Document Title	Specification of DMA Driver
Document Category	Software specification (SWS)
Document UID	1148
Document Status	published
Part of AUTOSAR Standard	Classic Platform
Part of Standard Release	R26-11

Document Change History			
Date	**Release**	**Changed by**	**Description**
2026-11-26	R26-11	AUTOSAR Release Management	• Initial release

Disclaimer

This work (specification and/or software implementation) and the material contained in it, as released by AUTOSAR, is for the purpose of information only. AUTOSAR and the companies that have contributed to it shall not be liable for any use of the work.

The material contained in this work is protected by copyright and other types of intellectual property rights. The commercial exploitation of the material contained in this work requires a license to such intellectual property rights.

This work may be utilized or reproduced without any modification, in any form or by any means, for informational purposes only. For any other purpose, no part of the work may be utilized or reproduced, in any form or by any means, without permission in writing from the publisher.

The work has been developed for automotive applications only. It has neither been developed, nor tested for non-automotive applications.

The word AUTOSAR and the AUTOSAR logo are registered trademarks.

Table of Contents

1 Introduction and functional overview 7
1.1 Scope . . . 7
1.2 Functional Overview . . . 7
1.3 Typical Use Cases . . . 8
1.4 Typical Not Supported Use Cases . . . 9
2 Definition of acronyms and abbreviations 10
2.1 Acronyms and abbreviations . . . 10
2.2 Definition of terms . . . 10
3 Related documentation 12
3.1 Input documents & related standards and norms . . . 12
3.2 Related specification . . . 12
4 Constraints and assumptions 13
4.1 Limitations . . . 13
5 Dependencies to other modules 14
6 Requirements Tracing 15
7 Functional specification 16
7.1 Technical Functional Overview . . . 16
7.1.1 DMA Driver Common Functionality . . . 16
7.1.2 DMA Channels and Transfer Elements . . . 16
7.1.3 DMA Transfers Initiated by Hardware . . . 17
7.1.4 DMA Transfers Initiated by Software . . . 17
7.1.5 Scatter-Gather Capability . . . 17
7.1.6 Descriptor Inner/Outer Loop Model . . . 17
7.1.7 Optional Scatter-Gather Descriptor Chaining . . . 19
7.1.8 Data Width Adaptation and Transfer Semantics . . . 21
7.1.8.1 Scope . . . 21
7.1.8.2 Functional Description . . . 21
7.2 Detailed Functional Specification . . . 22
7.2.1 Module State Machine . . . 22
7.2.2 Suspend and Resume Scope Explanation . . . 25
7.2.3 Channel State Machine Variants: Software-Triggered vs Hardware-Triggered . . . 27
7.2.3.1 Software-Triggered Variant . . . 27
7.2.3.2 Hardware-Triggered Variant . . . 28
7.2.3.3 API Restriction: GetChannelStatus for Hardware-Triggered Channels . . . 29
7.2.3.4 Channel Lifecycle APIs per Trigger Mode . . . 29
7.2.3.5 API Applicability per Channel Operation Mode . . . 31

7.2.4 Partition/Core Access Handling . . . 32
7.2.5 Channel and Hardware Unit Configuration Scope . . . 33
7.2.6 Runtime Transfer Reconfiguration APIs . . . 33
7.2.7 Transfer Control Descriptor Completeness . . . 34
7.2.8 Inner and Outer Loop Execution . . . 35
7.2.9 Scatter-Gather Execution . . . 35
7.2.10 Polling-Based Supervision and Advanced Features . . . 38
7.3 Error Classification . . . 39
7.3.1 Development Errors . . . 39
7.3.1.1 Development Error Detection . . . 39
7.3.2 Runtime Errors . . . 41
7.3.2.1 Runtime Error Reporting . . . 41
7.3.3 Production Errors . . . 42
7.4 Security Events . . . 42
8 API specification 43
8.1 Imported types . . . 43
8.2 Type definitions . . . 43
8.2.1 Dma_ChannelIdType . . . 43
8.2.2 Dma_TcdIdxType . . . 43
8.2.3 Dma_TcdCfgType . . . 44
8.2.4 Dma_SourceConfigType . . . 45
8.2.5 Dma_DestinationConfigType . . . 46
8.2.6 Dma_InOutLoopConfigType . . . 46
8.2.7 Dma_ChannelStatusType . . . 47
8.2.8 Dma_CrcResultType . . . 47
8.2.9 Dma_ChannelInfoType . . . 48
8.2.10 Dma_ConfigType . . . 49
8.2.11 Std_ReturnType (Dma-specific extension) . . . 49
8.3 Function definitions . . . 50
8.3.1 Dma_Init . . . 52
8.3.2 Dma_GetLatestTransferIndex . . . 53
8.3.3 Dma_EnableChannelNotifications . . . 55
8.3.4 Dma_DisableChannelNotifications . . . 56
8.3.5 Dma_GetCrcResult . . . 57
8.3.6 Dma_GetTransferParameters . . . 59
8.3.7 Dma_UpdateTransferParameters . . . 61
8.3.8 Dma_ChannelTransferConfigUpdate . . . 64
8.3.9 Dma_DeInit . . . 65
8.3.10 Dma_StartChannelTransfer . . . 66
8.3.11 Dma_StopChannelTransfer . . . 68
8.3.12 Dma_GetChannelStatus . . . 70
8.3.13 Dma_EnableChannel . . . 72
8.3.14 Dma_DisableChannel . . . 74

8.3.15 Dma_SuspendChannelTransfer . . . 75
8.3.16 Dma_ResumeChannelTransfer . . . 76
8.4 Callback notifications . . . 77
8.5 Scheduled functions . . . 77
8.6 Expected interfaces . . . 77
8.6.1 Mandatory interfaces . . . 78
8.6.2 Optional interfaces . . . 78
8.6.3 Configurable interfaces . . . 78
8.6.3.1 Transfer completion callback interface . . . 79
8.6.3.2 Transfer error callback interface . . . 79
8.6.3.3 Hardware error callback interface . . . 81
8.7 Service Interfaces . . . 82
9 Sequence diagrams 83
9.1 Reading Convention for Sequence Numbers . . . 83
9.2 Software-Triggered Transfer Sequences . . . 83
9.2.1 Sequence Diagrams . . . 83
9.3 `Scatter-gather` Sequence Diagrams . . . 85
9.3.1 Scatter-Gather Chain Setup and Transfer Start . . . 86
9.3.2 Regular Scatter-Gather Nominal Path . . . 88
9.3.3 Scatter-Gather CRC Failure Path . . . 88
9.3.4 Recovery After Scatter-Gather Runtime Error . . . 90
9.4 Hardware-Triggered Transfer with Trigger Gating . . . 91
9.5 Cross-Cutting Operation Sequence Diagrams . . . 91
9.5.1 Runtime Reconfiguration and Read-Modify-Write Workflow . . . 92
9.5.2 Observation Paths (Notification Channel vs Polling Channel) . . . 93
10 Configuration specification 95
10.1 How to read this chapter . . . 95
10.2 Containers and configuration parameters . . . 95
10.2.1 Dma . . . 95
10.2.2 Configuration Set . . . 96
10.2.2.1 Channel . . . 97
10.2.2.2 Transfer Control Descriptor . . . 104
10.2.2.3 Hardware Unit . . . 114
10.2.2.4 Hardware Specific Configuration . . . 115
10.2.3 General . . . 119
10.3 Semantic Constraints . . . 120
10.4 Published Information . . . 123
A Not applicable requirements 124
B Change history of AUTOSAR traceable items 126
B.1 Traceable item history of this document according to AUTOSAR Release R26-11 . . . 126

B.1.1 Added Specification Items in R26-11 126
B.1.2 Changed Specification Items in R26-11 133
B.1.3 Deleted Specification Items in R26-11 133
B.1.4 Added Constraints in R26-11 134
B.1.5 Changed Constraints in R26-11 134
B.1.6 Deleted Constraints in R26-11 134

1 Introduction and functional overview

This specification describes the functionality, API and configuration of the AUTOSAR Basic Software DMA MCAL Driver module (Dma).

The DMA MCAL Driver provides a hardware abstraction for Direct Memory Access controllers. It supports one or more DMA controller instances on a microcontroller and enables deterministic transfer of data between memory regions and peripheral interfaces with reduced CPU load, while keeping the upper layers independent from microcontroller-specific DMA details.

1.1 Scope

The scope of this specification includes:

- initialization and de-initialization of DMA hardware resources,
- configuration and control of `DMA channels` and transfer descriptors,
- start, stop and status handling of DMA transfers,
- notification of transfer completion and transfer errors,
- interaction with AUTOSAR error handling concepts.

The scope excludes direct specification of MCU- or vendor-specific register programming sequences. These details are handled by implementation and configuration for the target platform.

1.2 Functional Overview

The Dma module operates as an MCAL service that is typically used by complex drivers and upper-layer BSW modules requiring high-throughput data movement. After module initialization, configured `DMA channels` can be prepared and started for memory-to-memory, peripheral-to-memory, memory-to-peripheral, or peripheral-to-peripheral transfers, depending on hardware capabilities.

Transfer completion is detected by interrupt or polling mechanisms. In case of errors, the module reports failures according to AUTOSAR error classification and allows upper layers to restore a defined operating state.

Figure 1.1 illustrates the DMA concept. The DMA controller is a bus master on the same system bus as the CPU and gains access to memory and peripherals through bus arbitration. The CPU configures a transfer and starts it through the DMA controller's register interface. Peripherals signal readiness for a transfer via dedicated hardware request (trigger) lines to the DMA controller, which are separate from the

data path. Once a transfer completes, the DMA controller notifies the CPU by interrupt.

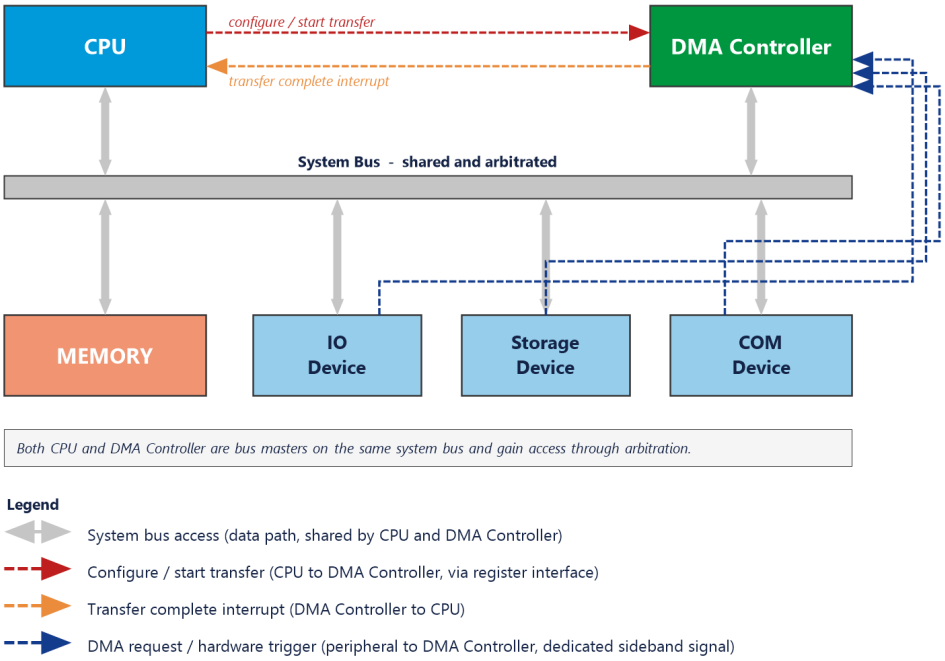


Figure 1.1: DMA functional concept overview

The DMA concept shown in Figure 1.1 provides the following key benefits:

- Reduced CPU load: the CPU configures transfer parameters but does not participate in data movement, remaining available for other tasks.
- Flexible triggering: transfers can be initiated by software commands or by hardware events (for example, ADC completion, timer events).
- Multi-pipeline capability: depending on hardware, multiple transfer pipelines can execute pseudo-parallel data flows to and from memory.
- Channel independence: multiple `DMA channels` operate independently, enabling concurrent data movement flows, sharing the system bus through arbitration.

1.3 Typical Use Cases

Common DMA usage patterns include:

- Memory-to-memory transfers: RAM-to-RAM and volatile-to-non-volatile memory copies (FLS, EEP, RamTest).
- Peripheral-to-memory transfers: data acquisition from ADC, capture units, and sensors (ADC, ICU).

- Memory-to-peripheral transfers: output generation and high-bandwidth communication (PWM, SPI, I2C, OCU).

1.4 Typical Not Supported Use Cases

Some use cases that may appear related to DMA transfers are not covered by this specification, because the respective peripheral controller integrates its own private DMA engine that is configured and managed by the corresponding peripheral driver rather than by the Dma module. For example:

- Ethernet (ETH): Ethernet controllers typically integrate a private, descriptor-based DMA engine for frame transfer, which is managed by the Eth Driver and is therefore not in scope of this specification.

2 Definition of acronyms and abbreviations

2.1 Acronyms and abbreviations

The table below contains all acronyms, abbreviations or initialisms that are component-specific and not available in the AUTOSAR glossary [1].

Acronym/ Abbreviation	Description
DSIZE	Destination transfer element size (hardware register field)
SSIZE	Source transfer element size (hardware register field)
TCD	Transfer Control Descriptor

Table 2.1: Acronyms and Abbreviations

The table below contains all document-independant acronyms or abbreviations that are defined in the AUTOSAR glossary [1].

Acronym/ Abbreviation	Description
BSW	Basic Software
CRC	Cyclic Redundancy Check
DET	Default Error Tracer, supports development and runtime error reporting, see [2, Specification of Default Error Tracer]
DMA	Direct Memory Access
DTC	Diagnostic Trouble Code
ECU	Electronic Control Unit
ISR	Interrupt Service Routine
MCAL	Microcontroller Abstraction Layer
MCU	MicroController Unit
RTE	Run-Time Environment, provides the infrastructure services that enable communication to occur between AUTOSAR SWCs as well as acting as the means by which AUTOSAR SWCs access basic software modules including the OS and communication service. This includes the RTE's responsibility for the realization of the scheduling of the SWCs. See [3, Specification of RTE Software]
SOME/IP	Scalable Service-Oriented Middleware over IP
SWC/SW-C	Software Component

Table 2.2: Acronyms and Abbreviations used from AUTOSAR Glossary

2.2 Definition of terms

The table below contains all definitions of terms that are used in this document.

Terms	Description
Data width adaptation	Hardware-supported packing or unpacking behavior applied when source and destination transfer widths differ.
DMA channel	Logical transfer resource of the DMA controller that executes data movement configured by a mapped TCD.
Hardware-triggered transfer	DMA transfer on a `DMA channel` whose `DmaChannelOperationMode` is `DMA_CHANNEL_MODE_HARDWARE`, started by a configured hardware event from a peripheral or device.
Scatter-gather	Execution mode where a channel automatically loads successive TCDs from memory to process descriptor sequences.
Software-triggered transfer	DMA transfer on a `DMA channel` whose `DmaChannelOperationMode` is `DMA_CHANNEL_MODE_SOFTWARE_POLLING` or `DMA_CHANNEL_MODE_SOFTWARE_INTERRUPT`, started by software service invocation through the DMA driver API.
TCD	Transfer Control Descriptor used by DMA hardware to define a transfer.
Transfer element	Smallest data unit handled per DMA transfer step according to configured source and destination transfer widths.

Table 2.3: Definition of terms in the scope of this Document

3 Related documentation

3.1 Input documents & related standards and norms

[1] Glossary
AUTOSAR_FO_TR_Glossary

[2] Specification of Default Error Tracer
AUTOSAR_CP_SWS_DefaultErrorTracer

[3] Specification of RTE Software
AUTOSAR_CP_SWS_RTE

[4] General Specification of Basic Software Modules
AUTOSAR_CP_SWS_BSWGeneral

3.2 Related specification

AUTOSAR provides a General Specification on Basic Software modules [4, SWS BSW General], which is also valid for Dma.

Thus, the specification SWS BSW General is considered as additional and required specification for Dma.

4 Constraints and assumptions

- The Dma module assumes a DMA hardware unit that supports channel-based configuration and trigger-controlled transfer execution.
- The exact number of channels, transfer descriptor formats, address alignment constraints, and supported transfer widths are hardware dependent.

4.1 Limitations

The following limitations apply to this specification:

- Direct use of the Dma module by application-layer SWCs is not supported. Memory protection constraints, task context restrictions, and configuration ownership across the BSW/application boundary are not resolved for that usage scenario.
- The static configuration requires correctly aligned configured source and destination addresses according to the configured transfer-element width and hardware-specific DMA alignment constraints.
- Fault categorization granularity is hardware dependent; therefore, fine-grained diagnostics beyond standardized channel state reporting may differ between implementations.
- DMA driver channel chaining is not supported. Channel chaining, which allows multiple independent `DMA channels` to be linked in sequence where one channel's completion triggers the next channel's transfer, is not implemented. Only `Scatter-gather` descriptor chaining is supported, which allows linking multiple transfer descriptors within a single `DMA channel` to perform complex data movement patterns without CPU intervention between descriptors.

5 Dependencies to other modules

The Dma module depends on the following AUTOSAR BSW infrastructure and platform services:

- MCU/startup and interrupt infrastructure for DMA interrupt routing and execution context setup,
- DET for development error reporting (when enabled),

The Dma module may be used by other MCAL, complex drivers, and BSW modules that require high-throughput data movement with reduced CPU intervention.

In addition, the RTE may use the Dma module to offload data movement within the middleware layer. A representative example is the copying of payload into a transformer buffer during SOME/IP serialization, where DMA transfers reduce CPU load on high-frequency or large-payload signals.

Direct access to the Dma module by application-layer SWCs is not supported by this specification. Memory protection constraints, task context restrictions, and configuration ownership across the BSW/application boundary have not been resolved for that usage scenario.

6 Requirements Tracing

The following tables reference requirements from applicable upstream specifications (for example CP_RS_DMA and CP_SWS_BSWGeneral) and link those requirements to their fulfillment in this document.

Requirement	Description	Satisfied by
[SRS_BSW_00331]	All Basic Software Modules shall strictly separate error and status information	[CP_SWS_Dma_00126]
[SRS_BSW_00369]	All AUTOSAR Basic Software Modules shall not return specific development error codes via the API	[CP_SWS_Dma_00126]
[SRS_BSW_00377]	A Basic Software Module can return a module specific types	[CP_SWS_Dma_00126]
[SRS_BSW_00441]	Naming convention for type, macro and function	[CP_SWS_Dma_00126]

Table 6.1: Requirements Tracing

7 Functional specification

This chapter specifies the required functional behavior of the DMA MCAL Driver. The requirements are written as AUTOSAR specification items and define externally visible module behavior.

7.1 Technical Functional Overview

7.1.1 DMA Driver Common Functionality

A DMA (Direct Memory Access) controller is a hardware peripheral capable of initiating data transfers from one or more source locations to one or more destination locations without direct CPU intervention during data movement.

The DMA module provides efficient transfer services via dedicated channels and supports the following transfer directions between memory (volatile or non-volatile) and memory-mapped devices:

- MEM_TO_MEM,
- MEM_TO_DEV,
- DEV_TO_MEM,
- DEV_TO_DEV.

Support for logical or virtual channels allows multiple users to access DMA services, including multi-core or partitioned system contexts. Additional capabilities include Scatter-gather operation.

7.1.2 DMA Channels and Transfer Elements

A DMA controller may include one hardware unit with one or more channels that can operate independently. Each DMA transfer is described by a TCD (Transfer Control Descriptor). A DMA channel is associated with a transfer by loading the corresponding TCD.

A transfer can be part of a linked transfer sequence consisting of multiple DMA transfers. Each transfer itself is composed of Transfer elements as defined by the configured transfer length in the TCD.

Each channel copies data from a source address to a destination address one Transfer element at a time until all elements are transferred, the transfer is aborted by software, or a hardware error occurs. Element widths (for example, 8-bit, 16-bit, 32-bit, or 64-bit) depend on hardware capabilities and may differ between source and destination interfaces.

The effective number of internal bus transactions may differ from the configured element count depending on source and destination widths as well as bus and DMA engine implementation details.

7.1.3 DMA Transfers Initiated by Hardware

The DMA driver supports transfers triggered by hardware events generated by peripherals or devices (for example ADC, timers, or communication peripherals); these are Hardware-triggered transfers.

For Hardware-triggered transfers, completion and error observation is supported via interrupt-based notification only. Polling is conceptually not reliable for this transfer mode because transfer progress and completion are driven by asynchronous external hardware events.

7.1.4 DMA Transfers Initiated by Software

The DMA driver supports transfers started by software services provided by the driver; these are Software-triggered transfers.

Software-initiated DMA transfers are asynchronous (non-blocking). After a transfer is triggered, program execution continues while the DMA engine processes the transfer.

For Software-triggered transfers, the DMA driver supports transfer state observation via polling and via interrupt-based notification.

Notification callbacks are bound to interrupt handling context and are executed only when interrupt-based notification is configured and enabled.

7.1.5 Scatter-Gather Capability

Scatter-gather mode allows a channel to load a new TCD from memory automatically, enabling execution of multiple transfer descriptors without CPU intervention between each descriptor load.

Typical use cases include scattering data to multiple destinations and gathering data from multiple sources using one physical DMA channel.

7.1.6 Descriptor Inner/Outer Loop Model

The inner-loop and outer-loop model is the baseline TCD execution model and is required for DMA transfer configuration, independent of whether Scatter-gather chaining is used.

The inner-loop and outer-loop control fields are part of the active DmaTcd configuration.

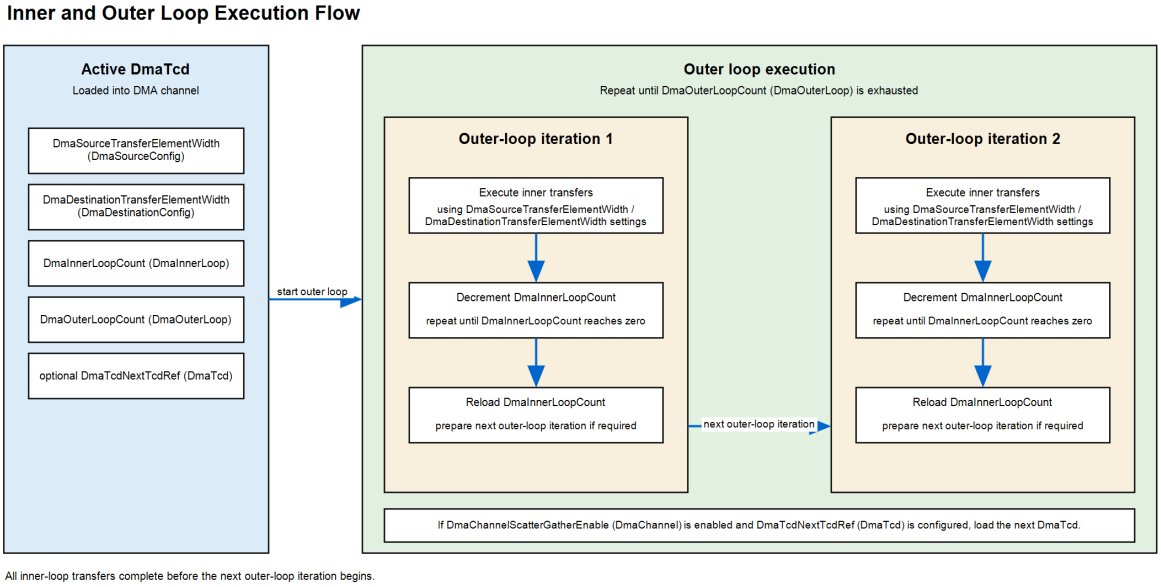


Figure 7.1: Inner and outer loop execution flow for one TCD

For this specification, the following loop semantics apply:

- Inner loop: one transfer step controlled by transfer size and inner-loop byte count.
- Outer loop: repetition count of the inner loop until transfer completion.

The DMA driver configures descriptor fields consistently such that inner loop execution and outer loop execution are unambiguous for the underlying hardware.

The execution flow shown in Figure 7.1 is interpreted as follows:

1. The DMA channel loads the active DmaTcd and initializes inner-loop and outer-loop control fields.
2. During one outer-loop iteration, the DMA controller performs all configured inner transfers and decrements the current inner-loop counter after each inner transfer.
3. When one outer-loop iteration completes, the DMA controller reloads the inner-loop counter for the next outer-loop iteration according to descriptor configuration.
4. After all outer-loop iterations complete, execution of the current DmaTcd ends.

The inner loop defines the transfer granularity within one outer-loop iteration, while the outer loop defines how often this inner-transfer pattern is repeated for the same DmaTcd. In practical terms, this separates transfer shape (inner-loop behavior) from repetition count (outer-loop behavior).

Inner/Outer-Loop Examples (informative)

- Example 1 (linear buffer copy): DmaSourceTransferElementWidth and DmaDestinationTransferElementWidth are both 4 bytes, initial Dma-

InnerLoopCount = 8, DmaOuterLoopCount = 1. The DMA controller performs 8 inner transfers of 4 bytes. Total payload for this DmaTcd is 32 bytes, and completion is reached after one outer-loop iteration.

- Example 2 (repeated frame transfer): DmaSourceTransferElementWidth and DmaDestinationTransferElementWidth are both 2 bytes, initial DmaInnerLoopCount = 16, DmaOuterLoopCount = 3. Each outer-loop iteration transfers 32 bytes (16 x 2 bytes). Since the outer loop runs three times, total payload is 96 bytes before advancing to the next DmaTcd, if configured.

Figure 7.2 illustrates the transfer element counts and resulting payload sizes for both examples.

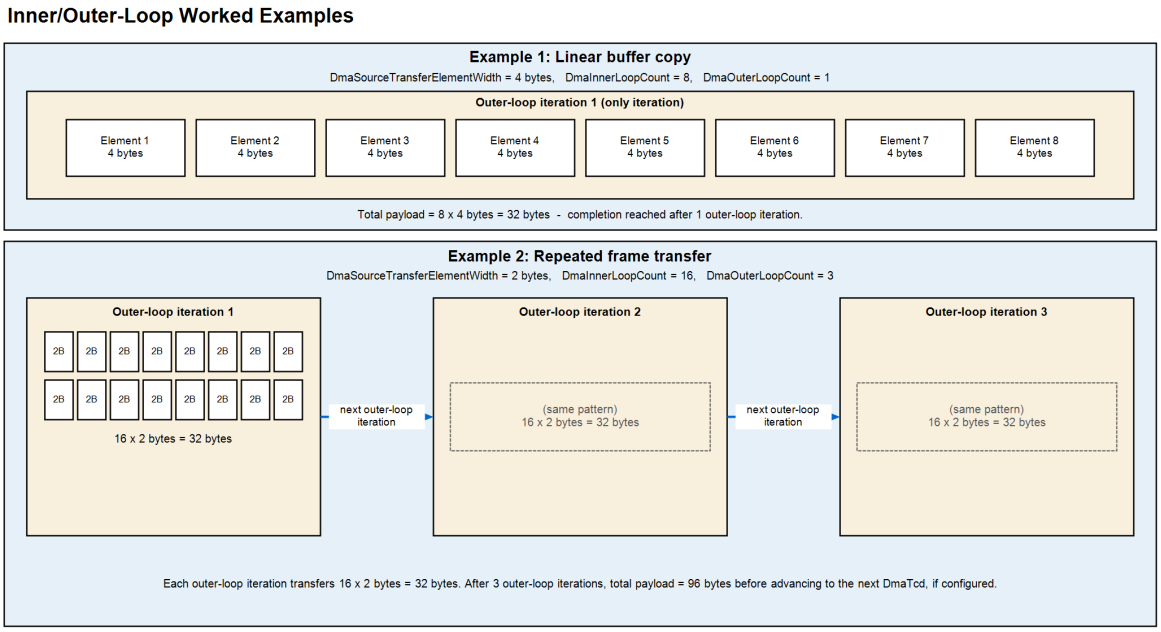


Figure 7.2: Inner/outer-loop worked examples for linear buffer copy and repeated frame transfer

7.1.7 Optional Scatter-Gather Descriptor Chaining

Scatter-gather descriptor chaining is an optional feature. If enabled, the DMA controller loads the next DmaTcd from memory after completion of the current DmaTcd, using the configured link information in the current DmaTcd.

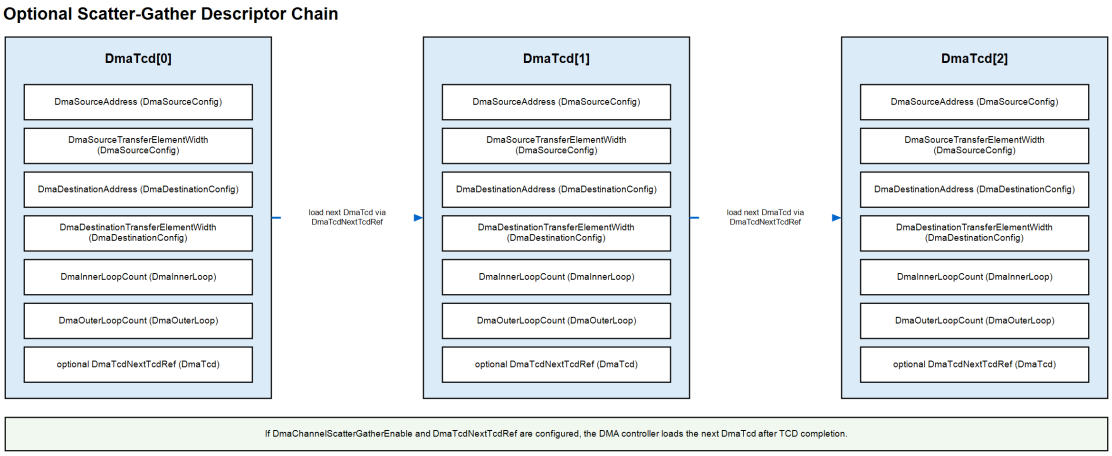


Figure 7.3: Optional `Scatter-gather` descriptor chain

The next-TCD link shown in Figure 7.3 corresponds to the configuration parameter `DmaTcdNextTcdRef`. Optional activation of descriptor chaining is controlled by `DmaChannelScatterGatherEnable`.

At runtime, the link of a `DmaTcd` to its successor is represented by the `DmaTcdId` of the successor (next-`TCD` link). It is provided to the application as `NextTcdIdx` by `Dma_GetTransferParameters`() and can be changed with `Dma_UpdateTransferParameters`(), see [CP_SWS_Dma_00392] and [CP_SWS_Dma_00393]. The value `DMA_TCD_IDX_NONE` ([CP_SWS_Dma_00390]) marks the end of the chain.

[CP_SWS_Dma_00391] Next TCD Link Initialization

Status: DRAFT

⌈During `Dma_Init`(), the Dma module shall initialize the next-`TCD` link of each configured `DmaTcd` with the `DmaTcdId` of the `DmaTcd` referenced by `DmaTcdNextTcdRef`, or with `DMA_TCD_IDX_NONE` if `DmaTcdNextTcdRef` is not configured.⌋

Note

A `Scatter-gather` chain may be cyclic, for example `TCD` $0 \rightarrow 2 \rightarrow 5 \rightarrow 1 \rightarrow 0$. In this case the DMA hardware processes the chain continuously until the transfer is stopped or disabled. Since every next-`TCD` link identifies a configured `DmaTcd`, the number of distinct `TCDs` in a chain is bounded by the number of configured `DmaTcd` containers. A check that walks along the chain can therefore stop when it reaches the end of the chain or a `TCD` that has already been visited.

If `Scatter-gather` chaining is not configured, execution of the current `DmaTcd` ends after the configured inner-loop and outer-loop counts complete.

Scatter-Gather Chaining Example (informative)

- First DmaTcd instance: DmaSourceTransferElementWidth and DmaDestinationTransferElementWidth of 1 byte each, DmaInnerLoopCount = 64, and DmaOuterLoopCount = 2, transferring 128 bytes total. Its DmaTcdNextTcdRef is configured, so after two outer-loop iterations the DMA controller loads a second DmaTcd instance automatically without CPU intervention.
- Second DmaTcd instance: DmaSourceTransferElementWidth and DmaDestinationTransferElementWidth of 4 bytes each, DmaInnerLoopCount = 5, and DmaOuterLoopCount = 1, transferring a further 20 bytes. Its DmaTcdNextTcdRef is not configured, so execution ends after this instance and no further DmaTcd is loaded.

