OCU driver user guide

TRAVEO™ T2G family

About this document

Scope and purpose

This guide describes the architecture, configuration and use of the output compare unit (OCU) driver. It will help you to understand the functionality of the driver and will provide a reference to the driver's API.

The installation, build process, and general information about the use of EB tresos Studio software are not within the scope of this document.

Intended audience

This document is intended for anyone who uses the OCU driver of the TRAVEO™ T2G family.

Document structure

Chapter 1 General overview gives a brief introduction to the OCU driver, explains the embedding in the AUTOSAR environment and describes the supported hardware and development environment.

Chapter 2 Using the OCU driver details the steps required to use the OCU driver in your application.

Chapter 3 Structure and dependencies describes the file structure and the dependencies for the OCU driver.

Chapter 4 EB tresos Studio configuration interface describes the driver's configuration.

Chapter 5 Functional description gives a functional description of all services offered by the OCU driver.

Chapter 6 Hardware resources gives a description of all hardware resources used.

The Appendix A and Appendix B provides a complete API reference and access register table.

Abbreviations and definitions

Abbreviation	Definition
ADC	Analog Digital Converter
API	Application Programming Interface
ASCII	American Standard Code for Information Interchange
ASIL	Automotive Safety Integrity Level
AUTOSAR	Automotive Open System Architecture
BSW	Basic Software. Standardized part of software which does not fulfill a vehicle functional job.
Channel	Represents a functional counter instance; one compare register of hardware configuration.
Compare threshold	Target value that is compared with the content of the counter each time the counter is increased by one unit.
DEM	Diagnostic Event Manager
DET	Default Error Tracer

About this document

Abbreviation	Definition
DMA	Direct Memory Access
DW	Data Wire, CPU feature is used for peripheral-to-memory and memory-to-peripheral data transfers. DW is also called Peripheral-DMA (P-DMA) controller. Generically, this feature is called “DMA”.
EcuM	ECU State Manager
EB tresos Studio	Elektrobit Automotive configuration framework
Free running counter	A counter that runs from a minimum (respectively a maximum) to a maximum (respectively a minimum) value and restarts automatically from the minimum (respectively a maximum) after reaching the maximum (respectively the minimum) value.
GCE	Generic Configuration Editor
IRQ	Interrupt Request
ISR	Interrupt Service Routine
µC	Microcontroller
MCU	Microcontroller Unit
MCAL	Microcontroller Abstraction Layer
OS	Operating System
OCU	Output Compare Unit
TCPWM	Timer Counter Pulse Width Modulation
Tick	Defines the timer resolution, the duration of a timer increment
UART	Universal Asynchronous Receiver-Transmitter
UTF-8	8-Bit Universal Character Set Transformation Format

Related documents

AUTOSAR requirements and specifications

Bibliography

[1] General specification of basic software modules, AUTOSAR release 4.2.2.

[2] Specification of OCU driver, AUTOSAR release 4.2.2.

[3] Specification of standard types, AUTOSAR release 4.2.2.

[4] Specification of ECU configuration parameters, AUTOSAR release 4.2.2.

[5] Specification of default error tracer, AUTOSAR release 4.2.2.

Elektrobit automotive documentation

Bibliography

[6] EB tresos Studio for ACG8 user’s guide.

Hardware documentation

The hardware documents are listed in the delivery notes.

About this document

Related standards and norms

Bibliography

[7] Layered software architecture, AUTOSAR release 4.2.2.

Table of contents

Table of contents

Table of contents

1 General overview

1.1 Introduction to the OCU driver

The OCU driver is a set of software routines, which enable you to create OCU signals on dedicated output pins of the CPU.

For this purpose, the OCU driver provides services for modifying the pin action and the pin state of an OCU signal. In addition, the OCU driver provides services for initialing and de-initializing the OCU module, setting comparison threshold, triggering peripheral action, and enabling or disabling of a notification callback function.

The OCU driver is not responsible for initializing or configuring hardware ports. This task is performed by the PORT driver.

The driver conforms to the AUTOSAR standard and is implemented according to the *Specification of OCU driver* [2].

The OCU driver is delivered with a plugin for the EB tresos Studio, which allows you to statically configure the driver. The driver provides an interface to enable OCU channels and to configure default threshold, default pin state and other parameters.

1.2 User profile

This guide is intended for users with a least basic knowledge of the following:

- Automotive embedded systems
- C programming language
- AUTOSAR standard
- Target hardware architecture

1.3 Embedding in the AUTOSAR environment

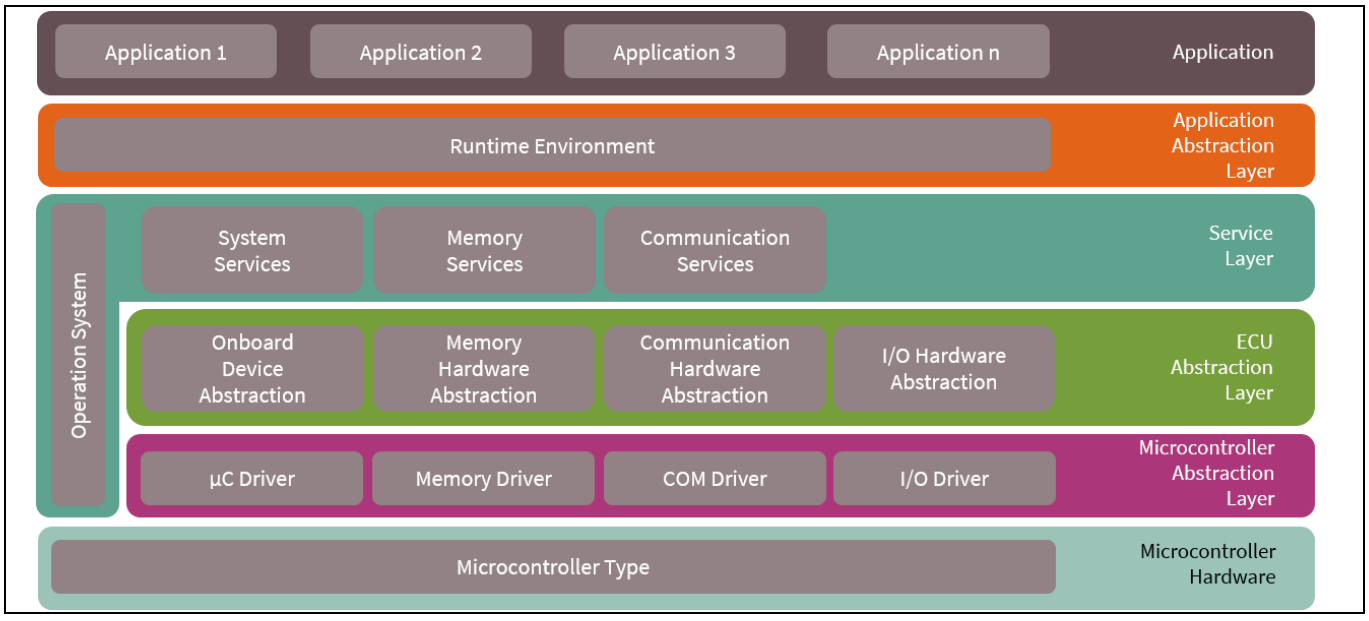

Figure 1 Overview of AUTOSAR software layers

Figure 1 depicts the layered AUTOSAR software architecture. The OCU driver (Figure 2) is part of group of I/O drivers within the microcontroller abstraction layer (MCAL), the lowest layer of basic software in the AUTOSAR environment.

For an exact overview of the AUTOSAR layered software architecture, see the *layered software architecture* [7].

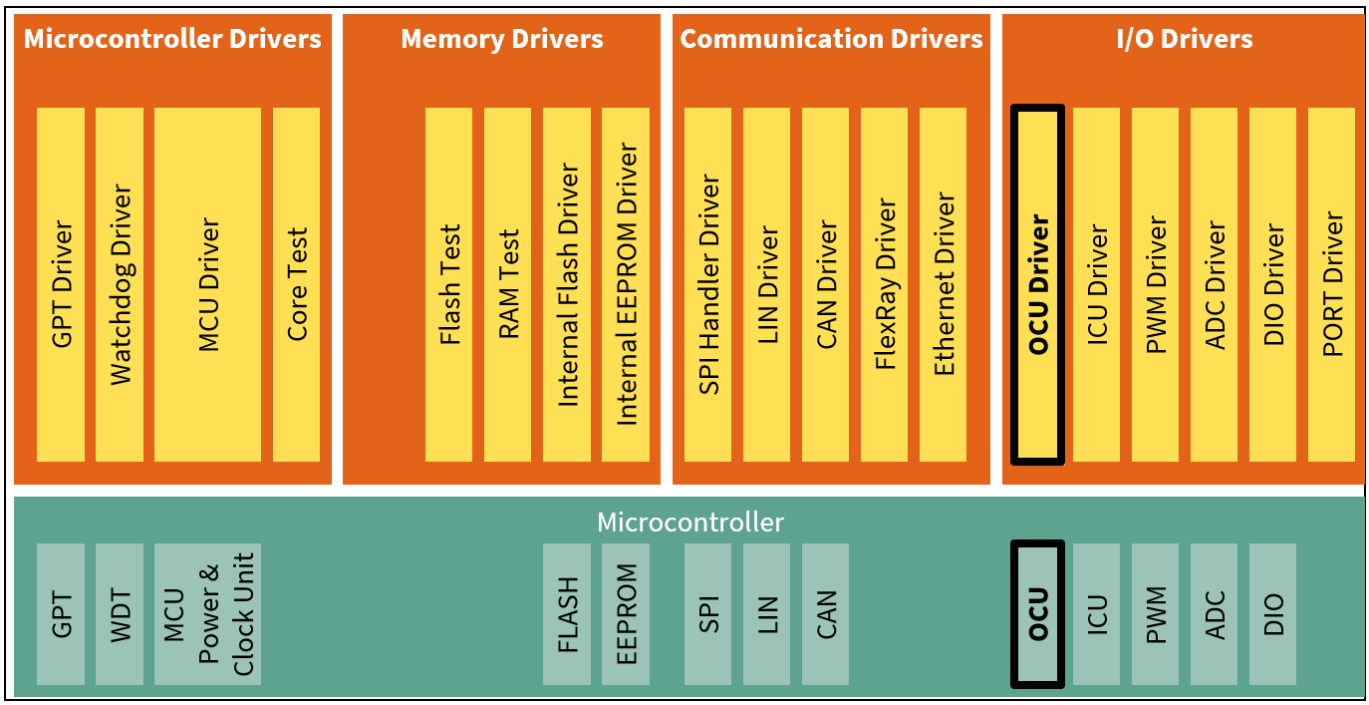

Figure 2 OCU driver in MCAL layer

1.4 Supported hardware

This version of the OCU driver supports the TRAVEO™ T2G family. No special external hardware devices are required.

