MCAL PWM Module Software Design Document

Document Version : 98
Document Owner : Texas Instruments
Document Status : Published
Last Approval Date : Jun 30, 2022

1 1 Revision History

Version	Date	Author	Document Status	Comments
0.1	25 Apr 2022	Rakesh L	DONE	First version
0.2	25 May 2022	Rakesh L	DONE	Updated for review comments
v.98	22 Jun 2022	Rakesh L	DONE	Added Comala Workflow

2 2 Terms and Abbreviations

Abbreviation /Term	Meaning / Explanation
CS	Chip Select
DIO	Digital Input Output
ECU	Electric Control Unit
DMA	Direct Memory Access
ICU	Input Capture Unit
MISO	Master Input Slave Output
MMU	Memory Management Unit
MOSI	Master Output Slave Input

Abbreviation /Term	Meaning / Explanation
Master	A device controlling other devices (slaves, see below)
Slave	A device being completely controlled by a master device
NMI	Non Maskable Interrupt
OS	Operating System
PLL	Phase Locked Loop
PWM	Pulse Width Modulation
RX	Reception (in the context of bus communication)
SPAL	The name of this working group
SFR	Special Function Register

Abbreviation /Term	Meaning / Explanation
RTE	Run Time Environment
DET	Default Error Tracer – module to which errors are reported
DEM	Diagnostic Event Manager
SPI	Serial Peripheral Interface

3 3 Introduction

This document describes the functionality, API and configuration of the AUTOSAR BSW module PWM

- Supported AUTOSAR Release**: 4.3.1**
- Supported Configuration Variants**: Pre-Compile and Post-Build**
- Vendor ID**: PWM_VENDOR_ID (44)**
- Module ID**: PWM_MODULE_ID (121)**

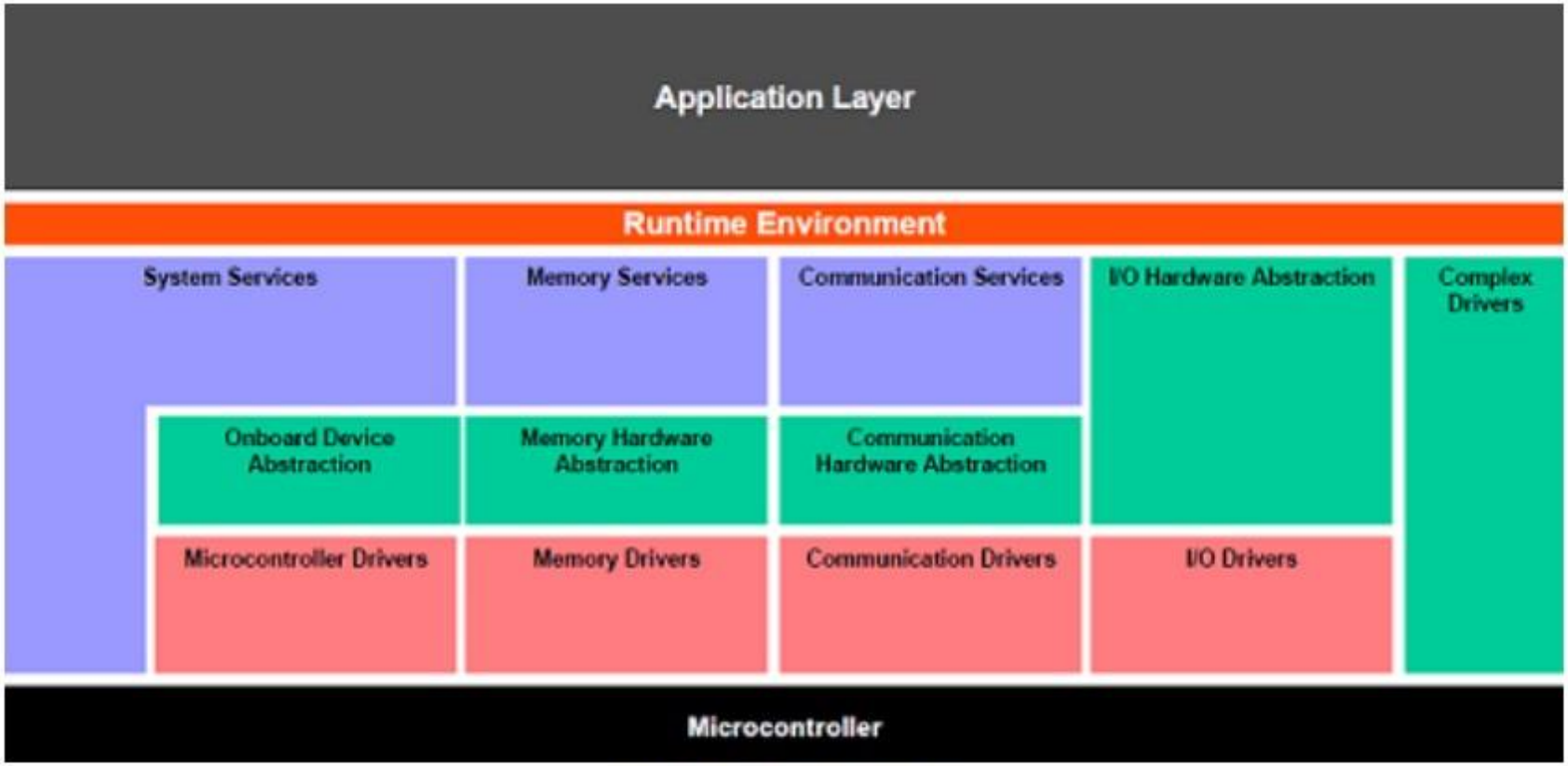
3.1 3.1 Overview

The figure below depicts the AUTOSAR layered architecture as 3 distinct layers,

- Application
- Runtime Environment (RTE) and
- Basic Software (BSW).

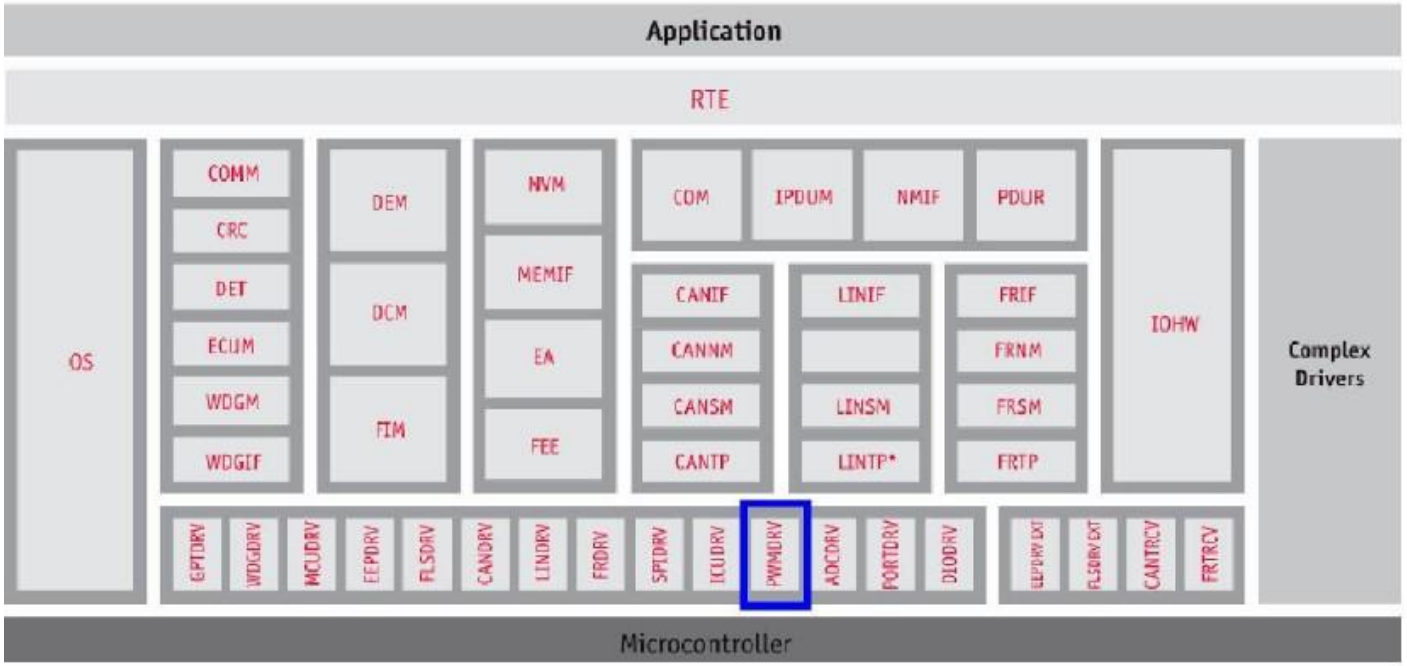
The BSW is further divided into 4 layers:

- Services
- Electronic Control Unit Abstraction • Micro Controller Abstraction (MCAL) and
- Complex Drivers.



AUTOSAR Architecture

MCAL is the lowest abstraction layer of the Basic Software. It contains software modules that interact with the Microcontroller and its internal peripherals directly. The PWM driver is a part of the microcontroller (peripheral) Driver module which is a part of the Basic Software. The figure below shows the position of the PWM driver in the AUTOSAR Architecture.



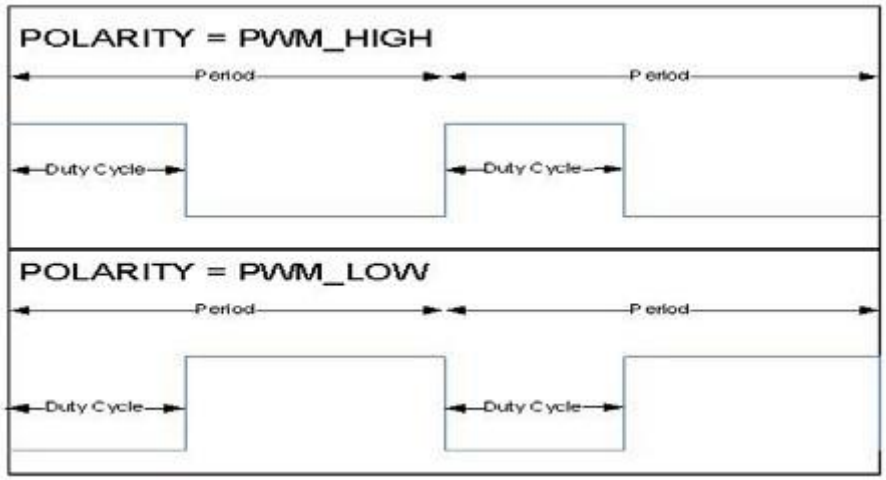
3.2 3.2 Purpose and Scope

The Detailed Design document provides the design details of PWM driver and aims to provide a guide to a design that could be implemented by a software developer. The scope of this document is to describe the software design procedure of PWM module.

3.3 3.3 Module Overview

The PWM module implements an interface in C programming language for handling the PWM functionality of the device.This PWM driver takes care of initializing and deinitializing the PWM unit and offers services to:

- Generate pulses with variable pulse width(Duty Cycle - can range from 0% to 100%)
- Set parameters of a PWM channels waveform(Duty Cycle and Period)
- Enable/Disable notifications
- It uses hardware IP "dmtimer_dmc1ms_10_rel.1.0.x" .Refer to SoC User Manual for specific details.



PWM Signal Description

3.4 **3.4 Hardware Overview**

In this design, the PWM Functionality can be achieved by using either of two available IPs, the DM Timers.

