INTERNATIONAL STANDARD

ISO 21806-3

First edition
2020-10

Road vehicles — Media Oriented Systems Transport (MOST) —

Part 3:
Application layer conformance test plan

Véhicules routiers — Système de transport axé sur les médias —

Partie 3: Plan d'essais de conformité de la couche d'application

Reference number
ISO 21806-3:2020(E)

Contents

Page

Foreword

ISO (the International Organization for Standardization) is a worldwide federation of national standards bodies (ISO member bodies). The work of preparing International Standards is normally carried out through ISO technical committees. Each member body interested in a subject for which a technical committee has been established has the right to be represented on that committee. International organizations, governmental and non-governmental, in liaison with ISO, also take part in the work. ISO collaborates closely with the International Electrotechnical Commission (IEC) on all matters of electrotechnical standardization.

The procedures used to develop this document and those intended for its further maintenance are described in the ISO/IEC Directives, Part 1. In particular, the different approval criteria needed for the different types of ISO documents should be noted. This document was drafted in accordance with the editorial rules of the ISO/IEC Directives, Part 2 (see www.iso.org/directives).

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. ISO shall not be held responsible for identifying any or all such patent rights. Details of any patent rights identified during the development of the document will be in the Introduction and/or on the ISO list of patent declarations received (see www.iso.org/patents).

Any trade name used in this document is information given for the convenience of users and does not constitute an endorsement.

For an explanation of the voluntary nature of standards, the meaning of ISO specific terms and expressions related to conformity assessment, as well as information about ISO's adherence to the World Trade Organization (WTO) principles in the Technical Barriers to Trade (TBT), see www.iso.org/iso/foreword.html.

This document was prepared by Technical Committee ISO/TC 22, *Road vehicles*, Subcommittee SC 31, *Data communication*.

A list of all parts in the ISO 21806 series can be found on the ISO website.

Any feedback or questions on this document should be directed to the user's national standards body. A complete listing of these bodies can be found at www.iso.org/members.html.

Introduction

The Media Oriented Systems Transport (MOST) communication technology was initially developed at the end of the 1990s in order to support complex audio applications in cars. The MOST Cooperation was founded in 1998 with the goal to develop and enable the technology for the automotive industry. Today, MOST[1] enables the transport of high quality of service (QoS) audio and video together with packet data and real-time control to support modern automotive multimedia and similar applications. MOST is a function-oriented communication technology to network a variety of multimedia devices comprising one or more MOST nodes.

Figure 1 shows a MOST network example.

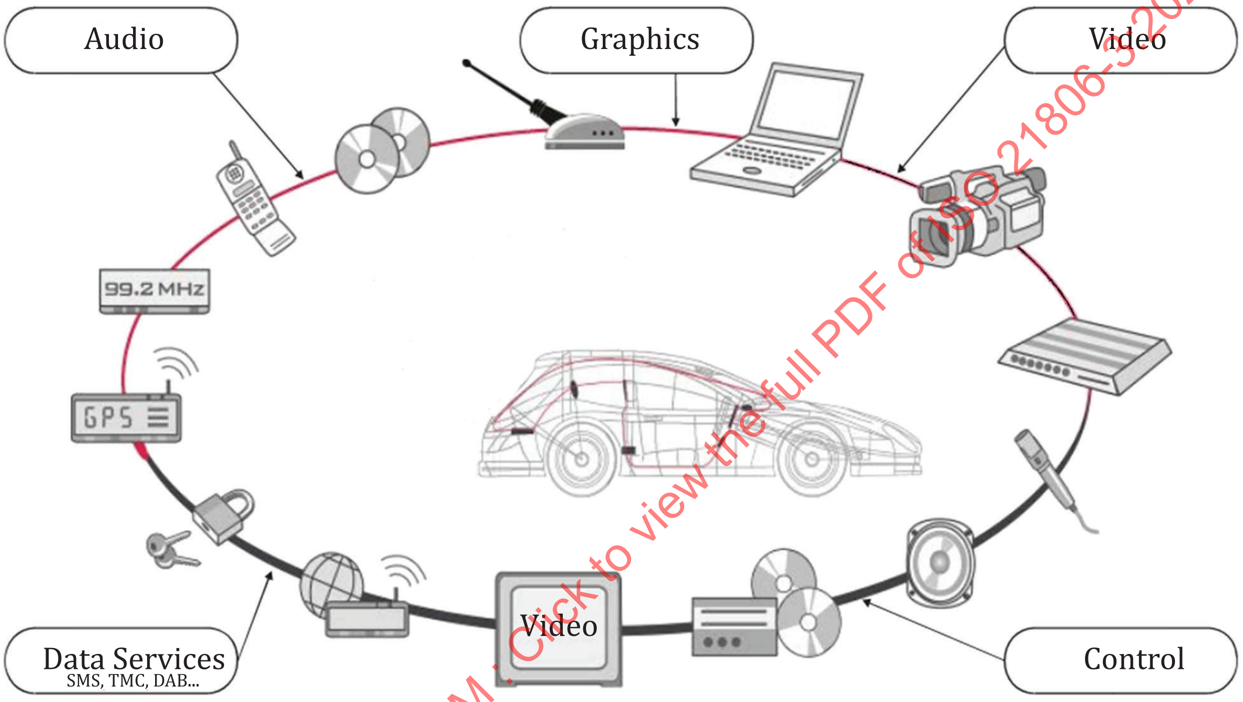


Figure 1 — MOST network example

The MOST communication technology provides:

— synchronous and isochronous streaming,

— small overhead for administrative communication control,

— a functional and hierarchical system model,

— API standardization through a function block (FBlock) framework,

— free partitioning of functionality to real devices,

— service discovery and notification, and

— flexibly scalable automotive-ready Ethernet communication according to ISO/IEC/IEEE 8802-3[2].

MOST is a synchronous time-division-multiplexing (TDM) network that transports different data types on separate channels at low latency. MOST supports different bit rates and physical layers. The network clock is provided with a continuous data signal.

1) MOST® is the registered trademark of Microchip Technology Inc. This information is given for the convenience of users of this document and does not constitute an endorsement by ISO.

Within the synchronous base data signal, the content of multiple streaming connections and control data is transported. For streaming data connections, bandwidth is reserved to avoid interruptions, collisions, or delays in the transport of the data stream.

MOST specifies mechanisms for sending anisochronous, packet-based data in addition to control data and streaming data. The transmission of packet-based data is separated from the transmission of control data and streaming data. None of them interfere with each other.

A MOST network consists of devices that are connected to one common control channel and packet channel.

In summary, MOST is a network that has mechanisms to transport the various signals and data streams that occur in multimedia and infotainment systems.

The ISO standards maintenance portal (https://standards.iso.org/iso/) provides references to MOST specifications implemented in today's road vehicles because easy access via hyperlinks to these specifications is necessary. It references documents that are normative or informative for the MOST versions 4V0, 3V1, 3V0, and 2V5.

The ISO 21806 series has been established in order to specify requirements and recommendations for implementing the MOST communication technology into multimedia devices and to provide conformance test plans for implementing related test tools and test procedures.

To achieve this, the ISO 21806 series is based on the open systems interconnection (OSI) basic reference model in accordance with ISO/IEC 7498-1[1] and ISO/IEC 10731[3], which structures communication systems into seven layers as shown in Figure 2. Stream transmission applications use a direct stream data interface (transparent) to the data link layer.

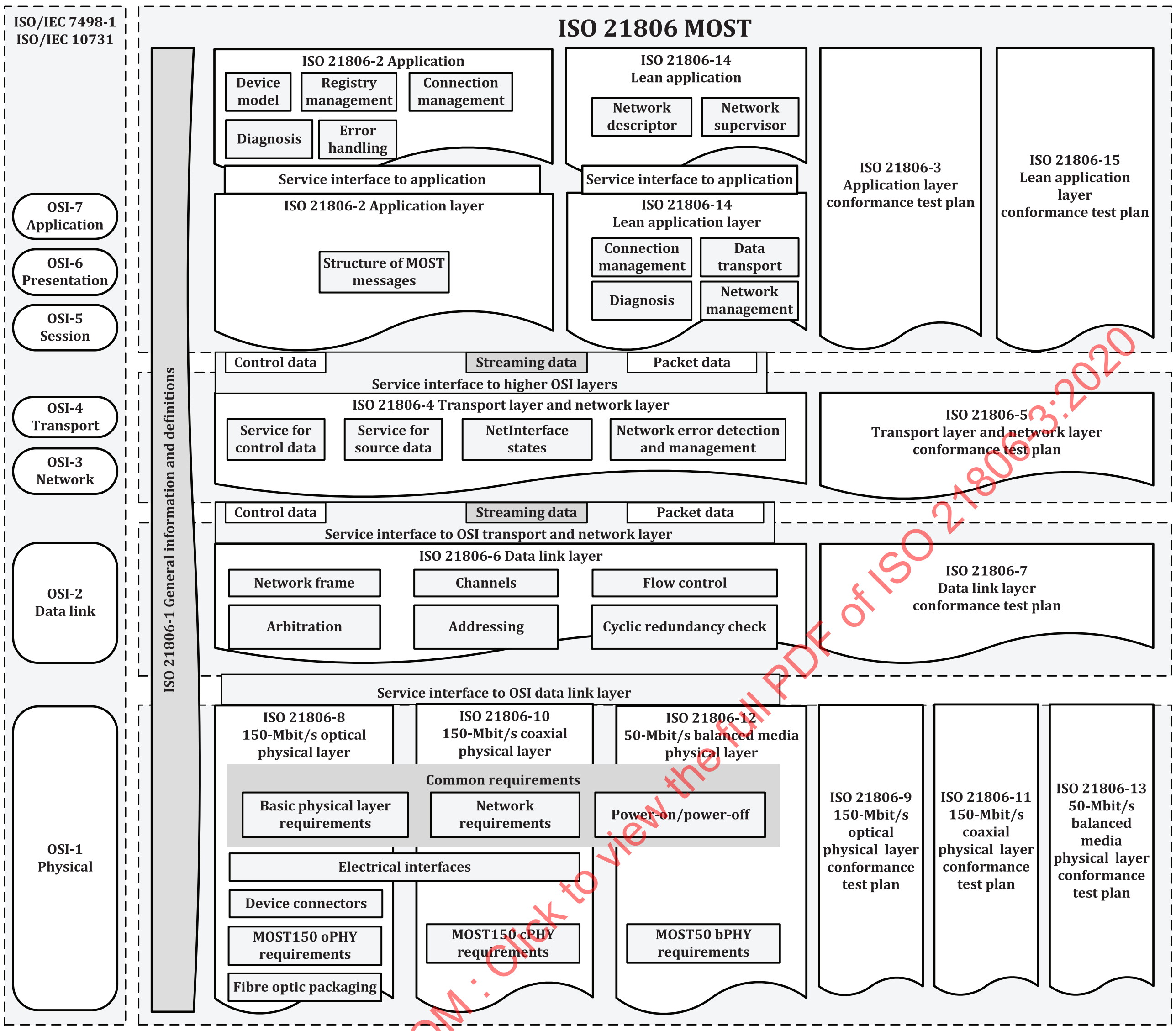

ISO/IEC 7498-1
ISO/IEC 10731
ISO 21806 MOST
OSI-7 Application
OSI-6 Presentation
OSI-5 Session
OSI-4 Transport
OSI-3 Network
OSI-2 Data link
OSI-1 Physical
ISO 21806-1 General information and definitions
ISO 21806-2 Application
Device model
Registry management
Connection management
Diagnosis
Error handling
Service interface to application
ISO 21806-2 Application layer
Structure of MOST messages
ISO 21806-14 Lean application
Network descriptor
Network supervisor
Service interface to application
ISO 21806-14 Lean application layer
Connection management
Data transport
Diagnosis
Network management
ISO 21806-3 Application layer conformance test plan
ISO 21806-15 Lean application layer conformance test plan
Control data
Streaming data
Packet data
Service interface to higher OSI layers
ISO 21806-4 Transport layer and network layer
Service for control data
Service for source data
NetInterface states
Network error detection and management
ISO 21806-5 Transport layer and network layer conformance test plan
Control data
Streaming data
Packet data
Service interface to OSI transport and network layer
ISO 21806-6 Data link layer
Network frame
Channels
Flow control
Arbitration
Addressing
Cyclic redundancy check
ISO 21806-7 Data link layer conformance test plan
Service interface to OSI data link layer
ISO 21806-8 150-Mbit/s optical physical layer
ISO 21806-10 150-Mbit/s coaxial physical layer
ISO 21806-12 50-Mbit/s balanced media physical layer
Common requirements
Basic physical layer requirements
Network requirements
Power-on/power-off
Electrical interfaces
Device connectors
MOST150 oPHY requirements
Fibre optic packaging
MOST150 cPHY requirements
MOST50 bPHY requirements
ISO 21806-9 150-Mbit/s optical physical layer conformance test plan
ISO 21806-11 150-Mbit/s coaxial physical layer conformance test plan
ISO 21806-13 50-Mbit/s balanced media physical layer conformance test plan

Figure 2 — The ISO 21806 series reference according to the OSI model

The International Organization for Standardization (ISO) draws attention to the fact that it is claimed that compliance with this document may involve the use of a patent.

ISO takes no position concerning the evidence, validity and scope of this patent right.

The holder of this patent right has assured ISO that he/she is willing to negotiate licences under reasonable and non-discriminatory terms and conditions with applicants throughout the world. In this respect, the statement of the holder of this patent right is registered with ISO. Information may be obtained from the patent database available at www.iso.org/patents.

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights other than those in the patent database. ISO shall not be held responsible for identifying any or all such patent rights.

Road vehicles — Media Oriented Systems Transport (MOST) —

Part 3: **Application layer conformance test plan**

1 Scope

This document specifies the conformance test plan (CTP) for the application layer for MOST, a synchronous time-division-multiplexing network, as specified in ISO 21806-2.

This document specifies conformance test cases (CTCs) in the following categories:

— device model;

— data and basic data types;

— registry management;

— connection management;

— error management;

— diagnosis.

Interoperability testing is not in the scope of this document.

2 Normative references

The following documents are referred to in the text in such a way that some or all of their content constitutes requirements of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

ISO/IEC 9646-1:1994, *Information technology — Open Systems Interconnection — Conformance testing methodology and framework — Part 1: General concepts*

ISO 21806-1:2020, *Road vehicles — Media Oriented Systems Transport (MOST) — Part 1: General information and definitions*

ISO 21806-2:2020, *Road vehicles — Media Oriented Systems Transport (MOST) — Part 2: Application layer*

ISO 21806-4:2020, *Road vehicles — Media Oriented Systems Transport (MOST) — Part 4: Transport layer and network layer*

3 Terms and definitions

For the purposes of this document, the terms and definitions given in ISO 21806-1, ISO 21806-2, ISO 21806-4, ISO/IEC 9646-1, and the following apply.

ISO and IEC maintain terminological databases for use in standardisation at the following addresses:

— ISO Online browsing platform: available at https://www.iso.org/obp

— IEC Electropedia: available at http://www.electropedia.org/

3.1
REPEAT
pseudo code command for an iteration

3.2
REPEAT END
pseudo code command for ending an iteration

4 Symbols and abbreviated terms

4.1 Symbols

---	empty cell/undefined

4.2 Abbreviated terms

CTC	conformance test case
CTP	conformance test plan
CR	central registry
DR	decentral registry
IUT	implementation under test
LT	lower tester
MPI	maximum position information
MSC	Message Sequence Chart
NCE	network change event
OSI	Open Systems Interconnection
UT	upper tester

5 Conventions

This document is based on OSI service conventions as specified in ISO/IEC 10731[3] and ISO/IEC 9646-1 for conformance test system set-up.

6 CTP overview

6.1 Test set-up

All CTCs are based on the same test set-up with an upper tester (UT) and a lower tester (LT). The LT contains the lower tester pre-IUT (LT pre-IUT) and the lower tester post-IUT (LT post-IUT).

Figure 3 specifies the test set-up.

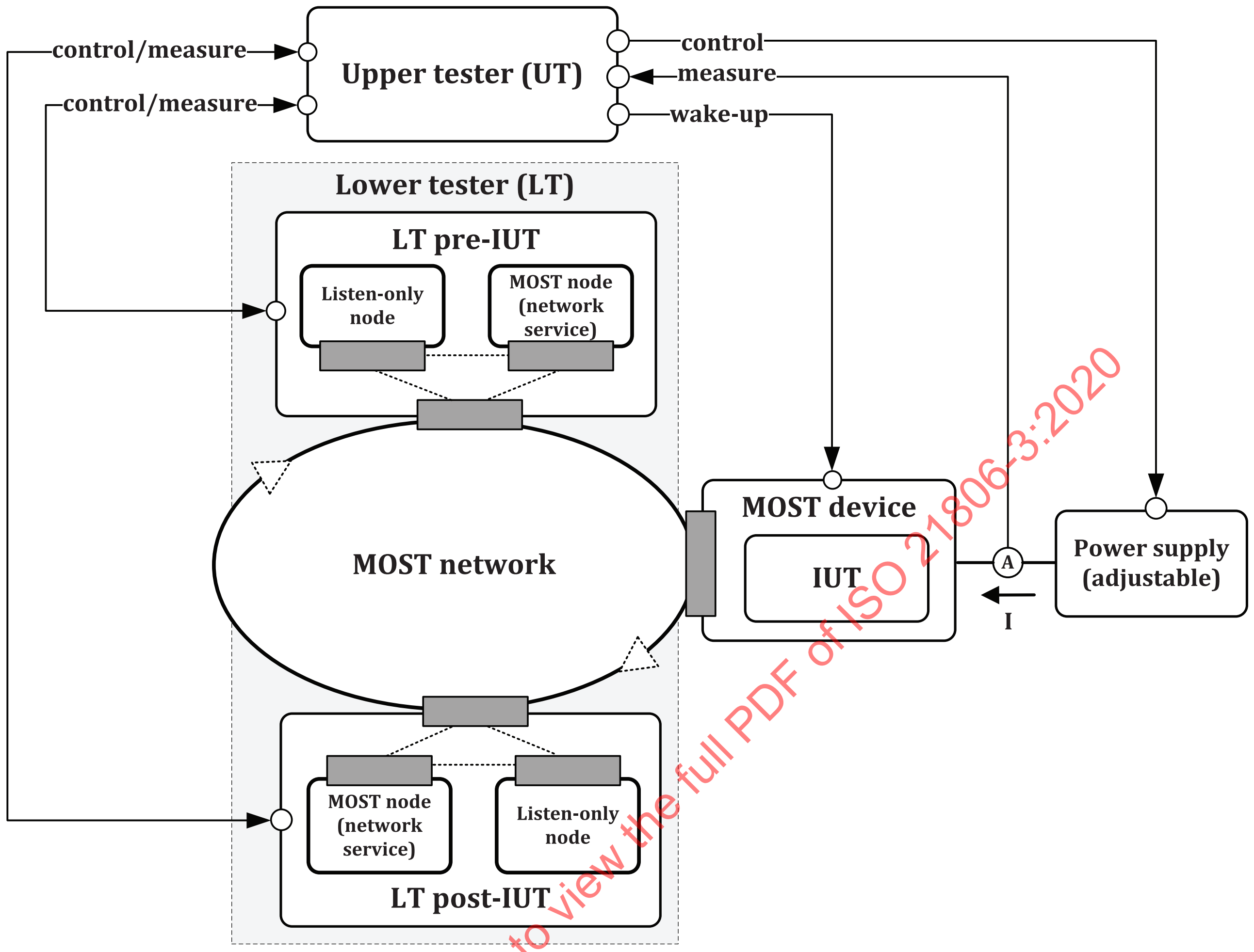


Figure 3 — Test set-up

The LT pre-IUT and the LT post-IUT implement the application layer services and the lower layer services of a MOST node in accordance with the ISO 21806 series. They also contain a listen-only node in front of the MOST node to log the whole communication. The MOST node is able to operate as TimingMaster or TimingSlave; alternatively, it can be physically disconnected from the MOST network. If it is disconnected, the associated LT pre-IUT or LT post-IUT serves as listen-only node.

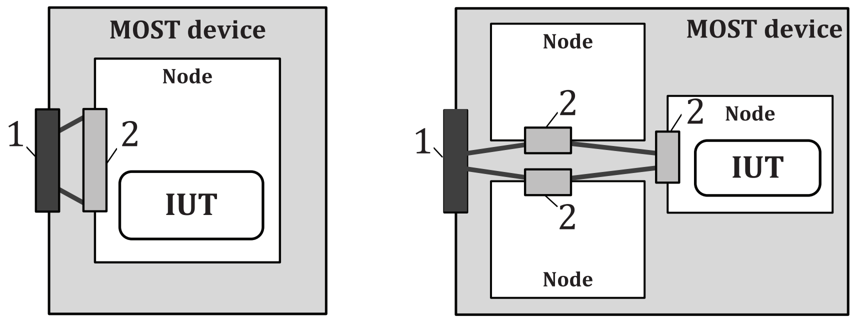
Every CTC specifies the roles of the LT pre-IUT and the LT post-IUT.

During testing of the MOST device that implements the IUT, avoid over-temperature by following the manufacturer recommendations regarding cooling.

The power supply of the MOST device that contains the IUT is adjustable and the power consumption can be monitored by the UT. This is necessary to determine whether a node has entered `s_NetInterface_Sleep`.

A MOST device contains one or more nodes, which are connected to an external MOST physical interface. One of the nodes contains the implementation under test (IUT). All tests and timings, specified by the CTP, are related to the external MOST physical interface.

Figure 4 shows a MOST device with one node and a MOST device with three internal nodes.



1 external MOST physical interface

2 internal MOST physical interface

Figure 4 — MOST device with one node and MOST device with three nodes

6.2 Conformance test plan organisation

CTCs are independent of one another. Each CTC checks the behaviour of the IUT for requirements stated in ISO 21806-2. Within CTCs, which require variations of individual parameters, each specified value of the parameter is iterated.

The measurement uncertainty for each CTC shall be in accordance with Annex A.

7 CTP general information

7.1 CTC remarks

7.1.1 Timer naming

For conformance testing of the IUT, the UT and LT need minimum and maximum timers. The names of the timers used by this document are based on ISO 21806-2 and ISO 21806-4. To obtain the timer name, for minimum and maximum, "$_{\text{min}}$" and "$_{\text{max}}$" are appended, respectively. Table 1 shows a timer naming definition example for t_{Config}.

Table 1 — Timer naming example

Name	Minimum value name	Typical value name	Maximum value name	Unit	Purpose
t_{Config}	$t_{\text{Config\_min}}$	t_{Config}	$t_{\text{Config\_max}}$	ms	Time before `ev_Init_Error_Shutdown` or delay for RBD result.

7.1.2 Deadlock prevention

This document specifies the timeouts $t_{\text{DeadLockShort}}$, $t_{\text{DeadLockMid}}$, and $t_{\text{DeadLockLong}}$ to prevent deadlock situations during conformance testing. These are the default values:

— $t_{\text{DeadLockShort}}$: 1 s;

— $t_{\text{DeadLockMid}}$: 20 s;

— $t_{\text{DeadLockLong}}$: 5 min.

These timeouts are only relevant for conformance testing and may be extended.

7.1.3 Un-initialised logical node address

The variable `uninitialised_node_address` is defined as the address of an un-initialised node, which is specified in ISO 21806-2.

7.1.4 Addresses of MOST nodes in the LT

The address of a MOST node in the LT is the default logical node address corresponding to the node position.

If this address is in conflict with the address of a node that contains the IUT (e.g. if a supplier uses static addresses in the dynamic address range), the affected MOST node in the LT shall use a valid free address.

7.1.5 Device manufacturer information list

This list contains all information that is provided by the device manufacturer for conformance testing. It also includes remarks and references to corresponding CTCs.

Table 2 shows the device manufacturer information list, which does not include information stored in FBlock EnhancedTestability.

Table 2 — Device manufacturer information list

Category	Item/property	Description	Reference to CTC
MOST network configuration	IUT in the TimingMaster	Determines whether the IUT is part of the TimingMaster.	All CTCs
	IUT in the NetworkMaster	Determines whether the IUT is part of the NetworkMaster.	All CTCs
	IUT in the PowerMaster	Determines whether the IUT is part of the PowerMaster.	All CTCs
	IUT in the connection manager	---	CTC_3.1-3, CTC_3.1-4, CTC_3.1-5, CTC_3.1-6, CTC_3.2-3, CTC_3.2-4, CTC_3.2-5, CTC_3.2-6, CTC_3.2-7, CTC_3.2-8, CTC_3.2-9, CTC_3.2-14
	Multi-node device	If the IUT is part of a MOST device that contains more than one node, the following information is provided: — number of nodes in the MOST device; — topology of the MOST device (position of PowerMaster and TimingMaster/ NetworkMaster); — position of the node that contains the IUT.	All CTCs
	IUT sample frequency	If the IUT is not part of the TimingMaster, the LT provides the correct network frame rate (44,1 kHz or 48,0 kHz).	All CTCs
	Required value of boundary descriptor (if the TimingMaster is in the LT)	Value of the boundary descriptor. Unless otherwise stated, all CTCs are performed with this value of the boundary descriptor.	All CTCs
	$mi_{\text{MaxInvalidReg}}$	The maximum number of permitted conflicting node address registrations by a NetworkSlave.	CTC_2.6.2-3a
	$mi_{\text{MaxSetNewInstID}}$	When an invalid InstID registration occurs, the NetworkMaster sends a request to the NetworkSlave for setting a new InstID.	CTC_2.6.2-6
	$t_{\text{Config\_max}}$	Time before ev_Init_Error_Shutdown or delay for RBD result.	CTC_2.1.1-6b
	$t_{\text{ConfigurationAnnounce}}$	Limit for the NetworkMaster to set the central registry state.	CTC_2.6.2-4a, CTC_2.6.2-5
	$t_{\text{WaitForAnswer\_min}}$ $t_{\text{WaitForAnswer\_max}}$	Time the NetworkMaster waits for all NetworkSlaves to respond.	CTC_2.6.2-1, CTC_2.6.2-3b, CTC_2.6.2-5

Table 2 *(continued)*

Category	Item/property	Description	Reference to CTC
Power management	Node that contains the IUT supports s_NetInterface_Sleep	Determines whether the node that contains the IUT supports s_NetInterface_Sleep: — yes: the MOST device that contains the IUT reduces its power consumption below threshold before timeout expires; — no: the reduction of power consumption is not detectable.	CTC_2.3.2-3
	s_NetInterface_Sleep: $I_{\mathrm{NetInterfaceSleep\_Threshold}}$	Threshold of current for s_NetInterface_Sleep detection	See 7.1.6.
	s_NetInterface_Sleep: $t_{\mathrm{PwrSwitchOffDelay\_min}}$ $t_{\mathrm{PwrSwitchOffDelay\_max}}$	$t_{\mathrm{PwrSwitchOffDelay\_min}}$ Specific timeout for s_NetInterface_Sleep; after the end of network activity, the node that contains the IUT does not enter s_NetInterface_Sleep (reduced power consumption) before $t_{\mathrm{PwrSwitchOffDelay\_min}}$ expires. $t_{\mathrm{PwrSwitchOffDelay\_max}}$ MOST device specific timeout for s_NetInterface_Sleep; after the end of network activity, the node that contains the IUT enters s_NetInterface_Sleep (reduced power consumption) before $t_{\mathrm{PwrSwitchOffDelay\_max}}$ expires.	CTC_2.3.2-3, CTC_2.6.4-1
	Wake-up preconditions	Preconditions for the node that contains the IUT for wake-up. Supplemented by information whether the node that contains the IUT needs additional conditions during operation (e.g. ignition ON) to stay in s_NetInterface_Normal_Operation.	See 7.1.6.
	Node that contains the IUT is capable of waking via network startup (i.e. switching on its MOST output)	---	CTC_2.4.1-2
	Delay between connection to power (of the MOST device that contains the IUT) and the ability of the node that contains the IUT to detect wake-up events	Potentially, the UT (see Figure 3) waits for a short period of time between connecting the MOST device that contains the IUT to power and switching on the MOST output to wake up the node that contains the IUT. Otherwise, the node that contains the IUT does not detect a wake-up event.	All CTCs
	s_NetInterface_Normal_Operation: Delay until all FBlocks of the node that contains the IUT are available after Configuration.Status(OK) (equivalent to $t_{\mathrm{WaitForApplication}}$)	This delay covers: — NetworkMaster: period of time the node that contains the IUT needs to add own FBlocks to its central registry after Configuration.Status(OK); — NetworkSlave: delay between ev_Init_Ready and availability of application.	See 7.1.6. CTC_2.6.2-3b
	$t_{\mathrm{WaitBeforeScan}}$	Specific limit for the NetworkMaster to start an FBlock scan.	CTC_2.4.1-9, CTC_2.6.2-5, CTC_2.6.4-8

Table 2 *(continued)*

Category	Item/property	Description	Reference to CTC
Addressing	Node that contains the IUT uses static node address in dynamic address range	If the node that contains the IUT uses a static logical node address that is in the specified dynamic address range, the address is provided.	See 7.1.6. CTC_2.6.4-4
	Free address	Logical node address that may be used by a MOST node in the LT during testing.	CTC_2.6.4-3
	Free FBlock range	FBlocks that are not used by the node that contains the IUT and which may be used by the UT.	CTC_2.6.2-4b, CTC_2.6.2-4c
	Group address of the node that contains the IUT	---	CTC_2.7-1
General communication	t_{Property}	Limit for responding to a command that reads a property.	CTC_2.1.0-1, CTC_2.1.0-2, CTC_2.8.3-2, CTC_2.8.4-3, CTC_3.0-1, CTC_3.1-1, CTC_3.2-1
	$t_{\text{NotificationProperty}}$	Limit for reacting to a `Notification.Set` message.	CTC_2.8.3-1a, CTC_2.8.3-1b, CTC_2.8.3-7
Physical parameter (voltage levels)	$U_{\text{IUT\_Operating}}$	At this voltage level, the MOST device that contains the IUT operates normally. Unless otherwise stated, all CTCs are performed at this voltage level.	All CTCs
Messaging	Node that contains the IUT supports segmented messages	The node that contains the IUT is able to send and receive segmented messages.	CTC_2.8.4-2, CTC_2.8.4-3, CTC_2.8.4-7, CTC_2.8.4-8
Sink/Source	List of FBlocks, containing sink and/or source functionality	The list contains all FBlocks reported by `NetBlock.FBlockIDs.Status`.	CTC_3.0-1
	MOST devices with sinks: List of all supported sink numbers with `ContentType`, `ContentDescription` (data type of the parameter) and `TransmissionClass`	---	CTC_3.1-1, CTC_3.1-3, CTC_3.1-4, CTC_3.1-5, CTC_3.1-6
	MOST devices with sources: List of all supported source numbers with `ContentType`, `ContentDescription` (data type of the parameter) and `TransmissionClass`	---	CTC_3.2-1, CTC_3.2-3, CTC_3.2-4, CTC_3.2-5, CTC_3.2-6, CTC_3.2-7, CTC_3.2-14
	MOST devices with sources: `BlockWidth` and `ConnectionLabel`	---	CTC_3.2-3
	Node that contains the IUT supports `SourceActivity`	---	CTC_3.2-14

7.1.6 States of the node that contains the IUT

Table 3 specifies how the NetInterface state `s_NetInterface_Normal_Operation` is effectuated and detected in the node that contains the IUT.

Table 3 — Effectuate and detect s_NetInterface_Normal_Operation

Effectuate state	Detect state
a) The IUT is contained in a NetworkSlave: — the UT shall start the network; — wait for the node that contains the IUT to open its bypass (MPI is nominal[a]); — send `NetworkMaster.Configuration.Status(NotOK)`; — perform an FBlock scan (including retries if the address of the node that contains the IUT is invalid); — send `NetworkMaster.Configuration.Status(OK)`; — wait for $t_{\text{WaitForApplication}}$.	a) The IUT is contained in a NetworkSlave: the node that contains the IUT responds to `NetBlock.FBlockIDs.Get`.
b) The IUT is contained in the NetworkMaster: — the UT shall behave like a NetworkSlave; it shall process and respond to all requests from the node that contains the IUT so that the node can enter central registry state OK; — the UT shall respond to an FBlock scan by the node that contains the IUT; additionally, the UT shall wait for the node to open its bypass (MPI is nominal[a]) if the node is part of a multi-node device; — finally, the UT shall wait for $t_{\text{WaitForApplication}}$.	b) The IUT is contained in the NetworkMaster: the node that contains the IUT responds to `NetBlock.FBlockIDs.Get`.

[a] The nominal MPI is the total number of nodes in the set-up, based on the IUT manufacturer information and the test equipment.

Table 4 specifies how the NetInterface state `s_NetInterface_Sleep` is effectuated and detected in the node that contains the IUT.