Figure 7.4 illustrates the resulting payload of both DmaTcd instances and the end of the chain after the second instance.

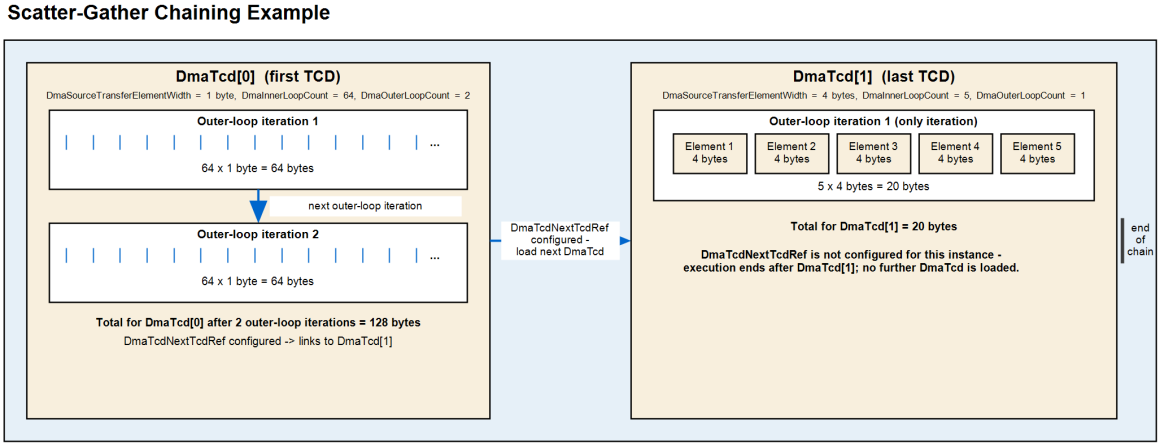


Figure 7.4: Scatter-gather chaining worked example: first DmaTcd instance chained to a second, last instance

7.1.8 Data Width Adaptation and Transfer Semantics

7.1.8.1 Scope

This subsection describes how the DMA Driver handles transfers where the configured source and destination data widths differ. The DMA controller may perform automatic Data width adaptation through data packing or unpacking based on the configured transfer attributes.

7.1.8.2 Functional Description

The DMA Driver configures the underlying hardware such that data transfers are performed in accordance with:

- Source transfer width (SSIZE)
- Destination transfer width (DSIZE)

If SSIZE and DSIZE differ, the DMA controller performs width adaptation as described below.

Address Handling:

- Source address increment follows the source width.
- Destination address increment follows the destination width per write operation.



Figure 7.5: Types of basic DMA transfers

As shown in Figure 7.5, the source side is the reference for counting `Transfer elements` (`DmaInnerLoopCount` counts source-side elements): if SSIZE is wider than DSIZE, one source element is unpacked into multiple destination elements; if SSIZE is narrower than DSIZE, multiple source elements are packed into one destination element. The applicable width ratio is defined in [CP_SWS_Dma_CONSTR_00005].

7.2 Detailed Functional Specification

7.2.1 Module State Machine

Figures 7.6, 7.7, 7.8, and 7.9 illustrate the module and channel state models used by this specification.

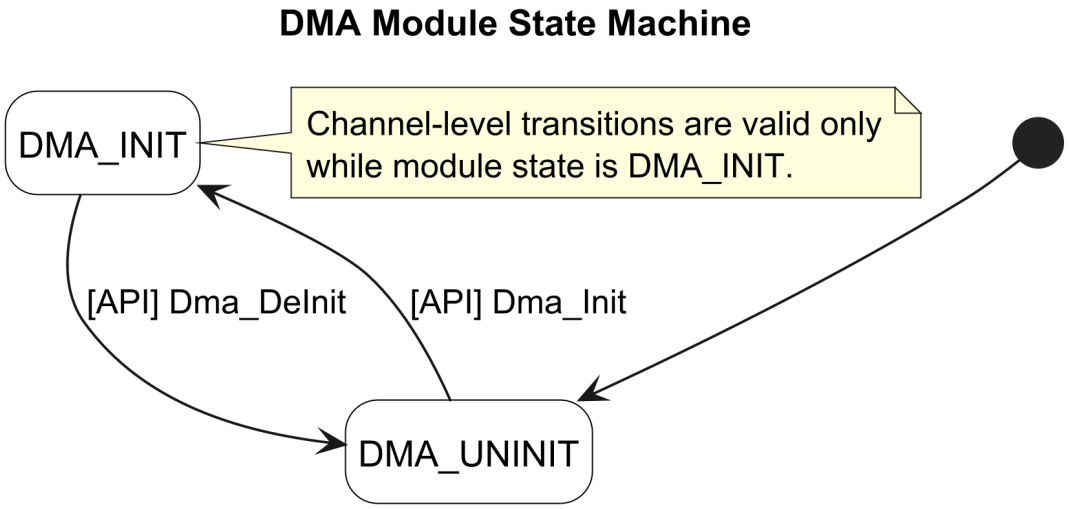


Figure 7.6: DMA module state machine overview

The following state semantics apply:

In addition to the module state, each configured channel maintains an independent state (see Figures 7.7, 7.8, and 7.9). Channel states evolve independently of each other: a hardware-level DMA error is detected and latched per channel, so only the affected `DMA channel` transitions to `DMA_CHANNEL_HARDWARE_ERROR` ([CP_SWS_Dma_00206]) while other channels continue to operate. However, the underlying DMA hardware cannot be reliably restored to a defined state by reinitializing the affected channel alone; safe recovery requires reinitializing the DMA peripheral as a whole via `Dma_DeInit`() followed by `Dma_Init`() ([CP_SWS_Dma_00002]). In this sense, fault detection is channel-scoped, but recovery is module-scoped.

For `Software-triggered transfers`, completion, transfer-error, and hardware-error outcomes can be observed either via polling (Figure 7.7) or via interrupt-driven notifications (Figure 7.8). Both diagrams show the identical set of base transitions (start, stop, suspend/resume); they differ only in how the driver observes and reports the end of a transfer.

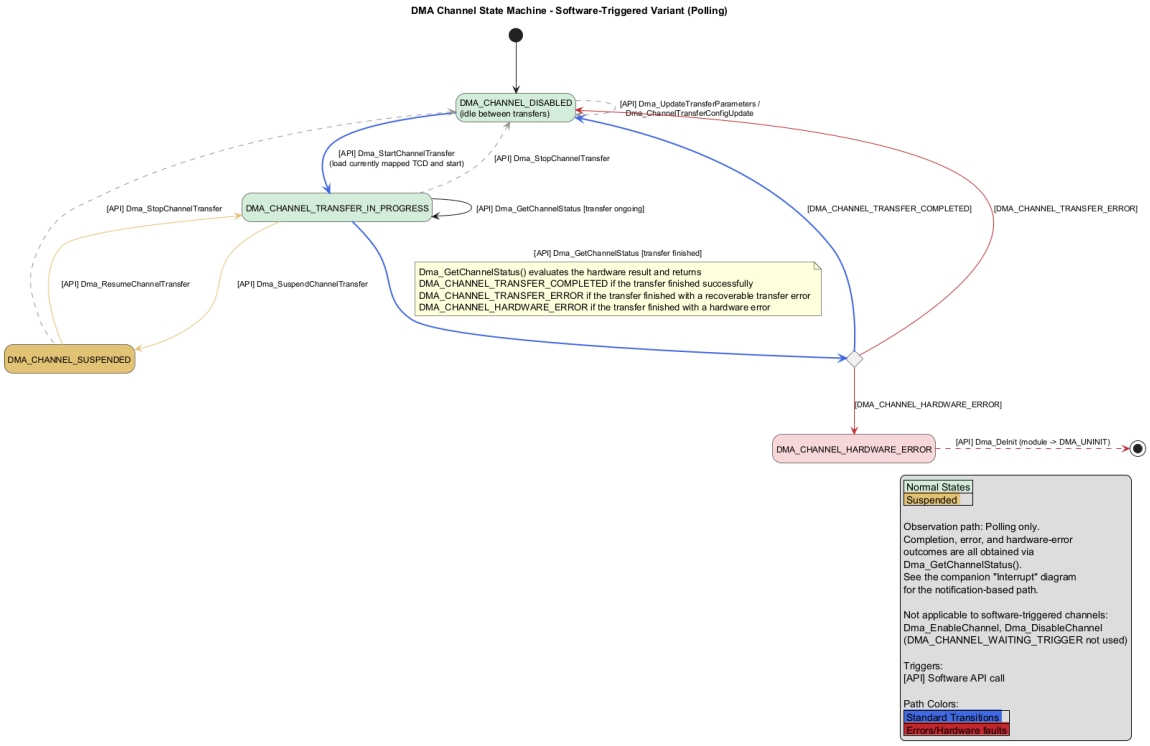


Figure 7.7: `DMA channel` state machine for `Software-triggered transfers`: includes suspend/resume and polling-based completion observation via GetChannelStatus

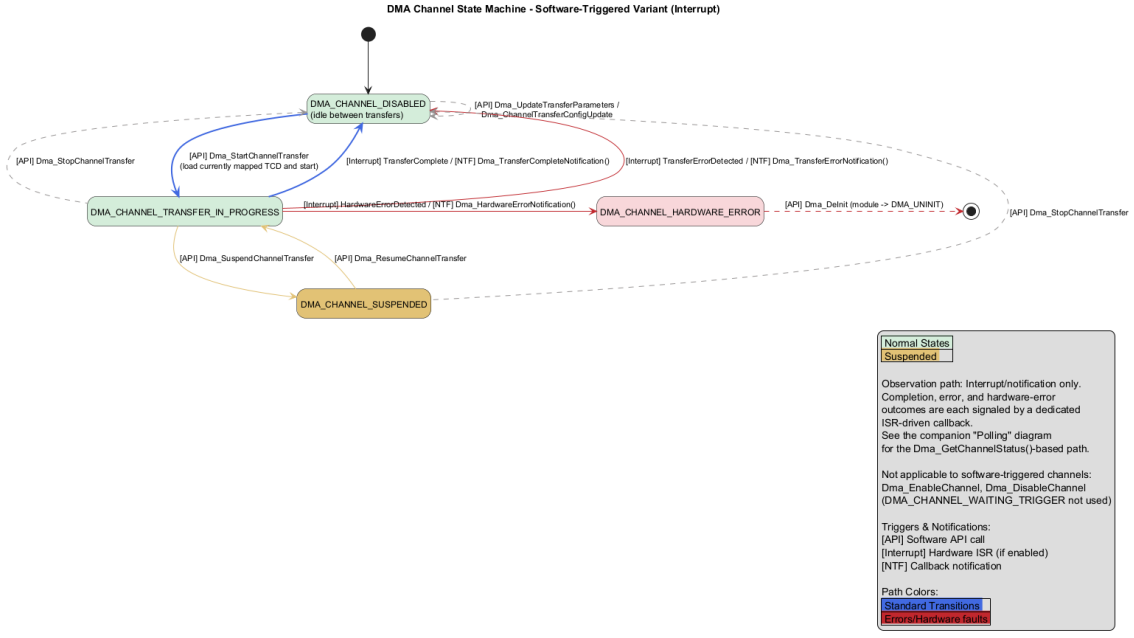


Figure 7.8: `DMA channel` state machine for `Software-triggered transfers`: includes suspend/resume and interrupt-driven completion observation via notification callbacks

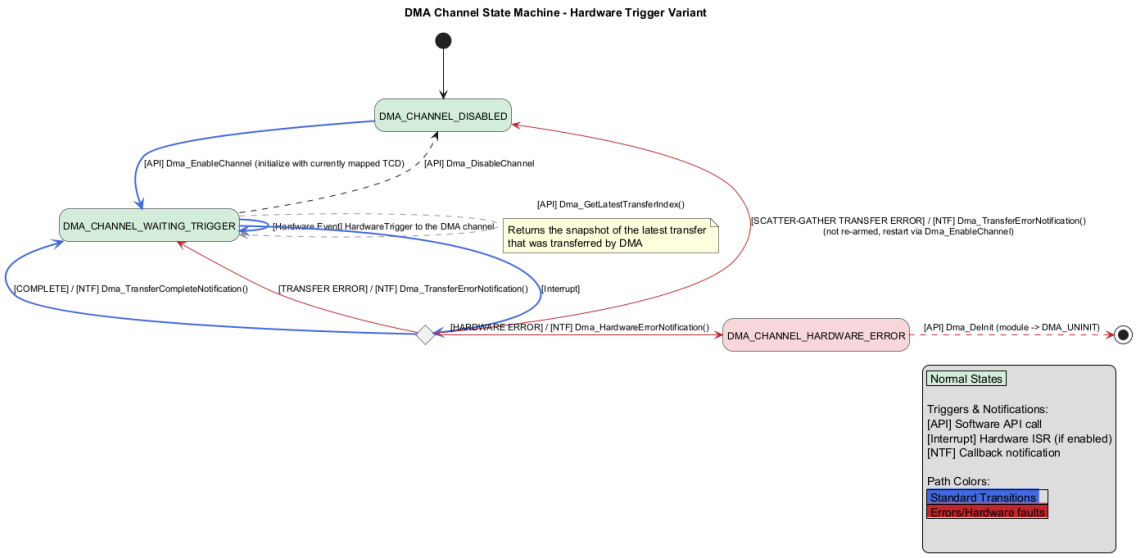


Figure 7.9: `DMA channel` state machine for `Hardware-triggered transfers`: simplified two-state model with notification-only completion

7.2.2 Suspend and Resume Scope Explanation

This subsection is informative. It first motivates typical use cases for suspending and resuming a channel transfer, then explains why the suspend and resume services are restricted to `Software-triggered transfers` in this specification although some DMA hardware implementations may provide channel hold or suspend mechanisms also for peripheral-triggered operation.

Typical Use Cases (informative)

Suspending a channel transfer instead of stopping it preserves the DMA context (source address, destination address, remaining element count, and descriptor position), so that a later `Dma_ResumeChannelTransfer`() call can continue the transfer exactly where it left off, without re-fetching or restarting the transfer descriptor. Typical use cases include:

- **Transfer prioritization**: A lower-priority, long-running transfer (for example a large MEM_TO_MEM background copy) is temporarily suspended to free system bus bandwidth for a more time-critical transfer on another channel, and resumed once the higher-priority transfer has completed.
- **Cooperative bus-load management**: A large transfer is deliberately split into time-sliced segments by repeatedly suspending and resuming it, avoiding prolonged monopolization of the system bus by a single channel.

- **Low-power mode preparation**: An in-progress transfer is suspended before the ECU enters a low-power mode so that its DMA context is preserved, and resumed after wake-up without restarting the transfer from the beginning.
- **Diagnostic inspection**: A transfer is paused to allow safe inspection of transfer progress (for example via `Dma_GetLatestTransferIndex`()) before deciding whether to continue, without losing the accumulated transfer state.

These use cases apply to `Software-triggered transfers`, for which suspend and resume are guaranteed by this specification. The remainder of this subsection explains why the same guarantee cannot be given for `Hardware-triggered transfers`.

For a `Software-triggered transfer`, the DMA Driver initiates execution through `Dma_StartChannelTransfer`() and controls the logical start of the transfer. In this case, suspend and resume can be specified with a clear API semantics limited to the `DMA channel` execution context.

For a `Hardware-triggered transfer`, the transfer progress depends not only on the `DMA channel` state but also on the behavior of the connected peripheral, trigger source, FIFO depth, and integration-specific trigger gating. Suspending the `DMA channel` alone does not generally guarantee preservation of the end-to-end transfer path state.

Typical implications are:

- For DEV_TO_MEM transfers, the peripheral may continue to produce data while the `DMA channel` is suspended. This can cause FIFO overflow or data loss.
- For MEM_TO_DEV transfers, the peripheral may continue to consume data while the `DMA channel` is suspended. This can cause FIFO underrun, frame truncation, or protocol-specific transfer errors.
- Even if the DMA hardware preserves source address, destination address, remaining transfer count, and descriptor position, the connected peripheral state may still no longer be restartable in a portable way.

Therefore, this specification limits the guaranteed suspend/resume semantics to `Software-triggered transfers`. Support for suspending `Hardware-triggered transfers`, including any required peripheral pausing or trigger masking, depends on the underlying hardware and system integration and is outside the guaranteed semantics of this DMA Driver API.

Examples (informative)

- Example 1: MEM_TO_MEM `Software-triggered transfer`. The application starts the transfer by calling `Dma_StartChannelTransfer`(). If `Dma_SuspendChannelTransfer`() is then called, the DMA controller may preserve the current source address, destination address, remaining element count, and

descriptor position. A later call to Dma_ResumeChannelTransfer() can continue the transfer from that preserved DMA context.

- Example 2: DEV_TO_MEM Hardware-triggered transfer. An ADC or UART continues to generate requests or data while the DMA channel is suspended. Even if the DMA channel context is preserved, the peripheral-side FIFO can overflow and received data can be lost. Therefore, a portable suspend/resume guarantee cannot be derived from the DMA channel state alone.
- Example 3: MEM_TO_DEV Hardware-triggered transfer. A communication peripheral may continue transmission timing while the DMA channel is suspended. If no new data is supplied, FIFO underrun or protocol-specific faults can occur. In such a case, resuming the DMA channel alone does not necessarily restore a valid end-to-end transfer state.

7.2.3 Channel State Machine Variants: Software-Triggered vs Hardware-Triggered

This subsection is informative. It explains the rationale behind the two channel-level state machine variants; the corresponding normative behavior is specified by the SWS items in Section 7.2 and by the API-level items in Chapter 8.

This specification distinguishes between two channel-level state machine variants based on the transfer trigger model: Software-triggered transfers and Hardware-triggered transfers. Which variant applies to a given DMA channel is selected by the configuration parameter DmaChannelOperationMode: a value of DMA_CHANNEL_MODE_SOFTWARE_POLLING or DMA_CHANNEL_MODE_SOFTWARE_INTERRUPT configures a software-triggered channel (differing only in the completion/error observation path), while DMA_CHANNEL_MODE_HARDWARE configures a hardware-triggered channel.

7.2.3.1 Software-Triggered Variant

For Software-triggered transfers (DmaChannelOperationMode is DMA_CHANNEL_MODE_SOFTWARE_POLLING or DMA_CHANNEL_MODE_SOFTWARE_INTERRUPT; Figures 7.7 and 7.8), the application initiates transfer execution via the Dma_StartChannelTransfer() API. The driver maintains complete control over transfer progression and can offer the full feature set:

- **Single-Step Start**: Dma_StartChannelTransfer() loads the current channel-to-TCD mapping and starts the transfer in one call. DMA_CHANNEL_DISABLED is the idle state before and after each transfer, so Dma_EnableChannel() and Dma_DisableChannel() are not used for software-triggered channels.

- **Configurable Observation Path**: The observation path is selected per channel at pre-compile time by DmaChannelOperationMode and cannot be switched at runtime:
  - DMA_CHANNEL_MODE_SOFTWARE_POLLING: completion, recoverable errors, and hardware faults are observed via Dma_GetChannelStatus() without requiring interrupt infrastructure ([CP_SWS_Dma_00312]).
  - DMA_CHANNEL_MODE_SOFTWARE_INTERRUPT: completion, recoverable errors, and hardware faults are reported via the notification callbacks, which are enabled by default after Dma_Init() ([CP_SWS_Dma_00380]). Dma_GetChannelStatus() can still be used to check whether a transfer is ongoing, but the transfer result is delivered by the callback.

  In both cases, Dma_GetLatestTransferIndex() can be used to obtain the current transfer progress.
- **Suspend and Resume**: The driver can halt transfer execution at any point via Dma_SuspendChannelTransfer() and restore it via Dma_ResumeChannelTransfer(). The DMA channel context (source address, destination address, element count, descriptor position) is preserved and predictable because the application controls the transfer lifecycle.

7.2.3.2 Hardware-Triggered Variant

For Hardware-triggered transfers (DmaChannelOperationMode is DMA_CHANNEL_MODE_HARDWARE; Figure 7.9), the transfer is initiated by external hardware events (e.g., ADC completion, timer pulse, communication peripheral request). The DMA channel state model is simplified to reflect the reduced CPU control:

- **Trigger Arming**: Dma_EnableChannel() loads the current channel-to-TCD mapping and arms the hardware request line; Dma_DisableChannel() disarms it. Dma_StartChannelTransfer() and Dma_StopChannelTransfer() are not used for hardware-triggered channels.
- **Operational States**: During normal hardware-triggered operation, channels transition between DMA_CHANNEL_DISABLED and DMA_CHANNEL_WAITING_TRIGGER. Transfer execution is driven entirely by hardware; no intermediate DMA_CHANNEL_TRANSFER_IN_PROGRESS state is exposed because the CPU does not initiate or directly control the transfer.
- **Latched Hardware-Error State**: A non-recoverable hardware fault moves the channel to DMA_CHANNEL_HARDWARE_ERROR. This state is latched and can only be left by Dma_DeInit(), because the underlying DMA peripheral is assumed to require full re-initialization before further use.
- **Notification-Based Observation**: Hardware-triggered completion and error observation uses interrupt-based callbacks (Dma_TransferCompleteNotification(), Dma_TransferErrorNotification(), Dma_Hardware-

ErrorNotification()). For progress information, Dma_GetLatestTransferIndex() remains available.

- **No Suspend or Resume**: Hardware-triggered transfers cannot be reliably suspended at the DMA channel level alone. Suspending the DMA channel does not suspend the connected peripheral's data production or consumption, leading to FIFO overflow/underrun and potential data loss. Therefore, suspend and resume services are not applicable.
- **Rationale for Simplified State Model**: The reduction in state complexity reflects the elimination of CPU-driven control. Since the CPU does not explicitly start or stop the Hardware-triggered transfer, exposing an intermediate DMA_CHANNEL_TRANSFER_IN_PROGRESS state would be misleading. Completion is observed asynchronously, while a non-recoverable hardware fault is represented explicitly by the latched DMA_CHANNEL_HARDWARE_ERROR state.

7.2.3.3 API Restriction: GetChannelStatus for Hardware-Triggered Channels

For hardware-triggered channels, the Dma_GetChannelStatus() API is not applicable because:

- The transfer is driven by asynchronous hardware events, not CPU requests.
- Between successive status polling calls, the channel state may change (e.g., transfer completes, error occurs) without CPU notification.
- Polling-based observation implies synchronous CPU execution; Hardware-triggered transfers are asynchronous and event-driven.
- The intended observation path is interrupt-based completion/error notification, not CPU-driven status polling.

For Hardware-triggered transfers, transfer completion and error conditions are observed through interrupt notifications. Current transfer progress can still be obtained through Dma_GetLatestTransferIndex().

7.2.3.4 Channel Lifecycle APIs per Trigger Mode

The channel lifecycle APIs are split by trigger mode:

- Software-triggered channels use Dma_StartChannelTransfer() and Dma_StopChannelTransfer(). The start loads the current TCD and triggers the transfer in one step.
- Hardware-triggered channels use Dma_EnableChannel() and Dma_DisableChannel(). The enable loads the current TCD and arms the hardware request line; the transfers themselves are started by the peripheral.

Table 7.1 summarizes the applicability of the channel APIs per trigger mode.

API	Software-triggered channel	Hardware-triggered channel
Dma_EnableChannel()	Not applicable ([CP_SWS_Dma_00373])	DISABLED → WAITING_TRIGGER, loads TCD ([CP_SWS_Dma_00003]); rejected in any other state ([CP_SWS_Dma_00379])
Dma_DisableChannel()	Not applicable ([CP_SWS_Dma_00374])	WAITING_TRIGGER → DISABLED ([CP_SWS_Dma_00014]); rejected in HARDWARE_ERROR ([CP_SWS_Dma_00378])
Dma_StartChannelTransfer()	DISABLED → TRANSFER_IN_PROGRESS, loads TCD ([CP_SWS_Dma_00004])	Not applicable ([CP_SWS_Dma_00375])
Dma_StopChannelTransfer()	TRANSFER_IN_PROGRESS or SUSPENDED → DISABLED ([CP_SWS_Dma_00005]); rejected in HARDWARE_ERROR ([CP_SWS_Dma_00377])	Not applicable ([CP_SWS_Dma_00376])
Dma_SuspendChannelTransfer(), Dma_ResumeChannelTransfer()	Applicable ([CP_SWS_Dma_00015], [CP_SWS_Dma_00016])	Not applicable ([CP_SWS_Dma_00371], [CP_SWS_Dma_00372])
Dma_GetChannelStatus()	Applicable ([CP_SWS_Dma_00006])	Not applicable ([CP_SWS_Dma_00334])
Dma_EnableChannelNotifications(), Dma_DisableChannelNotifications()	Only in DISABLED ([CP_SWS_Dma_00381], [CP_SWS_Dma_00382]); enabled by default after init ([CP_SWS_Dma_00380]); start rejected while disabled ([CP_SWS_Dma_00383])	Only in DISABLED ([CP_SWS_Dma_00381], [CP_SWS_Dma_00382]); enabled by default after init ([CP_SWS_Dma_00380]); enable rejected while disabled ([CP_SWS_Dma_00384])
Dma_UpdateTransferParameters(), Dma_ChannelTransferConfigUpdate()	In DISABLED, which is the idle state between transfers	In DISABLED, after Dma_DisableChannel()

Table 7.1: Channel lifecycle APIs per trigger mode

7.2.3.5 API Applicability per Channel Operation Mode

This subsection is informative. Table 7.2 lists all services and callbacks of the Dma module and shows for which value of `DmaChannelOperationMode` they are used: hardware-triggered (`DMA_CHANNEL_MODE_HARDWARE`), software-triggered with polling (`DMA_CHANNEL_MODE_SOFTWARE_POLLING`), and software-triggered with notification (`DMA_CHANNEL_MODE_SOFTWARE_INTERRUPT`). Entries marked **Yes (main flow)** form the nominal sequence of a transfer in the respective operation mode; the other services are optional or used for reconfiguration, abort, and diagnosis. "No" means that the call is rejected or the callback is not configured. The specification items and state preconditions are given in Table 7.1 and in Chapter 8.

Service / callback	Hardware-triggered	Software-triggered, polling	Software-triggered, notification
Module services			
`Dma_Init`()	**Yes (main flow)**	**Yes (main flow)**	**Yes (main flow)**
`Dma_DeInit`()	**Yes (main flow)**	**Yes (main flow)**	**Yes (main flow)**
Channel lifecycle			
`Dma_EnableChannel`()	**Yes (main flow)**	No	No
`Dma_DisableChannel`()	Yes, before reconfiguration	No	No
`Dma_StartChannelTransfer`()	No	**Yes (main flow)**	**Yes (main flow)**
`Dma_StopChannelTransfer`()	No	Yes, to abort a transfer	Yes, to abort a transfer
`Dma_SuspendChannelTransfer`(), `Dma_ResumeChannelTransfer`()	No	Yes, if supported	Yes, if supported
Observation			
`Dma_GetChannelStatus`()	No	**Yes (main flow)**	Yes, only to check whether a transfer is ongoing
`Dma_GetLatestTransferIndex`()	Yes	Yes	Yes
`Dma_GetCrcResult`()	Yes, if CRC is configured	Yes, if CRC is configured	Yes, if CRC is configured
`Dma_EnableChannelNotifications`(), `Dma_DisableChannelNotifications`()	Yes, enabled by default after init	No	Yes, enabled by default after init

▽

△

Descriptor handling			
Dma_GetTransfer-Parameters()	Yes	Yes	Yes
Dma_Update-Transfer-Parameters(), Dma_Channel-TransferConfig-Update()	Yes, after Dma_Disable-Channel()	Yes, between transfers	Yes, between transfers
Callbacks (configurable interfaces)			
Dma_Transfer-Complete-Notification(), Dma_TransferError-Notification(), Dma_HardwareError-Notification()	**Yes (main flow)**	No	**Yes (main flow)**

Table 7.2: API applicability per channel operation mode

7.2.4 Partition/Core Access Handling

[CP_SWS_Dma_00364] Wrong Partition or Core Access Rejection Behavior

Status: DRAFT

⌈

If DmaChannelEcucPartitionRef is configured and a DMA API with a ChannelId input parameter is called from a partition/core context that does not match the configured DmaChannelEcucPartitionRef of the referenced channel, the Dma module shall reject the request, shall not modify the driver-managed channel state, and shall report a development error when DmaDevErrorDetect is enabled. The reported development error shall be DMA_E_WRONG_PARTITION.⌋

[CP_SWS_Dma_00319] DMA Channel Partition Request Scope

Status: DRAFT

⌈

For each configured DmaChannel, the Dma module shall determine the partition/core request scope from DmaChannelEcucPartitionRef as follows:

- If DmaChannelEcucPartitionRef is configured, the Dma module shall only process requests that belong to the configured partition/core context. For rejection behavior and development error reporting in case of a partition/core mismatch, see [CP_SWS_Dma_00364].

- If DmaChannelEcucPartitionRef is not configured, the Dma module shall not restrict request processing by partition/core context.

⌋

7.2.5 Channel and Hardware Unit Configuration Scope

[CP_SWS_Dma_00034] DMA Channel Configuration Scope

Status: DRAFT

⌈

For each configured DMA channel, the Dma module shall apply only channel configuration entries assigned to the corresponding DmaChannel, including the configured channel-to-TCD mapping (DmaChannelTcdRef).⌋

[CP_SWS_Dma_00320] DMA Hardware Unit Configuration Independence

Status: DRAFT

⌈

For each configured DmaHwUnit, the Dma module shall keep the configuration of the DMA channels mapped to it via DmaChannelHwUnitRef independent from other DMA hardware units.⌋

7.2.6 Runtime Transfer Reconfiguration APIs

This subsection is informative. It explains the intended use of Dma_UpdateTransferParameters() and Dma_ChannelTransferConfigUpdate().

The static ECUC configuration is typically validated offline by AUTOSAR tooling and by configuration constraints. Therefore, runtime configuration checks are primarily relevant when software modifies transfer-related information after initialization.

The following API roles are distinguished:

- Dma_UpdateTransferParameters() updates runtime-updatable fields of the currently addressed DmaTcd, for example DmaSourceAddress, DmaDestinationAddress, loop counters, width-related transfer attributes, or the next-TCD link (NextTcdIdx).
- Dma_ChannelTransferConfigUpdate() changes the channel-to-TCD assignment by updating which configured DmaTcd is referenced by DmaChannelTcdRef.

Typical use cases are:

- updating DmaSourceAddress and DmaDestinationAddress at runtime while keeping the same overall transfer structure,

- adapting loop-count related transfer lengths for a new data block,
- remapping a channel to another preconfigured DmaTcd for a different transfer profile,
- switching between alternative transfer setups without re-initializing the complete DMA driver.

Example (informative)

An application may configure a transfer once during initialization and later use Dma_UpdateTransferParameters() to adjust only DmaSourceAddress and DmaDestinationAddress for the next buffer to be copied. If a different preconfigured transfer layout is needed, the application may instead use Dma_ChannelTransferConfigUpdate() to select another configured DmaTcd.

7.2.7 Transfer Control Descriptor Completeness

Since the transfer parameters can be provided at runtime, the containers DmaSourceConfig, DmaDestinationConfig, DmaInOutLoopConfig and DmaInnerLoop are optional in the ECU configuration. A transfer can only be started or enabled once all mandatory transfer parameters are available, see [CP_SWS_Dma_00385], [CP_SWS_Dma_00386] and [CP_SWS_Dma_00387].

[CP_SWS_Dma_00385] Complete Transfer Control Descriptor

Status: DRAFT

⌈A DmaTcd is complete if a value is available for each of the following transfer parameters, either from the ECU configuration or from a previous successful call of Dma_UpdateTransferParameters() for this DmaTcd:

- DmaSourceAddress,
- DmaSourceTransferElementWidth,
- DmaDestinationAddress,
- DmaDestinationTransferElementWidth, and
- DmaInnerLoopCount.