3.4.1 3.4.1 DM Timer

DM Timer available on the device are used. TI SoCs includes multiple timers, some of the key features provided are listed below:

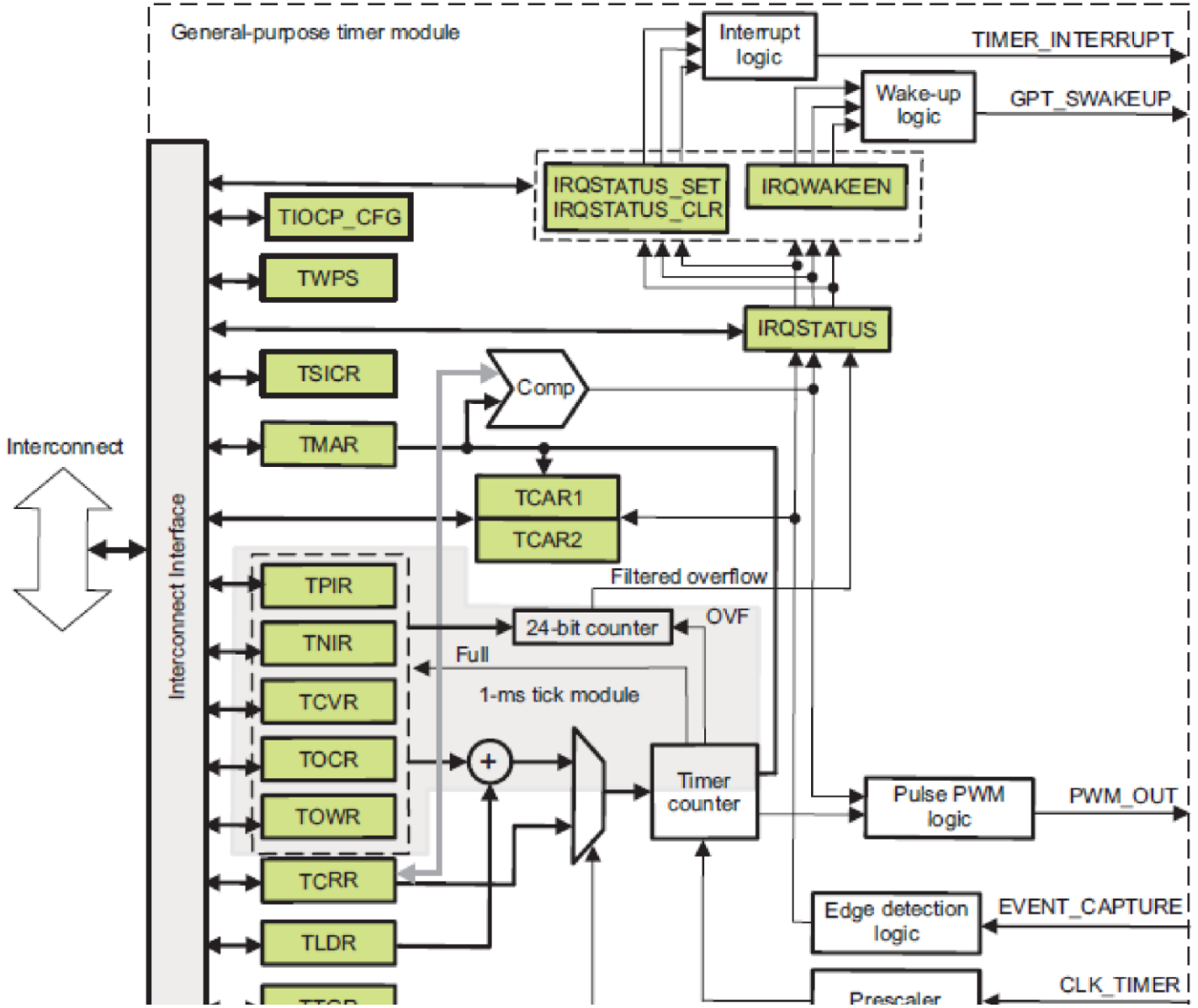
- Free running 32 bit up counter

- Auto reload mode (can be used for continuous counter operation)
- Support dynamic Start / Stop counter operation
- Programmable clock dividers (2n, where n = [0-8])
- 2 timers modules could be operated in cascaded mode to provide 64bit counter
- Programmable interrupt generation on overflow, compare and capture
- Programmable clock source

There is Support for 3 basic functional modes Timer mode, Capture mode & Compare mode.

Timer pins can be configured to be used as PWM output pins. However, not all timer pins can be configured to be PWM output.

The following image shows the block diagram of Timer which can generate a PWM output:


General-purpose timer module
Interrupt logic
TIMER_INTERRUPT
Wake-up logic
GPT_SWAKEUP
IRQSTATUS_SET
IRQSTATUS_CLR
IRQWAKEEN
TIOCP_CFG
TWPS
IRQSTATUS
TSICR
Comp
Interconnect
TMAR
Interconnect Interface
TCAR1
TCAR2
Filtered overflow
TPIR
24-bit counter
OVF
TNIR
Full
1-ms tick module
TCVR
TOCR
Timer counter
Pulse PWM logic
PWM_OUT
TOWR
TCRR
Edge detection logic
EVENT_CAPTURE
TLDR
Prescaler
CLK_TIMER

Timer Block Diagram

3.5 **3.5 Requirements**

The PWM Driver implements a standardized interface specified in the AUTOSAR_SWS_PWM Driver document Reference 1 - Autosar 4.3.1.

3.5.1 3.5.1 Features Supported

- Below listed are some of the key features that are expected to be supported
- Changing of frequency and duty cycle for a PWM channel at runtime besides the default configuration.
- The PWM signal that can be generated is a square wave with variable duty cycle and period .

Design Identifier	Description
MCAL-6772 - SWS_Pwm_00065 : PWM General File Structure PUBLISHED	SWS_Pwm_00065 : PWM General File Structure
MCAL-6808 - SWS_Pwm_00088 : PWM APIs Reentrancy PUBLISHED	SWS_Pwm_00088 : PWM APIs Reentrancy
MCAL-6845 - SWS_Pwm_00070 : PWM Time Unit PUBLISHED	SWS_Pwm_00070 : PWM Time Unit

Design Identifier	Description
MCAL-6883 - SWS_Pwm_00106 : Pwm_ChannelType PUBLISHED	SWS_Pwm_00106 : Pwm_ChannelType
MCAL-6763 - SWS_Pwm_00107 : Pwm_PeriodType PUBLISHED	SWS_Pwm_00107 : Pwm_PeriodType
MCAL-6775 - SWS_Pwm_00108 : Pwm_OutputStateType PUBLISHED	SWS_Pwm_00108 : Pwm_OutputStateType
MCAL-6836 - SWS_Pwm_00109 : Pwm_EdgeNotificationType PUBLISHED	SWS_Pwm_00109 : Pwm_EdgeNotificationType
MCAL-6796 - SWS_Pwm_00110 : Pwm_ChannelClassType PUBLISHED	SWS_Pwm_00110 : Pwm_ChannelClassType
MCAL-6806 - SWS_Pwm_00116 : Pwm_Init API : Function calls before init PUBLISHED	SWS_Pwm_00116 : Pwm_Init API : Function calls before init
MCAL-6837 - SWS_Pwm_00089 : PWM Module integrity check PUBLISHED	SWS_Pwm_00089 : PWM Module integrity check

Design Identifier	Description
MCAL-6837 - SWS_Pwm_00089 : PWM Module integrity check PUBLISHED	SWS_Pwm_00089 : PWM Module integrity check
MCAL-6826 - SWS_Pwm_00104 : Dem_ReportErrorStatus API PUBLISHED	SWS_Pwm_00104 : Dem_ReportErrorStatus API
MCAL-6809 - SWS_Pwm_00001 : PWM Emulation PUBLISHED	SWS_Pwm_00001 : PWM Emulation
MCAL-6845 - SWS_Pwm_00070 : PWM Time Unit PUBLISHED	SWS_Pwm_00070 : PWM Time Unit
MCAL-6826 - SWS_Pwm_00104 : Dem_ReportErrorStatus API PUBLISHED	SWS_Pwm_00104 : Dem_ReportErrorStatus API

3.5.2 3.5.2 Features Not Supported / NON Compliance

- **[NON Compliance]** APIs related to setting or getting power state of the device is not supported, as hardware itself doesn't support this feature.
- **[NON Compliance]** PwmMcuClockReferencePoint doesn't refer to McuClockReferencePoint.
- PWM_FIXED_PERIOD_SHIFTED Pwm Channel Class type is not supported
- Standard AUTOSAR PWM specification, categorizes few BSW General Requirements as non-requirements, please refer MCAL-3648 for details • Supports additional configuration parameters, refer section (Pwm_RegisterReadback)

3.6 **3.6 Assumptions**

Below listed are assumed to valid for this design/implementation, exceptions and other deviations are listed for each explicitly. Care should be taken to ensure these assumptions are addressed.

1. The functional clock to the PWM module is expected to be on before calling any PWM module API.
2. The PWM driver as such doesn't perform any PRCM programming to get the functional clock.
3. The clock-source selection for PWM is not performed by the PWM driver, other entities such as SBL, MCAL module MCU shall perform the same.

Design Identifier	Description
MCAL-6809 - SWS_Pwm_00001 : PWM Emulation PUBLISHED	SWS_Pwm_00001 : PWM Emulation

3.7 **3.7 Constraints**

Some of the critical constraints of this design are listed below

- In cases where MCU module is not employed (supported) to configure the clock source for PWM module . The PWM module configurator shall refer to MCU clock source as listed in specification.
- The PWM SWS does not cover PWM emulation on general purpose I/O

- Timer when configured for toggle mode with TCLR[11-10] TRG = 0x2 (overflow and match), and TCLR[7] SCPWM Bit = 1 for polarity Low, the first event that toggles the PWM line is an overflow event. If a match event occurs first,it does not toggle the PWM line

3.7.1 3.7.1 PWM

Setting Prescalers: Period value(Num of ticks) which determines the period is calculated using below formula:

- PRD = (TBCLK/PWM_FREQ) / 2.
- TBCLK – Time base clock relative to the system clock.
- PWM_FREQ – Required output frequency.
- /2 – Because UpDown counter is selected.(Please refer DAR for more details). The TRM gives details on this formula.
- TBCLK is derived from below formula.
- TBCLK = SYS_CLK / (HSPCLKDIV x CLKDIV)
- SYS_CLK – System Clock to the PWM which is 125Mhz.
- CLKDIV – Time-base Clock Prescale Bits. These bits determine part of the time-base clock prescale value.

Actual Division Value = / (1 << CLKDIV) i.e

Actual Division Value	CLKDIV Value
0x0	/ 1

Actual Division Value	CLKDIV Value
0x1	/ 2
0x2	/ 4
0x3	/ 8
0x4	/ 16
0x5	/ 32
0x6	/ 64
0x7	/ 128

HSPCLKDIV – High-Speed Time-base Clock Prescale Bits. These bits determine part of the time-base clock prescale value.

Actual Division Value = / (HSPCLKDIV * 2) i.e

Actual Division Value	HSPCLKDIV
0x0	/ 1
0x1	/ 2
0x2	/ 4
0x3	/ 6
0x4	/ 8
0x5	/ 10
0x6	/ 12
0x7	/ 14

There is a chance that period value calculated is fractional, and therefore required frequency cannot be achieved. The clock dividers need to be selected correctly to avoid such errors. Please refer to the excel sheet provided to derive the correct CLKDIV and HSPCLKDIV values. Using this excel, you can test prescaler combinations with required frequency and ensure that the "Difference" is 0. The Period value should be less than or equal to 65535 (16bit HW register size for period).

3.8 3.8 Hardware and SW platforms

Hardware Platforms

- Refer to specified SoC User Manual to check if PWM module is supported. **Software Platforms**
- Bare-Metal

3.9 3.9 Dependencies

PWM driver shall depend on these modules to realize the required functionality. PWM uses Timer hardware present in the device to realize the functionality, this peripheral requires 2 different clock to be operational, namely ICLK and FCLK.

3.9.1 3.9.1 SBL

- **ICLK**: Interface clock required for internal operation of the peripheral. This is not expected to change and typically programmed by SBL, please refer the device specific manual for details and valid value.
- **FCLK**: Functional clock, used to drive the counter of the timer module. As per AUTOSAR PWM module specification Reference 1 - Autosar 4.3.1. the PWM driver shall refer MCU Clock reference point. Other entity would require to select the right clock source for the peripheral.

3.10 3.10 Stakeholders

- Developers
- Test Engineers
- Customer Integrator

3.11 3.11 References

	Specification	Comment/Link
1	AUTOSAR 4.3.1	AUTOSAR_SWS_PWMDriver
2	BSW General Requirements / Coding guidelines	Autosar and Coding guidelines for the Mcal drivers.
3	Software Product Specification (SPS)	Product Functional requirements.
4	Software Architecture	Mcal Software Architecture.

4 4 Design Description

Refer AUTOSAR Specification mentioned in Reference 1 - Autosar 4.3.1 section 1.4 for concepts such as channel, job, sequences.

