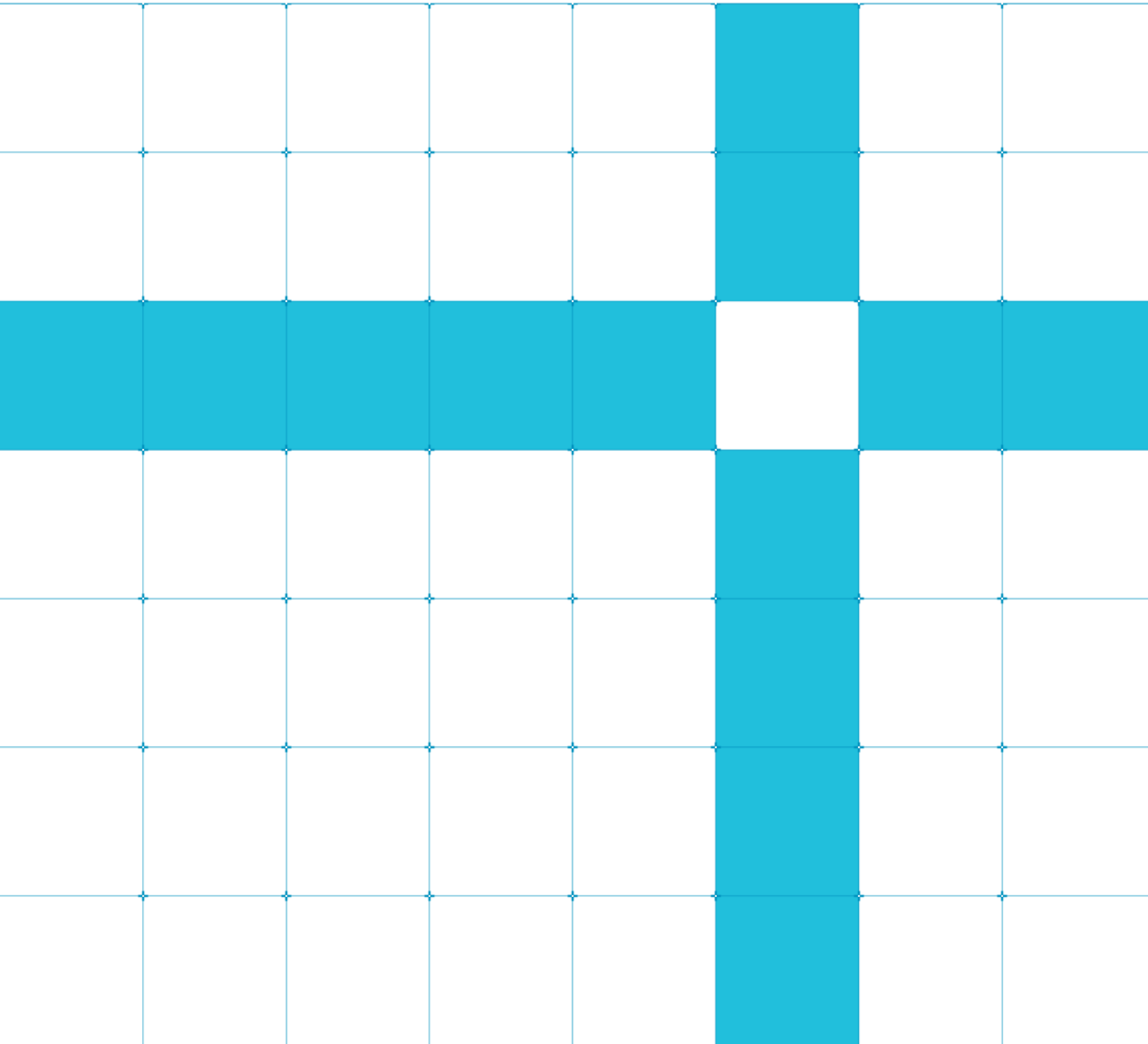




Arm® TBSA-v8M
Architecture Test Scenario
Document

Version <1.2>



Arm® TBSA-v8M

Arch Test Scenario Document

Copyright © 2018 Arm Limited (or its affiliates). All rights reserved.

Release Information

Document History

Version	Date	Confidentiality	Change
1.0	18 June 2018	Non-Confidential	First version of the document.
1.1	29 June 2018	Non-Confidential	Second version of the document.
1.2	30 August 2019	Non-Confidential	Third version of the document.

Non-Confidential Proprietary Notice

This document is protected by copyright and other related rights and the practice or implementation of the information contained in this document may be protected by one or more patents or pending patent applications. No part of this document may be reproduced in any form by any means without the express prior written permission of Arm. **No license, express or implied, by estoppel or otherwise to any intellectual property rights is granted by this document unless specifically stated.**

Your access to the information in this document is conditional upon your acceptance that you will not use or permit others to use the information for the purposes of determining whether implementations infringe any third party patents.

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

This document may include technical inaccuracies or typographical errors.

TO THE EXTENT NOT PROHIBITED BY LAW, IN NO EVENT WILL ARM BE LIABLE FOR ANY DAMAGES, INCLUDING WITHOUT LIMITATION ANY DIRECT, INDIRECT, SPECIAL, INCIDENTAL, PUNITIVE, OR CONSEQUENTIAL DAMAGES, HOWEVER CAUSED AND REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF ANY USE OF THIS DOCUMENT, EVEN IF ARM HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

This document consists solely of commercial items. You shall be responsible for ensuring that any use, duplication or disclosure of this document complies fully with any relevant export laws and regulations to assure that this document or any portion thereof is not exported, directly or indirectly, in violation of such export laws. Use of the word "partner" in reference to Arm's customers is not intended to create or refer to any partnership relationship with any other company. Arm may make changes to this document at any time and without notice.

If any of the provisions contained in these terms conflict with any of the provisions of any click through or signed written agreement covering this document with Arm, then the click through or signed written agreement prevails over and supersedes the conflicting provisions of these terms. This document may be translated into other languages for convenience, and you agree that if there is any conflict between the English version of this document and any translation, the terms of the English version of the Agreement shall prevail.

The Arm corporate logo and words marked with ® or ™ are registered trademarks or trademarks of Arm Limited (or its subsidiaries) in the US and/or elsewhere. All rights reserved. Other brands and names mentioned in this document may be the trademarks of their respective owners. Please follow Arm's trademark usage guidelines at <http://www.arm.com/company/policies/trademarks>.

Copyright © 2018 Arm Limited (or its affiliates). All rights reserved.

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 beta, that is for a product under development.

Web Address

<http://www.arm.com>

Contents

1 About this document	8
1.1. References	8
1.2. Terms and Abbreviations	8
1.3. Scope	8
2 Introduction	9
2.1. Limitations of the TBSA-v8M test suite	9
3 Correlation between architecture and tests	11
4 Verification scenarios	16
4.1. Base system scenarios	16
4.1.1 R010_TBSA_BASE: A Non-Trusted world operation must only access Non-Trusted world assets	16
4.1.2 R020_TBSA_BASE: A Trusted world operation can access both Trusted and Non-Trusted world assets	16
4.1.3 R030_TBSA_BASE: The SoC must be based on an Armv8-M architecture PE with the Security Extension and MPU implemented	17
4.2. Infrastructure scenarios	17
4.2.1 R010_TBSA_INFRA: A Trusted operation can issue Secure or Non-secure transactions	17
4.2.2 R020_TBSA_INFRA: A Non-Trusted operation must only issue Non-secure transactions	17
4.2.3 R030_TBSA_INFRA: A Non-secure Transaction must only access Non-secure storage	17
4.2.4 R040_TBSA_INFRA If programmable address remapping logic is implemented in the interconnect then its configuration must only be possible from the Trusted world	17
4.2.5 R050_TBSA_INFRA A unified address map that uses target side filtering to disambiguate Non-secure and Secure transactions must only permit all Secure or all Non-secure transactions to any one region. Secure and Non-secure aliased accesses to the same address region are not permitted.	18
4.2.6 R060_TBSA_INFRA The target transaction filters configuration space must only be accessed from the Trusted world	18
4.2.7 R070_TBSA_INFRA Security Exception Interrupts must be wired or configured as Secure interrupt sources	19
4.2.8 R080_TBSA_INFRA Configuration of the on-chip interconnect that modifies routing or the memory map must only be possible from the Trusted world unless it is not possible for such modifications to affect secure transactions	19
4.2.9 R100_TBSA_INFRA If shared volatile storage is implemented, then the associated location or region must be scrubbed before it can be reallocated from Trusted to Non-Trusted	20
4.2.10 R110_TBSA_INFRA If shared volatile storage is implemented, then the associated location must not be executable or NSC immediately after it is reallocated from Non-Trusted to Trusted	20
4.2.11 R120_TBSA_INFRA An interrupt originating from a Trusted operation must be mapped only to a Trusted target. By default, this must be the case following a system reset	20
4.2.12 R130_TBSA_INFRA Any configuration to mask or route a Trusted interrupt must only be carried out from the Trusted world	21
4.2.13 R140_TBSA_INFRA The interrupt network might be configured to route an interrupt originating from a Trusted operation to a Non-Trusted target	21
4.2.14 R150_TBSA_INFRA Any status flags recording Trusted interrupt events must only be read from the Trusted world, unless specifically configured by the Trusted world, to be readable by the Non-Trusted world	21

