GPT 3.0 driver user guide

TRAVEO™ T2G family

About this document

Scope and purpose

This guide describes the architecture, configuration, and use of the general purpose timer (GPT) driver. This document explains the functionality of the driver and provides a reference for the driver's API.

The installation, build process, and general information on the use of the EB tresos Studio software are not within the scope of this document. See the *EB tresos Studio for ACG8 user's guide* [7].

Intended audience

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

Document structure

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

Chapter 2 Using the GPT driver details the steps on how to use the GPT driver in your application.

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

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

Chapter 5 Functional description gives a functional description of the services offered by the GPT 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	Description
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
Basic Software (also BSW)	Standardized part of software which does not fulfill a vehicle functional job.
Channel	Represents a functional timer instance; one compare register of hardware configuration.
CLK_EXT	External Clock
CLK_INT	Internal Clock
CTI	Cross Triggering Interface

About this document

Abbreviation	Description
DEM	Diagnostic Event Manager
DET	Default Error Tracer
EB tresos ECU AUTOSAR Suite	A collection of AUTOSAR Basic Software modules and a Runtime Environment integrated in a common configuration and build environment.
EB tresos Studio	Elektrobit Automotive configuration framework
EcuM	ECU State Manager
GPT	General Purpose Timer
GPT Predef Timer	A GPT Predef Timer is a free running up counter provided by the GPT driver. A GPT Predef Timer has predefined physical time unit and range.
IRQ	Interrupt Request
ISR	Interrupt Service Routine
MCAL	Microcontroller Abstraction Layer
MCU	Microcontroller Unit
OS	Operating System
Target time	Time at which the value is reached. The behavior depends on the configuration and the enabled functionality.
TCPWM	Timer Counter Pulse Width Modulation
Tick	Defines the timer resolution, the duration of a timer increment.
Timer channel	Represents a logical timer entity assigned to a timer hardware.
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 GPT 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.

[6] Specification of memory mapping, AUTOSAR release 4.2.2.

Elektrobit automotive documentation

Bibliography

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

About this document

Hardware documentation

Bibliography

The hardware documents are listed in the delivery notes.

Related standards and norms

Bibliography

[8] Layered software architecture, AUTOSAR release 4.2.2.

Table of contents

Table of contents

Table of contents

1 General overview

1.1 Introduction to GPT driver

The general purpose timer (GPT) driver provides services for starting and stopping a channel in the hardware timer module of the TRAVEO™ T2G family microcontroller. It supports the individual target time (one-shot mode) and repeating target time (continuous mode).

If configured and enabled, a notification function will be called as soon as the requested target time has elapsed. Notifications can be enabled and disabled at runtime.

The driver also has a lower-power Sleep mode in which a configured subset of the channels operates. The driver can notify the ECU state manager (EcuM) module when a timer channel causes the microcontroller to leave the sleep state.

The driver provides services for the following:

- Driver initialization and de-initialization
- Starting and stopping a GPT channel
- Getting the elapsed timer value
- Getting the timer value remaining until the next target time will be reached
- Controlling time triggered interrupt notifications
- Controlling time triggered wakeup interrupts

The driver also supports GPT Predef timers. These timers have predefined tick durations and predefined number of bits.

The driver is responsible for initializing the hardware resources that it uses exclusively. Resources that can be shared among several drivers must be initialized externally.

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

A plugin for the EB tresos Studio software is provided. The plugin creates the data tables and other program elements needed by the driver from a configuration file provided by the user.

1.2 User profile

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

- Embedded systems
- C programming language
- AUTOSAR standard
- Target hardware architecture

1.3 Embedding in the AUTOSAR environment

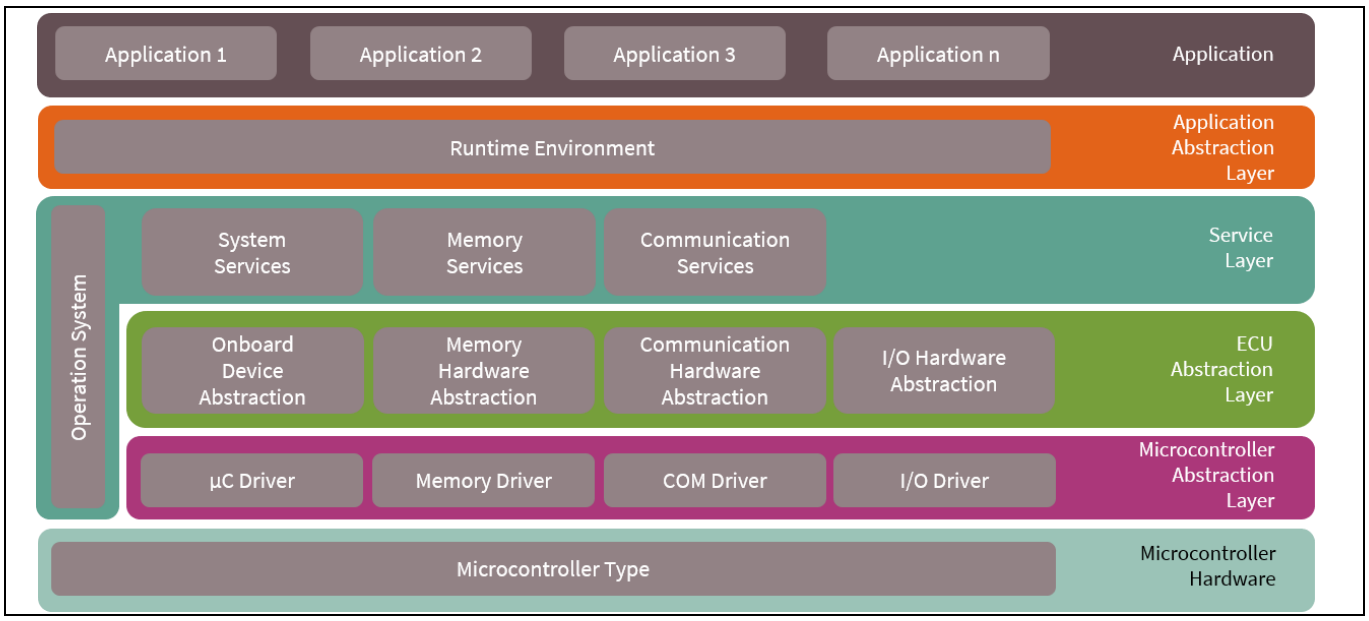

Figure 1 Overview of AUTOSAR software layers

Figure 1 shows the layered AUTOSAR software architecture. The GPT driver (Figure 2) is part of the microcontroller abstraction layer (MCAL), the lowest layer of basic software in the AUTOSAR environment.

As an internal microcontroller driver, GPT driver provides a standardized and microcontroller-independent interface to higher software layers for accessing general-purpose timer channels of the ECU hardware.

For an overview of the AUTOSAR layered software architecture, see *layered software architecture* [8].

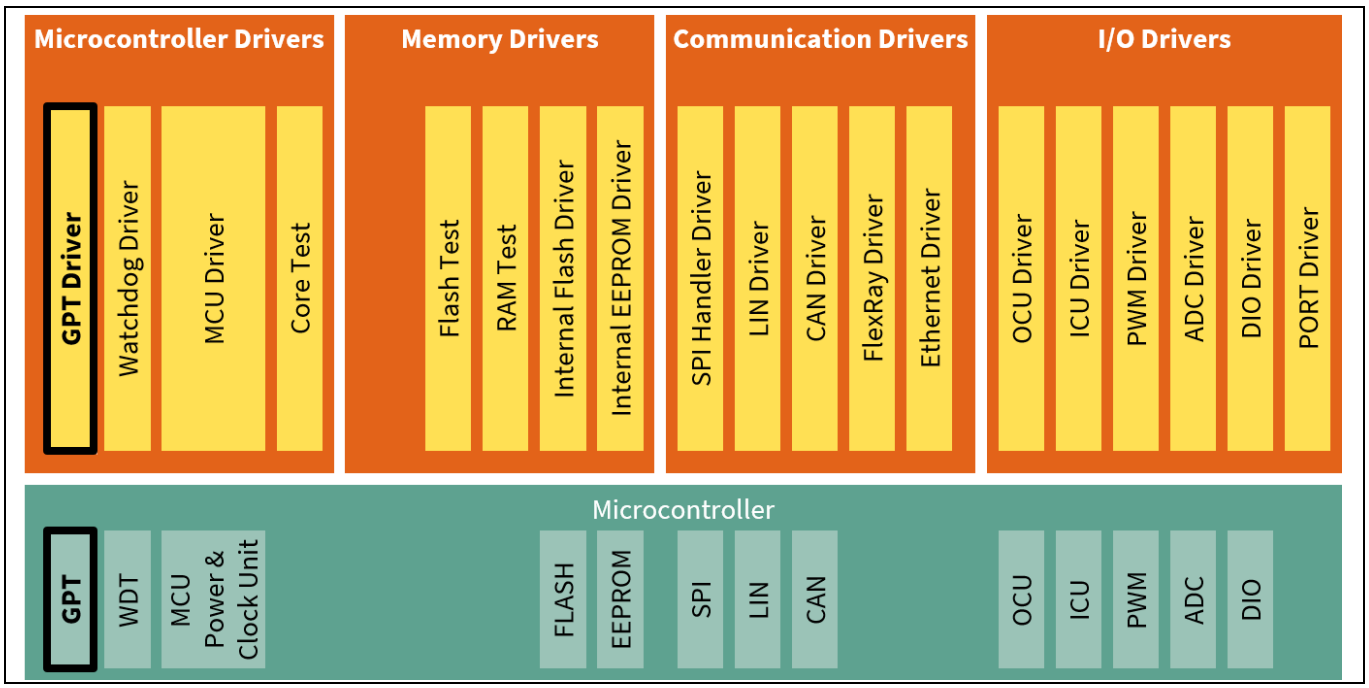

Figure 2 GPT driver in MCAL layer

1.4 Supported hardware

This version of the GPT driver supports TRAVEO™ T2G family microcontroller. No further special external hardware devices are required. The supported derivatives are listed in the release notes.

The GPT driver supports the TRAVEO™ T2G family microcontroller's timer counter pulse width modulation (TCPWM).

The 16-bit TCPWM includes 16-bit timer channels using a peripheral clock as clock source. The maximal countable interval is 2^16 - 1 = 0xFFFF.

The 32-bit TCPWM includes 32-bit timer channels using a peripheral clock as clock source. The maximal countable interval is 2^32 - 1 = 0xFFFFFFFF.

The TCPWM is an up counter with automatic counter reset in continuous mode. The TC event interrupt is used to reset the next interval.

The TCPWM is a global resource that can be used by multiple GPT channels and other modules, so the selection of the source clock rate is outside the scope of the GPT driver.

1.5 Development environment

The development environment corresponds to AUTOSAR release 4.2.2 [2]. The Base, Make, MCU, Port, and Resource modules are required for the proper functionality of the GPT driver.

1.6 Character set and encoding

All source code files of the GPT 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.

1.7 Multicore support

The GPT driver supports multicore type II. The multicore type III can also be supported for some APIs (for example, read-only API or atomic-write API). For each multicore type, see the following sections.

Note: If multicore type III is required, the section including the data related to the read-only API or atomic write API must be allocated to the memory, and can be read from any cores.

1.7.1 Multicore type

In this section, type I, type II, and type III are defined as multicore characteristics.

1.7.1.1 Single core only (multicore type I)

For this multicore type, the driver available on a single core only. This type is referred as “Multicore Type I”.

Multicore type I has the following characteristics:

- The peripheral channels are accessed by one core.