4.1 4.1 Fundamental Operation

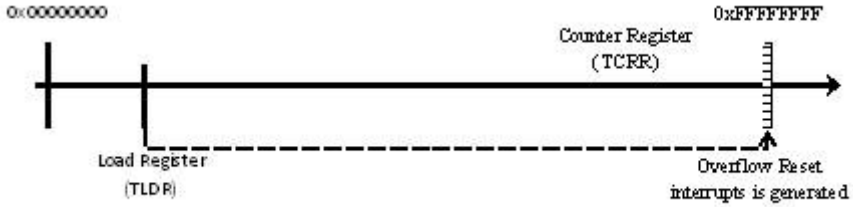
4.1.1 4.1.1 Timer

The timer can be configured to provide a programmable PWM output. The timer PWM (POTIMERPWM) output pin can be configured to toggle on an event. The TCLR[11-10] TRG bit field determines on which register value the PWM pin toggles. Either overflow or both overflow and match can be selected to toggle the timer PWM pin when a compare condition occurs. TMAR, TLDR The internal overflow pulse is set each time the (0xFFFF FFFF – TLDR[31-0] LOAD_VALUE + 1) value is reached, and the internal match pulse is set when the counter reaches the value of TMAR. Depending on the value of the TCLR[12] PT bit and TCLR[11-10] TRG bit field, the timer provides pulse or PWM event on the output

pin (POTIMERPWM). The TLDR and TMAR must keep values below the overflow value (0xFFFF FFFF) by at least two units. If the PWM trigger events are both overflow and match, the difference between the values kept in the TMAR and the value in the TLDR must be at least two units. When match event is used, the compare mode TCLR[6] CE bit must be set.

The following sequence of steps needs to be performed, before the timer can be started

- Timer is programmed as configured (in continuous mode or Autoreload mode)
- The initial count of the counter TCLR needs to be loaded
- The reload register (TLDR) needs to be loaded
- The timer is started and the counter (register TCRR), starts counting on every pulse
- As depicted in above figure, TCRR has moved w.r.t to TLDR



Timer expired

- When the timer expires, the TCRR is loaded with value present in TLDR as show above
- An interrupt can be triggered at this point

For generating a PWM signal from the timer interrupts, the following cases w.r.t. duty cycle are considered:

Case 1: Duty cycle of 0%

- The output will be the inverse of the configured polarity parameter. For example, if the polarity is configured as LOW, the PWM signal will always be HIGH.

CASE 2: Duty cycle of 100%

- The output will be equal to the configured polarity parameter. For example, if the polarity is configured as LOW, the PWM signal will always be LOW.

Case 3: Duty cycle of 50%

- When the When a duty cycle of 50% for a PWM signal is required, both the time that the pulse remains HIGH and LOW is equal. In this case, the timer is operated in overflow mode alone (without compare). In one cycle of the PWM signal, the timer should generate an interrupt twice. Hence, the overflow rate for the timer (OVF_Rate) is half the period of the PWM signal required. The load value for TLDR register can be calculated using the following formula:

```
OVF_Rate = (0xFFFF FFFF – TLDR + 1) × (timer-functional clock period) × PS
```

where

- OVF_Rate = (PWM Period /2)
- Timer-functional clock period is the period of the input clock to the timer
- PS is the Prescaler Clock Ratio Value used to divide the timer counter input clock frequency as shown below:

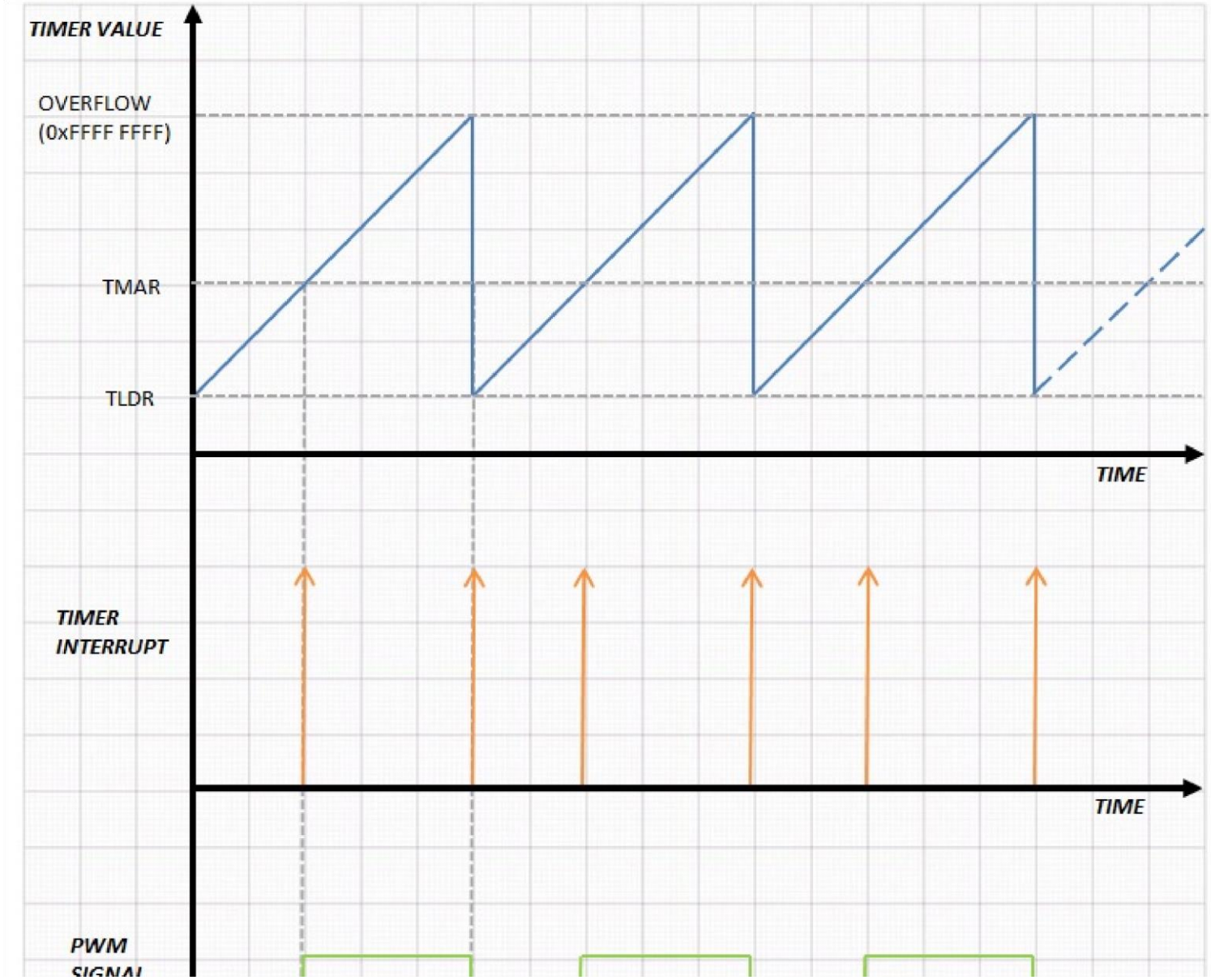
Prescaler Clock Ratio Values

TCLR[5] PRE	TCLR[4-2] PTV	Divisor (PS)
0	X	1
1	0	2
1	1	4
1	2	8
1	3	16
1	4	32

Prescaler Clock Ratio Values

Case 4: 0%< Duty cycle < 100%

When PWM signal with 0%< duty cycle < 100% for a is required, the timer is operated in overflow and match mode (with compare). In one cycle of the PWM signal, the timer generates an interrupt twice, one when the match condition is met and one when overflow is triggered as shown in the figure below.Hence, in this scenario, the overflow rate for the timer (OVF_Rate) is same as the period of the PWM signal required.


TIMER VALUE
OVERFLOW
(0xFFFF FFFF)
TMAR
TLDR
TIME
TIMER
INTERRUPT
TIME
PWM

Generation of PWM signal using Timer The

load value for TLDR and TMAR registers can be calculated using the following formula:

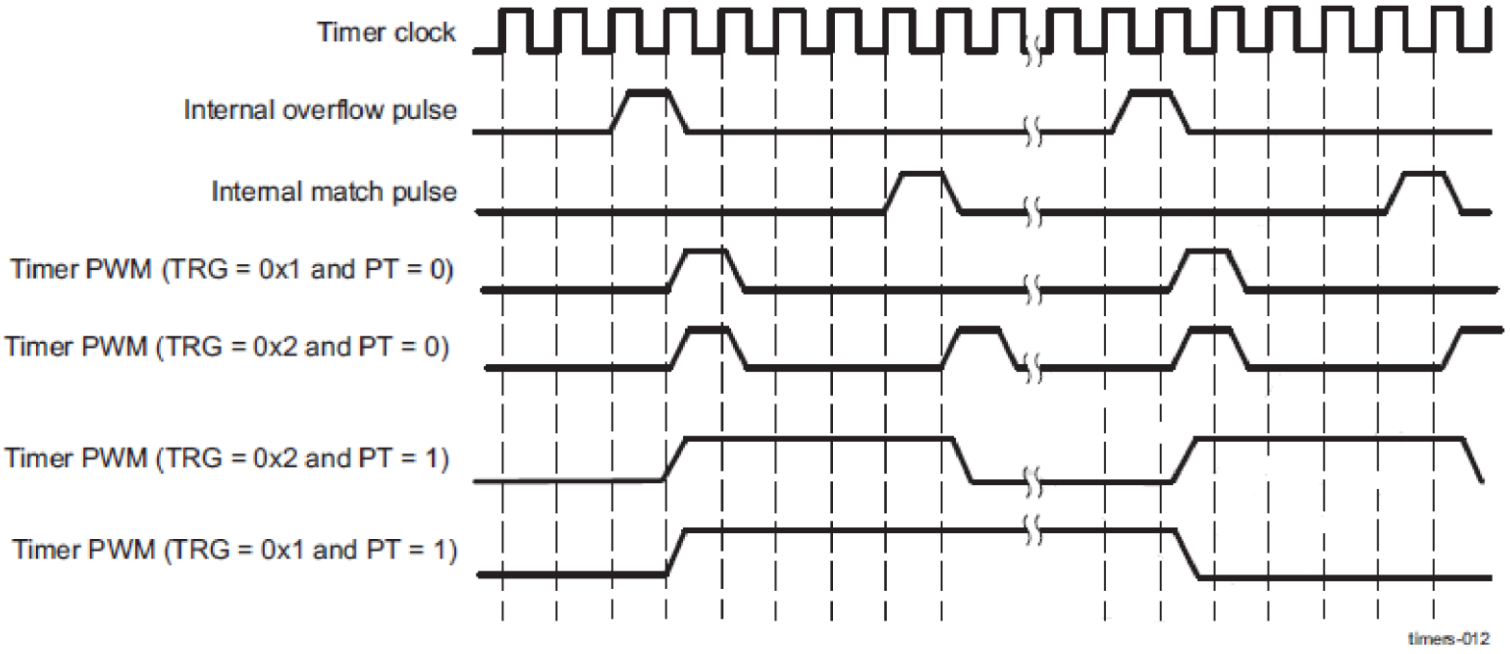
```
OVF_Rate = (0xFFFF FFFF – TLDR + 1) × (timer-functional clock period) × PS
OVF_Rate * (1- (Dutycyle/100)) = (0xFFFF FFFF – TMAR + 1) × (timer-functional clock period) × PS
```

where

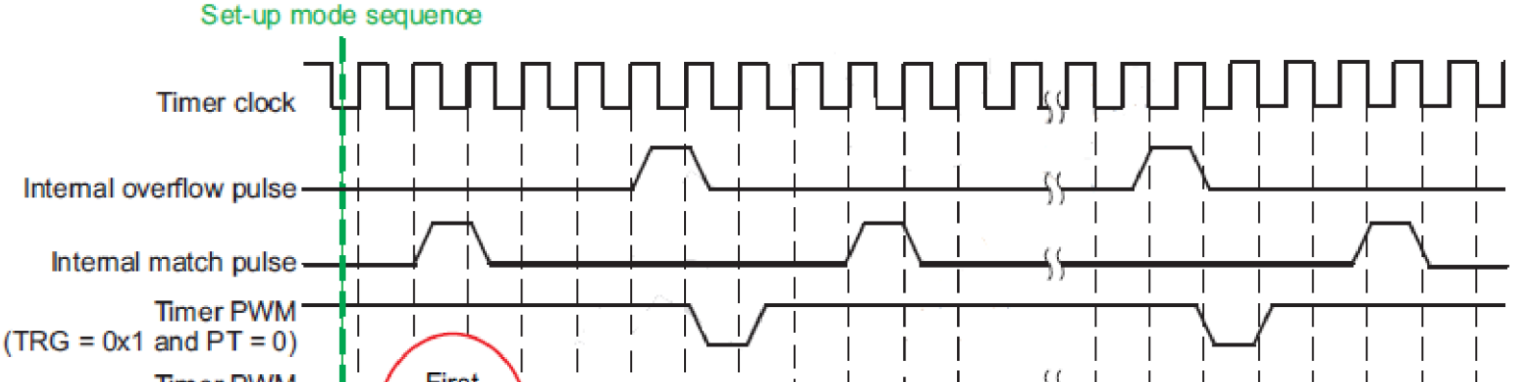
- OVF_Rate = (PWM Period)
- Timer-functional clock period is the period of the input clock to the timer
- PS is the Prescaler Clock Ratio Value.