DmaOuterLoopCount is not required; if no value is available, no outer-loop repetition is performed.⌋

Note

Completeness is determined by whether a value has been provided, not by the value itself, since for example an address value of 0 can be a valid address on some microcontrollers. How the Dma module keeps track of provided values is implementation-specific.

7.2.8 Inner and Outer Loop Execution

[CP_SWS_Dma_00028] Inner and Outer Loop Execution Semantics

Status: DRAFT

⌈

During execution of a configured DmaTcd, the Dma module shall configure and supervise the channel such that all configured inner-loop transfers and outer-loop repetitions are completed according to DmaInnerLoopCount and DmaOuterLoopCount. DmaInnerLoopCount counts source-side Transfer elements; if the destination-side transfer element width differs from the source-side width, the number of destination-side elements written per inner-loop pass differs accordingly, per the width ratio defined in [CP_SWS_Dma_CONSTR_00005].⌋

7.2.9 Scatter-Gather Execution

Scatter-Gather Chain Validation (informative)

A Scatter-gather chain is validated at the following points in time:

- At configuration time: DmaTcdNextTcdRef is only configured for Scatter-gather channels, and all DmaTcd containers of a chain are assigned to the same partition as the channel ([CP_SWS_Dma_CONSTR_00006], [CP_SWS_Dma_CONSTR_00007]).
- When a next-TCD link is set via Dma_UpdateTransferParameters(): only this link is checked. All channels using the addressed DmaTcd are in DMA_CHANNEL_DISABLED ([CP_SWS_Dma_00326]), Scatter-gather is enabled for these channels ([CP_SWS_Dma_00370]), and the target is a configured DmaTcd of the same partition ([CP_SWS_Dma_00393]).
- Before transfer start (Dma_StartChannelTransfer() or Dma_EnableChannel()): the complete chain, starting at the DmaTcd of the current channel-to-TCD mapping, is checked for completeness ([CP_SWS_Dma_00386], [CP_SWS_Dma_00387]), address and alignment validity ([CP_SWS_Dma_00313]), width and loop consistency ([CP_SWS_Dma_00314]), and descriptor-chain consistency ([CP_SWS_Dma_00027]). The check ends at the end of the chain or at a TCD that has already been visited (cyclic chain).

- During execution: the DMA hardware loads the next descriptor autonomously; the Dma module does not check the chain at this point. Errors detected by the hardware are handled as runtime transfer errors ([CP_SWS_Dma_00029], [CP_SWS_Dma_00030]).

Since the descriptors and next-TCD links of a chain cannot be changed while a channel using them is active, the chain processed by the hardware is the chain validated before transfer start.

[CP_SWS_Dma_00027] Scatter-Gather Descriptor Validation

Status: DRAFT

⌈

If Scatter-gather is configured for a channel, the Dma module shall validate descriptor-chain configuration before transfer start. Relevant configuration parameters are (DmaChannelScatterGatherEnable), (DmaChannelTcdRef), and (DmaTcdNextTcdRef), together with the current next-TCD links ([CP_SWS_Dma_00391], [CP_SWS_Dma_00392]).⌋

[CP_SWS_Dma_00313] Scatter-Gather Descriptor Address and Alignment Validation

Status: DRAFT

⌈

If Scatter-gather is configured for a channel, the Dma module shall validate descriptor address validity and alignment constraints before transfer start, based on source and destination address-related descriptor configuration parameters (DmaSourceAddress, DmaDestinationAddress) and descriptor linking via the current next-TCD links ([CP_SWS_Dma_00391], [CP_SWS_Dma_00392]).⌋

[CP_SWS_Dma_00314] Scatter-Gather Width and Loop Consistency Validation

Status: DRAFT

⌈

If Scatter-gather is configured for a channel, the Dma module shall validate, for each DmaTcd in the descriptor chain, that the configured transfer widths and loop counters satisfy the width ratio defined in [CP_SWS_Dma_CONSTR_00005] before transfer start. Relevant configuration parameters are DmaSourceTransferElementWidth, DmaDestinationTransferElementWidth, DmaInnerLoopCount, and DmaOuterLoopCount.⌋

[CP_SWS_Dma_00307] Scatter-Gather Configuration Error Detection

Status: DRAFT

⌈

If development error detection is enabled (DmaDevErrorDetect == True) and Scatter-gather related runtime configuration data supplied through Dma_UpdateTransferParameters() or Dma_ChannelTransferConfigUpdate() is invalid, the corresponding API shall raise DMA_E_PARAM_CONFIG. Relevant Scatter-gather parameters are DmaChannelScatterGatherEnable, DmaChannelTcdRef, NextTcdIdx, DmaInnerLoopCount, and DmaOuterLoopCount.⌋

[CP_SWS_Dma_00321] Scatter-Gather Configuration Rejection Status

Status: DRAFT

⌈

If invalid Scatter-gather related runtime configuration is supplied through Dma_UpdateTransferParameters() or Dma_ChannelTransferConfigUpdate(), the called API shall return E_NOT_OK. The rejected configuration includes the Scatter-gather related parameters DmaChannelScatterGatherEnable, DmaChannelTcdRef, and NextTcdIdx.⌋

[CP_SWS_Dma_00029] Scatter-Gather Runtime Error Signaling

Status: DRAFT

⌈

If a runtime transfer error is detected while processing a Scatter-gather descriptor chain, the Dma module shall signal the transfer error condition by invoking Dma_TransferErrorNotification() if notifications are enabled.⌋

[CP_SWS_Dma_00315] Scatter-Gather Runtime Error Status Reflection

Status: DRAFT

⌈

If a runtime transfer error is detected while processing a Scatter-gather descriptor chain, the Dma module shall reflect the error in the channel status reporting interfaces.⌋

[CP_SWS_Dma_00316] Scatter-Gather CRC Failure Error Classification

Status: DRAFT

⌈

If CRC support is configured and CRC verification fails for DmaTcd Dn, the Dma module shall treat this as a runtime transfer error.⌋

Note: The transfer error could be indicated either immediately upon detection, but latest upon completion of the full transfer.

[CP_SWS_Dma_00030] Scatter-Gather Runtime Error Countermeasure

Status: DRAFT

⌈

If a runtime transfer error occurs for a Scatter-gather channel, the Dma module shall stop further descriptor processing, update the channel status to DMA_CHANNEL_TRANSFER_ERROR, preserve error-relevant transfer information for software diagnosis, and transition the channel to DMA_CHANNEL_DISABLED state after the transfer error status has been made observable to software. This applies to software-triggered and hardware-triggered channels; a hardware-triggered channel is not re-armed for hardware triggering automatically.⌋

[CP_SWS_Dma_00031] Scatter-Gather Recovery Handling

Status: DRAFT

⌈

After a Scatter-gather runtime transfer error, the Dma module shall allow restart only after explicit software recovery action. Depending on the application and error cause, this recovery action may consist of a transfer-parameter update via Dma_UpdateTransferParameters() or a channel reconfiguration via Dma_ChannelTransferConfigUpdate(), followed by the restart request defined in [CP_SWS_Dma_00322].⌋

[CP_SWS_Dma_00322] Scatter-Gather Recovery Restart Trigger

Status: DRAFT

⌈

After a Scatter-gather runtime transfer error, restart shall require an explicit restart request: Dma_StartChannelTransfer() for a channel whose DmaChannelOperationMode is DMA_CHANNEL_MODE_SOFTWARE_POLLING or DMA_CHANNEL_MODE_SOFTWARE_INTERRUPT, or Dma_EnableChannel() for a channel whose DmaChannelOperationMode is DMA_CHANNEL_MODE_HARDWARE. In both cases the restart loads the current channel-to-TCD mapping, so the descriptor chain restarts from its first descriptor.⌋

7.2.10 Polling-Based Supervision and Advanced Features

[CP_SWS_Dma_00018] Data Width Adaptation Transfer

Status: DRAFT

⌈

When source and destination transfer widths differ for a configured transfer, the Dma module shall configure and execute the transfer according to the hardware-supported width adaptation semantics for packing or unpacking.⌋

7.3 Error Classification

Chapter [4, General Specification of Basic Software Modules] 7.2 *"Error Handling"* describes the error handling of the Basic Software in detail. Above all, it constitutes a classification scheme consisting of five error types which may occur in BSW modules.

Based on this foundation, the following section specifies particular errors arranged in the respective subsections below.

7.3.1 Development Errors

[CP_SWS_Dma_00201] Definition of development errors in module Dma

Status: DRAFT

⌈

Type of error	*Related error code*	*Error value*
API service called with invalid channel identifier	DMA_E_PARAM_CHANNEL	0x00
API service called while module is already initialized	DMA_E_ALREADY_INITIALIZED	0x01
API service called with NULL or invalid pointer parameter	DMA_E_PARAM_POINTER	0x02
API service used without module initialization	DMA_E_UNINIT	0x03
API service called with invalid configuration data	DMA_E_PARAM_CONFIG	0x04
API service called from wrong partition/core context for the referenced channel	DMA_E_WRONG_PARTITION	0x05
API service called for a channel that is in a state or channel operation mode incompatible with the requested operation (used by CP_SWS_Dma_00334, CP_SWS_Dma_00371- CP_SWS_Dma_00376).	DMA_E_INVALID_STATE	0x06

⌋

7.3.1.1 Development Error Detection

Development error checks specified in this document, including the rejection of APIs called for a channel with a non-matching `DmaChannelOperationMode` ([CP_SWS_Dma_00334], [CP_SWS_Dma_00371] to [CP_SWS_Dma_00376]), are performed only if `DmaDevErrorDetect` is enabled (see [SWS_BSW_00042] [4]).

[CP_SWS_Dma_00212] Channel Parameter Validation

Status: DRAFT

⌈

If development error detection is enabled (`DmaDevErrorDetect` == True), all channel-related API functions (`Dma_EnableChannel`(), `Dma_DisableChannel`(),

Dma_StartChannelTransfer(), Dma_StopChannelTransfer(), Dma_GetChannelStatus(), Dma_SuspendChannelTransfer(), Dma_ResumeChannelTransfer(), Dma_ChannelTransferConfigUpdate(), Dma_EnableChannelNotifications(), Dma_DisableChannelNotifications()) shall raise the development error DMA_E_PARAM_CHANNEL if the passed ChannelId does not match the DmaChannelId of any configured DmaChannel. For Dma_GetLatestTransferIndex() and Dma_GetCrcResult(), see [CP_SWS_Dma_00304] and [CP_SWS_Dma_00325].⌋

[CP_SWS_Dma_00388] TCD Index Parameter Validation

Status: DRAFT

⌈

If development error detection is enabled (DmaDevErrorDetect == True), the API functions Dma_GetTransferParameters(), Dma_UpdateTransferParameters() and Dma_ChannelTransferConfigUpdate() shall raise the development error DMA_E_PARAM_CONFIG if the passed TcdIdx does not match the DmaTcdId of any configured DmaTcd (see [CP_SWS_Dma_00389]). This includes the value DMA_TCD_IDX_NONE ([CP_SWS_Dma_00390]).⌋

[CP_SWS_Dma_00202] Uninitialized Module Detection

Status: DRAFT

⌈

If development error detection is enabled (DmaDevErrorDetect == True), all module API functions (except Dma_Init()) shall raise the development error DMA_E_UNINIT if the Dma module is not in DMA_INIT state at the time of function call.⌋

[CP_SWS_Dma_00209] DMA Channel Partition Mapping Validation

Status: DRAFT

⌈

If development error detection is enabled (DmaDevErrorDetect == True), Dma_Init() shall raise the development error DMA_E_PARAM_CONFIG if a configured Ecuc Partition Ref (DmaChannelEcucPartitionRef) is invalid, unresolved, or inconsistent with the configured channel to DMA hardware unit assignment (DmaChannelHwUnitRef).⌋

[CP_SWS_Dma_00203] Scatter-Gather Descriptor Configuration Validation

Status: DRAFT

⌈

If development error detection is enabled (DmaDevErrorDetect == True), Dma_StartChannelTransfer() shall raise DMA_E_PARAM_CONFIG when Scatter-gather descriptor validation fails before transfer start (for example, invalid next-TCD

index, inconsistent inner-loop and outer-loop counters, or unsupported descriptor field combination).⌋

7.3.2 Runtime Errors

Runtime errors represent error conditions detected during normal module operation. They are reported via `Det_ReportRuntimeError`() and may occur under valid usage and configuration. The following runtime error codes are defined for the Dma module:

[CP_SWS_Dma_00204] Definition of runtime errors in module Dma

Status: DRAFT

⌈

Type of error	*Related error code*	*Error value*
Recoverable transfer-level fault detected for a channel	DMA_E_TRANSFER_ERROR	0x10
Non-recoverable hardware fault detected for a channel	DMA_E_HARDWARE_ERROR	0x11
Scatter-gather descriptor fetch or chain-link failure detected at runtime	DMA_E_DESCRIPTOR_ERROR	0x12
Transfer start requested while channel is already processing a transfer	DMA_E_CHANNEL_BUSY	0x13

⌋

7.3.2.1 Runtime Error Reporting

[CP_SWS_Dma_00205] Transfer Error Reporting

Status: DRAFT

⌈

When a recoverable transfer-level fault is detected for a channel and the transfer-error notification is issued, the Dma module shall call `Det_ReportRuntimeError`() with error code `DMA_E_TRANSFER_ERROR`.⌋

[CP_SWS_Dma_00206] Hardware Error Reporting

Status: DRAFT

⌈

When a `DMA channel` enters `DMA_CHANNEL_HARDWARE_ERROR` state due to a non-recoverable hardware fault, the Dma module shall call `Det_ReportRuntimeError`() with error code `DMA_E_HARDWARE_ERROR`.⌋

[CP_SWS_Dma_00207] Transfer Restart After Transfer Error

Status: DRAFT

⌈

After a recoverable transfer-level fault has been detected for a channel whose `Dma-ChannelOperationMode` is `DMA_CHANNEL_MODE_SOFTWARE_POLLING` or `DMA_CHANNEL_MODE_SOFTWARE_INTERRUPT`, the Dma module shall permit a new transfer to be started on that channel via `Dma_StartChannelTransfer`() without requiring prior channel reconfiguration. For hardware-triggered channels, the restart after a `Scatter-gather` runtime transfer error is specified by [CP_SWS_Dma_00322]. Recovery from `DMA_CHANNEL_HARDWARE_ERROR` requires module re-initialization and is governed by the hardware error state semantics.⌋

[CP_SWS_Dma_00210] Descriptor Error Reporting

Status: DRAFT

⌈

When a `Scatter-gather` descriptor fetch or chain-link failure is detected at runtime, the Dma module shall call `Det_ReportRuntimeError`() with error code `DMA_E_DESCRIPTOR_ERROR`.⌋

[CP_SWS_Dma_00211] Channel Busy Reporting

Status: DRAFT

⌈

If `Dma_StartChannelTransfer`() is called for a software-triggered channel in `DMA_CHANNEL_TRANSFER_IN_PROGRESS` or `DMA_CHANNEL_SUSPENDED` state, the Dma module shall call `Det_ReportRuntimeError`() with error code `DMA_E_CHANNEL_BUSY` and shall reject the request as specified in [CP_SWS_Dma_00302].⌋

7.3.3 Production Errors

There are no production errors.

7.4 Security Events

The module does not report security events.

8 API specification

8.1 Imported types

In this chapter all types included from the following files are listed.

[CP_SWS_Dma_00127] Definition of imported datatypes of module Dma

Status: DRAFT

⌈

Module	*Header File*	*Imported Type*
Std	Std_Types.h	Std_ReturnType

⌋

s

8.2 Type definitions

8.2.1 Dma_ChannelIdType

[CP_SWS_Dma_00130] Definition of datatype Dma_ChannelIdType

Status: DRAFT

⌈

Name	Dma_ChannelIdType (draft)
Kind	Type
Derived from	uint16
Description	– **Tags:** atp.Status=draft
Available via	Dma.h

⌋

8.2.2 Dma_TcdIdxType

[CP_SWS_Dma_00135] Definition of datatype Dma_TcdIdxType

Status: DRAFT

⌈

Name	Dma_TcdIdxType (draft)
Kind	Type
Derived from	uint16

▽

△

Description	Type for the index of a transfer control descriptor (TCD). The value of a TcdIdx is equal to the Dma TcdId of the addressed DmaTcd (0 .. 65534). The value 0xFFFF (DMA_TCD_IDX_NONE) does not identify any DmaTcd and indicates that no TCD is referenced. **Tags:** atp.Status=draft
Available via	Dma.h

⌋

[CP_SWS_Dma_00389] TCD Index Mapping

Status: DRAFT

⌈A TcdIdx parameter of type `Dma_TcdIdxType` shall address the `DmaTcd` whose `DmaTcdId` is equal to the value of TcdIdx. Invalid values are handled according to [CP_SWS_Dma_00388].⌋

[CP_SWS_Dma_00390] Definition of DMA_TCD_IDX_NONE

Status: DRAFT

⌈The Dma module shall provide in Dma.h the symbolic constant `DMA_TCD_IDX_NONE` of type `Dma_TcdIdxType` with the value 0xFFFF. `DMA_TCD_IDX_NONE` does not identify any `DmaTcd`; it indicates that no `TCD` is referenced.⌋

Note

Since `DmaTcdId` is limited to the range 0 to 65534, the value 0 is a valid `TCD` identifier. `DMA_TCD_IDX_NONE` is used for `NextTcdIdx` to indicate the end of a `Scatter-gather` chain and for `CurrentTcdIdx` if no `TCD` is associated with the channel.

8.2.3 Dma_TcdCfgType

[CP_SWS_Dma_00136] Definition of datatype Dma_TcdCfgType

Status: DRAFT

⌈

Name	Dma_TcdCfgType (draft)	
Kind	Structure	
Elements	SourceConfig	
	Type	Dma_SourceConfigType
	Comment	Runtime source-transfer subset with members: SourceAddress (SourceAddress) and TransferElementWidth (TransferElementWidth).
	DestinationConfig	
	Type	Dma_DestinationConfigType

▽

△

	Comment	Runtime destination-transfer subset with members: Destination Address (DestinationAddress) and TransferElementWidth (Transfer ElementWidth).
	InOutLoop	
	Type	Dma_InOutLoopConfigType
	Comment	Runtime loop subset with members: InnerLoop.Count (InnerLoop Count) and OuterLoop.Count (OuterLoopCount).
	NextTcdIdx	
	Type	Dma_TcdIdxType
	Comment	Index (DmaTcdId) of the next TCD in a scatter-gather chain. DMA_ TCD_IDX_NONE indicates the end of the chain. The link is initialized from DmaTcdNextTcdRef during Dma_Init() and can be changed with Dma_UpdateTransferParameters().
Description	Dma_TcdCfgType is the data type for the configuration of one transfer control descriptor (TCD). It contains the runtime values required to describe a DMA transfer block. Hardware-extension members are intentionally not standardized in this definition; implementation-specific extensions may be added by a vendor in an extension area, analogous to ECUC vendor-specific extension concepts. **Tags:** atp.Status=draft	
Available via	Dma.h	

⌋

8.2.4 Dma_SourceConfigType

[CP_SWS_Dma_00137] Definition of datatype Dma_SourceConfigType

Status: DRAFT

⌈

Name	Dma_SourceConfigType (draft)	
Kind	Structure	
Elements	SourceAddress	
	Type	uint64
	Comment	Source address value corresponding to SourceAddress.
	TransferElementWidth	
	Type	uint32
	Comment	Source transfer-element width corresponding to TransferElement Width.
Description	Type associated to the runtime source-transfer parameter subset. **Tags:** atp.Status=draft	
Available via	Dma.h	

⌋

8.2.5 Dma_DestinationConfigType

[CP_SWS_Dma_00138] Definition of datatype Dma_DestinationConfigType

Status: DRAFT

⌈

Name	Dma_DestinationConfigType (draft)	
Kind	Structure	
Elements	DestinationAddress	
	Type	uint64
	Comment	Destination address value corresponding to DestinationAddress.
	TransferElementWidth	
	Type	uint32
	Comment	Destination transfer-element width corresponding to TransferElement Width.
Description	Type associated to the runtime destination-transfer parameter subset. **Tags:** atp.Status=draft	
Available via	Dma.h	

⌋

8.2.6 Dma_InOutLoopConfigType

[CP_SWS_Dma_00139] Definition of datatype Dma_InOutLoopConfigType

Status: DRAFT

⌈

Name	Dma_InOutLoopConfigType (draft)	
Kind	Structure	
Elements	InnerLoopCount	
	Type	uint32
	Comment	Inner-loop count corresponding to InnerLoopCount.
	OuterLoopCount	
	Type	uint32
	Comment	Outer-loop count corresponding to OuterLoopCount.
Description	Type associated to runtime inner/outer-loop transfer parameters. **Tags:** atp.Status=draft	
Available via	Dma.h	

⌋

8.2.7 Dma_ChannelStatusType

[CP_SWS_Dma_00132] Definition of datatype Dma_ChannelStatusType

Status: DRAFT

⌈

Name	Dma_ChannelStatusType (draft)		
Kind	Enumeration		
Range	DMA_CHANNEL_ DISABLED	0x00	Channel is idle. Software-triggered channels accept Dma_StartChannelTransfer; hardware-triggered channels do not accept hardware triggers.
	DMA_CHANNEL_ WAITING_TRIGGER	0x01	Channel is enabled and waiting for a hardware trigger (hardware-triggered channels only).
	DMA_CHANNEL_ TRANSFER_IN_ PROGRESS	0x02	Channel is currently processing a transfer and no final transfer result is available yet. This state is non-terminal and shall not represent successful completion.
	DMA_CHANNEL_ SUSPENDED	0x03	Channel transfer is suspended and may be resumed if supported.
	DMA_CHANNEL_ TRANSFER_COMPLETED	0x04	Returned status value indicating that the most recently evaluated transfer has finished successfully. This value does not represent a persistent channel state.
	DMA_CHANNEL_ TRANSFER_ERROR	0x05	Returned status value indicating that the most recently evaluated transfer ended with a recoverable transfer error. This value does not represent a persistent channel state.
	DMA_CHANNEL_ HARDWARE_ERROR	0x06	Returned status value indicating that the channel is in a non-recoverable hardware error state.
Description	Type associated to the status of each DMA channel. **Tags:** atp.Status=draft		
Available via	Dma.h		

⌋

8.2.8 Dma_CrcResultType

[CP_SWS_Dma_00133] Definition of datatype Dma_CrcResultType

Status: DRAFT

⌈

Name	Dma_CrcResultType (draft)		
Kind	Enumeration		
Range	DMA_CRC_FAILED	0x00	CRC verification failed.
	DMA_CRC_PASSED	0x01	CRC verification passed successfully.

▽

△

Description	CRC result status indicating verification outcome of DMA transfer. **Tags:** atp.Status=draft
Available via	Dma.h

⌋

8.2.9 Dma_ChannelInfoType

[CP_SWS_Dma_00134] Definition of datatype Dma_ChannelInfoType

Status: DRAFT

⌈

Name	Dma_ChannelInfoType (draft)	
Kind	Structure	
Elements	CurrentTcdIdx	
	Type	Dma_TcdIdxType
	Comment	Index of the TCD currently associated with the channel for the latest transfer progress snapshot. DMA_TCD_IDX_NONE, if no TCD is associated with the channel.
	CurrentInnerLoopCount	
	Type	uint32
	Comment	Current inner-loop count of the latest transfer progress snapshot (see InnerLoopCount of Dma_InOutLoopConfigType).
	CurrentOuterLoopCount	
	Type	uint32
	Comment	Current outer-loop count of the latest transfer progress snapshot (see OuterLoopCount of Dma_InOutLoopConfigType)
	TotalTransferredSize	
	Type	uint64
	Comment	Total size of data transferred since transfer start, in bytes.
	CurrentSrcAddress	
	Type	uint64
	Comment	Source address currently being transferred.
	CurrentDestAddress	
	Type	uint64
	Comment	Destination address currently being transferred.
Description	Type associated to the information regarding the transfer progress on a given DMA channel. **Tags:** atp.Status=draft	
Available via	Dma.h	

⌋

8.2.10 Dma_ConfigType

[CP_SWS_Dma_00143] Definition of datatype Dma_ConfigType

Status: DRAFT

⌈

Name	Dma_ConfigType (draft)	
Kind	Structure	
Elements	Implementation-specific	
	Type	–
	Comment	The contents of the initialization data structure are DMA driver specific.
Description	This type of external data structure shall contain the initialization data for the DMA Driver. **Tags:** atp.Status=draft	
Available via	Dma.h	

⌋

8.2.11 Std_ReturnType (Dma-specific extension)

In addition to the standard values `E_OK` and `E_NOT_OK`, the Dma module defines the following module-specific extension value of `Std_ReturnType`, as permitted by the Specification of Standard Types for module-specific `Std_ReturnType` extensions:

[CP_SWS_Dma_00126] Definition of Std_ReturnType-extension for module Dma

Status: DRAFT
Upstream requirements: SRS_BSW_00331, SRS_BSW_00369, SRS_BSW_00377, SRS_BSW_00441

⌈

Range	DMA_E_ALIGNMENT_UNSUPPORTED	0x02	The requested source and/or destination address alignment, or the associated transfer-element-width combination, is not supported by the selected DMA implementation strategy.
Description	Overlayed return value of Std_ReturnType for Dma API Dma_UpdateTransferParameters(). **Tags:** atp.Status=draft		
Available via	Dma.h		

⌋

8.3 Function definitions

[CP_SWS_Dma_00360] API Call Context Rule

Status: DRAFT

⌈For DMA services defined in this chapter, APIs shall be callable from task and ISR context unless explicitly specified otherwise in the respective API definition.⌋

[CP_SWS_Dma_00361] Synchronous API Execution Rule

Status: DRAFT

⌈For DMA services defined in this chapter with *Sync/Async* marked as Synchronous, the API call shall return without waiting for DMA transfer completion. Transfer completion and transfer error observation remain asynchronous and are handled via notification callbacks or polling, depending on the configured observation path.⌋

[CP_SWS_Dma_00362] Same-Channel API Serialization Rule

Status: DRAFT

⌈For all channel-related services marked as non-reentrant for the same channel in their respective function definitions, any invocation for a given channel while a prior invocation of any non-reentrant channel service for the same channel has not yet returned, the Dma module shall reject the second invocation, shall return E_NOT_OK and shall not modify the driver-managed channel state. This rule governs the synchronous API call execution and is independent of the asynchronous DMA transfer executing on the channel.⌋

Informative: Channel API Serialization and Async-Transfer Callability

The following table summarizes for each channel-related service (1) whether a concurrent same-channel invocation is serialized while a prior non-reentrant call for that channel has not returned, and (2) whether the service may be called while a DMA transfer is executing asynchronously on the channel. Services marked as reentrant (`Dma_GetChannelStatus`(), `Dma_GetCrcResult`()) are not subject to the serialization rule above.

API	While a prior non-reentrant call for same channel has not returned	While DMA transfer is executing asynchronously
`Dma_EnableChannel`() (hardware-triggered channels only)	E_NOT_OK	No — requires `DMA_CHANNEL_DISABLED` ([CP_SWS_Dma_00379])

▽

△

Dma_DisableChannel() (hardware-triggered channels only)	E_NOT_OK	Yes — stops any ongoing transfer
Dma_StartChannel-Transfer() (software-triggered channels only)	E_NOT_OK	No — requires DMA_CHANNEL_DISABLED
Dma_StopChannel-Transfer() (software-triggered channels only)	E_NOT_OK	Yes — intended to abort an ongoing transfer
Dma_SuspendChannel-Transfer()	E_NOT_OK	Yes — operates on in-progress transfer
Dma_ResumeChannel-Transfer()	E_NOT_OK	No — requires DMA_CHANNEL_SUSPENDED
Dma_GetChannel-Status()	Permitted — reentrant	Yes — intended for polling
Dma_GetLatest-TransferIndex()	E_NOT_OK	Yes — read-only snapshot
Dma_EnableChannel-Notifications()	E_NOT_OK	No — requires DMA_CHANNEL_DISABLED ([CP_SWS_Dma_00381])
Dma_DisableChannel-Notifications()	E_NOT_OK	No — requires DMA_CHANNEL_DISABLED ([CP_SWS_Dma_00382])
Dma_GetCrcResult()	Permitted — reentrant	No — result available after transfer completion only
Dma_GetTransfer-Parameters()	E_NOT_OK	No — requires descriptor not active in hardware
Dma_UpdateTransfer-Parameters()	E_NOT_OK	No — requires every channel using the descriptor in DMA_CHANNEL_DISABLED ([CP_SWS_Dma_00326])
Dma_ChannelTransfer-ConfigUpdate()	E_NOT_OK	No — requires DMA_CHANNEL_DISABLED

Table 8.1: Channel API serialization and callability during asynchronous transfer

8.3.1 Dma_Init

[CP_SWS_Dma_00100] Definition of API function Dma_Init

Status: DRAFT

⌈

Service Name	Dma_Init (draft)	
Syntax	`void Dma_Init ( const Dma_ConfigType* ConfigPtr )`	
Service ID [hex]	0x00	
Sync/Async	Synchronous	
Reentrancy	Non Reentrant	
Parameters (in)	ConfigPtr	Pointer to DMA Driver configuration set.
Parameters (inout)	None	
Parameters (out)	None	
Return value	None	
Description	Initializes the DMA Driver base state and the DMA peripheral base configuration. **Tags:** atp.Status=draft	
Available via	Dma.h	

⌋

[CP_SWS_Dma_00001] Module Initialization Behavior

Status: DRAFT

⌈

If `Dma_Init`() is called, the Dma module shall configure the DMA controller and initialize internal driver state according to the pre-compile configuration. Any initialization that is common across partitions/cores (for example shared DMA hardware bring-up) is implementation-defined.⌋

Guidance for integration: `Dma_Init`() should be called per EcuC partition/core context. Each call should initialize the configured channels that belong to the calling partition/core context.

[CP_SWS_Dma_00032] DMA Channel Initialization Scope

Status: DRAFT

⌈

In `Dma_Init`(), the Dma module shall initialize all configured DMA hardware units and `DMA channels` of the configuration set (see [CP_SWS_Dma_00001] for the general initialization behavior).⌋

[CP_SWS_Dma_00318] DMA Logical Channel to Hardware Unit Mapping

Status: DRAFT

⌈In Dma_Init(), the Dma module shall map each configured logical channel to the DMA hardware unit referenced by DmaChannelHwUnitRef. This mapping shall support multiple DMA hardware units on the target microcontroller.⌋

Note

How a DmaHwUnit is addressed (for example by a symbolic instance, an instance index or a register base address) and which physical channel of the hardware unit is used for a DmaChannel is implementation-specific; the corresponding parameters are provided in DmaHwUnitHwSpecificConfig and DmaChannelHwSpecificConfig.

[CP_SWS_Dma_00380] Default Notification State after Initialization

Status: DRAFT

⌈In Dma_Init(), the Dma module shall enable the notifications of each initialized channel whose DmaChannelOperationMode is DMA_CHANNEL_MODE_SOFTWARE_INTERRUPT or DMA_CHANNEL_MODE_HARDWARE, so that Dma_TransferCompleteNotification(), Dma_TransferErrorNotification(), and Dma_HardwareErrorNotification() are invoked without a prior call of Dma_EnableChannelNotifications().⌋

[CP_SWS_Dma_00300] Module Initialization Error Detection

Status: DRAFT

⌈

If development error detection is enabled (DmaDevErrorDetect == True) and Dma_Init() is called while the Dma module is already in DMA_INIT state, the function shall raise the development error DMA_E_ALREADY_INITIALIZED.⌋

8.3.2 Dma_GetLatestTransferIndex

[CP_SWS_Dma_00109] Definition of API function Dma_GetLatestTransferIndex

Status: DRAFT

⌈

Service Name	Dma_GetLatestTransferIndex (draft)

Syntax	`Std_ReturnType Dma_GetLatestTransferIndex (`
	`  Dma_ChannelIdType ChannelId,`
	`  Dma_ChannelInfoType* ChannelInfoPtr`
	`)`

▽

△

Service ID [hex]	0x09	
Sync/Async	Synchronous	
Reentrancy	Reentrant for different channels; non-reentrant for the same channel	
Parameters (in)	ChannelId	Channel identifier for which latest transfer progress is requested.
Parameters (inout)	None	
Parameters (out)	ChannelInfoPtr	Pointer to channel information snapshot structure to be populated.
Return value	Std_ReturnType	`E_OK`: snapshot available; `E_NOT_OK`: snapshot not available.
Description	Returns a snapshot of the latest transfer progress of the given channel. The output parameter provides the current TCD index, the current inner-loop and outer-loop counts, the total transferred size in bytes since transfer start, and the current source and destination addresses. For software-triggered channels, calling Dma_StartChannelTransfer clears the channel info. For hardware-triggered channels, calling Dma_EnableChannel clears the channel info. **Tags:** atp.Status=draft	
Available via	Dma.h	

⌋

[CP_SWS_Dma_00010] Get Latest Transfer Index Behavior

Status: DRAFT

⌈When `Dma_GetLatestTransferIndex`() is called, the Dma module shall populate all members of the `Dma_ChannelInfoType` output structure with a consistent snapshot of the latest transfer progress for the channel, including `CurrentTcdIdx`, `CurrentInnerLoopCount`, `CurrentOuterLoopCount`, `TotalTransferredSize` (in bytes), `CurrentSrcAddress`, and `CurrentDestAddress`.⌋

Note: The latest transfer progress could be due to hardware limitation also an earlier progress of the transfer.

