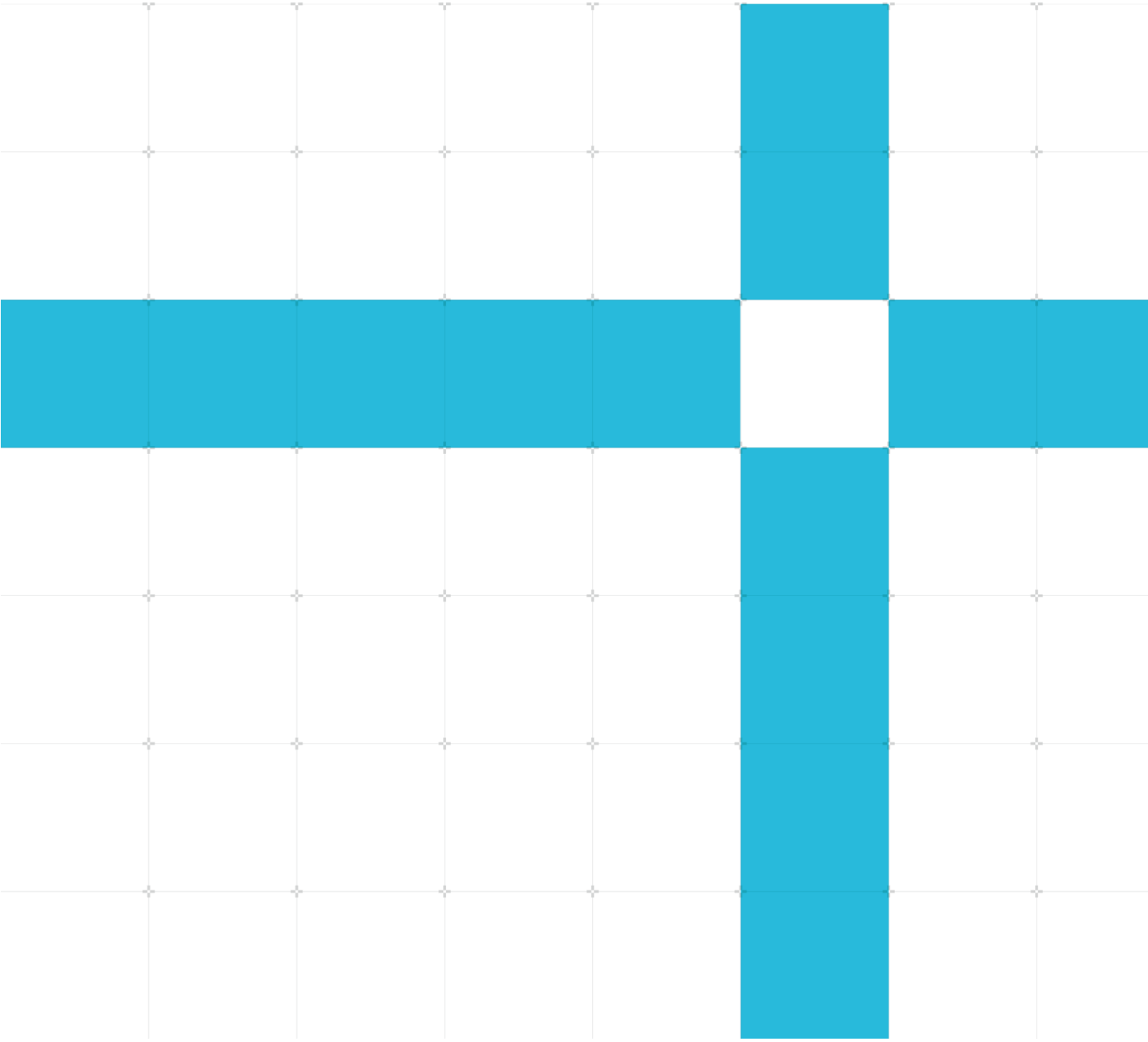
Arm® PFDI Architecture Compliance

Revision: r0p1

Test Scenario

Non-Confidential

Issue 01
ARM040-1254092399-18973



Arm Platform Fault Detection Interface Scenario Document

Release information

Document history

Issue	Date	Confidentiality	Change
01	23 September 2025	Non-Confidential	Initial (BET0-aligned) release 0.8.0

Non-Confidential Proprietary Notice

Arm Limited. Company 02557590 registered in England.