The supported derivatives are listed in the release notes.

1.5 Development environment

The development environment corresponds to AUTOSAR release 4.2.2. The modules BASE, MAKE, MCU, PORT and RESOURCE are needed for proper functionality of the OCU driver.

1.6 Character set and encoding

All source code files of the OCU driver are restricted to the ASCII character set. The files are encoded in UTF-8 format, with only the 7-bit subset (values 0x00 … 0x7F) being used.

2 Using the OCU driver

This chapter describes all necessary steps to incorporate the OCU driver into your application.

2.1 Installation and prerequisites

Note: Before you start, see the EB tresos Studio for ACG8 user's guide [6] for the following information.

1. *The installation procedure of EB tresos ECU AUTOSAR components*
2. *The usage of the EB tresos Studio*
3. *The usage of the EB tresos ECU AUTOSAR build environment (EB tresos Studio for ACG8 user's guide [6] includes the steps to setup and integrate the own application within the EB tresos ECU AUTOSAR build environment)*

The installation of the OCU driver corresponds with the general installation procedure for EB tresos AUTOSAR components given in the documents *EB tresos Studio for ACG8 user's guide* [6]. If the driver has been successfully installed, the driver will appear in the module list of the EB tresos Studio (see the *EB tresos Studio for ACG8 user's guide* [6]).

This document assumes that the project is properly set up and is using the application template as described in the *EB tresos Studio for ACG8 user's guide* [6]. This template provides the necessary folder structure, project, and makefiles needed to configure and compile your application within the build environment. You also must be familiar with the use of the command shell.

2.2 Configuring the OCU driver

The OCU driver can be configured with any AUTOSAR compliant GCE tool. Save the configuration in a separate file named e.g. *Ocu.epc*. More information about the OCU driver configuration can be found in chapter 4 EB tresos Studio configuration interface.

2.3 Adapting your application

To use the OCU driver in your application, include the header files of OCU, MCU and PORT driver by adding the following lines of code to your source file:

```
#include "Mcu.h"
#include "Port.h"
#include "Ocu.h"
```

This publishes all needed function and data prototypes and symbolic names of the configuration into the application. In addition, you must implement the error callout function for ASIL safety extension.

To use the OCU driver, the appropriate Port pins, TCPWM clock setting, and OCU interrupts must be configured in PORT driver, MCU driver, and OS. For detailed information, see chapter 6 Hardware resources.

Initialization of MCU, PORT, and OCU driver needs to be done in the following order:

```
Mcu_Init(&Mcu_Config[0]);
Port_Init(&Port_Config[0]);
Ocu_Init(&Ocu_Config[0]);
```

Note: As a reference, the symbolic name can also be specified (e.g. `OcuConf_OcuConfigSet_OcuConfigSet_0`).

The function `Mcu_Init()` is called with a pointer to a structure of type `Mcu_ConfigType`, which is published by the MCU driver itself.

The function `Port_Init()` is called with a pointer to a structure of type `Port_ConfigType`, which is published by the PORT driver itself.

The function `Ocu_Init()` is called with a pointer to a structure of type `Ocu_ConfigType`, which is published by the OCU driver itself.

After initialization of the OCU driver, its API functions can be used. Each configured OCU channel operates, but no callback functions are called until you enable the functions.

`OcuStartTriggerSelect<n>` (<n> = 0 or 1) specifies the input trigger of TCPWM instance <n> to start OCU counters synchronously. When this parameter is configured, a trigger signal is required to start all TCPWM instance <n> counters. It is necessary to call `Port_ActTrigger()` to issue a trigger signal after calling `Ocu_Init()`. In this case, the group trigger configuration is also required in PORT driver. If `Port_ActTrigger()` is not called, you cannot use the OCU functions. When this parameter is not configured, `Ocu_Init()` starts the TCPWM instance <n> counters sequentially.

Note: TCPWM instance number depends on subderivative. See the hardware manual.

Your application must provide the notification functions and its declarations that you configured. The file containing the declarations must be included using the `OcuGeneral/OcuIncludeFile` parameter. The notification functions take no parameters and have void return type:

```
void MyNotificationFunction(void)
{
/* Insert your code here */
}
```

The notification function is called from an interrupt context.

2.4 Starting the build process

Do the following to build your application:

Note: For a clean build, use the build command with target `clean_all`. before (`make clean_all`).

1. Type the following in the command shell to generate the necessary dependent files:

```
> make generate
```

The details of the generated files are described in 3.3 Generated files.

2. Then, type the following command to resolve the required file dependencies:

```
> make depend
```

3. Type the following command to compile and link the application:

```
> make (optional target: all)
```

The application is now built. All files are compiled and linked to a binary file, which can be downloaded to the target hardware.

2.5 Measuring stack consumption

Do the following to measure stack consumption. It requires the Base module for proper measurement.

Note: *All files (including library files) should be rebuilt with a dedicated compiler option. The executable file built in this step must be used only to measure stack consumption.*

To measure stack consumption:

1. Add the following compiler option to the Makefile to enable stack consumption measurement:

   ```
   -DSTACK_ANALYSIS_ENABLE
   ```

2. Type the following command to clean library files.

   ```
   > make clean_lib
   ```

3. Follow the build process described in 2.4 Starting the build process.
4. Follow the instructions in the release notes and measure the stack consumption.

2.6 Memory mapping

The *Ocu_MemMap.h* file in the *$(TRESOS_BASE)/plugins/MemMap_TS_T40D13M0I0R0/include* directory is a sample. This file is replaced by the file generated by MEMMAP module. Input to MEMMAP module is generated as *Ocu_Bswmd.arxml* in the *$(PROJECT_ROOT)/ output/generated/swcd* directory of your project folder.

2.6.1 Memory allocation keyword

- `OCU_START_SEC_CODE_ASIL_B / OCU_STOP_SEC_CODE_ASIL_B`

  The memory section type is CODE. All executable code is allocated in this section.

- `OCU_START_SEC_CONST_ASIL_B_UNSPECIFIED / OCU_STOP_SEC_CONST_ASIL_B_UNSPECIFIED`

  The memory section type is CONST. The following constants are allocated in this section:

  - OCU hardware configuration setting
  - OCU channel configuration setting
  - OCU whole configuration setting

- `OCU_START_SEC_VAR_INIT_ASIL_B_UNSPECIFIED / OCU_STOP_SEC_VAR_INIT_ASIL_B_UNSPECIFIED`

  The memory section type is VAR. The following variables are allocated in this section:

  - Point to whole configuration setting
  - Information for OCU driver status

- `OCU_START_SEC_VAR_NO_INIT_ASIL_B_UNSPECIFIED / OCU_STOP_SEC_VAR_NO_INIT_ASIL_B_UNSPECIFIED`

  The memory section type is VAR. The following variables are allocated in this section:

  - Information for OCU channel status
  - Information for count direction

3 Structure and dependencies

The OCU driver consists of static, configuration, and generated files.

3.1 Static files

- $(PLUGIN_PATH)=$(TRESOS_BASE)/plugins/OCU_TS_* is the path to the OCU driver plugin.
- *$(PLUGIN_PATH)/lib_src* contains all static source files of the OCU driver. These files contain the functionality of the driver which does not depend on the current configuration. The files are grouped into a static library.
- *$(PLUGIN_PATH)/lib_include* contains all internal header files for the OCU driver.
- *$(PLUGIN_PATH)/src* comprises configuration dependent source files or special derivative files. Each file will be rebuilt when the configuration is changed.

All necessary source files will automatically be compiled and linked during the build process and all include paths will be set if the OCU driver is enabled.

- *$(PLUGIN_PATH)/include* is the basic public include directory needed by the user to include *Ocu.h*.
- *$(PLUGIN_PATH)/autosar* directory contains the AUTOSAR ECU parameter definition with vendor, architecture, and derivative specific adaptations to create a correct matching parameter configuration for the OCU driver.

3.2 Configuration files

The configuration of the OCU driver is done via EB tresos Studio. The file containing the OCU driver's configuration is named *Ocu.xdm* and is in the *$(PROJECT_ROOT)/config* directory. This file serves as input for the generation of the configuration dependent source and header files during the build process.

3.3 Generated files

During the build process, the following files are generated based on the current configuration description and are in the *output/generated* sub folder of your project folder.

The following basic files s are used to configure common behavior.

- *include/Ocu_Cfg.h* provides settings of configurations with pre-compile attribute, for example, declares interrupt service routine (ISR) function, and provides all symbolic names of the configuration. This is included in *Ocu.h*.
- *include/Ocu_Cfg_Include.h* includes the header files specified by `OcuIncludeFile`.
- *include/Ocu_PBcfg.h* provides settings of configurations with post-build attribute, for example, symbolic names of module configurations. It will be included by *Ocu.h*.
- *src/Ocu_Irq.c* contains the OCU channel ISR implementation.
- *src/Ocu_PBcfg.c* contains the constant structure for the OCU configuration.

Note: *Generated source files need not to be added to your application make file. These files will be compiled and linked automatically during the build process.*

- *swcd/Ocu_Bswmd.arxml* contains BSW module description.

Note: *Additional steps are required for the generation of BSW module description. In EB tresos Studio, follow the menu path **Project** > **Build Project** and click **generate_swcd**.*

3.4 Dependencies

3.4.1 PORT driver

Although the OCU driver can be successfully compiled and linked without an AUTOSAR compliant PORT driver, the latter is required to configure and initialize all ports. Also, the configuration of triggers is required and it is necessary to call `Port_ActTrigger()` to issue a trigger signal after calling `Ocu_Init()` and if the `OcuStartTriggerSelect0` or `OcuStartTriggerSelect1` parameter is configured. If `Port_ActTrigger()` is not called, you cannot use the OCU functions. The PORT driver needs to be initialized before the OCU driver is initialized. See the *PORT driver's user guide* for details.

3.4.2 MCU driver

The MCU driver needs to be initialized and all MCU clock reference points referenced by the OCU driver channels via configuration parameter `OcuChannelClockSource` must have been activated (via calls of MCU API functions) before initializing the OCU driver. See the *MCU driver's user guide* for details.

3.4.3 AUTOSAR OS

The OS must be used to configure and create the ISR vector table entries for the OCU used channels. See the hardware and derivatives specified in 6.3 Interrupts.

3.4.4 DET

If the default error detection is enabled in the OCU driver configuration, the DET module needs to be installed, configured, and integrated into the application also.