[CP_SWS_Dma_00304] Get Latest Transfer Index Error Detection

Status: DRAFT

⌈If development error detection is enabled (`DmaDevErrorDetect` == True) and `Dma_GetLatestTransferIndex`() is called with an invalid `ChannelId`, the function shall raise the development error `DMA_E_PARAM_CHANNEL`.⌋

8.3.3 Dma_EnableChannelNotifications

[CP_SWS_Dma_00110] Definition of API function Dma_EnableChannelNotifications

Status: DRAFT

⌈

Service Name	Dma_EnableChannelNotifications (draft)	
Syntax	`Std_ReturnType Dma_EnableChannelNotifications ( Dma_ChannelIdType ChannelId )`	
Service ID [hex]	0x0A	
Sync/Async	Synchronous	
Reentrancy	Reentrant for different channels; non-reentrant for the same channel	
Parameters (in)	ChannelId	Channel identifier for which notifications shall be enabled.
Parameters (inout)	None	
Parameters (out)	None	
Return value	Std_ReturnType	`E_OK`: notifications enabled; `E_NOT_OK`: request rejected, including when the channel is not in `DMA_CHANNEL_DISABLED` (SWS_Dma_00381/00382).
Description	Enables the transfer-complete, transfer-error, and hardware-error notifications of the channel. Only allowed in DMA_CHANNEL_DISABLED. Notifications are enabled by default after Dma_Init (SWS_Dma_00380). **Tags:** atp.Status=draft	
Available via	Dma.h	

⌋

[CP_SWS_Dma_00007] Enable Channel Notifications Behavior

Status: DRAFT

⌈When `Dma_EnableChannelNotifications`() is called for a channel in `DMA_CHANNEL_DISABLED` state, the Dma module shall enable transfer-completion, transfer-error, and hardware-error notifications for the channel according to the configured callback settings. Once enabled, the Dma module shall invoke `Dma_TransferCompleteNotification`() on successful transfer completion, `Dma_TransferErrorNotification`() on transfer error detection, and `Dma_HardwareErrorNotification`() on hardware error detection for that channel. Relevant callback configuration parameters are `DmaChannelCompletedNotificationCallback`, `DmaChannelTransferErrorNotificationCallback`, and `DmaChannelHardwareErrorNotificationCallback`.⌋

[CP_SWS_Dma_00323] Enable Channel Notifications Callback Configuration Error Detection

Status: DRAFT

⌈If development error detection is enabled (DmaDevErrorDetect == True) and Dma_EnableChannelNotifications() is called for a channel where none of DmaChannelCompletedNotificationCallback , DmaChannelTransferErrorNotificationCallback, or DmaChannelHardwareErrorNotificationCallback is configured, the function shall raise the development error DMA_E_PARAM_CONFIG.⌋

[CP_SWS_Dma_00381] Enable Channel Notifications Wrong-State Rejection Behavior

Status: DRAFT

⌈If Dma_EnableChannelNotifications() is called for a channel that is not in DMA_CHANNEL_DISABLED state, the Dma module shall reject the request, shall keep the notification setting and the channel state unchanged, and shall return E_NOT_OK.⌋

8.3.4 Dma_DisableChannelNotifications

[CP_SWS_Dma_00111] Definition of API function Dma_DisableChannelNotifications

Status: DRAFT

⌈

Service Name	Dma_DisableChannelNotifications (draft)	
Syntax	`Std_ReturnType Dma_DisableChannelNotifications ( Dma_ChannelIdType ChannelId )`	
Service ID [hex]	0x0B	
Sync/Async	Synchronous	
Reentrancy	Reentrant for different channels; non-reentrant for the same channel	
Parameters (in)	ChannelId	Channel identifier for which notifications shall be disabled.
Parameters (inout)	None	
Parameters (out)	None	
Return value	Std_ReturnType	E_OK: notifications disabled; E_NOT_OK: request rejected, including when the channel is not in DMA_CHANNEL_DISABLED (SWS_Dma_00381/00382).
Description	Disables the transfer-complete, transfer-error, and hardware-error notifications of the channel. Only allowed in DMA_CHANNEL_DISABLED. Notifications are enabled by default after Dma_Init (SWS_Dma_00380). **Tags:** atp.Status=draft	
Available via	Dma.h	

⌋

[CP_SWS_Dma_00008] Disable Channel Notifications Behavior

Status: DRAFT

⌈When `Dma_DisableChannelNotifications`() is called for a channel in `DMA_CHANNEL_DISABLED` state, the Dma module shall disable transfer-completion, transfer-error, and hardware-error notifications for the channel and shall not invoke `Dma_TransferCompleteNotification`(), `Dma_TransferErrorNotification`() or `Dma_HardwareErrorNotification`() for this channel after this call. Relevant callback configuration parameters are `DmaChannelCompletedNotificationCallback`, `DmaChannelTransferErrorNotificationCallback`, and `DmaChannelHardwareErrorNotificationCallback`.⌋

[CP_SWS_Dma_00382] Disable Channel Notifications Wrong-State Rejection Behavior

Status: DRAFT

⌈If `Dma_DisableChannelNotifications`() is called for a channel that is not in `DMA_CHANNEL_DISABLED` state, the Dma module shall reject the request, shall keep the notification setting and the channel state unchanged, and shall return E_NOT_OK.⌋

Informative: Notification Setting and Channel State

Because the notification setting can only be changed in `DMA_CHANNEL_DISABLED` state ([CP_SWS_Dma_00381], [CP_SWS_Dma_00382]), and a channel with disabled notifications cannot be started or enabled ([CP_SWS_Dma_00383], [CP_SWS_Dma_00384]), a channel whose `DmaChannelOperationMode` is `DMA_CHANNEL_MODE_SOFTWARE_INTERRUPT` or `DMA_CHANNEL_MODE_HARDWARE` never executes a transfer without an active observation path, and no completion or error event is lost. The notification APIs therefore do not switch between notification-based and polling-based observation; the observation path is selected per channel by `DmaChannelOperationMode`.

8.3.5 Dma_GetCrcResult

[CP_SWS_Dma_00112] Definition of API function Dma_GetCrcResult

Status: DRAFT

⌈

Service Name	Dma_GetCrcResult (draft)

Syntax	`Std_ReturnType Dma_GetCrcResult (`
	`  Dma_ChannelIdType ChannelId,`
	`  Dma_CrcResultType* CrcResultPtr`
	`)`

Service ID [hex]	0x0C

▽

△

Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	ChannelId	Channel identifier for which CRC result shall be queried.
Parameters (inout)	None	
Parameters (out)	CrcResultPtr	Pointer to destination for the channel CRC verification result.
Return value	Std_ReturnType	E_OK: CRC result written to CrcResultPtr; E_NOT_OK: request rejected or CRC result unavailable.
Description	Gets CRC result after a transfer was completed. For scatter-gather operation with CRC support, if CRC verification fails in descriptor Dn. For scatter-gather operation with CRC support, a CRC verification failure in descriptor Dn stops further descriptor processing (see CP_SWS_Dma_00030). **Tags:** atp.Status=draft	
Available via	Dma.h	

⌋

[CP_SWS_Dma_00011] Get CRC Result Behavior

Status: DRAFT

⌈When Dma_GetCrcResult() is called for a channel with CRC support and a CRC result is available, the Dma module shall write the channel-related CRC verification result to CrcResultPtr according to the configured CRC mechanism and shall return E_OK.⌋

[CP_SWS_Dma_00324] Get CRC Result Failure Return Behavior

Status: DRAFT

⌈When Dma_GetCrcResult() is called and CRC support is not configured for the channel, i.e. DmaCrcHwSpecificConfig is not configured for the corresponding DmaChannel, or no CRC result is available, the Dma module shall return E_NOT_OK.⌋

Note

The presence of DmaCrcHwSpecificConfig only indicates that CRC support is configured for a DmaChannel. How the CRC calculation is actually configured and activated (for example selection of the CRC hardware, polynomial, seed value and result register) is vendor-specific and defined by the parameters that the implementation adds to DmaCrcHwSpecificConfig.

[CP_SWS_Dma_00325] Get CRC Result Error Detection

Status: DRAFT

⌈If development error detection is enabled (DmaDevErrorDetect == True) and Dma_GetCrcResult() is called with an invalid ChannelId, the function shall raise the development error DMA_E_PARAM_CHANNEL.⌋

8.3.6 Dma_GetTransferParameters

[CP_SWS_Dma_00140] Definition of API function Dma_GetTransferParameters

Status: DRAFT

⌈

Service Name	Dma_GetTransferParameters (draft)	

Syntax	`Std_ReturnType Dma_GetTransferParameters (`
	`  Dma_TcdIdxType TcdIdx,`
	`  Dma_TcdCfgType* TcdCfgPtr`
	`)`	

Service ID [hex]	0x11	
Sync/Async	Synchronous	
Reentrancy	Non Reentrant	
Parameters (in)	TcdIdx	Transfer control descriptor index to be read.
Parameters (inout)	None	
Parameters (out)	TcdCfgPtr	Pointer to the buffer receiving the runtime-updatable TCD parameters.
Return value	Std_ReturnType	`E_OK`: parameters returned; `E_NOT_OK`: read rejected.
Description	Returns runtime-updatable transfer parameters of a selected DmaTCD as Dma_TcdCfgType. The returned data includes SourceConfig, DestinationConfig, InOutLoop, and NextTcdIdx and can be used as basis for subsequent update through Dma_UpdateTransferParameters(). **Tags:** atp.Status=draft	
Available via	Dma.h	

⌋

Read-Modify-Write Workflow (informative)

The intended workflow is: (1) read the current runtime-updatable descriptor subset via `Dma_GetTransferParameters`(), (2) modify the returned Dma_TcdCfgType in application context, and (3) write the updated subset back using `Dma_UpdateTransferParameters`(). This workflow supports deterministic parameter adjustment while preserving unchanged fields. The interaction sequence is illustrated in Figure 9.8.

[CP_SWS_Dma_00365] Get Transfer Parameters Behavior

Status: DRAFT

⌈When `Dma_GetTransferParameters`() is called while the addressed `DmaTcd` is neither active in hardware nor already fetched for `Scatter-gather` execution, the Dma module shall copy the runtime-updatable transfer parameter subset of the addressed `DmaTcd` to TcdCfgPtr and return E_OK.⌋

[CP_SWS_Dma_00369] Get Transfer Parameters Optional Source or Destination Configuration Handling

Status: DRAFT

⌈If `Dma_GetTransferParameters`() is called, the Dma module shall set members of `Dma_TcdCfgType` as follows when the corresponding configuration parameter is

not configured in the addressed DmaTcd: In the following list, the returned datatype element is listed first, followed by the corresponding configuration parameter it depends on.

- Returned datatype element SourceAddress shall be NULL, if configuration parameter DmaSourceAddress is not configured.
- Returned datatype element TransferElementWidth shall be 0, if configuration parameter DmaSourceTransferElementWidth is not configured.
- Returned datatype element DestinationAddress shall be NULL, if configuration parameter DmaDestinationAddress is not configured.
- Returned datatype element TransferElementWidth shall be 0, if configuration parameter DmaDestinationTransferElementWidth is not configured.
- Returned datatype element InnerLoopCount shall be 0, if configuration parameter DmaInnerLoopCount is not configured.
- Returned datatype element OuterLoopCount shall be 0, if configuration parameter DmaOuterLoopCount is not configured.
- Returned datatype element NextTcdIdx shall be DMA_TCD_IDX_NONE, if no next TCD is linked to the addressed DmaTcd (see [CP_SWS_Dma_00391] and [CP_SWS_Dma_00392]).

The function shall return E_OK.⌋

[CP_SWS_Dma_00366] Get Transfer Parameters Wrong-State Rejection Behavior

Status: DRAFT

⌈If Dma_GetTransferParameters() is called while the addressed descriptor is active in hardware or already fetched for Scatter-gather execution, the Dma module shall reject the request, shall not write output data to TcdCfgPtr, and shall return E_NOT_OK.⌋

[CP_SWS_Dma_00367] Get Transfer Parameters Error Detection

Status: DRAFT

⌈If development error detection is enabled (DmaDevErrorDetect == True) and Dma_GetTransferParameters() is called with null TcdCfgPtr, the function shall raise development error DMA_E_PARAM_POINTER.⌋

8.3.7 Dma_UpdateTransferParameters

[CP_SWS_Dma_00113] Definition of API function Dma_UpdateTransferParameters

Status: DRAFT

⌈

Service Name	Dma_UpdateTransferParameters (draft)	

Syntax	`Std_ReturnType Dma_UpdateTransferParameters (`
	`  Dma_TcdIdxType TcdIdx,`
	`  const Dma_TcdCfgType* TcdCfgPtr`
	`)`	

Service ID [hex]	0x0D	
Sync/Async	Synchronous	
Reentrancy	Non Reentrant	
Parameters (in)	TcdIdx	Transfer control descriptor index to be updated.
	TcdCfgPtr	Pointer to runtime TCD parameter subset to be written.
Parameters (inout)	None	
Parameters (out)	None	
Return value	Std_ReturnType	E_OK: parameters updated; E_NOT_OK: update rejected, e.g. a channel using the addressed TCD is not in DMA_CHANNEL_DISABLED state (SWS_Dma_00326), or scatter-gather is disabled for a channel using the addressed TCD while the descriptor contains a NextTcdIdx value other than DMA_TCD_IDX_NONE. The descriptor data is not modified. DMA_E_ALIGNMENT_UNSUPPORTED: the requested source and/or destination address alignment, or the associated transfer-element-width combination, is not supported by the selected DMA implementation strategy. The descriptor data is not modified.
Description	Updates the runtime-updatable parameters of the transfer control descriptor (TCD) addressed by TcdIdx. This API is intended for read-modify-write handling of descriptor data while every channel using the addressed TCD (via DmaChannelTcdRef or the next-TCD links) is in DMA_CHANNEL_DISABLED, which for software-triggered channels is the idle state between two transfers. It allows the application to adjust transfer settings such as source and destination addresses, transfer-element widths, and loop counts without reinitializing the channel. **Tags:** atp.Status=draft	
Available via	Dma.h	

⌋

[CP_SWS_Dma_00012] Update Transfer Parameters Behavior

Status: DRAFT

⌈When Dma_UpdateTransferParameters() is called while every DmaChannel that uses the addressed DmaTcd is in DMA_CHANNEL_DISABLED state, the Dma module shall update transfer parameters according to the provided configuration data and hardware capabilities. Relevant configurable descriptor parameters include DmaSourceAddress, DmaSourceTransferElementWidth, DmaDestinationAddress, DmaDestinationTransferElementWidth, DmaInnerLoopCount, DmaOuterLoopCount, and the next-TCD link (NextTcdIdx, see [CP_SWS_Dma_00392]). A DmaChannel uses a DmaTcd if the DmaTcd is referenced by its current channel-to-TCD mapping (DmaChannelTcdRef, possibly changed via

Dma_ChannelTransferConfigUpdate()) or is reached from there via the current next-TCD links ([CP_SWS_Dma_00391], [CP_SWS_Dma_00392]).⌋

[CP_SWS_Dma_00368] Update Transfer Parameters Runtime Alignment Handling

Status: DRAFT

⌈Misaligned source and destination addresses may occur at runtime. The caller shall provide transfer addresses, and the Dma module shall handle supported alignment mismatches according to hardware capabilities. If direct hardware support is limited, the Dma module may optionally handle alignment mismatches by software-managed transfer segmentation (for example leading segment, aligned bulk segment, and trailing segment). If a provided alignment combination is not supported by the selected implementation strategy, the Dma module shall reject the update, and shall return DMA_E_ALIGNMENT_UNSUPPORTED, the Dma-specific extension value of Std_ReturnType.⌋

Note

If an alignment combination is not supported for the requested transfer-element width, the transfer-element width should be adapted to a lower supported width as part of error handling. This adaptation can reduce transfer performance because more transfer segments/accesses may be required. From a functional perspective, the transferred data content is unchanged.

[CP_SWS_Dma_00326] Update Transfer Parameters Wrong-State Rejection Behavior

Status: DRAFT

⌈If Dma_UpdateTransferParameters() is called while at least one DmaChannel that uses the addressed DmaTcd (see [CP_SWS_Dma_00012]) is not in DMA_CHANNEL_DISABLED state, the Dma module shall reject the request, shall not modify descriptor data, and shall return E_NOT_OK.⌋

Note

For software-triggered channels, DMA_CHANNEL_DISABLED is the idle state between two transfers: the channel returns to this state after transfer completion, after a recoverable transfer error, and after Dma_StopChannelTransfer(). Descriptor data can therefore be updated directly before the next Dma_StartChannelTransfer() call, without an intermediate disable and enable cycle. For hardware-triggered channels, Dma_DisableChannel() is still required before an update, because a hardware request could otherwise be served with partially updated descriptor data.

[CP_SWS_Dma_00370] Update Transfer Parameters Scatter-Gather Rejection Behavior

Status: DRAFT

⌈If `Dma_UpdateTransferParameters`() is called with a `NextTcdIdx` other than `DMA_TCD_IDX_NONE` and `DmaChannelScatterGatherEnable` is disabled for at least one `DmaChannel` that uses the addressed `DmaTcd` (see [CP_SWS_Dma_00012]), the Dma module shall reject the request, shall not modify the descriptor data, and shall return E_NOT_OK. This is the runtime counterpart of the configuration-time constraint [CP_SWS_Dma_CONSTR_00006].⌋

[CP_SWS_Dma_00392] Update Transfer Parameters Next TCD Link

Status: DRAFT

⌈If `Dma_UpdateTransferParameters`() accepts the update (see [CP_SWS_Dma_00012]), the Dma module shall set the next-`TCD` link of the addressed `DmaTcd` to the value of `NextTcdIdx`.⌋

[CP_SWS_Dma_00393] Update Transfer Parameters Next TCD Validation

Status: DRAFT

⌈If `Dma_UpdateTransferParameters`() is called with a `NextTcdIdx` other than `DMA_TCD_IDX_NONE` that

- does not match the `DmaTcdId` of any configured `DmaTcd`, or
- identifies a `DmaTcd` that is not assigned to the same partition as the addressed `DmaTcd` (see [CP_SWS_Dma_CONSTR_00007]),

the Dma module shall reject the request, shall not modify the descriptor data, and shall return E_NOT_OK. The corresponding development error is specified in [CP_SWS_Dma_00307].⌋

Note

A `NextTcdIdx` that identifies the addressed `DmaTcd` itself or a `DmaTcd` earlier in the same chain creates a cyclic chain. This is not an error (see the note on cyclic chains in Section 7.1.7).

8.3.8 Dma_ChannelTransferConfigUpdate

[CP_SWS_Dma_00114] Definition of API function Dma_ChannelTransferConfigUpdate

Status: DRAFT

⌈

Service Name	Dma_ChannelTransferConfigUpdate (draft)	
Syntax	`Std_ReturnType Dma_ChannelTransferConfigUpdate ( Dma_ChannelIdType ChannelId, Dma_TcdIdxType TcdIdx )`	
Service ID [hex]	0x0E	
Sync/Async	Synchronous	
Reentrancy	Reentrant for different channels; non-reentrant for the same channel	
Parameters (in)	ChannelId	Channel identifier whose transfer configuration shall be updated.
	TcdIdx	TCD index to which the channel default mapping shall be changed.
Parameters (inout)	None	
Parameters (out)	None	
Return value	Std_ReturnType	`E_OK`:remapping accepted; `E_NOT_OK`: remapping rejected.
Description	Allows users to update channel transfer configuration by remapping the default TCD associated to the channel to a different TCD. This service can only be called if the channel is in DMA_CHANNEL_DISABLED state. If the selected TCD mapping leads to invalid scatter-gather chain configuration, the service shall return E_NOT_OK and, when DmaDevErrorDetect is enabled, report DMA_E_PARAM_CONFIG. **Tags:** atp.Status=draft	
Available via	Dma.h	

⌋

[CP_SWS_Dma_00013] Channel Transfer Configuration Update Behavior

Status: DRAFT

⌈When `Dma_ChannelTransferConfigUpdate`() is called in `DMA_CHANNEL_DISABLED` state, the Dma module shall update the channel transfer configuration according to the supplied transfer descriptor configuration. Relevant configuration mappings are `DmaChannelId` for channel selection and `DmaChannelTcdRef` for channel-to-TCD assignment.⌋

[CP_SWS_Dma_00327] Channel Transfer Configuration Update Wrong-State Rejection Behavior

Status: DRAFT

⌈If `Dma_ChannelTransferConfigUpdate`() is called while the addressed channel is not in `DMA_CHANNEL_DISABLED` state, the Dma module shall reject the request, shall keep the channel-to-TCD mapping unchanged, and shall return E_NOT_OK.⌋

[CP_SWS_Dma_00306] Channel Transfer Configuration Update Error Detection

Status: DRAFT

⌈If development error detection is enabled (DmaDevErrorDetect == True) and Dma_ChannelTransferConfigUpdate() is called with invalid or inconsistent configuration data, the function shall raise the development error DMA_E_PARAM_CONFIG. Applicable configuration mappings are DmaChannelId, DmaChannelTcdRef, and DmaTcdNextTcdRef.⌋

8.3.9 Dma_DeInit

[CP_SWS_Dma_00101] Definition of API function Dma_DeInit

Status: DRAFT

⌈

Service Name	Dma_DeInit (draft)

Syntax	`void Dma_DeInit (`
	`  void`
	`)`

Service ID [hex]	0x01
Sync/Async	Synchronous
Reentrancy	Non Reentrant
Parameters (in)	None
Parameters (inout)	None
Parameters (out)	None
Return value	None
Description	De-initializes the DMA Driver: stops ongoing or suspended transfers of software-triggered channels and disables triggering of hardware-triggered channels (see SWS_Dma_00002). **Tags:** atp.Status=draft
Available via	Dma.h

⌋

[CP_SWS_Dma_00002] Module De-initialization Behavior

Status: DRAFT

⌈If Dma_DeInit() is called, the entire module instance shall be de-initialized by stopping any ongoing or suspended transfer and disabling channel triggering for all Dma channels in its scope, with the same effect as Dma_StopChannelTransfer() for software-triggered channels and Dma_DisableChannel() for hardware-triggered channels. This also applies to channels in DMA_CHANNEL_HARDWARE_ERROR state, for which Dma_DeInit() is the only way to leave this state. The DMA hardware shall be brought to a defined idle state such that a subsequent call to Dma_Init() can fully configure the DMA controller without residual hardware state interfering. Thereafter, the module state will be *DMA_UNINIT*.⌋

Informative: Hardware Error Recovery Strategy

A non-recoverable hardware error (channel state DMA_CHANNEL_HARDWARE_ERROR) requires a full module reset by calling Dma_DeInit() followed by Dma_Init(). The driver cannot recover the failed DMA hardware state in place; a channel cannot be re-enabled or re-started while the module remains in *DMA_INIT* state. The de-initialization sequence brings the DMA engine to a defined idle state, returns the module to *DMA_UNINIT*, and allows a fresh configuration to be applied before the channel is used again. This recovery strategy is also illustrated in Figure 7.6 and Figures 7.7, 7.8, and 7.9.

Example: Coordinated Recovery Handling

In practice, the reset is not primarily an *EcuM* task, but a software-level recovery action. After a hardware error is detected, the application or a higher-level coordination entity may first stop dependent software activities, perform DTC reporting, and then call Dma_DeInit() and Dma_Init() before restarting the affected DMA channels. This coordination is required because the DMA engine reset can affect other parallel activities that share the same hardware or the same functional context.

8.3.10 Dma_StartChannelTransfer

[CP_SWS_Dma_00102] Definition of API function Dma_StartChannelTransfer

Status: DRAFT

⌈

Service Name	Dma_StartChannelTransfer (draft)	

Syntax	`Std_ReturnType Dma_StartChannelTransfer (`
	`  Dma_ChannelIdType ChannelId`
	`)`	

Service ID [hex]	0x02	
Sync/Async	Synchronous	
Reentrancy	Reentrant for different channels; non-reentrant for the same channel	
Parameters (in)	ChannelId	Channel identifier for the DMA transfer to be started.
Parameters (inout)	None	
Parameters (out)	None	
Return value	Std_ReturnType	E_OK: transfer start accepted; E_NOT_OK: transfer start rejected, including when the channel is not in DMA_CHANNEL_DISABLED state, the channel is hardware-triggered, scatter-gather descriptor validation fails, or notifications of the channel are disabled (SWS_Dma_00383).
Description	Loads the current channel-to-TCD mapping into the channel and starts the transfer (software-triggered channels only). For scatter-gather configuration, descriptor-chain checks are performed before transfer start; invalid descriptor configuration shall be rejected. **Tags:** atp.Status=draft	

▽

△

Available via	Dma.h

⌋

[CP_SWS_Dma_00004] Start Channel Transfer Behavior

Status: DRAFT

⌈If Dma_StartChannelTransfer() is called for a channel whose DmaChannelOperationMode is DMA_CHANNEL_MODE_SOFTWARE_POLLING or DMA_CHANNEL_MODE_SOFTWARE_INTERRUPT, and the channel is in DMA_CHANNEL_DISABLED state, and the start is not rejected according to [CP_SWS_Dma_00383] or [CP_SWS_Dma_00386], the Dma module shall perform the channel-specific initialization for the current channel-to-TCD mapping (including descriptor data updated via Dma_UpdateTransferParameters() or Dma_ChannelTransferConfigUpdate()), start the transfer, clear channel transfer information, and transition the channel to DMA_CHANNEL_TRANSFER_IN_PROGRESS state as one atomic operation.⌋

Informative: Start Latency of Software-Triggered Transfers

Because Dma_StartChannelTransfer() includes loading the current TCD into the DMA channel, its execution time can be longer than a pure trigger operation. An implementation may skip re-programming descriptor fields that have not changed since the previous start, provided that the resulting transfer is identical to a transfer started from a freshly loaded descriptor. This also covers the restart of a Scatter-gather chain, where the hardware descriptor registers typically hold the last descriptor of the previous chain and have to be reloaded with the first descriptor.

[CP_SWS_Dma_00363] Start Channel Transfer Completion Observation Rule

Status: DRAFT

⌈When Dma_StartChannelTransfer() returns E_OK, the Dma module shall report the transfer completion asynchronously, through the notification callbacks if they are enabled, or through the channel status returned by Dma_GetChannelStatus().⌋

[CP_SWS_Dma_00302] Start Channel Transfer Rejection Behavior

Status: DRAFT

⌈If Dma_StartChannelTransfer() is called for a channel whose DmaChannelOperationMode is DMA_CHANNEL_MODE_SOFTWARE_POLLING or DMA_CHANNEL_MODE_SOFTWARE_INTERRUPT, and the channel is not in DMA_CHANNEL_DISABLED state, the Dma module shall reject the transfer start request, shall keep the channel state unchanged, and shall return E_NOT_OK. If the channel is in DMA_CHANNEL_TRANSFER_IN_PROGRESS or DMA_CHANNEL_SUSPENDED state, the runtime error DMA_E_CHANNEL_BUSY is additionally reported ([CP_SWS_Dma_00211]).⌋

[CP_SWS_Dma_00383] Start Channel Transfer Disabled-Notification Rejection

Status: DRAFT

⌈If `Dma_StartChannelTransfer`() is called for a channel whose `DmaChannelOperationMode` is `DMA_CHANNEL_MODE_SOFTWARE_INTERRUPT` while the notifications of the channel are disabled ([CP_SWS_Dma_00008]), the Dma module shall reject the transfer start request, shall keep the channel in `DMA_CHANNEL_DISABLED` state, and shall return E_NOT_OK.⌋

[CP_SWS_Dma_00386] Start Channel Transfer Incomplete Descriptor Rejection

Status: DRAFT

⌈If `Dma_StartChannelTransfer`() is called for a channel whose `DmaChannelOperationMode` is `DMA_CHANNEL_MODE_SOFTWARE_POLLING` or `DMA_CHANNEL_MODE_SOFTWARE_INTERRUPT`, and the `DmaTcd` of the current channel-to-TCD mapping, or any `DmaTcd` reached from there via the current next-`TCD` links ([CP_SWS_Dma_00391], [CP_SWS_Dma_00392]), is not complete according to [CP_SWS_Dma_00385], the Dma module shall reject the transfer start request, shall not start a transfer, shall keep the channel in `DMA_CHANNEL_DISABLED` state, and shall return E_NOT_OK.⌋

[CP_SWS_Dma_00375] Start Channel Transfer Hardware-Triggered Channel Rejection

Status: DRAFT

⌈If development error detection is enabled (`DmaDevErrorDetect` == True) and `Dma_StartChannelTransfer`() is called for a channel whose `DmaChannelOperationMode` is `DMA_CHANNEL_MODE_HARDWARE`, the function shall raise the development error `DMA_E_INVALID_STATE`.⌋

8.3.11 Dma_StopChannelTransfer

[CP_SWS_Dma_00103] Definition of API function Dma_StopChannelTransfer

Status: DRAFT

⌈

Service Name	Dma_StopChannelTransfer (draft)	
Syntax	`Std_ReturnType Dma_StopChannelTransfer ( Dma_ChannelIdType ChannelId )`	
Service ID [hex]	0x03	
Sync/Async	Synchronous	
Reentrancy	Reentrant for different channels; non-reentrant for the same channel	
Parameters (in)	ChannelId	Channel identifier for the DMA transfer to be stopped.
Parameters (inout)	None	

▽

△

Parameters (out)	None	
Return value	Std_ReturnType	E_OK: transfer stop accepted; E_NOT_OK: transfer stop rejected, including when the channel is in DMA_CHANNEL_HARDWARE_ERROR state (only Dma_DeInit leaves this state, SWS_Dma_00377/00378).
Description	Stops or aborts an ongoing or suspended transfer on the given software-triggered channel and transitions the channel to DMA_CHANNEL_DISABLED. **Tags:** atp.Status=draft	
Available via	Dma.h	

⌋

[CP_SWS_Dma_00005] Stop Channel Transfer Behavior

Status: DRAFT

⌈When Dma_StopChannelTransfer() is called for a channel whose DmaChannelOperationMode is DMA_CHANNEL_MODE_SOFTWARE_POLLING or DMA_CHANNEL_MODE_SOFTWARE_INTERRUPT, and the channel is not in DMA_CHANNEL_HARDWARE_ERROR state, the Dma module shall stop or abort the channel transfer (also a suspended transfer) and set the channel state to DMA_CHANNEL_DISABLED.⌋

[CP_SWS_Dma_00377] Stop Channel Transfer Hardware-Error State Rejection

Status: DRAFT

⌈If Dma_StopChannelTransfer() is called for a channel in DMA_CHANNEL_HARDWARE_ERROR state, the Dma module shall reject the request, shall keep the channel in DMA_CHANNEL_HARDWARE_ERROR state, and shall return E_NOT_OK. This state can only be left by Dma_DeInit().⌋

[CP_SWS_Dma_00376] Stop Channel Transfer Hardware-Triggered Channel Rejection

Status: DRAFT

⌈If development error detection is enabled (DmaDevErrorDetect == True) and Dma_StopChannelTransfer() is called for a channel whose DmaChannelOperationMode is DMA_CHANNEL_MODE_HARDWARE, the function shall raise the development error DMA_E_INVALID_STATE.⌋

8.3.12 Dma_GetChannelStatus

[CP_SWS_Dma_00104] Definition of API function Dma_GetChannelStatus

Status: DRAFT

⌈

Service Name	Dma_GetChannelStatus (draft)	

Syntax	`void Dma_GetChannelStatus (`
	`  Dma_ChannelIdType ChannelId,`
	`  Dma_ChannelStatusType* ChannelStatusPtr`
	`)`	

Service ID [hex]	0x04	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	ChannelId	Channel identifier for the queried runtime status.
Parameters (inout)	None	
Parameters (out)	ChannelStatusPtr	Pointer to output variable that receives the current channel status.
Return value	None	
Description	Provides the status of the DMA transfer on the given channel through the output pointer for software-triggered channels. If the driver-managed channel status is DMA_CHANNEL_TRANSFER_IN_PROGRESS, Dma_GetChannelStatus evaluates the hardware transfer state and returns DMA_CHANNEL_TRANSFER_IN_PROGRESS, DMA_CHANNEL_TRANSFER_COMPLETED, DMA_CHANNEL_TRANSFER_ERROR, or DMA_CHANNEL_HARDWARE_ERROR according to the observed result. When the channel is not in DMA_CHANNEL_TRANSFER_IN_PROGRESS, the function returns the corresponding current status value. For hardware-triggered channels, this API is forbidden and development error detection applies as specified below. **Tags:** atp.Status=draft	
Available via	Dma.h	

⌋

[CP_SWS_Dma_00006] Get Channel Status Behavior

Status: DRAFT

⌈When `Dma_GetChannelStatus`() is called, the Dma module shall provide the current channel status value through the output parameter for channels whose `Dma_ChannelOperationMode` is `DMA_CHANNEL_MODE_SOFTWARE_POLLING` or `DMA_CHANNEL_MODE_SOFTWARE_INTERRUPT`.⌋

[CP_SWS_Dma_00330] Get Channel Status Ongoing-Transfer Polling Behavior

Status: DRAFT

⌈If `Dma_GetChannelStatus`() is called while the current driver-managed channel status is `DMA_CHANNEL_TRANSFER_IN_PROGRESS` and the retrieved hardware transfer state indicates that the transfer is still executing, the Dma module shall keep the driver-managed channel status in `DMA_CHANNEL_TRANSFER_IN_PROGRESS` and shall return `DMA_CHANNEL_TRANSFER_IN_PROGRESS`.⌋

[CP_SWS_Dma_00331] Get Channel Status Successful-Completion Polling Behavior

Status: DRAFT

⌈If Dma_GetChannelStatus() is called while the current driver-managed channel status is DMA_CHANNEL_TRANSFER_IN_PROGRESS and the retrieved hardware transfer state indicates successful completion, the Dma module shall transition the channel state to DMA_CHANNEL_DISABLED and shall return DMA_CHANNEL_TRANSFER_COMPLETED.⌋

[CP_SWS_Dma_00332] Get Channel Status Transfer-Error Polling Behavior

Status: DRAFT

⌈If Dma_GetChannelStatus() is called while the current driver-managed channel status is DMA_CHANNEL_TRANSFER_IN_PROGRESS and the retrieved hardware transfer state indicates a recoverable transfer error, the Dma module shall transition the channel state to DMA_CHANNEL_DISABLED and shall return DMA_CHANNEL_TRANSFER_ERROR.⌋

[CP_SWS_Dma_00333] Get Channel Status Hardware-Error Polling Behavior

Status: DRAFT

⌈If Dma_GetChannelStatus() is called while the current driver-managed channel status is DMA_CHANNEL_TRANSFER_IN_PROGRESS and the retrieved hardware transfer state indicates a hardware error, the Dma module shall transition the channel state to DMA_CHANNEL_HARDWARE_ERROR, keep the channel in this latched error state until Dma_DeInit() is called, and shall return DMA_CHANNEL_HARDWARE_ERROR.⌋

Informative: Polling-Mode Status Synchronization Rationale

In polling mode, hardware transfer progress can advance independently from the driver-managed channel state until Dma_GetChannelStatus() is called. Therefore, this API acts as synchronization point between software and hardware view. During this synchronization, the API may return a transfer result value (for example DMA_CHANNEL_TRANSFER_COMPLETED, DMA_CHANNEL_TRANSFER_ERROR, or DMA_CHANNEL_HARDWARE_ERROR) while transitioning the driver-managed channel state to the next operational state. In most cases this next state is DMA_CHANNEL_DISABLED; for non-recoverable hardware error it is DMA_CHANNEL_HARDWARE_ERROR. No additional persistent channel states are introduced for these short-lived observation points.