110 Fulbourn Road, Cambridge, England CB1 9NJ.

(LES-PRE-20349)

Confidentiality Status

This document is Non-Confidential. The right to use, copy and disclose this document may be subject to license restrictions in accordance with the terms of the agreement entered into by Arm and the party that Arm delivered this document to.

Unrestricted Access is an Arm internal classification.

Product Status

The information in this document is for a Beta product.

Feedback

Arm welcomes feedback on this product and its documentation. To provide feedback on [Product Name], create a ticket on **https://support.developer.arm.com**.

To provide feedback on the document, fill the following survey:
https://developer.arm.com/documentation-feedback-survey.

Inclusive language commitment

Arm values inclusive communities. Arm recognizes that we and our industry have used terms that can be offensive. Arm strives to lead the industry and create change.

This document includes terms that can be offensive. We will replace these terms in a future issue of this document. If you find offensive terms in this document, please email **terms@arm.com**.

Web Address

www.arm.com

Contents

Contents

1 Introduction

1.1 Product revision status

The r*mpn* identifier indicates the revision status of the product described in this book, for example, r*1*p*2*, where:

rm Identifies the major revision of the product, for example, r*1*.

pn Identifies the minor revision or modification status of the product, for example, p*2*.

1.2 Intended audience

This document is for engineers verifying an implementation of the Arm® Platform Fault Detection Interface (PFDI) using the Arm Compliance Suite (ACS) and associated harness.

1.3 Conventions

The following subsections describe conventions used in Arm documents.

1.3.1 Glossary

The Arm Glossary is a list of terms used in Arm documentation, together with definitions for those terms. The Arm Glossary does not contain terms that are industry standard unless the Arm meaning differs from the generally accepted meaning.

See the Arm Glossary for more information: **https://developer.arm.com/glossary**.

1.3.2 Typographical Conventions

Convention	Use
italic	Introduces citations.
bold	Highlights interface elements, such as menu names. Denotes signal names. Also used for terms in descriptive lists, where appropriate.
`monospace`	Denotes text that you can enter at the keyboard, such as commands, file and program names, and source code.
`monospace` **`bold`**	Denotes language keywords when used outside example code.
`monospace underline`	Denotes a permitted abbreviation for a command or option. You can enter the underlined text instead of the full command or option name.
<and>	Encloses replaceable terms for assembler syntax where they appear in code or code fragments. For example: `MRC p15, 0, <Rd>, <CRn>, <CRm>, <Opcode_2>`

Convention	Use
SMALL CAPITALS	Used in body text for a few terms that have specific technical meanings, that are defined in the Arm® Glossary. For example, IMPLEMENTATION DEFINED, IMPLEMENTATION SPECIFIC, UNKNOWN, and UNPREDICTABLE.

1.4 Useful resources

Arm products	Document ID	Confidentiality
Platform Fault Detection Interface Specification	110468	Non-Confidential

Arm architecture and specifications	Document ID	Confidentiality
Arm® Architecture Reference Manual for A-profile architecture	DDI0487F	Non-Confidential

Arm tests its PDFs only in Adobe Acrobat and Acrobat Reader. Arm cannot guarantee the quality of its documents when used with any other PDF reader.

Adobe PDF reader products can be downloaded at **http://www.adobe.com**.

1.5 Feedback

Arm welcomes feedback on this product and its documentation.

1.5.1 Feedback on this product

If you have any comments or suggestions about this product, contact your supplier and give:

- The product name Arm Platform Fault Detection Interface Scenario.
- The product revision or version.
- An explanation with as much information as you can provide. Include symptoms and diagnostic procedures if appropriate.

1.5.2 Feedback on content

If you have comments on content, send an email to **support-systemready-acs@arm.com** and give:

- The title Platform Fault Detection Interface Scenario.
- The number ARM040-1254092399-18973.
- If applicable, the page number(s) to which your comments refer.
- A concise explanation of your comments.
- Arm also welcomes general suggestions for additions and improvements.

2 Platform Fault Detection Interface

The PFDI defines a standard interface that enables System Software to request fault detection checks from Platform Firmware.

PFDI allows firmware-resident mechanisms to perform platform or PE-level validation of hardware and firmware state. The interface is designed to be implemented by Platform Firmware and invoked from System Software using SMC calls in accordance with Arm SMCCC.