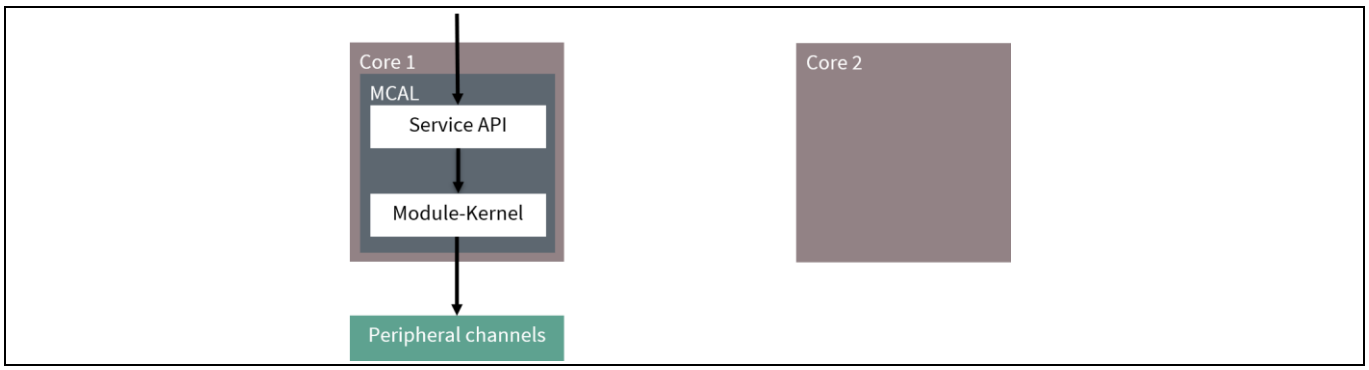

Figure 3 Overview of multicore type I

1.7.1.2 Core-dependent instances (multicore type II)

For this multicore type, the driver has core-dependent instances with individually allocable hardware. This type is referred as “Multicore Type II”.

Multicore type II has the following characteristics:

- The driver code is shared among all cores:
  - A common binary is used for all cores.
  - A configuration is common for all cores.
- Each core runs an instance of the driver.
- Peripheral channels and their data can be individually allocable to cores but cannot be shared among cores.
- One core will be the master; and the master core must be initialized first.
  - Cores other than the master core are called satellite cores.

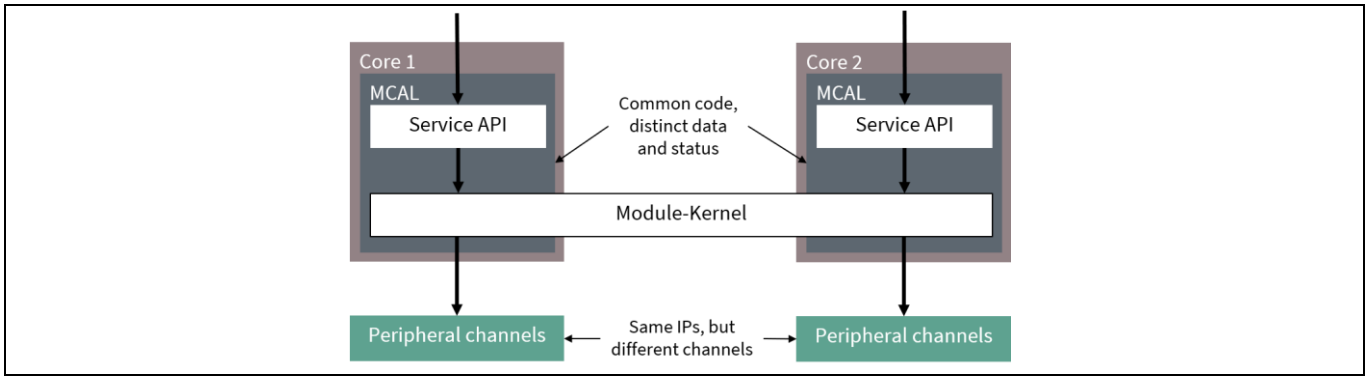

Figure 4 Overview of the multicore type II

1.7.1.3 Core-independent instances (multicore type III)

For this multicore type, the driver has core-independent instances with globally available hardware. This type is referred as “Multicore Type III”.

Multicore type III has the following characteristics:

- The code of the driver is shared among all cores:
  - A common binary is used for all cores.
  - A configuration is common for all cores.

- Each core runs an instance of the driver.
- Peripheral channels are globally available for all cores.

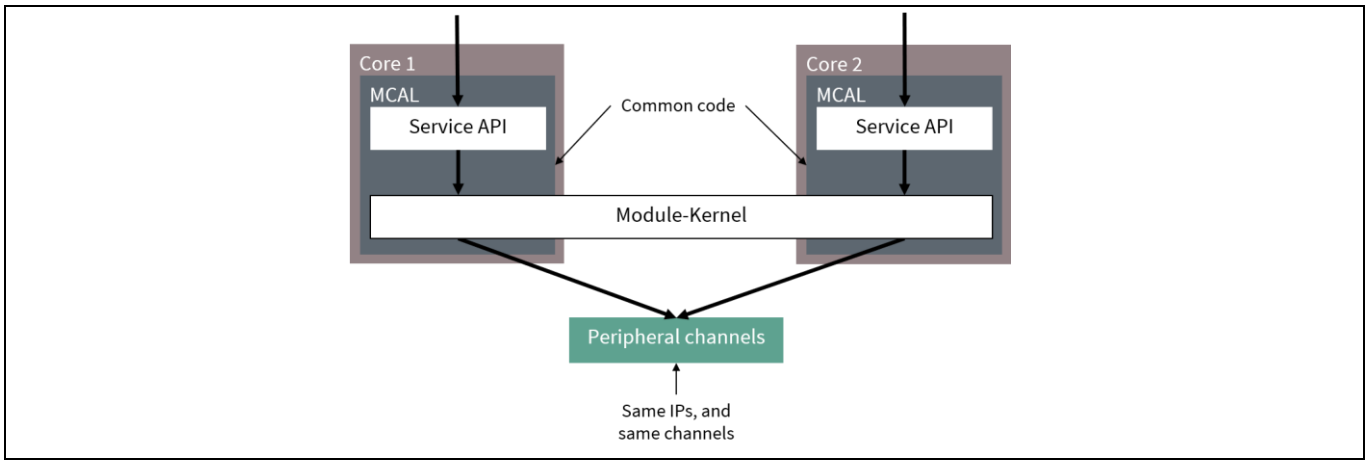

Figure 5 Overview of the multicore type III

1.7.2 Virtual core support

The GPT driver supports a number of cores. The configured cores need not be equal to the physical cores.

The GPT driver calls a configurable callout function (`GptGetCoreIdFunction`) to identify the core that is currently executing the code. This function can be implemented in the integration scope. The function can be written such that it does not return the physical core, but instead returns the SW partition ID, OS application ID, or any attribute/parameter. By interpreting these as the core, the GPT driver can support multiple SW partitions on a single physical core.

2 Using the GPT driver

This chapter describes all necessary steps to incorporate the general purpose timer (GPT) driver into your application.

2.1 Installation and prerequisites

Note: Before you start, see the EB tresos Studio for ACG8 user's guide [7] 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 (It includes the steps to setup and integrate the own application within the EB tresos ECU AUTOSAR build environment)*

The installation of the GPT driver complies with the general installation procedure for EB tresos ECU AUTOSAR components given in the documents mentioned above. If the driver has successfully been installed, the driver will appear in the module list of the EB tresos Studio.

This document assumes that you have set up your project using the application template as described in the *EB tresos Studio for ACG8 user's guide* [7]. This template provides the necessary folder structure, project and makefiles needed to configure and compile your application within the build environment. You must be familiar with the use of the command shell.

2.2 Configuring the GPT driver

This section gives a brief overview about the configuration structure defined by AUTOSAR to use the GPT driver.

The following basic containers are used to configure common behavior.

- `GptConfigurationOfOptApiServices`: This container is mainly used to restrict or extend the API of the GPT driver.
- `GptDriverConfiguration`: This container is mainly used to enable or disable default error tracer (DET), enable or disable wakeup reporting, and to configure GPT Predef timer.

For detailed information and description, see chapter 4 EB tresos Studio configuration interface.

The `GptChannelConfiguration` container groups configured channels. Each `GptChannelConfiguration` has a parameter:

- `GptChannelId`: Assigns a logical number of the GPT channel.
- `GptChannelMode`: Selects the behavior of the timer channel after the target time is reached (can be continuous or one-shot).
- `GptChannelTickFrequency`: Specifies the tick frequency of the timer channel in Hz.
- `GptChannelTickValueMax`: Specifies the maximum value in ticks that the timer channel can count.
- `GptEnableWakeup`: Enables or disables the wakeup capability of the channel.
- `GptNotification`: Specifies the callback function name.
- `GptChannelClkSrcRef`: Selects the reference clock to `GptClockReferencePoint`.
- `GptWakeupSourceRef`: Specifies the reference to the EcuM wakeup reason if the GPT channel is configured. It will be reported to EcuM.

The `GptChannelConfiguration` container can be configured for different GPT channels.

When selecting the hardware timer channel, it is important to ensure that each timer is only used for a single channel. Multiple GPT channels using the same timer are not supported. It is also important to ensure that the hardware timers are not in used by another AUTOSAR module (such as the OS).

For each channel that is configured, an ISR is needed. The ISR must be called `GPT_Isr_Vector_<IRQ No>_Cat1` for Category-1 ISR or `GPT_Isr_Vector_<IRQ No>_Cat2` for Category-2 ISR. The priority of this ISR determines the interrupt priority of the timer channel.

2.2.1 Architecture details

- `GptErrorCalloutFunction`: Specifies the error callout handler which is called when errors are detected during runtime.
- `GptIncludeFile`: Specifies the file name that is used to include some definitions (such as the declaration for the error callout handler).
- `GptSafetyFunctionApi`: Adds or removes the services `Gpt_CheckChannelStatus()` and `Gpt_CheckPredefTimerStatus()` services from the code.
- `GptSetPrescalerApi`: Adds or removes the service `Gpt_SetPrescaler()` from the code.
- `GptSetPredefTimerPrescalerApi`: Adds or removes the service `Gpt_SetPredefTimerPrescaler()` from the code.
- `GptPredefTimerStartTriggerSelect`: Specifies the input trigger associated with Predef timers to start synchronously.
- `GptTimer`: Selects the hardware timer to use for the logical channel.
- `GptChannelClkSrc`: Specifies the clock source of the timer channel.
- `GptInputTriggerSelection`: Specifies input trigger as an external clock input only when clock source `CLK_EXT` is selected.
- `GptChannelPrescale`: Specifies the prescaler of the external clock.
- `GptEnableDebug`: Enables or disables the debug capability to stop a timer channel when the processor is in debug mode.
- `GptHwTriggerOutputLine`: Specifies the output trigger channel. The trigger connection between output trigger channel and trigger destination such as ADC channel should be configured with PORT module.
- `GptPredefTimer`: Selects the hardware timer to use for the logical channel.
- `GptPredefTimerClkSrcRef`: Specifies the clock source of the GPT Predef timer.
- `GptPredefTimerType`: Specifies the type for GPT Predef timer.
- `GptPredefTimerTickFrequencyResult`: Shows calculated tick frequency of `GptPredefTimer` in Hz.
- `GptPredefTimerEnableDebug`: Enables or disables the debug capability to stop a GPT PredefTimer when the processor is in debug mode.
- `GptCoreAssignment`: Specifies the reference to a container of `GptCoreConfiguration` to select an assignment core for a channel.
- `GptCoreConsistencyCheckEnable`: Enables or disables the core consistency check during run-time.
- `GptGetCoreIdFunction`: Specifies the API to be called to get the core ID.
- `GptMasterCoreReference`: Specifies the reference to a container of `GptCoreConfiguration` to select the master core configuration.
- `GptCoreConfigurationId`: Specifies a logical number of the core ID.
- `GptCoreId`: Specifies the core ID. This ID is returned from the configured `GptGetCoreIdFunction` to identify the executing core.

2.3 Adapting your application

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

```
#include "Mcu.h" /* if using MCU clock reference point */
#include "Port.h" /* if using external clock or triggers */
#include "Gpt.h" /* GPT Driver */
```

Note: MCU and PORT driver should be included as necessary. See 3.4 Dependencies.