Informative: Reporting of the Transfer Result

DMA_CHANNEL_DISABLED only indicates that a software-triggered channel is ready for the next transfer; it does not indicate how the previous transfer ended. The result of a transfer is reported exactly once:

- for a channel whose DmaChannelOperationMode is DMA_CHANNEL_MODE_SOFTWARE_INTERRUPT or DMA_CHANNEL_MODE_HARDWARE, by the corresponding notification callback (Dma_TransferCompleteNotification(), Dma_TransferErrorNotification(), or Dma_HardwareErrorNotification());
- for a channel whose DmaChannelOperationMode is DMA_CHANNEL_MODE_SOFTWARE_POLLING, by the single Dma_GetChannelStatus() call that observes the end of the transfer ([CP_SWS_Dma_00331], [CP_SWS_Dma_00332], [CP_SWS_Dma_00333]).

The application is responsible for retaining this result if it is needed later. A transfer aborted by Dma_StopChannelTransfer() is known to the caller, and a hardware error remains visible as the latched DMA_CHANNEL_HARDWARE_ERROR state until Dma_DeInit().

Although Dma_GetChannelStatus() is reentrant, the result of a polled transfer is returned only to the first caller that observes the end of the transfer; subsequent callers already obtain DMA_CHANNEL_DISABLED. Therefore only one software context should poll the status of a given channel.

[CP_SWS_Dma_00334] Get Channel Status Hardware-Triggered Channel Rejection

Status: DRAFT

⌈If development error detection is enabled (DmaDevErrorDetect == True) and Dma_GetChannelStatus() is called for a channel whose DmaChannelOperationMode is DMA_CHANNEL_MODE_HARDWARE, the function shall raise the development error DMA_E_INVALID_STATE.⌋

8.3.13 Dma_EnableChannel

[CP_SWS_Dma_00106] Definition of API function Dma_EnableChannel

Status: DRAFT

⌈

Service Name	Dma_EnableChannel (draft)	
Syntax	`Std_ReturnType Dma_EnableChannel ( Dma_ChannelIdType ChannelId )`	
Service ID [hex]	0x06	
Sync/Async	Synchronous	
Reentrancy	Reentrant for different channels; non-reentrant for the same channel	
Parameters (in)	ChannelId	Channel identifier to be enabled.
Parameters (inout)	None	

▽

△

Parameters (out)	None	
Return value	Std_ReturnType	E_OK: trigger enable accepted; E_NOT_OK: trigger enable rejected, e.g. channel not in DMA_CHANNEL_DISABLED, channel is software-triggered, or notifications of the channel are disabled (SWS_Dma_00384).
Description	Performs the channel-specific initialization for the current channel-to-TCD mapping and arms the channel for hardware triggering. Only applicable to channels with ChannelOperationMode DMA_CHANNEL_MODE_HARDWARE. **Tags:** atp.Status=draft	
Available via	Dma.h	

⌋

[CP_SWS_Dma_00003] Enable Channel Behavior

Status: DRAFT

⌈When Dma_EnableChannel() is called while the module is in DMA_INIT state for a channel whose DmaChannelOperationMode is DMA_CHANNEL_MODE_HARDWARE, the channel is in DMA_CHANNEL_DISABLED state, the notifications of the channel are enabled, and the request is not rejected according to [CP_SWS_Dma_00387], the Dma module shall perform the channel-specific initialization for the current channel-to-TCD mapping, because the channel-to-TCD mapping can be changed at runtime. The Dma module shall then enable the channel for hardware triggering and clear channel transfer information.⌋

[CP_SWS_Dma_00309] Enable Channel State Transition Behavior

Status: DRAFT

⌈When Dma_EnableChannel() is called while the module is in DMA_INIT state for a channel whose DmaChannelOperationMode is DMA_CHANNEL_MODE_HARDWARE, the channel is in DMA_CHANNEL_DISABLED state, the notifications of the channel are enabled, and the request is not rejected according to [CP_SWS_Dma_00387], the Dma module shall transition the channel to DMA_CHANNEL_WAITING_TRIGGER state.⌋

[CP_SWS_Dma_00379] Enable Channel Rejection Behavior

Status: DRAFT

⌈If Dma_EnableChannel() is called for a channel whose DmaChannelOperation-Mode is DMA_CHANNEL_MODE_HARDWARE, and the channel is not in DMA_CHANNEL_DISABLED state (for example DMA_CHANNEL_WAITING_TRIGGER or DMA_CHANNEL_HARDWARE_ERROR), the Dma module shall reject the request, shall not load descriptor data, shall keep the channel state unchanged, and shall return E_NOT_OK.⌋

[CP_SWS_Dma_00384] Enable Channel Disabled-Notification Rejection

Status: DRAFT

⌈If Dma_EnableChannel() is called for a channel whose DmaChannelOperation-Mode is DMA_CHANNEL_MODE_HARDWARE while the notifications of the channel are disabled ([CP_SWS_Dma_00008]), the Dma module shall reject the request, shall not

arm the channel for hardware triggering, shall keep the channel in DMA_CHANNEL_DISABLED state, and shall return E_NOT_OK.⌋

[CP_SWS_Dma_00387] Enable Channel Incomplete Descriptor Rejection

Status: DRAFT

⌈If Dma_EnableChannel() is called for a channel whose DmaChannelOperationMode is DMA_CHANNEL_MODE_HARDWARE, and the DmaTcd of the current channel-to-TCD mapping, or any DmaTcd reached from there via the current next-TCD links ([CP_SWS_Dma_00391], [CP_SWS_Dma_00392]), is not complete according to [CP_SWS_Dma_00385], the Dma module shall reject the request, shall not arm the channel for hardware triggering, shall keep the channel in DMA_CHANNEL_DISABLED state, and shall return E_NOT_OK.⌋

[CP_SWS_Dma_00373] Enable Channel Software-Triggered Channel Rejection

Status: DRAFT

⌈If development error detection is enabled (DmaDevErrorDetect == True) and Dma_EnableChannel() is called for a channel whose DmaChannelOperationMode is DMA_CHANNEL_MODE_SOFTWARE_POLLING or DMA_CHANNEL_MODE_SOFTWARE_INTERRUPT, the function shall raise the development error DMA_E_INVALID_STATE.⌋

8.3.14 Dma_DisableChannel

[CP_SWS_Dma_00107] Definition of API function Dma_DisableChannel

Status: DRAFT

⌈

Service Name	Dma_DisableChannel (draft)	
Syntax	`Std_ReturnType Dma_DisableChannel ( Dma_ChannelIdType ChannelId )`	
Service ID [hex]	0x07	
Sync/Async	Synchronous	
Reentrancy	Reentrant for different channels; non-reentrant for the same channel	
Parameters (in)	ChannelId	Channel identifier to be disabled.
Parameters (inout)	None	
Parameters (out)	None	
Return value	Std_ReturnType	E_OK: trigger disable accepted, including successful stop/abort of an ongoing transfer when present; E_NOT_OK: trigger disable rejected, including when the channel is in DMA_CHANNEL_HARDWARE_ERROR state (only Dma_DeInit leaves this state, SWS_Dma_00377/00378)

▽

△

Description	Disables channel functionality. If a DMA transfer is ongoing, the transfer is stopped or aborted before transitioning the channel to DMA_CHANNEL_DISABLED. No DMA transfers can be executed until Dma_EnableChannel is called. Only applicable to hardware-triggered channels. **Tags:** atp.Status=draft
Available via	Dma.h

⌋

[CP_SWS_Dma_00014] Disable Channel Behavior

Status: DRAFT

⌈When `Dma_DisableChannel`() is called for a channel whose `DmaChannelOperationMode` is `DMA_CHANNEL_MODE_HARDWARE`, and the channel is not in `DMA_CHANNEL_HARDWARE_ERROR` state, the Dma module shall disable channel hardware triggering. If a channel transfer is ongoing, the Dma module shall stop or abort the transfer before setting the channel state to `DMA_CHANNEL_DISABLED`.⌋

[CP_SWS_Dma_00378] Disable Channel Hardware-Error State Rejection

Status: DRAFT

⌈If `Dma_DisableChannel`() is called for a channel in `DMA_CHANNEL_HARDWARE_ERROR` state, the Dma module shall reject the request, shall keep the channel in `DMA_CHANNEL_HARDWARE_ERROR` state, and shall return E_NOT_OK. This state can only be left by `Dma_DeInit`().⌋

[CP_SWS_Dma_00374] Disable Channel Software-Triggered Channel Rejection

Status: DRAFT

⌈If development error detection is enabled (`DmaDevErrorDetect` == True) and `Dma_DisableChannel`() is called for a channel whose `DmaChannelOperationMode` is `DMA_CHANNEL_MODE_SOFTWARE_POLLING` or `DMA_CHANNEL_MODE_SOFTWARE_INTERRUPT`, the function shall raise the development error `DMA_E_INVALID_STATE`.⌋

8.3.15 Dma_SuspendChannelTransfer

[CP_SWS_Dma_00115] Definition of API function Dma_SuspendChannelTransfer

Status: DRAFT

⌈

Service Name	Dma_SuspendChannelTransfer (draft)

Syntax	`Std_ReturnType Dma_SuspendChannelTransfer (`
	`  Dma_ChannelIdType ChannelId`
	`)`

Service ID [hex]	0x0F
Sync/Async	Synchronous
Reentrancy	Reentrant for different channels; non-reentrant for the same channel

▽

△

Parameters (in)	ChannelId	DMA channel identifier (Dma_ChannelIdType)
Parameters (inout)	None	
Parameters (out)	None	
Return value	Std_ReturnType	E_OK: suspension successful E_NOT_OK: suspension failed or channel not in active transfer state.
Description	Suspends the DMA transfer on the given software-triggered channel. Channel suspension requires a transfer to have been initiated on that channel via Dma_StartChannelTransfer. This service is specified only for software-triggered channels. For hardware-triggered channels, any suspend capability is hardware- and integration-specific and is outside the guaranteed semantics of this API. Optional service. **Tags:** atp.Status=draft	
Available via	Dma.h	

⌋

[CP_SWS_Dma_00015] Suspend Channel Transfer Behavior

Status: DRAFT

⌈When Dma_SuspendChannelTransfer() is called for a channel whose DmaChannelOperationMode is DMA_CHANNEL_MODE_SOFTWARE_POLLING or DMA_CHANNEL_MODE_SOFTWARE_INTERRUPT, and the channel is in DMA_CHANNEL_TRANSFER_IN_PROGRESS state and suspension is supported, the Dma module shall suspend the transfer and transition the channel to DMA_CHANNEL_SUSPENDED state.⌋

[CP_SWS_Dma_00371] Suspend Channel Transfer Hardware-Triggered Channel Rejection

Status: DRAFT

⌈If development error detection is enabled (DmaDevErrorDetect == True) and Dma_SuspendChannelTransfer() is called for a channel whose DmaChannelOperationMode is DMA_CHANNEL_MODE_HARDWARE, the function shall raise the development error DMA_E_INVALID_STATE.⌋

8.3.16 Dma_ResumeChannelTransfer

[CP_SWS_Dma_00116] Definition of API function Dma_ResumeChannelTransfer

Status: DRAFT

⌈

Service Name	Dma_ResumeChannelTransfer (draft)

Syntax	`Std_ReturnType Dma_ResumeChannelTransfer (`
	`  Dma_ChannelIdType ChannelId`
	`)`

Service ID [hex]	0x10
Sync/Async	Synchronous
Reentrancy	Reentrant for different channels; non-reentrant for the same channel

▽

△

Parameters (in)	ChannelId	DMA channel identifier (Dma_ChannelIdType)
Parameters (inout)	None	
Parameters (out)	None	
Return value	Std_ReturnType	`E_OK`: resumption successful; `E_NOT_OK`: resumption failed or channel not in suspended state.
Description	Resumes the DMA transfer on the given software-triggered channel for which the transfer had been suspended via Dma_SuspendChannelTransfer. This service only applies to channels in suspended state and continues the transfer from the point of suspension. Optional service. **Tags:** atp.Status=draft	
Available via	Dma.h	

⌋

[CP_SWS_Dma_00016] Resume Channel Transfer Behavior

Status: DRAFT

⌈When `Dma_ResumeChannelTransfer`() is called for a channel whose `DmaChannelOperationMode` is `DMA_CHANNEL_MODE_SOFTWARE_POLLING` or `DMA_CHANNEL_MODE_SOFTWARE_INTERRUPT`, and the channel is in `DMA_CHANNEL_SUSPENDED` state and resume is supported, the Dma module shall resume the transfer and transition the channel back to `DMA_CHANNEL_TRANSFER_IN_PROGRESS` state.⌋

[CP_SWS_Dma_00372] Resume Channel Transfer Hardware-Triggered Channel Rejection

Status: DRAFT

⌈If development error detection is enabled (`DmaDevErrorDetect` == True) and `Dma_ResumeChannelTransfer`() is called for a channel whose `DmaChannelOperationMode` is `DMA_CHANNEL_MODE_HARDWARE`, the function shall raise the development error `DMA_E_INVALID_STATE`.⌋

8.4 Callback notifications

This is a list of functions provided for other modules.

8.5 Scheduled functions

These functions are directly called by Basic Software Scheduler. The following functions have no return value and no parameter. All functions are non reentrant.

8.6 Expected interfaces

In this chapter all interfaces required from other modules are listed.

8.6.1 Mandatory interfaces

The DMA Driver module requires some interfaces to fulfill its core functionality.

[CP_SWS_Dma_00128] Definition of mandatory interfaces required by module Dma

Status: DRAFT

⌈

API Function	*Header File*	*Description*
Det_ReportRuntimeError	Det.h	Service to report runtime errors. If a callout has been configured then this callout shall be called.

⌋

8.6.2 Optional interfaces

This section defines all interfaces which are required to fulfill an optional functionality of the

[CP_SWS_Dma_00129] Definition of optional interfaces requested by module Dma

Status: DRAFT

⌈

API Function	*Header File*	*Description*
Det_ReportError	Det.h	Service to report development errors.

⌋

8.6.3 Configurable interfaces

In this section, all interfaces are listed where the target function could be configured. The target function is usually a callback function. The names of this kind of interfaces are not fixed because they are configurable.

Same Callback Configured for Multiple Channels

The same function name may be configured as the target callback for more than one `DMA channel` (for example, the same application function configured as `DmaChannelCompletedNotificationCallback` for several channels). In this case, that function can effectively be called reentrant, even though each individual callback interface is specified as non-reentrant: if the underlying events occur close together (for example nested interrupts, or concurrently on a multi-core system), the driver may invoke the same configured function again before a previous invocation for another

channel has returned. The implementation of a callback function shared across channels needs to safely handle such concurrent or nested invocations and needs to use the `Channel` parameter to distinguish the reporting channel.

8.6.3.1 Transfer completion callback interface

The target callback function for transfer completion, denoted `<Dma_TransferCompleteNotification>` in this specification, is configurable per channel via `DmaChannelCompletedNotificationCallback`. It is used only if the callback is configured and notifications of the channel are enabled, which is the default after `Dma_Init`() ([CP_SWS_Dma_00380]).

[CP_SWS_Dma_00123] Definition of configurable interface <Dma_TransferCompleteNotification>

Status: DRAFT

⌈

Service Name	<Dma_TransferCompleteNotification> (draft)	
Syntax	`void <Dma_TransferCompleteNotification> ( Dma_ChannelIdType Channel )`	
Sync/Async	Synchronous	
Reentrancy	Non Reentrant	
Parameters (in)	Channel	Channel identifier of the completed DMA transfer.
Parameters (inout)	None	
Parameters (out)	None	
Return value	None	
Description	Callback function configured per channel and called when a DMA transfer on the respective channel is completed successfully, provided the callback is configured and notifications are enabled. In polling mode, this callback is not used. **Tags:** atp.Status=draft	
Available via	Dma.h	

⌋

8.6.3.2 Transfer error callback interface

The target callback function for transfer error, denoted `<Dma_TransferErrorNotification>` in this specification, is configurable per channel via `DmaChannelTransferErrorNotificationCallback`. It is used only if the callback is configured and notifications of the channel are enabled, which is the default after `Dma_Init`() ([CP_SWS_Dma_00380]).

[CP_SWS_Dma_00124] Definition of configurable interface <Dma_TransferError Notification>

Status: DRAFT

⌈

Service Name	<Dma_TransferErrorNotification> (draft)	
Syntax	`void <Dma_TransferErrorNotification> ( Dma_ChannelIdType Channel )`	
Sync/Async	Synchronous	
Reentrancy	Non Reentrant	
Parameters (in)	Channel	Channel identifier for which a DMA transfer error was detected.
Parameters (inout)	None	
Parameters (out)	None	
Return value	None	
Description	Callback function configured per channel and called when a DMA transfer error is detected for the respective channel, provided the callback is configured and notifications are enabled. In polling mode, this callback is not used. **Tags:** atp.Status=draft	
Available via	Dma.h	

⌋

[CP_SWS_Dma_00310] Transfer Completion Notification Reporting

Status: DRAFT

⌈

When a configured channel transfer reaches completion and the notifications of that channel are enabled (default after `Dma_Init`(), see [CP_SWS_Dma_00380]), the Dma module shall call `Dma_TransferCompleteNotification`() for the respective channel if the callback is configured.⌋

[CP_SWS_Dma_00311] Transfer Error Notification Reporting

Status: DRAFT

⌈

When a configured channel transfer reaches error indication and the notifications of that channel are enabled (default after `Dma_Init`(), see [CP_SWS_Dma_00380]), the Dma module shall call `Dma_TransferErrorNotification`() for the respective channel if the callback is configured.⌋

[CP_SWS_Dma_00312] Polling Status Reflection Behavior

Status: DRAFT

⌈

For a channel whose `DmaChannelOperationMode` is `DMA_CHANNEL_MODE_SOFTWARE_POLLING`, the Dma module shall report completion and error conditions via the channel status returned by `Dma_GetChannelStatus`(). The transfer notification callbacks are not called for this mode.⌋

Informative: CRC Result Availability

When a DMA transfer error is detected, the application can query the CRC result using `Dma_GetCrcResult`() in two scenarios:

1. **Interrupt-driven path (ISR-based notification):** Within or immediately after the `Dma_TransferErrorNotification`() callback, the CRC result for the erroneous transfer is available and can be queried. This allows the application to determine whether the error is related to CRC verification failure.
2. **Polling path (no ISR):** After `Dma_GetChannelStatus`() returns a negative result status (`DMA_CHANNEL_TRANSFER_ERROR` or `DMA_CHANNEL_HARDWARE_ERROR`), the CRC result for the current or last transfer can be queried. The CRC result reflects the status of the most recently completed transfer on the channel.

In both cases, `Dma_GetCrcResult`() returns E_OK and provides valid CRC information if CRC support is configured for the channel and a CRC result is available.

8.6.3.3 Hardware error callback interface

The target callback function for hardware error, denoted `<Dma_HardwareErrorNotification>` in this specification, is configurable per channel via `DmaChannelHardwareErrorNotificationCallback`. It is used only if the callback is configured and notifications of the channel are enabled, which is the default after `Dma_Init`() ([CP_SWS_Dma_00380]).

[CP_SWS_Dma_00125] Definition of configurable interface <Dma_HardwareErrorNotification>

Status: DRAFT

⌈

Service Name	<Dma_HardwareErrorNotification> (draft)	
Syntax	`void <Dma_HardwareErrorNotification> ( Dma_ChannelIdType Channel )`	
Sync/Async	Synchronous	
Reentrancy	Non Reentrant	
Parameters (in)	Channel	Channel identifier for which a DMA hardware error was detected.
Parameters (inout)	None	
Parameters (out)	None	
Return value	None	
Description	Callback function configured per channel and called when a DMA hardware error is detected for the respective channel, provided the callback is configured and notifications are enabled. In polling mode, this callback is not used. A hardware error indicates a bus-level or peripheral fault that is distinct from a DMA transfer configuration error. **Tags:** atp.Status=draft	
Available via	Dma.h	

⌋

[CP_SWS_Dma_00328] Hardware Error Notification Reporting

Status: DRAFT

⌈

When a DMA hardware error is detected on a configured channel and the notifications of that channel are enabled (default after `Dma_Init`(), see [CP_SWS_Dma_00380]), the Dma module shall call `Dma_HardwareErrorNotification`() for the respective channel if the callback is configured.⌋

8.7 Service Interfaces

The Dma MCAL Driver does not define AUTOSAR software component service interfaces (Sender-Receiver or Client-Server interfaces) and does not define AUTOSAR ports.

9 Sequence diagrams

9.1 Reading Convention for Sequence Numbers

Some sequence diagrams in this chapter use message numbering. Where shown, the displayed numbers indicate only chronological message order inside the specific diagram.

The numbering is interpreted as follows:

- Numbers are local to one diagram and restart at the beginning of that diagram.
- Numbers are not requirement identifiers and are not trace IDs.
- The semantic meaning of a step is given by the message text and the enclosing group/alt/loop block title.
- If a diagram is edited, numbering may shift automatically; therefore stable references use figure labels and message text, not absolute step numbers.

9.2 Software-Triggered Transfer Sequences

This section documents the nominal interaction flow for `Software-triggered transfer` channel usage, including initialization, optional transfer preparation, transfer execution, optional suspend/resume handling, start of the next transfer, and de-initialization.

9.2.1 Sequence Diagrams

Figure 9.1 shows the setup and transfer execution, Figure 9.2 shows the suspend/resume handling and the transfer completion.

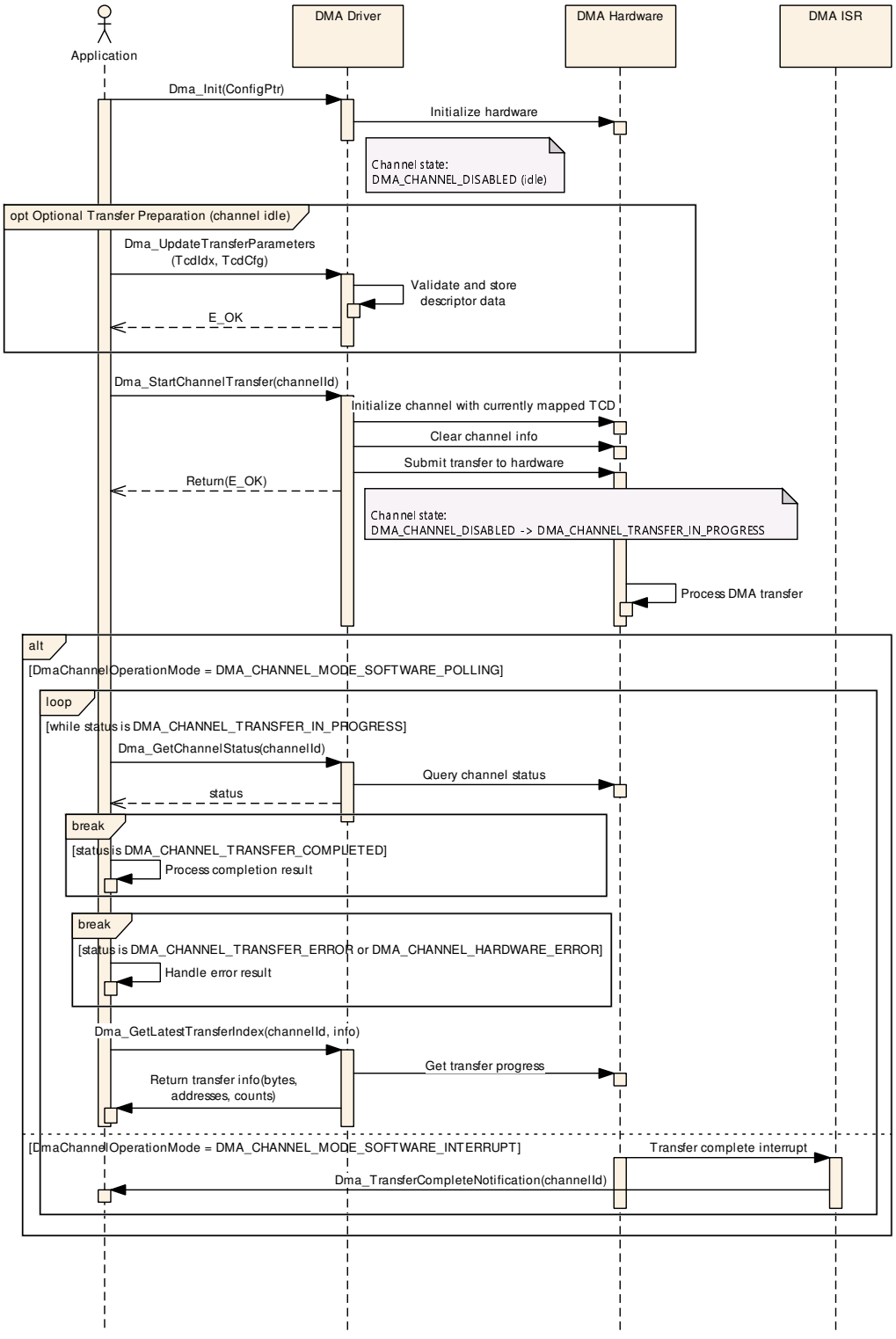


Figure 9.1: Use-case 1: Software-triggered DMA setup and transfer execution

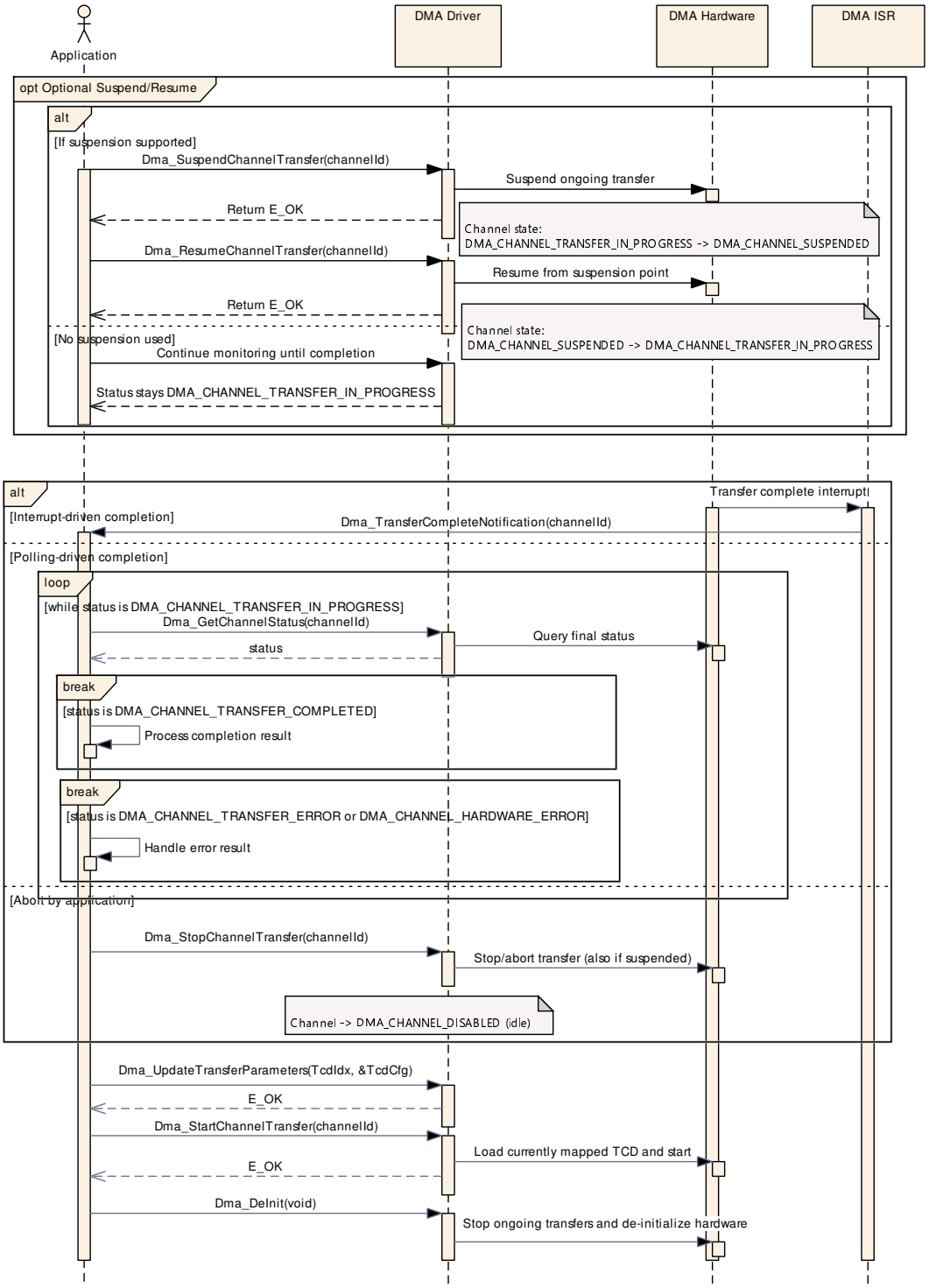


Figure 9.2: Use-case 2: Software-triggered DMA suspend/resume and completion

9.3 `Scatter-gather` Sequence Diagrams

This section groups the nominal and error-related `Scatter-gather` interaction flows. All figures are designed for DIN A4 readability.