Table 4 — Effectuate and detect s_NetInterface_Sleep state

Effectuate state	Detect state
Switch off MOST output	a) When `s_NetInterface_Sleep` is detectable the following applies: — when monitoring power consumption, the current reaches or drops below $I_{\text{NetInterfaceSleep\_Threshold}}$; — if the LT detects network activity when $t_{\text{PwrSwitchOffDelay\_max}}$ expires, the test shall be stopped; the timer shall be started as soon as the node that contains the IUT switches off the MOST output; — if the current does not reach or drop below $I_{\text{NetInterfaceSleep\_Threshold}}$ within timeout $t_{\text{PwrSwitchOffDelay\_max}}$, the test shall be stopped. b) When `s_NetInterface_Sleep` is not detectable the following applies: — when the timeout $t_{\text{PwrSwitchOffDelay\_max}}$ expires and if the LT does not detect network activity, it shall be assumed that the node that contains the IUT is in `s_NetInterface_Sleep`, independent of power consumption; — if the LT detects network activity when $t_{\text{PwrSwitchOffDelay\_max}}$ expires, the test shall be stopped.

Table 5 specifies how the NetInterface state `s_NetInterface_Off` is effectuated and detected in the node that contains the IUT.

Table 5 — Effectuate and detect s_NetInterface_Off state

Effectuate state	Detect state
The LT shall switch off the MOST output.	No network activity

7.1.7 Procedures

Table 6 specifies the procedures of the LT.

Table 6 — Procedures of the LT

Purpose	Description
Perform wake-up	a) If the LT contains the TimingMaster, it shall execute this sequence to perform a wake-up: 1. switch on the MOST output; 2. wait for network activity (timeout $t_{\text{DeadLockShort}}$); 3. wait for stable lock (timeout $t_{\text{DeadLockMid}}$). b) If the LT does not contain the TimingMaster, it shall execute this sequence to perform a wake-up: 1. generate a wake-up event; 2. wait for network activity; 3. switch on the MOST output; 4. wait for stable lock; 5. wait for the lock flag to evaluate to true. If the LT does not generate a wake-up event and detects network activity, it shall switch on the MOST output. In some cases, the node that contains the IUT needs some preconditions for wake-up. These preconditions are established before testing is started. The preconditions depend on the device manufacturer. If `EnhancedTestability.AutoWakeup` is triggered, the node that contains the IUT does not enter `s_NetInterface_Sleep` before creating the corresponding wake-up event. The node enters `s_NetInterface_Off`. This state is not detectable by monitoring the power consumption of the MOST device that contains the IUT. With entering state `s_NetInterface_Off`, the node that contains the IUT switches off the MOST output.
Perform shutdown	a) If the IUT is part of the PowerMaster, the LT shall execute this sequence to perform shutdown: — trigger shutdown by means of FBlock EnhancedTestability; — if no network activity is detected, the node that contains the IUT has performed shutdown; — if $t_{\text{DeadLockMid}}$ expires after triggering shutdown and the LT still detects network activity, it shall switch off the MOST output. b) If the IUT is part of a PowerSlave, the LT shall switch off the MOST output to perform shutdown. c) If the IUT is part of the PowerMaster, no preconditions are established that prevent the node that contains the IUT from performing shutdown.
Generate unlock	To generate an unlock event of predictable duration, the LT

Table 6 *(continued)*

Purpose	Description
	— shall invalidate or delay the preamble at the beginning of at least every third network frame during the period of unlock (see ISO 21806-6), and — shall avoid a PLL unlock.
Network change event with unlock	The NCE shall be generated between the TimingMaster and the node that contains the IUT.
Network change event without unlock	The NCE shall be generated between the node that contains the IUT and the TimingMaster.

7.1.8 Violation of prerequisites of the CTC

If the node that contains the IUT does not meet the prerequisites of the CTC (such as network activity, lock, FBlock scan performed, central registry state OK), the CTC results in "IUT not ok: The IUT does not meet the prerequisites".

7.2 CTC items

7.2.1 FBlock EnhancedTestability

The FBlock EnhancedTestability is used to trigger sequences that are specified in the CTC. These are normally triggered by a project specific, sometimes complicated, mechanism. FBlock EnhancedTestability implements neither notification nor processing messages. The UT shall initialise FBlock EnhancedTestability every time the `s_NetInterface_Normal_Operation` state is reached. The FBlock is only available during `s_NetInterface_Normal_Operation`. All properties are reset to their default state when entering `s_NetInterface_Normal_Operation`, unless otherwise stated. The functions in this FBlock describe a general interface for starting functionality partly implemented in the application, partly in the MOST network service. If an application callback returns wrong or unexpected values, the FBlock sends an error message with ErrorCode $0B_{16}$ (device malfunction).

If the FBlock EnhancedTestability returns errors, the corresponding CTC result is indicated as “IUT not ok”.

7.2.2 Multi-node devices

A multi-node device is a MOST device with an external MOST physical interface, which is connected to more than one internal node.

When dealing with devices with several external MOST physical interfaces, each interface shall be treated as a separate IUT. One of those interfaces may belong to a multi-node device. In a multi-node device, each node shall be treated individually.

NOTE A MOST device passes conformance testing successfully if all contained IUTs pass all relevant CTCs.

7.2.3 Node kinds excluded from conformance testing

CTCs for remote controlled nodes according to ISO 21806-2 are not in the scope of this document.

8 CTC specification

8.1 Static FBlock behaviour

8.1.1 CTC_2.1.0-1 – Generic FBlock property test

Table 7 specifies the CTC_2.1.0-1 - Generic FBlock property test.

Table 7 — CTC_2.1.0-1 - Generic FBlock property test

Item	Content
CTC # – Title	CTC_2.1.0-1 - Generic FBlock property test
Purpose	This CTC verifies that FBlock communication with properties of the NetBlock, the Network-Master, sinks, and sources is possible. This CTC applies to all MOST devices.
Reference	ISO 21806-2:2020, 7.6 AL – Operation type (OPType)
Prerequisite	The UT shall effectuate `s_NetInterface_Normal_Operation` in the node that contains the IUT.
Set-up	— If the IUT is part of a MOST device that contains the TimingMaster, the LT pre-IUT shall be a TimingSlave. — If the IUT is part of a MOST device that does not contain the TimingMaster, the LT pre-IUT shall be the TimingMaster. — The LT post-IUT shall be a listen-only node.
Step	1. The UT shall set the `FBlockID` and `InstID` to the values of the current test iteration. 2. The UT, using the LT pre-IUT, shall send `FBlockID.InstID.FktIDs.Get` to the IUT to retrieve the list of implemented functions of the IUT. 3. The UT, in an outer loop, shall loop through the `FktIDs` that were returned in the list and correspond to properties. 4. The UT, in an inner loop, shall loop through the `OPTypes` (0_{16} to F_{16}). 5. UT: for every iteration of the loop, the UT, using the LT pre-IUT, shall send `FBlockID.InstID.FktID.OPType` to the IUT. 6. The UT shall evaluate the response of the IUT as specified in Table 8. 7. The UT shall shut down the IUT and re-establish the prerequisites for the next test iteration, after processing all inner and outer loop iterations.
Iteration	REPEAT Step 1 to Step 7 with the NetBlock and every available `InstID` (obtained by `NetBlock.FBlockIDs)` of the following FBlocks of the — IUT: NetworkMaster; — IUT: FBlocks containing sink functions (identified by CTC TEST_GSI_GSO_identification); — IUT: FBlocks containing source functions (identified by CTC TEST_GSI_GSO_identification). REPEAT END.
Expected response	Step 6: IUT ok: the IUT provides permitted responses as specified in Table 8. Step 6: IUT not ok: the IUT does not respond as expected to the requests to properties.

Table 7 *(continued)*

Item	Content
Remark	1. Responses are only evaluated from actually tested `FBlockID`, `InstID` and `FktID`. 2. Any other message of the node that contains the IUT is ignored during the CTC. 3. The FBlock library determines whether the function is a property or a method. 4. Usually, a MOST device does not implement all functions and `OPTypes` of the FBlock library; if implemented, they are expected to behave according to the FBlock library. 5. No range checks are performed. 6. In the case that the `OPType` test allows "no answer" as permitted response, the observation time period is 2 × t_{Property} for properties respectively $t_{\mathrm{DeadLockMid}}$ for methods. 7. For the NetBlock, error 03_{16} (function not available) is also a permitted response. 8. All commands (`OPType` 0_{16} to 8_{16}) from the node that contains the IUT are ignored for all FBlocks. 9. For the NetBlock and the NetworkMaster FBlock, `OPTypes` 9_{16} to F_{16} are not tested.

Table 8 specifies the `OPTypes` for CTC_2.1.0-1 - Generic FBlock property test.

Table 8 — OPTypes for CTC_2.1.0-1 - Generic FBlock property test

OPType	OPType name and parameters	Permitted response
0_{16}	`Set` (without parameter)	— `Status`; — No answer (within 2 × t_{Property}); — All errors except 01_{16}, 02_{16}, 03_{16}, $0A_{16}$, $0C_{16}$.
1_{16}	`Get` (without parameter)	— `Status`; — All errors except 01_{16}, 02_{16}, 03_{16}, $0A_{16}$, $0C_{16}$.
2_{16}	`SetGet` (without parameter)	— `Status`; — All errors except 01_{16}, 02_{16}, 03_{16}, $0A_{16}$, $0C_{16}$.
3_{16}	`Increment(1)`	— `Status`; — All errors except 01_{16}, 02_{16}, 03_{16}, $0A_{16}$, $0C_{16}$.
4_{16}	`Decrement(1)`	— `Status`; — All errors except 01_{16}, 02_{16}, 03_{16}, $0A_{16}$, $0C_{16}$.
5_{16}	Not allowed	Not allowed
6_{16}	`StartResultAck` (`SenderHandle 1234`$_{16}$)	— `ErrorAck(SenderHandle 1234`$_{16}$, `ErrorCode 04`$_{16}$); — `Error(ErrorCode 04`$_{16}$).
7_{16}	`AbortAck` (`SenderHandle 1234`$_{16}$)	— `ErrorAck(SenderHandle 1234`$_{16}$, `ErrorCode 04`$_{16}$); — `Error(ErrorCode 04`$_{16}$).
8_{16}	`StartAck` (`SenderHandle 1234`$_{16}$)	— `ErrorAck(SenderHandle 1234`$_{16}$, `ErrorCode 04`$_{16}$); — `Error(ErrorCode 04`$_{16}$).
9_{16}	`ErrorAck`	No answer
A_{16}	`ProcessingAck`	No answer
B_{16}	`Processing`	No answer
C_{16}	`Status`	No answer

Table 8 *(continued)*

OPType	OPType name and parameters	Permitted response
$\mathtt{D}_{16}$	`ResultAck`	No answer
$\mathtt{E}_{16}$	Not allowed	Not allowed
$\mathtt{F}_{16}$	`Error`	No answer

8.1.2 CTC_2.1.0-2 – Generic FBlock method test

Table 9 specifies the CTC_2.1.0-2 - Generic FBlock method test.

Table 9 — CTC_2.1.0-2 - Generic FBlock method test

Item	Content
CTC # – Title	CTC_2.1.0-2 - Generic FBlock method test
Purpose	This CTC verifies that FBlock communication with methods of the NetBlock, the NetworkMaster, sinks, and sources is possible. This CTC applies to all MOST devices.
Reference	ISO 21806-2:2020, 7.6 AL – Operation type (OPType)
Prerequisite	The UT shall effectuate `s_NetInterface_Normal_Operation` in the node that contains the IUT.
Set-up	— If the IUT is part of a MOST device that contains the TimingMaster, the LT pre-IUT shall be a TimingSlave. — If the IUT is part of a MOST device that does not contain the TimingMaster, the LT pre-IUT shall be the TimingMaster. — The LT post-IUT shall be a listen-only node.
Step	1. The UT shall set the `FBlockID` and `InstID` to the values of the current test iteration. 2. The UT shall use the LT pre-IUT to send `FBlockID.InstID.FktIDs.Get` to the IUT to retrieve the list of implemented functions of the IUT. 3. The UT in an outer loop; the UT shall loop through the FktIDs that correspond to methods, which are reported in the list. 4. The UT in an inner loop, the UT shall loop through the `OPTypes` ($\mathtt{0}_{16}$ to $\mathtt{F}_{16}$). 5. The UT for every iteration of the loop, the UT, using the LT pre-IUT, shall send `FBlockID.InstID.FktID.OPType` to the IUT. 6. The UT shall evaluate the response of the IUT as specified in Table 10. 7. The UT after processing all inner and outer loop iterations, the UT shall shutdown the IUT and re-establish the prerequisites for the next test iteration.
Iteration	Repeat Step 1 to Step 7 with the NetBlock and every available `InstID` (obtained by `NetBlock.FBlockIDs)` of the following FBlocks of the — IUT: NetworkMaster; — IUT: FBlocks containing sink functions (identified by TEST_GSI_GSO_identification); — IUT: FBlocks containing source functions (identified by TEST_GSI_GSO_identification). Repeat END.
Expected response	Step 6: IUT ok: the IUT responds as expected to the requests to methods. Step 6: IUT not ok: the IUT does not respond as expected to the requests to methods.

Table 9 *(continued)*

Item	Content
Remark	1. Responses are only evaluated from actually tested `FBlockID`, `InstID` and `FktID`. 2. Any other message of the node that contains the IUT are ignored during the CTC. 3. The FBlock library determines whether the function is a property or a method. 4. Usually, a MOST device does not implement all functions and `OPTypes` of the FBlock library; if implemented, they are expected to behave according to the FBlock library. 5. There are no range checks. 6. Each `OPType` test presumes that the previous test completes the communication process, for example if after `StartResult` the `Result` is not received, then the process is expected to be terminated by sending `Abort`. If `Abort` fails, the IUT is reset to finish the test step and continue with the next step. 7. In the case that the `OPType` test allows "no answer" as permitted response, the observation time period is 2 × t_{Property} for properties respectively $t_{\text{DeadLockMid}}$ for methods. 8. In the case that the `OPType` test allows `Processing` as permitted response, `Processing` might be received before result or an error, maximum 50 `Processing` are accepted by the UT, then `Abort` is sent. 9. For the NetBlock, error 03_{16} (function not available) is also a permitted response. 10. All commands (`OPType` 0_{16} to 8_{16}) from the node that contains the IUT are ignored for all FBlocks. 11. For the NetBlock and the NetworkMaster FBlock, `OPTypes` 9_{16} to F_{16} not tested. 12. During this CTC it is considered that sending of an error is optional in function `DTCP_Control`.

Table 10 specifies the OPTypes for CTC_2.1.0-2 - Generic FBlock method test.

Table 10 — OPTypes for CTC_2.1.0-2 - Generic FBlock method test

OPType	OPType name and parameters	Permitted response
0_{16}	`Start`	— None; — `Result`; — `Processing`; — All errors except 01_{16}, 02_{16}, 03_{16}, $0A_{16}$, $0C_{16}$.
1_{16}	`Abort`	— None; — All errors except 01_{16}, 02_{16}, 03_{16}, $0A_{16}$, $0C_{16}$.
2_{16}	`StartResult`	— `ProcessingResult`; — All errors except 01_{16}, 02_{16}, 03_{16}, $0A_{16}$, $0C_{16}$.
3_{16}	`Increment`	— `Error` 04_{16}
4_{16}	`Decrement`	— `Error` 04_{16}
5_{16}	Not allowed	Not allowed

Table 10 *(continued)*

OPType	OPType name and parameters	Permitted response

6_{16}	`StartResultAck` (`SenderHandle 1234`$_{16}$)	— `ProcessingAck(SenderHandle 1234`$_{16}$);
		 — `ResultAck(SenderHandle 1234`$_{16}$);
		 — All `ErrorAcks` (`SenderHandle 1234`$_{16}$) except `01`$_{16}$, `02`$_{16}$, `03`$_{16}$, `0A`$_{16}$, `0C`$_{16}$;
		 — All `Errors` except `01`$_{16}$, `02`$_{16}$, `03`$_{16}$, `0A`$_{16}$, `0C`$_{16}$.

7_{16}	`AbortAck` (`SenderHandle 1234`$_{16}$)	— None; — All `ErrorAcks` (`SenderHandle 1234`$_{16}$) except `01`$_{16}$, `02`$_{16}$, `03`$_{16}$, `0A`$_{16}$, `0C`$_{16}$; — All `Errors` except `01`$_{16}$, `02`$_{16}$, `03`$_{16}$, `0A`$_{16}$, `0C`$_{16}$.

8_{16}	`StartAck` (`SenderHandle 1234`$_{16}$)	— None;
		 — `ResultAck(SenderHandle 1234`$_{16}$);
		 — `ProcessingAck(SenderHandle 1234`$_{16}$);
		 — All `ErrorAcks` (`SenderHandle 1234`$_{16}$) except `01`$_{16}$, `02`$_{16}$, `03`$_{16}$, `0A`$_{16}$, `0C`$_{16}$;
		 — All `Errors` except `01`$_{16}$, `02`$_{16}$, `03`$_{16}$, `0A`$_{16}$, `0C`$_{16}$.

9_{16}	`ErrorAck`	No answer
A_{16}	`ProcessingAck`	No answer
B_{16}	`Processing`	No answer
C_{16}	`Result`	No answer
D_{16}	`ResultAck`	No answer
E_{16}	Not allowed	Not allowed
F_{16}	`Error`	No answer

8.2 Power management

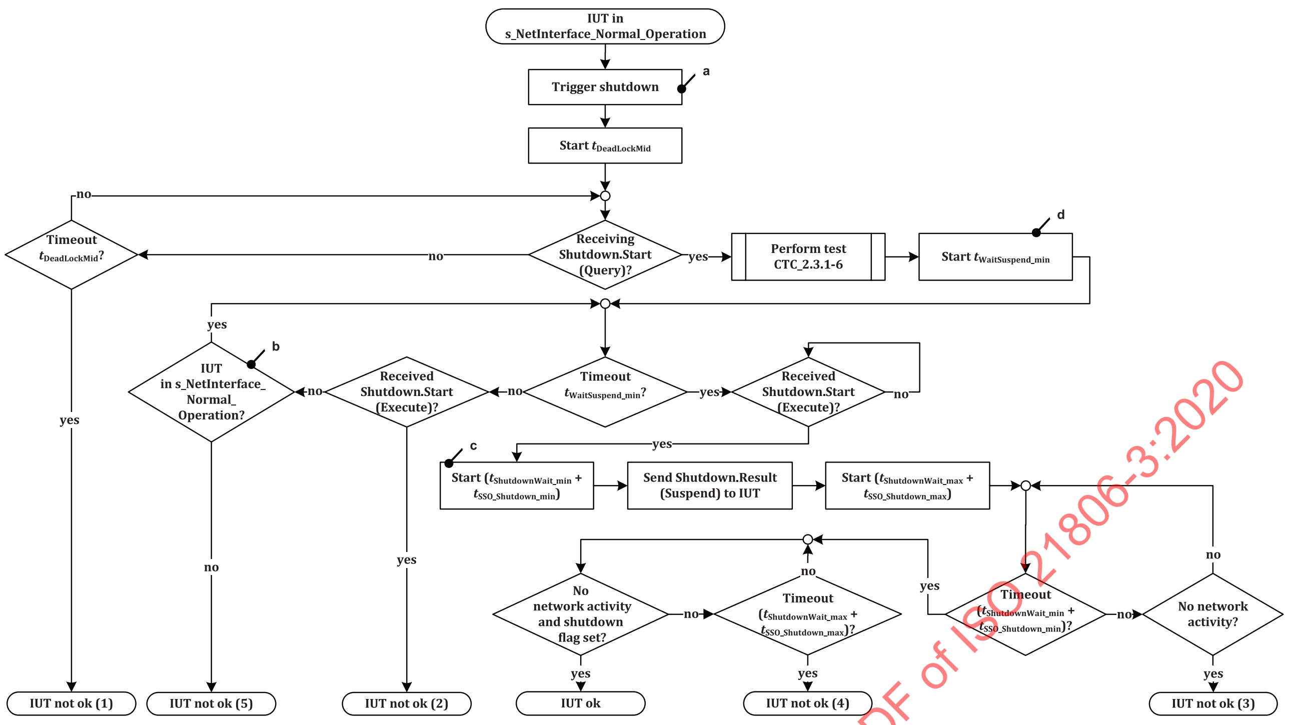
8.2.1 Power management – PowerMaster

8.2.1.1 CTC_2.3.1-3 – Timeout execute/timeout suspend test

Table 11 specifies the CTC_2.3.1-3 – Timeout execute/timeout suspend test.

Table 11 — CTC_2.3.1-3 – Timeout execute/timeout suspend test

Item	Content
CTC # – Title	CTC_2.3.1-3 – Timeout execute/timeout suspend test
Purpose	This CTC verifies that the PowerMaster is capable of correctly performing a shutdown, when there are no shutdown-suspend requests from the PowerSlaves within the specified time constraints: — for `Shutdown.Start(Query)`, $t_{\text{WaitSuspend}}$ is used; — for `Shutdown.Start(Execute)`, $t_{\text{ShutdownWait}} + t_{\text{SSOShutdown}}$ is used. This CTC applies to all MOST devices that contain the PowerMaster.
Reference	ISO 21806-2:2020, 6.9.3 APP – Network shutdown
Prerequisite	— The UT shall effectuate `s_NetInterface_Normal_Operation` in the node that contains the IUT. — Preconditions that prevent the node that contains the IUT from performing shutdown shall be switched off.
Set-up	— If the IUT is part of a MOST device that contains the TimingMaster, the LT pre-IUT shall be a TimingSlave. — If the IUT is part of a MOST device that does not contain the TimingMaster, the LT pre-IUT shall be the TimingMaster. — The LT post-IUT shall be a listen-only node. — PowerSlaves in multi-node devices shall be configured to not send `NetBlock.Shutdown.Result(Suspend)`.
Step	1. The UT shall execute the sequence as specified in Figure 5.
Iteration	Not applicable
Expected response	Step 1: IUT ok: the IUT performs a shutdown correctly. Step 1: IUT not ok (1): the IUT does not start the shutdown procedure. Step 1: IUT not ok (2): the IUT sends `Shutdown.Start(Execute)` too early. Step 1: IUT not ok (3): the IUT switches off the MOST output too early. Step 1: IUT not ok (4): the IUT switches off the MOST output too late or does not set the shutdown flag. Step 1: IUT not ok (5): the IUT does not perform `s_NetInterface_Normal_Operation` during $t_{\text{WaitSuspend}}$
Remark	---


IUT in s_NetInterface_Normal_Operation
Trigger shutdown
a
Start $t_{DeadLockMid}$
no
Timeout $t_{DeadLockMid}$?
no
Receiving Shutdown.Start (Query)?
yes
Perform test CTC_2.3.1-6
d
Start $t_{WaitSuspend\_min}$
yes
b
IUT in s_NetInterface_Normal_Operation?
no
Received Shutdown.Start (Execute)?
no
Timeout $t_{WaitSuspend\_min}$?
yes
Received Shutdown.Start (Execute)?
no
yes
c
Start ($t_{ShutdownWait\_min}$ + $t_{SSO\_Shutdown\_min}$)
Send Shutdown.Result (Suspend) to IUT
Start ($t_{ShutdownWait\_max}$ + $t_{SSO\_Shutdown\_max}$)
yes
no
yes
no
No network activity and shutdown flag set?
no
Timeout ($t_{ShutdownWait\_max}$ + $t_{SSO\_Shutdown\_max}$)?
yes
Timeout ($t_{ShutdownWait\_min}$ + $t_{SSO\_Shutdown\_min}$)?
no
No network activity?
no
yes
yes
yes
IUT not ok (1)
IUT not ok (5)
IUT not ok (2)
IUT ok
IUT not ok (4)
IUT not ok (3)

a The UT, using the LT pre-IUT, shall send `EnhancedTestability.Shutdown.Start`(00_{16}) to instruct the IUT to perform a shutdown.
The IUT performs a shutdown by sending `NetBlock.Shutdown.Start(Query)`.

b `s_NetInterface_Normal_Operation` shall be checked every 500 ms.

c The IUT waits for $t_{ShutdownWait\_min}$ + $t_{SSO\_Shutdown\_min}$ before switching off the MOST output.
Message is received within $t_{DeadLockLong}$. Otherwise, the CTC is stopped to avoid deadlocks.

d The IUT waits for $t_{WaitSuspend\_min}$ (CTC_2.3.1-6 can be performed) before sending `NetBlock.Shutdown.Start(Execute)`. It ignores `NetBlock.Shutdown.Result(Suspend)`.

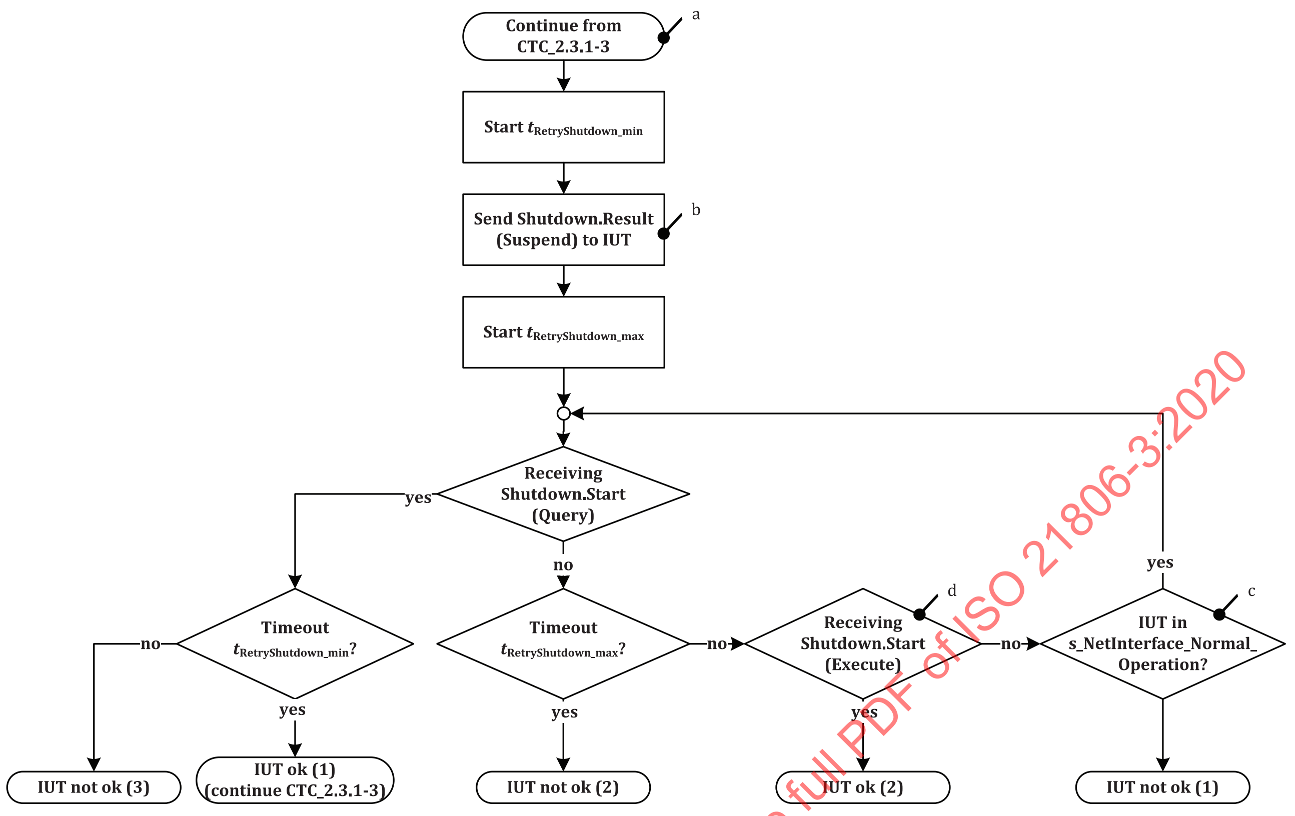
Figure 5 — CTC_2.3.1-3 - Timeout execute/timeout suspend test

8.2.1.2 CTC_2.3.1-6 – Timeout suspend test

Table 12 specifies the CTC_2.3.1-6 – Timeout suspend test.

Table 12 — CTC_2.3.1-6 – Timeout suspend test

Item	Content
CTC # – Title	CTC_2.3.1-6 – Timeout suspend test
Purpose	This CTC verifies that the PowerMaster is capable of postponing the execution of a shutdown due to shutdown-suspend requests from the PowerSlaves and that it repeats the shutdown attempt within the specified time constraints. This CTC applies to all MOST devices that contain the PowerMaster.
Reference	ISO 21806-2:2020, 6.9.3 APP – Network shutdown
Prerequisite	The UT shall effectuate `s_NetInterface_Normal_Operation` in the node that contains the IUT.
Set-up	— If the IUT is part of a MOST device that contains the TimingMaster, the LT pre-IUT shall be a TimingSlave. — If the IUT is part of a MOST device that does not contain the TimingMaster, the LT pre-IUT shall be the TimingMaster. — The LT post-IUT shall be a listen-only node. — PowerSlaves in multi-node devices shall be configured to not send `Shutdown.Result(Suspend)`.
Step	1. The UT shall execute the sequence as specified in Figure 6.
Iteration	Not applicable
Expected response	Step 1: IUT ok (1): the IUT postpones the execution of the shutdown. Step 1: IUT ok (2): the IUT overrides the suspend request. Step 1: IUT not ok (1): the IUT does not remain in `s_NetInterface_Normal_Operation` during $t_{\text{RetryShutdown}}$. Step 1: IUT not ok (2): the IUT does not send `Shutdown.Start(Query)`. Step 1: IUT not ok (3): the IUT sends `Shutdown.Start(Query)` too early.
Remark	---


Continue from CTC_2.3.1-3
a
Start $t_{RetryShutdown\_min}$
Send Shutdown.Result (Suspend) to IUT
b
Start $t_{RetryShutdown\_max}$
Receiving Shutdown.Start (Query)
yes
no
Timeout $t_{RetryShutdown\_min}$?
no
yes
Timeout $t_{RetryShutdown\_max}$?
no
yes
Receiving Shutdown.Start (Execute)
d
no
yes
IUT in s_NetInterface_Normal_ Operation?
c
yes
IUT not ok (3)
IUT ok (1) (continue CTC_2.3.1-3)
IUT not ok (2)
IUT ok (2)
IUT not ok (1)

[a] The CTC shall continue from CTC_2.3.1-3.

[b] The UT, using the LT pre-IUT, shall send a `Shutdown.Result(Suspend)` to the IUT before $t_{WaitSuspend}$ expires.

[c] The node that contains the IUT stays in `s_NetInterface_Normal_Operation` until $t_{RetryShutdown\_min}$ expires. `s_NetInterface_Normal_Operation` shall be checked every 500 ms.

[d] The IUT starts a new shutdown attempt before $t_{RetryShutdown\_max}$ expires.