3.4.5 Error callout handler

The error callout handler is called on every error that is detected, regardless of whether default error detection is enabled or disabled. The error callout handler is an ASIL safety extension that is not specified by AUTOSAR. It is configured via the configuration parameter `OcuErrorCalloutFunction`.

3.4.6 BSW scheduler

The OCU driver uses the following services of the BSW scheduler to enter and leave critical sections:

- `SchM_Enter_Ocu_OCU_EXCLUSIVE_AREA_0(void)`
- `SchM_Exit_Ocu_OCU_EXCLUSIVE_AREA_0(void)`

You must ensure that the BSW scheduler is properly configured and initialized before using the OCU driver.

4 EB tresos Studio configuration interface

Use an AUTOSAR compliant GCE to adapt the configuration files as desired.

Note: The ECU parameter description of the OCU driver basically corresponds to one defined by AUTOSAR. However, as there are some vendor-specific extensions, you must use the ECU parameter description file delivered with the OCU driver. There are two types of file:

- The EB tresos Studio configuration description *($(TRESOS_BASE)/plugins/Ocu_TS_<VARIANT>/config/Ocu.xdm)*. This file is automatically loaded when the module is selected in EB tresos. The file contains specific support for calculated values and error checks.
- The AUTOSAR compatible configuration description *($(TRESOS_BASE)/plugins/Ocu_TS_<VARIANT>/autosar/Ocu.arxml)*. This file can be used as basis for GCEs other than Tresos. The file does not contain specific support for calculated values and error checks, but only general ranges that cover all possible values for all derivatives.

4.1 General configuration

The module comes preconfigured with default settings. These settings must be adapted, if necessary.

4.2 Standard parameters

This section lists all configuration parameters required by AUTOSAR. Deviations or hardware specific restrictions from the specification are mentioned where applicable.

4.2.1 Container OcuGeneral

4.2.1.1 OcuDevErrorDetect

Description

Enables or disables development error notification for the OCU driver.

Type

`EcucBooleanParamDef`

Remarks

Setting this parameter to FALSE will disable the notification of development errors via DET. However, in contrast to AUTOSAR specification, detection of development errors is still enabled and errors will be reported via `OcuErrorCalloutFunction`.

4.2.2 Container OcuConfigSet

4.2.2.1 OcuCountdirection

Description

Indicates the count direction for the whole OCU driver.

Type

`EcucEnumerationParamDef`

Range

`OCU_DOWNCOUNTING`: Counter will reckon from the maximum to the minimum value.
`OCU_UPCOUNTING`: Counter will reckon from the minimum to the maximum value.

Remarks

None

4.2.3 Container OcuChannel

4.2.3.1 OcuAssignedHardwareChannel

Description

The physical hardware channel assigned to this logical channel.

Type

`EcucIntegerParamDef`

Remarks

This parameter is not used by the OCU driver and therefore not evaluated. Instead, OCU physical channel can be selected by the `OcuTimer` parameter.

4.2.3.2 OcuChannelId

Description

Channel Id of the OCU channel.

Type

`EcucIntegerParamDef`

Range

0...[65535 or number of configured channels - 1, whichever is lower]

Value must be unique among all channels.

Remarks

This value is assigned to a symbolic name. Use only the symbolic channel names defined in *Ocu_Cfg.h* for API calls.

4.2.3.3 OcuChannelTickDuration

Description

Specifies the tick duration of the counter of the channel.

Type

`EcucIntegerParamDef`

Remarks

The range of this parameter is limited to 1 as the hardware does not support any other value.

4.2.3.4 OcuDefaultThreshold

Description

Value of comparison threshold used for initialization.

Type

`EcucIntegerParamDef`

Range

0...4294967294

Remarks

The value should be less than or equal to `OcuMaxCounterValue`. For a 32-bit TCPWM counter, the range is 0...4294967294. For a 16-bit TCPWM counter, the rage is 0...65534.

4.2.3.5 OcuHardwareTriggeredAdc

Description

This parameter is used to allow the OCU channel to trigger an ADC channel upon compare match, if this is supported by hardware. The value of the parameter represents the ADC physical channel to trigger.

Type

`EcucIntegerParamDef`

Remarks

This parameter is not used by the OCU driver and therefore not evaluated. Instead, triggering ADC channel can be configured by parameter `OcuHwTriggerOutputLine`.

4.2.3.6 OcuHardwareTriggeredDMA

Description

This parameter is used to allow the OCU channel to trigger a DMA channel upon compare match, if this is supported by hardware. The value of the parameter represents the DMA physical channel to trigger.

Type

`EcucIntegerParamDef`

Remarks

This parameter is not used by the OCU driver and therefore not evaluated. Instead, triggering DataWire (DW) channel can be configured by parameter `OcuHwTriggerOutputLine`.

4.2.3.7 OcuMaxCounterValue

Description

Maximum value of ticks the counter of the OCU channel can count.

Type

`EcucIntegerParamDef`

Range

1...4294967294

Remarks

The maximum value of OcuMaxCounterValue depends on the counter width of the TCPWM resource. For a 32-bit TCPWM counter, the range is 1...4294967294. For a 16-bit TCPWM counter, the rage is 1...65534.

4.2.3.8 OcuNotification

Description

Definition of the notification callback function.

Type

`EcucFunctionNameDef`

Remarks

This parameter can be configured as an empty string or NULL can be entered. Both will result in no notification being called. Otherwise, the function name must be unique.

4.2.3.9 OcuOuptutPinUsed

Description

Information about the usage of an output pin on this channel.

Type

`EcucBooleanParamDef`

Range

TRUE: The channel uses an output pin.

FALSE: The channel does not use an output pin.

Remarks

None

4.2.3.10 OcuOutputPinDefaultState

Description

Specifies the state that a pin associated with a channel will be set to after initialization.

Type

`EcucEnumerationParamDef`

Range

OCU_HIGH or OCU_LOW.

Remarks

None

4.2.4 Container OcuConfigurationOfOptionalApis

4.2.4.1 OcuDeInitApi

Description

Adds or removes the service `Ocu_DeInit()` from the code.

Type

`EcucBooleanParamDef`

Remarks

None

4.2.4.2 OcuGetCounterApi

Description

Adds or removes the service `Ocu_GetCounter()` from the code.

Type

`EcucBooleanParamDef`

Remarks

None

4.2.4.3 OcuSetAbsoluteThresholdApi

Description

Adds or removes the service `Ocu_SetAbsoluteThreshold()` from the code.

Type

`EcucBooleanParamDef`

Remarks

None

4.2.4.4 OcuSetRelativeThresholdApi

Description

Adds or removes the service `Ocu_SetRelativeThreshold()` from the code.

Type

`EcucBooleanParamDef`

Remarks

None

4.2.4.5 OcuSetPinStateApi

Description

Adds or removes the service `Ocu_SetPinState()` from the code.

Type

`EcucBooleanParamDef`

Remarks

None

4.2.4.6 OcuSetPinActionApi

Description

Adds or removes the service `Ocu_SetPinAction()` from the code.

Type

`EcucBooleanParamDef`

Remarks

None

4.2.4.7 OcuNotificationSupported

Description

Adds or removes the services `Ocu_EnableNotification()` and `Ocu_DisableNotification()` from the code.

Type

`EcucBooleanParamDef`

Remarks

None

4.2.4.8 OcuVersionInfoApi

Description

Adds or removes the service `Ocu_GetVersionInfo()` from the code.

Type

`EcucBooleanParamDef`

Remarks

None

4.2.5 Container OcuHWSpecificSettings

4.2.5.1 OcuClockSource

Description

Specifies clock sources if provided by hardware.

Type

`EcucEnumerationParamDef`

Remarks

This parameter is not used by the OCU driver and therefore not evaluated. `OcuClockSource` for the whole OCU driver cannot be set because the functionality is not supported by the hardware. In the other way, `OcuChannelClockSource` can be set inside each counter.

4.2.5.2 OcuPrescale

Description

Specifies clock prescale factor if supported by hardware.

Type

`EcucEnumerationParamDef`

Remarks

This parameter is not used by the OCU driver and therefore not evaluated. `OcuPrescale` for whole OCU driver cannot be set because the functionality is not supported by the hardware. In the other way, `OcuChannelPrescale` can be set inside each counter.

4.2.6 Container OcuGroup

4.2.6.1 OcuGroupId

Description

Group Id of the OCU channel group.

Type

`EcucIntegerParamDef`

Remarks

Hardware does not support OcuGroup. So, this parameter is not used by the OCU driver and therefore not evaluated.

4.2.6.2 OcuGroupDefinition

Description

Assignment of OcuChannels to an OcuGroup.

Type

`EcucReferenceDef`

Remarks

Hardware does not support OcuGroup. So, this parameter is not used by the OCU driver and therefore not evaluated.

4.3 Vendor and driver specific parameters

4.3.1 Container OcuGeneral

4.3.1.1 OcuErrorCalloutFunction

Description

Error callout function. Syntax:

```
void ErrorCalloutHandler
(
  uint16 ModuleId,
  uint8  InstanceId,
  uint8  ApiId,
  uint8  ErrorId
);
```

The error callout function is called on every error. The ASIL level of this function limits the ASIL level of the OCU driver.

Type

`EcucFunctionNameDef`

Remarks

`OcuErrorCalloutFunction` must be a valid C function name.

4.3.1.2 OcuIncludeFile

Description

A list of filenames that will be included within the driver. Any application-specific symbol that is used by the OCU configuration (e.g. error callout function) should be included by configuring this parameter.

Type

`EcucStringParamDef`

Remarks

`OcuIncludeFile` must be a unique filename with extension.*h*.

4.3.1.3 OcuStartTriggerSelect0

Description

Specifies the input trigger of TCPWM instance 0 to start OCU counters synchronously. When this parameter is configured, a trigger signal is required to start all TCPWM instance 0 counters. It is necessary to call `Port_ActTrigger()` to issue a trigger signal after calling `Ocu_Init()`. In this case, the group trigger configuration is also required in PORT driver. If `Port_ActTrigger()` is not called, you cannot use the OCU functions. When this parameter is not configured, `Ocu_Init()` starts the TCPWM instance 0 counters sequentially.

Type

`EcucEnumerationParamDef`

Range

- `TCPWM_0_TR_ALL_CNT_IN_0: Input trigger 0 of TCPWM instance 0`
- `TCPWM_0_TR_ALL_CNT_IN_1: Input trigger 1 of TCPWM instance 0`
- `...`
- `TCPWM_0_TR_ALL_CNT_IN_n: Input trigger n of TCPWM instance 0`

Note: The number of available input triggers depends on the hardware resources.