4.2.15 R160_TBSA_INFRA A TBSA-v8M system must integrate a Secure RAM	22
4.2.16 R170_TBSA_INFRA Secure RAM must be mapped into the Trusted world only	22
4.2.17 R180_TBSA_INFRA If the mapping of Secure RAM into regions is programmable, then configuration of the regions must only be possible from the Trusted world	22
4.2.18 R190_TBSA_INFRA The advanced power mechanism must integrate a Trusted management function to control clocks and power. It must not be possible to directly access clock and power functionality from the Non-trusted world	22
4.2.19 R210_TBSA_INFRA If access to a peripheral or a subset of its operations can be dynamically switched between Trusted world and Non-trusted world, then this must be done only under the control of the Trusted world	23
4.2.20 R220_TBSA_INFRA If the peripheral stores assets in local embedded storage, a Non-trusted operation must not be able to access the local assets of a Trusted operation	23
4.2.21 R230_TBSA_INFRA A Trusted operation must be able to distinguish the originating world of commands and data arriving at its interface, by using the address.	23
4.3. Fuse scenarios.....	23
4.3.1 R020_TBSA_FUSE: A fuse is permitted to transition in one direction only, from its unprogrammed state to its programmed state. The reverse operation must be prevented.....	23
4.3.2 R040_TBSA_FUSE: It must be possible to blow at least a subset of the fuses when the device has left the silicon manufacturing facility.....	24
4.3.3 R080_TBSA_FUSE: A confidential fuse whose recipient is a hardware IP must not be readable by any software process	24
4.3.4 R090_TBSA_FUSE: A confidential fuse whose recipient is a hardware IP must be connected to the IP using a path that is not visible to software or any other hardware IP.....	24
4.3.5 R100_TBSA_FUSE: A confidential fuse whose recipient is a software process might be readable by that process and must be readable by privileged software	25
4.3.6 R110_TBSA_FUSE: A confidential fuse whose recipient is a Trusted world software process must be protected by a hardware filtering mechanism that can only be configured by secure software, for example an NS-bit filter	25
4.3.7 R120_TBSA_FUSE: It must be possible to fix a lockable fuse in its current state, regardless of whether it is programmed or unprogrammed... ..	25
4.3.8 R140_TBSA_FUSE: A bulk fuse must also be a lockable fuse to ensure that any unprogrammed bits cannot be programmed later	26
4.4. Key scenarios	26
4.4.1 R010_TBSA_KEY: A key must be treated as an atomic unit. It must not be possible to use a key in a cryptographic operation before it has been fully created, during an update operation, or during its destruction	26
4.4.2 R020_TBSA_KEY: Any operations on a key must be atomic. It must not be possible to interrupt the creation, update, or destruction of a key	26
4.4.3 R030_TBSA_KEY: When a key is no longer required by the system, it must be put beyond use to prevent a hack at a later time from revealing it.....	26
4.4.4 R070_TBSA_KEY: A static key must be stored in an immutable structure, for example a ROM or a set of Bulk-Lockable fuses	27
4.4.5 R140_TBSA_KEY: A Trusted hardware key must not be directly accessible by any software	27
4.4.6 R160_TBSA_KEY: A TBSA-v8M device must either entirely embed a root of trust public key (ROTPK), or the information that is needed to securely identify it	28
4.4.7 R180_TBSA_KEY: An elliptic-curve-based ROTPK must be at least 256 bits in size	28
4.4.8 R190_TBSA_KEY: An RSA-based ROTPK must be at least 3072 bits in size	28
4.4.9 R200_TBSA_KEY: If a cryptographic hash of the ROTPK is stored in on chip non-volatile memory, rather than the key itself, it must be immutable	28
4.4.10 R220_TBSA_KEY: A TBSA-v8M device must embed a hardware unique root key (HUK) in Confidential-Lockable-Bulk fuses	28
4.4.11 R240_TBSA_KEY: The HUK must only be accessible by Trusted code or Trusted hardware that acts on behalf of Trusted code	29
4.5. Boot scenarios	29
4.5.1 R010_TBSA_BOOT A TBSA-M device must embed a Boot ROM with the initial code that is needed to perform a Trusted system boot.	29

4.5.2 R020_TBSA_BOOT If the device supports warm boot, a flag or register that survives warm boot must exist, to enable distinguishing between warm and cold boots. This register or flag must be programmable only by the Trusted world and must be reset after a cold boot	29
4.5.3 R030_TBSA_BOOT On a cold boot, the primary processor must boot from the Boot ROM. It must not be possible to boot from any other storage unless Trusted Kernel debug is enabled	30
4.5.4 R090_TBSA_BOOT If a boot status register is implemented, then it must be accessible only by the Trusted world	30
4.5.5 R100_TBSA_BOOT In an Assisted architecture, the key to decrypt the Trusted Boot Firmware image must be visible only to the acceleration peripheral	30
4.6. Timer scenarios.....	30
4.6.1 R030_TBSA_TIME At least one Trusted timer must exist	30
4.6.2 R040_TBSA_TIME A Trusted timer must only be modified by a Trusted access. Examples of modifications are the timer being refreshed, suspended, or reset	31
4.6.3 R050_TBSA_TIME The clock source that drives a Trusted timer must be a Trusted clock source	31
4.6.4 R060_TBSA_TIME At least one Trusted watchdog timer must exist.....	31
4.6.5 R070_TBSA_TIME After a system reset, a Trusted watchdog timer must be started before the execution of immutable boot code transfers control to the next firmware stage.....	31
4.6.6 R080_TBSA_TIME A Trusted watchdog timer must only be modified by a Trusted access. Examples of modifications are the timer being refreshed, suspended, or reset.....	32
4.6.7 R100_TBSA_TIME A Trusted watchdog timer must be able to trigger a reset of the SoC, after a predefined period. This value can be fixed in hardware or programmed by a Trusted access	32
4.6.8 R110_TBSA_TIME A Trusted watchdog timer must implement a flag that indicates the occurrence of a timeout event that causes a Warm reset, to allow post-reset software to distinguish this from a powerup cold boot.	32
4.6.9 R120_TBSA_TIME The clock source driving a Trusted watchdog timer must be a Trusted clock source	32
4.6.10 R130_TBSA_TIME A TRTC must be configured only by a Trusted world access	32
4.6.11 R150_TBSA_TIME On initial power up and following any other outage of power to the TRTC, a validity mechanism must indicate that the TRTC is not Trusted.....	33
4.6.12 R160_TBSA_TIME: The TRTC must be driven by a Trusted clock source	33
4.7. Version Counter scenarios.....	33
4.7.1 R010_TBSA_COUNT An on-chip non-volatile Trusted firmware version counter implementation must provide a counter range of at least 0 to 63.....	33
4.7.2 R020_TBSA_COUNT An on-chip non-volatile Non-Trusted firmware version counter implementation must provide a counter range of at least 0 to 255	34
4.7.3 R030_TBSA_COUNT It must only be possible to increment a version counter through a Trusted access.....	34
4.7.4 R040_TBSA_COUNT It must only be possible to increment a version counter; it must not be possible to decrement it	34
4.7.5 R050_TBSA_COUNT When a version counter reaches its maximum value, it must not roll over, and no further changes must be possible	34
4.7.6 R060_TBSA_COUNT A version counter must be non-volatile, and the stored value must survive a power down period up to the lifetime of the device	34
4.8. Debug scenarios.....	35
4.8.1 R010_TBSA_DEBUG All debug functionality must be protected by a DPM so that only an authorized external entity can access the debug functionality. There might be scenarios where all external entities can access the debug functionality.....	35
4.8.2 R020_TBSA_DEBUG A DPM must be implemented either solely in hardware or together with software running in the Trusted world	35
4.8.3 R030_TBSA_DEBUG There must be a DPM to permit access to all assets (Trusted)	35
4.8.4 R040_TBSA_DEBUG There must be a DPM to permit access to all Non-Trusted world assets. This mechanism must not permit access to Trusted world assets.....	35

4.8.5 R050_TBSA_DEBUG All DPMs must implement the following fuse-controlled states: Closed - Only an unlock operation is permitted (to transition to Open). This is determined by a Boolean value (dpm_enable) that is stored in a Public-Open-Bitwise fuse or derived from the Device Lifecycle state stored in fuses.	36
4.8.6 R090_TBSA_DEBUG The DPM controlling Trusted world functionality must also have another fuse controlled state: Locked - The unlock operation is disabled (no state transition is possible). This is determined by a Boolean value (dpm_lock) that is stored in a Public-Open-Bitwise fuse or derived from the Device Lifecycle state stored in fuses.	36
4.8.7 R120_TBSA_DEBUG All DPMs must have the following state: Open - Debug is permitted. The Open state can only be entered from the Closed state after a successful unlock operation.	37
4.8.8 R150_TBSA_DEBUG The Trusted world DPM must be enabled, using the respective dpm_enable fuses, or locked, using the respective dpm_lock fuses, before any Trusted world assets are provisioned to the system.	37
4.8.9 R200_TBSA_DEBUG A password unlock token must be at least 128bits in length.	37
4.8.10 R210_TBSA_DEBUG Each debug protection mechanism must use a unique password unlock token.	37
4.8.11 R220_TBSA_DEBUG The unique ID must be included in a certificate unlock token.	38
4.8.12 R230_TBSA_DEBUG An unlock operation using a certificate unlock token must use an approved asymmetric algorithm to check the certificate signature.....	38
4.8.13 R240_TBSA_DEBUG An unlock operation using a certificate unlock token must have access to an asymmetric public key stored on the device. The asymmetric public key that is used to authenticate the certificate unlock token must be immutably stored on the device or have been loaded as a certificate during secure boot and authenticated by a chain of certificates that begins with the ROTPK.....	38
4.8.14 R250_TBSA_DEBUG A certificate unlock token must indicate which DPM(s) it is able to unlock using an authenticated field	39
4.8.15 R260_TBSA_DEBUG A loadable public key for certificate unlock token authentication must include an authenticated field indicating which DPM(s) it is authorized to unlock	39
4.8.16 R270_TBSA_DEBUG A certificate unlock token must only unlock a DPM that its public key is authorized to unlock.....	39
4.8.17 R280_TBSA_DEBUG The device must implement registers, that, when written to by software, unlock the associated hardware debug features. Access to the secure DPM registers must be restricted to privileged Secure world software	39
4.9. External Interface Peripheral scenarios	39
4.9.1 R040_TBSA_EIP Any sensitive user data that is stored must be stored in Secure storage.	39

1 About this document

This document describes the test scenarios for Trusted Base System Architecture for Armv8-M.