This publishes all required 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.

Declare the error callout function in the specified file with the `GptIncludeFile` parameter and implement it in your application (see Error Callout API in the 7.5 Required callback functions section).

The error callout function name can be configured by the `GptErrorCalloutFunction` parameter.

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

For the master core:

```
Mcu_Init(&Mcu_Config[0]); /* if using MCU clock reference point */
Port_Init(&Port_Config[0]); /* if using external clock or triggers */
Gpt_Init(&Gpt_Config[0]);
```

For the satellite core:

```
Mcu_Init(&Mcu_Config[0]);
      Gpt_Init(&Gpt_Config[0]);
```

Note: As a reference, the symbolic name can also be specified (e.g. `GptConf_GptConfigSet_GptConfigSet_0`).

Note: MCU and PORT driver should be initialized as necessary. See 3.4 Dependencies.

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. This function must be called on the master core only.

The function `Gpt_Init()` is called with a pointer to a structure of type `Gpt_ConfigType` which is published by the GPT driver itself.

The master core must be initialized prior to the satellite core. All cores must be initialized with the same configuration.

Once the GPT driver has been initialized, all enabled Predef timer channels start. The timer channels can be started and stopped as determined by system requirements. Whenever a timer is started by `Gpt_StartTimer()` and has reached the target time (and provided that notification has been enabled for the channel), the notification function for that channel is called by the driver.

Your application must provide the notification functions and its declarations that you configured. The file containing the declarations must be included using the `GptDriverConfiguration/GptIncludeFile` 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. On the command shell, type the following command to generate the necessary configuration-dependent files. See 3.3 Generated files.

   ```
   > make generate
   ```

2. Type the following command to resolve 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 CPU cores.

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

2.6.1 Memory allocation keyword

- `GPT_START_SEC_CODE_ASIL_B / GPT_STOP_SEC_CODE_ASIL_B`

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

- `GPT_START_SEC_CONST_ASIL_B_UNSPECIFIED / GPT_STOP_SEC_CONST_ASIL_B_UNSPECIFIED`

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

  - Gpt configuration setting

- `GPT_CORE[GptCoreConfigurationId]_START_SEC_VAR_INIT_ASIL_B_GLOBAL_UNSPECIFIED / GPT_CORE[GptCoreConfigurationId]_STOP_SEC_VAR_INIT_ASIL_B_GLOBAL_UNSPECIFIED`

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

  - Pointer to configuration setting
  - Information for GPT driver state

- `GPT_CORE[GptCoreConfigurationId]_START_SEC_VAR_CLEARED_ASIL_B_GLOBAL_UNSPECIFIED / GPT_CORE[GptCoreConfigurationId]_STOP_SEC_VAR_ CLEARED_ASIL_B_GLOBAL_UNSPECIFIED`

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

  - Information for target GPT timer and GPT Predef timer state

2.6.2 Memory allocation and constraints

All memory sections that store init or uninit status must be zero-initialized before any driver function is executed on any core. If core consistency checks are disabled, inconsistent parameters would be detected and reported by PPU and SMPU.

- `GPT_CORE[GptCoreConfigurationId]_START_VAR_[INIT_POLICY]_ASIL_B_GLOBAL_[ALIGNMENT]` / `GPT_CORE[GptCoreConfigurationId]_STOP_VAR_[INIT_POLICY]_ASIL_B_GLOBAL_[ALIGNMENT]`

  This section is read / write accessed from the core represented by `GptCoreConfigurationId` and read accessed from the other cores. Therefore, this section must not be allocated to TCRAM. For the core represented by `GptCoreConfigurationId`, this section must be allocated to either non-cache-able or write-through cache-able SRAM area. For performance, it is recommended to allocate the section to write-through cache-able SRAM. For other cores, this section must be allocated to non-cache-able SRAM area.

  For multicore type III, this section is read accessed from other cores. So, this section must not be allocated to TCRAM. For the core represented by `GptCoreConfigurationId`, this section must be allocated to either non-cache-able or write-through cache-able SRAM area. For performance, it is recommended to allocate the section to non-cache-able SRAM. For the other cores, this section must be allocated to non-cache-able SRAM area.

- STACK section

  TCRAM has dedicated memory for each core at the same address, and because of its performance it is recommended to allocate STACK to TCRAM.

Note: This restriction is applied only to Arm® Cortex®-M7 devices because they include TCMs and inner cache. There is no restriction when using Cortex®-M4 devices.
For the details of `INIT_POLICY` and `ALIGNMENT`, see the specification of memory mapping [6].

3 Structure and dependencies

The general purpose timer (GPT) driver consists of static, configuration, and generated files.

3.1 Static files

- $(PLUGIN_PATH)=$(TRESOS_BASE)/plugins/GPT_TS_* is the path to the GPT driver plugin.
- *$(PLUGIN_PATH)/lib_src* contains all static source files of the GPT driver. These files contain the functionality of the driver which does not depend on the current configuration. The files are grouped into a static library.
- *$(PLUGIN_PATH)/lib_include* contains all the internal header files for the GPT 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 GPT driver is enabled.

- *$(PLUGIN_PATH)/include* is the basic public include directory that is required and should include *Gpt.h*.
- *$(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 GPT driver.

3.2 Configuration files

The configuration of the GPT driver is done via EB tresos Studio. The file containing the GPT driver's configuration is named *Gpt.xdm* and is in the *$(PROJECT_ROOT)/config* directory. This file serves as an input to generate 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. They are in the *output/generated* sub folder of your project folder.

- *include/Gpt_Cfg.h* provides settings of configurations with pre-compile attribute, for example, declares ISR function and provides all symbolic names of the configuration. It will be included in *Gpt.h*.
- *include/Gpt_Cfg_Include.h* includes the header files specified by `GptIncludeFile`.
- *include/Gpt_Notification.h* provides settings of configurations with post-build attribute, for example, symbolic names of module configurations. It will be included in *Gpt.h*.
- *include/Gpt_PBcfg.h* provides configurations with post-build attributes such as channel configurations. It will be included by *Gpt.h*.
- *src/Gpt_Irq.c* contains the GPT channel ISR implementation.
- *src/Gpt_PBcfg.c* contains the constant structure for the GPT 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/Gpt_Bswmd.arxml* contains BswModuleDescription.

*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 MCU driver

The MCU driver needs to be initialized and all MCU clock reference points referenced by the GPT driver channels via configuration parameters `GptClockReferencePoint`, `GptChannelClkSrcRef`, and `GptPredefTimerClkSrcRef` must have been activated (via calls of MCU API functions) before initializing the GPT driver. `Mcu_GetCoreID` can optionally be set to configuration parameter `GptGetCoreIdFunction`. See the *MCU driver's user guide* for details.

3.4.2 PORT driver

Although the GPT driver can successfully be compiled and linked without an AUTOSAR-compliant PORT driver, the latter is required to configure and initialize the port pin for an external clock if an external clock is selected for the clock source. Trigger connection configurations are required for synchronization start of Predef timer, and trigger of peripherals such as ADC channel from GPT channel. Calling Port_ActTrigger () is required to start synchronization of the Predef timer. Otherwise, the GPT driver will show an undefined behavior.

3.4.3 ECU state manager

The GPT driver can be compiled and linked without an EcuM when the attribute `GptReportWakeupSource` is disabled. The EcuM is needed if the attribute `GptReportWakeupSource` is enabled or the ISR of a wakeup source services the wakeup event.

3.4.4 AUTOSAR OS

The OS must be used to configure and create the ISR vector table entries for the GPT. See 6.3 Interrupts. `GetCoreID` can optionally be set to configuration parameter `GptGetCoreIdFunction`.

3.4.5 DET

If the development error detection feature is enabled in the GPT driver, the DET needs to be installed, configured, and integrated into the application as well.

3.4.6 BSW scheduler

The GPT driver uses the following services of the basic software scheduler to enter and leave critical sections:

- `SchM_Enter_Gpt_GPT_EXCLUSIVE_AREA_[GptCoreConfigurationId](void)`
- `SchM_Exit_Gpt_GPT_EXCLUSIVE_AREA_[GptCoreConfigurationId](void)`

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

You must ensure that all interrupts are not occurred during enter critical section.

3.4.7 Error callout handler

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

4 EB tresos Studio configuration interface

The GUI is not part of the current delivery; see *EB tresos Studio for ACG8 user's guide* [7].

4.1 GptDriverConfiguration

The module comes preconfigured with default settings. These settings should be adapted when necessary.

- `GptDevErrorDetect` enables or disables development error notification for the GPT driver.

  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 as safety mechanisms (fault detection).

- `GptReportWakeupSource` determines whether the EcuM is informed of the wakeup events.
- `GptPredefTimer100us32bitEnable` enables or disables the GPT Predef timer that has 32 bits length and 100 µs tick duration.
- `GptPredefTimer1usEnablingGrade` specifies the grade of enabling the GPT Predef timers with 1 µs tick duration.
- `GptErrorCalloutFunction` specifies the error callout function name. The function is called on every error. The ASIL level of this function limits the ASIL level of the GPT driver.

Note: *`GptErrorCalloutFunction` must be a valid C function name; otherwise an error would occur in the configuration phase.*

- `GptIncludeFile` lists the filenames that will be included within the driver. Any application-specific symbol that is used by the GPT configuration (such as error callout function) should be included by configuring this parameter.

Note: *`GptIncludeFile` must be a unique filename with an extension .h; otherwise some errors would occur in the configuration phase.*

4.2 GptClockReferencePoint

This container contains one or more `McuClockReferencePoint` references (defined in module MCU).

- `GptClockReference` defines reference to a container of `McuClockReferencePoint` to select an input clock.

Note: *`GptClockReference` must specify each `McuClockReferencePoint` corresponding to the clock source selected by `GptChannelClkSrc` of each `GptChannelConfiguration` container, because the clock frequency has been set in the MCU module.*

4.3 Configuration of optional API services

- `GptDeinitApi` adds or removes the service `Gpt_DeInit()` to or from the code.
- `GptEnableDisableNotificationApi` adds or removes the services `Gpt_EnableNotification()` and `Gpt_DisableNotification()` to or from the code.
- `GptTimeElapsedApi` adds or removes the service `Gpt_GetTimeElapsed()` to or from the code.
- `GptTimeRemainingApi` adds or removes the service `Gpt_GetTimeRemaining()` to or from the code.
- `GptVersionInfoApi` adds or removes the service `Gpt_GetVersionInfo()` to or from the code.

- GptWakeupFunctionalityApi adds or removes the services Gpt_SetMode(), Gpt_EnableWakeup(), Gpt_DisableWakeup(), and Gpt_CheckWakeup() to or from the code.
- GptSafetyFunctionApi adds or removes the services Gpt_CheckChannelStatus() and Gpt_CheckPredefTimerStatus() to or from the code.
- GptSetPrescalerApi adds or removes the service GptSetPrescaler() to or from the code.
- GptSetPredefTimerPrescalerApi adds or removes the service GptSetPredefTimerPrescaler() to or from the code.

4.4 GptChannelConfiguration

Each GptChannelConfigSet object contains one or more GptChannelConfiguration container objects. This object has the following attributes:

- GptChannelId is a zero-based, consecutive integer value assigned to the symbolic name. This is used as logical channel ID. This value will be assigned to the following symbolic names:
  The symbolic name is prefixed with GptConf_GptChannelConfiguration.
- GptCoreAssignment specifies the reference to the GptCoreConfiguration for the channel core assignment.

Note: *GptCoreAssignment must have the target's GptCoreConfiguration setting.*
The same resource cannot be allocated to multiple cores.

- GptTimer selects the hardware timer associated with this GPT channel:
  - 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_1_n: Timer Counter Pulse Width Modulation Instance 1, Channel n

Note: *Selectable timer depends on the subderivative.*
ICU, OCU, PWM drivers and OS use TCPWM channels. GPT driver must not use TCPWM channel that is used by other modules.