Figure 6 — CTC_2.3.1-6 - Timeout suspend test

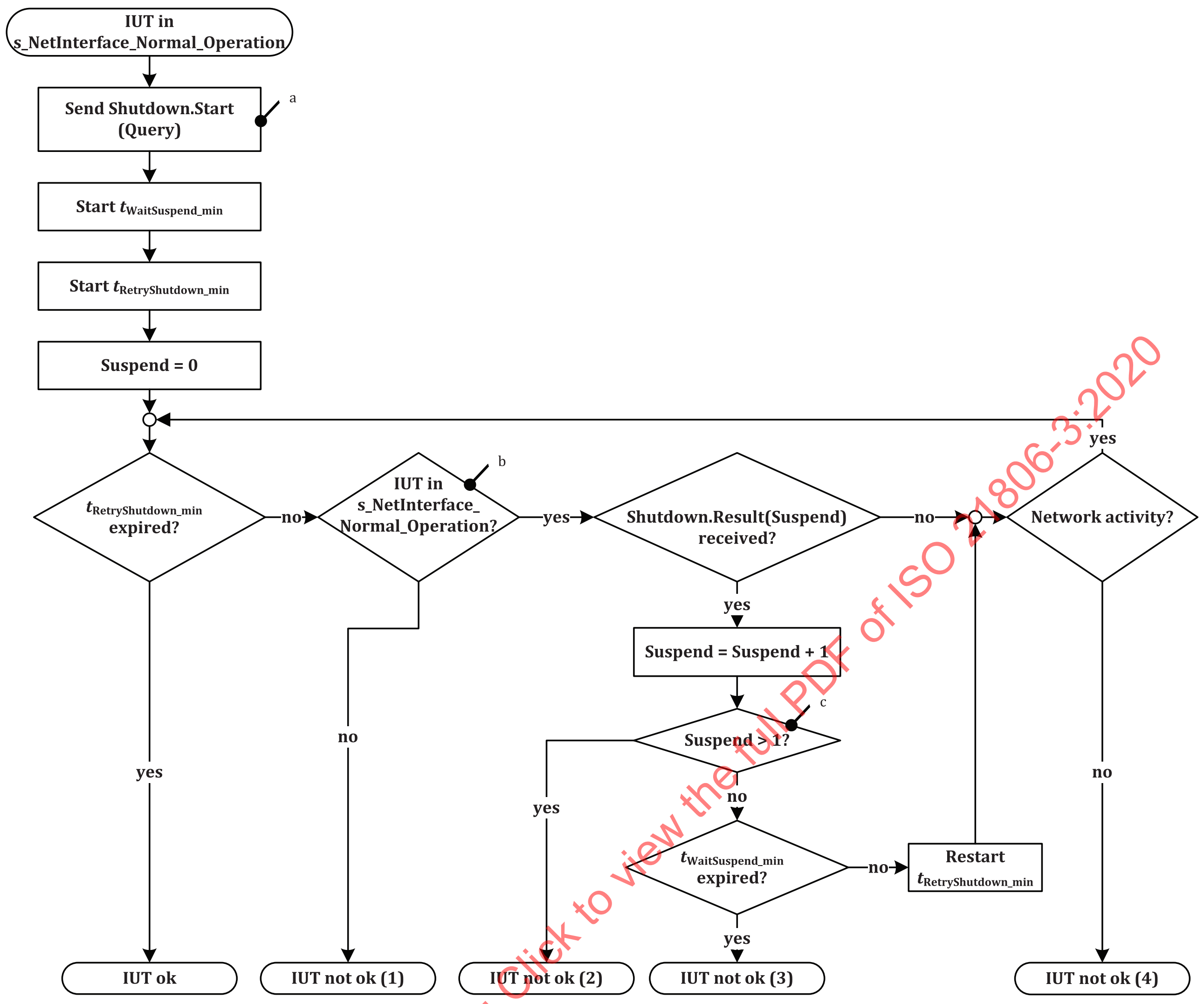
8.2.2 Power management – PowerSlave

8.2.2.1 CTC_2.3.2-2 – Shutdown.Start(Query) test

Table 13 specifies the CTC_2.3.2-2 – Shutdown.Start(Query) test.

Table 13 — CTC_2.3.2-2 – Shutdown.Start(Query) test

Item	Content
CTC # – Title	CTC_2.3.2-2 – Shutdown.Start(Query) test
Purpose	This CTC verifies that a PowerSlave does not switch off the MOST output due to reception of `Shutdown.Start(Query)`. The CTC also verifies that, if the PowerSlave attempts to suspend the shutdown, it does so within the specified time constraints. This CTC applies to all MOST devices that do not contain the PowerMaster.
Reference	ISO 21806-2:2020, 6.9.3 APP – Network shutdown
Prerequisite	The UT shall effectuate `s_NetInterface_Normal_Operation` in the node that contains the IUT.
Set-up	— If the IUT is part of a MOST device that contains the TimingMaster, the LT pre-IUT shall be a TimingSlave. — If the IUT is part of a MOST device that does not contain the TimingMaster, the LT pre-IUT shall be the TimingMaster. — The LT post-IUT shall be in listen-only node.
Step	1. The UT shall execute the sequence as specified in Figure 7.
Iteration	Not applicable
Expected response	Step 1: IUT ok: the IUT remains in `s_NetInterface_Normal_Operation`. Step 1: IUT not ok (1): the IUT does not remain in `s_NetInterface_Normal_Operation` after `Shutdown.Start(Query)`. Step 1: IUT not ok (2): the IUT sends `Shutdown.Result(Suspend)` too often. Step 1: IUT not ok (3): the IUT sends `Shutdown.Result(Suspend)` too late. Step 1: IUT not ok (4): the IUT switches off the MOST output by itself.
Remark	If CTC ends with "IUT ok", continue with CTC_2.3.2-3 – CTC_2.3.2-3 – Shutdown.Start(Execute) test.


IUT in s_NetInterface_Normal_Operation
Send Shutdown.Start (Query)
a
Start $t_{\mathrm{WaitSuspend\_min}}$
Start $t_{\mathrm{RetryShutdown\_min}}$
Suspend = 0
$t_{\mathrm{RetryShutdown\_min}}$ expired?
no
IUT in s_NetInterface_ Normal_Operation?
b
yes
Shutdown.Result(Suspend) received?
no
Network activity?
yes
yes
Suspend = Suspend + 1
c
Suspend > 1?
no
yes
$t_{\mathrm{WaitSuspend\_min}}$ expired?
no
Restart $t_{\mathrm{RetryShutdown\_min}}$
yes
no
yes
no
IUT ok
IUT not ok (1)
IUT not ok (2)
IUT not ok (3)
IUT not ok (4)

a The UT, using the LT pre-IUT, shall send `Shutdown.Start(Query)`.

b The node that contains the IUT remains in `s_NetInterface_Normal_Operation`. `s_NetInterface_Normal_Operation` shall be checked every 500 ms.

c The IUT is allowed to send `Shutdown.Result(Suspend)` once. In that case, `Shutdown.Result(Suspend)` is sent within $t_{\mathrm{WaitSuspend\_min}}$ after `Shutdown.Start(Query)`.

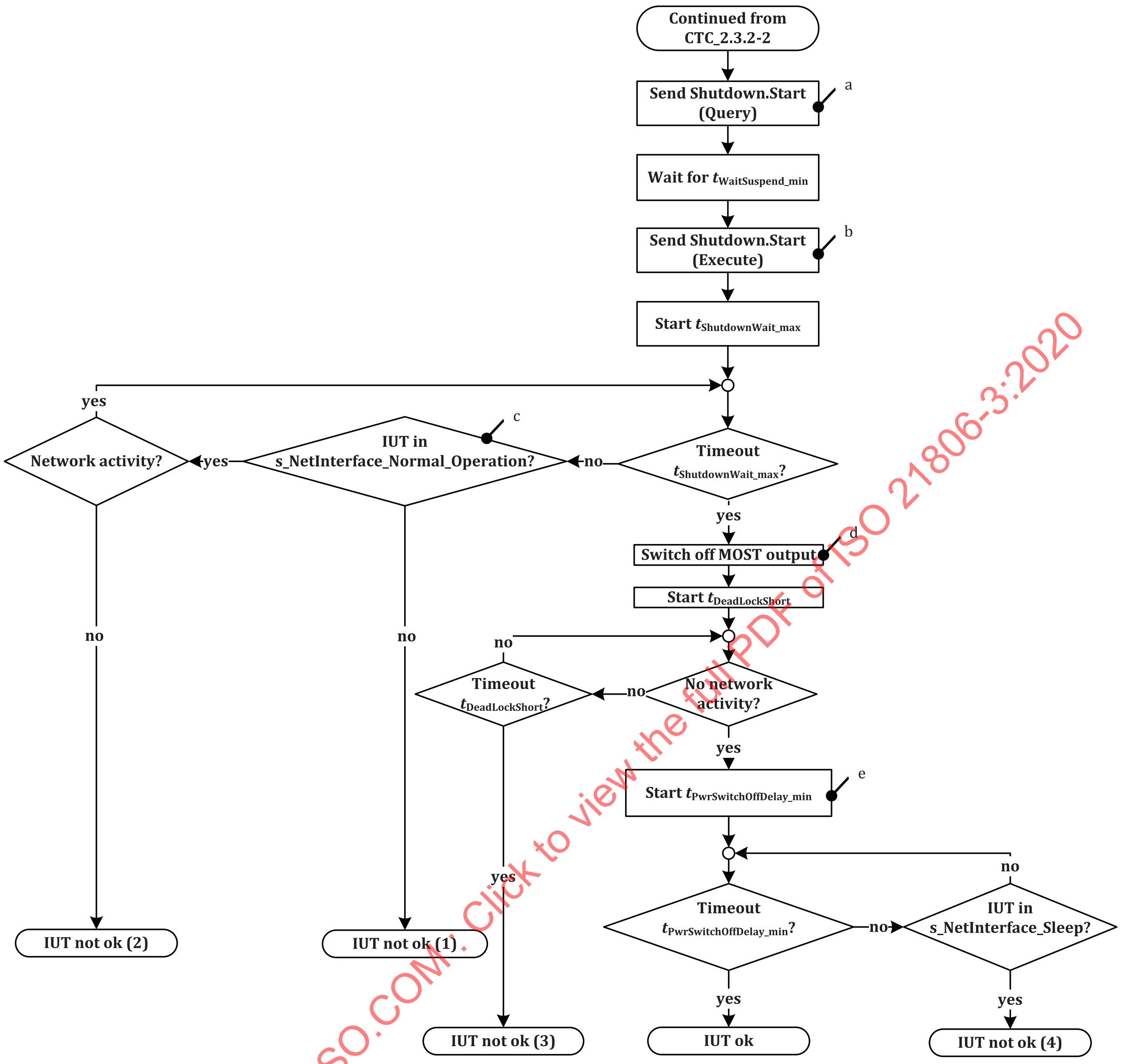
Figure 7 — CTC_2.3.2-2 - Shutdown.Start(Query) test

8.2.2.2 CTC_2.3.2-3 – Shutdown.Start(Execute) test

Table 14 specifies the CTC_2.3.2-3 – Shutdown.Start(Execute) test.

Table 14 — CTC_2.3.2-3 – Shutdown.Start(Execute) test

Item	Content
CTC # – Title	CTC_2.3.2-3 – Shutdown.Start(Execute) test
Purpose	This CTC verifies that a PowerSlave, after receiving `Shutdown.Start(Execute)`, remains in `s_NetInterface_Normal_Operation` until the LT pre-IUT switches off the MOST output. Additionally, it verifies that the IUT does not enter `s_NetInterface_Sleep` before $t_{\text{PwrSwitchOffDelay}}$ expires. This CTC applies to all MOST devices that do not contain the PowerMaster.
Reference	ISO 21806-2:2020, 6.9.3 APP – Network shutdown
Prerequisite	The CTC shall continue directly from CTC_2.3.2-2 – CTC_2.3.2-2 – Shutdown.Start(Query) test.
Set-up	— If the IUT is part of a MOST device that contains the TimingMaster, the LT pre-IUT shall be a TimingSlave. — If the IUT is part of a MOST device that does not contain the TimingMaster, the LT pre-IUT shall be the TimingMaster. — The LT post-IUT shall be a listen-only node. — Power consumption of the IUT is monitored.
Step	1. The UT shall execute the sequence as specified in Figure 8.
Iteration	Not applicable
Expected response	Step 1: IUT ok: the IUT remains in `s_NetInterface_Normal_Operation`. Step 1: IUT not ok (1): the IUT does not remain in `s_NetInterface_Normal_Operation` after reception of `Shutdown.Start(Execute)`. Step 1: IUT not ok (2): the IUT switches off the MOST output by itself too early. Step 1: IUT not ok (3): the IUT does not switch off the MOST output within $t_{\text{DeadLockShort}}$. Step 1: IUT not ok (4): the IUT enters `s_NetInterface_Sleep` too early.
Remark	---



Continued from CTC_2.3.2-2
Send Shutdown.Start (Query)
a
Wait for $t_{\mathrm{WaitSuspend\_min}}$
Send Shutdown.Start (Execute)
b
Start $t_{\mathrm{ShutdownWait\_max}}$
yes
Network activity?
yes
IUT in s_NetInterface_Normal_Operation?
c
no
Timeout $t_{\mathrm{ShutdownWait\_max}}$?
yes
Switch off MOST output
d
Start $t_{\mathrm{DeadLockShort}}$
no
no
no
Timeout $t_{\mathrm{DeadLockShort}}$?
no
No network activity?
yes
Start $t_{\mathrm{PwrSwitchOffDelay\_min}}$
e
no
yes
Timeout $t_{\mathrm{PwrSwitchOffDelay\_min}}$?
no
IUT in s_NetInterface_Sleep?
IUT not ok (2)
IUT not ok (1)
yes
yes
IUT not ok (3)
IUT ok
IUT not ok (4)

[a] The UT, using the LT pre-IUT, shall send `Shutdown.Start(Query)` and wait for $t_{\mathrm{WaitSuspend\_min}}$.

[b] The UT, using the LT pre-IUT, shall send `Shutdown.Start(Execute)`.

[c] The IUT prepares for shutdown. The node that contains the IUT remains in NetInterface state `s_NetInterface_Normal_Operation`. `s_NetInterface_Normal_Operation` shall be checked every 500 ms.

[d] Any kind of communication is possible until the LT pre-IUT switches off the MOST output. The IUT does not switch off the MOST before the LT pre-IUT does.

[e] When the LT pre-IUT switches off the MOST output, the node that contains the IUT does not enter `s_NetInterface_Sleep` before $t_{\mathrm{PwrSwitchOffDelay\_min}}$ is expired.

Figure 8 — CTC_2.3.2-3 - Shutdown.Start(Execute) test

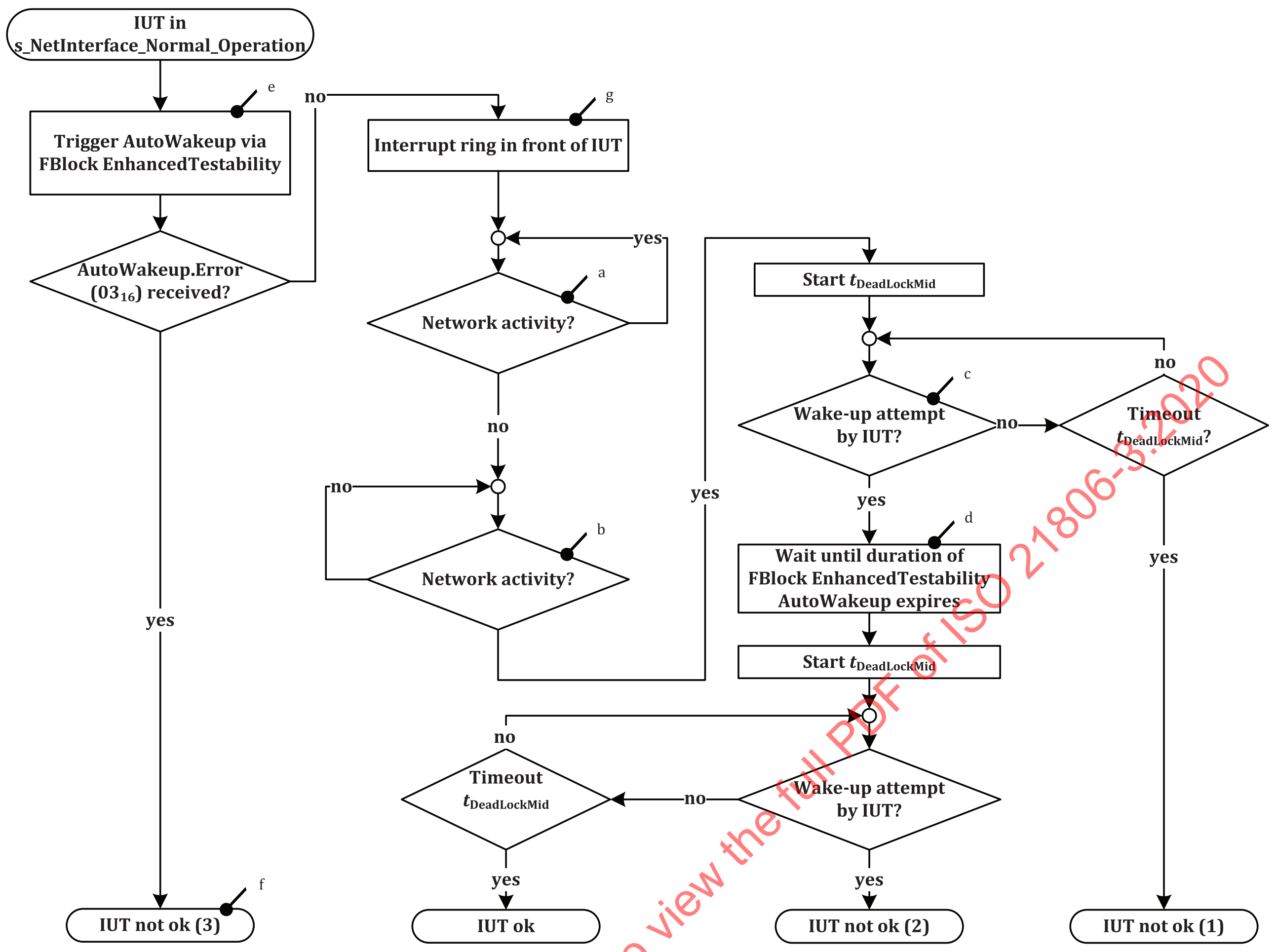
8.3 Error management

8.3.1 CTC_2.4.1-2 – Restart continue test

Table 15 specifies the CTC_2.4.1-2 – Restart continue test.

Table 15 — CTC_2.4.1-2 – Restart continue test

Item	Content
CTC # – Title	CTC_2.4.1-2 – Restart continue test
Purpose	This CTC verifies that the PowerMaster is capable of performing a network startup attempt. This CTC applies to all MOST devices that contain the PowerMaster and the TimingMaster in the same node and are capable of performing a network startup (i.e. switching on their MOST outputs).
Reference	ISO 21806-2:2020, 6.9.2 APP – Network wake-up and startup
Prerequisite	The UT shall effectuate `s_NetInterface_Normal_Operation` in the node that contains the IUT.
Set-up	— The LT pre-IUT shall be a TimingSlave. — The LT post-IUT shall be a listen-only node.
Step	1. The UT shall execute the sequence as specified in Figure 9.
Iteration	Not applicable
Expected response	Step 1: IUT ok: the IUT performs a wake-up attempt. Step 1: IUT not ok (1): the IUT does not perform at least one wake-up attempt. Step 1: IUT not ok (2): the IUT continues wake-up attempts without wake-up event. Step 1: IUT not ok (3): the IUT does not support wake-up.
Remark	---



[a] To avoid deadlocks, CTC shall be aborted when $t_{\text{DeadLockMid}}$ expires.

The IUT is expected to switch off the MOST output at the latest after $t_{\text{SSO\_Shutdown\_max}}$ expires within $t_{\text{Shutdown\_max}}$.

[b] To avoid deadlocks, CTC shall be aborted when $t_{\text{DeadLockMid}}$ expires.

The IUT is expected to switch on the MOST output after `DelayTime` of `AutoWakeup` and a manufacturer dependent delay.

[c] The IUT performs at least one wake-up attempt. The wake-up attempt shall consist of:

1) MOST output on (at 1st loop already on),

2) wait for t_{Config},

3) MOST output off, and

4) wait for t_{Restart}.

[d] `EnhancedTestability.AutoWakeup` is used to simulate the wake-up event. The CTC shall pause until the timeout for `AutoWakeup` (`Duration` = 15 s) occurs. When the qualified local wake-up event disappears, the IUT does not perform further wake-up attempts.

To provide sufficient time, the UT may wait in addition for $t_{\text{DeadLockMid}}$ to expire.

[e] The UT, using the LT pre-IUT, shall send `EnhancedTestability.AutoWakeup` to the IUT, using `Duration`= 15 s to simulate an extended local wake-up event. Use `EnhancedTestability.AutoWakeup` with the following parameters:

— `DelayTime` = 1 s;

— `Duration` = 15 s.

[f] IUT does not support wake-up.

[g] The LT pre-IUT shall interrupt the ring in front of the IUT.

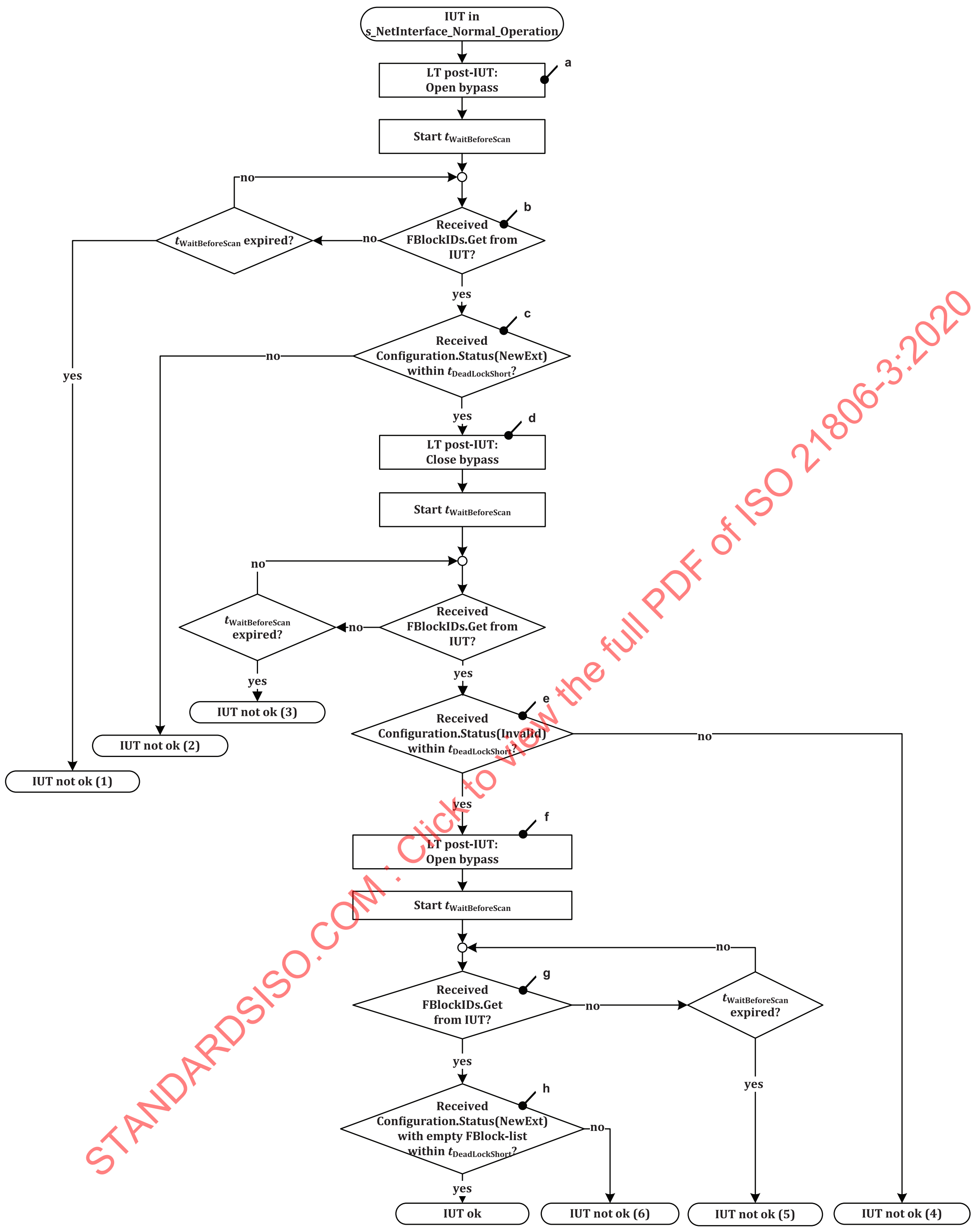
Figure 9 — CTC_2.4.1-2 - Restart continue test

8.3.2 CTC_2.4.1-9 – Reaction on network change event test

Table 16 specifies the CTC_2.4.1-9 – Reaction on network change event test.

Table 16 — CTC_2.4.1-9 – Reaction on network change event test

Item	Content
CTC # – Title	CTC_2.4.1-9 – Reaction on network change event test
Purpose	This CTC verifies that the NetworkMaster reacts correctly to NCEs by initiating an FBlock scan, announcing the central registry status and remaining within the specified time constraints. This CTC applies to all MOST devices that contain the NetworkMaster.
Reference	ISO 21806-2:2020, 6.8.3.2 APP – Network change event
Prerequisite	The UT shall effectuate `s_NetInterface_Normal_Operation` in the node that contains the IUT.
Set-up	— The LT pre-IUT shall be a TimingSlave. — The LT post-IUT shall be a TimingSlave with its bypass closed.
Step	1. The UT shall execute the sequence as specified in Figure 10.
Iteration	Not applicable
Expected response	Step 1: IUT ok: the IUT performs the expected actions after an NCE. Step 1: IUT not ok (1): the IUT does not check the FBlock configuration. Step 1: IUT not ok (2): the IUT does not send `Configuration.Status(NewExt)`. Step 1: IUT not ok (3): the IUT does not check the FBlock configuration. Step 1: IUT not ok (4): the IUT does not send `Configuration.Status(Invalid)`. Step 1: IUT not ok (5): the IUT does not check the FBlock configuration. Step 1: IUT not ok (6): the IUT does not send `Configuration.Status(NewExt)` with empty FBlock list.
Remark	When the NCE is generated, an unlock might occur.


IUT in s_NetInterface_Normal_Operation
LT post-IUT: Open bypass
a
Start $t_{WaitBeforeScan}$
no
b
Received FBlockIDs.Get from IUT?
no
$t_{WaitBeforeScan}$ expired?
yes
yes
c
Received Configuration.Status(NewExt) within $t_{DeadLockShort}$?
no
yes
d
LT post-IUT: Close bypass
Start $t_{WaitBeforeScan}$
no
Received FBlockIDs.Get from IUT?
no
$t_{WaitBeforeScan}$ expired?
yes
IUT not ok (3)
yes
e
Received Configuration.Status(Invalid) within $t_{DeadLockShort}$?
no
IUT not ok (2)
IUT not ok (1)
yes
f
LT post-IUT: Open bypass
Start $t_{WaitBeforeScan}$
no
g
Received FBlockIDs.Get from IUT?
no
$t_{WaitBeforeScan}$ expired?
yes
yes
h
Received Configuration.Status(NewExt) with empty FBlock-list within $t_{DeadLockShort}$?
no
yes
IUT ok
IUT not ok (6)
IUT not ok (5)
IUT not ok (4)

a The UT, using the LT post-IUT, shall open its bypass to generate an NCE.

b The IUT starts an FBlock scan within $t_{WaitBeforeScan}$. The UT shall respond to the `FBlockIDs.Get` request to LT post-IUT. The UT shall respond to every FBlock configuration check of the IUT addressed at the LT pre-IUT. The FBlock list shall differ from that reported for the LT post-IUT.

c The IUT sends `Configuration.Status(NewExt)`. The timer starts with the beginning of the FBlock configuration check.

d The LT post-IUT shall close its bypass. The IUT starts an FBlock scan within $t_{WaitBeforeScan}$.

[e] The IUT sends `Configuration.Status(Invalid)`. The timer starts with the beginning of the FBlock configuration check.

[f] The LT post-IUT shall generate an NCE without FBlock changes, that is, the LT post-IUT shall open its bypass and respond with an empty FBlock list to the FBlock scan of the IUT.

[g] The IUT starts an FBlock scan within $t_{\text{WaitBeforeScan}}$. The UT shall respond with empty `FBlockID` list to the `FBlockIDs.Get` request to LT post-IUT.

[h] The IUT sends `Configuration.Status(NewExt)` with an empty FBlock list. The timer starts with the beginning of the FBlock configuration check.

Figure 10 — CTC_2.4.1-9 - Reaction of network change event test

8.4 Central registry

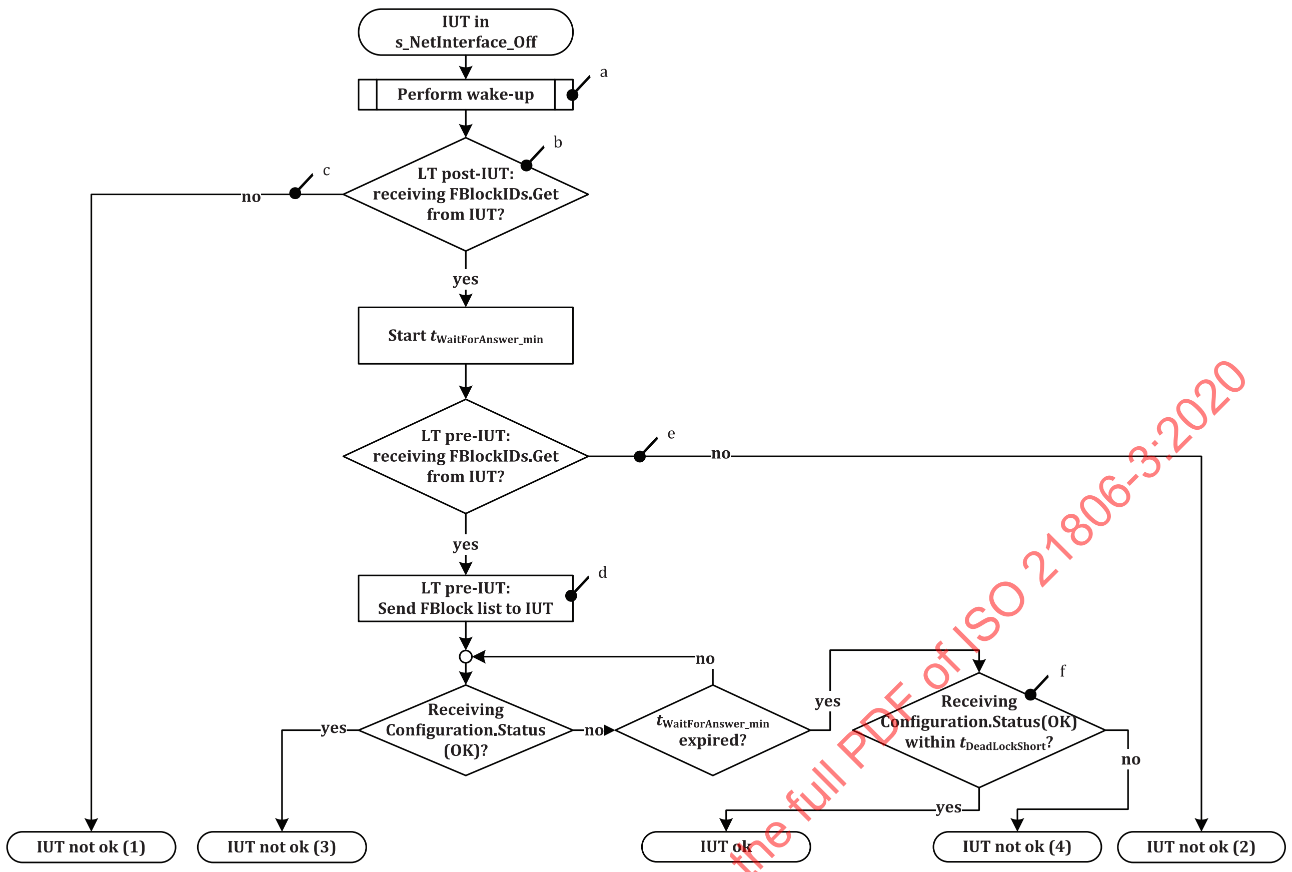
8.4.1 Central registry handling (NetworkMaster)

8.4.1.1 CTC_2.6.2-1 – FBlock polling test

Table 17 specifies the CTC_2.6.2-1 – FBlock polling test.

Table 17 — CTC_2.6.2-1 – FBlock polling test

Item	Content
CTC # – Title	CTC_2.6.2-1 – FBlock polling test
Purpose	This CTC verifies that the NetworkMaster requests the FBlock configuration from the NetworkSlaves, using `FBlockIDs.Get`. The CTC also verifies that the NetworkMaster distributes the central registry state and respects the relevant time constraints. This CTC applies to all MOST devices that contain the NetworkMaster.
Reference	ISO 21806-2:2020, 6.8.3.6.2 APP – Configuration request description
Prerequisite	— The UT shall effectuate `s_NetInterface_Off` in the node that contains the IUT. — The ring shall be closed.
Set-up	— The LT pre-IUT shall be a TimingSlave. — The LT post-IUT shall be a TimingSlave.
Step	1. The UT shall execute the sequence as specified in Figure 11.
Iteration	Not applicable
Expected response	Step 1: IUT ok: the IUT builds and distributes the central registry. Step 1: IUT not ok (1): the IUT does not check the FBlocks of the LT post-IUT. Step 1: IUT not ok (2): the IUT does not check the FBlocks of the LT pre-IUT. Step 1: IUT not ok (3): the IUT does not wait long enough for the response of the LT pre-IUT ($t_{\text{WaitForAnswer\_min}}$). Step 1: IUT not ok (4): the IUT does not send `Configuration.Status(OK)`.
Remark	---


IUT in s_NetInterface_Off
Perform wake-up
a
b
LT post-IUT: receiving FBlockIDs.Get from IUT?
c
no
yes
Start $t_{WaitForAnswer\_min}$
LT pre-IUT: receiving FBlockIDs.Get from IUT?
e
no
yes
LT pre-IUT: Send FBlock list to IUT
d
no
Receiving Configuration.Status (OK)?
yes
no
$t_{WaitForAnswer\_min}$ expired?
yes
f
Receiving Configuration.Status(OK) within $t_{DeadLockShort}$?
no
yes
IUT not ok (1)
IUT not ok (3)
IUT ok
IUT not ok (4)
IUT not ok (2)

[a] After wake-up, the IUT checks all FBlocks of the LT pre-IUT and of the LT post-IUT. If the IUT polls the LT pre-IUT before the LT post-IUT, the roles of the LT pre-IUT and the LT post-IUT shall be swapped.