1.1. References

Reference	Document	Author	Title
1	-	Arm	Trusted Base System Architecture for Armv8-M Specification

1.2. Terms and Abbreviations

This document uses the following terms and abbreviations.

Term	Meaning
AES	Advanced Encryption Standard
DPM	Debug Protection Mechanism
I2C	Inter-Integrated Circuit
IDAU	Implementation Defined Attribution Unit
MPC	Memory Protection Controller
MPU	Memory Protection Unit
NSC	Non-Secure Callable
NVIC	Nested Vector Interrupt Controller
NVM	Non-Volatile Memory
OTP	One-time Programmable
PAL	Platform Abstraction Layer
PE	Processing Element
PPC	Peripheral Protection Controller
SAU	Security Attribution Unit
SPI	Serial Peripheral Interface
TBSA	Trusted Base System Architecture
VAL	Validation Abstraction Layer

1.3. Scope

This document describes the verification scenarios and the relationship between verification scenarios, tests, and the architecture rules.

2 Introduction

The TBSA-v8M test suite verifies the features of the TBSA-v8M architecture as described in the TBSA-v8M Specification. The following are the features that are within and outside the scope of the TBSA-v8M test suite.

Features tested by TBSA-v8M test suite

The TBSA-v8M test suite verifies the scenarios that can be covered by a system level software.

Features outside the scope of the TBSA-v8M test suite

- Exhaustive testing of the complete SoC implementing TBSA-v8M architecture.
- Architecturally non-deterministic scenarios such as timing-sensitive scenarios.
- Hardware production requirements such as Entropy.

2.1. Limitations of the TBSA-v8M test suite

The following are the limitations of the TBSA-v8M test suite:

- Unless described in this document, any behavior that is defined as IMPLEMENTATION DEFINED in TBSA-v8M specification is not verified in this suite.
- For each verification scenario described in this document, unless specified, only a sample set of possible variants are verified.
- The following rules from the TBSA-v8M specification do not have a specific scenario or a test since they cannot be tested at system software level and therefore, waived by architects.

Rule Number	Rule description
R040_TBSA_BASE	The hardware and software of a TBSA-v8M device must work together to ensure that all the security requirements are met.
R090_TBSA_INFRA	All transactions must be constrained; it must not be possible for a transaction to bypass a constraining mechanism.
R240_TBSA_INFRA	A Trusted operation that exposes a Non-secure interface must apply a policy check to the Non-trusted commands and data before acting on them. The policy check must be atomic and following the check, it must not be possible to modify the checked commands or data
R010_TBSA_FUSE	A non-volatile storage technology must meet the lifetime requirements of the device, either through its intrinsic characteristics, or using error correction mechanisms.
R030_TBSA_FUSE	A fuse must only be programmed in accordance with its specified mechanism so that its reliable operation is not at risk.
R050_TBSA_FUSE	All fuse values must be stable before any parts of the SoC that depend on them are released from reset.
R060_TBSA_FUSE	Fuses that configure the security features of the device must be configured so that the programmed state of the fuse enables the feature. That is, the programming of a security configuration fuse always increases security within the SoC.
R070_TBSA_FUSE	Lifetime guarantee mechanisms to correct for in-field failures must not indicate which fuses have had errors detected or corrected; they just indicate that an error has been detected or corrected. This indicator must only be available after all fuses have been checked.

Rule Number	Rule description
R130_TBSA_FUSE	The locking mechanism for a lockable fuse can be shared with other lockable fuses depending on the functional requirements.
R150_TBSA_FUSE	Additional fuses that implement lifetime guarantee mechanisms must have the same confidential and write lock characteristics as the logical fuse itself.
R035_TBSA_KEY	A key must be used only by the cryptographic scheme for which it was created.
R080_TBSA_KEY	To prevent the re-derivation of previously used keys only Trusted code can have access to all of the Source Material
R090_TBSA_KEY	If an ephemeral key is stored in memory or in a register in clear text form, the storage location must be scrubbed before being used for another purpose.
R100_TBSA_KEY	A key that is accessible to or generated by the Non-Trusted world must only be used for Non-Trusted world cryptographic operations that are either implemented in Non-Trusted world software or have both clear text and cipher text in the Non-Trusted world.
R110_TBSA_KEY	A key that is accessible to or generated by the Trusted world can be used for operations in both Non-Trusted and Trusted worlds, and also across worlds, provided that the Non-Trusted world cannot access the key directly. The Trusted world can control the use of the key through a policy.
R150_TBSA_KEY	The Trusted world must be able to enforce a usage policy for any Trusted hardware key which can be used for Non-Trusted world cryptographic operations.
R230_TBSA_KEY	The HUK must have at least 128 bits of entropy.
R090_TBSA_TIME	Before needing a refresh, a Trusted watchdog timer must be capable of running for a time period that is long enough for the Non-Trusted re-flashing of early boot loader code.
R140_TBSA_TIME	All components of a TRTC must be implemented within the same power domain.
R010_TBSA_ENTROPY	The entropy source must be an integrated hardware block.
R020_TBSA_ENTROPY	The TRNG must produce samples of known entropy.
R030_TBSA_ENTROPY	The TRNG must pass the NIST 800-22 test suite.
R040_TBSA_ENTROPY	On production parts, it must not be possible to monitor the analog entropy source using an external pin.
R290_TBSA_DEBUG	The DPM_TP and DPM_NTP must be implemented solely in hardware or together with firmware in immutable boot ROM.
R010_TBSA_EIP	If an EIP is used to send or receive clear or unauthenticated Trusted world assets, it is implementing a Trusted operation and must meet the requirements of a Trusted peripheral.
R020_TBSA_EIP	When an EIP can receive commands from an external device, for example PCIe, then the system must enforce a policy to check that those commands do not breach the security of the TBSA-M device.
R050_TBSA_EIP	When a sensor has modes that allow it to be used for the acquisition of assets in both the Trusted world and the Non-Trusted world, activating features for Trusted world sensing must be under the control of the Trusted world.

3 Correlation between architecture and tests

The following table lists the correlation between the verification scenarios mentioned in this document and the features and rules of TBSA-v8M architecture.

Test name	Rule number	Rule description
Base system requirements		
test_b002	R001_TBSA_BASE	The SoC must provide a hardware-based mechanism for separating the Trusted World from the Non-Trusted world.
test_b002	R002_TBSA_BASE	The SoC must provide a hardware-based mechanism which is able to separate partitions within the Trusted world.
test_b001	R010_TBSA_BASE	A Non-Trusted world operation must only access Non-Trusted world assets.
test_b001	R020_TBSA_BASE	A Trusted world operation can access both Trusted and Non-Trusted world assets.
test_b002	R030_TBSA_BASE	The SoC must be based on an Armv8-M architecture PE with the Security Extension and MPU implemented.
Infrastructure requirements		
test_b001	R010_TBSA_INFRA	A Trusted operation can issue Secure or Non-secure transactions.
test_b001	R020_TBSA_INFRA	A Non-Trusted operation must only issue Non-secure transactions.
test_b001	R030_TBSA_INFRA	A Non-secure transaction must only access Non-secure storage.
test_b006	R040_TBSA_INFRA	If programmable address remapping logic is implemented in the interconnect, then its configuration must be possible only from the Trusted world.
test_b007	R050_TBSA_INFRA	A unified address map that uses target side filtering to disambiguate Non-secure and Secure transactions must only permit all Secure or all Non-secure transactions to any one region. Secure and Non-secure aliased accesses to the same address region is not permitted.
test_b005	R060_TBSA_INFRA	The target transaction filters configuration space must be accessed only from the Trusted world.
test_i004	R070_TBSA_INFRA	Security exception interrupts must be wired or configured as Secure interrupt sources.
test_b005	R080_TBSA_INFRA	Configuration of the on-chip interconnect that modifies routing or the memory map must only be possible from the Trusted world unless it is not possible for such modifications to affect Secure transactions.
test_b003	R100_TBSA_INFRA	If shared volatile storage is implemented, then the associated location or region must be scrubbed before it can be reallocated from Trusted to Non-Trusted.
test_b004	R110_TBSA_INFRA	If shared volatile storage is implemented, then the associated location must not be executable or NSC immediately after it is reallocated from Non-Trusted to Trusted.
test_b008	R120_TBSA_INFRA	An interrupt originating from a Trusted operation must be mapped only to a Trusted target. By default, this must be the case following a system reset.

