table 4.4: PFDI_FEATURES Request

Type	Register	Content
uint32	w0	• Function ID: 0xC400_02D1
uint32	w1	• ID of PFDI Function to be queried.
uint64	x2-x4	• Must be Zero (Reserved).

D0059

Table 4.5: PFDI_FEATURES Response

Type	Register	Content
int64	x0	• On success: – SMCCC_RET_SUCCESS: The function ID queried is supported. • On error: – SMCCC_RET_NOT_SUPPORTED: The function ID queried is not supported or PFDI not supported.
uint64	x1-x4	• Must be Zero (Reserved).

4.3.3 Requirements

R0060 A PFDI implementation shall handle calls to the PFDI_FEATURES function, according to the function definition described above, by returning information that is consistent with the PFDI functions described in this specification.

4.4 PFDI_PE_TEST_ID

4.4.1 Usage

D0061 Allows a client to query information about the hardware test mechanism of the PE from which the function is called.

S0062 System Software may use this function to discover information about the vendor and version of the self-test capabilities of the PE, such as the vendor of a Software Test Library. A vendor-specific field is provided, the meaning of which is defined by the vendor. An example usage of this field may be to distinguish between two different libraries from the same vendor for the same PE. The meaning of the major and minor version fields is also vendor-specific.

System Software may use this information, for example, as part of an assertion that the capabilities are as expected, or reported in a log.

The information returned by this function may be different on each PE.

4.4.2 Format

D0063

Table 4.6: PFDI_PE_TEST_ID Request

Type	Register	Content
uint32	w0	• Function ID: 0xC400_02D2
uint64	x1-x4	• Must be Zero (Reserved).

D0064

Table 4.7: PFDI_PE_TEST_ID Response

Type	Register	Content
int64	x0	• On success: ID information in X1. • On error: – PFDI_RET_UNKNOWN: No ID information available. – SMCCC_RET_NOT_SUPPORTED: PFDI not supported.
int64	x1	• On success: – Bits[63:32]: Must be Zero (Reserved). – Bits[31:24]: Vendor ID (See Vendor IDs Table 4.8). – Bits[23:20]: Must be Zero (Reserved). – Bits[19:16]: Vendor IMPLEMENTATION DEFINED value e.g. library name. – Bits[15:8]: Major version. – Bits[7:0]: Minor version. • On error: Must be Zero (Reserved).
uint64	x2-x4	• Must be Zero (Reserved).

D0065

Table 4.8: Vendor IDs

Vendor ID	Vendor Name
0	Arm Limited.
Others	Reserved.

4.4.3 Requirements

R0066 A PFDI implementation shall handle calls to the PFDI_PE_TEST_ID function, according to the function definition described above, to return the vendor and version of the hardware-test capabilities of the PE.

4.5 PFDI_PE_TEST_PART_COUNT

4.5.1 Usage

D0067 Allows a client to retrieve the number of test parts supported by the hardware test capabilities of the PE.

S0068 System Software can schedule tests at the granularity of test parts. The total test suite may be split into a number of test parts. A test part is the smallest schedulable unit of tests. System Software may schedule different test parts, or groups of test parts, to execute according to an application schedule and reducing the period of time in which a PE is offline.

The execution time for each test part should be provided by the vendor of the tests.

Test parts are numbered sequentially from 0 to test_count - 1. This function enables System Software to discover the range of parameter values to use in calls to PDFI_PE_TEST_RUN.

The information returned by this function may be different on each PE.

4.5.2 Format

D0069

Table 4.9: PFDI_PE_TEST_PART_COUNT Request

Type	Register	Content
uint32	w0	• Function ID: 0xC400_02D3
uint64	x1-x4	• Must be Zero (Reserved).

D0070

Table 4.10: PFDI_PE_TEST_PART_COUNT Response

Type	Register	Content
int64	x0	• On success: – Test Count (greater or equals to Zero). • On error: – SMCCC_RET_NOT_SUPPORTED: PFDI not supported.
uint64	x1-x4	• Must be Zero (Reserved).