[b] When the IUT queries the LT post-IUT, the UT shall start $t_{\mathrm{WaitForAnswer\_min}}$. For the LT post-IUT, the UT shall not respond to the FBlock scan.

[c] The UT shall stop the CTC if the IUT does not send `FBlockIDs.Get` within $t_{\mathrm{DeadLockShort}}$.

[d] For the LT pre-IUT, the UT shall respond to the FBlock request by the IUT with an FBlock list.

[e] The UT shall stop the CTC if the IUT does not send `FBlockIDs.Get` within $t_{\mathrm{DeadLockShort}}$.

[f] When the central registry is built up successfully, the IUT broadcasts `Configuration.Status(OK)` after $t_{\mathrm{WaitForAnswer\_min}}$ expires.

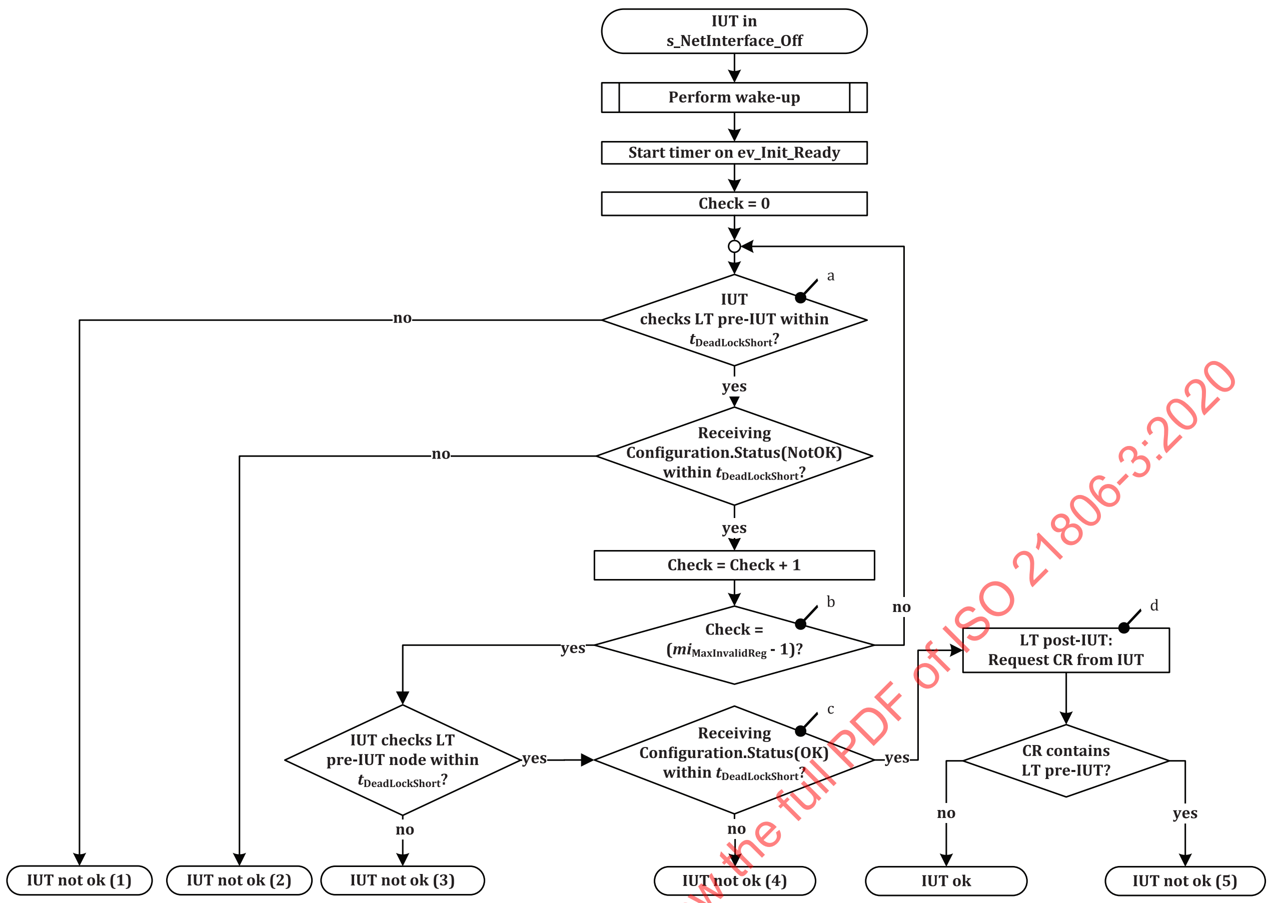
Figure 11 — CTC_2.6.2-1 - FBlock polling test

8.4.1.2 CTC_2.6.2-3a – Device ignore test (a)

Table 18 specifies the CTC_2.6.2-3a – Device ignore test (a).

Table 18 — CTC_2.6.2-3a – Device ignore test (a)

Item	Content
CTC # – Title	CTC_2.6.2-3a – Device ignore test (a)
Purpose	This CTC verifies that the NetworkMaster detects that a node does not have a valid logical node address, sets the central registry status to `NotOK`, attempts to gather FBlock information from that node up to the maximum number of invalid node registrations, and finally sets the central registry status to `OK`. This CTC applies to all MOST devices that contain the NetworkMaster.
Reference	ISO 21806-2:2020, 6.8.3.7 APP – Invalid announcements – Node address
Prerequisite	— The UT shall effectuate `s_NetInterface_Off` in the node that contains the IUT. — The ring shall be closed.
Set-up	— The LT pre-IUT shall be a TimingSlave with its logical node address set to `uninitialised_node_address`. — The LT post-IUT shall be a TimingSlave.
Step	1. The UT shall execute the sequence as specified in Figure 12.
Iteration	Not applicable
Expected response	Step 1: IUT ok: the IUT handles a node with an invalid logical node address correctly. Step 1: IUT not ok (1): the IUT does not check the LT pre-IUT. Step 1: IUT not ok (2): the IUT does not broadcast `Configuration.Status(NotOK)`. Step 1: IUT not ok (3): the IUT checks the LT pre-IUT less than $mi_{\text{MaxInvalidReg}}$ times. Step 1: IUT not ok (4): the IUT does not broadcast `Configuration.Status(OK)`. Step 1: IUT not ok (5): the IUT stores the LT pre-IUT into the central registry.
Remark	---


IUT in s_NetInterface_Off
Perform wake-up
Start timer on ev_Init_Ready
Check = 0
IUT checks LT pre-IUT within $t_{DeadLockShort}$?
a
no
yes
Receiving Configuration.Status(NotOK) within $t_{DeadLockShort}$?
Check = Check + 1
Check = ($mi_{MaxInvalidReg}$ - 1)?
b
IUT checks LT pre-IUT node within $t_{DeadLockShort}$?
Receiving Configuration.Status(OK) within $t_{DeadLockShort}$?
c
LT post-IUT: Request CR from IUT
d
CR contains LT pre-IUT?
IUT not ok (1)
IUT not ok (2)
IUT not ok (3)
IUT not ok (4)
IUT ok
IUT not ok (5)

[a] After wake-up, the IUT sends `FBlockIDs.Get`. The UT, using the LT pre-IUT, shall respond with `uninitialised_node_address`. The IUT detects that the LT pre-IUT has an `uninitialised_node_address` and broadcasts `Configuration.Status(NotOK)`.

[b] The LT pre-IUT shall not change its address. The IUT checks the LT pre-IUT as often as defined in $mi_{\mathrm{MaxInvalidReg}}$.

[c] The IUT sends `Configuration.Status(OK)`. The IUT is allowed to send `Configuration.Status(NotOK)` or start a rescan before broadcasting `Configuration.Status(OK)`.

[d] The IUT ignores the LT pre-IUT until the next NCE or network startup. The IUT does not store the address of the LT pre-IUT in the central registry. The UT, using the LT post-IUT, shall respond with a valid address. It requests the central registry from the IUT at end of the CTC.

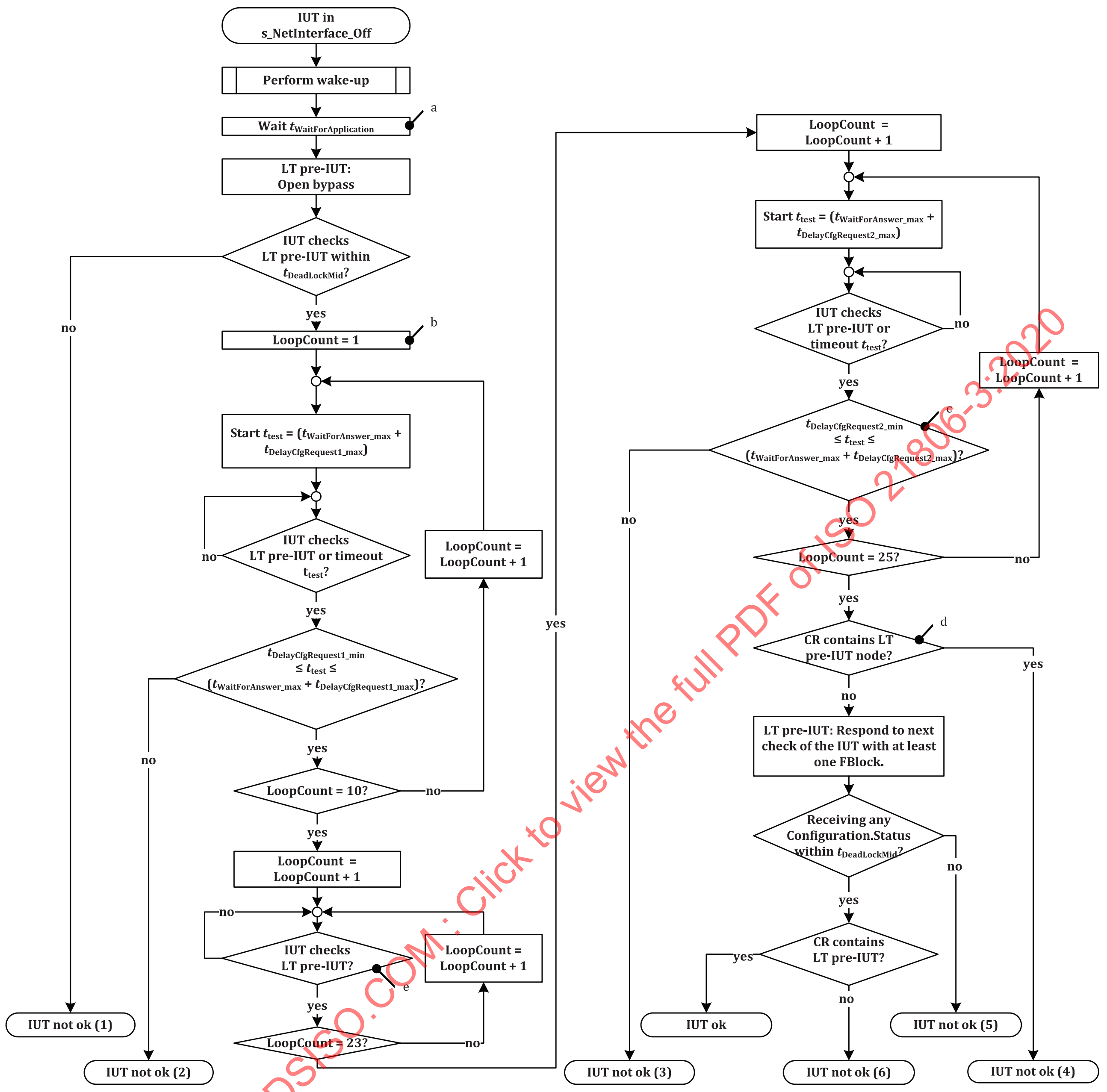
Figure 12 — CTC_2.6.2-3a - Device ignore test (a)

8.4.1.3 CTC_2.6.2-3b – Device ignore test (b)

Table 19 specifies the CTC_2.6.2-3b – Device ignore test (b).

Table 19 — CTC_2.6.2-3b – Device ignore test (b)

Item	Content
CTC # – Title	CTC_2.6.2-3b – Device ignore test (b)
Purpose	This CTC verifies that the NetworkMaster uses $t_{\texttt{DelayCfgRequest1}}$ or $t_{\texttt{DelayCfgRequest2}}$ to introduce pauses between requests to non-responding nodes. The CTC also verifies that the NetworkMaster adds the node to the central registry when it finally answers. This CTC applies to all MOST devices that contain the NetworkMaster.
Reference	ISO 21806-2:2020, 6.8.3.6.6 APP – Non-responding NetworkSlaves
Prerequisite	— The UT shall effectuate `s_NetInterface_Off` in the node that contains the IUT. — The ring shall be closed.
Set-up	— The LT pre-IUT shall be a TimingSlave with its bypass closed. — The LT post-IUT shall be a TimingSlave.
Step	1. The UT shall execute the sequence as specified in Figure 13.
Iteration	Not applicable
Expected response	Step 1: IUT ok: the IUT handles the non-responding node correctly. Step 1: IUT not ok (1): the IUT does not check the LT pre-IUT. Step 1: IUT not ok (2): the IUT checks the LT pre-IUT not within $t_{\text{DelayCfgRequest1\_min/max}}$. Step 1: IUT not ok (3): the IUT checks the LT pre-IUT not within $t_{\text{DelayCfgRequest2\_min/max}}$. Step 1: IUT not ok (4): the IUT erroneously stores the LT pre-IUT into the central registry. Step 1: IUT not ok (5): the IUT does not detect delayed responding the LT pre-IUT. Step 1: IUT not ok (6): the IUT does not store the LT pre-IUT into the central registry.
Remark	Not responding to requests of the IUT means: 1. The UT acknowledges reception but does not respond to the request from the IUT (it does not send any status; the IUT waits for $t_{\text{WaitForAnswer\_min}}$; refer to CTC_2.6.2-1). 2. The CTC does not focus on the border between loop count 20 and 21. The CTC only checks whether the IUT changes from short check interval to long check interval.


IUT in s_NetInterface_Off
Perform wake-up
Wait $t_{WaitForApplication}$
a
LT pre-IUT: Open bypass
IUT checks LT pre-IUT within $t_{DeadLockMid}$?
no
yes
LoopCount = 1
b
Start t_{test} = ($t_{WaitForAnswer\_max}$ + $t_{DelayCfgRequest1\_max}$)
IUT checks LT pre-IUT or timeout t_{test}?
LoopCount = LoopCount + 1
$t_{DelayCfgRequest1\_min} \leq t_{test} \leq (t_{WaitForAnswer\_max} + t_{DelayCfgRequest1\_max})$?
LoopCount = 10?
LoopCount = LoopCount + 1
IUT checks LT pre-IUT?
e
LoopCount = 23?
IUT not ok (1)
IUT not ok (2)
LoopCount = LoopCount + 1
Start t_{test} = ($t_{WaitForAnswer\_max}$ + $t_{DelayCfgRequest2\_max}$)
IUT checks LT pre-IUT or timeout t_{test}?
$t_{DelayCfgRequest2\_min} \leq t_{test} \leq (t_{WaitForAnswer\_max} + t_{DelayCfgRequest2\_max})$?
c
LoopCount = 25?
CR contains LT pre-IUT node?
d
LT pre-IUT: Respond to next check of the IUT with at least one FBlock.
Receiving any Configuration.Status within $t_{DeadLockMid}$?
CR contains LT pre-IUT?
IUT ok
IUT not ok (3)
IUT not ok (4)
IUT not ok (5)
IUT not ok (6)

[a] Manufacturer dependent.

[b] After wake-up, the IUT detects that the LT pre-IUT does not respond to its requests. The first check is performed and then the loop starts with '1' as the retries are counted.

The UT shall prevent potential shutdown during the CTC by using the LT post-IUT to send `Shutdown.Result(Suspend)` to the PowerMaster if it receives `Shutdown.Start(Query)`.

[c] The IUT does not stop checking the LT pre-IUT. Verify that the IUT checks the LT pre-IUT with a long check interval.

[d] The IUT does not add the LT pre-IUT to the central registry. The UT, using the LT pre-IUT, shall wait for `Configuration.Status(OK)` before checking the central registry.

After loop count 25, the UT, using the LT pre-IUT, shall respond to requests from the IUT. The IUT adds the LT pre-IUT to the central registry.

[e] Depending on the loop count, the IUT checks the LT pre-IUT with short interval ($t_{WaitForAnswer}$ + $t_{DelayCfgRequest1}$) or long interval ($t_{WaitForAnswer}$ + $t_{DelayCfgRequest2}$). To avoid deadlocks, the UT shall abort the CTC if $t_{DeadLockMid}$ expires without check.

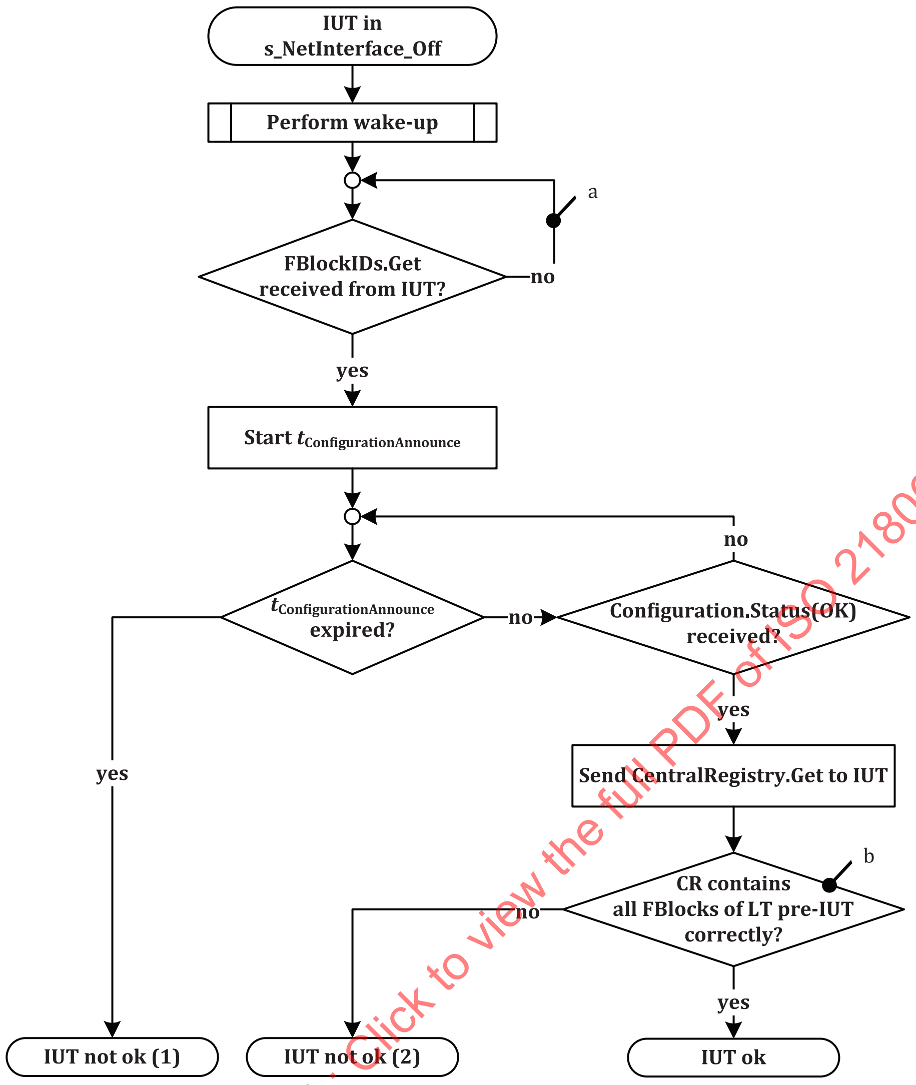
Figure 13 — CTC_2.6.2-3b - Device ignore test (b)

8.4.1.4 CTC_2.6.2-4a – Device integration test (a)

Table 20 specifies the CTC_2.6.2-4a – Device integration test (a).

Table 20 — CTC_2.6.2-4a – Device integration test (a)

Item	Content
CTC # – Title	CTC_2.6.2-4a – Device integration test (a)
Purpose	This CTC verifies that the NetworkMaster distributes the central registry state before $t_{\text{ConfigurationAnnounce}}$ expires. This CTC applies to all MOST devices that contain NetworkMaster.
Reference	ISO 21806-2:2020: — 6.8.3.6 APP – Scanning the MOST network; — 6.13.2.3 APP – Constraint $t_{\text{ConfigurationAnnounce}}$.
Prerequisite	— The UT shall effectuate `s_NetInterface_Off` in the node that contains the IUT. — The ring shall be closed.
Set-up	— The LT pre-IUT shall be a TimingSlave with valid address. It shall respond to `FBlockIDs.Get` from the IUT. — The LT post-IUT shall be a listen-only node.
Step	1. The UT shall execute the sequence as specified in Figure 14.
Iteration	Not applicable
Expected response	Step 1: IUT ok: the IUT builds the central registry, which contains the expected entry. Step 1: IUT not ok (1): the IUT does not send `Configuration.Status(OK)` within $t_{\text{ConfigurationAnnounce}}$. Step 1: IUT not ok (2): the IUT does not add the LT pre-IUT to the central registry.
Remark	---


IUT in s_NetInterface_Off
Perform wake-up
a
FBlockIDs.Get received from IUT?
no
yes
Start $t_{ConfigurationAnnounce}$
no
$t_{ConfigurationAnnounce}$ expired?
no
Configuration.Status(OK) received?
yes
yes
Send CentralRegistry.Get to IUT
b
CR contains all FBlocks of LT pre-IUT correctly?
no
yes
IUT not ok (1)
IUT not ok (2)
IUT ok

a After wake-up, the IUT adds the LT pre-IUT to the central registry and sends `Configuration.Status(OK)` within $t_{ConfigurationAnnounce}$. The IUT shall stop the CTC if the IUT does not send `FBlockIDs.Get` within $t_{DeadLockShort}$.

b The central registry shall contain all FBlocks associated with the address of the LT pre-IUT with correct IDs.

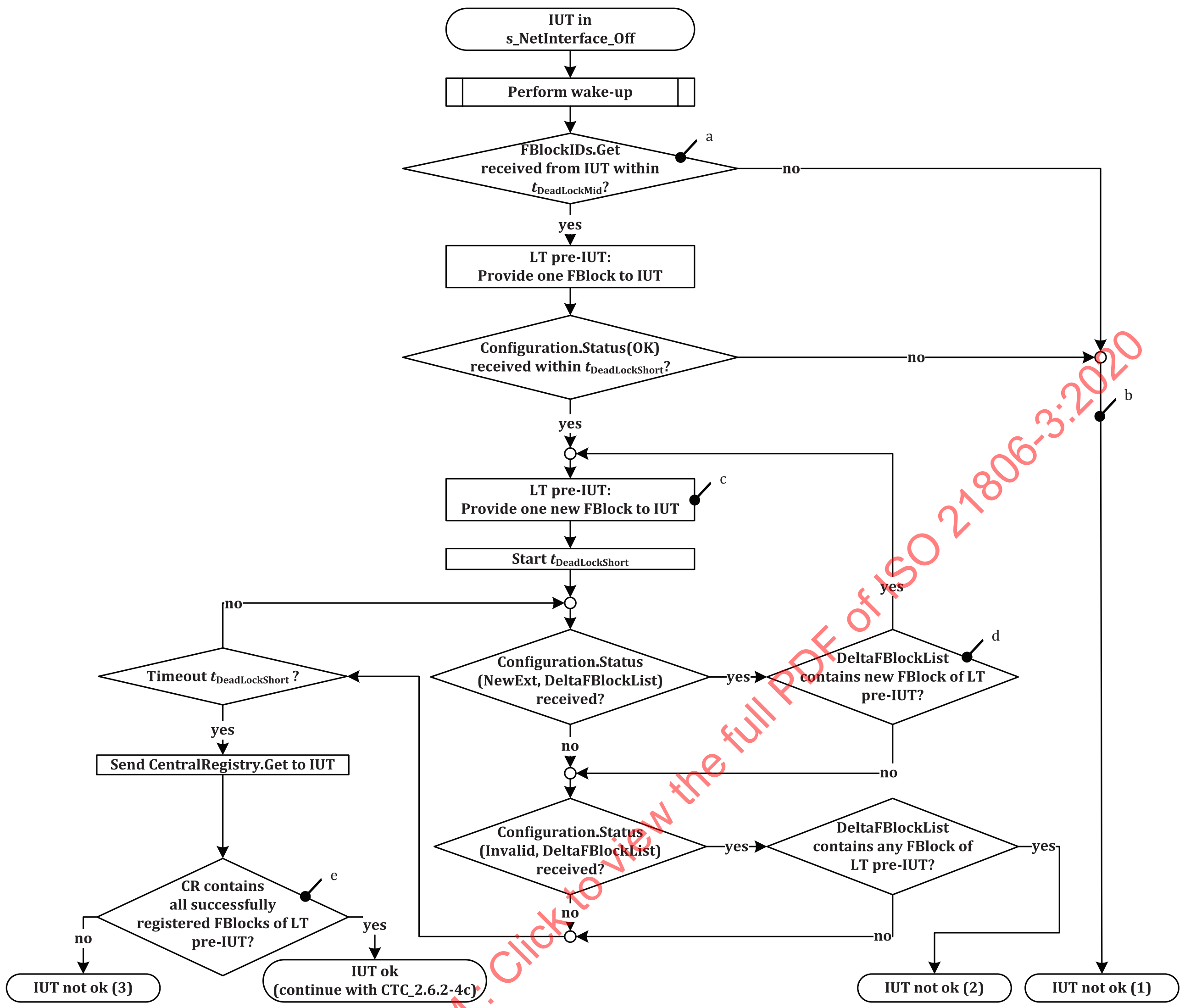
Figure 14 — CTC_2.6.2-4a - Device integration test (a)

8.4.1.5 CTC_2.6.2-4b – Device integration test (b)

Table 21 specifies the CTC_2.6.2-4b – Device integration test (b).

Table 21 — CTC_2.6.2-4b – Device integration test (b)

Item	Content
CTC # – Title	CTC_2.6.2-4b – Device integration test (b)
Purpose	This CTC verifies that the NetworkMaster stops adding FBlocks to the central registry when the central registry is full. This CTC applies to all MOST devices that contain the NetworkMaster.
Reference	ISO 21806-2:2020: — 6.8.2 APP – Central registry state; — 6.8.3.9.3 Appearing FBlocks in central registry state OK.
Prerequisite	— The UT shall effectuate `s_NetInterface_Off` in the node that contains the IUT. — The ring shall be closed.
Set-up	— The LT pre-IUT shall be a TimingSlave with valid address. — The LT post-IUT shall be a TimingSlave with valid address.
Step	1. The UT shall execute the sequence as specified in Figure 15.
Iteration	Not applicable
Expected response	Step 1: IUT ok: the IUT stops adding FBlocks when the central registry is full. Step 1: IUT not ok (1): the IUT does not perform initial scan and/or does not send `Configuration.Status(OK)` in time. Step 1: IUT not ok (2): the IUT deletes FBlocks associated with the LT pre-IUT from the central registry. Step 1: IUT not ok (3): the IUT does not store all successfully registered FBlocks associated with the LT pre-IUT in the central registry.
Remark	1. If the IUT supports dynamic central registry (detectable via FBlock EnhancedTestability), the LT pre-IUT and the LT post-IUT together have 257 FBlocks. 2. The UT is able to provide more FBlocks than the central registry of the IUT can store (detectable via FBlock `EnhancedTestability.CentralRegistrySize.Get`). 3. Potentially, the IUT adds own FBlocks to the central registry which reduces the available size of the central registry.



IUT in s_NetInterface_Off
Perform wake-up
FBlockIDs.Get received from IUT within $t_{DeadLockMid}$?
a
no
yes
LT pre-IUT: Provide one FBlock to IUT
Configuration.Status(OK) received within $t_{DeadLockShort}$?
no
yes
b
LT pre-IUT: Provide one new FBlock to IUT
c
Start $t_{DeadLockShort}$
no
Timeout $t_{DeadLockShort}$?
yes
Send CentralRegistry.Get to IUT
Configuration.Status (NewExt, DeltaFBlockList) received?
yes
DeltaFBlockList contains new FBlock of LT pre-IUT?
d
yes
no
no
Configuration.Status (Invalid, DeltaFBlockList) received?
yes
DeltaFBlockList contains any FBlock of LT pre-IUT?
yes
no
no
CR contains all successfully registered FBlocks of LT pre-IUT?
e
no
yes
IUT not ok (3)
IUT ok (continue with CTC_2.6.2-4c)
IUT not ok (2)
IUT not ok (1)

[a] The UT, using the LT post-IUT, shall respond with an empty FBlock list to any `FBlockIDs.Get` from the IUT during whole CTC. The UT, using the LT pre-IUT, shall provide more FBlocks than the IUT is able to store in the central registry.

[b] The UT shall stop the CTC if the IUT does not perform the initial scan correctly.

[c] To fill up the central registry, the UT shall provide step by step single FBlocks until the IUT stops broadcasting `ConfigurationStatus(NewExt, …)`. The UT, using the LT pre-IUT, shall send `FBlockIDs.Status(FBlockIDList)`.

The `FBlockIDList` shall contain all FBlocks provided before and one new FBlock.

[d] If the IUT sends `FBlockIDs.Get` to the LT pre-IUT during the CTC, the UT, using the LT pre-IUT, shall respond with an FBlock list that contains all FBlocks the LT pre-IUT indicates to the IUT at this time. This shall be done to prevent the IUT from deleting FBlocks associated with the LT pre-IUT from the central registry.

If the IUT tries to change the InstID of an FBlock associated with the LT pre-IUT, the UT shall execute that request according to ISO 21806-2.

[e] All FBlocks associated with the LT pre-IUT that have been accepted by the IUT are contained in the central registry at the end of the CTC. It shall not contain FBlocks associated with the LT pre-IUT that are not indicated by the IUT with `Configuration.Status(NewExt, DeltaFBlockList)`. Exception: The first FBlock provided by the UT, using the LT pre-IUT, during the initial scan is not announced with `Configuration.Status(NewExt, DeltaFBlockList)` but shall be stored in the central registry.

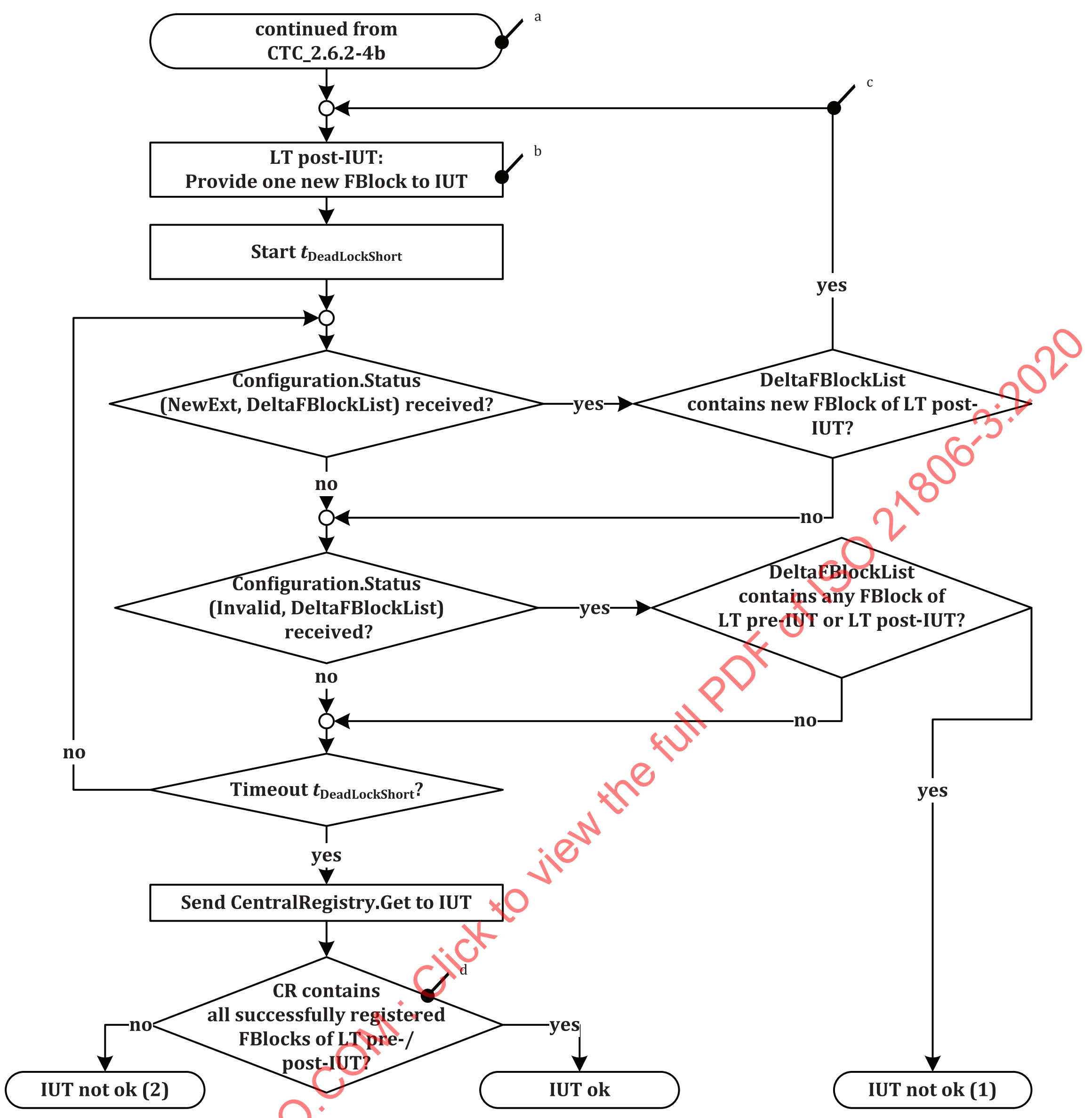
Figure 15 — CTC_2.6.2-4b - Device integration test (b)

8.4.1.6 CTC_2.6.2-4c – Device integration test (c)

Table 22 specifies the CTC_2.6.2-4c – Device integration test (c).