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

2.1 Platform Fault Detection Interface

Tests are organized by category. Each entry lists Test number, Rule ID (from the PFDI spec), Level, Scenario, and Algorithm. Unless stated otherwise, functions return a signed 64-bit value in x0 (non-negative = success; negative = error).

2.1.1 PFDI_VERSION

Rule ID	Test Number	Scenario	Algorithm
R0053	1	Verify that on a successful PFDI_VERSION call, the platform firmware: • returns the supported PFDI version in x0 • clears x1 to x4 to zero.	Call PFDI_VERSION, capturing x0 and outputs x1 to x4. • Validate that status in x0 indicates success and reserved bits [63:31] are zero. • Decode major and minor version fields from x0 and compare with expected constants. • Confirm that registers x1, x2, x3, and x4 are all zero.
R0155	-	Invoke PFDI_VERSION with non-zero values in x1 to x4. Verify that the return code in x0 is PFDI_RET_INVALID_PARAM.	Call PFDI_VERSION, with registers x1 to x4 deliberately set to non-zero values. • Capture the return value in x0. • Check that x0 equals PFDI_RET_INVALID_PARAM. • Pass If x0 == PFDI_RET_INVALID_PARAM. • Fail If x0 indicates success or any other error code. Test is not implemented yet, will be part of the future release.

2.1.2 PFDI_FEATURES

Rule ID	Test Number	Scenario	Algorithm
R0060	4	Invoke PFDI_FEATURES with a supported function ID in w1, capturing x0 and outputs x1..x4. • Confirm x0 == SMCCC_RET_SUCCESS. • Confirm x1, x2, x3, and x4 are all zero.	Call PFDI_FEATURE and store x0 to x4 for all PE's. For each PE, verify results: • Check if x0 == SMCCC_RET_SUCCESS, otherwise fail and print the error. • Check if x1 == 0 && x2 == 0 && x3 == 0 && x4 == 0, otherwise fail and print the non-zero regs. • If both checks pass, test is pass for that PE.
R0156	10	Call PFDI_FEATURES with a valid range but unsupported function ID • Check that x0 is SMCCC_RET_NOT_SUPPORTED. • Confirm x1 to x4 are zero	For each PE, invoke PFDI_FEATURES and store x0 to x4. • Verify x0 == SMCCC_RET_NOT_SUPPORTED. If not, fail for that PE and print the returned status. • Confirm that x1 to x4 are zero • Mark PASS only if every PE returns SMCCC_RET_NOT_SUPPORTED for the reserved/unsupported ID.
R0157	16	Call PFDI_FEATURES with an invalid function ID. • Verify x0 is PFDI_RET_INVALID_PARAM. • Confirm x1 to x4 are zero	Call PFDI_FEATURES on every PE, store x0 to x4. • For each PE, verify x0 == PFDI_RET_INVALID_PARAM. If not, fail and print PE index and returned x0. • Confirm x1 == 0 && x2 == 0 && x3 == 0 && x4 == 0. • If both checks pass, mark PASS for that PE and aggregate results.
R0179	-	Invoke PFDI_FEATURES with non-zero values in x1 to x4. Verify that the return code in x0 is PFDI_RET_INVALID_PARAM.	Call PFDI_FEATURES, with registers x1 to x4 deliberately set to non-zero values. • Capture the return value in x0. • Check that x0 equals PFDI_RET_INVALID_PARAM. • Pass If x0 == PFDI_RET_INVALID_PARAM. • Fail If x0 indicates success or any other error code. Test is not implemented yet, will be part of the future release.