- GptChannelMode (GPT_CH_MODE_CONTINUOUS or GPT_CH_MODE_ONESHOT) determines whether the channel operates in continuous mode or one-shot mode.
- GptChannelClkSrc selects the clock source that is used by the selected hardware timer.

Note: *Selectable clock source depends on the hardware timer that is selected by GptTimer. For example: CLK_INT or CLK_EXT.*
When the external clock input is selected (CLK_EXT), the timer counts at rising edges of the external clock.

- GptChannelClkSrcRef specifies the reference to the GptClockReferencePoint.

Note: *GptChannelClkSrcRef must specify the applicable clock source corresponding to the hardware timer selected by GptTimer from GptClockReference.*

- GptChannelTickFrequency specifies the tick frequency of the timer channel in Hz.

Note: If `GptChannelClkSrc` is `CLK_EXT`, you should set the tick frequency of the external input clock to `GptChannelTickFrequency` as a reference value. The tick frequency does not affect the generated code and is not used by the GPT channel.

This parameter is used for calculation of the divider. If the calculated value is not supported in hardware, an error message is reported.

Possible prescalers are 1, 2, 4, 8, 16, 32, 64, and 128 for TCPWM.

- `GptInputTriggerSelection` selects the input trigger which is used as an external clock input bound to one or more TCPWM channels.

Note: If `GptChannelClkSrc` is `CLK_INT`, `GptInputTriggerSelection` is not used.

The input trigger signal frequency should be half or less than the peripheral clock frequency referenced by GptChannelClkSrcRef.

The trigger connection between input trigger and TCPWM channel should be configured with PORT driver. The One-to-One trigger or trigger group is available.

ICU, OCU, PWM drivers, and `GptPredefTimerStartTriggerSelect` also use input triggers of TCPWM.

In the case that the same input trigger is configured in:

  - `IcuInputTriggerSelection`: a warning occurs.
  - `GptPredefTimerStartTriggerSelect` or other modules except for `IcuInputTriggerSelection`: an error occurs.
- `GptChannelPrescale` specifies the prescaler of external clock.
  - `GPT_PRESCALE_1`: Divided by 1

Note: The prescaler of external clock is fixed to `GPT_PRESCALE_1`.

- `GptChannelTickValueMax` is the maximum value, in ticks, that the timer channel can count. The minimum value for this parameter is 1.
- `GptEnableWakeup` determines whether this channel can be used as a wakeup source.
- `GptEnableDebug` determines whether this channel can be stopped when the processor is in debug mode.

Note: The configuration of trigger connection is required by PORT driver.

- `GptNotification (string)` is the name of the notification function for this channel.

Note: Notifications must be declared and defined outside the GPT module. The file containing the declarations must be included using the `GptDriverConfiguration/GptIncludeFile` parameter.

- `GptHwTriggerOutputLine` selects the output trigger for peripherals such as ADC channel.
  - `TR_OUT0`: Output trigger 0
  - `TR_OUT1`: Output trigger 1
  - `BOTH`: Both Output trigger 0 and trigger 1

Note: The configuration of trigger connection is required by PORT driver.

4.5 GPT wakeup configuration

- `GptWakeupSourceRef` specifies the reference to the wakeup source of EcuM if `GptEnableWakeup` is TRUE.

Note: If `GptWakeupSourceRef` is blank, the wakeup source value is set "0" by the GPT driver.

4.6 GptPredefTimerChannelConfiguration

Each `GptChannelConfigSet` object contains up to two `GptPredefTimerChannelConfiguration` container objects. This object has the following attributes:

- `GptPredefTimer` selects the hardware timer.
  - `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_1_n`: Timer Counter Pulse Width Modulation Instance 1, Channel n

Note: Selectable timer depends on the subderivative.
`GptTimer` uses TCPWM channels. `GptPredefTimer` must not use the same TCPWM channel as `GptTimer`.
ICU, OCU, PWM drivers and OS use TCPWM channels. GPT driver must not use TCPWM channel that is used by other modules.

- `GptPredefTimerClkSrcRef` specifies the reference to `GptClockReferencePoint`.

Note: `GptPredefTimerClkSrcRef` must specify the applicable clock source corresponding to the hardware timer selected by `GptPredefTimer` from `GptClockReference`.

- `GptPredefTimerType` specifies the GPT Predef timer with 100 µs tick duration or the grade of GPT Predef timer with 1 µs tick duration.
  - `GPT_PREDEF_TIMER_100US_32BIT`
  - `GPT_PREDEF_TIMER_1US_16_24_32BIT`
  - `GPT_PREDEF_TIMER_1US_16_24BIT`
  - `GPT_PREDEF_TIMER_1US_16BIT`

Note: One hardware timer is used per tick duration. For 1 µs tick duration timer, a lower bit timer is derived from a higher bit timer by a software mask operation.

- `GptPredefTimerTickFrequencyResult` shows a calculated tick frequency of the Predef timer channel in Hz.

Note: `GptPredefTimerTickFrequencyResult` shows the tick frequency of the Predef timer channel, in Hz, that is derived from `GptPredefTimerClkSrcRef`, `GptPredefTimerType` and the following prescaler value. If the tick frequency does not match with `GptPredefTimerType`, a warning message is displayed.
Possible prescalers are1, 2, 4, 8, 16, 32, 64, and 128 for TCPWM.

- `GptPredefTimerEnableDebug` determines whether this channel can be stopped when the processor is in the debug mode.

Note: The configuration of trigger is required.

4.7 GptPredefTimerStartTriggerConfiguration

- `GptPredefTimerStartTriggerSelect` specifies the input trigger for Predef timers to start synchronously.

Note: When both Predef timers that have 100 µs and 1 µs tick durations are configured, this parameter is available. When this parameter is not configured, Predef timers start sequentially.
The TCPWM instance of `GptPredefTimerStartTriggerSelect` must be the same as that of Predef timers.
ICU, OCU, PWM drivers and `GptInputTriggerSelection` also use input triggers of TCPWM. In the case that the same input trigger is configured, an error occurs.

4.8 GptMulticore

`GptMulticore` defines the multicore functional configuration of the GPT driver.

- `GptCoreConsistencyCheckEnable` enables or disables core consistency check during run-time. If enabled, GPT function checks if the provided parameter (channel) is allowed on the current core.

Note: Development error detect shall be enabled in GPT driver to enable this parameter.

- `GptGetCoreIdFunction` specifies the API to be called to get the core ID. For example: `GetCoreId()`

Note: `GptGetCoreIdFunction` must be a valid C function name.

- `GptMasterCoreReference` specifies the reference to the master core configuration.

Note: `GptMasterCoreReference` must have the target's `GptCoreConfiguration` setting.

4.9 GptCoreConfiguration

`GptCoreConfiguration` defines the core configuration of the GPT driver.

- `GptCoreConfigurationId` is a zero-based, consecutive integer value. This is used as a logical core ID.

Note: `GptCoreConfigurationId` must be unique across `GptCoreConfiguration`.

- `GptCoreId` is core ID assigned to GPT channels. This ID is returned from the configured `GptGetCoreIdFunction` execution to identify the executing core.

Note: `GptCoreId` must be unique across `GptCoreConfiguration`.
`The combination of GptCoreConfigurationId` and `GptCoreId` must be unique across `GptCoreConfiguration`.
`GptCoreConfiguration` can also be configured without GPT channel assignment.

5 Functional description

5.1 Function of the module

The GPT driver uses the counter/timer facilities of the microcontroller to present an abstract interval timer to the user program. One or more timer channels can be configured, each of which uses an independent interval timer or a compare register associated with a global free-running counter. The driver services the interrupts from the interval timers and calls the configured notification function when the interval ends (the target time is reached).

Timer channels can be configured as either one-shot or continuous. One-shot timers are stopped when the first target time is reached. Continuous timers restart automatically, thus providing a stream of regularly-spaced notification events until explicitly stopped.

The elapsed time since the last notification occurred (considering the channel has been started) as well as the time remaining until the next notification can be retrieved (see Figure 6).

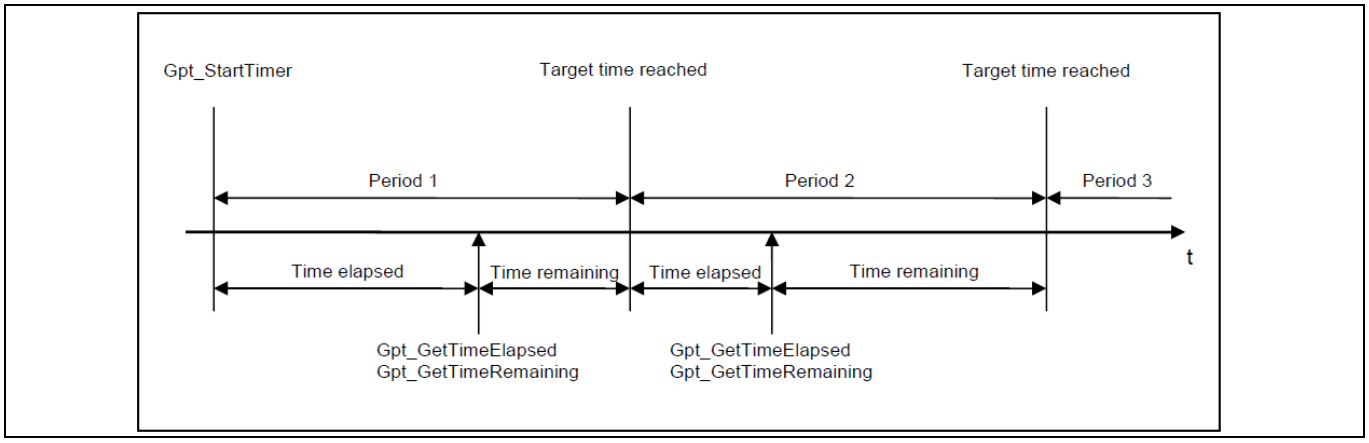

Figure 6 GPT driver functionality

Beside individual timer channels with individual properties, GPT Predef timers are defined. These timers have predefined tick durations and predefined number of bits.

The GPT driver only generates time bases and does not serve as an event counter or gated counter. This functionality is provided by another driver module. Neither does the driver measure time in any abstract sense. All units are specified in ticks of the respective timer. It is not even guaranteed that the channels all tick at the same rate.

5.2 Initialization

The GPT driver provides functions for module initialization. The GPT driver needs to be initialized once on each core before use. Initialization of the GPT driver is made by calling `Gpt_Init()`.

Note: This GPT driver supports post-build-time configuration, thus different configuration set pointer can be passed to the function `Gpt_Init()`. `Gpt_Init()` must be called on the master core before any cores are initialized. If `Gpt_Init()` is called on the satellite core, the master core must be already initialized. The same configuration set must be specified on all cores during initialization. If no channel is assigned to the satellite core, `Gpt_Init()` is not required on that core.

5.3 De-initialization

The GPT driver provides functions for module de-initialization once on each core after use. De-initialization of the GPT driver is made by calling `Gpt_DeInit()`.

Note: *`Gpt_DeInit()` must be called on the master core after all satellite cores are de-initialized. If `Gpt_DeInit()` is called on the satellite core, the master core must be already initialized. The integrated system must prevent other cores from calling the GPT API while `Gpt_DeInit()` is being called.*

5.4 Sleep mode

In addition to its normal operation mode, the driver can also operate in a lower-power Sleep mode. In this mode, only those channels that are configured and enabled for Sleep mode operation are permitted to run - all other channels on the current core are automatically stopped when Sleep mode is entered.

If a wakeup-enabled channel's target time has been reached during the Sleep mode, and provided the attribute `GptReportWakeupSource` is enabled for the driver, the ECU state manager (EcuM) module is notified by calling its `EcuM_CheckWakeup()` function.