Table 22 — CTC_2.6.2-4c – Device integration test (c)

Item	Content
CTC # – Title	CTC_2.6.2-4c – Device integration test (c)
Purpose	This CTC verifies that the NetworkMaster, when the central registry is full, does not substitute entries which already exist in the central registry with recently announced FBlocks. This CTC applies to all MOST devices that contain the NetworkMaster.
Reference	ISO 21806-2:2020: — 6.8.2 APP – Central registry state; — 6.8.3.9.3 Appearing FBlocks in central registry state OK.
Prerequisite	Continued from 2.6.2-4b; “IUT ok”.
Set-up	— The LT pre-IUT shall be a TimingSlave with valid address. — The LT post-IUT shall be a TimingSlave with valid address.
Step	1. The UT shall execute the sequence as specified in Figure 16.
Iteration	Not applicable
Expected response	Step 1: IUT ok: the IUT does not replace FBlock entries in the central registry. Step 1: IUT not ok (1): the IUT deletes FBlocks associated with the LT pre-IUT or the LT post-IUT from the central registry. Step 1: IUT not ok (2): the IUT does not add all successfully registered FBlocks associated with the LT pre-IUT and the LT post-IUT to the central registry.
Remark	If the IUT supports dynamic central registry (detectable via FBlock EnhancedTestability), the LT pre-IUT and the LT post-IUT together have 257 FBlocks.


continued from CTC_2.6.2-4b
a
c
LT post-IUT: Provide one new FBlock to IUT
b
Start $t_{DeadLockShort}$
yes
Configuration.Status (NewExt, DeltaFBlockList) received?
yes
DeltaFBlockList contains new FBlock of LT post-IUT?
no
no
Configuration.Status (Invalid, DeltaFBlockList) received?
yes
DeltaFBlockList contains any FBlock of LT pre-IUT or LT post-IUT?
no
no
no
Timeout $t_{DeadLockShort}$?
yes
yes
Send CentralRegistry.Get to IUT
d
CR contains all successfully registered FBlocks of LT pre-/ post-IUT?
no
yes
IUT not ok (2)
IUT ok
IUT not ok (1)

[a] The CTC shall continue from CTC_2.6.2-4b; "IUT ok". The UT, using the LT post-IUT, shall be able to provide more FBlocks than the central registry of the IUT can store (detectable via FBlock `EnhancedTestability.CentralRegistrySize.Get`).

Potentially, the IUT adds own FBlocks to the central registry, which reduces the available size of the central registry.

[b] To fill up the central registry, the UT, using the LT post-IUT, shall provide single FBlocks until the IUT stops broadcasting `ConfigurationStatus(NewExt, …)`. If the IUT sends `FBlockIDs.Get` to the LT pre-IUT or the LT post-IUT during the CTC, the UT shall respond with the same FBlock list as before for the LT pre-IUT and the LT post-IUT. This prevents the IUT from deleting FBlocks associated with the LT pre-IUT from the central registry.

[c] The IUT adds all FBlocks associated with the LT post-IUT that the IUT accepts to the central registry. Additionally, the IUT adds all FBlocks associated with the LT pre-IUT (based on CTC_2.6.2-4b) to the central registry.

The UT, using the LT post-IUT, shall send `FBlockIDs.Status(FBlockIDList)`. The `FBlockIDList` shall contain all FBlocks provided before and one new FBlock. If the IUT tries to change the InstID of an FBlock associated with the LT pre-IUT or the LT post-IUT, the UT shall execute that request according to ISO 21806-2.

[d] The central registry shall contain all FBlocks associated with the LT pre-IUT and the LT post-IUT that have been registered successfully. It shall not contain FBlocks associated with the LT pre-IUT or the LT post-IUT that are not indicated by the IUT with `Configuration.Status(NewExt, DeltaFBlockList)`.

Exception: The first FBlock provided by the UT, using the LT pre-IUT, during the initial scan is not announced with `Configuration.Status(NewExt, DeltaFBlockList)` but shall be stored in the central registry.

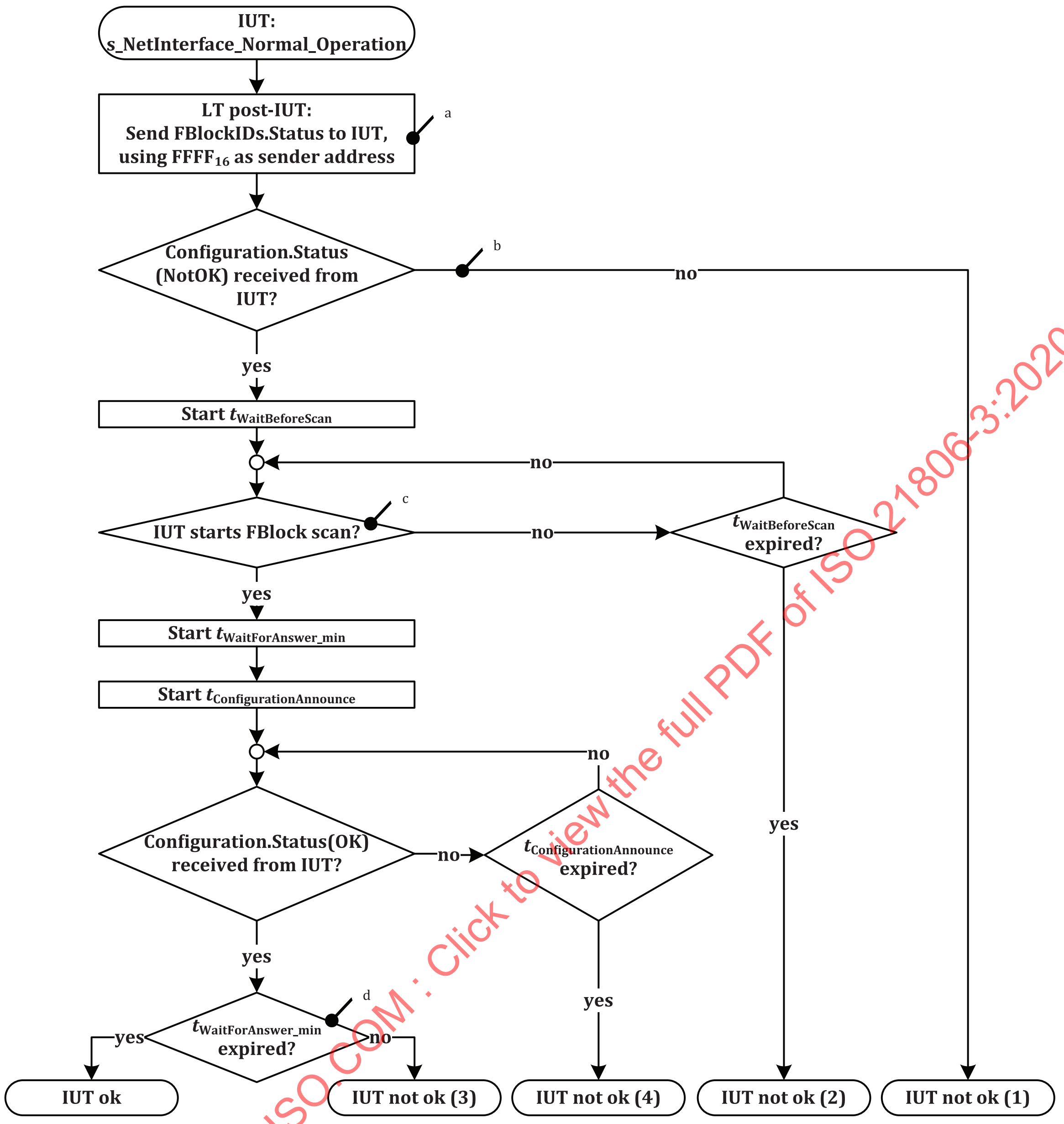
Figure 16 — CTC_2.6.2-4c - Device integration test (c)

8.4.1.7 CTC_2.6.2-5 – Config(OK) delay test

Table 23 specifies the CTC_2.6.2-5 – Config(OK) delay test.

Table 23 — CTC_2.6.2-5 – Config(OK) delay test

Item	Content
CTC # – Title	CTC_2.6.2-5 – Config(OK) delay test
Purpose	This CTC verifies that the NetworkMaster, after sending `Configuration.Status(NotOK)`, starts an FBlock scan before $t_{\text{WaitBeforeScan}}$ expires. This CTC also verifies that the NetworkMaster does not send `Configuration.Status(OK)` before $t_{\text{WaitForAnswer}}$ expires. This CTC applies to all MOST devices that contain the NetworkMaster.
Reference	ISO 21806-2:2020: — 6.8.3.3.3 APP – Setting the central registry state to NotOK; — 6.8.3.6.6 APP – Non-responding NetworkSlaves.
Prerequisite	The UT shall effectuate `s_NetInterface_Normal_Operation` in the node that contains the IUT.
Set-up	— The LT pre-IUT shall be a TimingSlave. — The LT post-IUT shall be a TimingSlave.
Step	1. The UT shall execute the sequence as specified in Figure 17.
Iteration	Not applicable
Expected response	Step 1: IUT ok: the IUT respects the timing constraints for announcing the central registry. Step 1: IUT not ok (1): the IUT does not send `Configuration.Status(NotOK)`. Step 1: IUT not ok (2): the IUT starts FBlock scan too late. Step 1: IUT not ok (3): the IUT sends `Configuration.Status(OK)` too early. Step 1: IUT not ok (4): the IUT does not send `Configuration.Status(OK)` in time.
Remark	---


IUT:
s_NetInterface_Normal_Operation
LT post-IUT:
Send FBlockIDs.Status to IUT,
using FFFF16 as sender address
a
Configuration.Status (NotOK) received from IUT?
b
no
yes
Start tWaitBeforeScan
no
IUT starts FBlock scan?
c
no
tWaitBeforeScan expired?
yes
Start tWaitForAnswer_min
Start tConfigurationAnnounce
no
Configuration.Status(OK) received from IUT?
no
tConfigurationAnnounce expired?
yes
yes
tWaitForAnswer_min expired?
d
yes
no
IUT ok
IUT not ok (3)
IUT not ok (4)
IUT not ok (2)
IUT not ok (1)

[a] The UT, using the LT post-IUT, shall send `FBlockIDs.Status` to the IUT, using sender address $\mathtt{FFFF}_{16}$.

[b] The IUT sends `Configuration.Status(NotOK)`. The UT shall stop the CTC if the IUT does not send `Configuration.Status(NotOK)` within $t_{\text{DeadLockShort}}$.

[c] The IUT starts an FBlock scan before $t_{\text{WaitBeforeScan}}$ expires. During this FBlock scan, the UT shall not respond to requests from the IUT to the LT post-IUT.

[d] The IUT does not send `Configuration.Status(OK)` before $t_{\text{WaitForAnswer\_min}}$ expires.

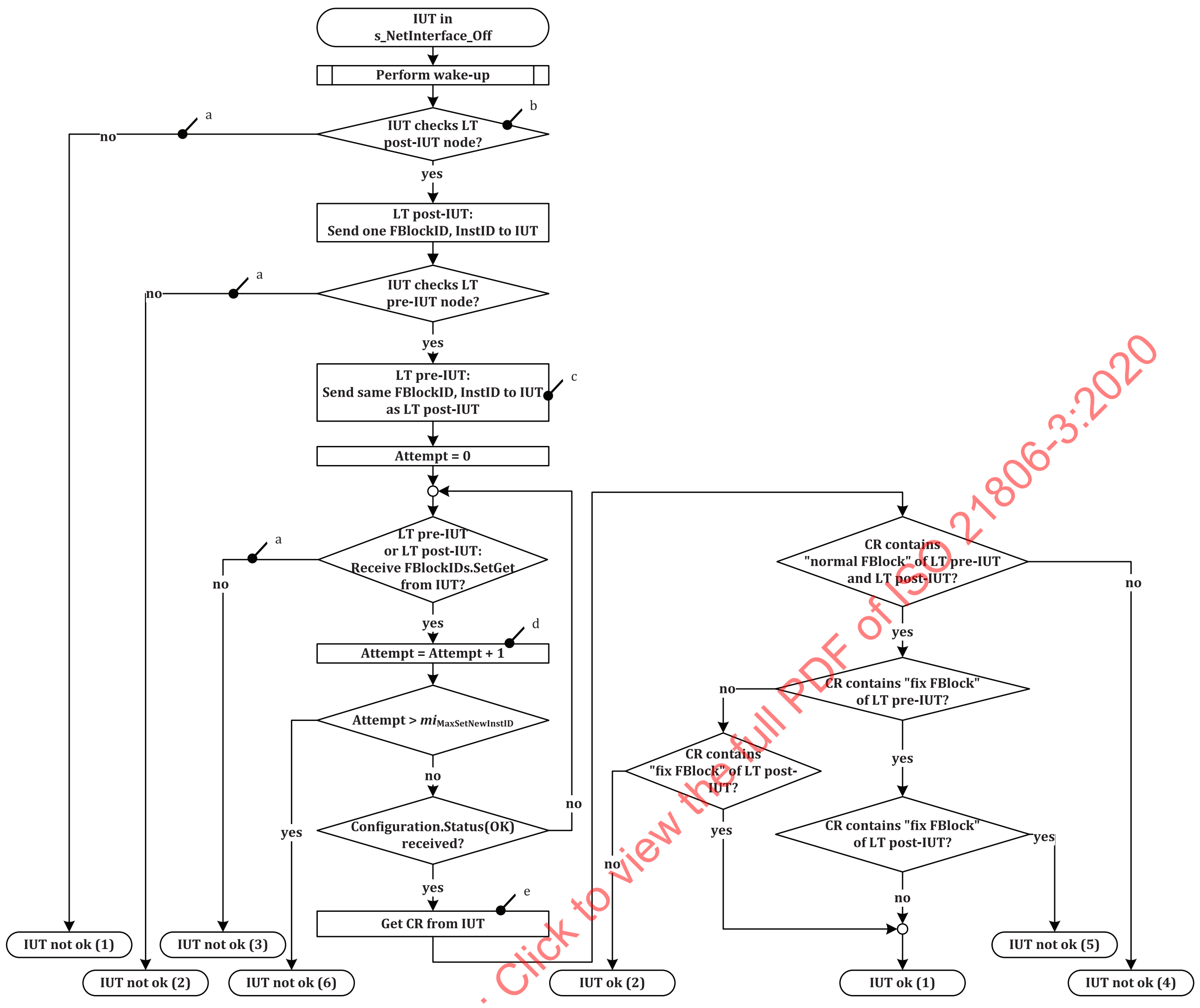
Figure 17 — CTC_2.6.2-5 - Config(OK) delay test

8.4.1.8 CTC_2.6.2-6 – Double FBlock test

Table 24 specifies the CTC_2.6.2-6 – Double FBlock test.

Table 24 — CTC_2.6.2-6 – Double FBlock test

Item	Content
CTC # – Title	CTC_2.6.2-6 – Double FBlock test
Purpose	This CTC verifies that the NetworkMaster detects duplicate `InstIDs`, attempts to correct those, and, if not successful, does not include the affected FBlocks in the central registry. This CTC applies to all MOST devices that contain the NetworkMaster.
Reference	ISO 21806-2:2020, 6.8.3.8 APP – Invalid announcements — InstID
Prerequisite	— The UT shall effectuate `s_NetInterface_Off` in the node that contains the IUT. — The ring shall be closed.
Set-up	— If the IUT is part of a MOST device that contains the TimingMaster, the LT pre-IUT shall be a TimingSlave. — If the IUT is part of a MOST device that does not contain the TimingMaster, the LT pre-IUT shall be the TimingMaster. — The LT post-IUT shall be a TimingSlave.
Step	1. The UT shall execute the sequence as specified in Figure 18.
Iteration	Not applicable
Expected response	Step 1: IUT ok (1): the IUT handles duplicate InstIDs correctly. Step 1: IUT ok (2): “fix FBlock” neither of the LT pre-IUT nor of the LT post-IUT stored in the central registry of the IUT. Step 1: IUT not ok (1): the IUT does not check the LT post-IUT. Step 1: IUT not ok (2): the IUT does not check the LT pre-IUT. Step 1: IUT not ok (3): the IUT does not change `FBlockID` of the LT pre-IUT or the LT post-IUT. Step 1: IUT not ok (4): the IUT does not store “normal FBlock” into the central registry. Step 1: IUT not ok (5): the IUT stores “fix FBlocks” of the LT pre-IUT and the LT post-IUT into the central registry. Step 1: IUT not ok (6): the IUT tries to change `FBlockIDs` of the LT pre-IUT or the LT post-IUT too often.
Remark	1. During the FBlock scan, the IUT might start checking the LT pre-IUT before the LT post-IUT responds. 2. The UT assigns 2 FBlocks to the LT pre-IUT and 2 FBlocks to the LT post-IUT. One with normal behaviour (named “normal FBlock”) that changes its `InstID` if commanded by the IUT and one FBlock which does not change its `InstID` if commanded (named “fix FBlock”). 3. For the “fix FBlocks”, the UT uses the same `FBlockID` and `InstID` for the LT pre-IUT and the LT post-IUT. For the “normal FBlock” the `FBlockIDs` and `InstIDs` differ.


IUT in s_NetInterface_Off
Perform wake-up
IUT checks LT post-IUT node?
no
yes
a
b
LT post-IUT: Send one FBlockID, InstID to IUT
IUT checks LT pre-IUT node?
LT pre-IUT: Send same FBlockID, InstID to IUT as LT post-IUT
c
Attempt = 0
LT pre-IUT or LT post-IUT: Receive FBlockIDs.SetGet from IUT?
d
Attempt = Attempt + 1
Attempt > $mi_{\text{MaxSetNewInstID}}$
Configuration.Status(OK) received?
e
Get CR from IUT
CR contains "normal FBlock" of LT pre-IUT and LT post-IUT?
CR contains "fix FBlock" of LT pre-IUT?
CR contains "fix FBlock" of LT post-IUT?
CR contains "fix FBlock" of LT post-IUT?
IUT not ok (1)
IUT not ok (2)
IUT not ok (3)
IUT not ok (6)
IUT ok (2)
IUT ok (1)
IUT not ok (5)
IUT not ok (4)

a The UT shall stop the CTC if the IUT does not send `FBlockIDs.Get` within $t_{\text{DeadLockShort}}$.

b After wake-up, the IUT performs an FBlock scan. It scans the MOST devices in correct order. Check of the LT post-IUT and the LT pre-IUT can be overlapped. The IUT can also start with checking the LT pre-IUT.

The UT, using the LT pre-IUT and the LT post-IUT, shall send the same `FBlockID`, `InstID` to the IUT.

c The UT, using the LT pre-IUT and the LT post-IUT, answers to requests to the associated NetBlocks without changing their FBlock configuration.

d The IUT detects the identical FBlocks and tries to change the `InstID` of the FBlock associated with the LT pre-IUT or the LT post-IUT. The UT shall not execute the requested `InstID` change.

e As soon as the IUT sends `Configuration.Status(OK)`, the UT shall request the central registry. The IUT provides a central registry that contains the FBlocks associated with the LT pre-IUT and the LT post-IUT. It does not contain both "fix FBlocks".

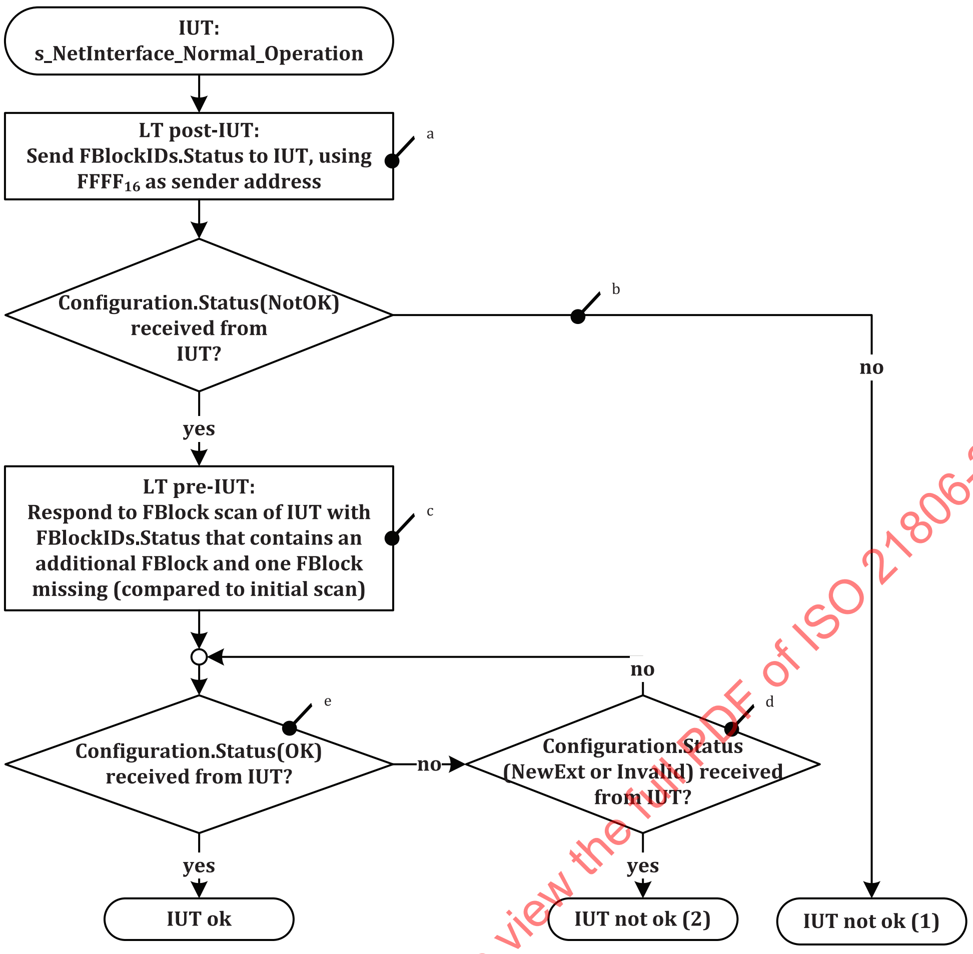
Figure 18 — CTC_2.6.2-6 - Double FBlock test

8.4.1.9 CTC_2.6.2-7 – Config(New) order test

Table 25 specifies the CTC_2.6.2-7 – Config(New) order test.

Table 25 — CTC_2.6.2-7 – Config(New) order test

Item	Content
CTC # – Title	CTC_2.6.2-7 – Config(New) order test
Purpose	This CTC verifies that the NetworkMaster sends `Configuration.Status(NotOK)` when encountering uninitialised node addresses and does not announce FBlocks with `NewExt`, `New`, or `Invalid` before sending an initial `Configuration.Status(OK)`. This CTC applies to all MOST devices that contain the NetworkMaster.
Reference	ISO 21806-2:2020, 6.8.3.3 APP – Setting the central registry state
Prerequisite	The UT shall effectuate `s_NetInterface_Normal_Operation` in the node that contains the IUT.
Set-up	— The LT pre-IUT shall be a TimingSlave. — The LT post-IUT shall be a TimingSlave.
Step	1. The UT shall execute the sequence as specified in Figure 19.
Iteration	Not applicable
Expected response	Step 1: IUT ok: the IUT reports central registry updates in the correct order. Step 1: IUT not ok (1): the IUT does not send `Configuration.Status(NotOK)`. Step 1: IUT not ok (2): the IUT sends `Configuration.Status(NewExt)` or `Configuration.Status(Invalid)` before sending `Configuration.Status(OK)`.
Remark	---


IUT:
s_NetInterface_Normal_Operation
LT post-IUT:
Send FBlockIDs.Status to IUT, using FFFF16 as sender address
a
Configuration.Status(NotOK) received from IUT?
b
no
yes
LT pre-IUT:
Respond to FBlock scan of IUT with FBlockIDs.Status that contains an additional FBlock and one FBlock missing (compared to initial scan)
c
no
e
d
Configuration.Status(OK) received from IUT?
no
Configuration.Status (NewExt or Invalid) received from IUT?
yes
yes
IUT ok
IUT not ok (2)
IUT not ok (1)

[a] The UT, using the LT post-IUT, shall send `FBlockIDs.Status` to the IUT with sender address FFFF$_{16}$.

[b] The IUT sends `Configuration.Status(NotOK)`. The UT shall stop the CTC if the IUT does not send `Configuration.Status(NotOK)` within $t_{\text{DeadLockShort}}$.

[c] The UT, using the LT pre-IUT, shall respond to FBlock scans within the period of time that the IUT waits for the answer of the LT post-IUT. The UT shall not respond to requests from the IUT to the LT post-IUT during the FBlock scan. During the FBlock scan, the UT, using the LT pre-IUT, shall indicate an additional FBlock and one missing FBlock (compared to the initial FBlock scan).

[d] The IUT does not broadcast `Configuration.Status(NewExt/New/Invalid)`. The UT shall stop the CTC if the IUT does not perform FBlock scan within $t_{\text{DeadLockMid}}$.

[e] The IUT broadcasts `Configuration.Status(OK)`.

Figure 19 — CTC_2.6.2-7 - Config(New) order test

8.4.1.10 CTC_2.6.3-1 – FBlock status change detection test

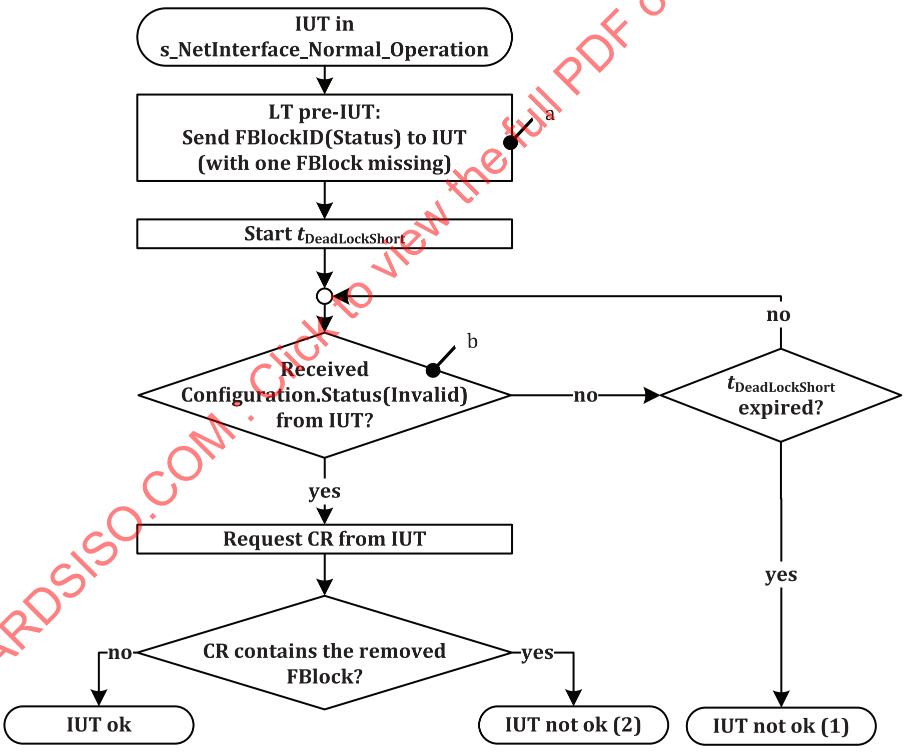
Table 26 specifies the CTC_2.6.3-1 – FBlock status change detection test.

Table 26 — CTC_2.6.3-1 – FBlock status change detection test

Item	Content
CTC # – Title	CTC_2.6.3-1 – FBlock status change detection test
Purpose	This CTC verifies that the NetworkMaster detects removed FBlocks in the FBlock list and re-acts by sending `Configuration.Status(Invalid)`. This CTC applies to all MOST devices that contain the NetworkMaster.

Table 26 *(continued)*

Item	Content
Reference	ISO 21806-2:2020, 6.8.3.9.2 APP – Disappearing FBlocks in central registry state OK
Prerequisite	— The UT shall effectuate `s_NetInterface_Normal_Operation` in the node that contains the IUT. — The ring shall be closed.
Set-up	— The LT pre-IUT shall be a TimingSlave with valid address. — The LT post-IUT shall be a listen-only node.
Step	1. The UT shall execute the sequence as specified in Figure 20.
Iteration	Not applicable
Expected response	Step 1: IUT ok: the IUT correctly handles removed FBlocks. Step 1: IUT not ok (1): the IUT does not broadcast `Configuration.Status(Invalid)`. Step 1: IUT not ok (2): the IUT does not delete unavailable FBlock from central registry.
Remark	---


IUT in s_NetInterface_Normal_Operation
LT pre-IUT: Send FBlockID(Status) to IUT (with one FBlock missing)
a
Start $t_{DeadLockShort}$
no
b
Received Configuration.Status(Invalid) from IUT?
no
$t_{DeadLockShort}$ expired?
yes
Request CR from IUT
yes
no
CR contains the removed FBlock?
yes
IUT ok
IUT not ok (2)
IUT not ok (1)

a The UT, using the LT pre-IUT, shall remove an FBlock from the central registry and indicate it to the IUT.

b The IUT broadcasts `Configuration.Status(Invalid)` and deletes the FBlock from the central registry.

Figure 20 — CTC_2.6.3-1 - FBlock status change detection test

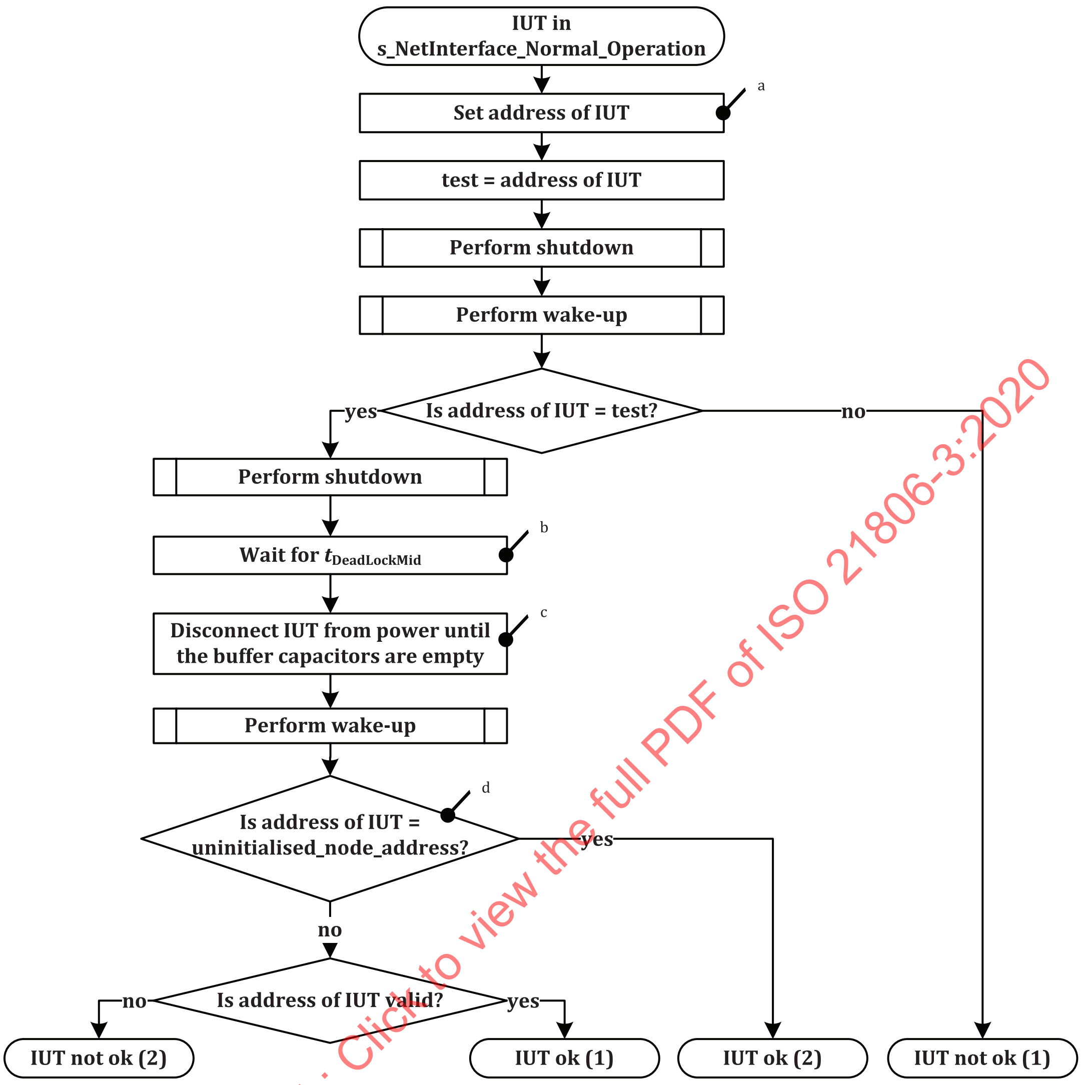
8.4.2 Central registry handling test (NetworkSlave)

8.4.2.1 CTC_2.6.4-1 – Address initialisation test

Table 27 specifies the CTC_2.6.4-1 – Address initialisation test.