4.5.3 Requirements

R0071 A PFDI implementation shall handle calls to the PFDI_PE_TEST_PART_COUNT function, according to the function definition described above, to return the number of test parts that are included in the test suite of the PE.

4.6 PFDI_PE_TEST_RUN

4.6.1 Usage

D0072 Allows a client to execute a range of test parts using the hardware test capability for the PE.

S0073 System Software uses this function to execute test parts in the range between start and end inclusive.

System Software is not required to save any PE state prior to calling PFDI_PE_TEST_RUN. The PFDI implementation is responsible for saving and restoring any state that it modifies.

System Software must not expect faults found through tests executed using this function to be reported via the return values. Faults are reported to System Software on a best effort basis, as hardware that has suffered a hardware fault can no longer be relied upon.

How a Firmware implementation handles test failures, including timeouts is IMPLEMENTATION DEFINED and may include performing a system-level fault reaction that puts the system into a safe state and forcibly discontinues execution of System Software.

4.6.2 Format

D0074

Table 4.11: PFDI_PE_TEST_RUN Request

Type	Register	Content
uint32	w0	• Function ID: 0xC400_02D4
int64	x1	• Start: – 0..test_count: lower bound of test part range to execute on this PE.
int64	x2	• End: – 0..test_count: upper bound of test part range to execute on this PE.
uint64	x3-x4	• Must be Zero (Reserved).

D0075

Table 4.12: PFDI_PE_TEST_RUN Response

Type	Register	Content
int64	x0	• On success: – SMCCC_RET_SUCCESS: Test parts executed and did not report an error. • On error: – SMCCC_RET_NOT_SUPPORTED: PFDI not supported. – PFDI_RET_FAULT_FOUND: A test part has executed and detected an error. – PFDI_RET_ERROR: Test part(s) have executed but failed to complete. – PFDI_RET_INVALID_PARAM: ∗ start > end ∗ start > test_count - 1 ∗ end > test_count - 1 ∗ start < 0 ∗ end < 0

Type	Register	Content
int64	x1	• When x0 is FAULT_FOUND: – PFDI_RET_UNKNOWN: test part that triggered the fault cannot be identified. – greater or equals to 0: test part that triggered the fault. • Else Must be Zero (Reserved).
uint64	x2-x4	• Must be Zero (Reserved).

4.6.3 Requirements

R_{0076} A PFDI implementation shall handle calls to the PFDI_PE_TEST_RUN function, according to the function definition described above.

R_{0077} Any PE state modified during execution of the hardware test parts shall be saved and restored by the PFDI implementation.

4.7 PFDI_PE_TEST_RESULT

4.7.1 Usage

D0078 Allows a client to query the result of the last hardware tests executed on the PE at power on.

S0079 System Software uses this function to obtain the results of any tests that were executed by the PFDI implementation at power on, prior to System Software boot.

System Software must not expect all faults found through tests executed at power on to be reported via the return values from this function. Faults are reported to System Software on a best effort basis, as hardware that has suffered a hardware fault can no longer be relied upon.

System Software should call `PFDI_PE_TEST_RESULT` once on each PE during its initialization sequence to determine whether the Platform Firmware reported any test failures detected at power-on.

4.7.2 Format

D0080

Table 4.13: PFDI_PE_TEST_RESULT Request

Type	Register	Content
uint32	w0	• Function ID: 0xC400_02D5
uint64	x1-x4	• Must be Zero (Reserved).

D0081

Table 4.14: PFDI_PE_TEST_RESULT Response

Type	Register	Content
int64	x0	• On success: – SMCCC_RET_SUCCESS: Test parts executed and did not report an error. • On error: – SMCCC_RET_NOT_SUPPORTED: PFDI not supported. – PFDI_RET_FAULT_FOUND: A test part has detected an error. – PFDI_RET_ERROR: Tests failed to complete. – PFDI_RET_NOT_RUN: No tests have executed at power on.
int64	x1	• When x0 is PFDI_RET_FAULT_FOUND: – PFDI_RET_UNKNOWN: test part that triggered the fault cannot be identified. – = 0: test part that triggered the fault. • Else Must be Zero (Reserved).
uint64	x2-x4	• Must be Zero (Reserved).