Test name	Rule number	Rule description
test_i001	R130_TBSA_INFRA	Any configuration to mask or route a Trusted interrupt must be carried out only from the Trusted world.
test_i002	R140_TBSA_INFRA	The interrupt network might be configured to route an interrupt originating from a Trusted operation to a Non-Trusted target.
test_i003	R150_TBSA_INFRA	Any status flags recording Trusted interrupt events must be read only from the Trusted world, unless specifically configured by the Trusted world to be readable by the Non-Trusted world.
test_m001	R160_TBSA_INFRA	A TBSA-v8M system must integrate a Secure RAM.
test_b001	R170_TBSA_INFRA	Secure RAM must be mapped into the Trusted world only.
test_m001	R180_TBSA_INFRA	If the mapping of Secure RAM into regions is programmable, then configuration of the regions must be possible only from the Trusted world.
test_p001	R190_TBSA_INFRA	The advanced power mechanism must integrate a Trusted management function to control clocks and power. It must not be possible to directly access clock and power functionality from the Non-Trusted world.
test_b005	R210_TBSA_INFRA	If access to a peripheral or a subset of its operations can be dynamically switched between Trusted world and Non-Trusted world, then this must be done only under the control of the Trusted world.
test_b001	R220_TBSA_INFRA	If the peripheral stores assets in local embedded storage, a Non-Trusted operation must not be able to access the local assets of a Trusted operation.
test_b004	R230_TBSA_INFRA	A Trusted operation must be able to distinguish the originating world of commands and data arriving at its interface, by using the address
Fuse requirements		
test_c005	R020_TBSA_FUSE	A fuse is permitted to transition in one direction only - from its unprogrammed state to its programmed state. The reverse operation must be prevented.
test_c010	R040_TBSA_FUSE	It must be possible to blow at least a subset of the fuses when the device has left the silicon manufacturing facility.
test_c007	R080_TBSA_FUSE	A confidential fuse whose recipient is a hardware IP must not be readable by any software process.
test_c007	R090_TBSA_FUSE	A confidential fuse whose recipient is a hardware IP must be connected to the IP using a path that is not visible to software or any other hardware IP.
test_c011	R100_TBSA_FUSE	A confidential fuse whose recipient is a software process might be readable by that process and must be readable by privileged software.
test_b005	R110_TBSA_FUSE	A confidential fuse whose recipient is a Trusted world software process must be protected by a hardware filtering mechanism that can only be configured by secure software, for example an NS-bit filter.
test_c009	R120_TBSA_FUSE	It must be possible to fix a lockable fuse in its current state, regardless of whether it is programmed or unprogrammed.
test_c009	R140_TBSA_FUSE	A bulk fuse must also be a lockable fuse to ensure that any unprogrammed bits cannot be programmed later.
Key requirements		

Test name	Rule number	Rule description
test_c001	R010_TBSA_KEY	A key must be treated as an atomic unit. It must not be possible to use a key in a cryptographic operation before it has been fully created, either during an update operation or during its destruction.
test_c001	R020_TBSA_KEY	Any operations on a key must be atomic. It must not be possible to interrupt the creation, update, or destruction of a key.
test_c008	R030_TBSA_KEY	When a key is no longer required by the system, it must be put beyond use to prevent a hack later from revealing it.
test_c004	R070_TBSA_KEY	A static key must be stored in an immutable structure, for example, a ROM or a set of bulk-lockable fuses.
test_c006	R140_TBSA_KEY	A Trusted hardware key must not be directly accessible by any software.
test_c002	R160_TBSA_KEY	A TBSA-v8M device must either entirely embed a <i>Root of Trust Public Key</i> (ROTPK), or the information that is needed to securely identify it.
test_c002	R180_TBSA_KEY	An elliptic-curve-based ROTPK must be at least 256 bits in size.
test_c002	R190_TBSA_KEY	An RSA-based ROTPK must be at least 3072 bits in size.
test_c002	R200_TBSA_KEY	If a cryptographic hash of the ROTPK is stored in on chip non-volatile memory, rather than the key itself, it must be immutable.
test_c003	R220_TBSA_KEY	A TBSA-v8M device must embed a <i>Hardware Unique root Key</i> (HUK) in Confidential-Lockable-Bulk fuses.
test_c003	R240_TBSA_KEY	The HUK must only be accessible by Trusted code or Trusted hardware that acts on behalf of Trusted code.
Boot requirements		
test_s001	R010_TBSA_BOOT	A TBSA-v8M device must embed a Boot ROM with the initial code that is needed to perform a Trusted system boot.
test_s001	R020_TBSA_BOOT	If the device supports warm boot, a flag or register that survives warm boot must exist to enable distinguishing between warm and cold boots. This register or flag must be programmable only by the Trusted world and must be reset after a cold boot.
test_s001	R030_TBSA_BOOT	On a cold boot, the primary processor must boot from the Boot ROM. It must not be possible to boot from any other storage unless Trusted Kernel debug is enabled. For detailed information about Trusted Kernel debug, see section 6.10.
test_s001	R090_TBSA_BOOT	If a boot status register is implemented, then it must be accessible only by the Trusted world
test_c006	R100_TBSA_BOOT	In an assisted architecture, the key to decrypt the Trusted Boot Firmware image must be visible only to the acceleration peripheral.
Timer requirements		
test_t001	R030_TBSA_TIME	At least one Trusted timer must exist.
test_t001	R040_TBSA_TIME	A Trusted timer must only be modified by a Trusted access. Examples of modifications are the timer being refreshed, suspended, or reset.
test_t001	R050_TBSA_TIME	The clock source that drives a Trusted timer must be a Trusted clock source.

Test name	Rule number	Rule description
test_t002	R060_TBSA_TIME	At least one Trusted watchdog timer must exist.
test_t002	R070_TBSA_TIME	After a system reset, a Trusted watchdog timer must be started before execution of the immutable boot code transfers control to the next firmware stage.
test_t002	R080_TBSA_TIME	A Trusted watchdog timer must only be modified by a Trusted access. Examples of modifications are the timer being refreshed, suspended, or reset.
test_t002	R100_TBSA_TIME	A Trusted watchdog timer must be able to trigger a reset of the SoC, after a predefined period. This value can be fixed in hardware or programmed by a Trusted access.
test_t002	R110_TBSA_TIME	A Trusted watchdog timer must implement a flag that indicates the occurrence of a timeout event that causes a Warm reset, to allow post-reset software to distinguish this from a powerup cold boot.
test_t002	R120_TBSA_TIME	The clock source driving a Trusted watchdog timer must be a Trusted clock source.
test_t003	R130_TBSA_TIME	A TRTC must be configured only by a Trusted world access.
test_t003	R150_TBSA_TIME	On initial power-up and following any other outage of power to the TRTC, a validity mechanism must indicate that the TRTC is not Trusted.
test_t003	R160_TBSA_TIME	The TRTC must be driven by a Trusted clock source.
Version counter requirements		
test_v001	R010_TBSA_COUNT	An on-chip non-volatile Trusted firmware version counter implementation must provide a counter range of at least 0 to 63.
test_v001	R020_TBSA_COUNT	An on-chip non-volatile Non-Trusted firmware version counter implementation must provide a counter range of at least 0 to 255.
test_v001	R030_TBSA_COUNT	It must only be possible to increment a version counter through a Trusted access.
test_v001	R040_TBSA_COUNT	It must only be possible to increment a version counter; it must not be possible to decrement it.
test_v001	R050_TBSA_COUNT	When a version counter reaches its maximum value, it must not roll over and no further changes must be possible.
test_v001	R060_TBSA_COUNT	A version counter must be non-volatile, and the stored value must survive a power down period up to the lifetime of the device.
Entropy source requirements		
Debug requirements		
test_d001	R010_TBSA_DEBUG	All debug functionality must be protected by a DPM so that only an authorized external entity can access the debug functionality. There might be scenarios where all external entities can access the debug functionality.
test_d008	R020_TBSA_DEBUG	A DPM must be implemented either solely in hardware or together with software running in the Trusted World.
test_d001	R030_TBSA_DEBUG	There must be a DPM to permit access to all assets (Trusted).

Test name	Rule number	Rule description
test_d002	R040_TBSA_DEBUG	There must be a DPM to permit access to all Non-Trusted world assets. This mechanism must not permit access to Trusted world assets.
test_d003 test_d005	R050_TBSA_DEBUG	All DPMs must implement the following fuse-controlled states: Closed - Only an unlock operation is permitted (to transition to Open). This is determined by a Boolean value (dpm_enable) that is stored in a Public-Open-Bitwise fuse or derived from the Device Lifecycle state stored in fuses.
test_d004 test_d005	R090_TBSA_DEBUG	The DPM controlling Trusted world functionality must also have another fuse controlled state: Locked - The unlock operation is disabled (no state transition is possible). This is determined by a Boolean value (dpm_lock) that is stored in a Public-Open-Bitwise fuse or derived from the Device Lifecycle state stored in fuses.
test_d001	R120_TBSA_DEBUG	All DPMs must have the following state: Open - Debug is permitted. The Open state can only be entered from the Closed state after a successful unlock operation.
test_d005	R150_TBSA_DEBUG	The Trusted world DPM must be enabled using the respective dpm_enable fuses, or locked, using the respective dpm_lock fuses before any Trusted world assets are provisioned to the system.
test_d006	R200_TBSA_DEBUG	A password unlock token must be at least 128bits in length.
test_d006	R210_TBSA_DEBUG	Each debug protection mechanism must use a unique password unlock token.
test_d007	R220_TBSA_DEBUG	The unique ID must be included in a certificate unlock token.
test_d007	R230_TBSA_DEBUG	An unlock operation using a certificate unlock token must use an approved asymmetric algorithm to check the certificate signature.
test_d007	R240_TBSA_DEBUG	An unlock operation using a certificate unlock token must have access to an asymmetric public key stored on the device. The asymmetric public key that is used to authenticate the certificate unlock token must be immutably stored on the device or have been loaded as a certificate during secure boot and authenticated by a chain of certificates that begins with the ROTPK.
test_d007	R250_TBSA_DEBUG	A certificate unlock token must indicate which DPM(s) it is able to unlock using an authenticated field.
test_d007	R260_TBSA_DEBUG	A loadable public key for certificate unlock token authentication must include an authenticated field indicating which DPM(s) it is authorized to unlock
test_d007	R270_TBSA_DEBUG	A certificate unlock token must only unlock a DPM that its public key is authorized to unlock.
test_d008	R280_TBSA_DEBUG	The device must implement registers, that, when written to by software, unlock the associated hardware debug features. Access to the secure DPM registers must be restricted to privileged Secure world software
External interface peripherals requirements		
test_b001	R040_TBSA_EIP	Any sensitive user data that is stored must be stored in Secure storage.