Remarks

ICU, GPT, and PWM drivers use input triggers of TCPWM. In the case that the same input trigger is used in `PwmStartTriggerSelect0`, a warning occurs. In the case that the same input trigger is configured in other modules, an error message occurs.

4.3.1.4 OcuStartTriggerSelect1

Description

Specifies the input trigger of TCPWM instance 1 to start OCU counters synchronously. When this parameter is configured, a trigger signal is required to start all TCPWM instance 1 counters. It is necessary to call `Port_ActTrigger()` to issue a trigger signal after calling `Ocu_Init()`. In this case, the group trigger configuration is also required in PORT driver. If `Port_ActTrigger()` is not called, you cannot use the OCU functions. When this parameter is not configured, `Ocu_Init()` starts the TCPWM instance 1 counters sequentially.

Type

`EcucEnumerationParamDef`

Range

- `TCPWM_1_TR_ALL_CNT_IN_0: Input trigger 0 of TCPWM instance 1`
- `TCPWM_1_TR_ALL_CNT_IN_1: Input trigger 1 of TCPWM instance 1`
- `...`
- `TCPWM_1_TR_ALL_CNT_IN_n: Input trigger n of TCPWM instance 1`

Note: The number of available input triggers depends on the hardware resources.

Remarks

ICU, GPT, and PWM drivers use input triggers of TCPWM. In the case that the same input trigger is used in `PwmStartTriggerSelect1`, a warning occurs. In the case that the same input trigger is configured in other modules, an error message occurs.

4.3.2 Container OcuChannel

4.3.2.1 OcuTimer

Description

Specifies the physical hardware timer assigned to this OCU channel.

Type

`EcucEnumerationParamDef`

Range

- `TCPWM_0_0: Timer Counter Pulse Width Modulation Instance 0 Channel 0`
- `TCPWM_0_1: Timer Counter Pulse Width Modulation Instance 0 Channel 1`
- `...`
- `TCPWM_0_n: Timer Counter Pulse Width Modulation Instance 0 Channel n`
- `TCPWM_1_0: Timer Counter Pulse Width Modulation Instance 1 Channel 0`
- `TCPWM_1_1: Timer Counter Pulse Width Modulation Instance 1 Channel 1`
- `...`
- `TCPWM_1_n: Timer Counter Pulse Width Modulation Instance 1 Channel n`

Note: *The number of available physical hardware timer channels depend on the configured target derivative.*

Remarks

Only one configuration per `OcuTimer` is allowed in the same configuration set.

`OcuTimer` shows all TCPWM resources on device. See Hardware documentation for the TCPWM connected to the pin for `OcuTimer`.

ICU, GPT, PWM drivers, and OS use TCPWM channels. OCU driver must not use TCPWM channel that is used by the other modules.

4.3.2.2 OcuHwTriggerOutputLine

Description

Specifies the output trigger channel. The available output trigger channel depends on device. The trigger connection between output trigger channel (trigger source) and trigger destination should be configured with PORT driver.

Type

`EcucEnumerationParamDef`

Range

`TR_OUT0`: Output trigger 0.

`TR_OUT1`: Output trigger 1.

`BOTH`: Both output trigger 0 and 1.

Remarks

The configuration of trigger connection is required by PORT driver.

4.3.2.3 OcuChannelClockSource

Description

This parameter contains reference to `McuClockReferencePoint`.

Type

`EcucReferenceDef`

Remarks

`OcuChannelClockSource` must have the target's `McuClockReferencePoint` setting.

4.3.2.4 OcuChannelPrescale

Description

OCU module specific prescaler factor per channel.

Type

`EcucEnumerationParamDef`

Range

- OCU_PRESCALE_1: Prescaler value of pre-scaling is divided by 1.
- OCU_PRESCALE_2: Prescaler value of pre-scaling is divided by 2.
- OCU_PRESCALE_4: Prescaler value of pre-scaling is divided by 4.
- OCU_PRESCALE_8: Prescaler value of pre-scaling is divided by 8.
- OCU_PRESCALE_16: Prescaler value of pre-scaling is divided by 16.
- OCU_PRESCALE_32: Prescaler value of pre-scaling is divided by 32.
- OCU_PRESCALE_64: Prescaler value of pre-scaling is divided by 64.
- OCU_PRESCALE_128: Prescaler value of pre-scaling is divided by 128.

Remarks

None

4.3.2.5 OcuDebugMode

Description

Specifies behavior of OCU channel when processor is in debug mode.

Type

`EcucEnumerationParamDef`

Range

- OCU_DEBUG_MODE_CONTINUE: The counter operation continues in debug mode.
- OCU_DEBUG_MODE_HALT: The counter operation freezes in debug mode.

Remarks

None

4.3.3 Container OcuConfigurationOfOptionalApis

4.3.3.1 OcuSafetyFunctionApi

Description

Adds or removes the service `Ocu_CheckChannelStatus()` from the code.

Type

`EcucBooleanParamDef`

Remarks

None

4.3.3.2 OcuSetPrescalerApi

Description

Adds or removes the service `Ocu_SetPrescaler()` from the code.

Type

`EcucBooleanParamDef`

Remarks

None

5 Functional description

5.1 Initialization

The OCU driver needs to be initialized once before use. Initialization of the OCU driver is made by a call to the `Ocu_Init()` function.

Note: This OCU driver supports post-build-time configuration, thus different configuration set pointer can be passed to the `Ocu_Init()` function.

No other function must be called before `Ocu_Init()` has been executed except `Ocu_CheckChannelStatus()` and `Ocu_GetVersionInfo()`.

Not all necessary initializations are performed by the OCU driver. The PORT driver needs to be called before the OCU driver is used. See 2.3 Adapting your application.

After initialization, all configured channels are stopped with disabled notification, but counters run.

After initialization, output pin level of all configured channels will be set to HIGH or LOW according to `OcuOutputPinDefaultState`.

All channels are initialized at once by the `Ocu_Init()` API. There is no API to individually initialize each channel.

5.2 Runtime reconfiguration

The OCU driver is not reconfigurable at runtime. The only way to change the driver's configuration is to stop all channels, de-initialize the driver, and reinitialize with a different configuration set.

5.3 OCU channel

The `OcuChannelId` defines the channel number to use. It specifies the OCU channel for which the service is done. A user given symbolic name is available or generated for each configured channel. Use this symbolic name for API calls.

5.4 OCU channel group

The OCU driver does not support channel group.

5.5 Notification

Notification can be enabled or disabled using the API functions `Ocu_EnableNotification()` and `Ocu_DisableNotification()`.

Notifications are disabled by default after calling `Ocu_Init()`.

If the configured notification function name (`OcuNotification`) is blank or “NULL”, no notification function will be generated.

Note: The notification can be disabled by calling Ocu_DisableNotification as mentioned in this chapter. However, it would not work with notifications that have already been handled, if Ocu_DisableNotification is called from a higher interruption during a few cycles before the user-defined notification function is called.

5.6 Starting a channel

After initialization, a channel can be started by the `Ocu_StartChannel()` function.

All configured compare match actions (e.g. Pin action, notification and output trigger) can be performed after starting a channel.

After a channel is stopped, it can be restarted by the `Ocu_StartChannel()` function.

5.7 Stopping a channel

A channel can be stopped by the `Ocu_StopChannel()` function.

Note: *The counter runs even if the channel is stopped.*
No further configured compare match actions are performed by the driver after the channel was stopped.

5.8 Output trigger to peripherals

The OCU driver generates output triggers to peripherals such as ADC/DW channel upon compare match.

The parameter `OcuHwTriggerOutputLine` allows the OCU channel to trigger peripheral actions. The trigger connection between output trigger channel (trigger source) and trigger destination is configured with PORT driver.

The available output trigger channel depends on the hardware.

5.9 Prescaler setting function

The OCU driver provides the prescaler setting function for OCU channel during runtime. This function is to maintain the same tick frequency of a channel by changing the prescaler setting without initialization when the input clock frequency for a channel is changed.

`Ocu_SetPrescaler()` changes the TCPWM prescaler setting of the selected channel according to the specified input clock frequency.

`Ocu_StopChannel()` stops the OCU channel before calling `Ocu_SetPrescaler()`. If the OCU channel is in running state, `Ocu_SetPrescaler()` raises `OCU_E_BUSY`.

The MCU driver changes the input clock frequency of `OcuConf_OcuChannel_MY_CHANNEL`:

```
Ocu_StopChannel(OcuConf_OcuChannel_MY_CHANNEL);
Ocu_SetPrescaler(OcuConf_OcuChannel_MY_CHANNEL, MY_CLOCK_FREQUENCY);
Ocu_StartChannel(OcuConf_OcuChannel_MY_CHANNEL);
```

Note: *`Ocu_SetPrescaler()` calculates the prescaler value based on the value of the input clock frequency and tick frequency.*
Calculating formula: `MY_CLOCK_FREQUENCY` divided by tick frequency (Round down decimals).
The following two cases cause a poor accuracy of tick duration:
1. The calculation result of input clock frequency divided by tick frequency does not meet the prescaler value (1, 2, 4, 8, 16, 32, 64, 128).
2. If the input clock frequency is close to the tick frequency, the appropriate prescaler value may not be set.

In these cases, the frequency of the input clock should be adjusted by the MCU driver to meet the prescaler value (1, 2, 4, 8, 16, 32, 64, 128).

5.10 Retrieving status of a channel

The channel status and driver status can be assessed by calling `Ocu_CheckChannelStatus()`. The internal register values are read out and correspondence of software and hardware status is determined.

5.11 Sleep mode

The OCU module does not have a Sleep mode; therefore, the OCU channel should be stopped before MCU enters Sleep mode.

Note: All TCPWM counters must be stopped before entering DeepSleep mode with Ocu_DeInit().

5.12 API parameter checking

The driver's services perform regular error checks.

When an error occurs, the error hook routine (configured via `OcuErrorCalloutFunction`) is called and the error code, service ID, module ID, and instance ID are passed as parameters.

If default error detection is enabled, all errors are reported to DET, a central error hook function within the AUTOSAR environment. The checking itself cannot be deactivated for safety reasons.