Table 27 — CTC_2.6.4-1 – Address initialisation test

Item	Content
CTC # – Title	CTC_2.6.4-1 – Address initialisation test
Purpose	This CTC verifies that a NetworkSlave sets its logical node by request of the NetworkMaster and retains it until the NetworkSlave is disconnected from power. Afterwards, it is verified that the NetworkSlave has a valid node address or an uninitialised node address. This CTC applies to all MOST devices that do not contain the NetworkMaster.
Reference	ISO 21806-2:2020, 6.8.4.3 APP – Specific behaviour after ev_Init_Ready
Prerequisite	— The UT shall effectuate `s_NetInterface_Normal_Operation` in the node that contains the IUT. — The ring shall be closed.
Set-up	— The LT pre-IUT shall be the TimingMaster. — The LT post-IUT shall be a listen-only node.
Step	1. The UT shall execute the sequence as specified in Figure 21.
Iteration	Not applicable
Expected response	Step 1: IUT ok (1): the IUT uses a valid node address. Step 1: IUT ok (2): the IUT uses the `uninitialised_node_address` before deriving a new address from the node position. Step 1: IUT not ok (1): the IUT changes address between two operation cycles. Step 1: IUT not ok (2): the IUT neither uses `uninitialised_node_address` nor valid node address.
Remark	After startup, an address other than a valid node address (in static or dynamic address range) or `uninitialised_node_address` is not permitted for the IUT.


IUT in s_NetInterface_Normal_Operation
Set address of IUT
a
test = address of IUT
Perform shutdown
Perform wake-up
Is address of IUT = test?
yes
no
Perform shutdown
Wait for $t_{DeadLockMid}$
b
Disconnect IUT from power until the buffer capacitors are empty
c
Perform wake-up
Is address of IUT = uninitialised_node_address?
d
yes
no
Is address of IUT valid?
no
yes
IUT not ok (2)
IUT ok (1)
IUT ok (2)
IUT not ok (1)

[a] The UT, using the LT pre-IUT, shall set the address of the IUT to a value that is different from the default address of the node. It shall be a valid address. The case that the address cannot be set is tolerated. The UT shall store the address of the IUT.

[b] The node that contains the IUT is able to store its state into non-volatile memory before switching off the power supply.

If $t_{DeadLockMid}$ is not sufficient, the manufacturer shall provide $t_{PwrSwitchOffDelay\_max}$.

[c] Disconnect the IUT for a minimum of one minute from power.

[d] After shutdown, disconnecting from power (for at least 1 minute) and wake-up, the IUT uses a valid node address (in static or dynamic address range) or `uninitialised_node_address`.

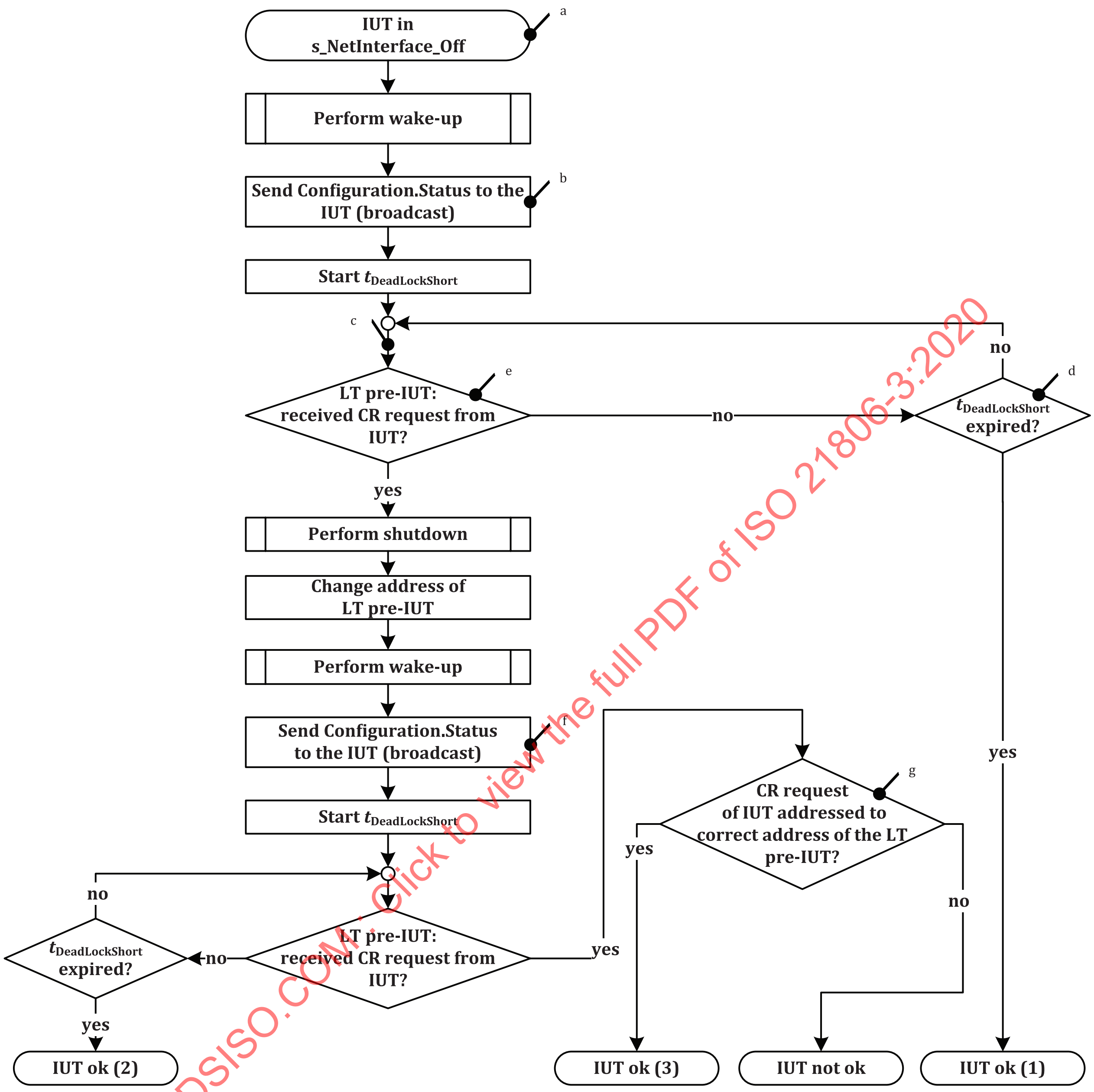
Figure 21 — CTC_2.6.4-1 - Address initialisation test

8.4.2.2 CTC_2.6.4-3 – NetworkMaster address storage test

Table 28 specifies the CTC_2.6.4-3 – NetworkMaster address storage test.

Table 28 — CTC_2.6.4-3 – NetworkMaster address storage test

Item	Content
CTC # – Title	CTC_2.6.4-3 – NetworkMaster address storage test
Purpose	This CTC verifies that a NetworkSlave correctly derives the address of the NetworkMaster from `Configuration.Status` messages. This CTC applies to all MOST devices that contain a NetworkSlave that communicates with the NetworkMaster during `s_NetInterface_Normal_Operation`.
Reference	ISO 21806-2:2020, 6.8.4.3.4 APP – Deriving the logical node address of the NetworkMaster
Prerequisite	— The UT shall effectuate `s_NetInterface_Off` in the node that contains the IUT. — The ring shall be closed.
Set-up	— If the IUT is part of a MOST device that contains the TimingMaster, the LT pre-IUT shall be a TimingSlave. — If the IUT is part of a MOST device that does not contain the TimingMaster, the LT pre-IUT shall be the TimingMaster. — The LT post-IUT shall be a listen-only node.
Step	1. The UT shall execute the sequence as specified in Figure 22.
Iteration	The CTC shall be performed three times, using — `Configuration.Status(OK)`, — `Configuration.Status(NewExt)` with empty FBlock list, and — `Configuration.Status(Invalid)`.
Expected response	Step 1: IUT ok (1): the IUT never requests the central registry. Step 1: IUT ok (2): the IUT does not request the central registry after shutdown and subsequent wake-up. Step 1: IUT ok (3): the IUT uses the correct address for the NetworkMaster. Step 1: IUT not ok: the IUT uses an incorrect address for the NetworkMaster.
Remark	In the case of `Invalid`, the `FBlockID C8`$_{16}$, `InstID 01`$_{16}$ is used.



IUT in s_NetInterface_Off
a
Perform wake-up
Send Configuration.Status to the IUT (broadcast)
b
Start $t_{DeadLockShort}$
c
LT pre-IUT: received CR request from IUT?
e
no
$t_{DeadLockShort}$ expired?
d
no
yes
yes
Perform shutdown
Change address of LT pre-IUT
Perform wake-up
Send Configuration.Status to the IUT (broadcast)
f
Start $t_{DeadLockShort}$
CR request of IUT addressed to correct address of the LT pre-IUT?
g
yes
no
no
$t_{DeadLockShort}$ expired?
no
LT pre-IUT: received CR request from IUT?
yes
yes
IUT ok (2)
IUT ok (3)
IUT not ok
IUT ok (1)

[a] The UT shall perform a wake-up. Between test loops, switch off the MOST output without sending `Shutdown.Start(Execute)`.

[b] After the node that contains the IUT is woken up, the UT, using the LT pre-IUT, shall broadcast `Configuration.Status`. The IUT derives the address of the LT pre-IUT. The CTC shall be performed three times:

— with sending `Configuration.Status(OK)`;

— with sending `Configuration.Status(NewExt)` with empty FBlock list;

— with sending `Configuration.Status(Invalid)` with FBlock not implemented in the IUT (use `FBlockID` $C8_{16}$, `InstID` 01_{16}).

[c] The UT, using the LT pre-IUT, shall trigger the IUT to send any message to the NetworkMaster (LT pre-IUT) to check whether the IUT has derived the address correctly. The IUT requests the central registry from the NetworkMaster automatically.

[d] If the IUT does not request the central registry within $t_{DeadLockShort}$, this function is not supported by the IUT.

[e] The request is addressed to the correct logical address of the LT pre-IUT. After shutdown, the UT shall change the address of the LT pre-IUT.

[f] Send same Configuration.Status as above.

[g] The IUT responds to the new address of the LT pre-IUT.

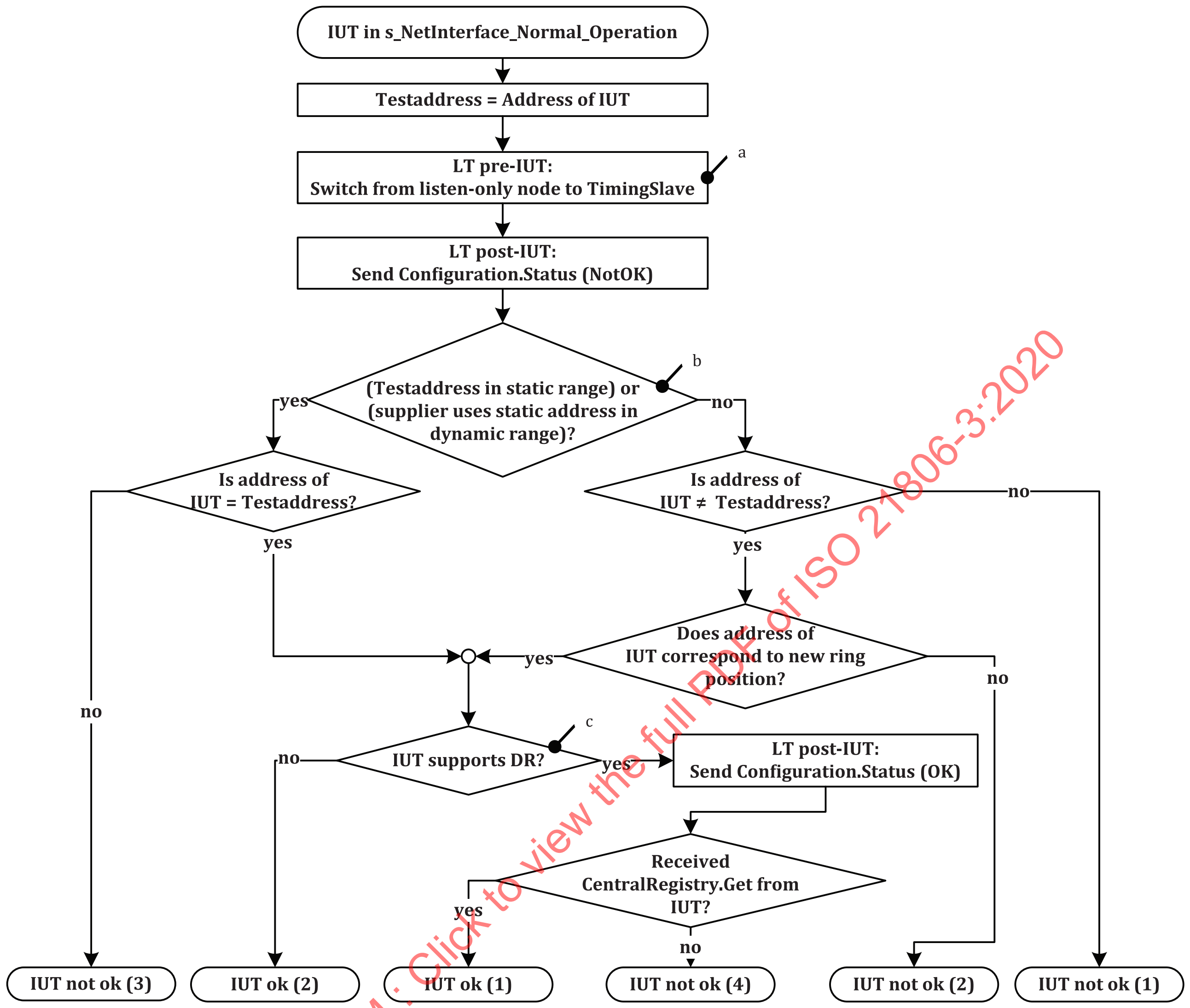
Figure 22 — CTC_2.6.4-3 - NetworkMaster address storage test

8.4.2.3 CTC_2.6.4-4 – Address reinitialisation test

Table 29 specifies the CTC_2.6.4-4 – Address reinitialisation test.

Table 29 — CTC_2.6.4-4 – Address reinitialisation test

Item	Content
CTC # – Title	CTC_2.6.4-4 – Address reinitialisation test
Purpose	This CTC verifies that a NetworkSlave is capable of changing its logical node address after an NCE and subsequently rebuilding its decentral registry. This CTC applies to all MOST devices that do not contain the NetworkMaster.
Reference	ISO 21806-2:2020: — 6.8.2.2 APP – Central registry state NotOK; — 7.2.2 AL – 16-bit addressing.
Prerequisite	— The UT shall effectuate `s_NetInterface_Normal_Operation` in the node that contains the IUT. — The ring shall be closed.
Set-up	— The LT pre-IUT shall be a listen-only node. — The LT post-IUT shall be the TimingMaster.
Step	1. The UT shall execute the sequence as specified in Figure 23.
Iteration	Not applicable
Expected response	Step 1: IUT ok (1): the IUT builds its decentral registry. Step 1: IUT ok (2): the IUT does not support a decentral registry. Step 1: IUT not ok (1): the IUT (dynamic address) does not change its address after receiving `Configuration.Status(NotOK)`. Step 1: IUT not ok (2): the IUT (dynamic address) does not derive new address from ring position. Step 1: IUT not ok (3): the IUT (not dynamic address) does not use previous address. Step 1: IUT not ok (4): the IUT does not delete/rebuild decentral registry (if supported).
Remark	---


IUT in s_NetInterface_Normal_Operation
Testaddress = Address of IUT
LT pre-IUT: Switch from listen-only node to TimingSlave
a
LT post-IUT: Send Configuration.Status (NotOK)
(Testaddress in static range) or (supplier uses static address in dynamic range)?
b
yes
no
Is address of IUT = Testaddress?
Is address of IUT ≠ Testaddress?
yes
no
yes
no
Does address of IUT correspond to new ring position?
yes
no
IUT supports DR?
c
no
yes
LT post-IUT: Send Configuration.Status (OK)
Received CentralRegistry.Get from IUT?
yes
no
IUT not ok (3)
IUT ok (2)
IUT ok (1)
IUT not ok (4)
IUT not ok (2)
IUT not ok (1)

[a] During runtime, the UT shall change the position of the node that contains the IUT (by connecting the previously disconnected MOST node in the LT pre-IUT to the network as TimingSlave). The UT, using the LT post-IUT, shall send `Configuration.Status(NotOK)`.

[b] If the address of the IUT is in the dynamic address range, it derives a new address from its ring position, else it continues using its previous address. Dynamic address range: 0100_{16} – $013F_{16}$.

[c] If the IUT does not request the central registry within $t_{\text{DeadLockShort}}$ after `Configuration.Status(OK)` (during network startup), the IUT does not support the decentral registry.

If the IUT implements a decentral registry, the UT, using the LT post-IUT, shall send `Configuration.Status(OK)`. The IUT sends `CentralRegistry(Get)` to the LT post-IUT to rebuild its decentral registry.

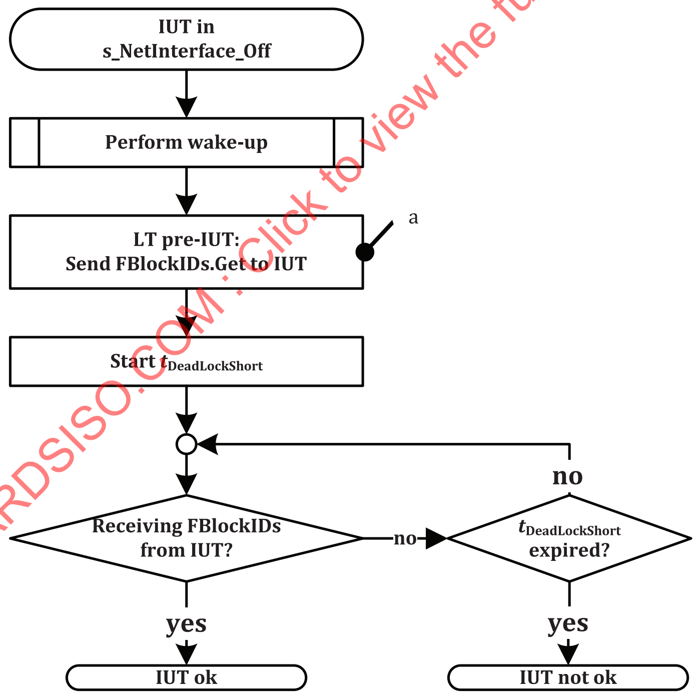
Figure 23 — CTC_2.6.4-4 - Address reinitialisation test

8.4.2.4 CTC_2.6.4-8 – FBlock response test

Table 30 specifies the CTC_2.6.4-8 – FBlock response test.

Table 30 — CTC_2.6.4-8 – FBlock response test

Item	Content
CTC # – Title	CTC_2.6.4-8 – FBlock response test
Purpose	This CTC verifies that a NetworkSlave correctly responds to an FBlock scan. This CTC applies to all MOST devices that do not contain the NetworkMaster.
Reference	ISO 21806-2:2020, 6.8.4.4.3 APP – Configuration requests and configuration changes
Prerequisite	— The UT shall effectuate `s_NetInterface_Off` in the node that contains the IUT. — The ring shall be closed.
Set-up	— The LT pre-IUT shall be a TimingSlave. — The LT post-IUT shall be a listen-only node.
Step	1. The UT shall execute the sequence as specified in Figure 24.
Iteration	Not applicable
Expected response	Step 1: IUT ok: the IUT correctly responds to an FBlock scan. Step 1: IUT not ok: the IUT does not respond with `FBlockIDs.Status`.
Remark	---


IUT in s_NetInterface_Off
Perform wake-up
LT pre-IUT: Send FBlockIDs.Get to IUT
a
Start $t_{DeadLockShort}$
Receiving FBlockIDs from IUT?
no
$t_{DeadLockShort}$ expired?
no
yes
yes
IUT ok
IUT not ok

[a] After wake-up, the UT, using the LT pre-IUT, shall send `FBlockIDs.Get` to the IUT after the IUT is in the ring (check NCE; timeout 5 s) and then wait $t_{WaitBeforeScan}$. The IUT responds with `FBlockIDs.Status`.

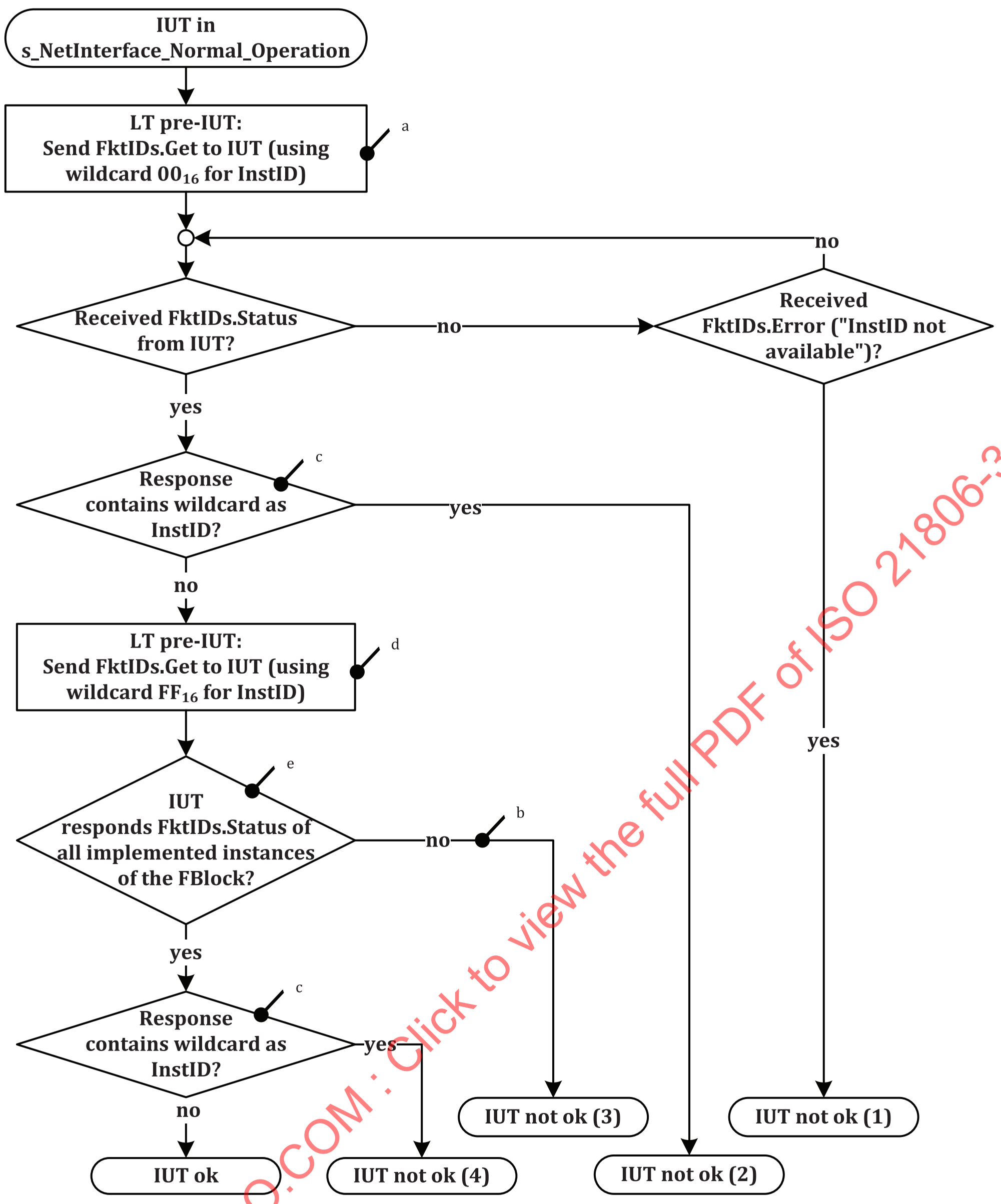
Figure 24 — CTC_2.6.4-8 - FBlock response test

8.4.3 CTC_2.6.4-10 – InstID wildcard test

Table 31 specifies the CTC_2.6.4-10 – InstID wildcard test.

Table 31 — CTC_2.6.4-10 – InstID wildcard test

Item	Content
CTC # – Title	CTC_2.6.4-10 – InstID wildcard test
Purpose	This CTC verifies that a node correctly recognizes `InstID` wildcards and responds accordingly. This CTC applies to all MOST devices.
Reference	ISO 21806-2:2020, 7.4.7 AL – Wildcard values for InstID
Prerequisite	The UT shall effectuate `s_NetInterface_Normal_Operation` in the node that contains the IUT.
Set-up	— If the IUT is part of a MOST device that contains the TimingMaster, the LT pre-IUT shall be a TimingSlave. — If the IUT is part of a MOST device that does not contain the TimingMaster, the LT pre-IUT shall be the TimingMaster. — The LT post-IUT shall be a listen-only node.
Step	1. The UT shall execute the sequence as specified in Figure 25.
Iteration	The CTC shall be performed with every single registered FBlock of the IUT.
Expected response	Step 1: IUT ok: the IUT correctly handles `InstID` wildcards. Step 1: IUT not ok (1): the IUT responds with error "`InstID` not available". Step 1: IUT not ok (2): the IUT uses wildcard for responding `FktIDs.Status`. Step 1: IUT not ok (3): the IUT does not respond with `FktIDs.Status` of all implemented instances of the current FBlock. Step 1: IUT not ok (4): the IUT uses wildcard for responding `FktIDs.Status`.
Remark	The UT sends `FBlockIDs.Get` with `InstID` = 00_{16} to request a list of all implemented instances of an FBlock from the IUT.


IUT in s_NetInterface_Normal_Operation
LT pre-IUT: Send FktIDs.Get to IUT (using wildcard 00_{16} for InstID)
a
Received FktIDs.Status from IUT?
no
Received FktIDs.Error ("InstID not available")?
no
yes
Response contains wildcard as InstID?
c
yes
no
LT pre-IUT: Send FktIDs.Get to IUT (using wildcard FF_{16} for InstID)
d
yes
IUT responds FktIDs.Status of all implemented instances of the FBlock?
e
no
b
yes
Response contains wildcard as InstID?
c
yes
no
IUT ok
IUT not ok (4)
IUT not ok (3)
IUT not ok (2)
IUT not ok (1)

[a] The UT, using the LT pre-IUT, shall send `FktIDs.Get` to the IUT, using wildcard 00_{16} for the `InstID`. The IUT responds with `FktIDs.Status`. The response of the IUT does not contain any wildcard as `InstID`.

[b] The UT shall stop the CTC if the IUT does not respond within $t_{DeadLockShort}$ or responds with an error.

[c] If the IUT is the NetworkMaster:

— the IUT responds with its own `InstID`;

— if this `InstID` is 00_{16}, the IUT is allowed to use 00_{16}.

[d] The UT, using the LT pre-IUT, shall send `FktIDs.Get` to the IUT, using wildcard FF_{16} for the `InstID`.

[e] The IUT responds with `FktIDs.Status` and reports all implemented instances of the current FBlock. The response of the IUT does not contain any wildcard as `InstID`. Implemented instances mean: All instances, indicated by the IUT via `FBlockIDs.Get`

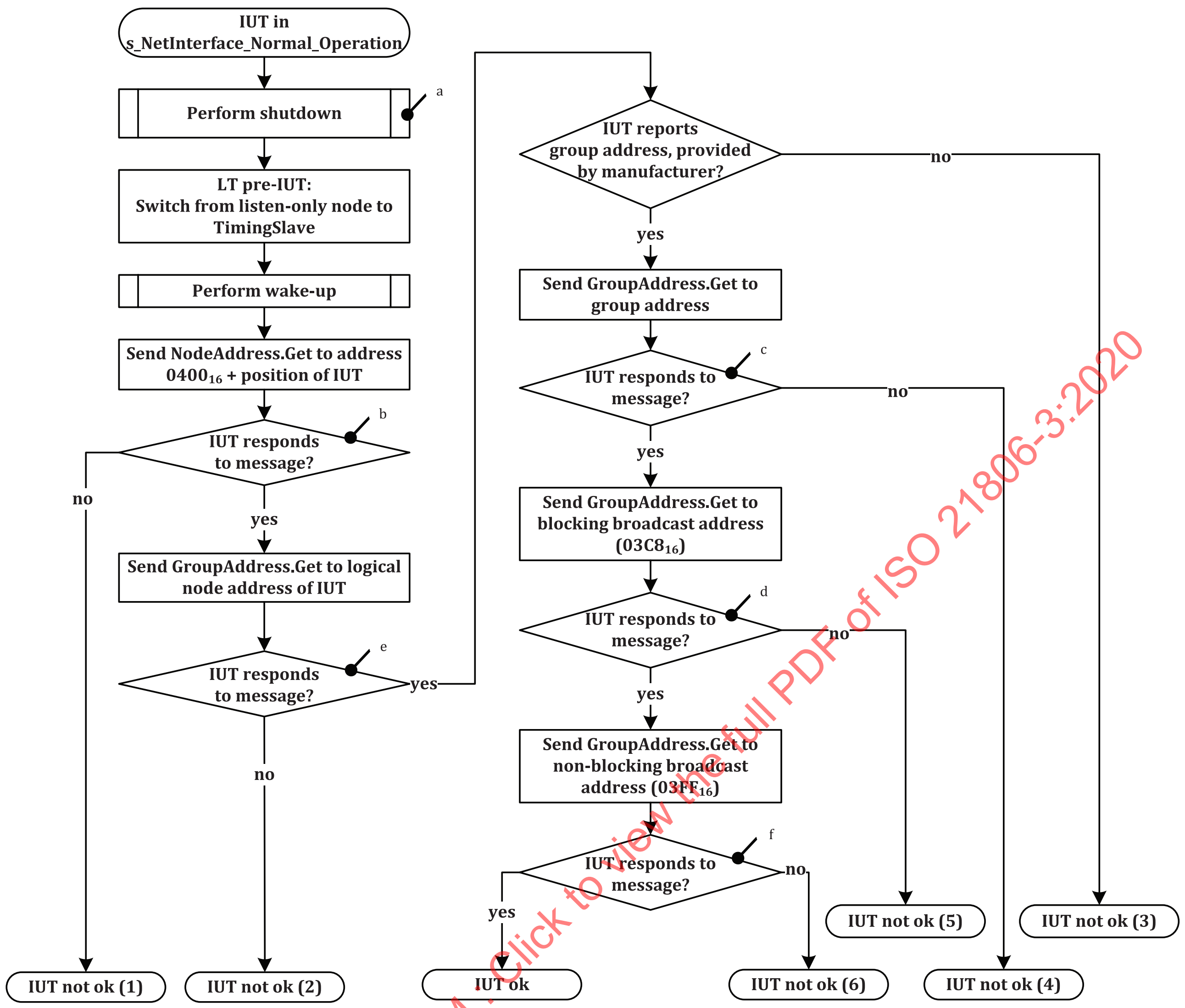
Figure 25 — CTC_2.6.4-10 - InstID wildcard test

8.5 CTC_2.7-1 – Node addressing test

Table 32 specifies the CTC_2.7-1 – Node addressing test.