Two example cases are shown in the following figure. The TCLR[7] SCPWM bit is set to 0 in one case and to 1 in the other case. To obtain the desired wave form, start the counter at 0xFFFF FFFE value (to ensure an overflow first) or adjust the line polarity (TCLR[7] SCPWM bit).

Timing Diagram of PWM With TCLR[7] SCPWM Bit = 0



Timing Diagram of PWM With TCLR[7] SCPWM Bit = 1



Timing Diagram for PWM through Timer

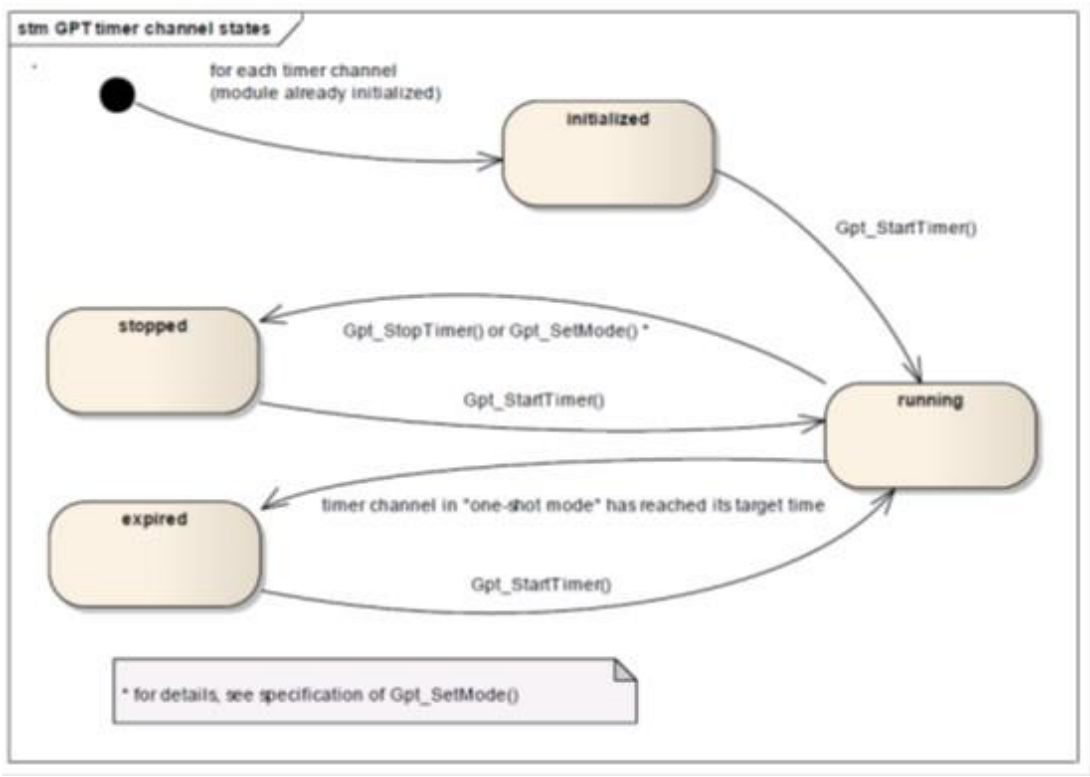
NOTE

- The Functional frequency range dictates the usage of posted and non-posted write modes for the timer. Incase of posted mode of operation, If a write access is pending for a register, reading from this register does not yield a correct result. Software synchronization must be used to avoid incorrect results. Functional frequency range: freq(timer clock) < freq(interface clock)/4.

4.2 **4.2 Dynamic Behavior**

4.2.1 4.2.1 States for Timer

As the PWM module is implemeted using a timer, as detailed in section 7.1 of 5, a timer would be in one of the following states. Initialized, running, stopped, expired. The diagram below shows transitions of states:



Timer States : Sourced from TIMER AUTOSAR Spec

4.3 4.3 Time Unit Ticks

Refer Reference 1 - Autosar 4.3.1. specifically section 7.2 of the specification for more details All time units used within the API services of the PWM module shall be of the unit ticks.

Design Identifier	Description
MCAL-6845 - SWS_Pwm_00070 : PWM Time Unit PUBLISHED	SWS_Pwm_00070 : PWM Time Unit

4.4 4.4 Duty Cycle Resolution and Scaling

Refer Reference 1 - Autosar 4.3.1. specifically section 7.7 of the specification for more details The width of the duty cycle parameter is 16 Bits and the parameter follows the following scaling scheme:

- 0x0000 means 0%.
- 0x8000 means 100%. 0x8000 gives the highest resolution while allowing 100% duty cycle to be represented with a 16 bit value.

As an implementation guide, the following source code example is given:

```
AbsoluteDutyCycle =((uint32)AbsolutePeriodTime * RelativeDutyCycle) >> 15;
```

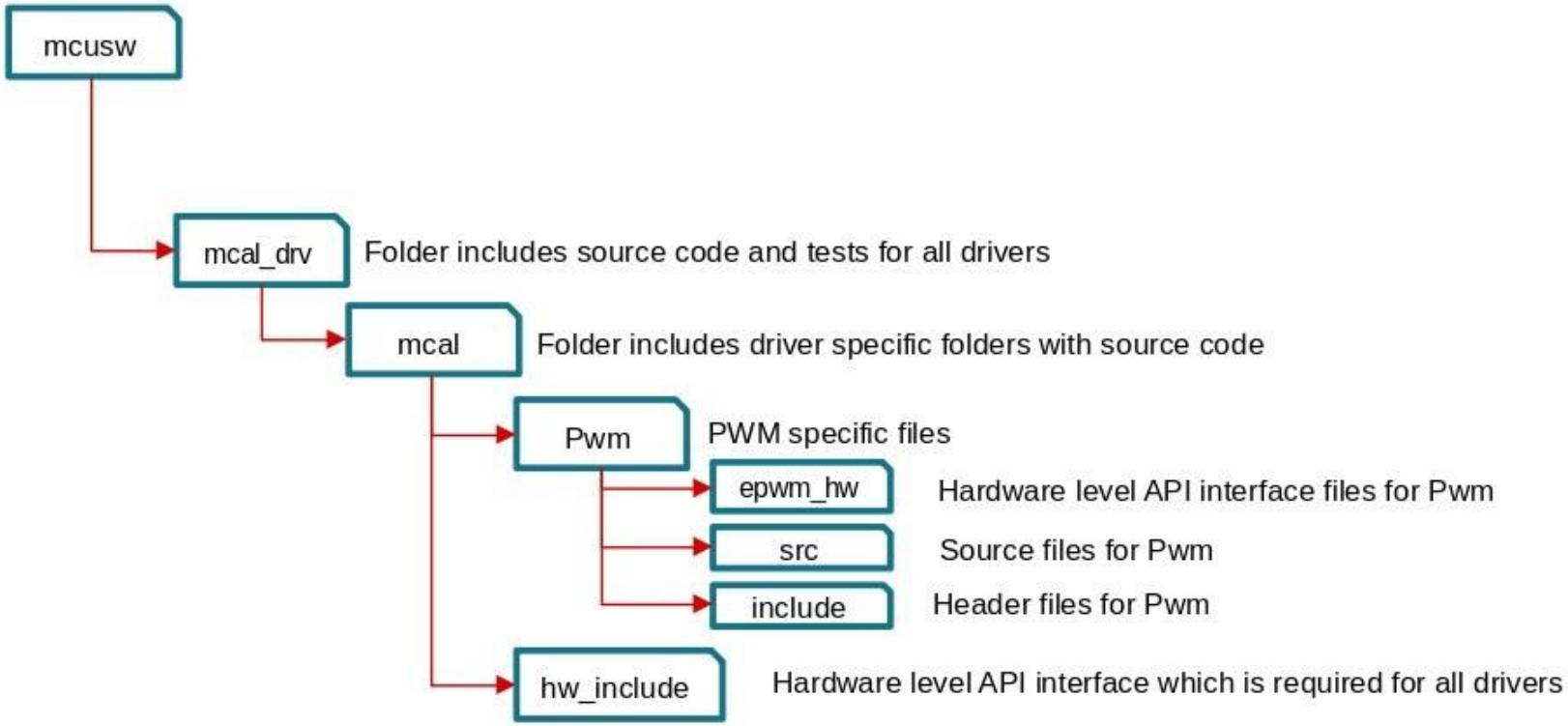
Design Identifier	Description
MCAL-6801 - SWS_Pwm_00058 : PWM Duty cycle Parameter Width PUBLISHED	SWS_Pwm_00058 : PWM Duty cycle Parameter Width
MCAL-6880 - SWS_Pwm_00059 : PWM Duty Cycle Scaling Scheme PUBLISHED	SWS_Pwm_00059 : PWM Duty Cycle Scaling Scheme

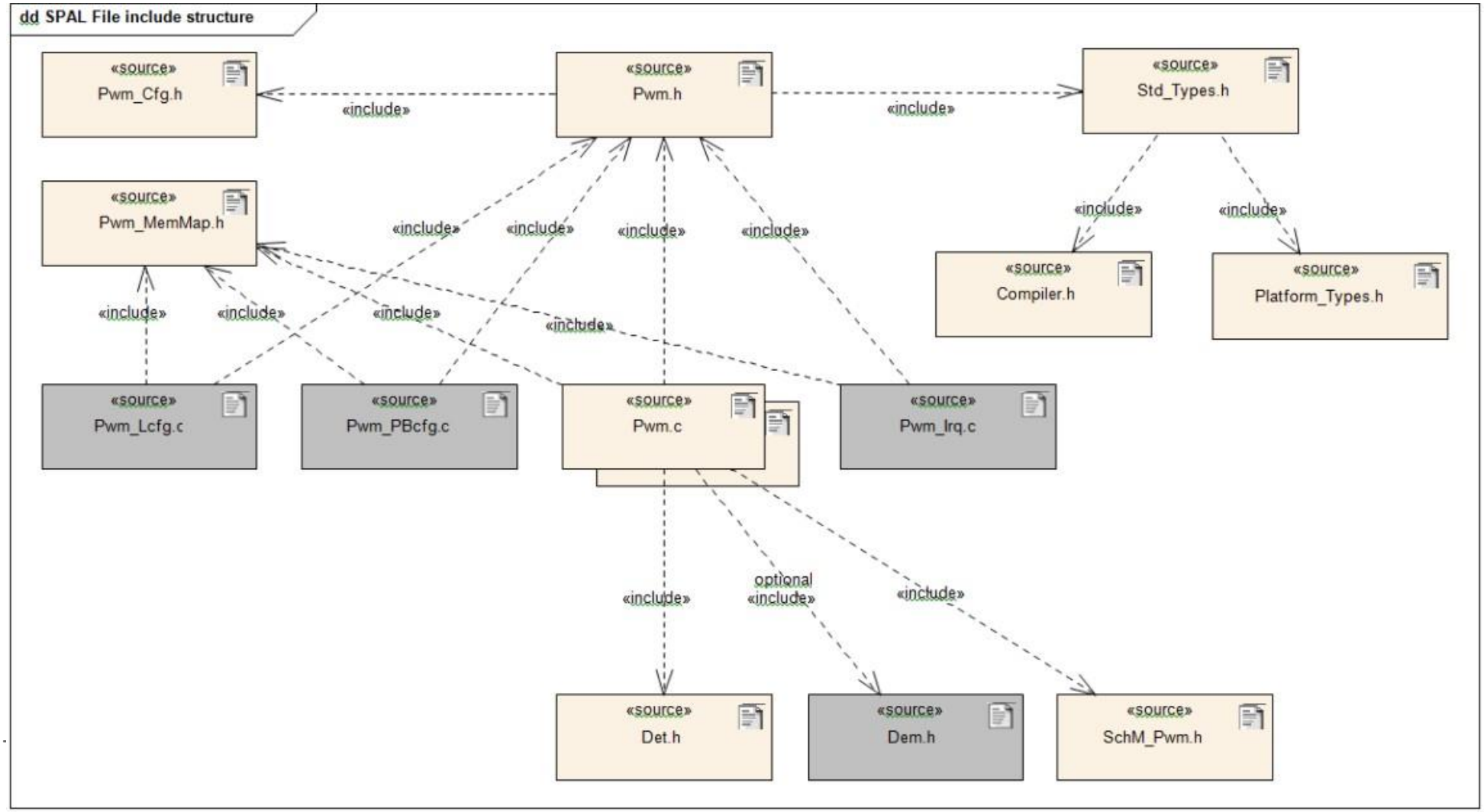
4.5 **4.5 Directory Structure**

The directory structure is as depicted in figures below, the source files can be categorized under “Driver Implementation” and “Example Application” **Driver**