Note: *The wakeup functionality is not available while the MCU is in DeepSleep or Hibernate mode due to the hardware architecture.*

Note: *All TCPWM counters must be stopped before entering DeepSleep mode by using one of the following options:*

1. *Call `Gpt_DeInit()`.*
2. *Call `Gpt_DisableWakeup()` if there are wakeup-enabled channels. After that, call `Gpt_SetMode(GPT_MODE_SLEEP)` to stop all timers including Predef timers.*

When the driver goes into the Sleep mode by calling `Gpt_SetMode(GPT_MODE_SLEEP)`, all channels on the current core are stopped except those that are configured with `GptEnableWakeup` = *'true'*. When `Gpt_SetMode(GPT_MODE_SLEEP)` is called from the master core, all enabled GPT Predef timers are stopped.

If no `GptWakeupConfiguration` is configured, the expired interrupt service routine (ISR) of the timer channel that reaches the target time can be used to leave the CPU sleep state. The EcuM will not be informed because no wakeup information was configured that could be reported to the EcuM.

Note: *When the driver is taken out of Sleep mode, the channels that were stopped on entry to Sleep mode are not automatically restarted. When `Gpt_SetMode(GPT_MODE_NORMAL)` is called from the master core, all enabled GPT Predef timers are restarted at value “0”.*

5.5 Output trigger to peripherals

The GPT driver generates output triggers to peripherals such as ADC channel when the GPT timer expires.

The parameter `GptHwTriggerOutputLine` is used to allow the GPT channel to trigger peripheral actions on all cores. The trigger connection between output trigger channel and trigger destination should be configured with PORT driver.

The available output trigger channel depends on the hardware.

5.6 Prescaler setting function

The GPT driver provides the prescaler setting function for the GPT channel and the GPT Predef timer during runtime. This function maintains the same tick frequency of a timer channel by changing the prescaler setting without initialization, when the input clock frequency for a timer channel is changed.

`Gpt_SetPrescaler()` and `Gpt_SetPredefTimerPrescaler()` change the TCPWM prescaler setting of the selected channel according to the specified input clock frequency.

The GPT channel should be stopped by `Gpt_StopTimer()` before the calling `Gpt_SetPrescaler()`. If the GPT channel is in running state, `Gpt_SetPrescaler()` rises the `GPT_E_BUSY`. In case of GPT Predef timer, `Gpt_SetPredefTimerPrescaler()` stops the GPT Predef timer before changing its prescaler and then restarts the GPT Predef timer. If `Gpt_SetPredefTimerPrescaler()` is called during Sleep mode, `Gpt_SetPredefTimerPrescaler()` does not restart the GPT Predef timer after changing its prescaler. In that case, the calling of `Gpt_SetMode(GPT_MODE_NORMAL)` restarts the GPT Predef timer.

- GPT channel case:

  Input clock frequency of `GptConf_GptChannelConfiguration_MY_CHANNEL` is changed by MCU driver.

```
Gpt_StopTimer(GptConf_GptChannelConfiguration_MY_CHANNEL);
Gpt_SetPrescaler(GptConf_GptChannelConfiguration_MY_CHANNEL,
 MY_CLOCK_FREQUENCY);
Gpt_StartTimer(GptConf_GptChannelConfiguration_MY_CHANNEL, MY_VALUE);
```

- GPT Predef timer case:

  Input clock frequency of `GPT_PREDEF_TIMER_100US_32BIT` is changed by MCU driver.

```
Gpt_SetPredefTimerPrescaler(GPT_PREDEF_TIMER_100US_32BIT, MY_CLOCK_FREQUENCY);
```

Note: `Gpt_SetPrescaler()` and `Gpt_SetPredefTimerPrescaler()` calculate the prescaler value based on the value of the input clock frequency and `GptChannelTickFrequency`.

Calculating formula: `MY_CLOCK_FREQUENCY` divided by `GptChannelTickFrequency` (Round down decimals)

The input frequency value of `Gpt_SetPrescaler()` and `Gpt_SetPredefTimerPrescaler()` arguments causes poor accuracy of the tick duration in the following cases. In such cases, the frequency of input clock should be adjusted by the MCU driver to meet the prescaler value (1, 2, 4, 8, 16, 32, 64, 128).

- The calculation result of `ClockFrequency` divided by TickFrequency does not meet the prescaler value (1, 2, 4, 8, 16, 32, 64, 128).
- If the input clock frequency is close to the TickFrequency, the appropriate prescaler value may not be set.

5.7 GPT predef timer synchronous start

The GPT driver starts two GPT Predef timers synchronously by configuring `GptPredefTimerStartTriggerSelect`. In this case, the trigger signal should be configured by the PORT driver. Besides, the TCPWM instance number should be the same between the trigger and both GPT Predef timers.

When this parameter is configured, `Gpt_Init()`, `Gpt_SetMode (GPT_MODE_NORMAL)` and `Gpt_SetPredefTimerPrescaler()` do not start the Predef timers. The calling of `Port_ActTrigger()` is necessary to generate a trigger signal for synchronous start. When `GptPredefTimerStartTriggerSelect` is not configured, Predef timers start sequentially by the calling of `Gpt_Init()`, `Gpt_SetMode(GPT_MODE_NORMAL)`, or `Gpt_SetPredefTimerPrescaler()` on the master core.

Examples of use cases are as follows.

- Initialization case:

```
Gpt_Init(GptConf_GptChannelConfiguration_INIT_PATTERN_0);
Port_ActTrigger(PortConf_PortTrGroupContainer_MY_TRIGGER_GROUP,
                PortConf_PortOutputTrigger_MY_OUTPUT_TRIGGER,
                PORT_TR_ACTIVATION_OUTPUT,
                PORT_TR_SENSITIVE_EDGE);
```

- Switching from Sleep mode to normal mode case:

```
Gpt_SetMode(GPT_MODE_SLEEP); GPT Predef Timers stop because of sleep mode.
Gpt_SetMode(GPT_MODE_NORMAL);
Port_ActTrigger(PortConf_PortTrGroupContainer_MY_TRIGGER_GROUP,
                PortConf_PortOutputTrigger_MY_OUTPUT_TRIGGER,
                PORT_TR_ACTIVATION_OUTPUT,
                PORT_TR_SENSITIVE_EDGE);
```

- Setting prescaler case:

  MCU driver changes the input clock frequency of GPT Predef timers.

```
Gpt_SetPredefTimerPrescaler(GPT_PREDEF_TIMER_100US_32BIT, MY_CLOCK_FREQUENCY);
Gpt_SetPredefTimerPrescaler(GPT_PREDEF_TIMER_1US_32BIT, MY_CLOCK_FREQUENCY);
Port_ActTrigger(PortConf_PortTrGroupContainer_MY_TRIGGER_GROUP,
                PortConf_PortOutputTrigger_MY_OUTPUT_TRIGGER,
                PORT_TR_ACTIVATION_OUTPUT,
                PORT_TR_SENSITIVE_EDGE);
```

Note: `Port_ActTrigger()` must be called immediately after `Gpt_Init()`, `Gpt_SetMode(GPT_MODE_NORMAL)`, and `Gpt_SetPredefTimerPrescaler()`.

5.8 Runtime reconfiguration

The GPT 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.

5.9 API parameter checking

The driver's services perform regular error checks.

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

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

If an error occurs, the GPT driver skips the service.

The development error checks that are performed are documented along with the APIs that perform them. The error codes and their numerical values can be found in the header file *Gpt.h*.

5.10 Production error detection

The GPT driver does not detect production errors.

5.11 Reentrancy

All services that operate on the driver (except for `Gpt_GetVersionInfo`) are not reentrant. Services that operate on a single channel are reentrant provided that the concurrent calls do not operate on the same channel.

5.12 Debugging support

The GPT driver does not support debugging.

5.13 Execution time dependencies

The execution of the API function is dependent on certain factors. Table 1 lists these dependencies.

Table 1 Execution time dependencies

Affected function	Dependency

`Gpt_Init()`
`Gpt_DeInit()`
`Gpt_CheckWakeup()`
`Gpt_SetMode()`
	Runtime depends on the number of configured channels because all configured channels are processed in these functions.

5.14 Functions available without core dependency

Some APIs can be called on any cores regardless resource assignment.

The following function is available on any core without any restriction:

- `Gpt_GetVersionInfo()`

The following functions are available on any cores with a specific section allocation described in the Note:

- `Gpt_GetTimeElapsed()`
- `Gpt_GetTimeRemaining()`
- `Gpt_GetPredefTimerValue()`

These functions can get the timer value from any cores.

Note: The section `VAR_[INIT_POLICY]_ASIL_B_GLOBAL_[ALIGNMENT]` must be allocated to the memory can be read from any cores to call these APIs on any cores.
For the details of `INIT_POLICY` and `ALIGNMENT`, see the specification of memory mapping [6].

6 Hardware resources

6.1 Ports and pins

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

The following physical hardware timer must be configured for GPT channel:

- `GptTimer` = `TCPWM_x_y` (x: instance number, y: channel number)

The PORT driver must be configured with the appropriate port pin when the external clock input is selected (`CLK_EXT`) for the clock source.

- `PortPinName` = `Pxxx_y` for GPT pin
- `PortPinDirection` = `PORT_PIN_IN`
- `PortPinInitialMode` = `TCPWM_x_LINE_y`
- `PortPinMode` = `TCPWM_x_LINE_y` (same as `PortPinInitialMode`)

The associated port pin for each GPT channel is subderivative dependent and refers to the target device's hardware manual.

6.2 Timer

The GPT driver uses the configured counter/timers of TCPWM. For each configured GPT channel or Predef timer channel, one hardware timer of the TRAVEO™ T2G family is reserved exclusively. This is done via configuration parameter `GptTimer` or `GptPredefTimer`.

6.3 Interrupts

The GPT driver uses the interrupts associated with the configured hardware timers. The ISR should be allocated to the same core as allocated timer. The ISR must be declared in the AUTOSAR OS as Category 1 Interrupt or Category 2 Interrupt.

Note: IRQ numbers depend on the subderivative. See the target device's hardware manual.

You can define the ISR as:

The IRQ-Name of each GPT channel must be `GPT_Isr_Vector_<IRQ No>_Cat1` for Category-1 ISR or `GPT_Isr_Vector_<IRQ No>_Cat2` for Category-2 ISR.

Note: `GPT_Isr_Vector_<IRQ No>_Cat2` must be called from the (OS) interrupt service routine. In the case of category1 usage, the address of `GPT_Isr_Vector_<IRQ No>_Cat1` must be the entry in the (OS) interrupt vector table.