2.1.3 PFDI_PE_TEST_ID

Rule ID	Test Number	Scenario	Algorithm
R0066	5	Call PFDI_PE_TEST_ID, capture x0 to x4. • Check x0 == SMCCC_RET_SUCCESS. • Validate reserved bits in x1 ([63:32] and [23:20]) are zero. • Confirm x2, x3, and x4 are zero	Invoke PFDI_PE_TEST_ID on every PE, store x0 to x4 If x0 == SMCCC_RET_SUCCESS, print the returned value and PE index. • Validate x1 encoding, reserved bits [63:32] == 0 and [23:20] == 0, otherwise fail and print x1 and PE index. • Confirm x2 == 0 && x3 == 0 && x4 == 0, otherwise fail and print the non-zero regs. • Optionally decode and log from x1: major = [15:8], minor = [7:0], vendor_id = [31:24]. • If all checks pass for a PE, mark PASS for that PE. If x0 == PFDI_RET_UNKNOWN • Confirm x1 == 0, otherwise FAIL and print the non-zero register If x0 is any other value ten mark as FAIL for that PE
R0158	-	Invoke PFDI_PE_TEST_ID with non-zero values in x1 to x4. • Verify that the return code in x0 is PFDI_RET_INVALID_PARAM.	Call PFDI_ PE_TEST_ID, with registers x1 to x4 deliberately set to non-zero values. • Capture the return value in x0. • Check that x0 equals PFDI_RET_INVALID_PARAM. • Pass If x0 == PFDI_RET_INVALID_PARAM. • Fail If x0 indicates success or any other error code. Test is not implemented yet, will be part of future release.

2.1.4 PFDI_PE_TEST_PART_COUNT

Rule ID	Test Number	Scenario	Algorithm
R0071	6	Call PFDI_PE_TEST_PART_COUNT, capture x0 to x4. • Verify x0 indicates success and represents the test-part count. • Confirm x1, x2, x3, and x4 are zero.	Invoke PFDI_PE_TEST_PART_COUNT on every PE, store x0 to x4. • Verify success: check that x0 does not indicate an error (e.g., x0 >= PFDI_ACS_SUCCESS). If error, fail and print x0 and PE index. • Treat x0 as the part count; optionally log it for each PE. • Confirm x1 == 0 && x2 == 0 && x3 == 0 && x4 == 0, otherwise fail and print the non-zero regs and PE index. • If all checks pass for a PE, mark PASS for that PE
R0160	-	Invoke PFDI_PE_TEST_PART_COUNT with non-zero values in x1 to x4. • Verify that the return code in x0 is PFDI_RET_INVALID_PARAM.	Call PFDI_ PE_TEST_PART_COUNT, with registers x1 to x4 deliberately set to non-zero values. • Capture the return value in x0. • Check that x0 equals PFDI_RET_INVALID_PARAM. • Pass If x0 == PFDI_RET_INVALID_PARAM. • Fail If x0 indicates success or any other error code. Test is not implemented yet, will be part of future release.

2.1.5 PFDI_PE_TEST_RUN

Rule ID	Test Number	Scenario	Algorithm
R0076	7	Get test-part count, select a valid [start, end] on each PE. Invoke PFDI_PE_TEST_RUN and store x0 to x4. • Check x0 == SMCCC_RET_SUCCESS and Confirm x1 to x4 are zero. • Check x0 == PFDI_RET_FAULT_FOUND and x1 == PFDI_RET_UNKNOWN or index of faulty test part and confirm x2 to x4 are zero • Check x0 == PFDI_RET_ERROR and confirm x1 to x4 are zero	For each PE, query PFDI_PE_TEST_PART_COUNT to get N. If error. • Choose a valid range [start=0, end=N-1]. • Call PFDI_PE_TEST_RUN and capture x0. Verify If x0 == SMCCC_RET_SUCCESS • x1 == 0 && x2 == 0 && x3 == 0 && x4 == 0. • If any check fails, print details and mark FAIL for that PE. Verify If x0 == PFDI_RET_FAULT_FOUND • Check if x1 is PFDI_RET_UNKNOWN or faulty test part index • Check if x2 == 0 && x3 == 0 && x4 == 0. • If any check fails, print details and mark FAIL for that PE. Verify if x0 == PFDI_RET_ERROR • Check ifx1 == 0 && x2 == 0 && x3 == 0 && x4 == 0. • If any check fails, print details and mark FAIL for that PE. FAIL if x0 is any other value
R0076	13	Call PFDI_PE_TEST_RUN (-1, -1) on all PEs and capture x0. • Verify the call isn't rejected as invalid; record x0 as the execution result.	For each PE, call PFDI_PE_TEST_RUN with start=-1, end=-1. • Verify the call is accepted (i.e., not an invalid-parameter status). • Treat x0 as the result (SMCCC_RET_SUCCESS, PFDI_ACS_FAULT_FOUND, or PFDI_ACS_ERROR) and log it per PE. • Mark PASS if the interface accepts -1, -1 and executes, otherwise FAIL.
R0163	-	Invoke PFDI_PE_TEST_RUN with non-zero values in x3 to x4. • Verify that the return code in x0 is PFDI_RET_INVALID_PARAM.	Call PFDI_ PE_TEST_RUN, with registers x3 to x4 deliberately set to non-zero values. • Capture the return value in x0. • Check that x0 equals PFDI_RET_INVALID_PARAM. • Pass If x0 == PFDI_RET_INVALID_PARAM. • Fail If x0 indicates success or any other error code. Test is not implemented yet, will be part of future release.
R0164	14*	Call PFDI_PE_TEST_RUN with each invalid (Start greater than End) pair. • Check x0 == PFDI_RET_INVALID_PARAM.	Construct invalid ((1, 0) Start > End) cases and run PFDI_PE_TEST_RUN: Verify: • x0 == PFDI_RET_INVALID_PARAM. • x1 (Fault Id) == 0. • If either check fails, print PE index, case index, returned x0/fault_id, and mark FAIL for that PE. • If all cases pass on a PE, mark PASS for that PE.