Implemented by

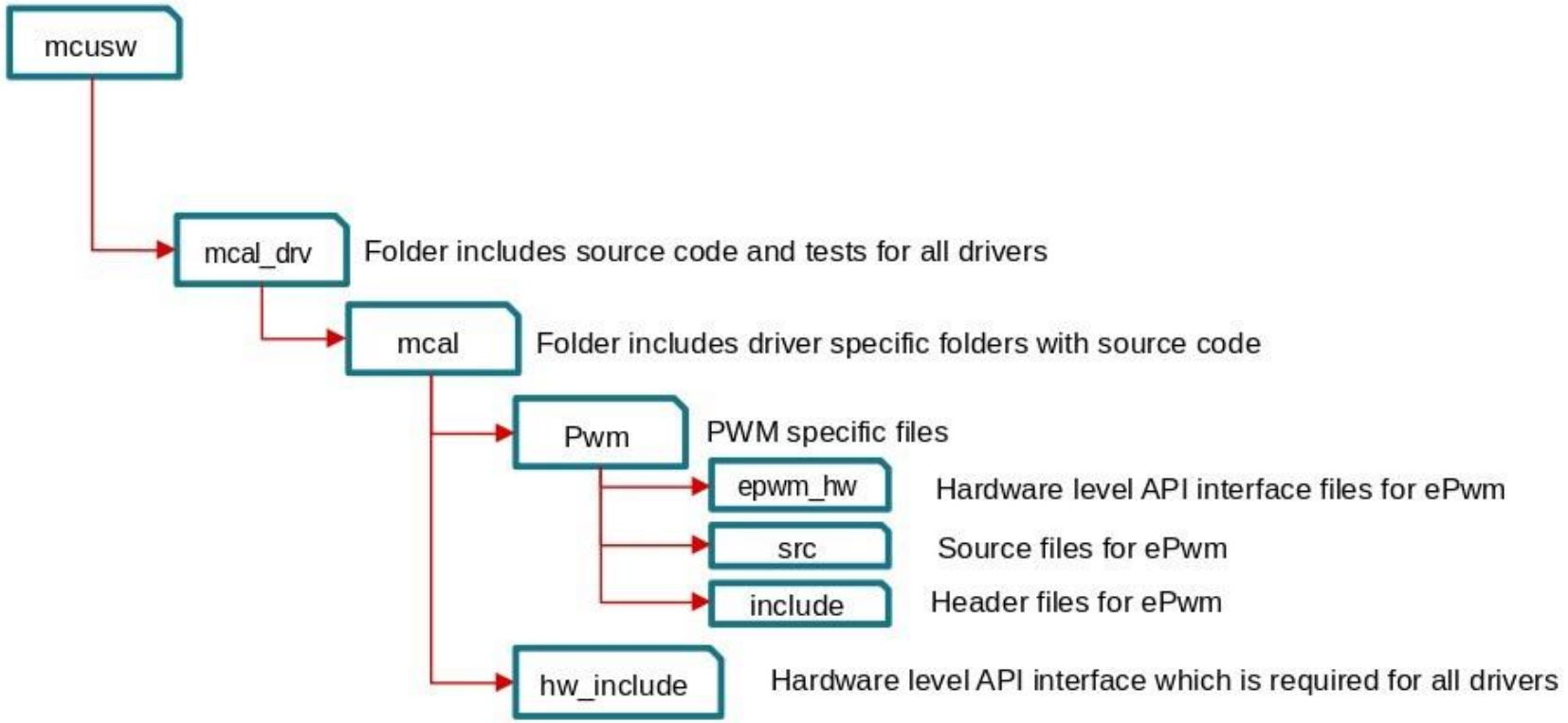
- Pwm.h and Pwm_Irq.h: Shall implement the interface provided by the driver
- Pwm.c, Pwm_Irq.c, Pwm_Gptimer.c, Pwm_Ehrpwm.c Pwm_Priv.c and Pwm_Priv.h: Shall implement the driver functionality • Pwm_Priv.c will act as an abstraction layer. Based on the IP configuration chosen, the respective API will be called.
- Gptimer specific API are present in Pwm_Gptimer.c.
- hw_include : Low level API, Shall be used by drivers.


mcusw
mcal_drv
Folder includes source code and tests for all drivers
mcal
Folder includes driver specific folders with source code
Pwm
PWM specific files
epwm_hw
Hardware level API interface files for Pwm
src
Source files for Pwm
include
Header files for Pwm
hw_include
Hardware level API interface which is required for all drivers


dd SPAL File include structure
«source» Pwm_Cfg.h
«include»
«source» Pwm.h
«include»
«source» Std_Types.h
«source» Pwm_MemMap.h
«include»
«include»
«include»
«include»
«include»
«include»
«source» Compiler.h
«include»
«source» Platform_Types.h
«include»
«include»
«include»
«include»
«source» Pwm_Lcfg.c
«source» Pwm_PBcfg.c
«source» Pwm.c
«source» Pwm_Irq.c
«include»
optional «include»
«include»
«source» Det.h
«source» Dem.h
«source» SchM_Pwm.h

Example Application

- Pwm_Cfg.h and Pwm_Lcfg.c: Shall implement the generated configuration for link-time variant
- Pwm_PBcfg.c : Shall implement the generated configuration for post-build variant
- PwmApp.c: Shall implement the example application that demonstrates the use of the driver.
- hw_include : Shall be used by example application.


mcusw
mcal_drv
Folder includes source code and tests for all drivers
mcal
Folder includes driver specific folders with source code
Pwm
PWM specific files
epwm_hw
Hardware level API interface files for ePwm
src
Source files for ePwm
include
Header files for ePwm
hw_include
Hardware level API interface which is required for all drivers

Design Identifier	Description
MCAL-6772 - SWS_Pwm_00065 : PWM General File Structure PUBLISHED	SWS_Pwm_00065 : PWM General File Structure
MCAL-6778 - SWS_Pwm_60075 : PWM File Structure PBCfg.h PUBLISHED	SWS_Pwm_60075 : PWM File Structure PBCfg.h
MCAL-6824 - SWS_Pwm_50075 : PWM File Structure Det.h PUBLISHED	SWS_Pwm_50075 : PWM File Structure Det.h
MCAL-6888 - SWS_Pwm_10075 : PWM File Structure Cfg.h PUBLISHED	SWS_Pwm_10075 : PWM File Structure Cfg.h

Design Identifier	Description
MCAL-6769 - SWS_Pwm_70075 : PWM File Structure Pwm_Irq.h PUBLISHED	SWS_Pwm_70075 : PWM File Structure Pwm_Irq.h
MCAL-6820 - SWS_Pwm_40075 : PWM File Structure MemMap.h PUBLISHED	SWS_Pwm_40075 : PWM File Structure MemMap.h

4.6 **4.6 Configurator**

The AUTOSAR PWM Driver Specification details mandatory parameters that shall be configurable via the configurator. Please refer section 10 of Reference 1 - Autosar 4.3.1.

Design Identifier	Description
MCAL-6770 - ECUC_Pwm_00004 : PwmGeneral PUBLISHED	ECUC_Pwm_00004 : PwmGeneral

Design Identifier	Description
MCAL-6876 - ECUC_Pwm_00131 : PwmDevErrorDetect PUBLISHED	ECUC_Pwm_00131 : PwmDevErrorDetect
MCAL-6791 - ECUC_Pwm_00132 : PwmDutycycleUpdatedEndperiod PUBLISHED	ECUC_Pwm_00132 : PwmDutycycleUpdatedEndperiod
MCAL-6792 - ECUC_Pwm_00139 : PwmIndex PUBLISHED	ECUC_Pwm_00139 : PwmIndex
MCAL-6819 - ECUC_Pwm_00133 : PwmNotificationSupported PUBLISHED	ECUC_Pwm_00133 : PwmNotificationSupported
MCAL-6869 - ECUC_Pwm_00134 : PwmPeriodUpdatedEndperiod PUBLISHED	ECUC_Pwm_00134 : PwmPeriodUpdatedEndperiod
MCAL-6866 - ECUC_Pwm_00027 : PwmChannel PUBLISHED	ECUC_Pwm_00027 : PwmChannel

Design Identifier	Description
MCAL-6811 - ECUC_Pwm_00136 : PwmChannelClass PUBLISHED	ECUC_Pwm_00136 : PwmChannelClass
MCAL-6868 - ECUC_Pwm_00137 : PwmChannelId PUBLISHED	ECUC_Pwm_00137 : PwmChannelId
MCAL-6790 - ECUC_Pwm_00138 : PwmDutycycleDefault PUBLISHED	ECUC_Pwm_00138 : PwmDutycycleDefault
MCAL-6825 - ECUC_Pwm_00122 : PwmIdleState PUBLISHED	ECUC_Pwm_00122 : PwmIdleState
MCAL-6860 - ECUC_Pwm_00123 : PwmNotification PUBLISHED	ECUC_Pwm_00123 : PwmNotification
MCAL-6844 - ECUC_Pwm_00124 : PwmPeriodDefault PUBLISHED	ECUC_Pwm_00124 : PwmPeriodDefault

Design Identifier	Description
MCAL-6855 - ECUC_Pwm_00125 : PwmPolarity PUBLISHED	ECUC_Pwm_00125 : PwmPolarity
MCAL-6758 - ECUC_Pwm_00140 : PwmChannelConfigSet PUBLISHED	ECUC_Pwm_00140 : PwmChannelConfigSet
MCAL-6873 - ECUC_Pwm_00126 : PwmConfigurationOfOptApiServices PUBLISHED	ECUC_Pwm_00126 : PwmConfigurationOfOptApiServices
MCAL-6829 - ECUC_Pwm_00128 : PwmSetDutyCycle PUBLISHED	ECUC_Pwm_00128 : PwmSetDutyCycle
MCAL-6829 - ECUC_Pwm_00128 : PwmSetDutyCycle PUBLISHED	ECUC_Pwm_00129 : PwmSetOutputToIdle

Design Identifier	Description
MCAL-6782 - ECUC_Pwm_00130 : PwmSetPeriodAndDuty PUBLISHED	ECUC_Pwm_00130 : PwmSetPeriodAndDuty
MCAL-6781 - ECUC_Pwm_00135 : PwmVersionInfoApi PUBLISHED	ECUC_Pwm_00135 : PwmVersionInfoApi
MCAL-6859 - TI Specific ECUC :PwmOutputChSelect PUBLISHED	TI Specific ECUC :PwmOutputChSelect

4.6.1 4.6.1 NON Standard configurable parameters

The design's specific configurable parameters are as follows:

Parameter	Usage comment
PwmDeviceVariant	This shall allow integrators to select the device variant for which integration is being performed. This parameter shall be used by driver to impose device specific constraints. The user guide shall detail the device specific constraints
PwmFunctionalClock	This is the value of the System clock frequency in Hz
PwmTypeofInterruptFunction	This parameter allows the selection of Type of ISR function
PwmEnableRegisterReadbackApi	This parameter enables the API to readback PWM critical registers
PwmClkPrescaler	This parameter allows the selection of pre-scalar value.The prescaler stage is clocked with the pwm clock and acts as a clock divider for the time-base clock.
PwmDefaultOSCounterId	This parameter stores Default Os Counter Id if node reference to OsCounter ref PwmOsCounterRef is not set
PwmOsCounterRef	This parameter contains a reference to the OsCounter, which is used by the Timer driver

Parameter	Usage comment
PwmTimeoutDuration	This parameter contains the Timer timeout upper limit

Design Identifier	Description
MCAL-6849 - TI Specific ECUC :PwmFunctionalClock PUBLISHED	TI Specific ECUC :PwmFunctionalClock
MCAL-6761 - TI Specific ECUC : PwmClkPrescaler PUBLISHED	TI Specific ECUC : PwmClkPrescaler
MCAL-6853 - TI Specific ECUC : PwmHSClkPrescaler PUBLISHED	TI Specific ECUC : PwmHSClkPrescaler

Design Identifier	Description
MCAL-6797 - TI Specific ECUC : PwmDeviceVariant PUBLISHED	TI Specific ECUC : PwmDeviceVariant
MCAL-6750 - TI Specific ECUC : PwmDefaultOSCounterId PUBLISHED	TI Specific ECUC : PwmDefaultOSCounterId
MCAL-6858 - TI Specific ECUC: PwmOsCounterRef PUBLISHED	TI Specific ECUC: PwmOsCounterRef
MCAL-6821 - TI Specific ECUC: PwmTimeoutDuration PUBLISHED	TI Specific ECUC: PwmTimeoutDuration
MCAL-6874 - TI Specific ECUC : PwmTypeofInterruptFunction PUBLISHED	TI Specific ECUC : PwmTypeofInterruptFunction

4.6.2 4.6.2 Variant Support

The driver shall support both VARIANT-POST-BUILD & VARIANT-PRE-COMPILE

Design Identifier	Description
MCAL-6755 - SWS_Pwm_00077 : PWM Module VARIANT-POST-BUILD PUBLISHED	SWS_Pwm_00077 : PWM Module VARIANT-POST-BUILD

4.7 **4.7 Error Classification**

Errors are classified in two categories, development error and runtime / production error.

4.7.1 4.7.1 Development Errors

Type of Error	***Related Error code***	***Value (Hex)***
API Pwm_Init service called with wrong configuration	PWM_E_INIT_FAILED	0x10

API service used without module initialization	PWM_E_UNINIT	0x11
API service used with an invalid channel Identifier	PWM_E_PARAM_CHANNEL	0x12
Usage of unauthorized PWM service on PWM channel configured a fixed period	PWM_E_PERIOD_UNCHANGEABLE	0x13
API Pwm_Init service called while the PWM driver has already been initialised	PWM_E_ALREADY_INITIALIZED	0x14
API Pwm_GetVersionInfo is called with a NULL parameter.	PWM_E_PARAM_POINTER	0x15

4.7.2 4.7.2 Runtime Errors

Type of Error	***Related Error code***	***Value (Hex)***
API Pwm_SetPowerState is called while the PWM module is still in use	PWM_E_NOT_DISENGAGED	0x16

Design Identifier	Description
MCAL-6768 - SWS_Pwm_10002 : PWM_Init API: Developement Error Detection PWM_E_PARAM_CONFIG PUBLISHED	SWS_Pwm_10002 : PWM_Init API: Developement Error Detection PWM_E_PARAM_CONFIG
MCAL-6833 - SWS_Pwm_20002 : PWM Without Init : Developement Error Detection PWM_E_UNINIT PUBLISHED	SWS_Pwm_20002 : PWM Without Init : Developement Error Detection PWM_E_UNINIT

Design Identifier	Description
MCAL-6872 - SWS_Pwm_30002 : PWM Developement Error Detection PWM_E_PARAM_CHANNEL PUBLISHED	SWS_Pwm_30002 : PWM Developement Error Detection PWM_E_PARAM_CHANNEL
MCAL-6802 - SWS_Pwm_40002 : PWM Developement Error Detection PWM_E_PERIOD_UNCHANGEABLE PUBLISHED	SWS_Pwm_40002 : PWM Developement Error Detection PWM_E_PERIOD_UNCHANGEABLE

Design Identifier	Description
MCAL-6856 - SWS_Pwm_50002 : PWM_Init : Developement Error Detection PWM_E_ALREADY_INITIALIZED PUBLISHED	SWS_Pwm_50002 : PWM_Init : Developement Error Detection PWM_E_ALREADY_INITIALIZED
MCAL-6838 - SWS_Pwm_00151 : Pwm_GetVersionInfo API : Developement Error Detection PWM_E_PARAM_POINTER PUBLISHED	SWS_Pwm_00151 : Pwm_GetVersionInfo API : Developement Error Detection PWM_E_PARAM_POINTER

Design Identifier	Description
MCAL-6847 - SWS_Pwm_00201 : Pwm Development Error Types PUBLISHED	SWS_Pwm_00201 : Pwm Development Error Types
MCAL-6879 - SWS_Pwm_00202 : Pwm RunTime Error Types PUBLISHED	SWS_Pwm_00202 : Pwm RunTime Error Types

4.7.3 4.7.3 Error Detection

The detection of development errors is configurable (ON / OFF) at pre-compile time. The switch PwmDevErrorDetect will activate or deactivate the detection of all development errors.

4.7.4 4.7.4 Error notification (DET)

All detected development errors are reported to Det_ReportError service of the Development Error Tracer (DET).

4.8 **4.8 Resource Behavior**

- **Code Size :** Implementation of this driver shall not exceed 40 kilo bytes of code and 5 KB of data section.

- **Stack Size** : Worst case stack utilization shall not exceed 2 kilo bytes

5 5 Implementation Details

5.1 5.1 Data structures and resources

MACROS, Data Types & Structures

The sections below lists some of key data structures that shall be implemented and used in driver implementation of Maximum number of channels

Type	Identifier	Comments
uint32	PWM_MAX_CHANNELS	Defines the maximum number of channels that are configured. Its required that configurations for all channel specified is valid.

5.1.1 5.1.1 Pwm_ChannelType

Used to specify the numeric identifier for a channel, please refer section Reference 1 - Autosar 4.3.1

5.1.2 5.1.2 Pwm_PeriodType

Used to specify the period of a PWM channel, refer section Reference 1 - Autosar 4.3.1

5.1.3 5.1.3 Pwm_OutputStateType

Used to specify the Output state of a PWM channel, refer section Reference 1 - Autosar 4.3.1

5.1.4 5.1.4 Pwm_EdgeNotificationType

Enumeration used to specify the type of edge notification of a PWM channel. Reference 1 - Autosar 4.3.1

5.1.5 5.1.5 Pwm_ChannelClassType

Enumeration used to specify the class of a PWM channel, whether the period is fixed or not. Refer section Reference 1 - Autosar 4.3.1

5.1.6 5.1.6 Pwm_ConfigType

Hardware dependent structure used to specify the initial data for the PWM driver. Refer section Reference 1 - Autosar 4.3.1

5.1.7 5.1.7 Pwm_FrequencyType

Used to specify the numeric identifier for frequency,

5.1.8 5.1.9 Pwm_ChannelConfigType

Structure represents a Pwm channel configuration

5.1.9 5.1.10 Pwm_RegisterReadbackType

Name	IP	Type	Range	Comments
pwmRev	GPT	uint32	0 to 0xFFFFFFFF	H/W version identifier, will not change for a given SoC
pwmTtgr	GPT	uint32	0 to 0xFFFFFFFF	Shall always read 0xFFFFFFFF
pwmTimerSynCtrl	GPT	uint32	0 to 0xFFFFFFFF	Interface control register, will read 0x00000000

Design Identifier	Description
MCAL-6883 - SWS_Pwm_00106 : Pwm_ChannelType PUBLISHED	SWS_Pwm_00106 : Pwm_ChannelType

Design Identifier	Description
MCAL-6763 - SWS_Pwm_00107 : Pwm_PeriodType PUBLISHED	SWS_Pwm_00107 : Pwm_PeriodType
MCAL-6775 - SWS_Pwm_00108 : Pwm_OutputStateType PUBLISHED	SWS_Pwm_00108 : Pwm_OutputStateType
MCAL-6836 - SWS_Pwm_00109 : Pwm_EdgeNotificationType PUBLISHED	SWS_Pwm_00109 : Pwm_EdgeNotificationType

Design Identifier	Description
MCAL-6796 - SWS_Pwm_00110 : Pwm_ChannelClassType PUBLISHED	SWS_Pwm_00110 : Pwm_ChannelClassType
MCAL-6799 - SWS_Pwm_00111 : Pwm_ConfigType Structure PUBLISHED	SWS_Pwm_00111 : Pwm_ConfigType Structure
MCAL-6885 - SWS_Pwm_00061 : Pwm_ConfigType PUBLISHED	SWS_Pwm_00061 : Pwm_ConfigType

Design Identifier	Description
MCAL-6852 - PWM : Register Readback API: Pwm_RegisterReadbackType PUBLISHED	PWM : Register Readback API: Pwm_RegisterReadbackType

5.2 5.2 Global Variables

This design expects that implementation will require to use following global variables.

Variable	Type	Description	Default Value
Pwm_DrvStatus	uint8	PWM driver status	PWM_STATUS_UNINIT
Pwm_ChObj	Pwm_ChObjType	PWM channel object	-

5.3 5.3 Dynamic Behavior - Control Flow Diagram

Not Applicable

5.4 5.4 Dynamic Behavior - Data Flow Diagram

Not Applicable

5.5 5.5 Application Parameters

Sections below highlight the design considerations for the implementation.

5.5.1 5.5.1 Pwm_Init

Parameter	Description	Possible Value ranges	Unit of Value	Default Value	Variant
CfgPtr	Pointer to configuration set	0xFFFFFFFF	-	-	N.A

5.5.2 5.5.2 Pwm_SetDutyCycle

Parameter	Description	Possible Value ranges	Unit of Value	Default Value	Variant
DutyCycle	Pointer to configuration set	0-100%	-	-	N.A
ChannelNumber	Channel number	0-30	-	1	N.A

5.5.3 5.5.3 Pwm_SetPeriodAndDuty

Parameter	Description	Possible Value ranges	Unit of Value	Default Value	Variant
DutyCycle	Duty cycle for a channel	0-100%	-	-	N.A
ChannelNumber	Chanelnumber	0-30	-	1	N.A
Period	Period in ticks	-	-	-	N.A

5.5.4 5.5.4 Pwm_SetOutputToIdle

Parameter	Description	Possible Value ranges	Unit of Value	Default Value	Variant
ChannelNumber	Chanelnumber	0-30	-	1	N.A

5.5.5 5.5.5 Pwm_DisableNotification

Parameter	Description	Possible Value ranges	Unit of Value	Default Value	Variant
ChannelNumber	Chanelnumber	0-30	-	1	N.A

5.5.6 5.5.6 Pwm_EnableNotification

Parameter	Description	Possible Value ranges	Unit of Value	Default Value	Variant
Notification	Edge Notification	1-3	-	-	N.A
ChannelNumber	Chanelnumber	0-30	-	1	N.A

5.5.7 5.5.7 Pwm_GetVersionInfo

Parameter	Description	Possible Value ranges	Unit of Value	Default Value	Variant
VersionInfoPtr	Pointer to store the version information of this module	0xFFFFFFFF	-	-	N.A

5.5.8 5.5.8 Pwm_RegisterReadback

Parameter	Description	Possible Value ranges	Unit of Value	Default Value	Variant
ChannelNumber	Chanelnumber	0-30	-	1	N.A
RegRbPtr	Pointer to where to store the readback values. If this pointer is NULL_PTR, then the API will return E_NOT_OK	0xFFFFFFFF	-	-	N.A

6 6 Safety Diagnostic Features

6.1 6.1 GPTimer Module

The following tests can be applied as diagnostics for this module (to provide diagnostic coverage on a specific function):

- **1oo2 Voting Using a Second Timer:** Application can use other time based modules on the device to periodically check and perform diagnostic on main counter. Example of other external time based modules include PMU (Performance Monitoring Unit), Timer Manager Module and Global Timebase Counter (GTC).
- **Software Test of Basic Functionality including error tests:** Application can use the Pwm_EnableNotification() API to verify correct frequency by counting the number of interrupts received in particular time period. Reference example will be provided in MCAL package. The Unit Test framework will be used to check basic functionality, as well as negative tests by injecting errors.
- **Software readback of written configuration:** The driver provides a Pwm_RegisterReadback() API that application can use to check configured register value. Reference example are added in the PwmApp_Gpt.
- **Periodic software readback of static configuration registers:** The driver will internally readback the configured registers to ensure correct value has been set.
- **Analog-to-Digital Converter Information redundancy techniques:** Information redundancy techniques can be applied via software as an additional runtime diagnostic on ADC conversion. This can be done by application to filter and perform plausibility check to ensure converted values are in expected range.