Table 32 — CTC_2.7-1 – Node addressing test

Item	Content
CTC # – Title	CTC_2.7-1 – Node addressing test
Purpose	This CTC verifies that a NetworkSlave correctly sets different address types and reacts to messages that are sent to addresses it is associated with. This CTC applies to all MOST devices that do not contain the NetworkMaster.
Reference	ISO 21806-2:2020, 7.2.2 AL – 16-bit addressing
Prerequisite	— The UT shall effectuate `s_NetInterface_Normal_Operation` in the node that contains the IUT. — The ring shall be closed.
Set-up	— Depending on the test step, the LT pre-IUT shall be a listen-only node or a TimingSlave. — The LT post-IUT shall be the TimingMaster.
Step	1. The UT shall execute the sequence as specified in Figure 26.
Iteration	Not applicable
Expected response	Step 1: IUT ok: the IUT responds to messages to its associated addresses. Step 1: IUT not ok (1): the IUT does not respond to “node position addressing”. Step 1: IUT not ok (2): the IUT does not respond to “logical node addressing”. Step 1: IUT not ok (3): the IUT does not report the correct group address (differs from manufacturer information). Step 1: IUT not ok (4): the IUT does not respond to “group addressing”. Step 1: IUT not ok (5): the IUT does not respond to “blocking broadcast addressing”. Step 1: IUT not ok (6): the IUT does not respond to “non-broadcast addressing”.
Remark	---


IUT in s_NetInterface_Normal_Operation
Perform shutdown
a
LT pre-IUT: Switch from listen-only node to TimingSlave
Perform wake-up
Send NodeAddress.Get to address 0400$_{16}$ + position of IUT
IUT responds to message?
b
no
yes
Send GroupAddress.Get to logical node address of IUT
IUT responds to message?
e
yes
no
IUT reports group address, provided by manufacturer?
no
yes
Send GroupAddress.Get to group address
IUT responds to message?
c
no
yes
Send GroupAddress.Get to blocking broadcast address (03C8$_{16}$)
IUT responds to message?
d
no
yes
Send GroupAddress.Get to non-blocking broadcast address (03FF$_{16}$)
IUT responds to message?
f
yes
no
IUT not ok (1)
IUT not ok (2)
IUT ok
IUT not ok (6)
IUT not ok (4)
IUT not ok (5)
IUT not ok (3)

[a] The UT shall shutdown the network and change the position of the node that contains the IUT. Then, the UT shall start up the network. The IUT derives the new node position address from its new node position.

The UT shall send messages using the node position address, the logical node address, and the group address of the IUT, as well as the blocking broadcast address and the non-blocking broadcast address.

[b] The IUT sends a valid status message. The sender's address and the property `NodeAddress` are correct.

[c] The IUT sends a valid status message. The sender's address is correct.

[d] The IUT sends a valid status message. The sender's address is correct.

[e] The IUT sends a valid status message. The sender's address and the property `NodeAddress` are correct.

[f] The IUT sends a valid status message. The sender's address is correct.

Figure 26 — CTC_2.7-1 - Node addressing test

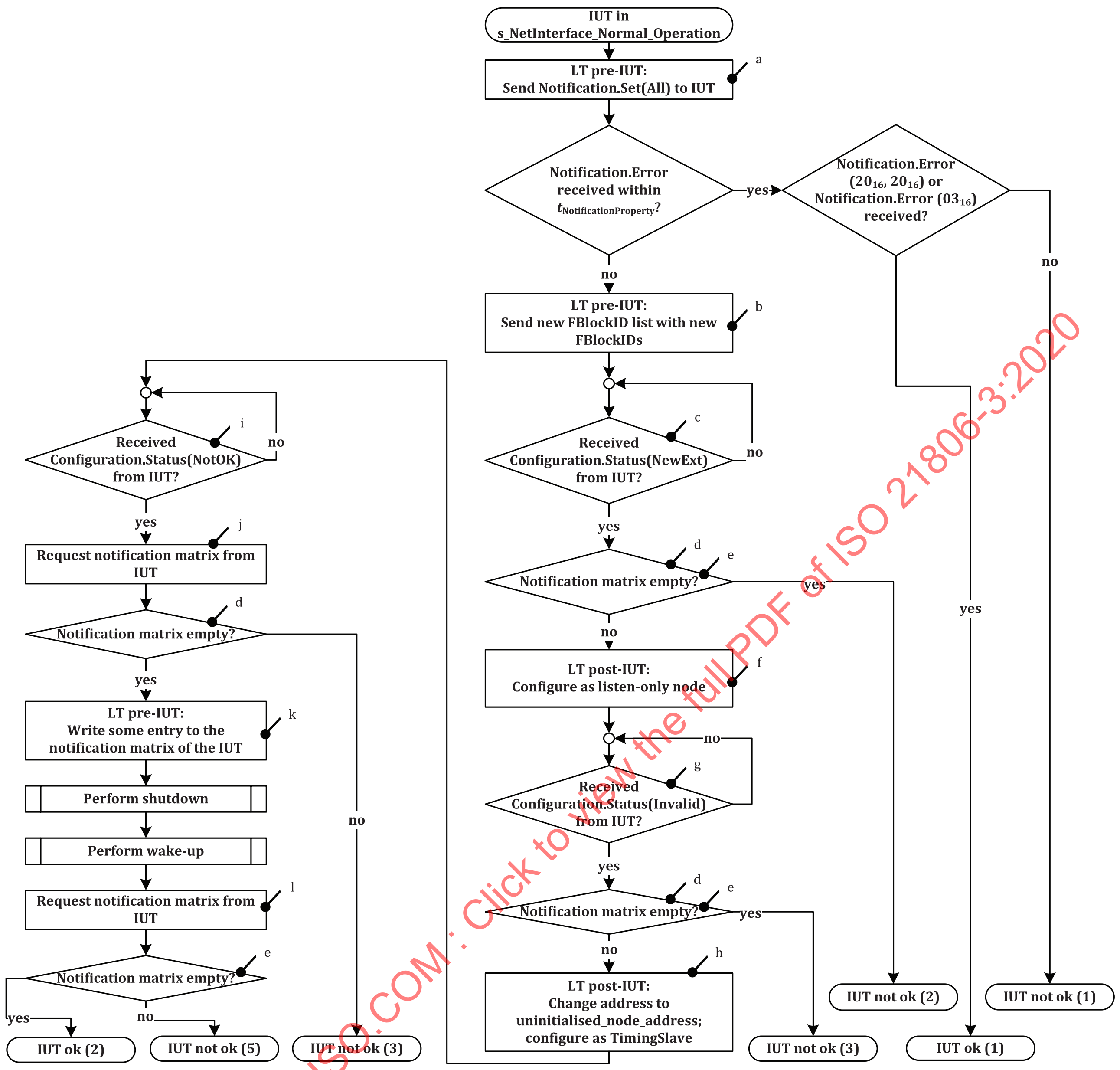
8.6 Notification matrix test

8.6.1 CTC_2.8.3-1a – Notification matrix storage test (NetworkMaster)

Table 33 specifies the CTC_2.8.3-1a – Notification matrix storage test (NetworkMaster).

Table 33 — CTC_2.8.3-1a – Notification matrix storage test (NetworkMaster)

Item	Content
CTC # – Title	CTC_2.8.3-1a – Notification matrix storage test (NetworkMaster)
Purpose	This CTC verifies that the NetworkMaster handles notifications correctly in conjunction with additions to and deletions from the central registry. This CTC applies to all MOST devices that contain the NetworkMaster.
Reference	ISO 21806-2:2020, 6.6 APP – Notification mechanisms
Prerequisite	— The UT shall effectuate `s_NetInterface_Normal_Operation` in the node that contains the IUT. — The ring shall be closed.
Set-up	— The LT pre-IUT shall be a TimingSlave. — The LT post-IUT is a listen-only node or TimingSlave, depending on the current test step.
Step	1. The UT shall execute the sequence as specified in Figure 27.
Iteration	The CTC shall be performed with every FBlock and function of the IUT that implements notifications, registered in the central registry except FBlock PowerMaster. All FBlocks shall be tested within one test loop (no separate test loop for every FBlock) to increase stress of the IUT.
Expected response	Step 1: IUT ok (1): the IUT does not support notifications. Step 1: IUT ok (2): the IUT handles notifications correctly. Step 1: IUT not ok (1): the IUT does not store any entry into its notification matrix. Step 1: IUT not ok (2): the IUT deletes the notification matrix after `Configuration.Status(NewExt)`. Step 1: IUT not ok (3): the IUT deletes the notification matrix after `Configuration.Status(Invalid)`. Step 1: IUT not ok (4): the IUT does not delete the notification matrix after `Configuration.Status(NotOK)`. Step 1: IUT not ok (5): the IUT does not delete the notification matrix after `ev_Init_Ready`.
Remark	1. At the beginning of the CTC, the notification matrix of the IUT does not contain any entry of the UT. 2. The CTC is based on generation of NCEs so the IUT performs an FBlock scan (check the FBlock configuration) on each NCE and the UT sets the FBlocks and address associated with the LT post-IUT accordingly.


IUT in s_NetInterface_Normal_Operation
LT pre-IUT: Send Notification.Set(All) to IUT
a
Notification.Error received within $t_{NotificationProperty}$?
yes
Notification.Error (20_{16}, 20_{16}) or Notification.Error (03_{16}) received?
no
no
LT pre-IUT: Send new FBlockID list with new FBlockIDs
b
Received Configuration.Status(NewExt) from IUT?
c
no
yes
Notification matrix empty?
d
e
yes
no
LT post-IUT: Configure as listen-only node
f
Received Configuration.Status(Invalid) from IUT?
g
no
yes
Notification matrix empty?
d
e
yes
no
h
LT post-IUT: Change address to uninitialised_node_address; configure as TimingSlave
yes
IUT not ok (2)
IUT not ok (1)
IUT not ok (3)
IUT ok (1)
Received Configuration.Status(NotOK) from IUT?
i
no
yes
j
Request notification matrix from IUT
Notification matrix empty?
d
yes
no
LT pre-IUT: Write some entry to the notification matrix of the IUT
k
Perform shutdown
Perform wake-up
Request notification matrix from IUT
l
Notification matrix empty?
e
yes
no
IUT ok (2)
IUT not ok (5)
IUT not ok (3)

a The UT, using the LT pre-IUT, shall add entries to the notification matrix of the IUT by sending `Notification.Set(All)`.

If the IUT supports notification, it does not respond with a notification error.

b The UT shall send new `FBlockID` lists with new `FBlockIDs` to the IUT.

c The IUT sends Configuration.Status(NewExt). It does not delete the entries from the notification matrix.

d Detectable by requesting notification matrix from the IUT. Check only entries made by the UT.

e Each notified function of each FBlock of the IUT is checked.

f The LT post-IUT shall leave the ring by disconnecting its MOST node from the network and leaving only the listen-only node connected.

g The IUT sends `Configuration.Status(Invalid)`. Its notification matrix is not empty. The UT shall stop the CTC if the IUT does not respond within $t_{DeadLockShort}$.

h The LT post-IUT shall change the logical node address of its MOST node to `uninitialised_node_address` and connect it to the network again. The UT shall respond to queries from the IUT to the LT post-IUT.

i The IUT broadcasts `Configuration.Status(NotOK)`.

[j] The IUT broadcasts `Configuration.Status(OK)` and its notification matrix does not contain any entry of the UT. Wait for `Configuration.Status(OK)` before requesting the notification matrix.

[k] The UT, using the LT pre-IUT, shall write some entry to the notification matrix of the IUT and shutdown the network.

[l] Check only entries made by the UT. At the next wake-up, the notification matrix of the IUT does not contain any entry of the UT.

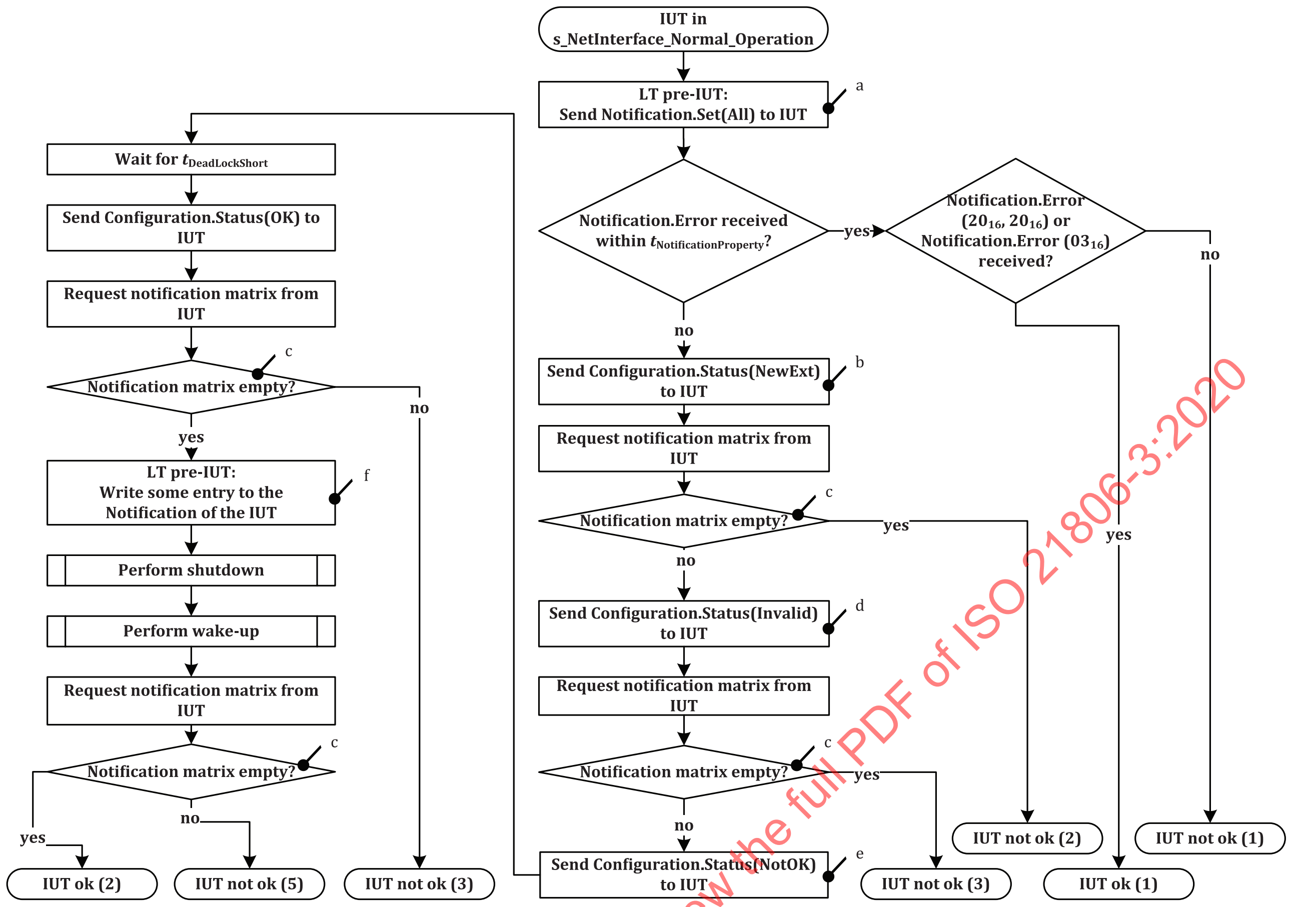
Figure 27 — CTC_2.8.3-1a - Notification matrix storage test (NetworkMaster)

8.6.2 CTC_2.8.3-1b – Notification matrix storage test (NetworkSlave)

Table 34 specifies the CTC_2.8.3-1b – Notification matrix storage test (NetworkSlave).

Table 34 — CTC_2.8.3-1b – Notification matrix storage test (NetworkSlave)

Item	Content
CTC # – Title	CTC_2.8.3-1b – Notification matrix storage test (NetworkSlave)
Purpose	This CTC verifies that a NetworkSlave handles its notification matrix and corresponding requests correctly. This CTC applies to all MOST devices that do not contain the NetworkMaster.
Reference	ISO 21806-2:2020, 6.6 APP – Notification mechanisms
Prerequisite	— The UT shall effectuate `s_NetInterface_Normal_Operation` in the node that contains the IUT. — The ring shall be closed.
Set-up	— The LT pre-IUT shall be the TimingMaster. — The LT post-IUT shall be a listen-only node.
Step	1. The UT shall execute the sequence as specified in Figure 28.
Iteration	The CTC shall be performed with every FBlock and function of the IUT, registered in the central registry except FBlock PowerMaster. All FBlocks shall be tested within one test loop (no separate test loop for every FBlock) to increase stress of the IUT.
Expected response	Step 1: IUT ok (1): the IUT passes the CTC. It does not support notification. Step 1: IUT ok (2): the IUT passes the CTC. Step 1: IUT not ok (1): the IUT does not store any entry into its notification matrix. Step 1: IUT not ok (2): the IUT deletes notification matrix if `Configuration.Status(NewExt)` is received. Step 1: IUT not ok (3): the IUT deletes notification matrix if Configuration.Status (Invalid) is received. Step 1: IUT not ok (4): the IUT does not delete the notification matrix if `Configuration.Status(NotOK)` is received. Step 1: IUT not ok (5): the IUT does not delete the notification matrix after `ev_Init_Ready`.
Remark	At the beginning of the CTC, the notification of the IUT does not contain any entry associated with the LT pre-IUT.


IUT in s_NetInterface_Normal_Operation
LT pre-IUT: Send Notification.Set(All) to IUT
Notification.Error received within $t_{NotificationProperty}$?
yes
Notification.Error (20_{16}, 20_{16}) or Notification.Error (03_{16}) received?
no
yes
no
Send Configuration.Status(NewExt) to IUT
Request notification matrix from IUT
Notification matrix empty?
yes
no
Send Configuration.Status(Invalid) to IUT
Request notification matrix from IUT
Notification matrix empty?
yes
no
Send Configuration.Status(NotOK) to IUT
IUT not ok (3)
IUT not ok (2)
IUT ok (1)
IUT not ok (1)
Wait for $t_{DeadLockShort}$
Send Configuration.Status(OK) to IUT
Request notification matrix from IUT
Notification matrix empty?
no
yes
LT pre-IUT: Write some entry to the Notification of the IUT
Perform shutdown
Perform wake-up
Request notification matrix from IUT
Notification matrix empty?
yes
no
IUT ok (2)
IUT not ok (5)
IUT not ok (3)
a
b
c
d
e
f

a The UT, using the LT pre-IUT, shall request addition to the notification matrix of the IUT by sending `Notification.Set(All)`. If the IUT supports notification, it does not generate a notification error.

b The UT shall send `Configuration.Status(NewExt)`. The IUT does not delete the entries from the notification matrix.

c Check only entries associated with the LT pre-IUT. Each notified function of each FBlock of the IUT is checked.

d The UT shall send `Configuration.Status(Invalid)`. The node that is stored in the notification matrix (LT pre-IUT) shall not be indicated as invalid. The IUT does not delete the entries from the notification matrix.

e The UT shall send `Configuration.Status(NotOK)`. The IUT deletes the entries of the UT from the notification matrix.

f The UT, using the LT pre-IUT, shall write an entry to the notification matrix of the IUT. The UT shall shutdown the network. At the next wake-up, the notification matrix of the IUT does not contain any entry of the UT.

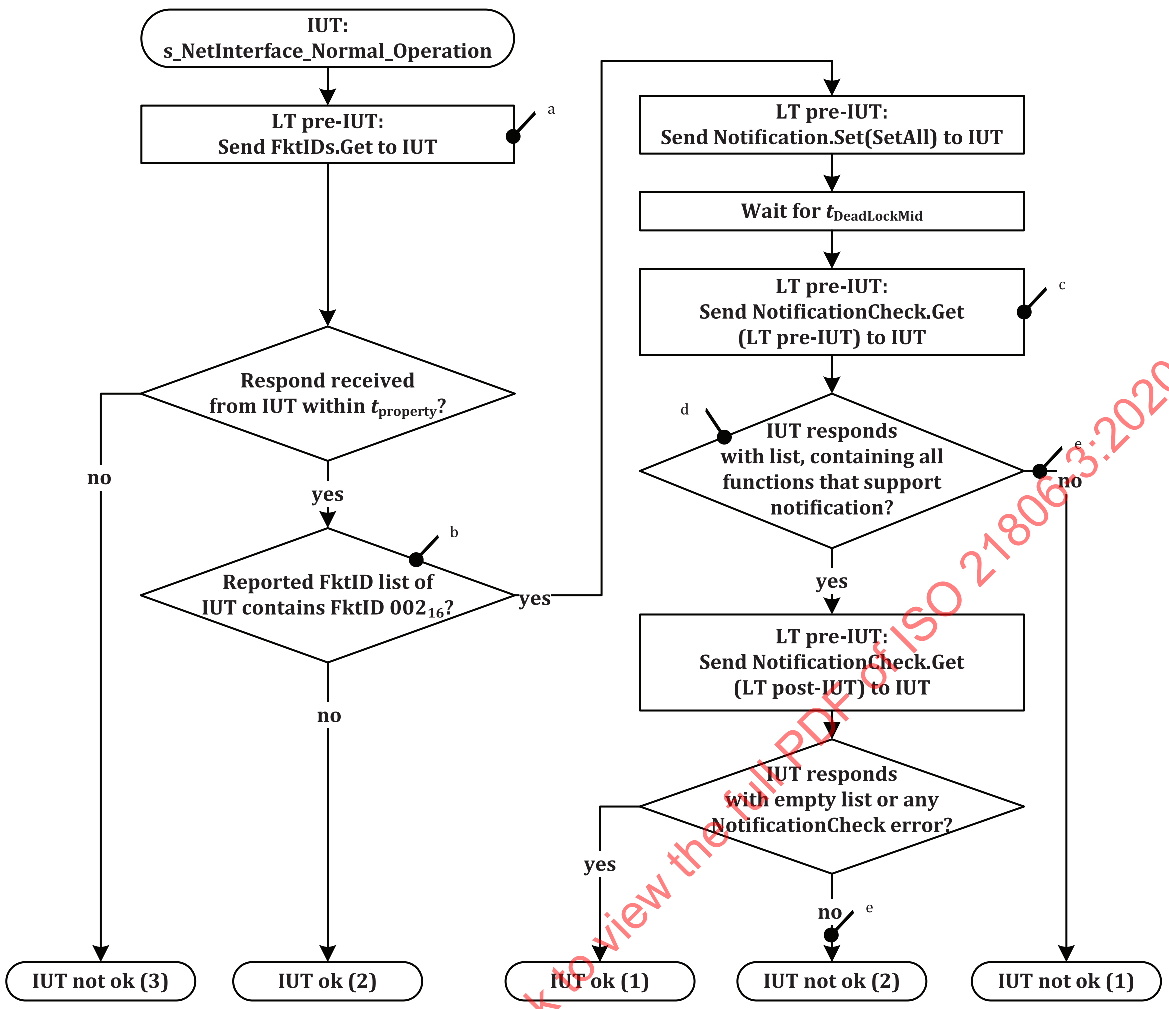
Figure 28 — CTC_2.8.3-1b - Notification matrix storage test (Slave)

8.6.3 CTC_2.8.3-2 – NotificationCheck test

Table 35 specifies the CTC_2.8.3-2 – NotificationCheck test.

Table 35 — CTC_2.8.3-2 – NotificationCheck test

Item	Content
CTC # – Title	CTC_2.8.3-2 – NotificationCheck test
Purpose	This CTC verifies that a node correctly responds to NotificationCheck requests. This CTC applies to all MOST devices.
Reference	ISO 21806-2:2020, 7.5 AL – Function identifier (FktID)
Prerequisite	The UT shall effectuate `s_NetInterface_Normal_Operation` in the node that contains the IUT.
Set-up	— If the IUT is part of a MOST device that contains the TimingMaster, the LT pre-IUT shall be a TimingSlave. — If the IUT is part of a MOST device that does not contain the TimingMaster, the LT pre-IUT shall be the TimingMaster. — The LT post-IUT shall be a TimingSlave.
Step	1. The UT shall execute the sequence as specified in Figure 29.
Iteration	The CTC shall be performed with every FBlock of the IUT that has an entry in the central registry.
Expected response	Step 1: IUT ok (1): the IUT responds to `NotificationCheck` requests. Step 1: IUT ok (2): the IUT does not support `NotificationCheck`. Step 1: IUT not ok (1): the IUT does not respond with a list, containing all functions that support notification. Step 1: IUT not ok (2): the list from the IUT (for the LT post-IUT) is not empty. Step 1: IUT not ok (3): the IUT does not respond to `FktIDs.Get` in time.
Remark	---


IUT:
s_NetInterface_Normal_Operation
LT pre-IUT:
Send FktIDs.Get to IUT
a
Respond received from IUT within $t_{property}$?
no
yes
Reported FktID list of IUT contains FktID 002_{16}?
b
yes
no
LT pre-IUT:
Send Notification.Set(SetAll) to IUT
Wait for $t_{DeadLockMid}$
LT pre-IUT:
Send NotificationCheck.Get
(LT pre-IUT) to IUT
c
d
IUT responds with list, containing all functions that support notification?
e
no
yes
LT pre-IUT:
Send NotificationCheck.Get
(LT post-IUT) to IUT
IUT responds with empty list or any NotificationCheck error?
yes
no
e
IUT not ok (3)
IUT ok (2)
IUT ok (1)
IUT not ok (2)
IUT not ok (1)

[a] The UT, using the LT pre-IUT, shall request the `FktID` list of the IUT to check whether `NotificationCheck` is supported. The UT shall not add entries associated with the LT post-IUT to the notification matrix of the IUT.

[b] If the IUT supports `NotificationCheck`, the UT, using the LT pre-IUT, shall request notification entries for all properties of the current FBlock that support notification from the IUT. `FktID` 002_{16} = `NotificationCheck`.

[c] The UT, using the LT pre-IUT, shall send `NotificationCheck` to the IUT.

[d] The IUT responds with a list that contains all functions supporting notification. If the FBlock does not contain any functions that support notification, the IUT may respond with an empty list or any `NotificationCheck` error.

[e] The UT shall stop the CTC if the IUT does not respond within $t_{DeadLockShort}$.

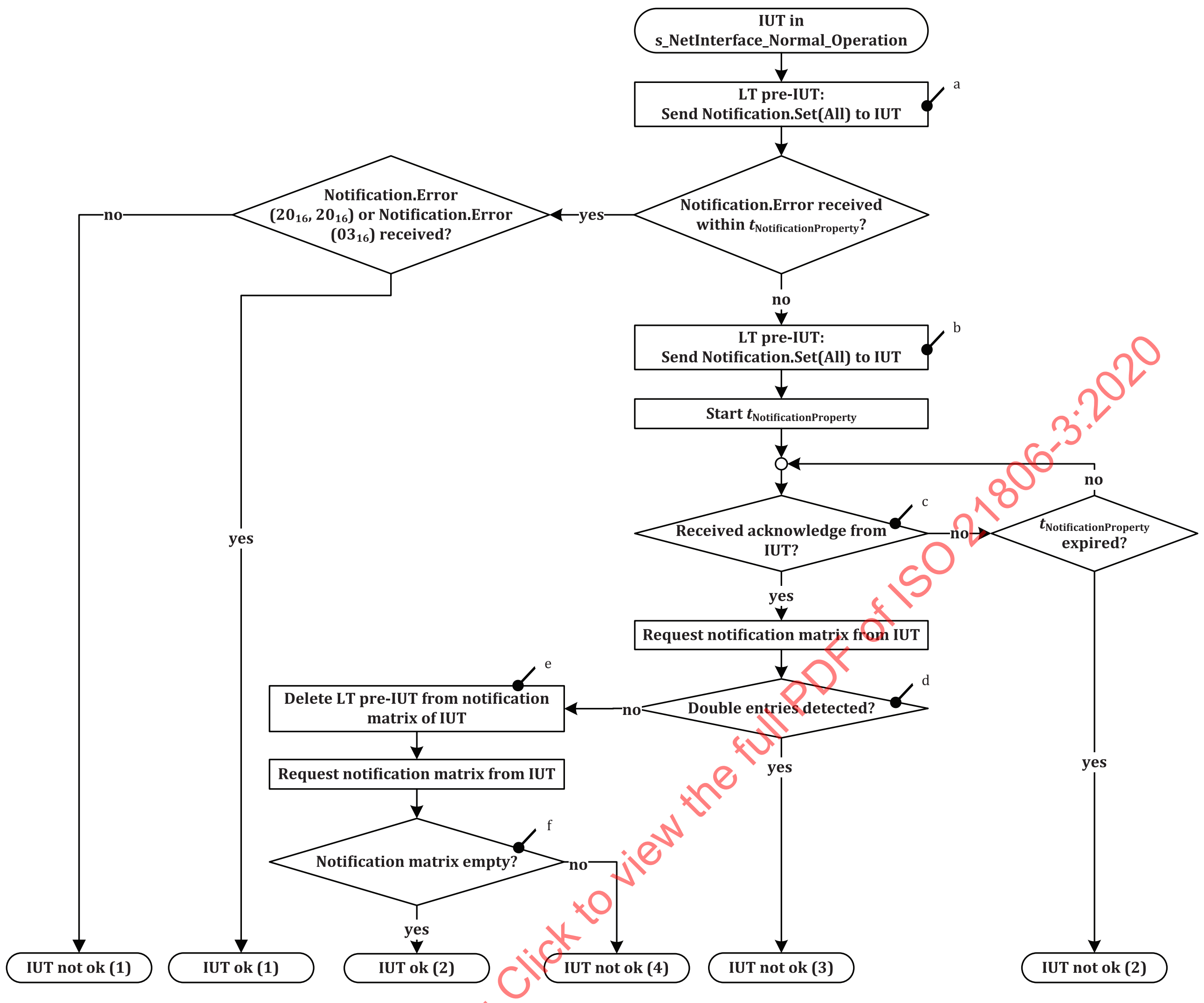
Figure 29 — CTC_2.8.3-2 - NotificationCheck test

8.6.4 CTC_2.8.3-7 – Notification matrix double entry test

Table 36 specifies the CTC_2.8.3-7 – Notification matrix double entry test.

Table 36 — CTC_2.8.3-7 – Notification matrix double entry test

Item	Content
CTC # – Title	CTC_2.8.3-7 – Notification matrix double entry test
Purpose	This CTC verifies that a node handles requests for creation of duplicate entries in the notification matrix correctly. This CTC applies to all MOST devices.
Reference	ISO 21806-2:2020, 6.6 APP – Notification mechanisms
Prerequisite	— The UT shall effectuate `s_NetInterface_Normal_Operation` in the node that contains the IUT. — The ring shall be closed.
Set-up	— If the IUT is part of a MOST device that contains the TimingMaster, the LT pre-IUT shall be a TimingSlave. — If the IUT is part of a MOST device that does not contain the TimingMaster, the LT pre-IUT shall be the TimingMaster. — The LT post-IUT shall be a TimingSlave.
Step	1. The UT shall execute the sequence as specified in Figure 30.
Iteration	The CTC shall be performed with every registered FBlock and function of the IUT, registered in the central registry except the FBlock PowerMaster. All FBlocks shall be tested within one test loop (no separate test loop for every FBlock) to increase stress of the IUT.
Expected response	Step 1: IUT ok (1): the IUT does not support notification. Step 1: IUT ok (2): the IUT handles duplicate notification requests correctly. Step 1: IUT not ok (1): the IUT does not store any entry into its notification matrix. Step 1: IUT not ok (2): the IUT does not acknowledge entries to the notification matrix within $t_{\text{NotificationProperty}}$. Step 1: IUT not ok (3): the IUT stores double entries into the notification matrix. Step 1: IUT not ok (4): the IUT does not delete entries from the notification matrix.
Remark	1. To check whether the IUT supports notification, the FBlock EnhancedTestability can be used to check the size of the notification matrix. 2. At the beginning of the CTC, the notification matrix of the IUT does not contain any entry associated with the LT pre-IUT.