4.7.3 Requirements

R0082 A PFDI implementation shall report the result of any hardware test parts executed for the PE prior to System Software boot using PFDI_PE_TEST_RESULT, or return a PFDI_RET_NOT_RUN response if no hardware test parts were executed prior to System Software boot.

4.8 PFDI_FW_CHECK

4.8.1 Usage

D0083 Allows a client to request the Platform Firmware to execute and report internal software-based integrity checks.

D0084 Platform Firmware configures hardware and manages the contents of memory that is used to present a Hardware Software Interface that is visible to System Software.

S0085 Platform Firmware may include safety or diagnostic routines that are periodically executed during boot or runtime. System Software uses this function to request execution of these routines and retrieve the result. These checks may involve integrity of internal data structures, firmware configuration, or runtime correctness mechanisms.

D0086 The scope and coverage of these checks are IMPLEMENTATION DEFINED and may vary by PE or platform.

4.8.2 Format

D0087

Table 4.15: PFDI_FW_CHECK Request

Type	Register	Content
uint32	w0	• Function ID: 0xC400_02D6
uint64	x1-x4	• Must be Zero (Reserved).

D0088

Table 4.16: PFDI_FW_CHECK Response

Type	Register	Content
int64	x0	• On success: – SMCCC_RET_SUCCESS: Firmware check executed and did not report an error. • On error: – SMCCC_RET_NOT_SUPPORTED: PFDI not supported. – PFDI_RET_FAULT_FOUND: Firmware check executed and found an error.
uint64	x1-x4	• Must be Zero (Reserved).

4.8.3 Requirements

R0089 A PFDI implementation shall handle calls to the PFDI_FW_CHECK function, according to the function definition described above.

R0090 The `PFDI_FW_CHECK` function shall trigger execution of internal firmware-resident software checks and return the result of the execution.

R0091 System Software should call this function during early initialization and optionally at runtime to verify firmware integrity and correct operation.

R0092 A successful call to `PFDI_FW_CHECK` shall indicate that the Platform Firmware has verified that its internal state and behavior remain consistent with the expectations and assumptions of the System Software interface contract.

X_{0093} The Hardware Software Interface (HSI) exposed by Platform Firmware includes configured hardware state and memory-resident data structures that are accessed by System Software. Faults affecting this state may invalidate assumptions made by System Software and compromise its correct behavior. The `PFDI_FW_CHECK` function allows Platform Firmware to verify the integrity of the HSI and confirm that its state remains consistent with the contract expected by System Software.

R_{0094} A return value of `PFDI_RET_FAULT_FOUND` indicates that Platform Firmware detected a fault during initialization. A return of `SMCCC_RET_SUCCESS` indicates that no fault was detected.

4.9 PFDI_FORCE_ERROR

4.9.1 Usage

D0095 Allows a client to force a subsequent call from the same PE to a PFDI function to return a specified error response.

S0096 System Software may call this function to force the next call to a specified PFDI function to a specified X0 return value. On the next call:

- from the same PE that called PFDI_FORCE_ERROR, and
- to the PFDI function specified in W1

Then:

- The return value specified as a parameter to PFDI_FORCE_ERROR will be returned in X0, and
- Other return values are UNPREDICTABLE.

This function is intended to support testing System Software that handles faults from PFDI function calls.

4.9.2 Format

D0097

Table 4.17: PFDI_FORCE_ERROR Request

Type	Register	Content
uint32	w0	• Function ID: 0xC400_02D7
uint32	w1	• Function ID that should report a failure.
int64	x2	• Value to return in X0 on next call to function identified by W1.
uint64	x3-x4	• Must be Zero (Reserved).