4 Verification scenarios

This section describes the verification scenarios associated with Trusted Base System Architecture of Armv8-M.

4.1. Base system scenarios

4.1.1 R010_TBSA_BASE: A Non-Trusted world operation must only access Non-Trusted world assets

Check that any Non-Trusted world access can only access Non-Trusted world asset (like memory, peripherals). If a Non-Trusted world operation accesses the trusted world asset (like memory, peripherals), it will result in Secure fault or Hard fault.

#test_b001:

Secure.c

- Install Fault handler.
- Get the memory and peripheral details from targetConfig.cfg.
- Perform a read and write operation to confirm the trusted accesses can access the trusted and non-trusted assets.

Non_Secure.c

- Get the memory and peripheral details from targetConfig.cfg.
- Perform a read and write operation to confirm the non-trusted accesses to trusted asset will result in fault.

4.1.2 R020_TBSA_BASE: A Trusted world operation can access both Trusted and Non-Trusted world assets

Check that Trusted world access can access both Trusted and Non-Trusted world asset (like memory, peripherals). Check that no spurious fault occurs when a Trusted world operation accesses the Non-Trusted world.

#test_b001:

Secure.c

- Install Fault handler.
- Get the memory and peripheral details from targetConfig.cfg.
- Perform a read and write operation to confirm the trusted accesses can access the trusted and non-trusted assets.

Non_Secure.c

- Get the memory and peripheral details from targetConfig.cfg.
- Perform a read and write operation to confirm if the Non-trusted accesses to trusted asset will result in fault.

4.1.3 R030_TBSA_BASE: The SoC must be based on an Armv8-M architecture PE with the Security Extension and MPU implemented

Check that CPUID, ID_PFR0, and MPU_TYPE indicate that the primary processor in the SoC is based on Armv8-M architecture with Security extension and MPU implemented.

#test_b002:

Secure.c

- Read CPUID architecture register as defined in ARMv8M and extract the information of Mainline or Baseline target implementation.
- Read ID_PFR architecture register as defined in ARMv8M to check whether the system implements security extensions.
- Read MPU_TYPE architecture register from both security states to confirm that both secure and non-secure MPU's are implemented in the system.

4.2. Infrastructure scenarios

4.2.1 R010_TBSA_INFRA: A Trusted operation can issue Secure or Non-secure transactions

Check the scenario as defined in rule R010_TBSA_BASE.

#test_b001:

Refer to the algorithm defined in rule R010_TBSA_BASE

4.2.2 R020_TBSA_INFRA: A Non-Trusted operation must only issue Non-secure transactions

Check the scenario as defined in rule R020_TBSA_BASE.

#test_b001:

Refer algorithm as defined in rule R010_TBSA_BASE.

4.2.3 R030_TBSA_INFRA: A Non-secure Transaction must only access Non-secure storage

Check that Non-secure transaction (read/write) accesses the Non-secure storage (like memory).

#test_b001:

Refer algorithm as defined in rule R010_TBSA_BASE.

4.2.4 R040_TBSA_INFRA If programmable address remapping logic is implemented in the interconnect then its configuration must only be possible from the Trusted world

Check that programming of remapping logic in the interconnect can be performed only from a Trusted world. A single set of meta data register file (from targetConfig.cfg input file) can be used to check that these bunch of registers are indeed programmed from

Trusted world. A single test algorithm can be used to cover all the registers in this space which can be programmed from the Trusted world.

#test_b006:

Secure.c

- Install Fault handler.

Non-secure.c

- Disable all types of fault in SHCSR architecture register such that only Hardfault is taken.
- Get the interconnect remap register details from targetConfig.cfg.
- Perform a read and write operation to confirm the Non-trusted accesses is not allowed and this operation shall result in fault.

4.2.5 R050_TBSA_INFRA A unified address map that uses target side filtering to disambiguate Non-secure and Secure transactions must only permit all Secure or all Non-secure transactions to any one region. Secure and Non-secure aliased accesses to the same address region are not permitted.

Check that access to two distinct address from Trusted and Non-trusted world doesn't land in one physical address.

#test_b007:

Secure.c

- Get detail of free secure block from target configuration file.
- Write a known pattern to the free block found.

Non-secure.c

- Check for the same pattern in all the available NS block of memory, if found fail the test otherwise pass the test

4.2.6 R060_TBSA_INFRA The target transaction filters configuration space must only be accessed from the Trusted world

Check that if any one of the target transaction filters like MPC or PPC is implemented, then it should be accessible only from the Trusted world.

#test_b005:

Secure.c

- Install Fault handler.

Non-secure.c

- Disable all types of fault in SHCSR architecture register such that only Hardfault is taken.
- Get the device base address of PPC and MPCs implemented in the System via targetConfig.cfg.

- Perform a read and write operation to this device base address to confirm the non-trusted accesses is not allowed and this operation shall result in fault.

4.2.7 R070_TBSA_INFRA Security Exception Interrupts must be wired or configured as Secure interrupt sources

Check that a trusted interrupt should be mapped only to Secure and it should not be possible to assert a Non-secure exception or interrupt even if when NVIC_ITNS is programmed.

#test_i004

Secure.c

- Get the interrupt source number and security attribute from targetConfig.cfg
- Install trusted interrupt handler
- Configure a pend bit for the given interrupt source number
- Check that the interrupt routing is appropriate.

Non-secure.c

- Install non-trusted interrupt handler
- Configure a pend bit for the given interrupt source number
- Check that the interrupt routing is appropriate.

4.2.8 R080_TBSA_INFRA Configuration of the on-chip interconnect that modifies routing or the memory map must only be possible from the Trusted world unless it is not possible for such modifications to affect secure transactions

If there is an on-chip interconnect configuration that modifies the routing, possibly like MPC or PPC, then check that the configuration of those are possible only from a Trusted world.

#test_b005

Secure.c

- Install Fault handler.

Non-secure.c

- Disable all types of fault in SHCSR architecture register such that only Hardfault is taken.
- Get the device base address of PPC and MPCs implemented in the System via targetConfig.cfg.
- Perform a read and write operation to this device base address to confirm the non-trusted accesses is not allowed and this operation shall result in fault.

4.2.9 R100_TBSA_INFRA If shared volatile storage is implemented, then the associated location or region must be scrubbed before it can be reallocated from Trusted to Non-Trusted

Check that when a memory is configured from a Secure to Non-secure, the memory location is scrubbed before it is given back to Non-Trusted memory. Hence if there is a write access in Trusted world written with value1. After configuring this memory location to Non-trusted, the Non-trusted memory should never read the 'value1' which was written by Trusted world.

#test_b003

Secure.c

- Get a memory block whose memory attribute can be configurable to either secure or non-secure from targetConfig.cfg.
- Configure the memory attribute as secure via MPC and then write a known pattern to these memory blocks.
- Reconfigure this memory block to non-secure via MPC.

Non-secure.c

- Read the memory block which was configured as Non-secure via MPC in 'secure.c'.
- Check that the memory block is scrubbed and does not contain the pattern written in 'secure.c'.

4.2.10 R110_TBSA_INFRA If shared volatile storage is implemented, then the associated location must not be executable or NSC immediately after it is reallocated from Non-Trusted to Trusted

Check that a when the shared memory is configured from Non-secure to Secure, then a function call executed from the remapped location should cause a fault since it is expected to not be executed.

#test_b004

Secure.c

- Install fault handler.

Non-secure.c

- Get a memory block (say A) which is configurable from targetConfig.cfg and ensure it is marked as non-secure.
- Copy a piece of function (say B) into this non-secure memory which will have valid result only when executed from trusted world.
- Configure memory block A to secure
- If the function B is attempted to execute, then a fault is expected.

4.2.11 R120_TBSA_INFRA An interrupt originating from a Trusted operation must be mapped only to a Trusted target. By default, this must be the case following a system reset

Check that interrupt originating from Trusted world operations (like Trusted Timer configuration and Watchdog) are being serviced by Trusted handlers configured initially.

#test_t001

Refer test algorithm as defined in rule R030_TBSA_TIME

4.2.12 R130_TBSA_INFRA Any configuration to mask or route a Trusted interrupt must only be carried out from the Trusted world

Check that an interrupt (if configurable to either Secure or Non-secure), then check that routing of this interrupt through NVIC_ITNS happens from Trusted world.

Check that an interrupt can be masked (disabling the interrupt line) only from a Trusted world by checking the pend bit status across both Secure and Non-secure worlds.

#test_i001

Secure.c

- Dummy functions for entry, exit and test_payload.

Non-secure.c

- Get a timer instance that is secure programmable from targetConfig.cfg
- Set the pending bit for the trusted timer (via secure functions).
- Route the trusted timer interrupt to non-trusted mode and check that pending bit is set in the non-trusted mode.
- Check that the trusted timer interrupt can be masked only from the trusted world (via secure functions).
- Clear pending bits for the trusted timer.

4.2.13 R140_TBSA_INFRA The interrupt network might be configured to route an interrupt originating from a Trusted operation to a Non-Trusted target

Check the scenario as defined in rule R130_TBSA_INFRA.

#test_i002

Secure.c

- Dummy functions for entry, exit and test_payload.