Rule ID	Test Number	Scenario	Algorithm
R0165	14*	Get test part via PFDI_PE_TEST_PART_COUNT. Call PFDI_PE_TEST_RUN with each invalid (Start greater than Max test parts) pair. • Check x0 == PFDI_RET_INVALID_PARAM.	For each PE, get the test-part count N = PFDI_PE_TEST_PART_COUNT. If this call fails, mark FAIL for that PE. Construct invalid ((N, N-1) Start > max test parts) cases and run PFDI_PE_TEST_RUN: Verify: • x0 == PFDI_RET_INVALID_PARAM. • x1 == 0 && x2 == 0 && x3 == 0 && x4 == 0. • If either check fails, print PE index, case index, returned x0/fault_id, and mark FAIL for that PE. • If all cases pass on a PE, mark PASS for that PE.
R0166	14*	Get test part via PFDI_PE_TEST_PART_COUNT. Call PFDI_PE_TEST_RUN with each invalid (End greater than Max test parts) pair. • Check x0 == PFDI_RET_INVALID_PARAM.	For each PE, get the test-part count N = PFDI_PE_TEST_PART_COUNT; if this call fails, mark FAIL for that PE. Construct invalid ((0, N) End > max test parts) cases and run PFDI_PE_TEST_RUN: Verify: • x0 == PFDI_RET_INVALID_PARAM. • x1 == 0 && x2 == 0 && x3 == 0 && x4 == 0. • If either check fails, print PE index, case index, returned x0/fault_id, and mark FAIL for that PE. • If all cases pass on a PE, mark PASS for that PE.
R0167	14*	Call PFDI_PE_TEST_RUN with each invalid (Start is -1 and End is not -1 and vice versa) pair. • Check x0 == PFDI_RET_INVALID_PARAM.	Construct invalid ((-1, 0) Start = -1, End != -1 or (0, -1) End = -1, Start != -1) cases and run PFDI_PE_TEST_RUN: Verify: • x0 == PFDI_RET_INVALID_PARAM. • x1 == 0 && x2 == 0 && x3 == 0 && x4 == 0. • If either check fails, print PE index, case index, returned x0, and mark FAIL for that PE. • If all cases pass on a PE, mark PASS for that PE.
R0168	14*	Get test part N via PFDI_PE_TEST_PART_COUNT. Call PFDI_PE_TEST_RUN with each invalid (Start less than -1 or vice versa) pair. • Check x0 == PFDI_RET_INVALID_PARAM.	For each PE, get the test-part count N = PFDI_PE_TEST_PART_COUNT; if this call fails, mark FAIL for that PE. Construct invalid ((-2, N-1) → Start < -1 or (0, -2) → End < -1) cases and run PFDI_PE_TEST_RUN: Verify: • x0 == PFDI_RET_INVALID_PARAM. • x1 == 0 && x2 == 0 && x3 == 0 && x4 == 0. • If either check fails, print PE index, case index, returned x0, and mark FAIL for that PE. • If all cases pass on a PE, mark PASS for that PE.

Note: Test IDs marked with an asterisk (*) are part of a single combined test. The split into separate tests will be handled in the next release.