D0098

Table 4.18: PFDI_FORCE_ERROR Response

Type	Register	Content
int64	x0	• On success: – SMCCC_RET_SUCCESS: Error scheduled. • On error: – SMCCC_RET_NOT_SUPPORTED: PFDI not supported. – PFDI_RET_ERROR: Error not scheduled.
uint64	x1-x4	• Must be Zero (Reserved).

4.9.3 Requirements

R0099 A PFDI implementation shall handle calls to the PFDI_FORCE_ERROR function, according to the function definition described above, and return the specified error code in X0 on the next invocation of the specified function from the same PE.

R0100 Following the subsequent call to the specified function, and in the absence of another call to PFDI_FORCE_ERROR, any further calls will behave normally.

Chapter 5
Compliance Requirements

I0101 This section defines the conditions under which a Platform Firmware implementation is considered compliant with the Platform Fault Detection Interface (PFDI).

5.1 Required Functions

R$_{0102}$ The following functions shall be implemented on all Processing Elements (PEs) that support PFDI:

- `PFDI_VERSION`
- `PFDI_FEATURES`
- `PFDI_PE_TEST_ID`
- `PFDI_PE_TEST_PART_COUNT`
- `PFDI_PE_TEST_RUN`
- `PFDI_PE_TEST_RESULT`
- `PFDI_FW_CHECK`
- `PFDI_FORCE_ERROR`

I$_{0103}$ All functions defined in this version of the specification are currently mandatory. Optionality may be introduced in future revisions.

5.2 Version Consistency

R$_{0104}$ All PEs that support PFDI shall return the same version number from the `PFDI_VERSION` function.

5.3 Feature Discovery

R$_{0105}$ The presence of each function on a PE shall be discoverable by System Software using the `PFDI_FEATURES` function.

R$_{0106}$ If a function is not supported on a PE, both the function and the corresponding `PFDI_FEATURES` query shall return `SMCCC_RET_NOT_SUPPORTED`.

5.4 Return Code Semantics

R$_{0107}$ Functions shall return results using the signed 64-bit return code model defined in the SMCCC and this specification.

R$_{0108}$ Return codes must distinguish between successful execution (values greater or equals to 0) and error conditions (values lower than 0).

5.5 PE-Local Consistency

R$_{0109}$ If a function is reported as supported by `PFDI_FEATURES` on a given PE, its behavior shall comply with the requirements defined for that function in this specification.

U$_{0110}$ Function support and behavior may vary between PEs, but consistency must be maintained per PE based on discovery results.

Glossary

Exception Level (EL)

A privileged execution level defined by the Arm architecture. EL3 is the highest level, followed by EL2, EL1, and EL0. Each level supports a different role, from secure firmware to user applications.

Hypervisor

System Software that runs at Exception Level 2 (EL2) and manages one or more virtual machines.

PE (Processing Element)

An abstract execution resource as defined in the Arm architecture. A PE typically maps to a CPU core but may represent other processing resources.

Platform Firmware

Firmware that runs at Exception Level 3 (EL3) or on system control processors. It is responsible for configuring the system at boot time and for providing a Hardware Software Interface (HSI) to System Software. Platform Firmware also implements runtime services, including the Platform Fault Detection Interface (PFDI), and responds to SMC calls initiated by System Software.

Secure Monitor Call (SMC)

An instruction used by lower Exception Levels (e.g., EL1 or EL2) to invoke services in higher privilege levels, typically EL3 firmware.

SMC Calling Convention (SMCCC)

The Arm-defined ABI and behavior model that specifies how System Software calls into Platform Firmware using SMC or HVC instructions, including register usage and return conventions.

Software Test Library (STL)

A vendor-specific software component embedded in Platform Firmware or System Software, used to implement runtime hardware self-tests tailored to the microarchitecture of a PE.

System Software

The operating system or hypervisor that runs at EL1 or EL2 and makes use of services provided by Platform Firmware via standard interfaces such as PFDI.

Test Part

The smallest schedulable unit of a PE self-test. The number and nature of test parts are reported by Platform Firmware and are typically vendor-defined.

Virtual Machine (VM)

A software-isolated execution environment managed by a hypervisor and generally running at EL1.