Non-secure.c

- Get a timer instance that is secure programmable from targetConfig.cfg.
- Set the pending bit for the trusted timer (via secure functions).
- Route the trusted timer interrupt to non-trusted mode and check that the non-trusted interrupt handler is serviced.
- Clear pending bits for the trusted timer.

4.2.14 R150_TBSA_INFRA Any status flags recording Trusted interrupt events must only be read from the Trusted world, unless specifically configured by the Trusted world, to be readable by the Non-Trusted world

Check that Trusted interrupt pending status flag can be read in Non-trusted or Non-secure world as well when the trusted interrupt is routed to Non-trusted target.

#test_i003

Secure.c

- Dummy functions for entry, exit and test_payload.

Non-secure.c

- Get a timer instance that is secure programmable from targetConfig.cfg.
- Set the pending bit for the trusted timer (via secure functions).
- Read and check the pending bit in both secure and non-secure state and confirm that the trusted timer pend status bit can be read from the trusted world.

4.2.15 R160_TBSA_INFRA A TBSA-v8M system must integrate a Secure RAM

#test_m001

Refer test algorithm as defined in rule R180_TBSA_INFRA

4.2.16 R170_TBSA_INFRA Secure RAM must be mapped into the Trusted world only

#test_b001

Refer test algorithm as defined in rule R010_TBSA_INFRA

4.2.17 R180_TBSA_INFRA If the mapping of Secure RAM into regions is programmable, then configuration of the regions must only be possible from the Trusted world

Check that if an implementation allows a memory region to be configured through protection controllers like MPC, then check that configuration of MPC can occur only from the Trusted world.

#test_m001

Secure.c

- Install fault handler.
- Get the memory instances of SRAM and Flash from targetConfig.cfg.
- If a memory block is marked as configurable, then program MPC such that the memory block is configured as Non-secure. Perform read and write access and confirm the value.
- Reprogram the memory block to original security state (Secure).

Non-secure.c

- Try to access the MPC configuration registers from the Non-trusted world. Check that a secure fault is triggered.
- Perform memory read and write accesses to confirm the access permissions via MPC.

4.2.18 R190_TBSA_INFRA The advanced power mechanism must integrate a Trusted management function to control clocks and power. It must not be possible to directly access clock and power functionality from the Non-trusted world

Check that clock and power domain controls can be accessed only from a Trusted world.

#test_p001

Secure.c

- Install fault handler.

Non-secure.c

- Get a clock and power controller device base address from targetConfig.cfg.
- Check that accessing these device registers shall result in fault.

4.2.19 R210_TBSA_INFRA If access to a peripheral or a subset of its operations can be dynamically switched between Trusted world and Non-trusted world, then this must be done only under the control of the Trusted world

Check that when a peripheral can be configured as either Trusted or Non-trusted entity, this configuration should be performed from Trusted world. If a Non-trusted world tries to configure the peripheral, then the access should result in Secure fault.

#test_b005 – Covered as a part of rule R060_TBSA_INFRA

4.2.20 R220_TBSA_INFRA If the peripheral stores assets in local embedded storage, a Non-trusted operation must not be able to access the local assets of a Trusted operation

Check that the local storages like FIFOs, buffers of the Trusted world must be accessible by Non-trusted world entity. If the local storage is mapped to memory, then it should be mapped to trusted world.

#test_b001

Refer test algorithms in rule R010_TBSA_BASE.

4.2.21 R230_TBSA_INFRA A Trusted operation must be able to distinguish the originating world of commands and data arriving at its interface, by using the address.

VAL APIs uses TT instruction to check the received address is from Trusted world or Non-trusted world.

#test_b004

Refer test algorithms in rule R110_TBSA_INFRA.

4.3. Fuse scenarios

4.3.1 R020_TBSA_FUSE: A fuse is permitted to transition in one direction only, from its unprogrammed state to its programmed state. The reverse operation must be prevented

First write the value 0xFFFFFFFF in fuse and then write 0xF0F0F0F0 in the same fuse. Check if the value is 0xFFFFFFFF.

#test_c005

Secure.c

- Obtain an empty fuse.
- Write the value 0x0000FFFF in the fuse.
- Make sure that the value is fused.
- Try writing the value 0x0000F0F0.
- Read the value and check that the value is 0x0000FFFF.

Non-secure.c

- Dummy entry, exit and payload functions.

4.3.2 R040_TBSA_FUSE: It must be possible to blow at least a subset of the fuses when the device has left the silicon manufacturing facility

Check if user otp area is present. Write a value and verify it.

#test_c010

Secure.c

- Get the current Life Cycle State of the device.
- For systems that doesn't have LCS, get the state from the target config.
- Check that the life cycle state is in Deployed LCS.
- Get the free fuse from the target config.
- Make sure that the fuse is empty.
- Blow the fuse and make sure that the value is written.

Non-secure.c

- Dummy entry, exit and payload functions.

4.3.3 R080_TBSA_FUSE: A confidential fuse whose recipient is a hardware IP must not be readable by any software process

Check if a confidential fuse is readable only by IP for target config. If yes, then accessing it should not reveal the key.

#test_c007

Secure.c

- Setup the interrupt handlers.
- Get the details of the confidential fuse from the target config.
- Trying to read the fuse address.
- The value should be read as zero or raise a fault.

Non-secure.c

- Dummy entry, exit and payload functions.

4.3.4 R090_TBSA_FUSE: A confidential fuse whose recipient is a hardware IP must be connected to the IP using a path that is not visible to software or any other hardware IP

Check the scenario covered as a part of R080_TBSA_FUSE.

#test_c007

Refer test algorithm as defined in rule R080_TBSA_FUSE

4.3.5 R100_TBSA_FUSE: A confidential fuse whose recipient is a software process might be readable by that process and must be readable by privileged software

Check that a fault is triggered when a read happens from an unprivileged software.

#test_c011

Secure.c

- Setup the interrupt handlers.
- Get the details of the confidential fuse from target config.
- Read the fuse value and make sure that it is not zero.
- Change the mode to un-privilege access.
- Try accessing the fuse again.
- It should either result in a fault or read as zero (RAZ).

Non-secure.c

- Dummy entry, exit and payload functions.

4.3.6 R110_TBSA_FUSE: A confidential fuse whose recipient is a Trusted world software process must be protected by a hardware filtering mechanism that can only be configured by secure software, for example an NS-bit filter

Try reading a confidential fuse from unprivileged software and expect a fault. Change the configuration to allow access from Non-secure and should be able to read. Also check if the configuration register is accessible only from Secure space.

#test_b005 – Covered as a part of rule R060_TBSA_INFRA

4.3.7 R120_TBSA_FUSE: It must be possible to fix a lockable fuse in its current state, regardless of whether it is programmed or unprogrammed

Check that the locked fuse does not get modified.

#test_c009

Secure.c

- Get the already locked fuse from the target config.
- Read the value of the fuse.
- Try to blow the fuse with value 0xFFFFFFFF (try to program all the bits).
- Read the value of the fuse.
- The value of the fuse must not be changed and must be the same as previous.

Non-secure.c

- Dummy entry, exit and payload functions.

4.3.8 R140_TBSA_FUSE: A bulk fuse must also be a lockable fuse to ensure that any unprogrammed bits cannot be programmed later

Refer scenario as defined in the rule R120_TBSA_FUSE.

#test_c009

Refer test algorithm as defined in rule R120_TBSA_FUSE.

4.4. Key scenarios

4.4.1 R010_TBSA_KEY: A key must be treated as an atomic unit. It must not be possible to use a key in a cryptographic operation before it has been fully created, during an update operation, or during its destruction

Program the timer to receive at least 5 interrupts before key generation. Start the key generation. If interrupt is serviced midway during key generation, copy the current values of the key (only partial key will be present) and disable the interrupt. If it is serviced after the key generation, copy the current value of the key (full key value will be present). Compare it against the generated key to see if full or partial key is copied.

#test_c001

Secure.c

- Get the details of the timer from target config
- Initialize the timer and interrupt handlers
- Check that we receive at least 5 exceptions before we generate the key
- If not, reinitialize the timer with lesser time interval
- Start key generation
- If interrupt is handled in the middle of key generation, save the partly generated key
- If interrupt is not handled, save the whole key
- Once key generation is done, check that saved key is equal to the generated key

Non-secure.c

4.4.2 R020_TBSA_KEY: Any operations on a key must be atomic. It must not be possible to interrupt the creation, update, or destruction of a key

Covered in rule R020_TBSA_KEY.

#test_c008

Refer test algorithm as defined in rule R010_TBSA_KEY.

4.4.3 R030_TBSA_KEY: When a key is no longer required by the system, it must be put beyond use to prevent a hack at a later time from revealing it

Make sure that a key is accessible. Then revoke the key. Accessing the key again should be read as zero or raise a fault.

#test_c001

Secure.c

- Get the details of the key to be revoked from the target config.
- Read the key and make sure that it is non-zero.
- Revoke the key.
- Read the key region again and make sure that it not the same as before.

Non-secure.c

- Dummy entry, exit and payload functions

4.4.4 R070_TBSA_KEY: A static key must be stored in an immutable structure, for example a ROM or a set of Bulk-Lockable fuses

Check if the key is in Bulk-lockable fuse. Try to modify the value and expect it to be unchanged.

#test_c004

Secure.c

- Check if the static key is present.
- Check if it is bulk and lockable (based on the input from target config).
- If the key is readable, read the key and store the value.
- Check that the key is not zero.
- Try modifying the key and make sure that it is not modified.

Non-secure.c

- Dummy entry, exit and payload functions.

4.4.5 R140_TBSA_KEY: A Trusted hardware key must not be directly accessible by any software

Accessing a trusted hardware key should be read as zero or raise a fault.