9.3.1 Scatter-Gather Chain Setup and Transfer Start

This subsection is informative. It shows how an application sets up a Scatter-gather chain at runtime and starts it (Figure 9.3). The chain consists of configured DmaTcd containers only; they are identified by their DmaTcdId:

- During Dma_Init(), the next-TCD link of each DmaTcd is initialized from DmaTcdNextTcdRef ([CP_SWS_Dma_00391]). A chain that is completely defined in the ECU configuration can be started without further preparation.
- To build or change a chain at runtime, the application uses the read-modify-write workflow (Dma_GetTransferParameters(), Dma_UpdateTransferParameters()) and sets NextTcdIdx to the DmaTcdId of the successor, or to DMA_TCD_IDX_NONE for the last TCD. Each link is validated when it is set ([CP_SWS_Dma_00370], [CP_SWS_Dma_00393]). Updating the chain tail first ensures that every link refers to a TCD whose transfer parameters are already complete.
- Dma_ChannelTransferConfigUpdate() selects the head of the chain, and Dma_StartChannelTransfer() checks that all TCDs of the chain are complete ([CP_SWS_Dma_00386]) before the hardware processes the chain autonomously.
- For further transfers with the same chain, only the buffer-related parameters of the affected TCDs need to be updated.

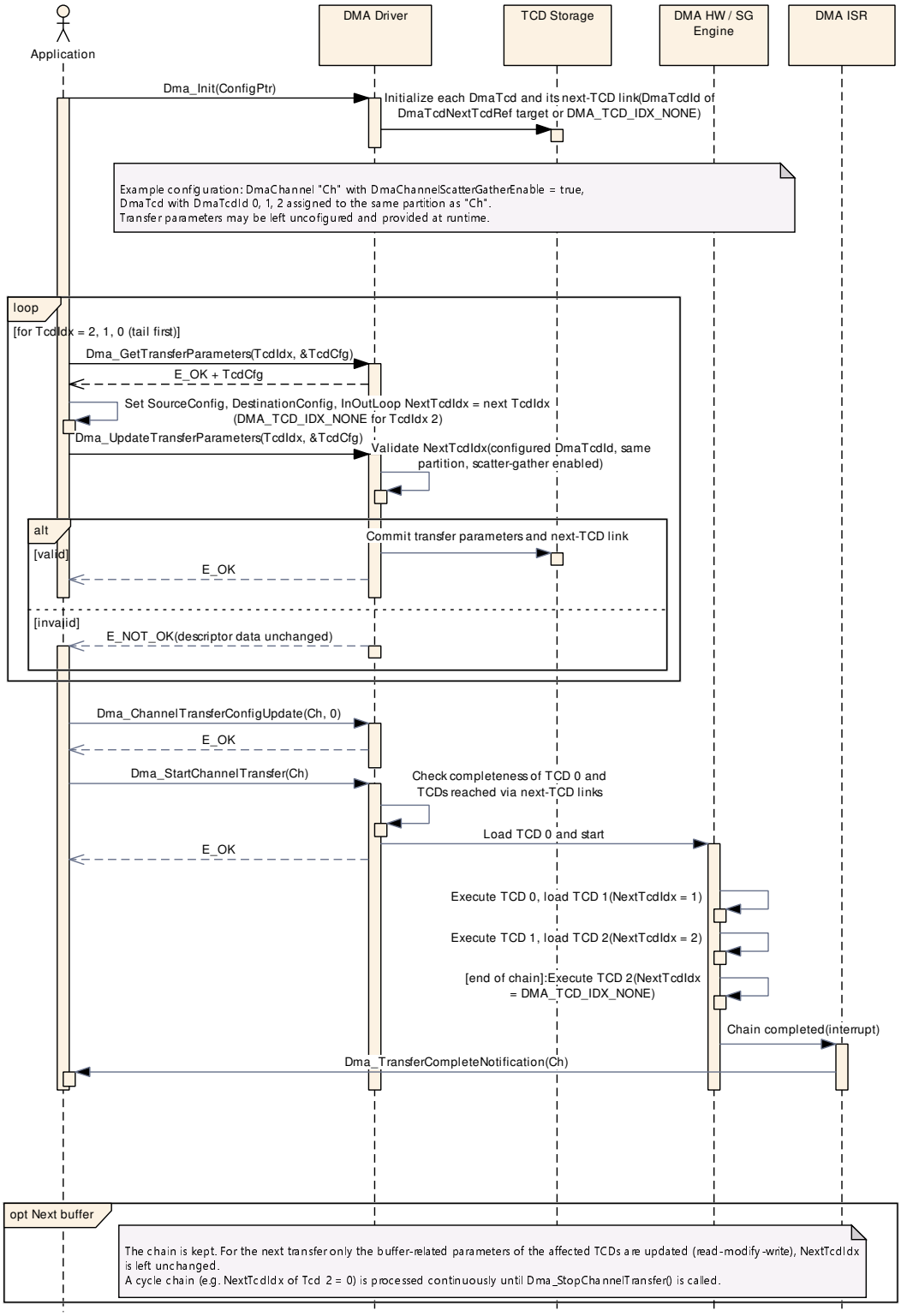


Figure 9.3: `Scatter-gather` chain setup via NextTcdIdx and transfer start

9.3.2 Regular Scatter-Gather Nominal Path

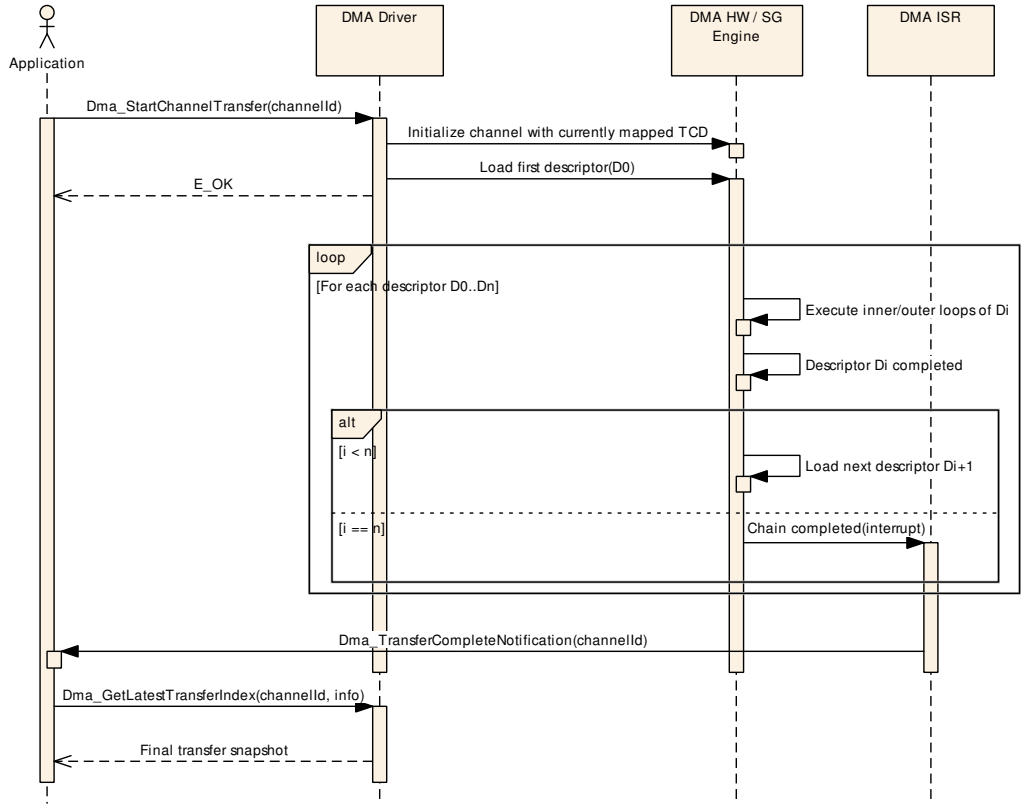


Figure 9.4: Regular `Scatter-gather` `TCDs` execution (nominal path)

9.3.3 Scatter-Gather CRC Failure Path

This subsection documents the runtime sequence when CRC verification fails in `TCD` Dn during `Scatter-gather` processing. The flow shows the best case, in which processing stops at Dn and Dn+1 is not loaded automatically. Depending on the DMA hardware, the next descriptor may already have been loaded before the CRC failure is detected; this specification therefore does not require a particular stop point. The normative behavior is limited to stopping further descriptor processing and reporting the transfer error ([CP_SWS_Dma_00030], [CP_SWS_Dma_00316]).

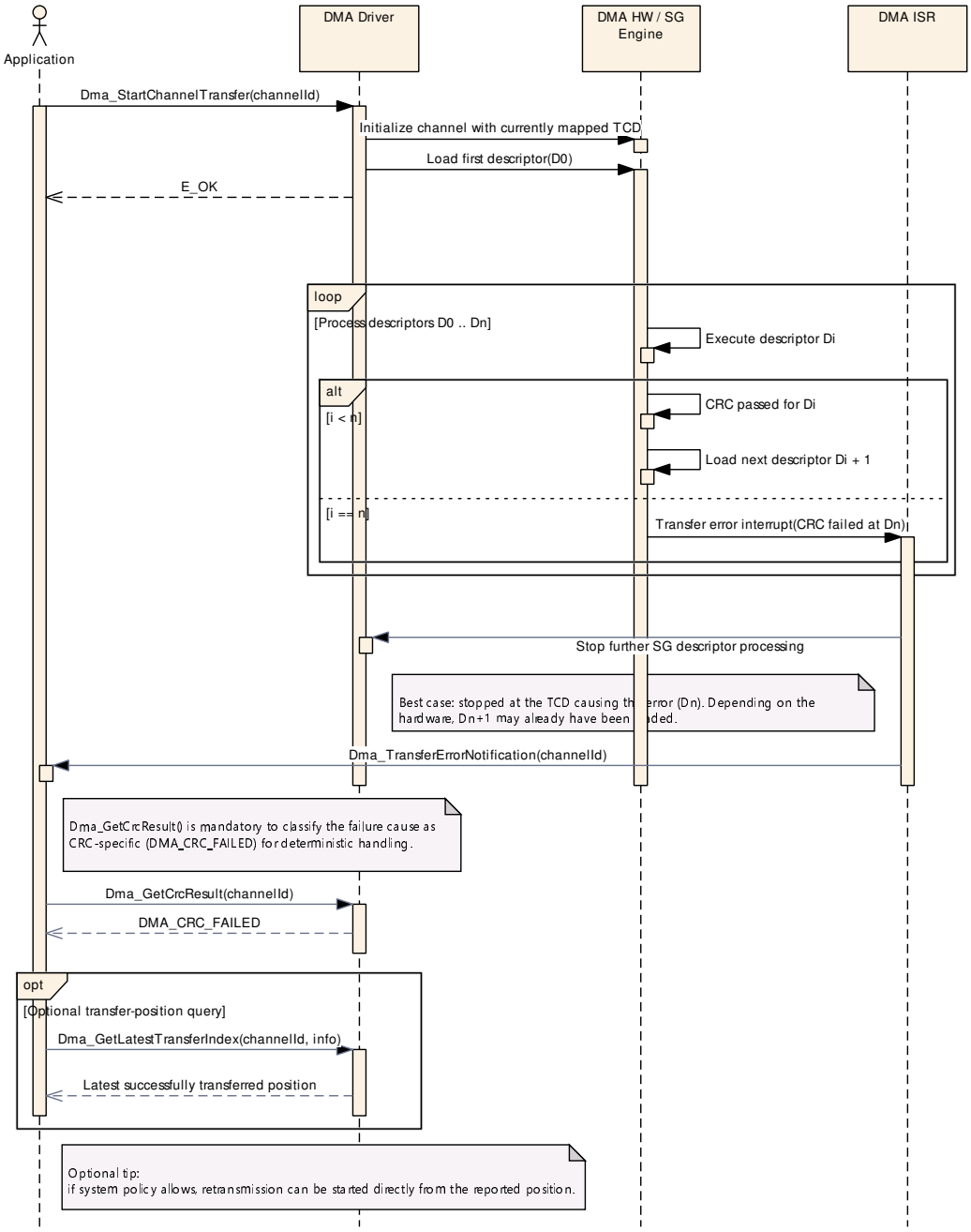


Figure 9.5: `Scatter-gather` CRC failure at `TCD` Dn

9.3.4 Recovery After Scatter-Gather Runtime Error

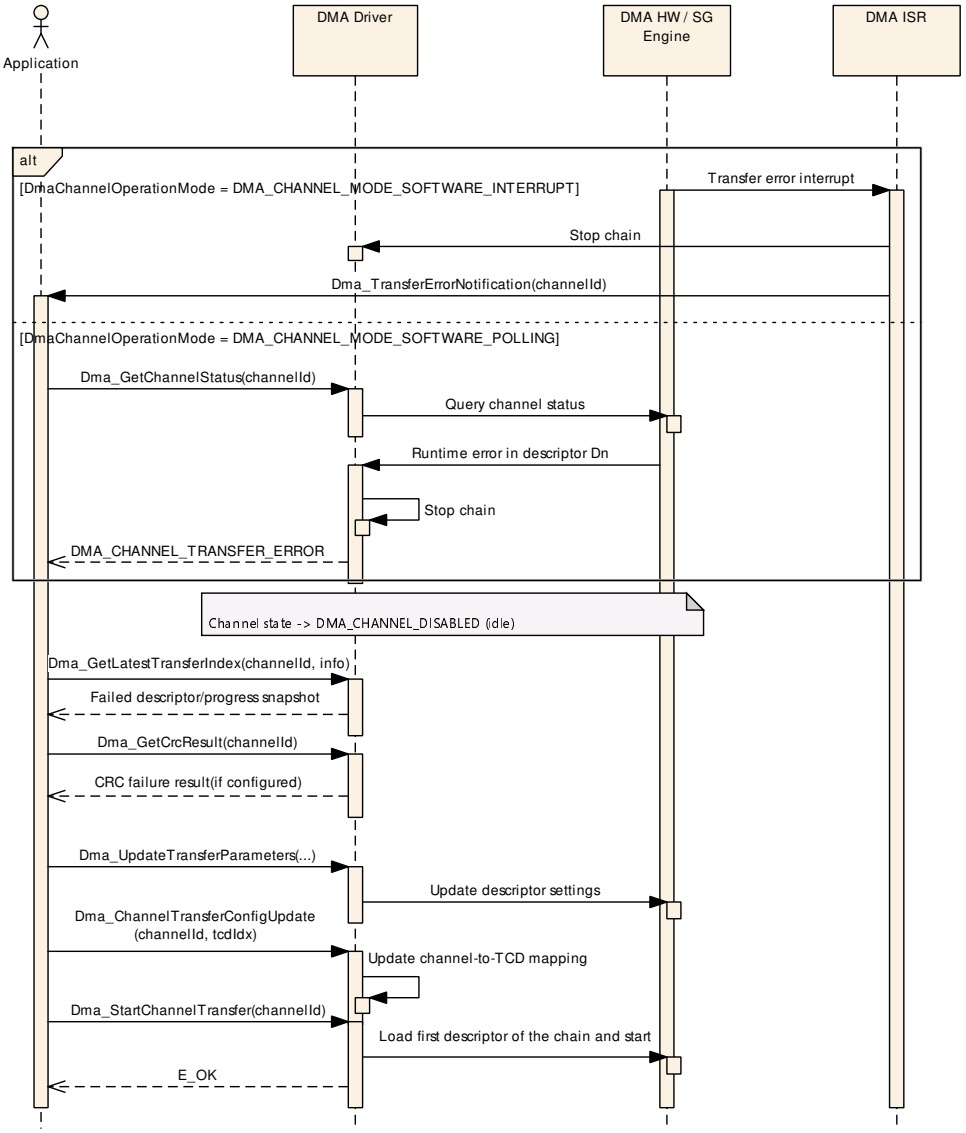


Figure 9.6: Recovery flow after `Scatter-gather` runtime error

9.4 Hardware-Triggered Transfer with Trigger Gating

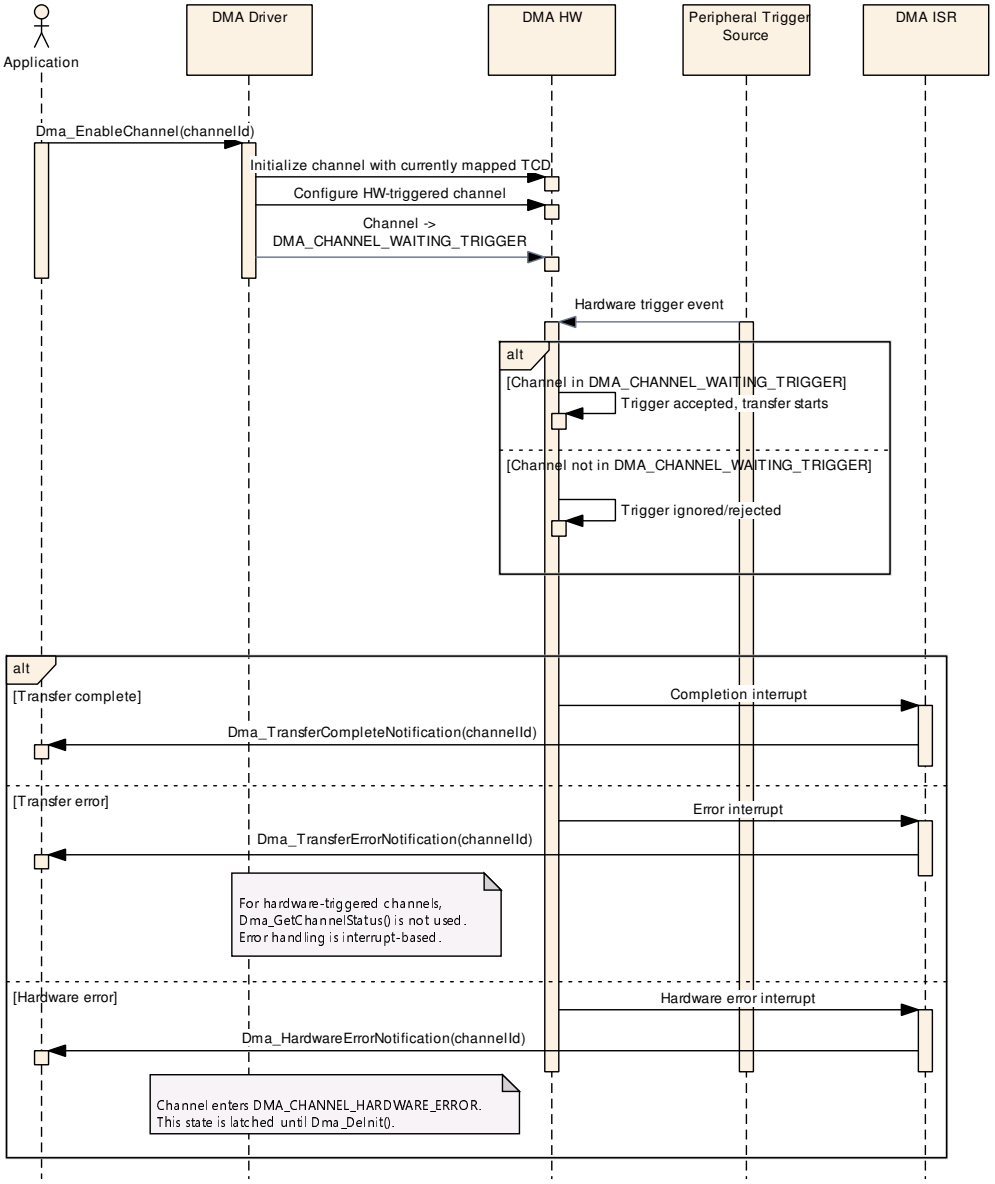


Figure 9.7: `Hardware-triggered transfer` including trigger acceptance gating

9.5 Cross-Cutting Operation Sequence Diagrams

This section groups supervision, reconfiguration, and observation-path behavior patterns.

9.5.1 Runtime Reconfiguration and Read-Modify-Write Workflow

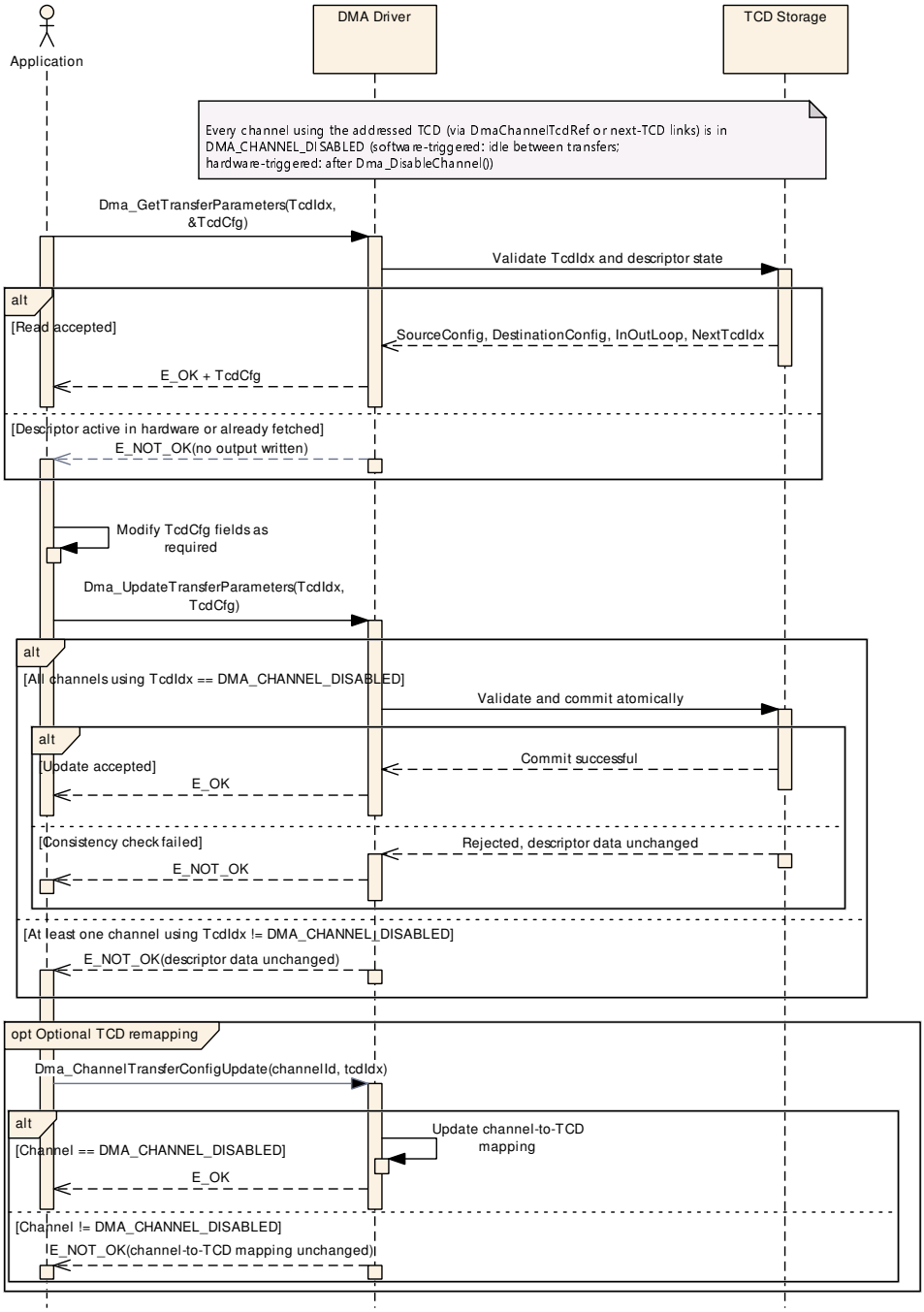


Figure 9.8: Runtime transfer reconfiguration workflow with DISABLED-state precondition, read-modify-write update, and optional TCD remapping

9.5.2 Observation Paths (Notification Channel vs Polling Channel)

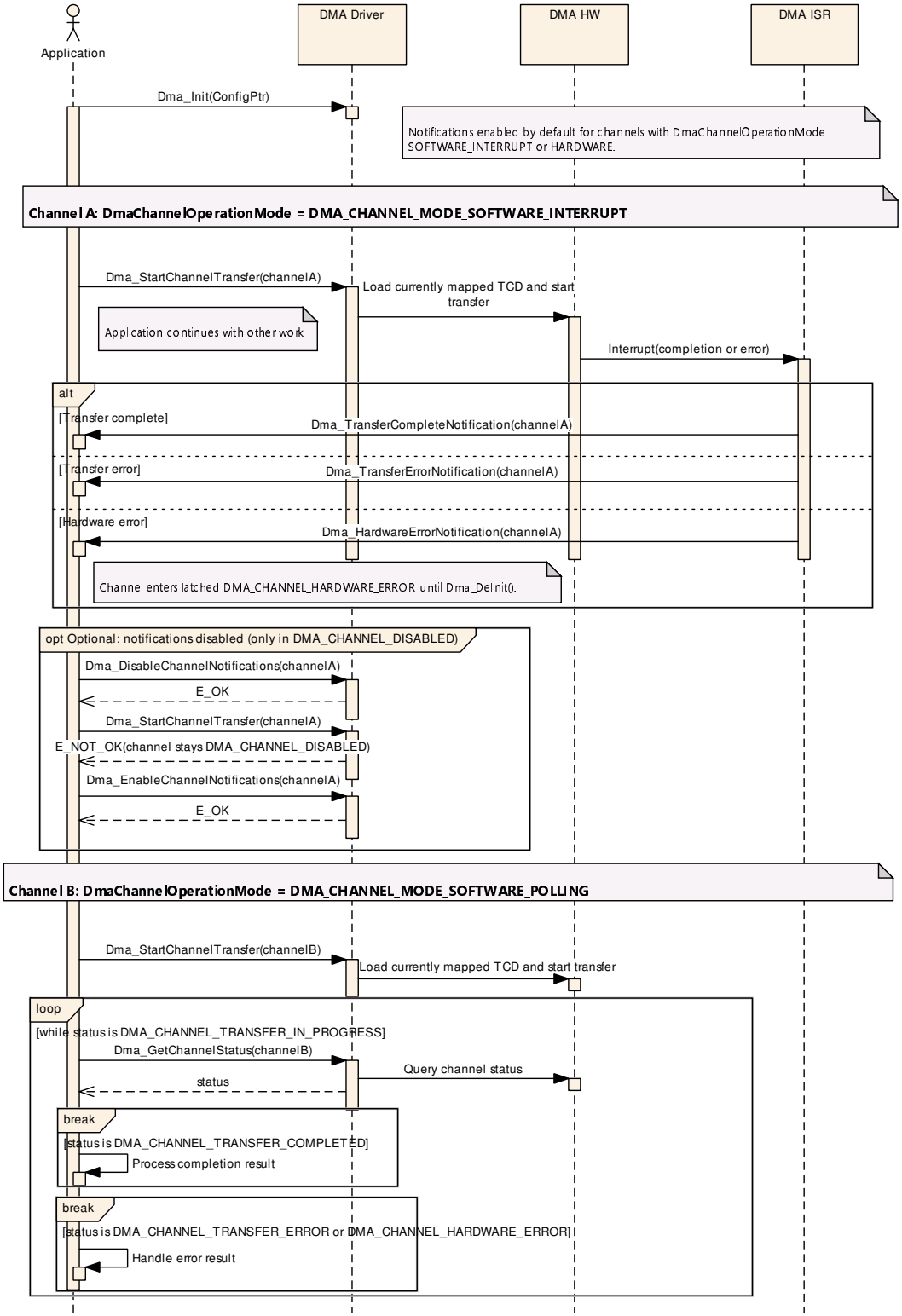


Figure 9.9: Notification-based and polling-based observation on channels with different `DmaChannelOperationMode`

The observation path is selected per channel by DmaChannelOperationMode and cannot be switched at runtime. Notifications are enabled by default after Dma_Init() ([CP_SWS_Dma_00380]). They can only be disabled while the channel is in DMA_CHANNEL_DISABLED state ([CP_SWS_Dma_00382]), and the channel cannot be started while they are disabled ([CP_SWS_Dma_00383]).

10 Configuration specification

In general, this chapter defines configuration parameters and their clustering into containers. In order to support the specification Chapter 10.1 describes fundamentals. It also specifies a template (table) that is used for the parameter specification. We intend to leave Chapter 10.1 in the specification to guarantee comprehension.

Chapter 10.2 specifies the structure (containers) and the parameters of the module Dma.

Chapter 10.4 specifies published information of the module Dma.

10.1 How to read this chapter

For details refer to [4] Chapter 10.1 *"Introduction to configuration specification"*.

10.2 Containers and configuration parameters

The following sections summarize all configuration parameters organized according to the DMA module configuration hierarchy. The top-level module configuration groups containerized parameter sets for DMA transfer-control-descriptor setup and global behavior switches. The detailed meanings of the parameters are described in Chapter 7 and Chapter 8.

10.2.1 Dma

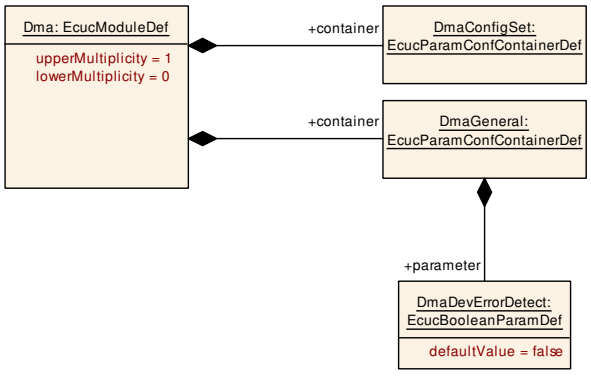


Figure 10.1: Dma configuration overview

[ECUC_Dma_00001] Definition of EcucModuleDef Dma

Status: DRAFT

⌈

Module Name	Dma
Description	Configuration of the Dma (Direct Memory Access) Driver module.
Post-Build Selectable Support	false
Post-Build Loadable Support	false

Included Containers		
Container Name	Multiplicity	Description
DmaConfigSet	1	This container contains the configuration parameters and sub containers of the AUTOSAR Dma module. **Tags:** atp.Status=draft
DmaGeneral	1	Container for global Dma module switches. **Tags:** atp.Status=draft

⌋

10.2.2 Configuration Set

This container groups the logical channel, hardware unit and transfer control descriptor configuration of the Dma module.