The following development error checks are performed by the services of the OCU driver:

- If one of the functions `Ocu_DeInit`, `Ocu_StartChannel`, `Ocu_StopChannel`, `Ocu_SetPinState`, `Ocu_SetPinAction`, `Ocu_GetCounter`, `Ocu_SetAbsoluteThreshold`, `Ocu_SetRelativeThreshold`, `Ocu_DisableNotification`, `Ocu_EnableNotification`, `Ocu_SetPrescaler` or `Ocu_InterruptHandler` is called before `Ocu_Init` is called, the error code `OCU_E_UNINIT` is reported.
- If one of the functions `Ocu_DeInit`, `Ocu_StartChannel`, `Ocu_StopChannel`, `Ocu_SetPinState`, `Ocu_SetPinAction`, `Ocu_GetCounter`, `Ocu_SetAbsoluteThreshold`, `Ocu_SetRelativeThreshold`, `Ocu_DisableNotification`, `Ocu_EnableNotification`, `Ocu_SetPrescaler`, `Ocu_CheckChannelStatus`, or `Ocu_InterruptHandler` is called when the current configuration set is invalid, the error code `OCU_E_UNINIT` is reported.
- If one of the functions `Ocu_StartChannel`, `Ocu_StopChannel`, `Ocu_SetPinState`, `Ocu_SetPinAction`, `Ocu_GetCounter`, `Ocu_SetAbsoluteThreshold`, `Ocu_SetRelativeThreshold`, `Ocu_DisableNotification`, `Ocu_EnableNotification`, `Ocu_CheckChannelStatus`, or `Ocu_SetPrescaler` is called with an invalid parameter `ChannelNumber`, the error code `OCU_E_PARAM_INVALID_CHANNEL` is reported.
- If the function `Ocu_Init` is called with an invalid parameter `ConfigPtr`, the error code `OCU_E_INIT_FAILED` is reported.
- If the function `Ocu_Init` is called when `Ocu_Init` was already called, the error code `OCU_E_ALREADY_INITIALIZED` is reported.
- If the function `Ocu_DeInit` is called when any of the channel status is still running, the error code `OCU_E_PARAM_INVALID_STATE` is reported.
- If one of the functions `Ocu_StartChannel` or `Ocu_SetPrescaler` is called when the current channel status is running, the error code `OCU_E_BUSY` is reported.
- If one of the functions `Ocu_SetPinState` or `Ocu_SetPinAction` is called when a pin is not associated with the channel, the error code `OCU_E_PARAM_NO_PIN` is reported.

- If the function `Ocu_SetPinState` is called with an invalid parameter `PinState` or the channel status is running, the error code `OCU_E_PARAM_INVALID_STATE` is reported.
- If the function `Ocu_SetPinAction` is called with an invalid parameter `PinAction`, the error code `OCU_E_PARAM_INVALID_ACTION` is reported.
- If the function `Ocu_SetPrescaler` is called with an invalid parameter `ClockFrequency`, the error code `OCU_E_PARAM_INVALID_CLOCK` is reported.
- If one of the functions `Ocu_GetVersionInfo` or `Ocu_CheckChannelStatus` is called with NULL pointer, the error code `OCU_E_PARAM_POINTER` is reported.
- If the function `Ocu_SetAbsoluteThreshold` is called with an invalid parameter `ReferenceValue`, the error code `OCU_E_PARAM_REFERENCEVALUE` is reported.
- If the function `Ocu_SetAbsoluteThreshold` is called with an invalid parameter `AbsoluteValue`, the error code `OCU_E_PARAM_ABSOLUTETHRESHOLD` is reported.
- If the function `Ocu_SetRelativeThreshold` is called with an invalid parameter `RelativeValue`, the error code `OCU_E_PARAM_RELATIVETHRESHOLD` is reported.
- If the function `Ocu_EnableNotification` or `Ocu_DisableNotification` is called when notification function pointer was NULL, the error code `OCU_E_NO_VALID_NOTIF` is reported.
- If one of the functions `Ocu_StartChannel`, `Ocu_SetPinState`, `Ocu_SetAbsoluteThreshold`, `Ocu_SetRelativeThreshold` or `Ocu_SetPrescaler` is called when the counter is not running due to not receiving the trigger signal, the error code `OCU_E_WAITING_TRIGGER` is reported.

5.13 Configuration checking

`OcuChannelId` defines the assignment of the channel to the physical OCU hardware channel. `OcuChannelId` must be unique in the same configuration set.

5.14 Reentrancy

The following functions are reentrant if called on different channel numbers:

- `Ocu_StartChannel()`
- `Ocu_StopChannel()`
- `Ocu_SetPinState()`
- `Ocu_SetPinAction()`
- `Ocu_SetAbsoluteThreshold()`
- `Ocu_SetRelativeThreshold()`
- `Ocu_DisableNotification()`
- `Ocu_EnableNotification()`
- `Ocu_CheckChannelStatus()`
- `Ocu_SetPrescaler()`

They are non-reentrant if called on the same channel number. The functions may be accessed by different tasks or interrupts simultaneously as long as the tasks or interrupts access disjunct sets of channels.
`Ocu_GetCounter` is reentrant even if called on the same channel number. `Ocu_GetVersionInfo` is reentrant. `Ocu_Init` and `Ocu_DeInit` are not reentrants.

5.15 Debugging support

The OCU driver does not support debugging.

5.16 Execution time dependencies

The execution time of certain API functions is dependent on certain factors. See Table 1 for details.

Table 1 Execution time dependencies

Affected function	Dependency
`Ocu_Init()` `Ocu_DeInit()`	Runtime depends on the number of configured channels because all configured channels are processed in these functions.

6 Hardware resources

6.1 Ports and pins

To use the OCU driver, you should configure the pins within the PORT driver first.

The physical hardware timer that need to be configured for OCU channel.

- OcuTimer = TCPWM_0_n or TCPWM_1_n

The PORT driver must be configured with the appropriate Port pin:

- PortPinName = Pxxx_y for OCU pin
- PortPinDirection = PORT_PIN_OUT
- PortPinInitialMode = TCPWM_0_LINE_n or TCPWM_1_LINE_n
- PortPinMode = TCPWM_0_LINE_n or TCPWM_1_LINE_n (same as PortPinInitialMode)

The associated Port pin for each OCU channel is subderivative dependent and see the hardware manual.

6.2 Timer

For each configured OCU channel, one hardware timer of the TRAVEO™ T2G family is reserved exclusively. This is done via configuration parameter `OcuTimer`.

6.3 Interrupts

The OCU driver uses the interrupts associated with the configured hardware resource. The ISR must be declared in the AUTOSAR OS as Category 1 Interrupt or Category 2 Interrupt.

Note: IRQ numbers depend on subderivative. See the hardware manual.

You can define the ISR. The ISR-Name of each OCU channel is specified as:

`Ocu_Isr_Vector_<IRQ No>_Cat1` for Category-1 ISR

`Ocu_Isr_Vector_<IRQ No>_Cat2` for Category-2 ISR

Note: `Ocu_Isr_Vector_<IRQ No>_Cat2` must be called from the (OS) interrupt service routine. In the case of Category-1 usage, the address of `Ocu_Isr_Vector_<IRQ No>_Cat1` must be the entry in the (OS) interrupt vector table.

Example: Category-1 ISR for the configured OCU Channel 0 located in the generated file *src/Ocu_Irq.c*:

```
ISR_NATIVE(Ocu_Isr_Vector_276_Cat1)
{
...
}
```

Example: Category-2 ISR for the configured OCU Channel 0 located in the generated file *src/Ocu_Irq.c*:

```
ISR(Ocu_Isr_Vector_276_Cat2)
{
...
```

```
}
```

Note: *On the Arm® Cortex®-M4 CPU, priority inversion of interrupts may occur under specific timing conditions in the integrated system with TRAVEO™ T2G MCAL. For more details, see the following errata notice.*

Arm® Cortex®-M4 Software Developers Errata Notice - 838869:
"Store immediate overlapping exception return operation might vector to incorrect interrupt"

If the user application cannot tolerate the priority inversion, a DSB instruction should be added at the end of the interrupt function to avoid the priority inversion.

TRAVEO™ T2G MCAL interrupts are handled by an ISR wrapper (handler) in the integrated system. Thus, if necessary, the DSB instruction should be added just before the end of the handler by the integrator.

6.4 Triggers

In general, a trigger input signal indicates the completion of a peripheral action or a peripheral event. A trigger output signal initiates a peripheral action. There are two kinds of trigger groups; Trigger (Multiplexer-based trigger) group and trigger 1-to-1 group. Trigger group is a multiplexer-based connectivity group. This type connects a peripheral input trigger to multiple peripheral output triggers. Trigger 1-to-1 group is a 1-to-1-based connectivity group. This type connects a peripheral input trigger to one specific peripheral output trigger.

For more detail about triggers, see hardware documentation.

OCU driver uses input/output trigger signal in the following cases:

- Triggering peripheral action

  OCU channel generates output trigger signal to peripherals. See 5.8 Output trigger to peripherals.

- Counters synchronous start

  A trigger multiplexer to TCPWM channels should be configured in PORT driver.

  PortOutputTrigger configuration setting:

  - PortTrOutputName: TCPWM_0_TR_ALL_CNT_IN_n or TCPWM_1_TR_ALL_CNT_IN_n
  - PortTrInputName: Selected input trigger (e.g. CPUSS_ZERO).
  - PortTrInvertEnable: Disable
  - PortTrSensitiveType: PORT_TR_SENSITIVE_EDGE

- Debug capability to stop counter channels

  A trigger multiplexer from CTI (Cross triggering interface) to TCPWM should be configured in PORT driver.

  PortOutputTrigger configuration setting:

  - PortTrOutputName: TCPWM_0_TR_DEBUG_FREEZE or TCPWM_1_TR_DEBUG_FREEZE
  - PortTrInputName: Select CTI signal
  - PortTrInvertEnable: Disable
  - PortTrSensitiveType: PORT_TR_SENSITIVE_LEVEL

7 Appendix A – API reference

7.1 Include files

The *Ocu.h* file includes the necessary external identifiers. Thus, the application only needs to include *Ocu.h* to make all API functions and data types available.

7.2 Data types

7.2.1 Ocu_ChannelType

Type

`uint16`

Description

Numeric identifier of an OCU channel

7.2.2 Ocu_ClkFrequencyType

Type

`uint32`

Description

Type for frequency of a counter input clock (Hz)

7.2.3 Ocu_ValueType

Type

`uint32`

Description

Type for reading the counter and writing the threshold values (in number of ticks). On the TRAVEO™ T2G microcontroller, the counter channels are 16-bit wide and 32-bit wide; therefore, the `Ocu_ValueType` is 32-bit wide.

7.2.4 Ocu_ChannelStateType

Type

```
typedef enum
{
  OCU_CH_UNINITIALIZED = 0,
  OCU_CH_STOPPED,
  OCU_CH_RUNNING
} Ocu_ChannelStateType;
```

Description

States of an OCU channel.

7.2.5 Ocu_DriverStatusType

Type

```
typedef enum
{
  OCU_S_UNINITIALIZED = 0,
  OCU_S_INITIALIZED
} Ocu_DriverStatusType;
```

Description

Driver states of the OCU module.