#test_c006

Secure.c

- Get the details of trusted hardware key.
- Make a read access to the key (in the Trusted world).
- Check that the value read is zero.

Non-secure.c

- Dummy entry, exit and payload functions.

4.4.6 R160_TBSA_KEY: A TBSA-v8M device must either entirely embed a root of trust public key (ROTPK), or the information that is needed to securely identify it

Check if the ROTPK is present in the system.

#test_c002

Secure.c

- Get the ROTPK details from the target config.
- If the key is ECC.
 - Check if the size is greater than equal to 256 bits.
 - Validate if the number of zeros in the key is equal to the zero count in the config fuse.
- If the key is RSA
 - Check if the size is greater than equal to 3072 bits.
 - Validate the number of zeros in the key is equal to the zero count in the config fuse.
- If the key is Hash of ROTPK
 - Check that it is immutable

Non-secure.c

- Dummy entry, exit and payload functions.

4.4.7 R180_TBSA_KEY: An elliptic-curve-based ROTPK must be at least 256 bits in size

Check if the key size is at least 256 bits and the number of zeros in the key is equal to the number of zeros in the fuse flag.

#test_c002

Refer test algorithm as given in rule R160_TBSA_KEY.

4.4.8 R190_TBSA_KEY: An RSA-based ROTPK must be at least 3072 bits in size

Check if the key size is at least 3072 bits and the number of zeros in the key is equal to the number of zeros in the fuse flag.

#test_c002

Refer test algorithm as given in rule R160_TBSA_KEY.

4.4.9 R200_TBSA_KEY: If a cryptographic hash of the ROTPK is stored in on chip non-volatile memory, rather than the key itself, it must be immutable

Check if the ROTPK fuse is immutable.

#test_c002

Refer test algorithm as given in rule R160_TBSA_KEY.

4.4.10 R220_TBSA_KEY: A TBSA-v8M device must embed a hardware unique root key (HUK) in Confidential-Lockable-Bulk fuses

Check if the HUK is in Confidential-Lockable-Bulk fuses from the target config.

#test_c003

Secure.c

- Get the details of the HUK from target config
- Check that the fuse type is confidential, bulk and lockable (using details from target config)
- If HUK is readable, then make sure that it is non-zero
- Else skip the test.

Non-secure.c

- Get the details of the HUK from target config
- Check that the fuse type is confidential, bulk and lockable (using details from target config)
- If HUK is readable, then copy the key value in secure mode
- Make a non-secure access to the HUK and make sure that it is not equal to value reading secure mode

4.4.11 R240_TBSA_KEY: The HUK must only be accessible by Trusted code or Trusted hardware that acts on behalf of Trusted code

Check if the HUK is accessible only by Trusted hardware from target config. If yes, then the value should be read as zero. If accessible from Trusted code, then it must raise a fault when accessed from Non-trusted code.

#test_c003

Refer test algorithm as given in rule R220_TBSA_KEY.

4.5. Boot scenarios

4.5.1 R010_TBSA_BOOT A TBSA-M device must embed a Boot ROM with the initial code that is needed to perform a Trusted system boot.

#test_s001

Refer test algorithm as given in rule R030_TBSA_BOOT

4.5.2 R020_TBSA_BOOT If the device supports warm boot, a flag or register that survives warm boot must exist, to enable distinguishing between warm and cold boots. This register or flag must be programmable only by the Trusted world and must be reset after a cold boot

If an implementation allows warm and cold reset, then check that on a warm reset, the platform layer shall read the flag register which indicates the type of reset. On accessing this flag register from Non-trusted world, a fault is expected.

#test_s001 (part a)

Secure.c

- Install fault handler

Non-secure.c

- Assuming that a flag register/memory will be implemented in a trusted world where the reset type will be preserved, and accessing this flag register/memory from a non-trusted world should result in a fault.

4.5.3 R030_TBSA_BOOT On a cold boot, the primary processor must boot from the Boot ROM. It must not be possible to boot from any other storage unless Trusted Kernel debug is enabled

#test_s001 (part b)

Secure.c

- Install fault handler

Non-secure.c

- Get the Boot ROM address range from targetConfig.cfg
- Read VTOR and check that it falls under the boot ROM address range.
- Check whether VTOR is relocated before 'tbsa_entry' configures VTOR
- If re-located throw a warning message else pass the test.

4.5.4 R090_TBSA_BOOT If a boot status register is implemented, then it must be accessible only by the Trusted world

If an implementation allows warm and cold reset, then check that on a warm reset, the platform layer shall read the flag register which indicates the type of reset. On accessing this flag register from Non-trusted world, a fault is expected.

#test_s001

Refer test algorithm as defined in rule R020_TBSA_BOOT

4.5.5 R100_TBSA_BOOT In an Assisted architecture, the key to decrypt the Trusted Boot Firmware image must be visible only to the acceleration peripheral

#test_c006

Refer test algorithm as defined in rule R140_TBSA_KEY

4.6. Timer scenarios

4.6.1 R030_TBSA_TIME At least one Trusted timer must exist

Check that at least one trusted time exists.

#test_t001

Secure.c

- Install fault handler

- Get the base address of trusted timer from targetConfig.cfg.
- Setup the interrupt handler and enable the timer. Check that a trusted interrupt from the trusted timer will be triggered.

Non-secure.c

- Access the base address of trusted timer from non-trusted world and check that the fault is triggered.

4.6.2 R040_TBSA_TIME A Trusted timer must only be modified by a Trusted access. Examples of modifications are the timer being refreshed, suspended, or reset

Check that the Trusted timer can be accessed only from a Trusted world.

#test_t001

Refer test algorithm as defined in rule R030_TBSA_TIME

4.6.3 R050_TBSA_TIME The clock source that drives a Trusted timer must be a Trusted clock source

Check that PLL configuration control registers can only be accessed through Trusted world.

#test_t001

Refer test algorithm as defined in rule R030_TBSA_TIME.

4.6.4 R060_TBSA_TIME At least one Trusted watchdog timer must exist

Check that at least one watchdog timer exists.

#test_t002

Secure.c

- Install fault handler.
- Get the base address of watchdog timer from targetConfig.cfg.
- Setup the interrupt handler and enable the timer. Check that a watchdog reset is asserted.

Non-secure.c

- Access the base address of watchdog timer from non-trusted world and check that the fault is triggered.
- Asserting a watchdog timer shall result in a reset. After the reset, flag should be implemented to be make sure that the software can distinguish between the timeout reset from watchdog vs power on cold boot.
- Access the base address of the clock source from non-trusted world and check that the fault is triggered.

4.6.5 R070_TBSA_TIME After a system reset, a Trusted watchdog timer must be started before the execution of immutable boot code transfers control to the next firmware stage

Check that after a watchdog timer reset, the watchdog is enabled by default.

#test_t002

Refer test algorithm as given in rule R060_TBSA_TIME.

4.6.6 R080_TBSA_TIME A Trusted watchdog timer must only be modified by a Trusted access. Examples of modifications are the timer being refreshed, suspended, or reset

Check that a watchdog timer can only be accessed from a trusted world.

#test_t002

Refer test algorithm as given in rule R060_TBSA_TIME.

4.6.7 R100_TBSA_TIME A Trusted watchdog timer must be able to trigger a reset of the SoC, after a predefined period. This value can be fixed in hardware or programmed by a Trusted access

Check that watchdog reset is asserted after a predetermined value for watchdog.

#test_t002

Refer test algorithm as given in rule R060_TBSA_TIME.

4.6.8 R110_TBSA_TIME A Trusted watchdog timer must implement a flag that indicates the occurrence of a timeout event that causes a Warm reset, to allow post-reset software to distinguish this from a powerup cold boot.

Check that a watchdog timer reset has asserted a warm reset through a flag register (in a platform abstraction layer).

#test_t002

Refer test algorithm as given in rule R060_TBSA_TIME.

4.6.9 R120_TBSA_TIME The clock source driving a Trusted watchdog timer must be a Trusted clock source

Check that PLL configuration control registers can only be accessed through Trusted world.

#test_t002

Refer test algorithm as given in rule R060_TBSA_TIME.

4.6.10 R130_TBSA_TIME A TRTC must be configured only by a Trusted world access

Check that TRTC control registers can be configured only from a Trusted world.

#test_t003

Secure.c

- Install fault handler

Non-secure.c

- Check that at least one trusted TRTC is implemented.
- Ensure that TRTC access from non-trusted world triggers a fault.
- Check that clock source base address of TRTC triggers a fault.
- Check whether TRTC is synchronized to server and the validity mechanism throws out the output as whether it is trusted or non-trusted.

4.6.11 R150_TBSA_TIME On initial power up and following any other outage of power to the TRTC, a validity mechanism must indicate that the TRTC is not Trusted

Refer scenario as defined in rule R130_TBSA_TIME.

#test_t003, refer test algorithm as given in rule R130_TBSA_TIME.

4.6.12 R160_TBSA_TIME: The TRTC must be driven by a Trusted clock source

Like trusted timer, check that PLL in the SoC can be configured only through Trusted world.

#test_t003, refer test algorithm as given in rule R130_TBSA_TIME.

4.7. Version Counter scenarios

4.7.1 R010_TBSA_COUNT An on-chip non-volatile Trusted firmware version counter implementation must provide a counter range of at least 0 to 63

Check that maximum number of trusted firmware version counters implement a counter range between 0 to 63.

#test_v001

Secure.c

- Install fault and reset handler.