7 7 Low Level Definitions

The detailed API and interface description is available as part of of the PWM Driver AUTOSAR Specification as listed in Reference 1 - Autosar 4.3.1 This section describes the API supported by the MCAL driver and the requirements covered by each of the API.

7.1 7.1 Driver API's

For the standard APIs please refer 8.3 of the PWM Driver AUTOSAR Specification as listed in Reference 1 - Autosar 4.3.1. Sections below highlight other design considerations for the implementation.

7.1.1 7.1.1 Pwm_Init

Refer section 8.3.1 of the PWM Driver AUTOSAR Specification as listed in Reference 1 - Autosar 4.3.1.

Design Identifier	Description
MCAL-6773 - SWS_Pwm_00046 : Pwm_Init API : Development Error Detection PWM_E_PARAM_CONFIG PUBLISHED	SWS_Pwm_00046 : Pwm_Init API : Development Error Detection PWM_E_PARAM_CONFIG

Design Identifier	Description
MCAL-6806 - SWS_Pwm_00116 : Pwm_Init API : Function calls before init PUBLISHED	SWS_Pwm_00116 : Pwm_Init API : Function calls before init
MCAL-6798 - SWS_Pwm_00118 : Pwm_Init API Development Error Detection : PWM_E_ALREADY_INITIALIZED. PUBLISHED	SWS_Pwm_00118 : Pwm_Init API Development Error Detection : PWM_E_ALREADY_INITIALIZED.
MCAL-6786 - SWS_Pwm_00121 : Pwm_Init API : Re-Initialization PUBLISHED	SWS_Pwm_00121 : Pwm_Init API : Re-Initialization
MCAL-6828 - SWS_Pwm_00007 : Pwm_Init API Variable configuration PUBLISHED	SWS_Pwm_00007 : Pwm_Init API Variable configuration
MCAL-6767 - SWS_Pwm_00062 : Pwm_Init API Resource configuration PUBLISHED	SWS_Pwm_00062 : Pwm_Init API Resource configuration
MCAL-6884 - SWS_Pwm_10009 : Pwm_Init API Channel configuration PUBLISHED	SWS_Pwm_10009 : Pwm_Init API Channel configuration

Design Identifier	Description
MCAL-6752 - SWS_Pwm_00052 : Pwm_Init API Disable Notification PUBLISHED	SWS_Pwm_00052 : Pwm_Init API Disable Notification
MCAL-6834 - SWS_Pwm_00093 : Pwm_Init API Call during runtime PUBLISHED	SWS_Pwm_00093 : Pwm_Init API Call during runtime
MCAL-6841 - SWS_Pwm_10120 : Pwm_Init API : Pre-Compile and Link time variants PUBLISHED	SWS_Pwm_10120 : Pwm_Init API : Pre-Compile and Link time variants
MCAL-6779 - SWS_Pwm_20009 : Pwm_Init API Signal Configuration Duty Cycle at 0 or 100 percentage PUBLISHED	SWS_Pwm_20009 : Pwm_Init API Signal Configuration Duty Cycle at 0 or 100 percentage
MCAL-6789 - SWS_Pwm_30009 : Pwm_Init API Signal Configuration PUBLISHED	SWS_Pwm_30009 : Pwm_Init API Signal Configuration

7.1.2 7.1.2 Pwm_DeInit

Refer section 8.3.2 of the PWM Driver AUTOSAR Specification as listed in Reference 1 - Autosar 4.3.1.

Design Identifier	Description
MCAL-6890 - SWS_Pwm_00010: Pwm_DeInit API PUBLISHED	SWS_Pwm_00010: Pwm_DeInit API
MCAL-6875 - SWS_Pwm_00011: Pwm_DeInit API Output Signal State PUBLISHED	SWS_Pwm_00011: Pwm_DeInit API Output Signal State
MCAL-6840 - SWS_Pwm_00012: Pwm_DeInit API Disable Interrupts and Notifications PUBLISHED	SWS_Pwm_00012: Pwm_DeInit API Disable Interrupts and Notifications
MCAL-6827 - SWS_Pwm_10080: Pwm_DeInit API Pre Compile time configurable PUBLISHED	SWS_Pwm_10080: Pwm_DeInit API Pre Compile time configurable
MCAL-6870 - SWS_Pwm_20080: Pwm_DeInit API Configurable by PwmDeInitApi PUBLISHED	SWS_Pwm_20080: Pwm_DeInit API Configurable by PwmDeInitApi

Design Identifier	Description
MCAL-6805 - SWS_Pwm_00117 : PWM Module : Development error PWM_E_UNINIT. PUBLISHED	SWS_Pwm_00117 : PWM Module : Development error PWM_E_UNINIT.
MCAL-6891 - SWS_Pwm_10051 : PWM Module Development error detection and reporting PUBLISHED	SWS_Pwm_10051 : PWM Module Development error detection and reporting
MCAL-6886 - SWS_Pwm_20051 : PWM Module Development erro r detection and skipping functionality PUBLISHED	SWS_Pwm_20051 : PWM Module Development erro r detection and skipping functionality

7.1.3 7.1.3 Pwm_SetDutyCycle

Refer 8.3.3 of the PWM Driver AUTOSAR Specification as listed in Reference 1 - Autosar 4.3.1.

Design Identifier	Description
MCAL-6759 - SWS_Pwm_00013: Pwm_SetDutyCycle API PUBLISHED	SWS_Pwm_00013: Pwm_SetDutyCycle API
MCAL-6839 - SWS_Pwm_00014: Pwm_SetDutyCycle API Duty Cycle 0 or 100 percentage and Polarity PUBLISHED	SWS_Pwm_00014: Pwm_SetDutyCycle API Duty Cycle 0 or 100 percentage and Polarity
MCAL-6751 - SWS_Pwm_00016: Pwm_SetDutyCycle API Output signal Modulation PUBLISHED	SWS_Pwm_00016: Pwm_SetDutyCycle API Output signal Modulation
MCAL-6832 - SWS_Pwm_00017: Pwm_SetDutyCycle API Update Duty Cycle PUBLISHED	SWS_Pwm_00017: Pwm_SetDutyCycle API Update Duty Cycle
MCAL-6801 - SWS_Pwm_00058 : PWM Duty cycle Parameter Width PUBLISHED	SWS_Pwm_00058 : PWM Duty cycle Parameter Width
MCAL-6880 - SWS_Pwm_00059 : PWM Duty Cycle Scaling Scheme PUBLISHED	SWS_Pwm_00059 : PWM Duty Cycle Scaling Scheme

Design Identifier	Description
MCAL-6863 - SWS_Pwm_00018: Pwm_SetDutyCycle API Output Signal Spike PUBLISHED	SWS_Pwm_00018: Pwm_SetDutyCycle API Output Signal Spike
MCAL-6805 - SWS_Pwm_00117 : PWM Module : Development error PWM_E_UNINIT. PUBLISHED	SWS_Pwm_00117 : PWM Module : Development error PWM_E_UNINIT.
MCAL-6864 - SWS_Pwm_00047 : PWM Module : Development error PWM_E_PARAM_CHANNEL PUBLISHED	SWS_Pwm_00047 : PWM Module : Development error PWM_E_PARAM_CHANNEL
MCAL-6891 - SWS_Pwm_10051 : PWM Module Development error detection and reporting PUBLISHED	SWS_Pwm_10051 : PWM Module Development error detection and reporting
MCAL-6886 - SWS_Pwm_20051 : PWM Module Development erro r detection and skipping functionality PUBLISHED	SWS_Pwm_20051 : PWM Module Development erro r detection and skipping functionality

Design Identifier	Description
MCAL-6851 - SWS_Pwm_10082: Pwm_SetDutyCycle API Pre Compile time configurable PUBLISHED	SWS_Pwm_10082: Pwm_SetDutyCycle API Pre Compile time configurable
MCAL-6813 - SWS_Pwm_20082: Pwm_SetDutyCycle API Pre Compile time Configurable by PwmSetDutyCycle PUBLISHED	SWS_Pwm_20082: Pwm_SetDutyCycle API Pre Compile time Configurable by PwmSetDutyCycle

7.1.4 7.1.4 Pwm_SetPeriodAndDuty

Refer section 8.3.4 of the PWM Driver AUTOSAR Specification as listed in Reference 1 - Autosar 4.3.1.

Design Identifier	Description
MCAL-6774 - SWS_Pwm_00019: Pwm_SetPeriodAndDuty API PUBLISHED	SWS_Pwm_00019: Pwm_SetPeriodAndDuty API

Design Identifier	Description
MCAL-6757 - SWS_Pwm_00076: Pwm_SetPeriodAndDuty API Period Update PUBLISHED	SWS_Pwm_00076: Pwm_SetPeriodAndDuty API Period Update
MCAL-6787 - SWS_Pwm_00020: Pwm_SetPeriodAndDuty API Output Signal Spike PUBLISHED	SWS_Pwm_00020: Pwm_SetPeriodAndDuty API Output Signal Spike
MCAL-6801 - SWS_Pwm_00058 : PWM Duty cycle Parameter Width PUBLISHED	SWS_Pwm_00058 : PWM Duty cycle Parameter Width
MCAL-6880 - SWS_Pwm_00059 : PWM Duty Cycle Scaling Scheme PUBLISHED	SWS_Pwm_00059: PWM Duty Cycle Scaling Scheme
MCAL-6805 - SWS_Pwm_00117 : PWM Module : Development error PWM_E_UNINIT. PUBLISHED	SWS_Pwm_00117 : PWM Module : Development error PWM_E_UNINIT.

Design Identifier	Description
MCAL-6815 - SWS_Pwm_00045 : PWM Module : Development error PWM_E_PERIOD_UNCHANGEABLE PUBLISHED	SWS_Pwm_00045 : PWM Module : Development error PWM_E_PERIOD_UNCHANGEABLE
MCAL-6864 - SWS_Pwm_00047 : PWM Module : Development error PWM_E_PARAM_CHANNEL PUBLISHED	SWS_Pwm_00047 : PWM Module : Development error PWM_E_PARAM_CHANNEL
MCAL-6891 - SWS_Pwm_10051 : PWM Module Development error detection and reporting PUBLISHED	SWS_Pwm_10051 : PWM Module Development error detection and reporting
MCAL-6886 - SWS_Pwm_20051 : PWM Module Development erro r detection and skipping functionality PUBLISHED	SWS_Pwm_20051 : PWM Module Development erro r detection and skipping functionality
MCAL-6822 - SWS_Pwm_00041: Pwm_SetPeriodAndDuty API Channel PUBLISHED	SWS_Pwm_00041: Pwm_SetPeriodAndDuty API Channel

Design Identifier	Description
MCAL-6793 - SWS_Pwm_10083: Pwm_SetPeriodAndDuty API Pre Compile Time Configurable PUBLISHED	SWS_Pwm_10083: Pwm_SetPeriodAndDuty API Pre Compile Time Configurable
MCAL-6785 - SWS_Pwm_20083: Pwm_SetPeriodAndDuty API Configurable through PwmSetPeriodAndDuty PUBLISHED	SWS_Pwm_20083: Pwm_SetPeriodAndDuty API Configurable through PwmSetPeriodAndDuty
MCAL-6777 - SWS_Pwm_00150: Pwm_SetPeriodAndDuty API : Period set as zero PUBLISHED	SWS_Pwm_00150: Pwm_SetPeriodAndDuty API : Period set as zero

7.1.5 7.1.5 Pwm_SetOutputToIdle

Refer section 8.3.5 of the PWM Driver AUTOSAR Specification as listed in Reference 1 - Autosar 4.3.1.