7.2.6 Ocu_ReturnType

Type

```
typedef enum
{
  OCU_CM_IN_REF_INTERVAL = 0,
  OCU_CM_OUT_REF_INTERVAL
} Ocu_ReturnType;
```

Description

Return information after setting a new threshold value.

7.2.7 Ocu_PinStateType

Type

```
typedef enum
{
  OCU_HIGH = 0,
  OCU_LOW = 1,
  OCU_PIN_STATE_INVALID_VALUE = 3
} Ocu_PinStateType;
```

Description

Output state of the pin linked to an OCU channel

7.2.8 Ocu_PinActionType

Type

```
typedef enum
{
  OCU_SET_HIGH = 0,
  OCU_SET_LOW,
  OCU_TOGGLE,
  OCU_DISABLE
} Ocu_PinActionType;
```

Description

Automatic action (by hardware) to be performed on a pin attached to an OCU channel.

7.2.9 Ocu_ChannelStatusType

Type

```
typedef struct
{
  Ocu_ValueType SettingThreshold;
  Ocu_PinStateType PinState;
  Ocu_PinActionType PinAction;
  Ocu_ChannelStateType ChannelState;
  Ocu_ChannelType ChannelId;
  uint8 Prescale;
  boolean NotificationEnabled;
} Ocu_ChannelStatusType;
```

Description

Status of an OCU channel.

7.2.10 Ocu_ConfigType

Type

```
typedef struct
```

Description

This is the type of the data structure containing the initialization data for the OCU driver.

7.3 Constants

7.3.1 Error codes

The service may return one of the error codes, listed in Table 2, if the default error detection is enabled.

Table 2 Error codes

Name	Value	Description
OCU_E_UNINIT	0x02	API services other than Ocu_GetVersionInfo() and Ocu_Init() are used without module initialization.
OCU_E_PARAM_INVALID_CHANNEL	0x03	API service is used with an invalid channel identifier.
OCU_E_PARAM_INVALID_STATE	0x04	API Ocu_SetPinState() called with an invalid pin state or when the channel is in the Running state.
OCU_E_PARAM_INVALID_ACTION	0x05	API Ocu_SetPinAction() called with an invalid pin action.
OCU_E_NO_VALID_NOTIF	0x06	Usage of Ocu_DisableNotification() or Ocu_EnableNotification() on a channel where a NULL pointer is configured as the notification function.
OCU_E_ALREADY_INITIALIZED	0x07	API Ocu_Init() is called while the OCU driver has already been initialized.
OCU_E_PARAM_POINTER	0x08	APIs Ocu_GetVersionInfo() or Ocu_CheckChannelStatus() is called with a NULL parameter.
OCU_E_BUSY	0x09	API Ocu_StartChannel() or Ocu_SetPrescaler() is called on a channel that is in state Running.
OCU_E_PARAM_NO_PIN	0x0A	Ocu_SetPinState() or Ocu_SetPinAction() called for a channel that does not have an associated output pin.
OCU_E_INIT_FAILED	0x0B	OCU initialization has been failed (e.g. selected configuration set does not exist).

7.3.2 Vendor specific error codes

In addition to the error codes listed in Table 2 OCU driver defines the errors listed in Table 3.

Table 3 Vendor specific error codes

Name	Value	Description
OCU_E_PARAM_INVALID_CLOCK	0xF0	API Ocu_SetPrescaler() is called with an invalid clock frequency.
OCU_E_PARAM_REFERENCEVALUE	0xF1	API Ocu_SetAbsoluteThreshold() is called with an invalid ReferenceValue.
OCU_E_PARAM_ABSOLUTETHRESHOLD	0xF2	API Ocu_SetAbsoluteThreshold() is called with an invalid AbsoluteValue.
OCU_E_PARAM_RELATIVETHRESHOLD	0xF3	API Ocu_SetRelativeThreshold() is called with an invalid RelativeValue.

Name	Value	Description
OCU_E_WAITING_TRIGGER	0xF4	API Ocu_StartChannel(),Ocu_SetPinState(), Ocu_SetAbsoluteThreshold(), Ocu_SetRelativeThreshold() or Ocu_SetPrescaler() is called when the counter is not running due to not receiving the trigger signal.

7.3.3 Version information

The version information listed in Table 4 is published in the driver's header file:

Table 4 Version information

Name	Value	Description
OCU_AR_RELEASE_MAJOR_VERSION	4	AUTOSAR specification major version
OCU_AR_RELEASE_MINOR_VERSION	2	AUTOSAR specification minor version
OCU_AR_RELEASE_REVISION_VERSION	2	AUTOSAR specification patch version
OCU_SW_MAJOR_VERSION	See release notes	Driver implementation major version
OCU_SW_MINOR_VERSION	See release notes	Driver implementation minor version
OCU_SW_PATCH_VERSION	See release notes	Driver implementation patch version

7.3.4 Module information

Table 5 Module information

Name	Value	Description
OCU_MODULE_ID	125	Module ID
OCU_VENDOR_ID	66	Vendor ID

7.3.5 API service IDs

Table 6 API service IDs

Name	Value	API service
OCU_API_SERVICE_INIT	0x00	Ocu_Init
OCU_API_SERVICE_DEINIT	0x01	Ocu_DeInit
OCU_API_SERVICE_START_CHANNEL	0x02	Ocu_StartChannel
OCU_API_SERVICE_STOP_CHANNEL	0x03	Ocu_StopChannel
OCU_API_SERVICE_SET_PIN_STATE	0x04	Ocu_SetPinState
OCU_API_SERVICE_SET_PIN_ACTION	0x05	Ocu_SetPinAction
OCU_API_SERVICE_GET_COUNTER	0x06	Ocu_GetCounter
OCU_API_SERVICE_SET_ABSOLUTE_THRESHOLD	0x07	Ocu_SetAbsoluteThreshold
OCU_API_SERVICE_SET_RELATIVE_THRESHOLD	0x08	Ocu_SetRelativeThreshold
OCU_API_SERVICE_GET_VERSION_INFO	0x09	Ocu_GetVersionInfo
OCU_API_SERVICE_DISABLE_NOTIFICATION	0x0A	Ocu_DisableNotification

Name	Value	API service
OCU_API_SERVICE_ENABLE_NOTIFICATION	0x0B	Ocu_EnableNotification
OCU_API_SERVICE_INTERRUPT_HANDLER	0xFC	Ocu_InterruptHandler
OCU_API_SERVICE_CHECK_CHANNEL_STATUS	0xFD	Ocu_CheckChannelStatus
OCU_API_SERVICE_SET_PRESCALER	0xFE	Ocu_SetPrescaler

7.3.6 Symbolic names

Table 7 Symbolic names

Name	Description
OcuConf_OcuChannel_<n>	Symbolic name for OCU channel *n* (n is channel's short name).
OcuConf_OcuConfigSet_<n>	Symbolic name for OCU configuration setting *n* (n is configuration set's short name).

7.4 Functions

7.4.1 Ocu_Init

Syntax

```
void Ocu_Init(
(
   const Ocu_ConfigType* ConfigPtr
)
```

Service ID

0x00

Parameters (in)

- `ConfigPtr` - Pointer to the configuration set.

Parameters (out)

None

Return value

None

DET errors

- `OCU_E_ALREADY_INITIALIZED` - The `Ocu_Init()` API is called while the OCU driver has already been initialized.
- `OCU_E_INIT_FAILED` - OCU initialization has failed (e.g. Selected configuration set does not exist).

DEM errors

None

Description

Service for OCU initialization. When the service returns, all channels are in the Stopped state with free-running counter started, notification disabled, and default pin state. This function will be called with a pointer to a selected configuration structure.

7.4.2 Ocu_DeInit

Syntax

```
void Ocu_DeInit
(
   void
)
```

Service ID

0x01

Parameters (in)

None

Parameters (out)

None

Return value

None

DET errors

- `OCU_E_UNINIT` - Driver is not yet initialized.
- `OCU_E_PARAM_INVALID_STATE` - The channel is in the Running state.

DEM errors

None

Description

Service for OCU de-initialization. This function disables all OCU interrupts and notifications and stops all free-running counters.

7.4.3 Ocu_StartChannel

Syntax

```
void Ocu_StartChannel
(
   Ocu_ChannelType ChannelNumber
)
```

Service ID

0x02

Parameters (in)

- `ChannelNumber` - Numeric identifier of the OCU channel.

Parameters (out)

None

Return value

None

DET errors

- `OCU_E_UNINIT` - Driver is not yet initialized.
- `OCU_E_PARAM_INVALID_CHANNEL` - `ChannelNumber` is invalid.
- `OCU_E_BUSY` - The channel is in the Running state.
- `OCU_E_WAITING_TRIGGER` - The counter is not running due to not receiving the trigger signal.

DEM errors

None

Description

Service to start an OCU channel.

7.4.4 Ocu_StopChannel

Syntax

```
void Ocu_StopChannel
(
   Ocu_ChannelType ChannelNumber
)
```

Service ID

0x03

Parameters (in)

- `ChannelNumber` - Numeric identifier of the OCU channel.

Parameters (out)

None

Return value

None

DET errors

- `OCU_E_UNINIT` - Driver is not yet initialized.
- `OCU_E_PARAM_INVALID_CHANNEL` - `ChannelNumber` is invalid.

DEM errors

None

Description

Service to stop an OCU channel.

7.4.5 Ocu_SetPinState

Syntax

```
void Ocu_SetPinState
(
   Ocu_ChannelType ChannelNumber,
   Ocu_PinStateType PinState
)
```

Service ID

0x04

Parameters (in)

- `ChannelNumber` - Numeric identifier of the OCU channel.
- `PinState` - Type of pin state: OCU_LOW or OCU_HIGH.

Parameters (out)

None

Return value

None

DET errors

- `OCU_E_UNINIT` - Driver is not yet initialized.
- `OCU_E_PARAM_INVALID_CHANNEL` - `ChannelNumber` is invalid.
- `OCU_E_PARAM_NO_PIN` - A channel that does not have an associated output pin.
- `OCU_E_PARAM_INVALID_STATE` - The channel is in the Running state or `PinState` is invalid.
- `OCU_E_WAITING_TRIGGER` - The counter is not running due to not receiving the trigger signal.

DEM errors

None

Description

Service to set the level of the pin associated to an OCU channel. This API stops the counter once to change the pin state; then the counter is reset and restarted. Because in the hardware feature, the register (e.g. `CTRL.MODE`, all `TR_IN_SELx`, `TR_IN_EDGE_SEL`, `TR_PWM_CTRL` and `TR_OUT_SEL` register fields) can only be set when the counter is not running.