Example: Category-1 ISR for the configured GPT channel 0 located in the generated file *src/Gpt_Irq.c*:

```
ISR_NATIVE(GPT_Isr_Vector_276_Cat1)
{
...
    }
```

Example: Category-2 ISR for the configured GPT channel 0 located in the generated file *src/Gpt_Irq.c*:

```
ISR(GPT_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 One-to-One group. Trigger group is a multiplexer-based connectivity group. This type connects a peripheral input trigger to multiple peripheral output triggers. Trigger One-to-One group is a One-to-One-based connectivity group. This type connects a peripheral input trigger to one specific peripheral output trigger.

For more detail about triggers, see the hardware documentation.

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

- Triggering peripheral action

  GPT channel generates output trigger signal to peripherals. See 5.5 Output trigger to peripherals.

- External clock source

  Input trigger signal is used for a clock source when the external clock input is selected (`CLK_EXT`) for the clock source. See `GptInputTriggerSelection` in 4.4 GptChannelConfiguration.

- Predef timers synchronous start

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

  `PortOutputTrigger` configuration setting:

  - `PortTrOutputName`: Select trigger signal to all channels of TCPWM (e.g. `TCPWM_0_TR_ALL_CNT_IN_8`).
  - `PortTrInputName`: Select trigger signal (e.g. `CPUSS_ZERO`)
  - `PortTrInvertEnable`: Disable
  - `PortTrSensitiveType`: `PORT_TR_SENSITIVE_EDGE`

- Debug capability to stop timer channels

6 Hardware resources

A trigger multiplexer from CTI to TCPWM for each TCPWM instance should be configured in PORT driver.

- `PortOutputTrigger` configuration setting:
- `PortTrOutputName`: `TCPWM_n_TR_DEBUG_FREEZE` (n: TCPWM instance number)
- `PortTrInputName`: Select CTI signal
- `PortTrInvertEnable`: Disable
- `PortTrSensitiveType`: `PORT_TR_SENSITIVE_LEVEL`

7 Appendix A – API reference

7.1 Include files

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

7.2 Data types

7.2.1 Gpt_ConfigType

Type

```
typedef struct
```

Description

Defines the structure of configuration data for the GPT driver.

7.2.2 Gpt_ChannelType

Type

```
uint16
```

Description

Defines the channel ID of the GPT channel.

7.2.3 Gpt_ValueType

Type

```
uint32
```

Description

On the TRAVEO™ T2G family microcontroller, the counter channels are 16-bit wide and 32-bit wide; therefore, the `Gpt_ValueType` is 32-bit wide. The maximum possible interval that can be timed by GPT is 2^16-1 (0xFFFF) ticks for a 16-bit timer and 2^32-1 (0xFFFFFFFF) ticks for a 32-bit timer.

7.2.4 Gpt_ModeType

Type

```
typedef enum
{
GPT_MODE_NORMAL = 0U,
GPT_MODE_SLEEP = 1U
} Gpt_ModeType;
```

Description

The mode of the GPT driver.

- `GPT_MODE_NORMAL`: Normal operation mode of the GPT.

- `GPT_MODE_SLEEP`: Operation for reduced power operation mode. Only in Sleep mode, the wakeup channels are available.

7.2.5 Gpt_PredefTimerType

Type

```
typedef enum
{
GPT_PREDEF_TIMER_1US_16BIT   = 0U,
GPT_PREDEF_TIMER_1US_24BIT   = 1U,
GPT_PREDEF_TIMER_1US_32BIT   = 2U,
GPT_PREDEF_TIMER_100US_32BIT = 3U
} Gpt_PredefTimerType;
```

Description

The type for the GPT Predef timer:

- `GPT_PREDEF_TIMER_1US_16BIT`: GPT Predef timer with a tick duration of 1 µs and a range of 16 bits.
- `GPT_PREDEF_TIMER_1US_24BIT`: GPT Predef timer with a tick duration of 1 µs and a range of 24 bits.
- `GPT_PREDEF_TIMER_1US_32BIT`: GPT Predef timer with a tick duration 1 µs and a range of 32 bits.
- `GPT_PREDEF_TIMER_100US_32BIT`: GPT Predef timer with a tick duration of 100 µs and a range of 32 bits.

7.2.6 Gpt_ChannelInterruptOccurredType

Type

`uint8`

Description

The flag to detect interrupt. For valid values, see Table 6.

7.2.7 Gpt_ChannelStateFlagType

Type

`uint8`

Description

The enable or disable flag of the GPT or EcuM notification.

7.2.8 Gpt_ChannelStateValueType

Type

`uint8`

Description

The status of the GPT timer. For valid values, see Table 7.

7.2.9 Gpt_ChannelStateType

Type

```
typedef struct
{
Gpt_ValueType                       interval;
Gpt_ClkFrequencyType                frequency;
Gpt_ChannelStateValueType           state;
Gpt_ChannelInterruptOccurredType    channelInterruptOccurred;
Gpt_ChannelStateFlagType            channelStateFlags;
uint8                               prescaler;
} Gpt_ChannelStateType;
```

Description

The status of the GPT channel:

- `Interval`: Interval time
- `frequency`: Tick frequency
- `state`: Status of channel
- `channelInterruptOccurred`: Status of the interrupt
- `channelStateFlags`: Status of notification/wakeup
- `prescaler`: Current prescaler register value

7.2.10 Gpt_DriverStateValueType

Type

```
uint8
```

Description

The initialization status of the GPT driver. For valid values, see Table 8.

7.2.11 Gpt_DriverStateType

Type

```
typedef struct
{
Gpt_ModeType                driverStateFlags;
Gpt_DriverStateValueType    driverInitialised;
} Gpt_DriverStateType;
```

Description

The status of the GPT driver:

- `driverStateFlags`: Status of sleep.
- `driverInitialised`: Status of initialization.

7.2.12 Gpt_ClkFrequencyType

Type

`uint32`

Description

Type for clock frequency in Hz.

7.3 Constants

7.3.1 Error codes

Table 2 lists the development error codes that can be reported by the GPT driver:

Table 2 Error codes

Name	Value	Description
GPT_E_UNINIT	0x0A	Module was not initialized.
GPT_E_BUSY	0x0B	Channel is already running.
GPT_E_MODE	0x0C	GPT driver was in a wrong mode.
GPT_E_ALREADY_INITIALIZED	0x0D	Module was already initialized.
GPT_E_INIT_FAILED	0x0E	Initialization function failed.
GPT_E_PARAM_CHANNEL	0x14	Invalid channel was passed on as parameter.
GPT_E_PARAM_VALUE	0x15	Invalid value was passed on as parameter.
GPT_E_PARAM_POINTER	0x16	Invalid pointer was passed on as parameter.
GPT_E_PARAM_PREDEF_TIMER	0x17	Invalid Predef timer was passed on as parameter.
GPT_E_PARAM_MODE	0x1F	Invalid mode parameter.
GPT_E_INTERNAL	0x22	Interrupt occurred while driver is not initialized. Predef timer was wrong status.
GPT_E_PARAM_CLOCK	0x30	Invalid clock frequency was passed on as parameter.
GPT_E_INVALID_CORE	0x40	Invalid core ID was passed on as parameter.
GPT_E_DIFFERENT_CONFIG	0x41	Different configuration pointer was passed on as parameter.

7.3.2 Version information

The version information listed in Table 3 is published in the driver's header file

Table 3 Version information

Name	Value	Description
GPT_AR_RELEASE_MAJOR_VERSION	4	AUTOSAR specification major version
GPT_AR_RELEASE_MINOR_VERSION	2	AUTOSAR specification minor version
GPT_AR_RELEASE_REVISION_VERSION	2	AUTOSAR specification patch version

Name	Value	Description
GPT_SW_MAJOR_VERSION	Refer to release notes.	Major version number
GPT_SW_MINOR_VERSION	Refer to release notes.	Minor version number
GPT_SW_PATCH_VERSION	Refer to release notes.	Patch version number

7.3.3 Module information

Table 4 Module information

Name	Value	Description
GPT_MODULE_ID	100	Module ID
GPT_VENDOR_ID	66	Vendor ID

7.3.4 API service IDs

The API service IDs listed in Table 5 are published in the driver's header file.

Table 5 API service IDs

Name	Value	API service
GPT_GETVERSIONINFO	0x00	Gpt_GetVersionInfo
GPT_INIT	0x01	Gpt_Init
GPT_DEINIT	0x02	Gpt_DeInit
GPT_GETTIMEELAPSED	0x03	Gpt_GetTimeElapsed
GPT_GETTIMEREMAINING	0x04	Gpt_GetTimeRemaining
GPT_STARTTIMER	0x05	Gpt_StartTimer
GPT_STOPTIMER	0x06	Gpt_StopTimer
GPT_ENABLENOTIFICATION	0x07	Gpt_EnableNotification
GPT_DISABLENOTIFICATION	0x08	Gpt_DisableNotification
GPT_SETMODE	0x09	Gpt_SetMode
GPT_DISABLEWAKEUP	0x0A	Gpt_DisableWakeup
GPT_ENABLEWAKEUP	0x0B	Gpt_EnableWakeup
GPT_CHECKWAKEUP	0x0C	Gpt_CheckWakeup
GPT_GETPREDEFTIMERVALUE	0x0D	Gpt_GetPredefTimerValue
GPT_CHECKPREDEFTIMERSTATUS	0xFB	Gpt_CheckPredefTimerStatus
GPT_SETPRESCALER	0xFC	Gpt_SetPrescaler
GPT_SETPREDEFTIMERPRESCALER	0xFD	Gpt_SetPredefTimerPrescaler
GPT_CHECKCHANNELSTATUS	0xFE	Gpt_CheckChannelStatus
GPT_TIMEREXPIRE	0xFF	Interrupt service for timer expiring.

7.3.5 Interrupt occurred flags

Table 6 Interrupt occurred flags

Name	Value	Description
GPT_NO_INTERRUPT_DETECT	0x00	No interrupt detected
GPT_INTERRUPT_DETECT	0x01	Interrupt detected

7.3.6 Channel state values

Table 7 Channel state values

Name	Value	Description
GPT_STATE_UNINITIALIZE	0x00	Uninitialized state
GPT_STATE_INITIALIZED	0x01	Initialized state
GPT_STATE_RUNNING	0x02	Running state
GPT_STATE_STOPPED	0x03	Stopped state
GPT_STATE_EXPIRED	0x04	Expired state

7.3.7 Driver state values

Table 8 Driver state values

Name	Value	Description
GPT_DRIVER_UNINITIALIZE	0x00	Uninitialized state
GPT_DRIVER_INITIALIZE	0x01	Initialized state

7.3.8 Invalid core ID value

Table 9 Invalid core ID

Name	Value	Description
GPT_INVALID_CORE	0xFF	Invalid core ID

7.4 Functions

7.4.1 Gpt_GetVersionInfo

Syntax

```
void Gpt_GetVersionInfo
(
Std_VersionInfoType* VersionInfoPtr
)
```

Service ID

0x00

Parameters (in)

None

Parameters (out)

- `VersionInfoPtr` – Indicates the location to place the GPT's version information

Return value

None

DET errors

- `GPT_E_PARAM_POINTER` – The `VersionInfoPtr` parameter is invalid (NULL).

DEM errors

None

Description

This function returns the GPT driver's version information in the caller-specified structure.

7.4.2 Gpt_Init

Syntax

```
void Gpt_Init
(
const Gpt_ConfigType* ConfigPtr
)
```

Service ID

0x01

Parameters (in)

- `ConfigPtr` – Pointer to a selected configuration structure.

Parameters (out)

None

Return value

None

DET errors

- `GPT_E_INVALID_CORE` – The current core ID is invalid.
- `GPT_E_INIT_FAILED` – The configuration pointer is incorrect, or the function is called on the satellite core while the master core is not initialized.
- `GPT_E_ALREADY_INITIALIZED` – The service is called on the master core when other cores are already initialized, or the service is called on the satellite core while the satellite core is already initialized.
- `GPT_E_DIFFERENT_CONFIG` – The configuration pointer is different from the initialized configuration of the master core.

DEM errors

None

Description

This service initializes the driver. When the service returns, all channels are in a quiescent state with notification and wakeup disabled. This function will be called with a pointer to a selected configuration structure.

Note: This service only affects GPT channels assigned to the current core. When this service is called from the master core, all enabled GPT Predef timers are started.

7.4.3 Gpt_DeInit

Syntax

```
void Gpt_DeInit
(
void
)
```

Service ID

0x02

Parameters (in)

None

Parameters (out)

None

Return value

None

DET errors

- `GPT_E_INVALID_CORE` – The current core ID is invalid.
- `GPT_E_UNINIT` – The driver has not been initialized.
- `GPT_E_BUSY` – Any channel on the core is in state “running” or all satellite core is not uninitialized when the service is called on the master core.

DEM errors

None

Description

This service de-initializes the driver. In production mode, all channels will be stopped prior disabling the driver.

Note: This service only affects GPT channels assigned to the current core. When this service is called from the master core, all enabled GPT Predef timers are stopped.

7.4.4 Gpt_GetTimeElapsed

Syntax