Design Identifier	Description
MCAL-6777 - SWS_Pwm_00150: Pwm_SetPeriodAndDuty API : Period set as zero PUBLISHED	SWS_Pwm_00150: Pwm_SetPeriodAndDuty API : Period set as zero
MCAL-6784 - SWS_Pwm_00021 : Pwm_SetOutputToIdle API PUBLISHED	SWS_Pwm_00021 : Pwm_SetOutputToIdle API
MCAL-6756 - SWS_Pwm_10084 : Pwm_SetOutputToIdle API Pre Compile Time Configurable PUBLISHED	SWS_Pwm_10084 : Pwm_SetOutputToIdle API Pre Compile Time Configurable
MCAL-6766 - SWS_Pwm_20084 : Pwm_SetOutputToIdle API Configurable through PwmSetOutputToIdle PUBLISHED	SWS_Pwm_20084 : Pwm_SetOutputToIdle API Configurable through PwmSetOutputToIdle
MCAL-6887 - SWS_Pwm_10086 : Pwm_SetOutputToIdle API: Re-activation through Pwm_SetPeriodAndDuty API PUBLISHED	SWS_Pwm_10086 : Pwm_SetOutputToIdle API: Re-activation through Pwm_SetPeriodAndDuty API
MCAL-6887 - SWS_Pwm_10086 : Pwm_SetOutputToIdle API: Re-activation through Pwm_SetPeriodAndDuty API PUBLISHED	SWS_Pwm_20086 : Pwm_SetOutputToIdle API : Re-activation through Pwm_SetDutyCycle API

Design Identifier	Description
MCAL-6887 - SWS_Pwm_10086 : Pwm_SetOutputToIdle API: Re-activation through Pwm_SetPeriodAndDuty API PUBLISHED	SWS_Pwm_00119 : Pwm_SetOutputToIdle API : Re-activation for fixed period type channels through Pwm_SetDutyCycle API

7.1.6 7.1.6 Pwm_DisableNotification

Refer section 8.3.7 of the PWM Driver AUTOSAR Specification as listed in Reference 1 - Autosar 4.3.1.

Design Identifier	Description
MCAL-6807 - SWS_Pwm_00023 : Pwm_DisableNotification API PUBLISHED	SWS_Pwm_00023 : Pwm_DisableNotification API
MCAL-6848 - SWS_Pwm_10112 : Pwm_DisableNotification API Pre-compile time configurable PUBLISHED	SWS_Pwm_10112 : Pwm_DisableNotification API Pre-compile time configurable

Design Identifier	Description
MCAL-6830 - SWS_Pwm_20112 : Pwm_DisableNotification API configurable through parameter and Error Detection supported PUBLISHED	SWS_Pwm_20112 : Pwm_DisableNotification API configurable through parameter and Error Detection supported
MCAL-6805 - SWS_Pwm_00117 : PWM Module : Development error PWM_E_UNINIT. PUBLISHED	SWS_Pwm_00117 : PWM Module : Development error PWM_E_UNINIT.
MCAL-6864 - SWS_Pwm_00047 : PWM Module : Development error PWM_E_PARAM_CHANNEL PUBLISHED	SWS_Pwm_00047 : PWM Module : Development error PWM_E_PARAM_CHANNEL
MCAL-6891 - SWS_Pwm_10051 : PWM Module Development error detection and reporting PUBLISHED	SWS_Pwm_10051 : PWM Module Development error detection and reporting
MCAL-6886 - SWS_Pwm_20051 : PWM Module Development erro r detection and skipping functionality PUBLISHED	SWS_Pwm_20051 : PWM Module Development erro r detection and skipping functionality

Design Identifier	Description
MCAL-6773 - SWS_Pwm_00046 : Pwm_Init API : Development Error Detection PWM_E_PARAM_CONFIG PUBLISHED	SWS_Pwm_00046 : Pwm_Init API : Development Error Detection PWM_E_PARAM_CONFIG

7.1.7 7.1.7 Pwm_EnableNotification

Refer section 8.3.8 of the PWM Driver AUTOSAR Specification as listed in Reference 1 - Autosar 4.3.1.

Design Identifier	Description
MCAL-6800 - SWS_Pwm_00024 : Pwm_EnableNotification PUBLISHED	SWS_Pwm_00024 : Pwm_EnableNotification
MCAL-6805 - SWS_Pwm_00117 : PWM Module : Development error PWM_E_UNINIT. PUBLISHED	SWS_Pwm_00117 : PWM Module : Development error PWM_E_UNINIT.

Design Identifier	Description
MCAL-6889 - SWS_Pwm_00081 : Pwm_EnableNotification Cancel Pending interrupts PUBLISHED	SWS_Pwm_00081 : Pwm_EnableNotification Cancel Pending interrupts
MCAL-6865 - SWS_Pwm_10113 : Pwm_EnableNotification : Pre-compile time configurable PUBLISHED	SWS_Pwm_10113 : Pwm_EnableNotification : Pre-compile time configurable
MCAL-6765 - SWS_Pwm_20113 : Pwm_EnableNotification configurable through parameter and error detection supported PUBLISHED	SWS_Pwm_20113 : Pwm_EnableNotification configurable through parameter and error detection supported
MCAL-6864 - SWS_Pwm_00047 : PWM Module : Development error PWM_E_PARAM_CHANNEL PUBLISHED	SWS_Pwm_00047 : PWM Module : Development error PWM_E_PARAM_CHANNEL
MCAL-6891 - SWS_Pwm_10051 : PWM Module Development error detection and reporting PUBLISHED	SWS_Pwm_10051 : PWM Module Development error detection and reporting

Design Identifier	Description
MCAL-6886 - SWS_Pwm_20051 : PWM Module Development erro r detection and skipping functionality PUBLISHED	SWS_Pwm_20051 : PWM Module Development erro r detection and skipping functionality

7.1.8 7.1.8 Pwm_GetVersionInfo

Refer section 8.3.13 of the PWM Driver AUTOSAR Specification as listed in Reference 1 - Autosar 4.3.1.

Design Identifier	Description
MCAL-6805 - SWS_Pwm_00117 : PWM Module : Development error PWM_E_UNINIT. PUBLISHED	SWS_Pwm_00117 : PWM Module : Development error PWM_E_UNINIT.

7.1.9 7.1.9 Pwm_RegisterReadback

As noted from previous implementation, the timer configuration registers could potentially be corrupted by other entities (s/w or h/w). One of the recommended detection methods would be to periodically read-back the configuration and confirm configuration is consistent. The service API defined below shall be implemented to enable this detection.

	Description	*Comments*
Service Name	Pwm_RegisterReadback	Can potentially be turned OFF
Syntax	Std_ReturnType Pwm_RegisterReadback(Pwm_ChannelType PwmChannel, Pwm_RegisterReadbackType * RegRbPtr)	Pwm_RegisterReadbackType defines the type, that holds critical values, refer below
Service ID	0x0F	
Sync / Async	Sync	
Reentrancy	Non Reentrant	
Parameter in	PwmChannel	Identifies a unique valid channel

Parameters out	RegRbPtr	A pointer of type Pwm_RegisterReadbackType, which holds the read back values
Return Value	Standard return type	E_OK or E_NOT_OK in case of invalid channel id

The critical register listed is a recommendation and implementation shall determine appropriate registers.

This service could potentially be turned OFF in the configurator.

Design Identifier	Description
MCAL-6871 - PWM Register Readback PUBLISHED	PWM Register Readback

8 8 Performance Objectives

8.1 8.1 Resource Consumption Objectives

ROM - Program(KB)	ROM - Data(KB)	RAM - Program(KB)	RAM - Data(KB)	Stack Size (KB)	EEPROM (KB)	% CPU Utilization
40	NA	NA	1	2	NA	NA

8.2 8.2 Critical timing and Performance

Not Applicable

9 9 Decision Analysis & Resolution (DAR)

Sections below list some of the important design decisions and rational behind those decision.

9.1 **9.1 Timer Mode configuration in Overflow Only Mode for Duty cycle of 50%**

The PWM signal generation when duty cycle is 50% and 0% < duty cycle < 100% can be done in the following ways:

No.	Decision Criteria	Alternatives	Selected alternative		
1	The PWM signal generation from the Mode timer should be optimized.	Use timer in either Overflow Mode or (Overflow and Compare Mode) configuration depending on duty cycle. **Advantages:** • Incase of 50% duty cycle, only the overflow register condition needs to be checked to generate the interrrupts and PWM signal **Disadvantages:** • Seperate one time Mode configuration depending on the duty cycle needs to be done in software	Overflow and Compare		

Rationale

The PWM module is more efficeintly implemented by operating timer in overflow mode alone when duty cycle of 50% is required as this avoids the continuous register compare to TMAR to generate the trigger at compare condition. The configuration to overflow or overflow and compare in different duty cycles needs to be done one time, and is a lesser overhead.

Trade-offs

- TMAR register needs to be performed continuously to detect the compare condition

No.	Decision Criteria	Alternatives	Selected alternative	Rationale	Trade-offs
		Use of Timer in compare and overflow Mode configurations irrespective of duty cycle. **Advantages:** • No need to have seperate Mode configuration of timer for different duty cycles. Timer can be configured in overflow and compare mode **Disadvantages:** • In 50% duty cycle, register compare to TMAR register needs to be performed continuously to detect the compare condition.			

9.2 9.2 Integration of GpTimer

The PWM module instance can use either Gptimer hardware or the EPWM hardware IP as the source for signal creation. The most efficient way of integrating both IP into PWM module has to be analyzed.

No.	Decision Criteria	Alternatives	Selected alternative	Rationale	Trade-offs
1	The PWM driver implementation and interface should be simple and user friendly. The user should be able to choose which one to use.	Use PWM Driver Index as the differentiation factor	Based on EPWM/GPT	**Advantages:** • The user can choose which driver instance they want to run (0 for EPWM and 1 for GPT). • This can be provided as a configurable parameter. User has the flexibility to choose.	

• Easier implementation and simpler integration. • Compatible with the Pre-Compile and Post_Build variants.	• Code size will be reduced as only the needed IP libraries will be included. **Disadvantages:** • N/A **Rationale** **Trade** **-offs**		AOption 1 is chosen as it is most compatible None with the configurator and also will make the integration simpler. As mentioned above, the PwmIndex (PWM_INSTANCE_ID) variable will be used as the configuration parameter to choose between EPWM(default) or Gptimer. PWM_Priv.c will call the respective API (will be differentiated using the PWM_INSTNACE_ID).

10 10 Testing Guidelines

The sections below identify some of the aspects of design that would require emphasis during testing of this design implementation

- **PWM Wave generation : Polarities**
- Test cases shall check the generation of PWM wave based on different initial polarities configured. This will be verified on the CRO.
- **PWM Wave generation : Duty Cycle, Period and Input clock frequency**
- Test cases shall perform equivalence class test and ensure different duty cycles, periods for a PWM signal can be supported based on input clock frequencies.
- Test cases should also check for conditions where the PWM parameters are reconfigured while the timer is running.
- **PWM Wave generation :Reset (Period = 0)**
- Test cases should check the behaviour on reset and with Period = 0
- **PWM Multiple Instances**
- Configure multiple PWM instances with duty cycle of 75% and 25%, reset and vary the duty cycles to ensure that the re-entrancy is maintained.

11 11 Template Revision History

Author Name	Description	Version	Date
Yaniv Machani	Initial version	0.1	03 Oct 2018
Yaniv Machani	Updated to include EP views	0.4	02 Nov 2018
Yaniv Weizman	Restructuring and editing to further meet the A-SPICE and EP requirements	0.5	27 Dec 2018
Yaniv Weizman	Adding link to Architecture review template	0.6	22 Oct 2019

Author Name	Description	Version	Date
Yaniv Weizman	Adding requirement type column for requirements table (Functional/Non-Functional). Adding DAR table	0.65	13 Nov 2019
Yaniv Weizman	Adding tables for Testing guidelines	0.7	18 Nov 2019
Krishna	Updated based on ASPICE requirements	0.8	20 Aug 2020
Krishna	Updated based on the feedback from Jon N	0.9	09 Oct 2020
Krishna	Updated the traceability scheme	1.0	17 Dec 2020