2.1.6 PFDI_PE_TEST_RESULT

Rule ID	Test Number	Scenario	Algorithm
R0082	8	Call PFDI_PE_TEST_RESULT, capture x0 to x4. • Check x0 == SMCCC_RET_SUCCESS and confirm x1 to x4 are zero. • Check x0 == PFDI_RET_FAULT_FOUND and x1 == PFDI_RET_UNKNOWN or index of faulty test part and confirm x2 to x4 are zero • Check x0 == PFDI_RET_ERROR or PFDI_RET_NOT_RUN and confirm x1 to x4 are zero	Invoke PFDI_PE_TEST_RESULT on every PE, store x0 to x4. Verify If x0 == SMCCC_RET_SUCCESS • x1 == 0 && x2 == 0 && x3 == 0 && x4 == 0. • If any check fails, print details and mark FAIL for that PE. Verify If x0 == PFDI_RET_FAULT_FOUND • Check if x1 is PFDI_RET_UNKNOWN or faulty test part index • Check if x2 == 0 && x3 == 0 && x4 == 0. • If any check fails, print details and mark FAIL for that PE. Verify if x0 == PFDI_RET_ERROR or PFDI_RET_NOT_RUN • Check ifx1 == 0 && x2 == 0 && x3 == 0 && x4 == 0. • If any check fails, print details and mark FAIL for that PE. FAIL if x0 is any other value
R0172	-	Invoke PFDI_PE_TEST_RESULT with non-zero values in x1 to x4. • Verify that the return code in x0 is PFDI_RET_INVALID_PARAM.	Call PFDI_ PE_TEST_RESULT, with registers x1 to x4 deliberately set to non-zero values. • Capture the return value in x0. • Check that x0 equals PFDI_RET_INVALID_PARAM. • Pass If x0 == PFDI_RET_INVALID_PARAM. • Fail If x0 indicates success or any other error code. Test is not implemented yet, will be part of future release.

2.1.7 PFDI_FW_CHECK

Rule ID	Test Number	Scenario	Algorithm
R0089	9	Invoke PFDI_FW_CHECK, capture x0 to x4. • Check x0 == SMCCC_RET_SUCCESS or PFDI_RET_FAULT_FOUND • Confirm x1, x2, x3, and x4 are all zero.	Call PFDI_FW_CHECK on all PEs and store x0 to x4 per-PE. For each PE, treat x0 as the firmware integrity result. • Verify x0 == SMCCC_RET_SUCCESS or PFDI_RET_FAULT_FOUND, otherwise fail and print the returned value and PE index • Verify x1 == 0 && x2 == 0 && x3 == 0 && x4 == 0. If any is non-zero, FAIL and print the offending registers and PE index. • If the zero-register check pass for a PE, mark PASS for that PE.
R0173	-	Invoke PFDI_FW_CHECK with non-zero values in x1 to x4. • Verify that the return code in x0 is PFDI_RET_INVALID_PARAM.	Call PFDI_ FW_CHECK, with registers x1 to x4 deliberately set to non-zero values. • Capture the return value in x0. • Check that x0 equals PFDI_RET_INVALID_PARAM. • Pass If x0 == PFDI_RET_INVALID_PARAM. • Fail If x0 indicates success or any other error code. Test is not implemented yet, will be part of future release

2.1.8 PFDI_FORCE_ERROR

Rule ID	Test Number	Scenario	Algorithm
R0099	12	Call PFDI_FORCE_ERROR for a supported PFDI function ID and verify x0 == SMCCC_RET_SUCCESS or PFDI_RET_ERROR. • Call that function once and verify x0 matches the injected error.	For each PE and for each targeted function ID, call PFDI_FORCE_ERROR and capture x0. Verify x0 == SMCCC_RET_SUCCESS. • Verify x1 == 0 && x2 == 0 && x3 == 0 && x4 == 0. If any is non-zero, FAIL and print the offending registers and PE index. • Immediately invoke the specified function once. • Check that the function's x0 equals the injected error code, otherwise fail and print the PE, function ID, and the observed returns. Verify x0 == PFDI_RET_ERROR. • Verify x1 == 0 && x2 == 0 && x3 == 0 && x4 == 0. If any is non-zero, FAIL and print the offending registers and PE index. FAIL if x0 is any other value
R0100	12	Call PFDI_FORCE_ERROR for a supported PFDI function ID and verify x0 == SMCCC_RET_SUCCESS. • Call that function once and verify x0 matches the injected error. • Call the same function again and verify it returns to normal behavior without the injected error.	For each PE and for each targeted function ID, call PFDI_FORCE_ERROR and capture x0. Observed the injected error, invoke the same function again. • Verify that it now behaves normally. • If the second invocation still returns the injected error code, fail and print the PE, function ID, and the repeated status.
R0176	-	Invoke PFDI_FORCE_ERROR on current PE • use a function ID not in 0xC400_02D0–0xC400_02D7 For each case, verify that the return code in x0 is PFDI_RET_INVALID_PARAM	Call PFDI_FORCE_ERROR, • Function ID as invalid For both cases above, • Pass if x0 equals PFDI_RET_INVALID_PARAM. • Verify x1 == 0 && x2 == 0 && x3 == 0 && x4 == 0 • Fail If x0 indicates success or any other error code Test is not implemented yet, will be part of future release.
R0180	-	Invoke PFDI_FORCE_ERROR with non-zero values in x1 to x4. Verify that the return code in x0 is PFDI_RET_INVALID_PARAM.	Call PFDI_FORCE_ERROR, with registers x1 to x4 deliberately set to non-zero values. • Capture the return value in x0. • Check that x0 equals PFDI_RET_INVALID_PARAM. • Pass If x0 == PFDI_RET_INVALID_PARAM. • Verify x1 == 0 && x2 == 0 && x3 == 0 && x4 == 0 • Fail If x0 indicates success or any other error code. Test is not implemented yet, will be part of future release