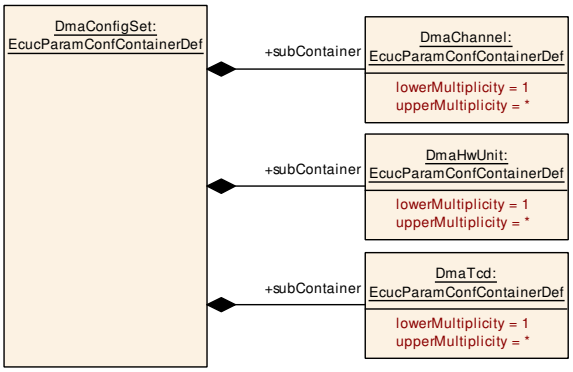


Figure 10.2: DmaConfigSet configuration overview

[ECUC_Dma_00002] Definition of EcucParamConfContainerDef DmaConfigSet

Status: DRAFT

⌈

Container Name	DmaConfigSet
Parent Container	Dma
Upstream mapping defined	–
Description	This container contains the configuration parameters and sub containers of the AUTOSAR Dma module. Its content is applied by Dma_Init(); only the pre-compile configuration variant is supported. **Tags:** atp.Status=draft
Multiplicity	1

▽

△

Pre-Compile only	true
Configuration Parameters	

No Included Parameters

Included Containers		
Container Name	**Multiplicity**	**Description**
DmaChannel	1..*	Container holding the DMA logical channel configuration. **Tags:** atp.Status=draft
DmaHwUnit	1..*	Container holding the DMA hardware unit configuration. **Tags:** atp.Status=draft
DmaTcd	1..*	Container holding the DMA Transfer Control Descriptor configuration. **Tags:** atp.Status=draft

⌋

10.2.2.1 Channel

This container specifies logical-channel properties, hardware mapping and callbacks for DMA transfer operation.

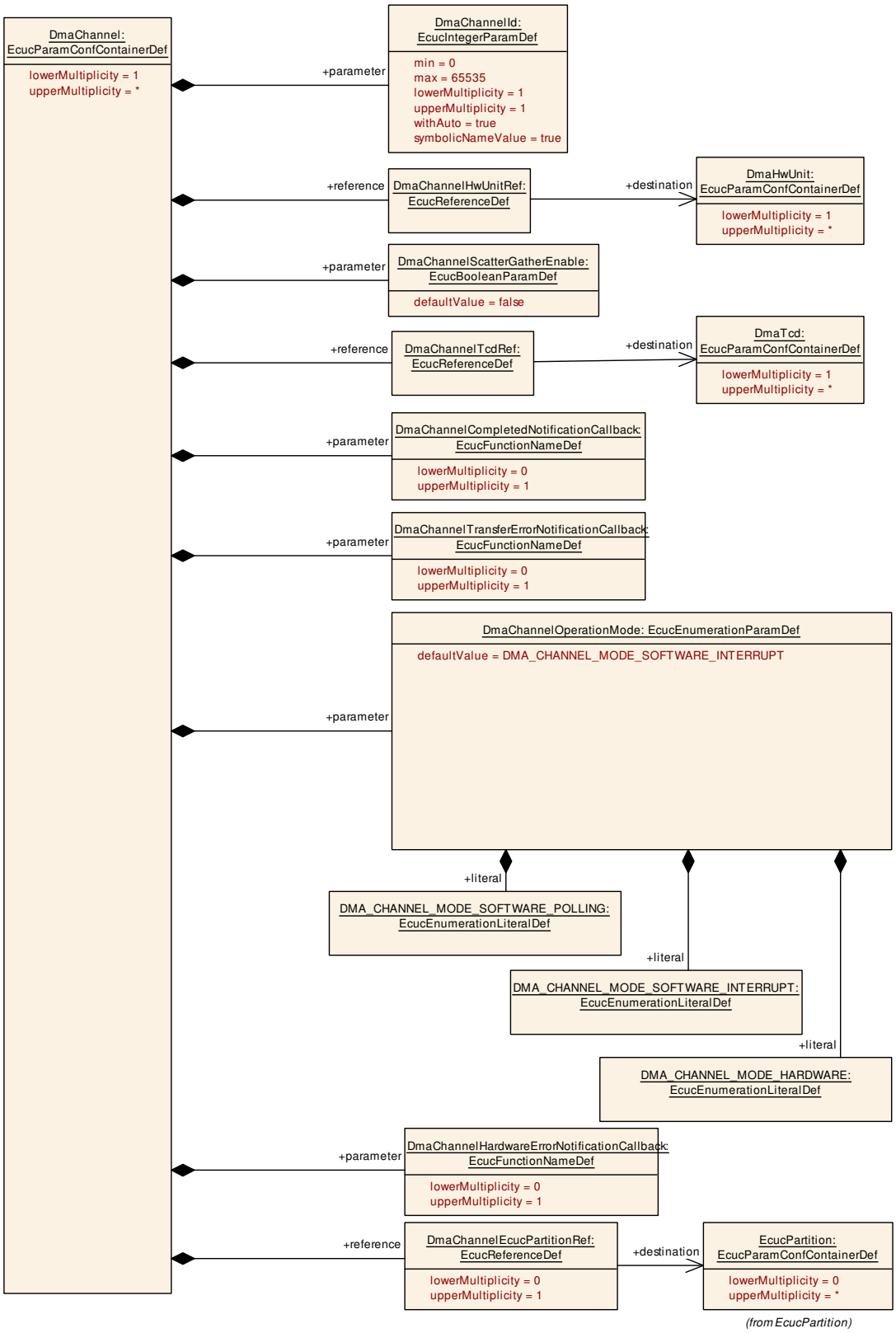


Figure 10.3: DmaChannel configuration overview (part I): parameters and references

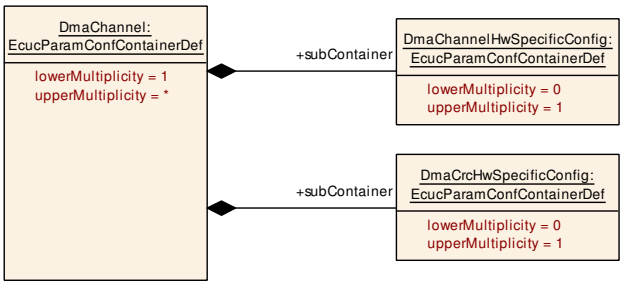


Figure 10.4: DmaChannel configuration overview (part II): sub containers

[ECUC_Dma_00005] Definition of EcucParamConfContainerDef DmaChannel

Status: DRAFT

⌈

Container Name	DmaChannel
Parent Container	DmaConfigSet
Upstream mapping defined	–
Description	Container holding the DMA logical channel configuration. **Tags:** atp.Status=draft
Multiplicity	1..*
Pre-Compile only	true
Configuration Parameters	

Included Parameters		
Parameter Name	**Multiplicity**	**ECUC ID**
DmaChannelCompletedNotificationCallback	0..1	[ECUC_Dma_00012]
DmaChannelHardwareErrorNotificationCallback	0..1	[ECUC_Dma_00015]
DmaChannelId	1	[ECUC_Dma_00008]
DmaChannelOperationMode	1	[ECUC_Dma_00014]
DmaChannelScatterGatherEnable	1	[ECUC_Dma_00010]
DmaChannelTransferErrorNotificationCallback	0..1	[ECUC_Dma_00013]
DmaChannelEcucPartitionRef	0..1	[ECUC_Dma_00016]
DmaChannelHwUnitRef	1	[ECUC_Dma_00009]
DmaChannelTcdRef	1	[ECUC_Dma_00011]

Included Containers		
Container Name	**Multiplicity**	**Description**
DmaChannelHwSpecificConfig	0..1	Extension point for hardware-specific parameters. Examples: selection of the physical channel within the hardware unit, HW MUX/request source setup, transfer priority, preemption enable/ disable, full/half transfer interrupt enable, and bandwidth control. **Tags:** atp.Status=draft
DmaCrcHwSpecificConfig	0..1	Container for the hardware-specific CRC configuration of the channel. CRC support is only configured for a channel if this container is present; how the CRC calculation is configured in detail is implementation-specific. Examples: CRC hardware channel selection, polynomial selection, initial CRC seed value, and CRC result register mapping. **Tags:** atp.Status=draft

⌋

[ECUC_Dma_00012] Definition of EcucFunctionNameDef DmaChannelCompletedNotificationCallback

Status: DRAFT

⌈

Parameter Name	DmaChannelCompletedNotificationCallback
Parent Container	DmaChannel
Upstream mapping defined	–
Description	This callback is invoked when the configured channel transfer completes. **Tags:** atp.Status=draft
Multiplicity	0..1
Type	EcucFunctionNameDef
Default value	–
Regular Expression	–
Pre-Compile only	true

⌋

[ECUC_Dma_00015] Definition of EcucFunctionNameDef DmaChannelHardwareErrorNotificationCallback

Status: DRAFT

⌈

Parameter Name	DmaChannelHardwareErrorNotificationCallback
Parent Container	DmaChannel
Upstream mapping defined	–
Description	This callback is invoked when the DMA hardware reports a hardware error for the channel. **Tags:** atp.Status=draft
Multiplicity	0..1
Type	EcucFunctionNameDef
Default value	–
Regular Expression	–
Pre-Compile only	true

⌋

[ECUC_Dma_00008] Definition of EcucIntegerParamDef DmaChannelId

Status: DRAFT

⌈

Parameter Name	DmaChannelId
Parent Container	DmaChannel
Upstream mapping defined	–

▽

△

Description	This parameter identifies the logical DMA channel used by upper-layer software. **Tags:** atp.Status=draft	
Multiplicity	1	
Type	EcucIntegerParamDef (Symbolic Name generated for this parameter)	
Range	0 .. 65535	
Default value	–	
With Auto	true	
Pre-Compile only	true	

⌋

[ECUC_Dma_00014] Definition of EcucEnumerationParamDef DmaChannelOperationMode

Status: DRAFT

⌈

Parameter Name	DmaChannelOperationMode	
Parent Container	DmaChannel	
Upstream mapping defined	–	
Description	Selects whether the channel is software-triggered (via Dma_StartChannelTransfer()) with polling-based or interrupt-based completion/error observation, or hardware-triggered by a configured hardware event with interrupt-based (notification-only) observation. This value determines which channel-level state machine variant and which API restrictions apply (see [CP_SWS_Dma_00334], [CP_SWS_Dma_00371], [CP_SWS_Dma_00372], [CP_SWS_Dma_00373], [CP_SWS_Dma_00374], [CP_SWS_Dma_00375], [CP_SWS_Dma_00376]). **Tags:** atp.Status=draft Dependency: See [CP_SWS_Dma_CONSTR_00003] and [CP_SWS_Dma_CONSTR_00004] for the mandatory and prohibited configuration of DmaChannelCompletedNotificationCallback, DmaChannelTransferErrorNotificationCallback, and DmaChannelHardwareErrorNotificationCallback depending on this value.	
Multiplicity	1	
Type	EcucEnumerationParamDef	
Range	DMA_CHANNEL_MODE_HARDWARE	Hardware-triggered channel; the transfer is started by a configured hardware event. Only interrupt-driven notification is supported; Dma_StartChannelTransfer(), Dma_StopChannelTransfer(), Dma_GetChannelStatus(), Dma_SuspendChannelTransfer(), and Dma_ResumeChannelTransfer() are rejected for this mode. **Tags:** atp.Status=draft
	DMA_CHANNEL_MODE_SOFTWARE_INTERRUPT	Software-triggered channel; completion and error are observed via interrupt-driven notification callbacks. Dma_EnableChannel() and Dma_DisableChannel() are rejected for this mode **Tags:** atp.Status=draft
	DMA_CHANNEL_MODE_SOFTWARE_POLLING	Software-triggered channel; completion and error are observed via Dma_GetChannelStatus() polling. Notification callbacks are not used. Dma_EnableChannel() and Dma_DisableChannel() are rejected for this mode. **Tags:** atp.Status=draft

▽

△

Default value	DMA_CHANNEL_MODE_SOFTWARE_INTERRUPT
Pre-Compile only	true

⌋

[ECUC_Dma_00010] Definition of EcucBooleanParamDef DmaChannelScatterGatherEnable

Status: DRAFT

⌈

Parameter Name	DmaChannelScatterGatherEnable
Parent Container	DmaChannel
Upstream mapping defined	–
Description	This switch enables or disables Scatter-gather operation for the configured channel. **Tags:** atp.Status=draft
Multiplicity	1
Type	EcucBooleanParamDef
Default value	false
Pre-Compile only	true

⌋

[ECUC_Dma_00013] Definition of EcucFunctionNameDef DmaChannelTransferErrorNotificationCallback

Status: DRAFT

⌈

Parameter Name	DmaChannelTransferErrorNotificationCallback
Parent Container	DmaChannel
Upstream mapping defined	–
Description	This callback is invoked when the DMA hardware reports a transfer error for the channel. **Tags:** atp.Status=draft
Multiplicity	0..1
Type	EcucFunctionNameDef
Default value	–
Regular Expression	–
Pre-Compile only	true

⌋

[ECUC_Dma_00016] Definition of EcucReferenceDef DmaChannelEcucPartitionRef

Status: DRAFT

⌈

Parameter Name	DmaChannelEcucPartitionRef
Parent Container	DmaChannel
Upstream mapping defined	–
Description	This reference assigns the channel configuration to the corresponding ECUC partition. **Tags:** atp.Status=draft Dependency: See [SWS_Dma_CONSTR_00007]: every DmaTcd reachable from this channel shall be assigned to the same partition.
Multiplicity	0..1
Type	Reference to EcucPartition
Pre-Compile only	true

⌋

[ECUC_Dma_00009] Definition of EcucReferenceDef DmaChannelHwUnitRef

Status: DRAFT

⌈

Parameter Name	DmaChannelHwUnitRef
Parent Container	DmaChannel
Upstream mapping defined	–
Description	This reference links the logical DMA channel to the DMA hardware unit (DMA controller instance) that executes its transfers. It allows the same channel configuration model to be mapped to different DMA hardware units on the target microcontroller. The selection of the physical channel within the hardware unit is implementation-specific (see DmaChannelHwSpecificConfig). **Tags:** atp.Status=draft
Multiplicity	1
Type	Reference to DmaHwUnit
Pre-Compile only	true

⌋

[ECUC_Dma_00011] Definition of EcucReferenceDef DmaChannelTcdRef

Status: DRAFT

⌈

Parameter Name	DmaChannelTcdRef
Parent Container	DmaChannel
Upstream mapping defined	–
Description	This reference identifies the transfer control descriptor configuration assigned to this channel. **Tags:** atp.Status=draft
Multiplicity	1
Type	Reference to DmaTcd

▽

△

Pre-Compile only	true

⌋

10.2.2.2 Transfer Control Descriptor

This container holds the Transfer Control Descriptor (TCD) configuration and associated transfer-side and hardware-specific settings.

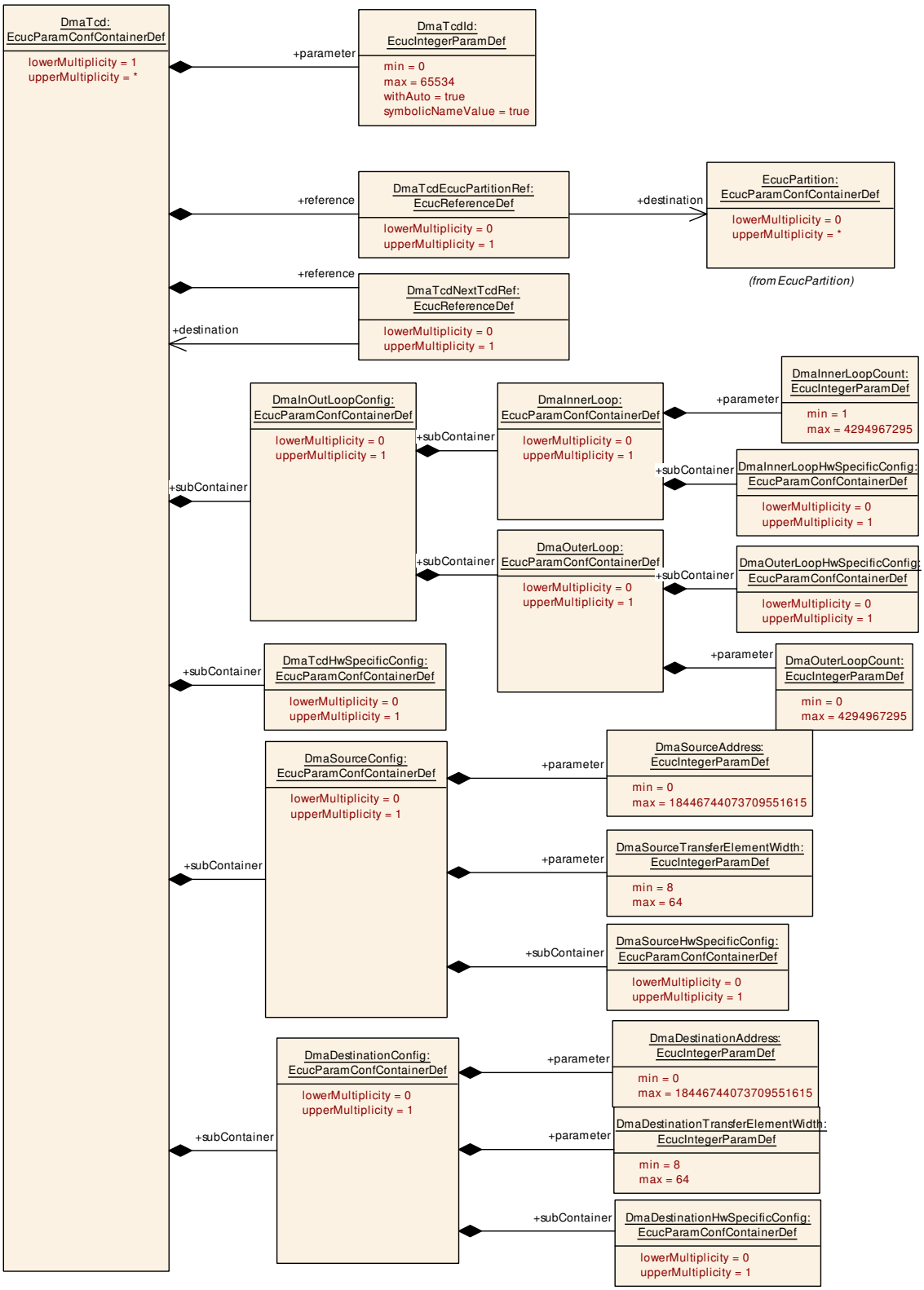


Figure 10.5: DmaTcd configuration overview

[ECUC_Dma_00007] Definition of EcucParamConfContainerDef DmaTcd

Status: DRAFT

⌈

Container Name	DmaTcd
Parent Container	DmaConfigSet
Upstream mapping defined	–
Description	Container holding the DMA Transfer Control Descriptor configuration. **Tags:** atp.Status=draft
Multiplicity	1..*
Pre-Compile only	true
Configuration Parameters	

Included Parameters		
Parameter Name	**Multiplicity**	**ECUC ID**
DmaTcdId	1	[ECUC_Dma_00020]
DmaTcdEcucPartitionRef	0..1	[ECUC_Dma_00021]
DmaTcdNextTcdRef	0..1	[ECUC_Dma_00022]

Included Containers		
Container Name	**Multiplicity**	**Description**
DmaDestinationConfig	0..1	Container holding the common destination-side parameters for a DMA transfer descriptor. **Tags:** atp.Status=draft
DmaInOutLoopConfig	0..1	Container for shared loop control settings for source and destination addressing behavior. **Tags:** atp.Status=draft
DmaSourceConfig	0..1	Container holding the common source-side parameters for a DMA transfer descriptor. **Tags:** atp.Status=draft
DmaTcdHwSpecificConfig	0..1	Extension point for hardware-specific parameters. Examples: minor-loop mapping controls, descriptor interpretation flags, arbitration hints, and controller-specific transfer control options. **Tags:** atp.Status=draft

⌋

[ECUC_Dma_00020] Definition of EcucIntegerParamDef DmaTcdId

Status: DRAFT

⌈

Parameter Name	DmaTcdId
Parent Container	DmaTcd
Upstream mapping defined	–
Description	Logical index of the TCD data block. The value is used as TcdIdx and NextTcdIdx at the API. The value 65535 (0xFFFF) is reserved for DMA_TCD_IDX_NONE. **Tags:** atp.Status=draft
Multiplicity	1

▽

△

Type	EcucIntegerParamDef (Symbolic Name generated for this parameter)	
Range	0 .. 65534	
Default value	–	
With Auto	true	
Pre-Compile only	true	

⌋

[ECUC_Dma_00021] Definition of EcucReferenceDef DmaTcdEcucPartitionRef

Status: DRAFT

⌈

Parameter Name	DmaTcdEcucPartitionRef
Parent Container	DmaTcd
Upstream mapping defined	–
Description	Reference to the ECUC partition assigned to this TCD. **Tags:** atp.Status=draft Dependency: See [CP_SWS_Dma_CONSTR_00007]: shall reference the same partition as every DmaChannel from which this DmaTcd is reachable.
Multiplicity	0..1
Type	Reference to EcucPartition
Pre-Compile only	true

⌋

[ECUC_Dma_00022] Definition of EcucReferenceDef DmaTcdNextTcdRef

Status: DRAFT

⌈

Parameter Name	DmaTcdNextTcdRef
Parent Container	DmaTcd
Upstream mapping defined	–
Description	Reference to the next TCD used in Scatter-gather configuration. **Tags:** atp.Status=draft Dependency: See [CP_SWS_Dma_CONSTR_00006]: this reference is only applicable if DmaChannelScatterGatherEnable is enabled for the owning channel.
Multiplicity	0..1
Type	Reference to DmaTcd
Pre-Compile only	true

⌋

Note

`DmaSourceConfig`, `DmaDestinationConfig`, `DmaInOutLoopConfig`, `DmaInnerLoop` and `DmaOuterLoop` are optional because their parameters can also be provided at runtime via `Dma_UpdateTransferParameters`(). A channel whose

DmaTcd is not complete according to [CP_SWS_Dma_00385] can neither be started nor enabled (see [CP_SWS_Dma_00386] and [CP_SWS_Dma_00387]).

Loop Configuration

This container defines shared inner-loop and outer-loop settings that are applied jointly to source and destination sides.

[ECUC_Dma_00023] Definition of EcucParamConfContainerDef DmaInOutLoop Config

Status: DRAFT

⌈

Container Name	DmaInOutLoopConfig
Parent Container	DmaTcd
Upstream mapping defined	–
Description	Container for shared loop control settings for source and destination addressing behavior. **Tags:** atp.Status=draft
Multiplicity	0..1
Pre-Compile only	true
Configuration Parameters	

No Included Parameters

Included Containers		
Container Name	**Multiplicity**	**Description**
DmaInnerLoop	0..1	Inner loop configuration for repeated element-level iterations shared by source and destination sides. **Tags:** atp.Status=draft
DmaOuterLoop	0..1	Outer loop configuration for repeated inner-loop passes shared by source and destination sides. **Tags:** atp.Status=draft

⌋

Inner Loop

This container defines inner-loop execution settings.

[ECUC_Dma_00024] Definition of EcucParamConfContainerDef DmaInnerLoop

Status: DRAFT

⌈

Container Name	DmaInnerLoop
Parent Container	DmaInOutLoopConfig
Upstream mapping defined	–
Description	Container for inner-loop settings shared by source and destination sides. **Tags:** atp.Status=draft
Multiplicity	0..1
Pre-Compile only	true
Configuration Parameters	

Included Parameters		
Parameter Name	**Multiplicity**	**ECUC ID**
DmaInnerLoopCount	1	[ECUC_Dma_00025]

Included Containers		
Container Name	**Multiplicity**	**Description**
DmaInnerLoopHwSpecificConfig	0..1	Extension point for hardware-specific parameters. Examples: per-side circular buffer mode, modulo window size, and address wrap behavior. **Tags:** atp.Status=draft

⌋

[ECUC_Dma_00025] Definition of EcucIntegerParamDef DmaInnerLoopCount

Status: DRAFT

⌈

Parameter Name	DmaInnerLoopCount	
Parent Container	DmaInnerLoop	
Upstream mapping defined	–	
Description	Number of source-side Transfer elements read per inner-loop pass, each of size DmaSourceTransferElementWidth. If the destination-side element width (DmaDestinationTransferElementWidth) differs, the DMA controller packs or unpacks the data accordingly, so the number of destination-side elements written per pass differs from this count; see [CP_SWS_Dma_CONSTR_00005] for the applicable width ratio and [CP_SWS_Dma_00028] for the resulting execution semantics. **Tags:** atp.Status=draft	
Multiplicity	1	
Type	EcucIntegerParamDef	
Range	1 .. 4294967295	
Default value	–	
Pre-Compile only	true	

⌋

Outer Loop

This container defines outer-loop execution settings.

[ECUC_Dma_00027] Definition of EcucParamConfContainerDef DmaOuterLoop

Status: DRAFT

⌈

Container Name	DmaOuterLoop
Parent Container	DmaInOutLoopConfig
Upstream mapping defined	–
Description	Container for outer-loop settings shared by source and destination sides. **Tags:** atp.Status=draft
Multiplicity	0..1
Pre-Compile only	true
Configuration Parameters	

Included Parameters		
Parameter Name	**Multiplicity**	**ECUC ID**
DmaOuterLoopCount	1	[ECUC_Dma_00029]

Included Containers		
Container Name	**Multiplicity**	**Description**
DmaOuterLoopHwSpecificConfig	0..1	Extension point for hardware-specific parameters. Examples: outer-loop reload control, shadow-register swap trigger, and end-of-loop rearm behavior. **Tags:** atp.Status=draft

⌋

[ECUC_Dma_00029] Definition of EcucIntegerParamDef DmaOuterLoopCount

Status: DRAFT

⌈

Parameter Name	DmaOuterLoopCount	
Parent Container	DmaOuterLoop	
Upstream mapping defined	–	
Description	Number of times the inner loop (DmaInnerLoopCount source-side Transfer elements each) is repeated for this TCD; see [CP_SWS_Dma_00028] for the resulting execution semantics and [CP_SWS_Dma_CONSTR_00005] for how this relates to the destination side when source and destination widths differ. A value of 0 configures no outer-loop repetition, i.e., the inner loop executes exactly once. **Tags:** atp.Status=draft	
Multiplicity	1	
Type	EcucIntegerParamDef	
Range	0 .. 4294967295	
Default value	–	
Pre-Compile only	true	

⌋

Source Configuration

This container holds the common source-side TCD configuration subset.

[ECUC_Dma_00031] Definition of EcucParamConfContainerDef DmaSourceConfig

Status: DRAFT

⌈

Container Name	DmaSourceConfig
Parent Container	DmaTcd
Upstream mapping defined	–
Description	Container holding the common source-side parameters for a DMA transfer descriptor. **Tags:** atp.Status=draft
Multiplicity	0..1
Pre-Compile only	true
Configuration Parameters	

Included Parameters		
Parameter Name	**Multiplicity**	**ECUC ID**
DmaSourceAddress	1	[ECUC_Dma_00032]
DmaSourceTransferElementWidth	1	[ECUC_Dma_00033]

Included Containers		
Container Name	**Multiplicity**	**Description**
DmaSourceHwSpecificConfig	0..1	Extension point for hardware-specific parameters. Examples: source bus master selection, source handshake line mapping, source access privilege/security attributes, and source alignment constraints. **Tags:** atp.Status=draft

⌋

[ECUC_Dma_00032] Definition of EcucIntegerParamDef DmaSourceAddress

Status: DRAFT

⌈

Parameter Name	DmaSourceAddress	
Parent Container	DmaSourceConfig	
Upstream mapping defined	–	
Description	Static initial value of the source address (the memory location or peripheral register the DMA channel reads from). This value can be changed at runtime via Dma_UpdateTransferParameters(); see [CP_SWS_Dma_CONSTR_00002] for the applicable alignment constraints. **Tags:** atp.Status=draft	
Multiplicity	1	
Type	EcucIntegerParamDef	
Range	0 .. 18446744073709551615	

▽

△

Default value	–
Pre-Compile only	true

⌋

[ECUC_Dma_00033] Definition of EcucIntegerParamDef DmaSourceTransferElementWidth

Status: DRAFT

⌈

Parameter Name	DmaSourceTransferElementWidth	
Parent Container	DmaSourceConfig	
Upstream mapping defined	–	
Description	Size of the Transfer element for the data to be transferred from source (8bit, 16bit, 32bit, 64bit). Range: 8, 16, 32, 64 (bit); at least these values are supported, see [CP_SWS_Dma_CONSTR_00001] for the mandatory minimum. Hardware-specific extensions to this set are allowed. **Tags:** atp.Status=draft	
Multiplicity	1	
Type	EcucIntegerParamDef	
Range	8 .. 64	
Default value	–	
Pre-Compile only	true	

⌋

Note

The value of `DmaSourceTransferElementWidth` is the transfer element width in bits (for example 8, 16, 32, or 64).

Destination Configuration

This container holds the common destination-side `TCD` configuration subset.

[ECUC_Dma_00035] Definition of EcucParamConfContainerDef DmaDestinationConfig

Status: DRAFT

⌈

Container Name	DmaDestinationConfig
Parent Container	DmaTcd
Upstream mapping defined	–
Description	Container holding the common destination-side parameters for a DMA transfer descriptor. **Tags:** atp.Status=draft
Multiplicity	0..1
Pre-Compile only	true
Configuration Parameters	

Included Parameters		
Parameter Name	**Multiplicity**	**ECUC ID**
DmaDestinationAddress	1	[ECUC_Dma_00036]
DmaDestinationTransferElementWidth	1	[ECUC_Dma_00037]

Included Containers		
Container Name	**Multiplicity**	**Description**
DmaDestinationHwSpecificConfig	0..1	Extension point for hardware-specific parameters. Examples: destination bus master selection, destination handshake behavior, destination access privilege/security attributes, and write alignment constraints. **Tags:** atp.Status=draft

⌋

[ECUC_Dma_00036] Definition of EcucIntegerParamDef DmaDestinationAddress

Status: DRAFT

⌈

Parameter Name	DmaDestinationAddress	
Parent Container	DmaDestinationConfig	
Upstream mapping defined	–	
Description	Static initial value of the destination address (the memory location or peripheral register the DMA channel writes to). This value can be changed at runtime via Dma_Update TransferParameters(); see [CP_SWS_Dma_CONSTR_00002] for the applicable alignment constraints. **Tags:** atp.Status=draft	
Multiplicity	1	
Type	EcucIntegerParamDef	
Range	0 .. 18446744073709551615	
Default value	–	
Pre-Compile only	true	

⌋

[ECUC_Dma_00037] Definition of EcucIntegerParamDef DmaDestinationTransferElementWidth

Status: DRAFT

⌈

Parameter Name	DmaDestinationTransferElementWidth
Parent Container	DmaDestinationConfig
Upstream mapping defined	–
Description	Size of the Transfer element for the data to be transferred to destination (8bit,16bit, 32bit, 64bit). Range: 8, 16, 32, 64 (bit); at least these values are supported, see [CP_SWS_Dma_CONSTR_00001] for the mandatory minimum. Hardware-specific extensions to this set are allowed. **Tags:** atp.Status=draft
Multiplicity	1
Type	EcucIntegerParamDef
Range	8 .. 64
Default value	–
Pre-Compile only	true

⌋

Note

The value of DmaDestinationTransferElementWidth is the transfer element width in bits (for example 8, 16, 32, or 64).

10.2.2.3 Hardware Unit

This container represents one DMA hardware unit (DMA controller instance) of the microcontroller. DMA channels are mapped to a hardware unit via DmaChannelHwUnitRef.

How a hardware unit is addressed (for example by a predefined symbolic instance such as DMA_A/DMA_B, by an instance index, or by a register base address) is implementation-specific and therefore not standardized. Such parameters are added by the implementation in DmaHwUnitHwSpecificConfig.