IUT in s_NetInterface_Normal_Operation
LT pre-IUT: Send Notification.Set(All) to IUT
a
Notification.Error received within $t_{NotificationProperty}$?
yes
Notification.Error (20_{16}, 20_{16}) or Notification.Error (03_{16}) received?
no
yes
no
LT pre-IUT: Send Notification.Set(All) to IUT
b
Start $t_{NotificationProperty}$
no
Received acknowledge from IUT?
c
no
$t_{NotificationProperty}$ expired?
yes
Request notification matrix from IUT
Double entries detected?
d
no
Delete LT pre-IUT from notification matrix of IUT
e
Request notification matrix from IUT
Notification matrix empty?
f
no
yes
yes
yes
IUT not ok (1)
IUT ok (1)
IUT ok (2)
IUT not ok (4)
IUT not ok (3)
IUT not ok (2)

a Add a notification entry for the LT pre-IUT to every function of the IUT that supports notification. To get all functions that can be used for notification, the UT, using the LT pre-IUT, shall send `Notification.Set(All)` to the IUT. If the IUT supports notification, it does not respond with a notification error.

b Add a notification entry for the LT pre-IUT to every function of the IUT that supports notification, again. The UT, using the LT pre-IUT, shall send `Notification.Set(All)` to the IUT, again (try to generate duplicate entries).

c Reception of the first (or single) segment of the first status or error message. For duplicate entries, the IUT sends the corresponding report but it does not modify the notification matrix.

d It is tolerated if there is no entry associated with the LT pre-IUT; this does not lead to IUT not ok (3).

e The UT shall remove entries associated with the LT pre-IUT from the notification matrix.

f Check only entries associated with the LT pre-IUT. Each notified function of each FBlock of the IUT is checked. The notification matrix does not contain any entry associated with the LT pre-IUT.

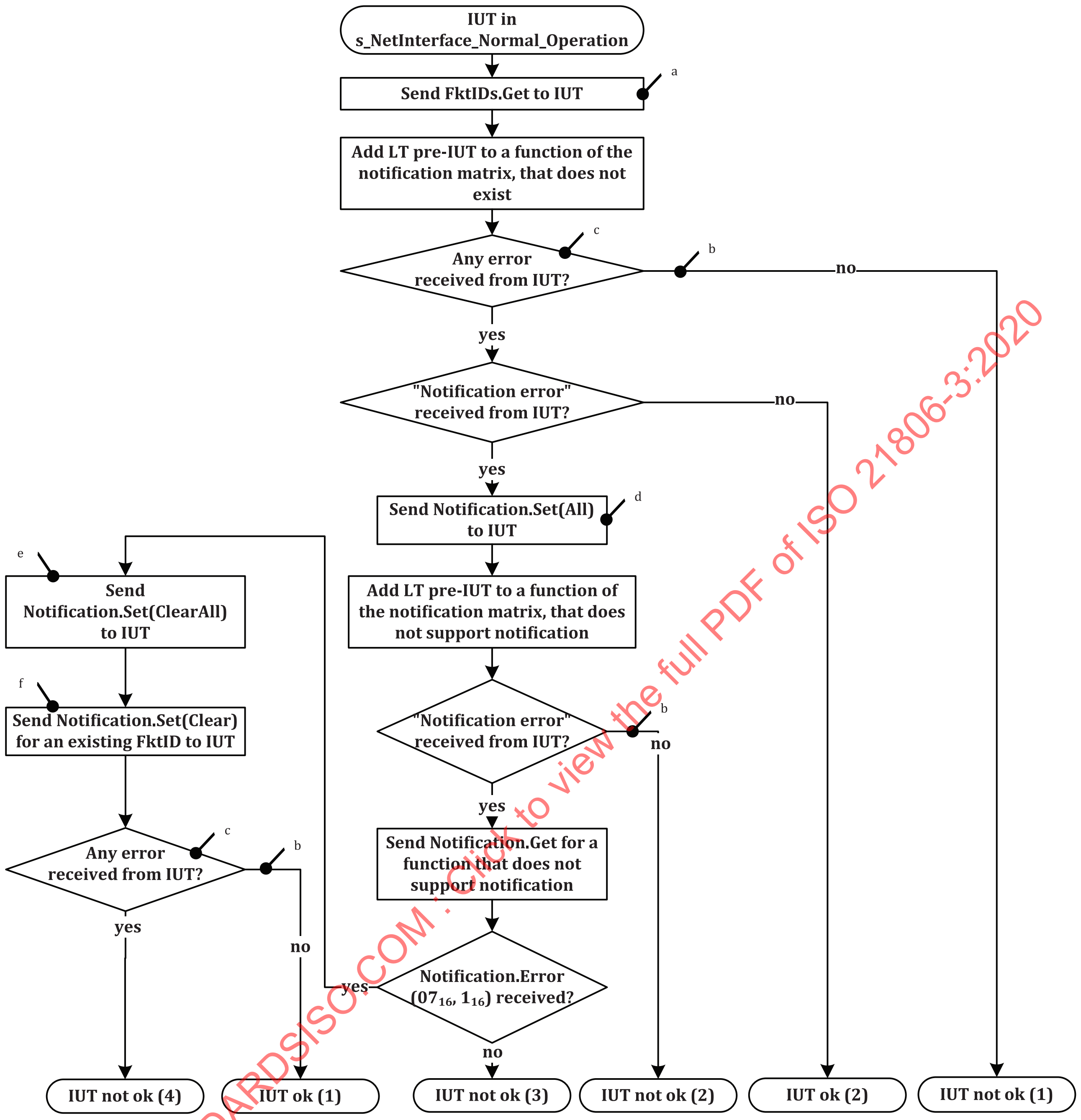
Figure 30 — CTC_2.8.3-7 - Notification matrix double entry test

8.6.5 CTC_2.8.3-10 – Notification error test

Table 37 specifies the CTC_2.8.3-10 – Notification error test.

Table 37 — CTC_2.8.3-10 – Notification error test

Item	Content
CTC # – Title	CTC_2.8.3-10 – Notification error test
Purpose	This CTC verifies that a node responds with an error message to incorrect notification requests. This CTC applies to all MOST devices.
Reference	ISO 21806-2:2020, 6.6 APP – Notification mechanisms
Prerequisite	— The UT shall effectuate `s_NetInterface_Normal_Operation` in the node that contains the IUT. — The ring shall be closed.
Set-up	— If the IUT is part of a MOST device that contains the TimingMaster, the LT pre-IUT shall be a TimingSlave. — If the IUT is part of a MOST device that does not contain the TimingMaster, the LT pre-IUT shall be the TimingMaster. — The LT post-IUT shall be a listen-only node.
Step	1. The UT shall execute the sequence as specified in Figure 31.
Iteration	The CTC shall be performed with every FBlock of the IUT registered in the central registry except FBlock PowerMaster. All FBlocks shall be tested within one test loop (no separate test loop for every FBlock) to increase stress of the IUT.
Expected response	Step 1: IUT ok (1): the IUT handles faulty notification requests correctly. Step 1: IUT ok (2): the IUT does not support notification. Step 1: IUT not ok (1): the IUT does not send any error if the function does not exist. Step 1: IUT not ok (2): the IUT does not send `Notification.Error` if the function does not support notification. Step 1: IUT not ok (3): the IUT does not send `Notification.Error`(07_{16}, 01_{16}) if the requested function does not support notification. Step 1: IUT not ok (4): the IUT responds with an error to `Notification.Set(Clear)`.
Remark	---


IUT in s_NetInterface_Normal_Operation
Send FktIDs.Get to IUT
a
Add LT pre-IUT to a function of the notification matrix, that does not exist
Any error received from IUT?
c
b
no
yes
"Notification error" received from IUT?
no
yes
Send Notification.Set(All) to IUT
d
Add LT pre-IUT to a function of the notification matrix, that does not support notification
"Notification error" received from IUT?
b
no
yes
Send Notification.Get for a function that does not support notification
Notification.Error (07_{16}, 1_{16}) received?
yes
no
e
Send Notification.Set(ClearAll) to IUT
f
Send Notification.Set(Clear) for an existing FktID to IUT
Any error received from IUT?
c
b
yes
no
IUT not ok (4)
IUT ok (1)
IUT not ok (3)
IUT not ok (2)
IUT ok (2)
IUT not ok (1)

a The UT, using the LT pre-IUT, shall request a list of all FktIDs of the IUT (by means of `FktIDs.Get`). It shall add a notification entry for function that does not exist. Use a function that is not received after sending `FktIDs.Get`.

b Timeout: $t_{\text{DeadLockShort}}$.

c The IUT responds with a notification error if the FBlock supports notification. If the IUT responds with `Error`(20_{16}, 20_{16}) or `Error`(03_{16}), the FBlock does not support notification.

d The UT, using the LT pre-IUT, shall send `Notification.Set(All)` to the IUT to get a list of all functions that support notification. It shall add an entry for a function that does not support notification. By comparison of the results of `FktIDs.Get` and `Notification.Set(All)`, the functions that do not support notification are identified. The IUT responds with an error.

If all functions support notification, skip this part and go directly to send `Notification.Set(ClearAll)` to the IUT.

e The UT, using the LT pre-IUT, shall send `Notification.Get` for a function that does not support notification. The IUT responds with an error.

f The UT, using the LT pre-IUT, shall clear the notification matrix by means of `Notification.Set(ClearAll)`.

g The UT, using the LT pre-IUT, shall request deletion of an entry for an existing, not notified function from the notification matrix of the IUT. Use a function that supports notification. The IUT does not return any error.

Figure 31 — CTC_2.8.3-10 - Notification error test

8.7 CTC_3.0-1 – TEST_GSI_GSO_Identification

The CTC for GSI and GSO identification is specified using message sequence charts, see Z.120[5].

Table 38 specifies the TEST_GSI_GSO_Identification.

Table 38 — TEST_GSI_GSO_Identification

Item	Content
CTC # – Title	CTC_3.0-1 – TEST_GSI_GSO_Identification
Purpose	This CTC verifies — that a node reports source and sink FBlocks and responds to `FktIDs.Get` correctly; — t_{Property}; — reported `FktIDs`. This CTC applies to all MOST devices.
Reference	ISO 21806-2:2020, 7.5 AL – Function identifier (FktID)
Preconditions	The UT shall effectuate `s_NetInterface_Normal_Operation` in the node that contains the IUT.
Set-up	— If the IUT is part of a MOST device that contains the TimingMaster, the LT pre-IUT shall be a TimingSlave; — If the IUT is part of a MOST device that does not contain the TimingMaster, the LT pre-IUT shall be the TimingMaster; — The LT post-IUT shall be a listen-only node.
Step	1. The UT shall execute the sequence as specified in Figure 32.
Iteration	The CTC shall be performed with every single FBlock, reported by the IUT by means of `NetBlock.FBlockIDs.Status`.
Expected response	Step 1: IUT ok (1): FBlock contains neither sink nor source. Step 1: IUT ok (2): FBlock contains a source. Step 1: IUT ok (3): FBlock contains a source and a sink. Step 1: IUT ok (4): FBlock contains a sink. Step 1: IUT not ok (1): the IUT does not send `FktIDs.Status`. Step 1: IUT not ok (2): FBlocks of the IUT, containing sinks/sources that do not match to the manufacturer list.

Table 38 *(continued)*

Item	Content
Remark	1. The CTC is performed to identify whether the IUT contains sink or source. 2. The manufacturer provides a list with all FBlocks of the IUT, containing sink and/or source functionality. 3. An IUT with result "IUT ok (1)" shall not run through any source or sink related test. 4. An IUT with result "IUT ok (2)" shall run through all source related tests (relevant FBlocks only) 5. An IUT with result "IUT ok (3)" shall run through all source and sink related tests (relevant FBlocks only) 6. An IUT with result "IUT ok (4)" shall run through all sink related tests (relevant FBlocks only)

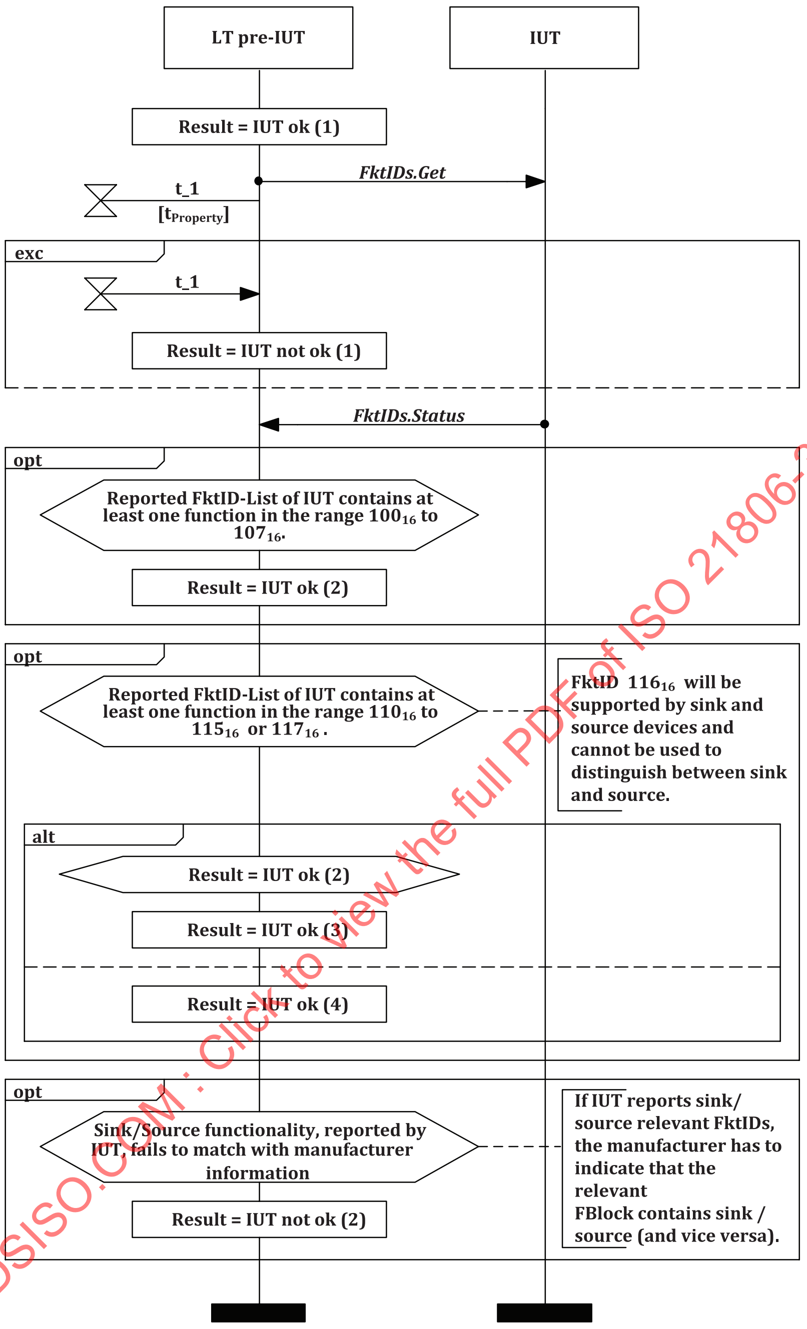

LT pre-IUT
IUT
Result = IUT ok (1)
FktIDs.Get
t_1
[tProperty]
exc
t_1
Result = IUT not ok (1)
FktIDs.Status
opt
Reported FktID-List of IUT contains at least one function in the range 100₁₆ to 107₁₆.
Result = IUT ok (2)
opt
Reported FktID-List of IUT contains at least one function in the range 110₁₆ to 115₁₆ or 117₁₆ .
FktID 116₁₆ will be supported by sink and source devices and cannot be used to distinguish between sink and source.
alt
Result = IUT ok (2)
Result = IUT ok (3)
Result = IUT ok (4)
opt
Sink/Source functionality, reported by IUT, fails to match with manufacturer information
Result = IUT not ok (2)
If IUT reports sink/ source relevant FktIDs, the manufacturer has to indicate that the relevant FBlock contains sink / source (and vice versa).

Figure 32 — CTC_3.0-1 - TEST_GSI_GSO_Identification

8.8 Obligatory tests for sink and source MOST devices

8.8.1 General

The CTCs for sink and sources are specified using message sequence charts, see Z.120[5].

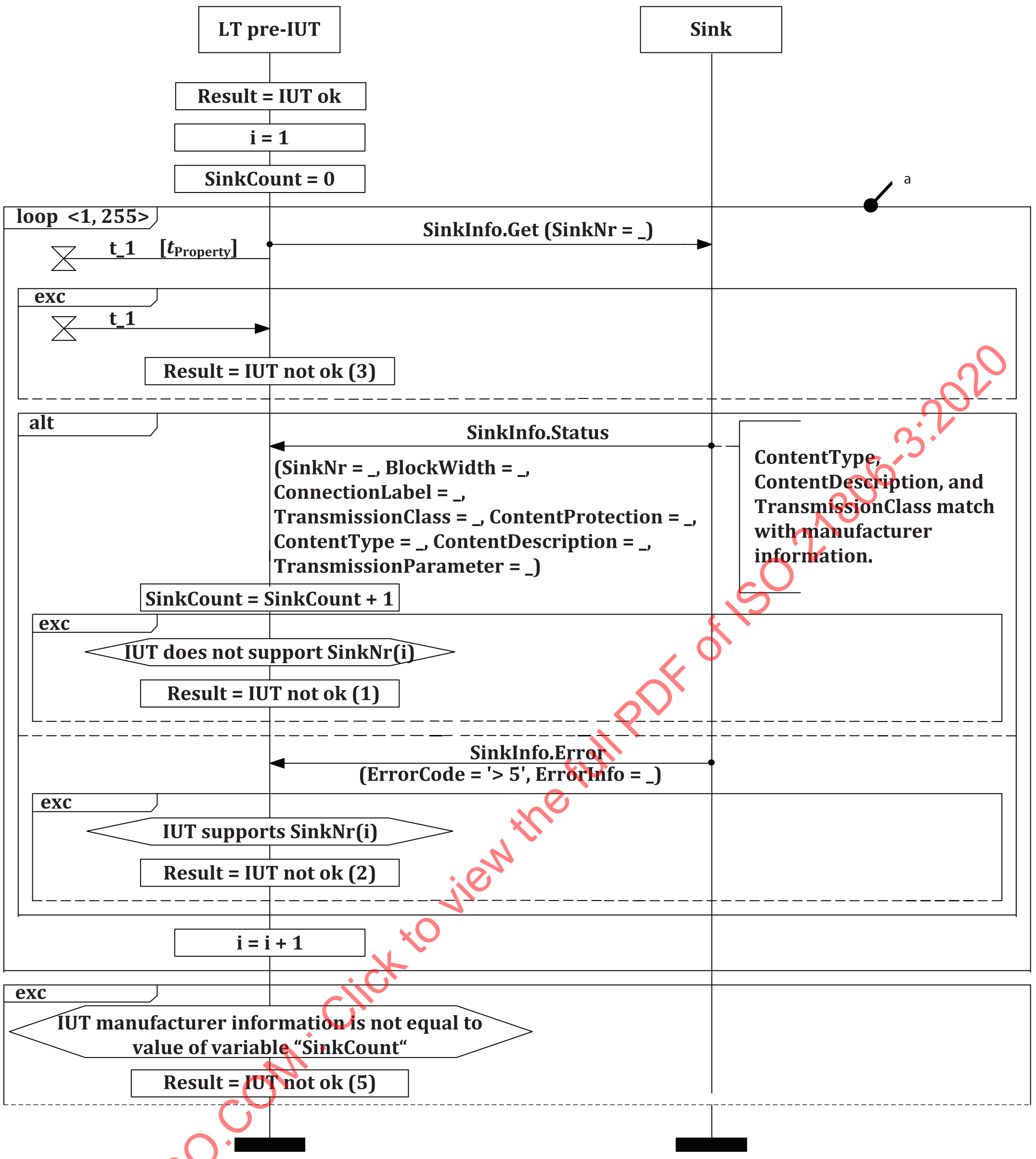
8.8.2 Sink MOST devices

8.8.2.1 CTC_3.1-1 – TEST_GSI_SinkInfo

Table 39 specifies the CTC_3.1-1 – TEST_GSI_SinkInfo.

Table 39 — CTC_3.1-1 – TEST_GSI_SinkInfo

Item	Content
CTC # – Title	CTC_3.1-1 – TEST_GSI_SinkInfo
Purpose	This CTC verifies that a sink correctly responds to `SinkInfo` requests. This CTC applies to all MOST devices that contain at least one sink.
Reference	ISO 21806-2:2020, 6.10.2.3.1 APP – SinkInfo function
Preconditions	The UT shall effectuate `s_NetInterface_Normal_Operation` in the node that contains the IUT.
Set-up	— If the IUT is part of a MOST device that contains the TimingMaster, the LT pre-IUT shall be a TimingSlave. — If the IUT is part of a MOST device that does not contain the TimingMaster, the LT pre-IUT shall be the TimingMaster. — The LT post-IUT shall be a listen-only node.
Step	1. The UT shall execute the sequence as specified in Figure 33.
Iteration	The UT shall perform the CTC with every `SinkNr` (1 to 255) according to the manufacturer information.
Expected response	Step 1: IUT ok: the IUT correctly responds to `SinkInfo` requests. Step 1: IUT not ok (1): the IUT responds status although `SinkNr` not supported. Step 1: IUT not ok (2): the IUT responds with an error although `SinkNr` supported. Step 1: IUT not ok (3): the IUT does not respond within t_{Property} to `SinkInfo.Get`. Step 1: IUT not ok (5): the IUT manufacturer information is not equal to the reported number of sinks.
Remark	---


LT pre-IUT
Sink
Result = IUT ok
i = 1
SinkCount = 0
a
loop <1, 255>
SinkInfo.Get (SinkNr = _)
t_1 [tProperty]
exc
t_1
Result = IUT not ok (3)
alt
SinkInfo.Status
(SinkNr = _, BlockWidth = _, ConnectionLabel = _, TransmissionClass = _, ContentProtection = _, ContentType = _, ContentDescription = _, TransmissionParameter = _)
ContentType, ContentDescription, and TransmissionClass match with manufacturer information.
SinkCount = SinkCount + 1
exc
IUT does not support SinkNr(i)
Result = IUT not ok (1)
SinkInfo.Error (ErrorCode = '> 5', ErrorInfo = _)
exc
IUT supports SinkNr(i)
Result = IUT not ok (2)
i = i + 1
exc
IUT manufacturer information is not equal to value of variable "SinkCount"
Result = IUT not ok (5)

a The UT shall process each `SinkNr` by sending `SinkInfo.Get` to the IUT. The UT shall skip a sink if the bandwidth of the sink is not determinable because of the data type (`ContentType` $\mathtt{C0}_{16}$ to $\mathtt{EF}_{16}$).
If the IUT supports a `SinkNr`, it responds with the correct `ContentType`, `ContentDescription` (data type of the parameter) and `TransmissionClass`. If the IUT does not support a `SinkNr`, it responds with an error.

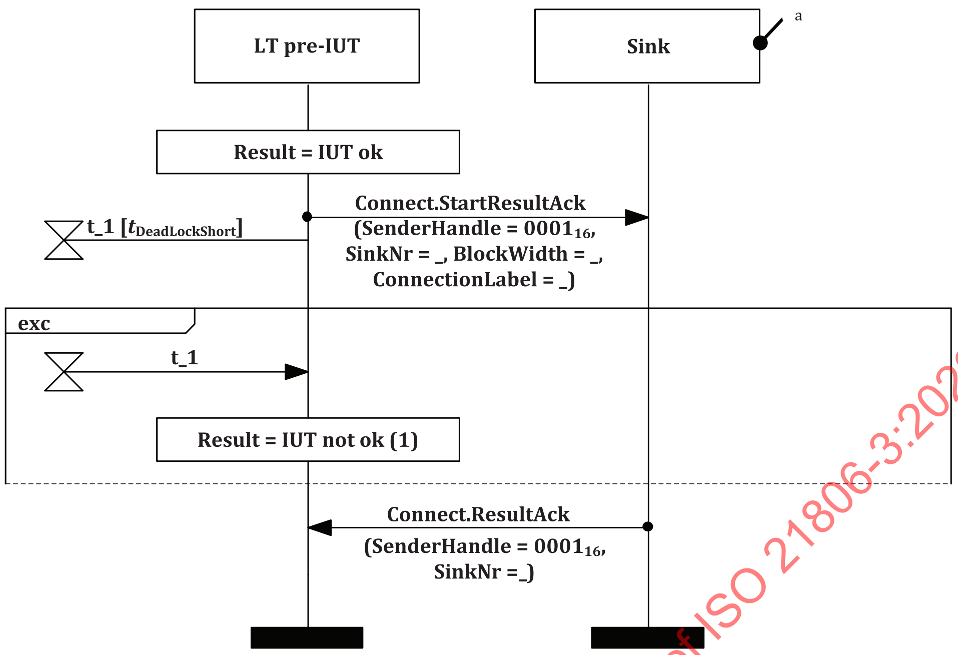
Figure 33 — CTC_3.1-1 - TEST_GSI_SinkInfo

8.8.2.2 CTC_3.1-3 – TEST_GSI_Connect

Table 40 specifies the CTC_3.1-3 – TEST_GSI_Connect.

Table 40 — CTC_3.1-3 – TEST_GSI_Connect

Item	Content
CTC # – Title	CTC_3.1-3 – TEST_GSI_Connect
Purpose	This CTC verifies that a sink handles `Connect.StartResultAck` correctly. This CTC applies to all MOST devices that contain at least one sink that supports transmission class “synchronous”. This CTC is only applicable if the node that contains the IUT does not contain the connection manager.
Reference	ISO 21806-2:2020, 6.10.3.2.1 APP – Connect function
Preconditions	— The UT shall effectuate `s_NetInterface_Normal_Operation` in the node that contains the IUT.
Set-up	— If the IUT is part of a MOST device that contains the TimingMaster, the LT pre-IUT shall be a TimingSlave. — If the IUT is part of a MOST device that does not contain the TimingMaster, the LT pre-IUT shall be the TimingMaster. — The LT post-IUT shall be a listen-only node.
Step	1. The UT shall execute the sequence as specified in Figure 34.
Iteration	The UT shall perform the CTC with every `SinkNr` that is supported by the IUT.
Expected response	Step 1: IUT ok: the IUT handles `Connect.StartResultAck` correctly. Step 1: IUT not ok (1): the IUT does not respond within $t_{\text{DeadLockShort}}$.
Remark	1. The UT ensures that the connections are in use by a source. Some sinks check the validity of the connections before connecting. Only if the TimingMaster indicates the connections as allocated by a source, the sink connects. 2. Timeout $t_{\text{DeadLockShort}}$ is valid for the whole CTC.


LT pre-IUT
Sink
a
Result = IUT ok
Connect.StartResultAck (SenderHandle = 0001$_{16}$, SinkNr = _, BlockWidth = _, ConnectionLabel = _)
t_1 [$t_{DeadLockShort}$]
exc
t_1
Result = IUT not ok (1)
Connect.ResultAck (SenderHandle = 0001$_{16}$, SinkNr =_)

[a] The UT shall process each `SinkNr` by sending `Connect.StartResultAck` to the IUT. The UT shall ensure that the connections to which a sink shall connect to are in use by a source.
The UT shall skip a sink if the bandwidth of the sink is not determinable because of the data type (`ContentType` `C0`$_{16}$ to `EF`$_{16}$).
The IUT processes `Connect.StartResultAck` correctly.
The UT shall use `DisConnect` before processing the next sink to avoid resource overflow.

Figure 34 — CTC_3.1-3 - TEST_GSI_Connect

8.8.2.3 CTC_3.1-4 – TEST_GSI_Connect_Repeat

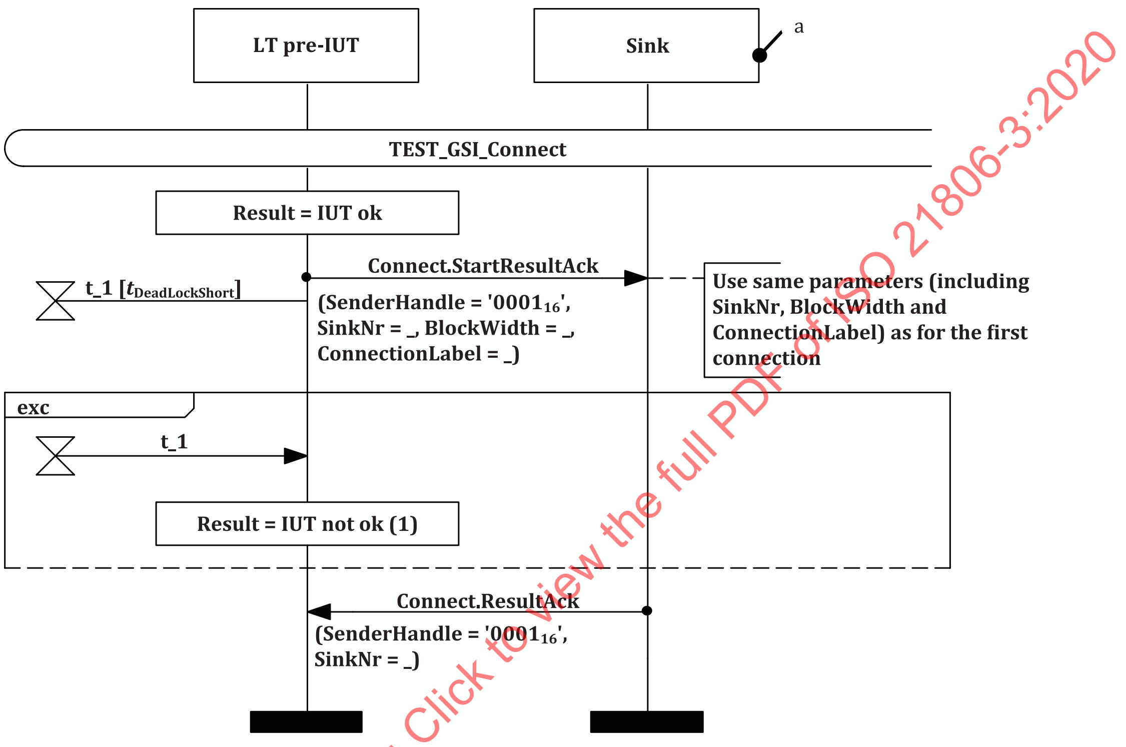
Table 41 specifies the CTC_3.1-4 – TEST_GSI_Connect_Repeat.

Table 41 — CTC_3.1-4 – TEST_GSI_Connect_Repeat

Item	Content
CTC # – Title	CTC_3.1-4 – TEST_GSI_Connect_Repeat
Purpose	This CTC verifies that sink handles repeated `Connect.StartResultAck` commands correctly. This CTC applies to all MOST devices that contain at least one sink that supports transmission class “synchronous”. This CTC is only applicable if the node that contains the IUT does not contain the connection manager.
Reference	ISO 21806-2:2020, 6.10.3.2.1 APP – Connect function
Preconditions	The UT shall effectuate `s_NetInterface_Normal_Operation` in the node that contains the IUT.
Set-up	— If the IUT is part of a MOST device that contains the TimingMaster, the LT pre-IUT shall be a TimingSlave. — If the IUT is part of a MOST device that does not contain the TimingMaster, the LT pre-IUT shall be the TimingMaster. — The LT post-IUT shall be a listen-only node.
Step	1. The UT shall execute the sequence as specified in Figure 35.

Table 41 *(continued)*

Item	Content
Iteration	The UT shall perform the test with every `SinkNr` supported by the IUT.
Expected response	Step 1: IUT ok: the IUT handles repeated `Connect.StartResultAck` commands correctly. Step 1: IUT not ok: the IUT does not respond within $t_{\text{DeadLockShort}}$.
Remark	Timeout $t_{\text{DeadLockShort}}$ is valid for the whole CTC.



LT pre-IUT
Sink
a
TEST_GSI_Connect
Result = IUT ok
Connect.StartResultAck
(SenderHandle = '0001₁₆', SinkNr = _, BlockWidth = _, ConnectionLabel = _)
t_1 [tDeadLockShort]
Use same parameters (including SinkNr, BlockWidth and ConnectionLabel) as for the first connection
exc
t_1
Result = IUT not ok (1)
Connect.ResultAck
(SenderHandle = '0001₁₆', SinkNr = _)

[a] The UT shall process each `SinkNr` by sending `Connect.StartResultAck` to the IUT. The UT shall repeat the `Connect.StartResultAck` request.
The UT shall skip a sink if the bandwidth of the sink is not determinable because of the data type (`ContentType` $\texttt{C0}_{16}$ to $\texttt{EF}_{16}$). The IUT processes repeated `Connect.StartResultAck` commands correctly.
The UT shall use `DisConnect` before processing the next sink to avoid resource overflow.

Figure 35 — CTC_3.1-4 - TEST_GSI_Connect_Repeat

8.8.2.4 CTC_3.1-5 – TEST_GSI_DisConnect

Table 42 specifies the CTC_3.1-5 – TEST_GSI_DisConnect.

Table 42 — CTC_3.1-5 – TEST_GSI_DisConnect

Item	Content
CTC # – Title	CTC_3.1-5 – TEST_GSI_DisConnect
Purpose	This CTC verifies that a sink handles `DisConnect.StartResultAck` correctly. This CTC applies to all MOST devices that contain at least one sink that supports transmission class “synchronous”. This CTC is only applicable if the node that contains the IUT does not contain the connection manager.