```
Gpt_ValueType Gpt_GetTimeElapsed
(
Gpt_ChannelType Channel
)
```

Service ID

0x03

Parameters (in)

- `Channel` – ID of the GPT channel.

Parameters (out)

None

Return value

The time elapsed, to start the channel or the last recurrence (target time reached) in continuous mode.

DET errors

- `GPT_E_UNINIT` – The driver has not been initialized.
- `GPT_E_PARAM_CHANNEL` – The channel ID is not valid.

DEM errors

None

Description

This service returns the number of ticks that has elapsed since the timer was most recently started, or restarted if the channel is configured for continuous operation. The return value is the value at the moment of accessing the counter register. If the channel has not yet run, the return value is 0. If the channel has stopped, the return value is the current time value.

7.4.5 Gpt_GetTimeRemaining

Syntax

```
Gpt_ValueType Gpt_GetTimeRemaining
(
Gpt_ChannelType Channel
)
```

Service ID

0x04

Parameters (in)

- `Channel` – ID of the GPT channel.

Parameters (out)

None

Return value

The time remaining until the target time.

DET errors

- `GPT_E_UNINIT` – The driver has not been initialized.
- `GPT_E_PARAM_CHANNEL` – The channel ID is not valid.

DEM errors

None

Description

This service returns the number of ticks that remain until the target time. The return value is calculated from the value at the moment of accessing the counter register. If the channel has not yet run, the return value is 0. If the channel is stopping, the return value is the current time value.

7.4.6 Gpt_StartTimer

Syntax

```
void Gpt_StartTimer
(
Gpt_ChannelType Channel,
Gpt_ValueType Value
)
```

Service ID

0x05

Parameters (in)

- `Channel` – ID of the GPT channel.
- `Value` – Target time in ticks.

Parameters (out)

None

Return value

None

DET errors

- `GPT_E_UNINIT` – The driver is not initialized.
- `GPT_E_BUSY` – The channel is already running.
- `GPT_E_PARAM_CHANNEL` – The channel ID is invalid.
- `GPT_E_PARAM_VALUE` – The value is 0 or not within range specified by the configuration.
- `GPT_E_INVALID_CORE` – The current core ID is invalid or different from the channel assignment core ID.

DEM errors

None

Description

This service starts the selected timer channel with the specified target time. When the target time has elapsed and notification is enabled, the configured notification function will be called, if notification is enabled. The maximum value for the target time parameter can be found in the description of `GptChannelTickValueMax` in 4.4 GptChannelConfiguration.

7.4.7 Gpt_StopTimer

Syntax

```
void Gpt_StopTimer
(
Gpt_ChannelType Channel
)
```

Service ID

0x06

Parameters (in)

- `Channel` – ID of the GPT channel.

Parameters (out)

None

Return value

None

DET errors

- `GPT_E_UNINIT` – The driver is not initialized.
- `GPT_E_PARAM_CHANNEL` – The channel ID is invalid.
- `GPT_E_INVALID_CORE` – The current core ID is invalid or different from the channel assignment core ID.

DEM errors

None

Description

This service stops the selected timer channel. The service is considered non-operational, if the channel is not running and no error is reported.

7.4.8 Gpt_EnableNotification

Syntax

```
void Gpt_EnableNotification
(
Gpt_ChannelType Channel
)
```

Service ID

0x07

Parameters (in)

- `Channel` – ID of the GPT channel.

Parameters (out)

None

Return value

None

DET errors

- `GPT_E_UNINIT` – The driver has not been initialized.
- `GPT_E_PARAM_CHANNEL` – The channel ID is invalid or the channel has no notification function.
- `GPT_E_INVALID_CORE` – The current core ID is invalid or different from the channel assignment core ID.

DEM errors

None

Description

This service enables the notification facility for the specified channel.

7.4.9 Gpt_DisableNotification

Syntax

```
void Gpt_DisableNotification
(
Gpt_ChannelType Channel
)
```

Service ID

0x08

Parameters (in)

- `Channel` – ID of the GPT channel.

Parameters (out)

None

Return value

None

DET errors

- `GPT_E_UNINIT` – The driver has not been initialized.
- `GPT_E_PARAM_CHANNEL` – The channel ID is invalid or the channel has no notification function.
- `GPT_E_INVALID_CORE` – The current core ID is invalid or different from the channel assignment core ID.

DEM errors

None

Description

This service disables the notification facility for the specified channel.

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

7.4.10 Gpt_SetMode

Syntax

```
void Gpt_SetMode
(
Gpt_ModeType Mode
)
```

Service ID

0x09

Parameters (in)

- `Mode` – The desired mode; either `GPT_MODE_NORMAL` or `GPT_MODE_SLEEP`.

Parameters (out)

None

Return value

None

DET errors

- `GPT_E_INVALID_CORE` – The current core ID is invalid.
- `GPT_E_UNINIT` – The driver is not initialized.
- `GPT_E_PARAM_MODE` – The mode is invalid.

DEM errors

None

Description

This service changes the mode of the current core. In Sleep mode, all channels of the current core that are not enabled for wakeup are stopped. When the channel for wakeup reaches the target time, events are reported to the EcuM using its `EcuM_SetWakeupEvent()` service. In the normal mode, all channels of the current core are permitted to run and the EcuM is not notified. When the current core is the master core, GPT Predef timers stops in Sleep mode and restarts at value “0” in normal mode.

Note: This service only affects wakeup enabled channels assigned to the current core. When the current core is the master core, this service affects all enabled GPT Predef timers.
A transition from the normal mode to the Sleep mode stops all GPT channels on the current core that are not enabled for wakeup, but the transition from Sleep mode to normal mode does not restart any GPT channels that were stopped during the previous transition.

7.4.11 Gpt_DisableWakeup

Syntax

```
void Gpt_DisableWakeup
(
Gpt_ChannelType Channel
)
```

Service ID

0x0A

Parameters (in)

- `Channel` – ID of the GPT channel.

Parameters (out)

None

Return value

None

DET errors

- `GPT_E_UNINIT` – The driver is not initialized.
- `GPT_E_PARAM_CHANNEL` – The channel ID is invalid, or wakeup facility is not enabled by configuration.
- `GPT_E_INVALID_CORE` – The current core ID is invalid or different from the channel assignment core ID.

DEM errors

None

Description

This service disables the wakeup facility for the specified channel. If the driver is subsequently put into Sleep mode, this channel will be stopped and cannot wake up the microcontroller.

7.4.12 Gpt_EnableWakeup

Syntax

```
void Gpt_EnableWakeup
(
Gpt_ChannelType Channel
)
```

Service ID

0x0B

Parameters (in)

- `Channel` – ID of the GPT channel.

Parameters (out)

None

Return value

None

DET errors

- `GPT_E_UNINIT` – The driver is not initialized.
- `GPT_E_PARAM_CHANNEL` – The channel ID is invalid, or wakeup facility is not enabled by configuration.
- `GPT_E_INVALID_CORE` – The current core ID is invalid or different from the channel assignment core ID.

DEM errors

None

Description

This service enables the wakeup facility for the specified channel. If this channel is running when the driver is subsequently put into Sleep mode, the channel will not be stopped; when the channel reaches the target time, it will wake up the microcontroller.

7.4.13 Gpt_CheckWakeup

Syntax

```
void Gpt_CheckWakeup
(
EcuM_WakeupSourceType WakeupSource
)
```

Service ID

0x0C

Parameters (in)

- `WakeupSource` – Information on wakeup source to be checked. The associated GPT channel can be determined from the configuration data.

Parameters (out)

None

Return value

None

DET errors

- `GPT_E_INVALID_CORE` – The current core ID is invalid.
- `GPT_E_UNINIT` – The GPT driver is not initialized.

DEM errors

None

Description

Checks if a wakeup-capable GPT channel is the source for a wakeup event and calls the EcuM service `EcuM_SetWakeupEvent` in the case of a valid GPT channel wakeup event.

Note: This service only affected GPT channels assigned to the current core.

7.4.14 Gpt_GetPredefTimerValue

Syntax

```
Std_ReturnType Gpt_GetPredefTimerValue
(
Gpt_PredefTimerType PredefTimer,
uint32* TimeValuePtr
)
```

Service ID

0x0D

Parameters (in)

- `PredefTimer` – GPT Predef timer

Parameters (out)

- `TimeValuePtr` – Pointer to the time value destination data in RAM

Return value

`E_OK` or `E_NOT_OK`

DET errors

- `GPT_E_UNINIT` – The GPT driver is not initialized.
- `GPT_E_PARAM_POINTER` – The `TimeValuePtr` parameter is invalid (NULL).
- `GPT_E_PARAM_PREDEF_TIMER` – The GPT Predef timer passed by the parameter `PredefTimer` is not enabled.
- `GPT_E_MODE` – The mode of the driver is `GPT_MODE_SLEEP`.

DEM errors

None

Description

Delivers the current value of the desired GPT Predef timer.

7.4.15 Gpt_CheckPredefTimerStatus

Syntax

```
Std_ReturnType Gpt_CheckPredefTimerStatus
(
Gpt_DriverStateType* DriverStatusPtr,
Gpt_ChannelStateType* ChannelStatusPtr,
Gpt_PredefTimerType   PredefTimer
)
```

Service ID

0xFB

Parameters (in)

- `PredefTimer` – GPT Predef timer.

  When `GptPredefTimer1usEnablingGrade` is configured with `GPT_PREDEF_TIMER_1US_16_24_32BIT_ENABLED`, the GPT Predef timer with 1us tick duration can be checked by any of `GPT_PREDEF_TIMER_1US_16BIT`, `GPT_PREDEF_TIMER_1US_24BIT` or `GPT_PREDEF_TIMER_1US_32BIT` specified by the parameter PredefTimer. When `GptPredefTimer1usEnablingGrade` is configured with `GPT_PREDEF_TIMER_1US_16_24BIT_ENABLED`, the GPT Predef timer with 1us tick duration can be checked by any of `GPT_PREDEF_TIMER_1US_16BIT` or `GPT_PREDEF_TIMER_1US_24BIT` specified by the parameter PredefTimer.

Parameters (out)

- `DriverStatusPtr` – Specifies the location where the GPT's driver status information can be placed.
- `ChannelStatusPtr` – Specifies the location where the GPT Predef timer's channel status information can be placed.

Return value

`E_OK` or `E_NOT_OK`

DET errors

- `GPT_E_UNINIT` – The GPT driver is not initialized.
- `GPT_E_PARAM_POINTER` – The `DriverStatusPtr` or `ChannelStatusPtr` parameter is invalid (NULL).
- `GPT_E_PARAM_PREDEF_TIMER` – The GPT Predef timer passed by the parameter `PredefTimer` is not enabled.
- `GPT_E_INVALID_CORE` – The current core is not the master core.

DEM errors

None

Description

This service reads out internal registers and checks consistency of the software and hardware status.

Note: *`Gpt_CheckPredefTimerStatus()` may return `E_NOT_OK` if the hardware status has not yet changed to the running after the following APIs. It may occur when the tick frequency of the GPT timer is very slow.*

- `Gpt_Init()`
- `Gpt_SetMode(GPT_MODE_NORMAL)`
- `Gpt_SetPredefTimerPrescaler()`
- `Port_ActTrigger()` for the GPT Predef timer synchronous start

When `GptPredefTimerStartTriggerSelect` is configured, `Gpt_CheckPredefTimerStatus()` is called before calling `Port_ActTrigger()`; it returns `E_NOT_OK` because the trigger is not issued.

If this API returns `E_NOT_OK`, confirm that the synchronous start trigger is triggered (if used) and retry after waiting for one tick time of the Predef timer.

This service is available only for the master core.

7.4.16 Gpt_SetPrescaler

Syntax

```
void Gpt_SetPrescaler
(
Gpt_ChannelType Channel,
Gpt_ClkFrequencyType ClockFrequency
)
```

Service ID

0xFC

Parameters (in)

- `Channel` – ID of the GPT channel.
- `ClockFrequency` – Changed clock frequency.

Parameters (out)

None

Return value

None

DET errors

- `GPT_E_UNINIT` – The GPT driver is not initialized.
- `GPT_E_PARAM_CHANNEL` - The channel parameter does not correspond to a configured channel.
- `GPT_E_PARAM_CLOCK` – The `ClockFrequency` parameter is 0 or cannot realize a configured timer due to incorrect frequency.
- `GPT_E_BUSY` – Channel is already running.

- GPT_E_INVALID_CORE – The current core ID is invalid or different from the channel assignment core ID.

DEM errors

None

Description

Set a timer prescaler.

7.4.17 Gpt_SetPredefTimerPrescaler

Syntax