7.4.6 Ocu_SetPinAction

Syntax

```
void Ocu_SetPinAction
(
   Ocu_ChannelType ChannelNumber,
   Ocu_PinActionType PinAction
)
```

Service ID

0x05

Parameters (in)

- `ChannelNumber` - Numeric identifier of the OCU channel.
- `PinAction` - Type of pin action: OCU_SET_LOW, OCU_SET_HIGH, OCU_TOGGLE or OCU_DISABLE.

Parameters (out)

None

Return value

None

DET errors

- `OCU_E_UNINIT` - Driver is not yet initialized.
- `OCU_E_PARAM_INVALID_CHANNEL` - `ChannelNumber` is invalid.
- `OCU_E_PARAM_INVALID_ACTION` - `PinAction` is invalid.
- `OCU_E_PARAM_NO_PIN` - A channel that does not have an associated output pin.

DEM errors

None

Description

Service to indicate the driver the actions that will be done automatically by hardware upon compare match. This API stops the counter once. Because in the hardware feature, the register (e.g. `CTRL.MODE`, all `TR_IN_SELx`, `TR_IN_EDGE_SEL`, `TR_PWM_CTRL` and `TR_OUT_SEL` register fields) can only be set when the counter is not running.

7.4.7 Ocu_GetCounter

Syntax

```
Ocu_ValueType Ocu_GetCounter
(
   Ocu_ChannelType ChannelNumber
)
```

Service ID

0x06

Parameters (in)

- `ChannelNumber` - Numeric identifier of the OCU channel.

Parameters (out)

None

Return value

- `Ocu_ValueType` - Content of the counter in ticks.

DET errors

- `OCU_E_UNINIT` - Driver is not yet initialized.
- `OCU_E_PARAM_INVALID_CHANNEL` - `ChannelNumber` is invalid.

DEM errors

None

Description

Service to read the current value of the counter.

7.4.8 Ocu_SetAbsoluteThreshold

Syntax

```
Ocu_ReturnType Ocu_SetAbsoluteThreshold
(
   Ocu_ChannelType ChannelNumber,
   Ocu_ValueType ReferenceValue,
   Ocu_ValueType AbsoluteValue
)
```

Service ID

0x07

Parameters (in)

- `ChannelNumber` - Numeric identifier of the OCU channel.
- `ReferenceValue` - Value given by the upper layer and used as a base to determine whether to call the notification before the function exits.
- `AbsoluteValue` - Value to compare with the content of the counter. This value is in ticks.

Parameters (out)

None

Return value

- `OCU_CM_OUT_REF_INTERVAL` - The compare match will not occur inside the current reference interval.

- `OCU_CM_IN_REF_INTERVAL` - The compare match will occur inside the current reference interval.

DET errors

- `OCU_E_UNINIT` - Driver is not yet initialized.
- `OCU_E_PARAM_INVALID_CHANNEL` - `ChannelNumber` is invalid.
- `OCU_E_PARAM_REFERENCEVALUE` - `ReferenceValue` is invalid.
- `OCU_E_PARAM_ABSOLUTETHRESHOLD` - `AbsoluteValue` is invalid.
- `OCU_E_WAITING_TRIGGER` - The counter is not running due to not receiving the trigger signal.

DEM errors

None

Description

Service to set the value of the channel threshold using an absolute input data. If this API is called when the channel is not in the running state, `OCU_CM_OUT_REF_INTERVAL` will always be returned.

7.4.9 Ocu_SetRelativeThreshold

Syntax

```
Ocu_ReturnType Ocu_SetRelativeThreshold
(
   Ocu_ChannelType ChannelNumber,
   Ocu_ValueType RelativeValue
)
```

Service ID

0x08

Parameters (in)

- `ChannelNumber` - Numeric identifier of the OCU channel.
- `RelativeValue` - Value to use for computing the new threshold.

Parameters (out)

None

Return value

- `OCU_CM_OUT_REF_INTERVAL` - The compare match will not occur inside the current reference interval.
- `OCU_CM_IN_REF_INTERVAL` - The compare match will occur inside the current reference interval.

DET errors

- `OCU_E_UNINIT` - Driver is not yet initialized.
- `OCU_E_PARAM_INVALID_CHANNEL` - `ChannelNumber` is invalid.
- `OCU_E_PARAM_RELATIVETHRESHOLD` - `RelativeValue` is invalid.
- `OCU_E_WAITING_TRIGGER` - The counter is not running due to not receiving the trigger signal.

DEM errors

None

Description

Service to set the value of the channel threshold relative to the current value of the counter. If this API is called when the channel is not in the running state, `OCU_CM_OUT_REF_INTERVAL` will always be returned.

The new threshold value is computed according to the following formula. The fomula is different depends on `OcuCountDirection`.

For `OcuCountDirection` is `OCU_UPCOUTING`: New threshold value = current counter value + `RelativeValue`

For `OcuCountDirection` is `OCU_DOWNCOUTING`: New threshold value = current counter value – `RelativeValue`

7.4.10 Ocu_GetVersionInfo

Syntax

```
void Ocu_GetVersionInfo
(
   Std_VersionInfoType* versioninfo
)
```

Service ID

0x09

Parameters (in)

None

Parameters (out)

- `versioninfo` - Pointer to where to store the version information of this module.

Return value

None

DET errors

- `OCU_E_PARAM_POINTER` - `versioninfo` is NULL pointer.

DEM errors

None

Description

Service to return the version information of this module.

7.4.11 Ocu_DisableNotification

Syntax

```
void Ocu_DisableNotification
(
   Ocu_ChannelType ChannelNumber
)
```

Service ID

0x0A

Parameters (in)

- `ChannelNumber` - Numeric identifier of the OCU channel.

Parameters (out)

None

Return value

None

DET errors

- `OCU_E_UNINIT` - Driver is not yet initialized.
- `OCU_E_PARAM_INVALID_CHANNEL` - `ChannelNumber` is invalid.
- `OCU_E_NO_VALID_NOTIF` - A NULL pointer is configured on a channel.

DEM errors

None

Description

Service to disable notifications from an OCU channel.

7.4.12 Ocu_EnableNotification

Syntax

```
void Ocu_EnableNotification
(
   Ocu_ChannelType ChannelNumber
)
```

Service ID

0x0B

Parameters (in)

- `ChannelNumber` - Numeric identifier of the OCU channel.

Parameters (out)

None

Return value

None

DET errors

- `OCU_E_UNINIT` - Driver is not yet initialized.
- `OCU_E_PARAM_INVALID_CHANNEL` - `ChannelNumber` is invalid.
- `OCU_E_NO_VALID_NOTIF` - A NULL pointer is configured on a channel.

DEM errors

None

Description

Service to enable notifications from an OCU channel.

7.4.13 Ocu_CheckChannelStatus

Syntax

```
Std_ReturnType Ocu_CheckChannelStatus
(
   Ocu_DriverStatusType* DriverStatusPtr,
   Ocu_ChannelStatusType* ChannelStatusPtr,
   Ocu_ChannelType ChannelNumber
)
```

Service ID

0xFD

Parameters (in)

- `ChannelNumber` - Numeric identifier of the OCU channel.

Parameters (out)

- `DriverStatusPtr` - Pointer to where to store the driver status information.
- `ChannelStatusPtr` - Pointer to where to store the channel status information.

Return value

- `E_OK` - No error has been detected.
- `E_NOT_OK` - Aborted due to errors.

DET errors

- `OCU_E_PARAM_POINTER` - `DriverStatusPtr` or `ChannelStatusPtr` is NULL pointer.
- `OCU_E_UNINIT` - Driver is not yet initialized.
- `OCU_E_PARAM_INVALID_CHANNEL` - `ChannelNumber` is invalid.

DEM errors

None

Description

Service to check the current status of the channel and driver.

This is a vendor-specific function.

Note: *`Ocu_CheckChannelStatus()` may return `E_NOT_OK` if the hardware status has not yet changed to the running after `Ocu_SetPinState()`, `Ocu_SetPinAction()` and `Ocu_SetPrescaler()`. It may occur when the tick frequency of the OCU timer is very slow.*

7.4.14 Ocu_SetPrescaler

Syntax

```
void Ocu_SetPrescaler
(
   Ocu_ChannelType ChannelNumber,
   Ocu_ClkFrequencyType ClockFrequency
)
```

Service ID

0xFE

Parameters (in)

- `ChannelNumber` - Numeric identifier of the OCU channel.
- `ClockFrequency` - Input clock frequency.

Parameters (out)

None

Return value

None

DET errors

- `OCU_E_UNINIT` - Driver is not yet initialized.
- `OCU_E_PARAM_INVALID_CHANNEL` - `ChannelNumber` is invalid.
- `OCU_E_BUSY` - The channel is in the running state.
- `OCU_E_PARAM_INVALID_CLOCK` - `ClockFrequency` is 0 or wrong frequency.
- `OCU_E_WAITING_TRIGGER` - The counter is not running due to not receiving the trigger signal.

DEM errors

None

Description

Service to set prescaler value for the specified channel. This API stops the counter once. Because there is no specific API to stop counter and pre-scaling register is static and can only be set when counter is not running.

This is a vendor-specific function.

7.5 Required callback functions

7.5.1 DET

If default error detection is enabled, the OCU driver uses the following callback function provided by DET. If you do not use DET, you must implement this function within your application.

Det_ReportError

Syntax

```
Std_ReturnType Det_ReportError
(
    uint16 ModuleId,
    uint8 InstanceId,
    uint8 ApiId,
    uint8 ErrorId
)
```

Reentrancy

Reentrant

Parameters (in)

- `ModuleId` - Module ID of calling module.
- `InstanceId` - Instance ID of calling module.
- `ApiId` - ID of the API service that calls this function.
- `ErrorId` - ID of the detected development error.

Return value

Returns always `E_OK`.

Description

Service for reporting development errors.

7.5.2 DEM

The OCU driver does not report to DEM.

7.5.3 OCU notifications

The following notification functions are used by the OCU driver to inform other software modules about the change of the OCU output signal.

The callback notification names are statically configurable by the OCU driver configuration via the `OcuNotification` parameter.

The notifications are enabled by calling `Ocu_EnableNotification()`.

The notifications are stopped by calling `Ocu_DisableNotification()`.

Ocu_Notification_<#Channel>

Syntax

```
void Ocu_Notification_<#Channel>
(
   void
)
```

Parameters (in)

None

Parameters (out)

None

Return value

None

Description

`Ocu_Notification_<#Channel>` is called upon compare match.

7.5.4 Callout functions

7.5.4.1 Error callout API

The AUTOSAR OCU module requires an error callout handler. Each error is reported to this handler; error checking cannot be switched OFF. The name of the function to be called can be configured by `OcuErrorCalloutFunction` parameter.