2.1.9 Compliance Requirements

Rule ID	Test Number	Scenario	Algorithm
R0102	3	Call PFDI_FEATURES for each mandatory function ID on all PEs and record x0. • Check x0 == SMCCC_RET_SUCCESS for every ID on every PE. • Report failure for any PE where a mandatory ID does not return success.	For each PE, iterate function IDs in the mandatory list: 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. • For each ID, call PFDI_FEATURES and store x0. • For each PE and each function ID, verify x0 == SMCCC_RET_SUCCESS. • If not, mark FAIL for that PE and print function ID, PE index, and the returned status. • If all mandatory IDs return success on all PEs, mark PASS.
R0040	11	Invoke representative PFDI calls and capture X5 to X17 before/after each call. • Verify X5 to X17 are identical pre/post for every call checked. • Any difference in X5 to X17 fail, otherwise pass	For each PFDI function Set x0 to the function ID. Store X5 to X17 before/after the SMC/HVC. • Compare pre vs post, on any mismatch, print the register and the before/after values and mark FAIL. • If no mismatches are seen for all invoked functions, mark PASS.
R0104	4	Call PFDI_VERSION on all PEs and store x0 to x4. • Validate status in x0 and that reserved bits are zero. • Confirm x1 to x4 are zero. • Decode major/minor and verify every PE reports the same version.	Invoke PFDI_VERSION on every PE, capturing x0 to x4 • For each PE, validate x0 indicates success and reserved bits [63:31] are zero, otherwise fail and print PE index. • Confirm x1 == 0 && x2 == 0 && x3 == 0 && x4 == 0, otherwise fail and print the non-zero regs. • Compare versions across Pes, all (major, minor) pairs must match. If any mismatch fail and print both versions and PE index. • If all checks pass for all PEs, mark the test PASS.
R0154	17	Call a PFDI function ID that's within the PFDI range but not implemented on the current PE. • Confirm the call returns SMCCC_RET_NOT_SUPPORTED in x0. • Optionally confirm x1 to x4 are zero if your implementation requires zeroing on error	For each PE, invoke a valid-range but unsupported PFDI function and capture x0 to x4. • Use a reserved ID. • Verify the return code: Expect x0 == SMCCC_RET_NOT_SUPPORTED. If not, FAIL and print the PE index and returned value. • check x1 to x4 == 0 and print any non-zero values. • Repeat for all PEs and aggregate results; PASS only if every PE returns SMCCC_RET_NOT_SUPPORTED
R0041	-	In a guest VM where PFDI is exposed, invoke PFDI calls: Confirm that returns, side registers (x1 to x4), and per-PE effects match the spec behaviors.	Indirectly Covered When a hypervisor implementation exposes PFDI to guest virtual machines, it shall forward the function ID, input arguments, and the originating PE ID unmodified to Platform Firmware, such that the behavior and results of the function invocation are consistent with those of a direct call from System Software.

Appendix A revisions

This appendix describes the technical changes between released issues of this book.

Table 0-1 Issue 01

Change	Location
Initial PFDI ACS Scenario (BET0-aligned)	Entire document