Non-secure.c

- Get details of version counter from targetConfig.cfg
- For each instance of version counter, check that it is mapped to trusted or non-trusted firmware.
- Based on the details of the version counter provided in the targetConfig.cfg, check for valid ranges.
- Check that firmware version number can only be incremented
- Check that firmware version counter's maximum range cannot be auto-rolled.
- Check that the update of the firmware version counter is only from trusted mode

4.7.2 R020_TBSA_COUNT An on-chip non-volatile Non-Trusted firmware version counter implementation must provide a counter range of at least 0 to 255

Check that maximum number of Non-trusted firmware version counters implement a counter range between 0 to 255.

#test_v001

Refer test algorithm as defined in rule R010_TBSA_COUNT.

4.7.3 R030_TBSA_COUNT It must only be possible to increment a version counter through a Trusted access

Check that a version counter can be accessed through Trusted world.

#test_v001

Refer test algorithm as defined in rule R010_TBSA_COUNT.

4.7.4 R040_TBSA_COUNT It must only be possible to increment a version counter; it must not be possible to decrement it

Check that version counter (both Trusted and Non-trusted) can only be possible to increment from Trusted world and cannot be decremented.

#test_v001

Refer test algorithm as defined in rule R010_TBSA_COUNT.

4.7.5 R050_TBSA_COUNT When a version counter reaches its maximum value, it must not roll over, and no further changes must be possible

Check that when the version counter reaches maximum value, it should not be possible to get into a new value as there are no new changes are possible.

#test_v001

Refer test algorithm as defined in rule R010_TBSA_COUNT.

4.7.6 R060_TBSA_COUNT A version counter must be non-volatile, and the stored value must survive a power down period up to the lifetime of the device

Check that version counter value is retained even after Power on reset. (Lifetime of the device cannot be verified).

#test_v001

Refer test algorithm as defined in rule R010_TBSA_COUNT.

4.8. Debug scenarios

4.8.1 R010_TBSA_DEBUG All debug functionality must be protected by a DPM so that only an authorized external entity can access the debug functionality. There might be scenarios where all external entities can access the debug functionality.

Check that an external entity can access the debug functionality through *Debug Protection Mechanisms* (DPM).

#test_d001

Secure.c

- Read targetConfig.cfg and check that a DPM is implemented in SoC.
- Check whether debugger is connected by checking the message passing.
- If the DPM is available, then set the state to open via unlock method.
- Check that the accesses under the DPM control have valid and correct.

Non-secure.c

- Dummy functions for entry, exit and payload.

4.8.2 R020_TBSA_DEBUG A DPM must be implemented either solely in hardware or together with software running in the Trusted world

Refer scenario listed in rule R280_TBSA_DEBUG.

4.8.3 R030_TBSA_DEBUG There must be a DPM to permit access to all assets (Trusted)

Check that all assets within the SoC has an associated DPM through which the accesses are controlled through Open and Closed states.

#test_d001

Refer test algorithm as defined in R010_TBSA_DEBUG.

4.8.4 R040_TBSA_DEBUG There must be a DPM to permit access to all Non-Trusted world assets. This mechanism must not permit access to Trusted world assets.

Check that the debug access through DPM allows only Non-trusted asset view while on the background check that a trusted watchdog timer is suspended from counting.

#test_d002

Secure.c

- Dummy functions for entry, exit and payload.

Non-secure.c

- Read targetConfig.cfg and check that a DPM is implemented in SoC.
- Configure the DPM to allow only the access to non-secure state only.
- If the DPM is available, then set the state to open via unlock method.
- Check that the accesses under the DPM control have valid and correct from the non-trusted world.

4.8.5 R050_TBSA_DEBUG All DPMs must implement the following fuse-controlled states: Closed - Only an unlock operation is permitted (to transition to Open). This is determined by a Boolean value (dpm_enable) that is stored in a Public-Open-Bitwise fuse or derived from the Device Lifecycle state stored in fuses.

Check that all DPMs (Trusted and Non-trusted) implemented in the SoC has fuse controlled states as defined as Open/Closed.

#test_d003

Secure.c

- Read targetConfig.cfg and check that a DPM is implemented in SoC.
- If the DPM is available, then set the state to closed. Check for the current state of the DPM.
- Check that the accesses under the DPM control should not be allowed as the DPM is in closed state.

Non-secure.c

- Dummy functions for entry, exit and payload.

4.8.6 R090_TBSA_DEBUG The DPM controlling Trusted world functionality must also have another fuse controlled state: Locked - The unlock operation is disabled (no state transition is possible). This is determined by a Boolean value (dpm_lock) that is stored in a Public-Open-Bitwise fuse or derived from the Device Lifecycle state stored in fuses.

Check that DPM controlling trusted functionality must implement fuse-controlled locked state (state machine check as per specification)

#test_d004

Secure.c

- Read targetConfig.cfg and check that a DPM is implemented in SoC.
- Get the current state of the DPM
 - If the DPM lock is implemented and lock is set, then proceed with access check (if there is only one DPM and it is in locked state then exit the test, as access check is not possible).
 - if DPM lock is implemented but not locked, then set a variable which will be used to set the DPM lock.
 - if DPM lock is not implemented then start the loop again to find next DPM with lock implemented.

Non-secure.c

- Dummy functions for entry, exit and payload.

4.8.7 R120_TBSA_DEBUG All DPMs must have the following state: Open - Debug is permitted. The Open state can only be entered from the Closed state after a successful unlock operation.

Check that all assets within the SoC has an associated DPM through which the accesses are controlled via Open and Closed states.

#test_d001

Refer test algorithm as defined in rule R030_TBSA_DEBUG.

4.8.8 R150_TBSA_DEBUG The Trusted world DPM must be enabled, using the respective dpm_enable fuses, or locked, using the respective dpm_lock fuses, before any Trusted world assets are provisioned to the system.

Check that DPM state machine cycle is checked for all states.

#test_d005

Secure.c

- Read targetConfig.cfg and check that a DPM is implemented in SoC.
- Get the current state of the DPM and check for the various states of the DPM state machine.

Non-secure.c

- Dummy functions for entry, exit and payload.

4.8.9 R200_TBSA_DEBUG A password unlock token must be at least 128bits in length.

Check that password unlock token is less than 128 bits for DPM.

#test_d006

Secure.c

- Read targetConfig.cfg and check that a DPM is implemented in SoC.
- Get the unlock token details from targetConfig.cfg and if the unlock token is password, then check that the DPMs gets unlocked via the password token.

Non-secure.c

- Dummy functions for entry, exit and payload.

4.8.10 R210_TBSA_DEBUG Each debug protection mechanism must use a unique password unlock token.

Check that each DPM has a unique password unlock token to unlock.

#test_d006

Secure.c

- Read targetConfig.cfg and check that a DPM is implemented in SoC.
- Get the unlock token details from targetConfig.cfg and if the unlock token is password, then check that the DPMs gets unlocked via the password token.
- If the number of DPMs implemented is more than one, then check that each password used is unique for each DPM.

Non-secure.c

- Dummy functions for entry, exit and payload.

4.8.11 R220_TBSA_DEBUG The unique ID must be included in a certificate unlock token.

Check that each DPM has a unique ID for the unlock certificate using public key.

#test_d007

Secure.c

- Read targetConfig.cfg and check that a DPM is implemented in SoC.
- Get the unlock token details from targetConfig.cfg and if the unlock token is certificate, then check that the DPMs gets unlocked via the certificate if it is valid. Using the public key base address and certificate base address obtained from the targetConfig.cfg, the unlock operation is performed.
- Also check that an authenticated field for DPM is needed for both public key and certificate and compared.

Non-secure.c

- Dummy functions for entry, exit and payload

4.8.12 R230_TBSA_DEBUG An unlock operation using a certificate unlock token must use an approved asymmetric algorithm to check the certificate signature

#test_d007

Refer test algorithm as defined in rule R220_TBSA_DEBUG.

4.8.13 R240_TBSA_DEBUG An unlock operation using a certificate unlock token must have access to an asymmetric public key stored on the device. The asymmetric public key that is used to authenticate the certificate unlock token must be immutably stored on the device or have been loaded as a certificate during secure boot and authenticated by a chain of certificates that begins with the ROTPK.

#test_d007

Refer test algorithm as defined in rule R220_TBSA_DEBUG.

4.8.14 R250_TBSA_DEBUG A certificate unlock token must indicate which DPM(s) it is able to unlock using an authenticated field

#test_d007

Refer test algorithm as defined in rule R220_TBSA_DEBUG.

4.8.15 R260_TBSA_DEBUG A loadable public key for certificate unlock token authentication must include an authenticated field indicating which DPM(s) it is authorized to unlock

#test_d007

Refer test algorithm as defined in rule R220_TBSA_DEBUG.

4.8.16 R270_TBSA_DEBUG A certificate unlock token must only unlock a DPM that its public key is authorized to unlock

#test_d007

Refer test algorithm as defined in rule R220_TBSA_DEBUG.

4.8.17 R280_TBSA_DEBUG The device must implement registers, that, when written to by software, unlock the associated hardware debug features. Access to the secure DPM registers must be restricted to privileged Secure world software

#test_d008

Secure.c

- Read targetConfig.cfg and check that a DPM is implemented in SoC.
- Install hard fault handler such that reset will be used as mechanism to come out of hard fault.
- Disable all faults such that hard fault is triggered on the occurrence of any error.
- Get the unlock token details from targetConfig.cfg and if the unlock token is certificate, then check that the DPMs gets unlocked via the certificate if it is valid in privileged mode.
- Check that in an unprivileged mode, an error is thrown.

Non-secure.c

- Dummy functions for entry, exit and payload

4.9. External Interface Peripheral scenarios

4.9.1 R040_TBSA_EIP Any sensitive user data that is stored must be stored in Secure storage.

#test_b001

Refer test algorithm as defined in rule R010_TBSA_BASE.