Syntax

```
void Error_Handler_Name
(
    uint16 ModuleId,
    uint8 InstanceId,
    uint8 ApiId,
    uint8 ErrorId
)
```

Reentrancy

Reentrant

Parameters (in)

- `ModuleId` - Module ID of calling module.
- `InstanceId` - Instance ID of calling module.
- `ApiId` - ID of the API service that calls this function.
- `ErrorId` - ID of the detected error.

Return value

None

7 Appendix A – API reference

Description

Service for reporting errors.

8 Appendix B – Access register table

8.1 TCPWM

Register	Bit No.	Access size	Value	Description	Timing	Mask value	Monitoring value
CTRL	31:0	Word (32 bits)	Depends on configuration value or API.	Counter control register	Ocu_Init Ocu_DeInit Ocu_SetPinAction Ocu_SetPinState Ocu_SetPrescaler	0x47333600	0x*40**600 (After Ocu_Init. Digit * depends on configuration value.) 0x000000F0 (After Ocu_DeInit.) 0x*40**600 (From Ocu_Init to Ocu_DeInit, the value depends on configuration value or API.)
STATUS	31:0	Word (32 bits)	-	Counter status register	Read only.	0x00000000 (Monitoring is not needed.)	0x00000000 (Monitoring is not needed.)
COUNTER	31:0	Word (32 bits)	0x00000000.	Counter count register	Ocu_Init Ocu_DeInit	0x00000000 (Monitoring is not needed.)	0x00000000 (Monitoring is not needed.)
CC0	31:0	Word (32 bits)	0xFFFFFFFF, Compare threshold value.	Counter compare/capture 0 register	Ocu_Init Ocu_DeInit Ocu_StartChannel Ocu_StopChannel Ocu_SetAbsoluteThreshold Ocu_SetRelativeThreshold	0x00000000 (Monitoring is not needed.)	0x00000000 (Monitoring is not needed.)
CC0_BUFF	31:0	Word (32 bits)	-	Counter buffered compare/capture 0 register	Do not use.	0x00000000 (Monitoring is not needed.)	0x00000000 (Monitoring is not needed.)

Register	Bit No.	Access size	Value	Description	Timing	Mask value	Monitoring value
CC1	31:0	Word (32 bits)	0x00000000, 0xFFFFFFFF, Maximum counter value - 1.	Counter compare/capture 1 register	Ocu_Init Ocu_DeInit Ocu_StartChannel Ocu_SetPinState	0x00000000 (Monitoring is not needed.)	0x00000000 (Monitoring is not needed.)
CC1_BUFF	31:0	Word (32 bits)	-	Counter buffered compare/capture 1 register	Do not use.	0x00000000 (Monitoring is not needed.)	0x00000000 (Monitoring is not needed.)
PERIOD	31:0	Word (32 bits)	Maximum counter value.	Counter period register	Ocu_Init Ocu_DeInit	0x00000000 (Monitoring is not needed.)	0x00000000 (Monitoring is not needed.)
PERIOD_BUFF	31:0	Word (32 bits)	-	Counter buffered period register	Do not use.	0x00000000 (Monitoring is not needed.)	0x00000000 (Monitoring is not needed.)
LINE_SEL	31:0	Word (32 bits)	-	Counter line selection register	Do not use.	0x00000000 (Monitoring is not needed.)	0x00000000 (Monitoring is not needed.)
LINE_SEL_BUFF	31:0	Word (32 bits)	-	Counter buffered line selection register	Do not use.	0x00000000 (Monitoring is not needed.)	0x00000000 (Monitoring is not needed.)
DT	31:0	Word (32 bits)	Depends on configuration value or API.	Counter PWM dead time register	Ocu_Init Ocu_DeInit Ocu_SetPrescaler	0x00000000 (Monitoring is not needed.)	0x00000000 (Monitoring is not needed.)
TR_CMD	31:0	Word (32 bits)	Depends on configuration value or API.	Counter trigger command register	Ocu_Init Ocu_SetPinAction Ocu_SetPinState Ocu_SetPrescaler	0x00000000 (Monitoring is not needed.)	0x00000000 (Monitoring is not needed.)

Register	Bit No.	Access size	Value	Description	Timing	Mask value	Monitoring value
TR_IN_SEL0	31:0	Word (32 bits)	0x00000100 | Reload trigger select value << 16, 0x00000100.	Counter input trigger selection register 0	Ocu_Init Ocu_DeInit	0xFFFFFFFF	0x00**0000 (After Ocu_Init. Digit * depends on configuration value.) 0x00000100 (After Ocu_DeInit.)
TR_IN_SEL1	31:0	Word (32 bits)	-	Counter input trigger selection register 1	Do not use.	0x00000000 (Monitoring is not needed.)	0x00000000 (Monitoring is not needed.)
TR_IN_EDGE_SEL	31:0	Word (32 bits)	0x00000FCF, 0x00000FFF.	Counter input trigger edge selection register	Ocu_Init Ocu_DeInit	0x00000FFF	0x00000F*F (After Ocu_Init. Digit * depends on configuration value.) 0x00000FFF (After Ocu_DeInit.)
TR_PWM_CTRL	31:0	Word (32 bits)	Depends on configuration value or API.	Counter trigger PWM control register	Ocu_Init Ocu_DeInit Ocu_SetPinAction Ocu_SetPinState	0x000000FF	0x000000** (After Ocu_Init. Digit * depends on configuration value.) 0x000000FF (After Ocu_DeInit.) 0x000000** (From Ocu_Init to Ocu_DeInit, the value depends on configuration value or API.)

Register	Bit No.	Access size	Value	Description	Timing	Mask value	Monitoring value
TR_OUT_SEL	31:0	Word (32 bits)	0x00000032, 0x00000077, 0x00000073, 0x00000033, 0x00000037.	Counter output trigger selection register	Ocu_Init Ocu_DeInit	0x00000000 (Monitoring is not needed.)	0x00000000 (Monitoring is not needed.)
INTR	31:0	Word (32 bits)	0x00000007, 0x00000002.	Interrupt request register	Ocu_Init Ocu_DeInit Ocu_StartChannel Ocu_StopChannel Ocu_EnableNotification Ocu_DisableNotification Ocu_Isr_Vector_[IRQ Number]_Cat1 Ocu_Isr_Vector_[IRQ Number]_Cat2	0x00000000 (Monitoring is not needed.)	0x00000000 (Monitoring is not needed.)
INTR_SET	31:0	Word (32 bits)	-	Interrupt set request register	Do not use.	0x00000000 (Monitoring is not needed.)	0x00000000 (Monitoring is not needed.)
INTR_MASK	31:0	Word (32 bits)	0x00000000, 0x00000002.	Interrupt mask register	Ocu_Init Ocu_DeInit Ocu_StartChannel Ocu_StopChannel Ocu_EnableNotification Ocu_DisableNotification	0x00000000 (Monitoring is not needed.)	0x00000000 (Monitoring is not needed.)
INTR_MASKED	31:0	Word (32 bits)	-	Interrupt masked request register	Read only.	0x00000000 (Monitoring is not needed.)	0x00000000 (Monitoring is not needed.)

Revision history

Document revision	Date	Description of changes
**	2018-05-30	Initial release
*A	2018-12-13	Added new configuration parameter OcuStartTriggerSelect0 and OcuStartTriggerSelect1 in chapter “4.3.1 Container OcuGeneral”. Added description of OcuStartTriggerSelect<n> in chapter “2.3 Adapting Your Application” and “3.4.1 PORT Driver”. Added new error code OCU_E_WAITING_TRIGGER in chapter “A.1.3 Constants: Vendor Specific Error Codes”. Added description of OCU_E_WAITING_TRIGGER in chapter “5.12 API Parameter Checking” and “A.1.4 Functions”. Changed resource name of TCPWM and Trigger in chapter “4.3.2 Container OcuChannel: OcuTimer”, “6.1 Ports and Pins” and “6.4 Triggers”. Modify description of Ocu_SetPinState, Ocu_SetPinAction, Ocu_SetPrescaler, Ocu_SetAbsoluteThreshold and Ocu_SetRelativeThreshold in chapter “A.1.4 Functions”. Modify Register CTRL, CC0, CC1, TR_PWM_CTRL, TR_CMD, TR_OUT_SEL Timing in chapter “B.1 Access Register Table”. Updated OcuHardwareTriggeredAdc and OcuHardwareTriggeredDMA in chapter “4.2.3 Container OcuChannel”. Updated hardware documentation information. Updated remarks of configuration parameter OcuTimer in chapter “4.3.2 Container OcuChannel” Added new configuration parameter OcuHwTriggerOutputLine in chapter “4.3.2 Container OcuChannel” Changed “5.8 Output Trigger to Peripherals” and improved description in chapter 1.1, 2.3, 5.1, 5.6 and 6.4. Added note of Ocu_CheckChannelStatus in chapter “A.1.4 Functions”.
*B	2019-06-14	Updated hardware documentation information.
*C	2020-09-03	Changed a memmap file include folder in chapter 2.6. Updated Remarks of OcuTimer in chapter 4.3.2.
*D	2020-11-19	MOVED TO INFINEON TEMPLATE.
*E	2021-02-09	Added note in chapter 5.5. Changed the description in chapter 7.1.3.4.
*F	2021-08-18	Added a note on DeepSleep mode in 5.11 Sleep mode Added a note on Arm® errata in 6.3 Interrupts
*G	2021-12-22	Updated to the latest branding guidelines.
*H	2022-06-24	Changed the range and remarks of OcuDefaultThreshold in Section 4.2.3.4. Changed the range and added remarks of OcuMaxCounterValue in Section 4.2.3.7.

Revision history

Document revision	Date	Description of changes
		Updated the description of Ocu_SetPinState() in Section 7.4.5. Updated the description of Ocu_SetRelativeThreshold() in Section 7.4.9.
*I	2023-12-08	Web release. No content updates.
*J	2024-03-18	Deleted the description in chapter 6.1.

Trademarks
All referenced product or service names and trademarks are the property of their respective owners.

Edition 2024-03-18

Published by

Infineon Technologies AG

81726 Munich, Germany

Do you have a question about this document?

Email:

erratum@infineon.com

Document reference

002-23350 Rev. *J

Warnings
Due to technical requirements products may contain dangerous substances. For information on the types in question please contact your nearest Infineon Technologies office.

Except as otherwise explicitly approved by Infineon Technologies in a written document signed by authorized representatives of Infineon Technologies, Infineon Technologies' products may not be used in any applications where a failure of the product or any consequences of the use thereof can reasonably be expected to result in personal injury.