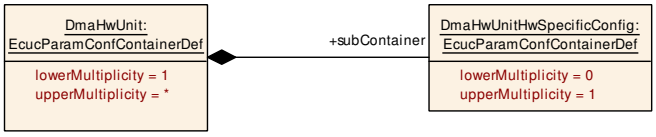


Figure 10.6: DmaHwUnit configuration overview

[ECUC_Dma_00006] Definition of EcucParamConfContainerDef DmaHwUnit

Status: DRAFT

⌈

Container Name	DmaHwUnit
Parent Container	DmaConfigSet
Upstream mapping defined	–
Description	Container holding the DMA hardware unit configuration. **Tags:** atp.Status=draft
Multiplicity	1..*
Pre-Compile only	true
Configuration Parameters	

No Included Parameters

Included Containers		
Container Name	**Multiplicity**	**Description**
DmaHwUnitHwSpecificConfig	0..1	Extension point for hardware-specific parameters. Examples: addressing of the hardware unit (symbolic instance, instance index or register base address), channel arbitration scheme (fixed priority or round-robin), debug/halt behavior, and unit-level error handling. **Tags:** atp.Status=draft

⌋

10.2.2.4 Hardware Specific Configuration

The containers in this section define extension points in the configuration hierarchy. Each of them is aggregated by exactly one parent container, so that hardware-specific additions can be defined individually for each location while the base DMA container structure and standard parameters remain unchanged. The listed examples are informative.

[ECUC_Dma_00017] Definition of EcucParamConfContainerDef DmaChannelHwSpecificConfig

Status: DRAFT

⌈

Container Name	DmaChannelHwSpecificConfig
Parent Container	DmaChannel
Upstream mapping defined	–
Description	Extension point for hardware-specific parameters. Examples: selection of the physical channel within the hardware unit, HW MUX/request source setup, transfer priority, preemption enable/disable, full/half transfer interrupt enable, and bandwidth control. **Tags:** atp.Status=draft
Multiplicity	0..1
Pre-Compile only	true
Configuration Parameters	

No Included Parameters

No Included Containers

⌋

[ECUC_Dma_00018] Definition of EcucParamConfContainerDef DmaCrcHwSpecificConfig

Status: DRAFT

⌈

Container Name	DmaCrcHwSpecificConfig
Parent Container	DmaChannel
Upstream mapping defined	–
Description	Container for the hardware-specific CRC configuration of the channel. CRC support is only configured for a channel if this container is present; how the CRC calculation is configured in detail is implementation-specific. Examples: CRC hardware channel selection, polynomial selection, initial CRC seed value, and CRC result register mapping. **Tags:** atp.Status=draft
Multiplicity	0..1
Pre-Compile only	true
Configuration Parameters	

No Included Parameters

No Included Containers

⌋

[ECUC_Dma_00030] Definition of EcucParamConfContainerDef DmaTcdHwSpecificConfig

Status: DRAFT

⌈

Container Name	DmaTcdHwSpecificConfig
Parent Container	DmaTcd
Upstream mapping defined	–
Description	Extension point for hardware-specific parameters. Examples: minor-loop mapping controls, descriptor interpretation flags, arbitration hints, and controller-specific transfer control options. **Tags:** atp.Status=draft
Multiplicity	0..1
Pre-Compile only	true
Configuration Parameters	

No Included Parameters

No Included Containers

⌋

[ECUC_Dma_00026] Definition of EcucParamConfContainerDef DmaInnerLoop HwSpecificConfig

Status: DRAFT

⌈

Container Name	DmaInnerLoopHwSpecificConfig
Parent Container	DmaInnerLoop
Upstream mapping defined	–
Description	Extension point for hardware-specific parameters. Examples: per-side circular buffer mode, modulo window size, and address wrap behavior. **Tags:** atp.Status=draft
Multiplicity	0..1
Pre-Compile only	true
Configuration Parameters	

No Included Parameters

No Included Containers

⌋

[ECUC_Dma_00028] Definition of EcucParamConfContainerDef DmaOuterLoop HwSpecificConfig

Status: DRAFT

⌈

Container Name	DmaOuterLoopHwSpecificConfig
Parent Container	DmaOuterLoop
Upstream mapping defined	–
Description	Extension point for hardware-specific parameters. Examples: outer-loop reload control, shadow-register swap trigger, and end-of-loop rearm behavior. **Tags:** atp.Status=draft
Multiplicity	0..1
Pre-Compile only	true
Configuration Parameters	

No Included Parameters

No Included Containers

⌋

[ECUC_Dma_00034] Definition of EcucParamConfContainerDef DmaSourceHw SpecificConfig

Status: DRAFT

⌈

Container Name	DmaSourceHwSpecificConfig
Parent Container	DmaSourceConfig
Upstream mapping defined	–
Description	Extension point for hardware-specific parameters. Examples: source bus master selection, source handshake line mapping, source access privilege/security attributes, and source alignment constraints. **Tags:** atp.Status=draft
Multiplicity	0..1
Pre-Compile only	true
Configuration Parameters	

No Included Parameters

No Included Containers

⌋

[ECUC_Dma_00038] Definition of EcucParamConfContainerDef DmaDestination HwSpecificConfig

Status: DRAFT

⌈

Container Name	DmaDestinationHwSpecificConfig
Parent Container	DmaDestinationConfig
Upstream mapping defined	–
Description	Extension point for hardware-specific parameters. Examples: destination bus master selection, destination handshake behavior, destination access privilege/security attributes, and write alignment constraints. **Tags:** atp.Status=draft
Multiplicity	0..1
Pre-Compile only	true
Configuration Parameters	

No Included Parameters

No Included Containers

⌋

[ECUC_Dma_00019] Definition of EcucParamConfContainerDef DmaHwUnitHw SpecificConfig

Status: DRAFT

⌈

Container Name	DmaHwUnitHwSpecificConfig
Parent Container	DmaHwUnit
Upstream mapping defined	–
Description	Extension point for hardware-specific parameters. Examples: addressing of the hardware unit (symbolic instance, instance index or register base address), channel arbitration scheme (fixed priority or round-robin), debug/halt behavior, and unit-level error handling. **Tags:** atp.Status=draft
Multiplicity	0..1
Pre-Compile only	true
Configuration Parameters	

No Included Parameters

No Included Containers

⌋

10.2.3 General

This container groups global Dma module switches, including the development error detection enable switch.

[ECUC_Dma_00003] Definition of EcucParamConfContainerDef DmaGeneral

Status: DRAFT

⌈

Container Name	DmaGeneral
Parent Container	Dma
Upstream mapping defined	–
Description	Container for global Dma module switches. **Tags:** atp.Status=draft
Multiplicity	1
Pre-Compile only	true
Configuration Parameters	

Included Parameters		
Parameter Name	**Multiplicity**	**ECUC ID**
DmaDevErrorDetect	1	[ECUC_Dma_00004]

No Included Containers

⌋

[ECUC_Dma_00004] Definition of EcucBooleanParamDef DmaDevErrorDetect

Status: DRAFT

⌈

Parameter Name	DmaDevErrorDetect
Parent Container	DmaGeneral
Upstream mapping defined	–
Description	Switches the development error detection and notification on or off. True: detection enabled. False: detection disabled. **Tags:** atp.Status=draft
Multiplicity	1
Type	EcucBooleanParamDef
Default value	false
Pre-Compile only	true

⌋

10.3 Semantic Constraints

[CP_SWS_Dma_CONSTR_00001] Transfer Element Width Allowed Configuration Values

Status: DRAFT

⌈For `DmaSourceTransferElementWidth` and `DmaDestinationTransferElementWidth`, at least the values 8, 16, 32, and 64 (bit) shall be supported and configurable. This value set is the minimum mandatory transfer-element-width set for all implementations. Hardware-specific extensions are allowed and may extend this set with additional widths in the module implementation's own ECU Configuration Definition (BSWMD).⌋

Note

For a given Basic Software module, exactly one concrete ECU Configuration Definition (BSWMD, an `EcucModuleDef`) is delivered per implementation, typically by the microcontroller or Basic Software vendor. That BSWMD contains at least the containers and parameters standardized by this specification, but the implementer may widen configuration ranges (as applies here to `DmaSourceTransferElementWidth` and `DmaDestinationTransferElementWidth`) or add further implementation-specific containers and parameters beyond the standardized set within that same BSWMD. This extension mechanism is defined by the AUTOSAR ECU Configuration meta-model, described in the ECU Configuration and BSW Module Description Template specifications, not by this document.

[CP_SWS_Dma_CONSTR_00002] Configured Address Alignment Constraint

Status: DRAFT

⌈For `DmaSourceAddress` and `DmaDestinationAddress` statically configured in the ECU configuration values, only correctly aligned addresses shall be used according to the configured transfer-element widths and target-hardware specific DMA alignment constraints. Runtime-provided addresses (for example via `Dma_UpdateTransferParameters`()) are specified by [CP_SWS_Dma_00368].⌋

[CP_SWS_Dma_CONSTR_00003] Channel Operation Mode Interrupt Notification Requirement

Status: DRAFT

⌈For a `DmaChannel` whose `DmaChannelOperationMode` is

- `DMA_CHANNEL_MODE_SOFTWARE_INTERRUPT` or
- `DMA_CHANNEL_MODE_HARDWARE`,

all of the following parameters shall be configured:

- `DmaChannelCompletedNotificationCallback`
- `DmaChannelTransferErrorNotificationCallback`
- `DmaChannelHardwareErrorNotificationCallback`

⌋

Rationale: For these modes, the notification callbacks are the only way to observe the transfer result:

- `DMA_CHANNEL_MODE_SOFTWARE_INTERRUPT`: The transfer is triggered by software, and all outcomes (transfer completion, transfer error, and hardware error) are reported via the notification callbacks. `Dma_GetChannelStatus`() only indicates whether a transfer is ongoing; after transfer completion or a transfer error, the channel returns to `DMA_CHANNEL_DISABLED` (see Section 7.2.3.1 and Figure 7.8).
- `DMA_CHANNEL_MODE_HARDWARE`: `Dma_GetChannelStatus`() is rejected for this mode ([CP_SWS_Dma_00334]).

[CP_SWS_Dma_CONSTR_00004] Channel Operation Mode Polling Notification Prohibition

Status: DRAFT

⌈For a `DmaChannel` whose `DmaChannelOperationMode` is

- `DMA_CHANNEL_MODE_SOFTWARE_POLLING`,

none of the following parameters shall be configured:

- `DmaChannelCompletedNotificationCallback`

- DmaChannelTransferErrorNotificationCallback
- DmaChannelHardwareErrorNotificationCallback

Rationale: These callbacks are not invoked in polling mode ([CP_SWS_Dma_00312]).⌋

[CP_SWS_Dma_CONSTR_00005] Transfer Element Width Ratio Constraint

Status: DRAFT

⌈If DmaSourceTransferElementWidth and DmaDestinationTransferElementWidth differ for a DmaTcd, the larger of the two widths shall be an integer multiple of the smaller width (for example, a 32bit source with a 16bit destination, or an 8bit source with a 32bit destination). DmaInnerLoopCount counts Transfer elements on the source side; the DMA controller packs or unpacks data on the destination side accordingly, so the number of destination-side elements written per inner-loop pass equals DmaInnerLoopCount multiplied by the ratio of source to destination width. See [CP_SWS_Dma_00314] for the corresponding runtime validation.⌋

Figure 7.5 shows the basic transfer types with different source and destination widths.

[CP_SWS_Dma_CONSTR_00006] Next Tcd Reference Scatter-Gather Enablement Constraint

Status: DRAFT

⌈DmaTcdNextTcdRef shall only be configured for a DmaTcd that is part of the Scatter-gather descriptor chain of a DmaChannel (reached via DmaChannelTcdRef and, transitively, via DmaTcdNextTcdRef chaining) whose DmaChannelScatterGatherEnable is set to true. See [CP_SWS_Dma_00370] for the corresponding runtime validation performed by Dma_UpdateTransferParameters().⌋

[CP_SWS_Dma_CONSTR_00007] Tcd Partition Consistency Constraint

Status: DRAFT

⌈Every DmaTcd that is reachable from a DmaChannel via DmaChannelTcdRef and, transitively, via DmaTcdNextTcdRef shall be assigned to the same partition as this DmaChannel, i.e.:

- If DmaChannelEcucPartitionRef is configured, DmaTcdEcucPartitionRef shall be configured and shall reference the same EcucPartition.
- If DmaChannelEcucPartitionRef is not configured, DmaTcdEcucPartitionRef shall not be configured either.

As a consequence, a DmaTcd shall not be shared between DmaChannels that are assigned to different partitions.⌋

Note

In this configuration specification, notations such as 8bit, 16bit, 32bit, and 64bit denote a transfer element width of 8, 16, 32, and 64 bits, i.e. the configuration values 8, 16, 32, and 64 of `DmaSourceTransferElementWidth` and `DmaDestinationTransferElementWidth`.

10.4 Published Information

For details refer to [4] Chapter 10.3 *"Published Information"*.

A Not applicable requirements

[CP_SWS_Dma_NA_00999] Requirements fulfilled by the General Specification of Basic Software Modules

Upstream requirements: SRS_BSW_00159, SRS_BSW_00167, SRS_BSW_00171, SRS_BSW_00339, SRS_BSW_00344, SRS_BSW_00345, SRS_BSW_00380, SRS_BSW_00396, SRS_BSW_00397, SRS_BSW_00398, SRS_BSW_00399, SRS_BSW_00400, SRS_BSW_00402, SRS_BSW_00404, SRS_BSW_00409, SRS_BSW_00423, SRS_BSW_00426, SRS_BSW_00427, SRS_BSW_00428, SRS_BSW_00429, SRS_BSW_00433, SRS_BSW_00437, SRS_BSW_00451, SRS_BSW_00458, SRS_BSW_00467, SRS_BSW_00470, SRS_BSW_00488, SRS_BSW_00489, SRS_BSW_00493, SRS_BSW_00496, SRS_BSW_00497

⌈These requirements are fulfilled by the General Specification of Basic Software Modules [4], which is a required specification for the DMA Driver (see Section 3.2). They are not specified again in this document.⌋

[CP_SWS_Dma_NA_00991] No production errors

Upstream requirements: SRS_BSW_00417, SRS_BSW_00422, SRS_BSW_00469, SRS_BSW_00471, SRS_BSW_00472

⌈The DMA Driver does not define production errors (see Section 7.3.3). Therefore these requirements are not applicable.⌋

[CP_SWS_Dma_NA_00992] No security events

Upstream requirements: SRS_BSW_00490, SRS_BSW_00491

⌈The DMA Driver does not define security events. Therefore these requirements are not applicable.⌋

[CP_SWS_Dma_NA_00993] No wake-up capability

Upstream requirements: SRS_BSW_00375

⌈The DMA Driver does not provide wake-up sources and does not handle wake-up events. Therefore this requirement is not applicable.⌋

[CP_SWS_Dma_NA_00995] Content of the specification

Upstream requirements: SRS_BSW_00383, SRS_BSW_00384, SRS_BSW_00386, SRS_BSW_00388, SRS_BSW_00389, SRS_BSW_00390, SRS_BSW_00392, SRS_BSW_00393, SRS_BSW_00395, SRS_BSW_00403

⌈These requirements address the content of a BSW module specification. They are fulfilled by the structure of this document, namely the configuration specification, the expected interfaces and the error classification, and are not specified by individual specification items.⌋

[CP_SWS_Dma_NA_00996] ECU integration and implementation

Upstream requirements: SRS_BSW_00416, SRS_BSW_00419

⌈These requirements address the integration of the ECU (initialization sequence) or the implementation of the module (placement of configuration data). They are not specified by individual specification items of the DMA Driver.⌋

[CP_SWS_Dma_NA_00997] Requirements not relevant for the DMA Driver

Upstream requirements: SRS_BSW_00004, SRS_BSW_00168, SRS_BSW_00170, SRS_BSW_00425, SRS_BSW_00432, SRS_BSW_00461, SRS_BSW_00478

⌈These requirements are not applicable to the DMA Driver:

- SRS_BSW_00004: the DMA Driver does not import include files of other modules that require a version check.
- SRS_BSW_00168, SRS_BSW_00170: the DMA Driver is not tested via diagnostic services and is not an AUTOSAR software component.
- SRS_BSW_00425: the requirement addresses the BSW module description template.
- SRS_BSW_00432, SRS_BSW_00478: the DMA Driver has no main function.
- SRS_BSW_00461: the DMA Driver is not called by a generic module.

⌋

B Change history of AUTOSAR traceable items

Please note that the lists in this chapter also include traceable items that have been removed from the specification in a later version. These items do not appear as hyperlinks in the document.

B.1 Traceable item history of this document according to AUTOSAR Release R26-11

B.1.1 Added Specification Items in R26-11

Number	Heading
[CP_SWS_Dma_00001]	Module Initialization Behavior
[CP_SWS_Dma_00002]	Module De-initialization Behavior
[CP_SWS_Dma_00003]	Enable Channel Behavior
[CP_SWS_Dma_00004]	Start Channel Transfer Behavior
[CP_SWS_Dma_00005]	Stop Channel Transfer Behavior
[CP_SWS_Dma_00006]	Get Channel Status Behavior
[CP_SWS_Dma_00007]	Enable Channel Notifications Behavior
[CP_SWS_Dma_00008]	Disable Channel Notifications Behavior
[CP_SWS_Dma_00010]	Get Latest Transfer Index Behavior
[CP_SWS_Dma_00011]	Get CRC Result Behavior
[CP_SWS_Dma_00012]	Update Transfer Parameters Behavior
[CP_SWS_Dma_00013]	Channel Transfer Configuration Update Behavior
[CP_SWS_Dma_00014]	Disable Channel Behavior
[CP_SWS_Dma_00015]	Suspend Channel Transfer Behavior
[CP_SWS_Dma_00016]	Resume Channel Transfer Behavior



△

Number	Heading
[CP_SWS_Dma_00018]	Data Width Adaptation Transfer
[CP_SWS_Dma_00027]	Scatter-Gather Descriptor Validation
[CP_SWS_Dma_00028]	Inner and Outer Loop Execution Semantics
[CP_SWS_Dma_00029]	Scatter-Gather Runtime Error Signaling
[CP_SWS_Dma_00030]	Scatter-Gather Runtime Error Countermeasure
[CP_SWS_Dma_00031]	Scatter-Gather Recovery Handling
[CP_SWS_Dma_00032]	DMA Channel Initialization Scope
[CP_SWS_Dma_00034]	DMA Channel Configuration Scope
[CP_SWS_Dma_00100]	Definition of API function Dma_Init
[CP_SWS_Dma_00101]	Definition of API function Dma_DeInit
[CP_SWS_Dma_00102]	Definition of API function Dma_StartChannelTransfer
[CP_SWS_Dma_00103]	Definition of API function Dma_StopChannelTransfer
[CP_SWS_Dma_00104]	Definition of API function Dma_GetChannelStatus
[CP_SWS_Dma_00106]	Definition of API function Dma_EnableChannel
[CP_SWS_Dma_00107]	Definition of API function Dma_DisableChannel
[CP_SWS_Dma_00109]	Definition of API function Dma_GetLatestTransferIndex
[CP_SWS_Dma_00110]	Definition of API function Dma_EnableChannelNotifications
[CP_SWS_Dma_00111]	Definition of API function Dma_DisableChannelNotifications
[CP_SWS_Dma_00112]	Definition of API function Dma_GetCrcResult
[CP_SWS_Dma_00113]	Definition of API function Dma_UpdateTransferParameters
[CP_SWS_Dma_00114]	Definition of API function Dma_ChannelTransferConfigUpdate

▽

△

Number	Heading
[CP_SWS_Dma_00115]	Definition of API function Dma_SuspendChannelTransfer
[CP_SWS_Dma_00116]	Definition of API function Dma_ResumeChannelTransfer
[CP_SWS_Dma_00123]	Definition of configurable interface <Dma_TransferCompleteNotification>
[CP_SWS_Dma_00124]	Definition of configurable interface <Dma_TransferErrorNotification>
[CP_SWS_Dma_00125]	Definition of configurable interface <Dma_HardwareErrorNotification>
[CP_SWS_Dma_00126]	Definition of Std_ReturnType-extension for module Dma
[CP_SWS_Dma_00127]	Definition of imported datatypes of module Dma
[CP_SWS_Dma_00128]	Definition of mandatory interfaces required by module Dma
[CP_SWS_Dma_00129]	Definition of optional interfaces requested by module Dma
[CP_SWS_Dma_00130]	Definition of datatype Dma_ChannelIdType
[CP_SWS_Dma_00132]	Definition of datatype Dma_ChannelStatusType
[CP_SWS_Dma_00133]	Definition of datatype Dma_CrcResultType
[CP_SWS_Dma_00134]	Definition of datatype Dma_ChannelInfoType
[CP_SWS_Dma_00135]	Definition of datatype Dma_TcdIdxType
[CP_SWS_Dma_00136]	Definition of datatype Dma_TcdCfgType
[CP_SWS_Dma_00137]	Definition of datatype Dma_SourceConfigType
[CP_SWS_Dma_00138]	Definition of datatype Dma_DestinationConfigType
[CP_SWS_Dma_00139]	Definition of datatype Dma_InOutLoopConfigType
[CP_SWS_Dma_00140]	Definition of API function Dma_GetTransferParameters
[CP_SWS_Dma_00143]	Definition of datatype Dma_ConfigType
[CP_SWS_Dma_00201]	Definition of development errors in module Dma

▽

△

Number	Heading
[CP_SWS_Dma_00202]	Uninitialized Module Detection
[CP_SWS_Dma_00203]	Scatter-Gather Descriptor Configuration Validation
[CP_SWS_Dma_00204]	Definition of runtime errors in module Dma
[CP_SWS_Dma_00205]	Transfer Error Reporting
[CP_SWS_Dma_00206]	Hardware Error Reporting
[CP_SWS_Dma_00207]	Transfer Restart After Transfer Error
[CP_SWS_Dma_00209]	DMA Channel Partition Mapping Validation
[CP_SWS_Dma_00210]	Descriptor Error Reporting
[CP_SWS_Dma_00211]	Channel Busy Reporting
[CP_SWS_Dma_00212]	Channel Parameter Validation
[CP_SWS_Dma_00300]	Module Initialization Error Detection
[CP_SWS_Dma_00302]	Start Channel Transfer Rejection Behavior
[CP_SWS_Dma_00304]	Get Latest Transfer Index Error Detection
[CP_SWS_Dma_00306]	Channel Transfer Configuration Update Error Detection
[CP_SWS_Dma_00307]	Scatter-Gather Configuration Error Detection
[CP_SWS_Dma_00309]	Enable Channel State Transition Behavior
[CP_SWS_Dma_00310]	Transfer Completion Notification Reporting
[CP_SWS_Dma_00311]	Transfer Error Notification Reporting
[CP_SWS_Dma_00312]	Polling Status Reflection Behavior
[CP_SWS_Dma_00313]	Scatter-Gather Descriptor Address and Alignment Validation
[CP_SWS_Dma_00314]	Scatter-Gather Width and Loop Consistency Validation

▽

△

Number	Heading
[CP_SWS_Dma_00315]	Scatter-Gather Runtime Error Status Reflection
[CP_SWS_Dma_00316]	Scatter-Gather CRC Failure Error Classification
[CP_SWS_Dma_00318]	DMA Logical Channel to Hardware Unit Mapping
[CP_SWS_Dma_00319]	DMA Channel Partition Request Scope
[CP_SWS_Dma_00320]	DMA Hardware Unit Configuration Independence
[CP_SWS_Dma_00321]	Scatter-Gather Configuration Rejection Status
[CP_SWS_Dma_00322]	Scatter-Gather Recovery Restart Trigger
[CP_SWS_Dma_00323]	Enable Channel Notifications Callback Configuration Error Detection
[CP_SWS_Dma_00324]	Get CRC Result Failure Return Behavior
[CP_SWS_Dma_00325]	Get CRC Result Error Detection
[CP_SWS_Dma_00326]	Update Transfer Parameters Wrong-State Rejection Behavior
[CP_SWS_Dma_00327]	Channel Transfer Configuration Update Wrong-State Rejection Behavior
[CP_SWS_Dma_00328]	Hardware Error Notification Reporting
[CP_SWS_Dma_00330]	Get Channel Status Ongoing-Transfer Polling Behavior
[CP_SWS_Dma_00331]	Get Channel Status Successful-Completion Polling Behavior
[CP_SWS_Dma_00332]	Get Channel Status Transfer-Error Polling Behavior
[CP_SWS_Dma_00333]	Get Channel Status Hardware-Error Polling Behavior
[CP_SWS_Dma_00334]	Get Channel Status Hardware-Triggered Channel Rejection
[CP_SWS_Dma_00360]	API Call Context Rule
[CP_SWS_Dma_00361]	Synchronous API Execution Rule
[CP_SWS_Dma_00362]	Same-Channel API Serialization Rule

▽

△

Number	Heading
[CP_SWS_Dma_00363]	Start Channel Transfer Completion Observation Rule
[CP_SWS_Dma_00364]	Wrong Partition or Core Access Rejection Behavior
[CP_SWS_Dma_00365]	Get Transfer Parameters Behavior
[CP_SWS_Dma_00366]	Get Transfer Parameters Wrong-State Rejection Behavior
[CP_SWS_Dma_00367]	Get Transfer Parameters Error Detection
[CP_SWS_Dma_00368]	Update Transfer Parameters Runtime Alignment Handling
[CP_SWS_Dma_00369]	Get Transfer Parameters Optional Source or Destination Configuration Handling
[CP_SWS_Dma_00370]	Update Transfer Parameters Scatter-Gather Rejection Behavior
[CP_SWS_Dma_00371]	Suspend Channel Transfer Hardware-Triggered Channel Rejection
[CP_SWS_Dma_00372]	Resume Channel Transfer Hardware-Triggered Channel Rejection
[CP_SWS_Dma_00373]	Enable Channel Software-Triggered Channel Rejection
[CP_SWS_Dma_00374]	Disable Channel Software-Triggered Channel Rejection
[CP_SWS_Dma_00375]	Start Channel Transfer Hardware-Triggered Channel Rejection
[CP_SWS_Dma_00376]	Stop Channel Transfer Hardware-Triggered Channel Rejection
[CP_SWS_Dma_00377]	Stop Channel Transfer Hardware-Error State Rejection
[CP_SWS_Dma_00378]	Disable Channel Hardware-Error State Rejection
[CP_SWS_Dma_00379]	Enable Channel Rejection Behavior
[CP_SWS_Dma_00380]	Default Notification State after Initialization
[CP_SWS_Dma_00381]	Enable Channel Notifications Wrong-State Rejection Behavior
[CP_SWS_Dma_00382]	Disable Channel Notifications Wrong-State Rejection Behavior
[CP_SWS_Dma_00383]	Start Channel Transfer Disabled-Notification Rejection

▽

△

Number	Heading
[CP_SWS_Dma_00384]	Enable Channel Disabled-Notification Rejection
[CP_SWS_Dma_00385]	Complete Transfer Control Descriptor
[CP_SWS_Dma_00386]	Start Channel Transfer Incomplete Descriptor Rejection
[CP_SWS_Dma_00387]	Enable Channel Incomplete Descriptor Rejection
[CP_SWS_Dma_00388]	TCD Index Parameter Validation
[CP_SWS_Dma_00389]	TCD Index Mapping
[CP_SWS_Dma_00390]	Definition of DMA_TCD_IDX_NONE
[CP_SWS_Dma_00391]	Next TCD Link Initialization
[CP_SWS_Dma_00392]	Update Transfer Parameters Next TCD Link
[CP_SWS_Dma_00393]	Update Transfer Parameters Next TCD Validation
[ECUC_Dma_00001]	Definition of EcucModuleDef Dma
[ECUC_Dma_00002]	Definition of EcucParamConfContainerDef DmaConfigSet
[ECUC_Dma_00003]	Definition of EcucParamConfContainerDef DmaGeneral
[ECUC_Dma_00004]	Definition of EcucBooleanParamDef DmaDevErrorDetect
[ECUC_Dma_00005]	Definition of EcucParamConfContainerDef DmaChannel
[ECUC_Dma_00006]	Definition of EcucParamConfContainerDef DmaHwUnit
[ECUC_Dma_00007]	Definition of EcucParamConfContainerDef DmaTcd
[ECUC_Dma_00008]	Definition of EcucIntegerParamDef DmaChannelId
[ECUC_Dma_00009]	Definition of EcucReferenceDef DmaChannelHwUnitRef
[ECUC_Dma_00010]	Definition of EcucBooleanParamDef DmaChannelScatterGatherEnable
[ECUC_Dma_00011]	Definition of EcucReferenceDef DmaChannelTcdRef
[ECUC_Dma_00012]	Definition of EcucFunctionNameDef DmaChannelCompletedNotification Callback
[ECUC_Dma_00013]	Definition of EcucFunctionNameDef DmaChannelTransferErrorNotification Callback
[ECUC_Dma_00014]	Definition of EcucEnumerationParamDef DmaChannelOperationMode
[ECUC_Dma_00015]	Definition of EcucFunctionNameDef DmaChannelHardwareErrorNotification Callback
[ECUC_Dma_00016]	Definition of EcucReferenceDef DmaChannelEcucPartitionRef
[ECUC_Dma_00017]	Definition of EcucParamConfContainerDef DmaChannelHwSpecificConfig
[ECUC_Dma_00018]	Definition of EcucParamConfContainerDef DmaCrcHwSpecificConfig

▽

△

Number	Heading
[ECUC_Dma_00019]	Definition of EcucParamConfContainerDef DmaHwUnitHwSpecificConfig
[ECUC_Dma_00020]	Definition of EcucIntegerParamDef DmaTcdId
[ECUC_Dma_00021]	Definition of EcucReferenceDef DmaTcdEcucPartitionRef
[ECUC_Dma_00022]	Definition of EcucReferenceDef DmaTcdNextTcdRef
[ECUC_Dma_00023]	Definition of EcucParamConfContainerDef DmaInOutLoopConfig
[ECUC_Dma_00024]	Definition of EcucParamConfContainerDef DmaInnerLoop
[ECUC_Dma_00025]	Definition of EcucIntegerParamDef DmaInnerLoopCount
[ECUC_Dma_00026]	Definition of EcucParamConfContainerDef DmaInnerLoopHwSpecificConfig
[ECUC_Dma_00027]	Definition of EcucParamConfContainerDef DmaOuterLoop
[ECUC_Dma_00028]	Definition of EcucParamConfContainerDef DmaOuterLoopHwSpecificConfig
[ECUC_Dma_00029]	Definition of EcucIntegerParamDef DmaOuterLoopCount
[ECUC_Dma_00030]	Definition of EcucParamConfContainerDef DmaTcdHwSpecificConfig
[ECUC_Dma_00031]	Definition of EcucParamConfContainerDef DmaSourceConfig
[ECUC_Dma_00032]	Definition of EcucIntegerParamDef DmaSourceAddress
[ECUC_Dma_00033]	Definition of EcucIntegerParamDef DmaSourceTransferElementWidth
[ECUC_Dma_00034]	Definition of EcucParamConfContainerDef DmaSourceHwSpecificConfig
[ECUC_Dma_00035]	Definition of EcucParamConfContainerDef DmaDestinationConfig
[ECUC_Dma_00036]	Definition of EcucIntegerParamDef DmaDestinationAddress
[ECUC_Dma_00037]	Definition of EcucIntegerParamDef DmaDestinationTransferElementWidth
[ECUC_Dma_00038]	Definition of EcucParamConfContainerDef DmaDestinationHwSpecificConfig

Table B.1: Added Specification Items in R26-11

B.1.2 Changed Specification Items in R26-11

none

B.1.3 Deleted Specification Items in R26-11

none

B.1.4 Added Constraints in R26-11

Number	Heading
[CP_SWS_Dma_CONSTR_00001]	Transfer Element Width Allowed Configuration Values
[CP_SWS_Dma_CONSTR_00002]	Configured Address Alignment Constraint
[CP_SWS_Dma_CONSTR_00003]	Channel Operation Mode Interrupt Notification Requirement
[CP_SWS_Dma_CONSTR_00004]	Channel Operation Mode Polling Notification Prohibition
[CP_SWS_Dma_CONSTR_00005]	Transfer Element Width Ratio Constraint
[CP_SWS_Dma_CONSTR_00006]	Next Tcd Reference Scatter-Gather Enablement Constraint
[CP_SWS_Dma_CONSTR_00007]	Tcd Partition Consistency Constraint

Table B.2: Added Constraints in R26-11

B.1.5 Changed Constraints in R26-11

none

B.1.6 Deleted Constraints in R26-11

none