```
void Gpt_SetPredefTimerPrescaler
(
Gpt_PredefTimerType PredefTimer,
Gpt_ClkFrequencyType ClockFrequency
)
```

Service ID

0xFD

Parameters (in)

- PredefTimer – GPT Predef timer.

  When GptPredefTimer1usEnablingGrade is configured with GPT_PREDEF_TIMER_1US_16_24_32BIT_ENABLED, the pre-scaling value of the GPT Predef timer with 1us tick duration can be changed by any of GPT_PREDEF_TIMER_1US_16BIT, GPT_PREDEF_TIMER_1US_24BIT or GPT_PREDEF_TIMER_1US_32BIT specified by the parameter PredefTimer. When GptPredefTimer1usEnablingGrade is configured with GPT_PREDEF_TIMER_1US_16_24BIT_ENABLED, the pre-scaling value of the GPT Predef timer with 1us tick duration can be changed by any of GPT_PREDEF_TIMER_1US_16BIT or GPT_PREDEF_TIMER_1US_24BIT specified by the parameter PredefTimer.

- ClockFrequency – Changed clock frequency.

Parameters (out)

None

Return value

None

DET errors

- GPT_E_UNINIT – The GPT driver is not initialized.
- GPT_E_PARAM_PREDEF_TIMER – The GPT Predef timer passed by the parameter PredefTimer is not enabled
- GPT_E_PARAM_CLOCK – The ClockFrequency parameter is 0 or cannot realize a configured timer due to incorrect frequency.
- GPT_E_INTERNAL – PredefTimer is not yet running.
- GPT_E_INVALID_CORE – The current core is not the master core.

DEM errors

None

Description

This service changes a pre-scaling value of the selected Predef timer by the specified input clock source frequency.

Note: This service is available only from the master core.

7.4.18 Gpt_CheckChannelStatus

Syntax

```
Std_ReturnType Gpt_CheckChannelStatus
(
Gpt_DriverStateType*  DriverStatusPtr,
Gpt_ChannelStateType* ChannelStatusPtr,
Gpt_ChannelType       Channel
)
```

Service ID

0xFE

Parameters (in)

- `Channel` – ID of the GPT channel.

Parameters (out)

- `DriverStatusPtr` – Specifies the location where the GPT's driver status information can be placed.
- `ChannelStatusPtr` – Specifies the location where the GPT's channel status information can be placed.

Return value

`E_OK` or `E_NOT_OK`

DET errors

- `GPT_E_UNINIT` – The GPT driver is not initialized.
- `GPT_E_PARAM_CHANNEL` – The channel ID is not valid.
- `GPT_E_PARAM_POINTER` – The `DriverStatusPtr` and `ChannelStatusPtr` parameters are invalid (NULL).
- `GPT_E_INVALID_CORE` – The current core and channel assignment core are different.

DEM errors

None

Description

This service reads out internal registers and checks consistency of the software and hardware status.

Note: `Gpt_CheckChannelStatus()` may return `E_NOT_OK` if the hardware status has not yet changed to the running after `Gpt_StartTimer()`. It may occur when the tick frequency of the GPT timer is very slow.

Also in case of configured to use the external clock and called this API without clocking after `Gpt_StartTimer()`*, it returns* `E_NOT_OK`*.*
If this API returns `E_NOT_OK`*, confirm that the external clock is supplied (if used) and retry after waiting for one tick time of the GPT timer.*

7.5 Required callback functions

7.5.1 DET

If the development error detection is enabled, the GPT 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 the calling module.
- `InstanceId` – `GptCoreConfigurationId` of the core that calls this function or GPT_INVALID_CORE.
- `ApiId` – ID of the API service that calls this function.
- `ErrorId` – ID of the detected development error.

Return value

Always returns `E_OK` (is required for services).

Description

Service for reporting development errors.

7.5.2 Callout functions

7.5.2.1 Error callout API

The AUTOSAR GPT 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 `GptErrorCalloutFunction` Parameter.

Syntax

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

Reentrancy

Reentrant

Parameters (in)

- `ModuleId` – Module ID of the calling module.
- `InstanceId` – `GptCoreConfigurationId` of the core that calls this function or GPT_INVALID_CORE.
- `ApiId` – ID of the API service that calls this function.
- `ErrorId` – ID of the detected error.

Return value

None

Description

Service for reporting errors.

7.5.2.2 Get core ID API

The AUTOSAR GPT module requires a function to get valid core ID. This function is being used to determine from which core the code is getting executed. The name of the function to be called can be configured by `GptGetCoreIdFunction` parameter.

Syntax

`uint8 GetCoreID_Function_Name (void)`

Reentrancy

Reentrant

Parameters (in)

None

Return value

- `CoreId` - ID of the current core.

Description

Service for getting valid core ID.

Note: *This function shall return the core ID configured in the* `GptMulticore/GptCoreConfiguration/GptCoreId`*.*
For example: Two cores are configured in the `GptCoreConfiguration`*.*

Executing core	GptCoreConfigurationId	GptCoreId
CM7_0	0	15
CM7_1	1	16

- Upon calling this function from core CM7_0, it shall return 15.
- Upon calling this function from core CM7_1, it shall return 16.

8 Appendix B – Access register table

8.1 TCPWM

Table 10 TCPWM access register table

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	Gpt_Init Gpt_DeInit Gpt_StartTimer Gpt_StopTimer Gpt_SetMode Gpt_SetPrescaler Gpt_SetPredefTimerPrescaler	0xC737070F	0x*00*0000 (After Gpt_Init. Digit * depends on configuration value.) 0x00000000 (After Gpt_DeInit.)
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)	Counter value	Counter count register	Gpt_Init Gpt_DeInit Gpt_StartTimer Gpt_SetMode Gpt_SetPredefTimerPrescaler	0x00000000 (Monitoring is not needed.)	0x00000000 (Monitoring is not needed.)
CC0	31:0	Word (32 bits)	-	Counter compare/capture 0 register	Not used.	0x00000000 (Monitoring is not needed.)	0x00000000 (Monitoring is not needed.)
CC0_BUFF	31:0	Word (32 bits)	-	Counter buffered compare/capture 0 register	Not used.	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)	-	Counter compare/capture 1 register	Not used.	0x00000000 (Monitoring is not needed.)	0x00000000 (Monitoring is not needed.)
CC1_BUFF	31:0	Word (32 bits)	-	Counter buffered compare/capture 1 register	Not used.	0x00000000 (Monitoring is not needed.)	0x00000000 (Monitoring is not needed.)
PERIOD	31:0	Word (32 bits)	Period value	Counter period register	Gpt_Init Gpt_DeInit Gpt_StartTimer Gpt_SetMode Gpt_SetPredefTimerPrescaler	0x00000000 (Monitoring is not needed.)	0x00000000 (Monitoring is not needed.)
PERIOD_BUFF	31:0	Word (32 bits)	-	Counter buffered period register	Not used.	0x00000000 (Monitoring is not needed.)	0x00000000 (Monitoring is not needed.)
LINE_SEL	31:0	Word (32 bits)	-	Counter line selection register	Not used.	0x00000000 (Monitoring is not needed.)	0x00000000 (Monitoring is not needed.)
LINE_SEL_BUFF	31:0	Word (32 bits)	-	Counter buffered line selection register	Not used.	0x00000000 (Monitoring is not needed.)	0x00000000 (Monitoring is not needed.)
DT	31:0	Word (32 bits)	Divider value	Counter PWM dead time register	Gpt_Init Gpt_DeInit Gpt_SetPrescaler Gpt_SetPredefTimerPrescaler	0x000000FF	0x000000** (Digit * depends on configuration value and API.) 0x00000000 (After Gpt_DeInit.)

Register	Bit No.	Access size	Value	Description	Timing	Mask value	Monitoring value
TR_CMD	31:0	Word (32 bits)	0x00000000 | start command << 4 | stop command << 3	Counter trigger command register	Gpt_Init Gpt_DeInit Gpt_StartTimer Gpt_SetMode Gpt_SetPredefTimerPrescaler	0x00000000 (Monitoring is not needed.)	0x00000000 (Monitoring is not needed.)
TR_IN_SEL0	31:0	Word (32 bits)	0x00000000 | trigger input number << 8	Counter input trigger selection register 0	Gpt_Init Gpt_DeInit	0x0000FF00	0x0000**00 (After Gpt_Init. Digit * depends on configuration value.) 0x00000100 (After Gpt_DeInit.)
TR_IN_SEL1	31:0	Word (32 bits)	0x00000000 | trigger input number	Counter input trigger selection register 1	Gpt_Init Gpt_DeInit Gpt_StopTimer Gpt_SetMode Gpt_SetPredefTimerPrescaler	0x00000000 (Monitoring is not needed.)	0x00000000 (Monitoring is not needed.)
TR_IN_EDGE_SEL	31:0	Word (32 bits)	0x00000000 | start edge << 8 | count edge	Counter input trigger edge selection register	Gpt_Init Gpt_DeInit	0x0000030C	0x00000*0*(After Gpt_Init. Digit * depends on configuration value.) 0x0000030C (After Gpt_DeInit.)
TR_PWM_CTRL	31:0	Word (32 bits)	-	Counter trigger PWM control register	Not used.	0x00000000 (Monitoring is not needed.)	0x00000000 (Monitoring is not needed.)

Register	Bit No.	Access size	Value	Description	Timing	Mask value	Monitoring value
TR_OUT_SEL	31:0	Word (32 bits)	0x00000000 | trigger output number	Counter output trigger selection register	Gpt_Init Gpt_DeInit	0x00000077	0x000000** (After Gpt_Init. Digit * depends on configuration value.) 0x00000032 (After Gpt_DeInit.)
INTR	31:0	Word (32 bits)	0x00000000 | Interrupt request flag	Interrupt request register	Gpt_Init Gpt_DeInit Gpt_EnableNotification Gpt_DisableNotification Gpt_DisableWakeup Gpt_EnableWakeup GPT_Isr_Vector_<IRQ No>_Cat1 GPT_Isr_Vector_<IRQ No>_Cat2	0x00000000 (Monitoring is not needed.)	0x00000000 (Monitoring is not needed.)
INTR_SET	31:0	Word (32 bits)	-	Interrupt set request register	Not used.	0x00000000 (Monitoring is not needed.)	0x00000000 (Monitoring is not needed.)
INTR_MASK	31:0	Word (32 bits)	0x00000000 | Interrupt mask	Interrupt mask register	Gpt_Init Gpt_DeInit Gpt_EnableNotification Gpt_DisableNotification Gpt_EnableWakeup Gpt_DisableWakeup	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

Revision	Issue date	Description of change
**	2020-09-01	Initial release
*A	2020-11-19	Changed a memmap file include folder in chapter 2.6. MOVED TO INFINEON TEMPLATE.
*B	2021-05-18	Chapter 4.4 Updated note on GptChannelTickFrequency. Chapter 5.4 Added note on DeepSleep. Chapter 7.4.9 Added note on Gpt_DisableNotification.
*C	2021-08-18	Added a note on DeepSleep mode in 5.4 Sleep mode Added a note on Arm® errata in 6.3 Interrupts
*D	2021-12-07	Updated to the latest branding guidelines
*E	2023-10-06	Corrected core identification keyword in section 2.6 and 5.14.
*F	2023-12-08	Web release. No content updates.

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

Edition 2023-12-08

Published by

Infineon Technologies AG

81726 Munich, Germany

Do you have a question about this document?

Email:

erratum@infineon.com

Document reference

002-30198 Rev. *F

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.