MCAL ePWM Module Software Design Document

Document Version : 41
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 April 2022	Rakesh L	DONE	First version
0.2	25 May 2022	Rakesh L	DONE	Updated for review comments
v.41	22 Jun 2022	Rakesh L	DONE	Added Comala Workflow

2 2 Terms and Abbreviations

Abbreviation /Term	Meaning / Explanation
DIO	Digital Input Output
ECU	Electric Control Unit
DMA	Direct Memory Access
ICU	Input Capture Unit
NMI	Non Maskable Interrupt
OS	Operating System
PLL	Phase Locked Loop
PWM	Pulse Width Modulation

Abbreviation /Term	Meaning / Explanation
RX	Reception (in the context of bus communication)
SPAL	The name of this working group
SFR	Special Function Register
RTE	Run Time Environment
DET	Default Error Tracer – module to which errors are reported
DEM	Diagnostic Event Manager

3 3 Introduction

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

- Supported AUTOSAR Release**: 4.3.1**

- Supported Configuration Variants**: Pre-Compile & Post Build**
- Vendor ID**: PWM_VENDOR_ID (44)**
- Module ID**: PWM_MODULE_ID (121)**

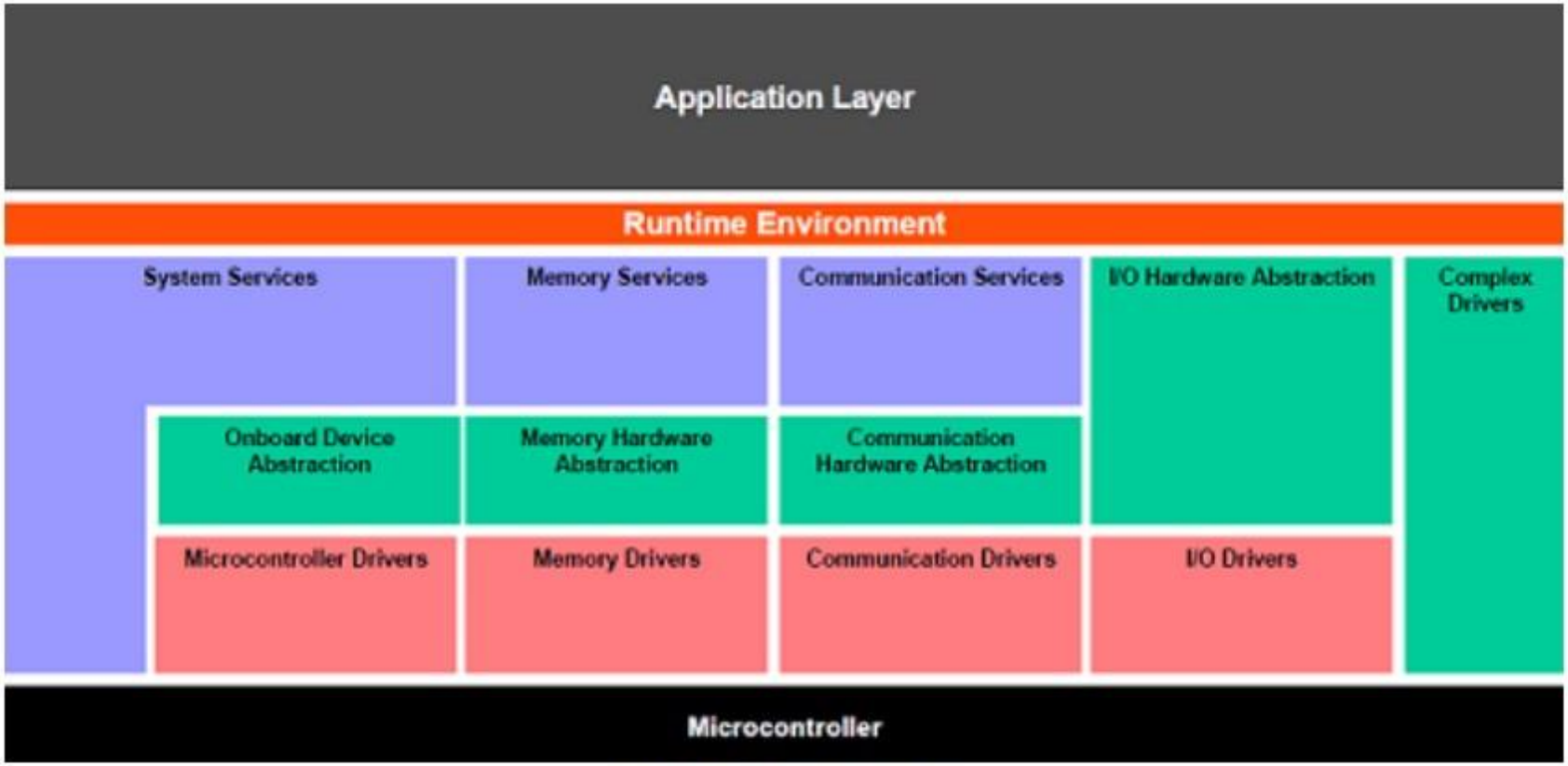
3.1 **3.1 Overview**

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

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

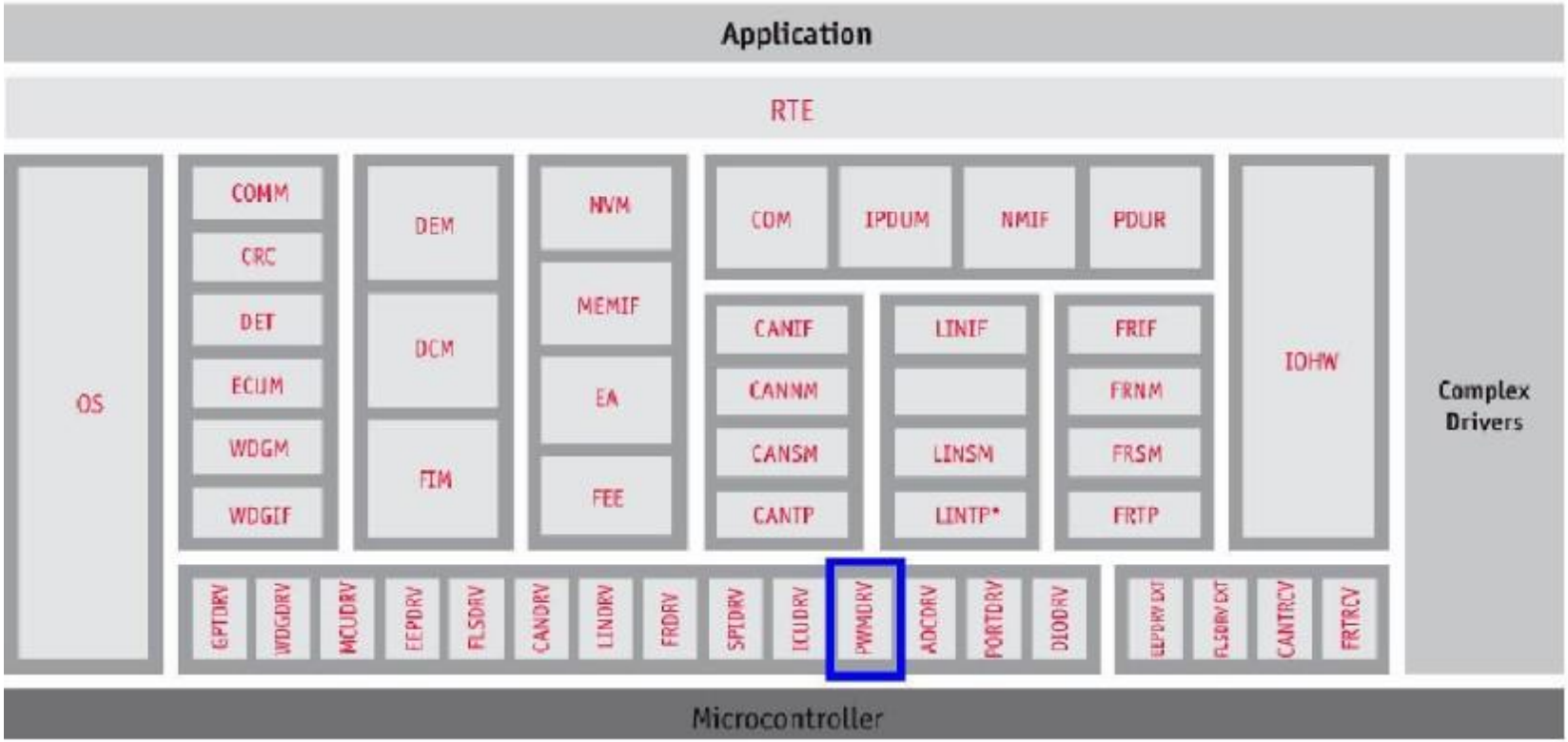
The BSW is further divided into 4 layers:

- 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.



AUTOSAR Architecture – PWM MCAL

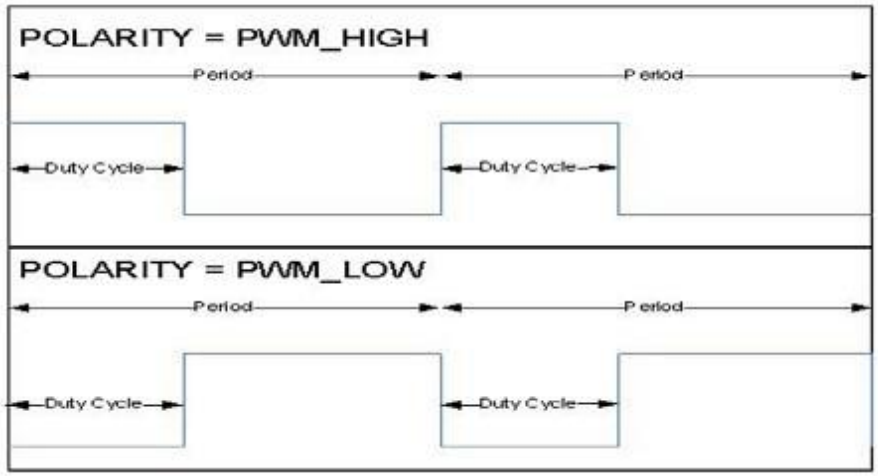
3.2 **3.2 Purpose and Scope**

The Detailed Design document provides the design details of EPWM 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 EPWM module.

3.3 **3.3 Module Overview**

The EPWM 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 "ehrpwm_10_rel.1.3.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 IPs, EPWM module.

3.4.1 3.4.1 EPWM Module

The ePWM(enhanced) unit described here addresses these requirements by allocating all needed timing and control resources on a per PWM channel basis. Cross coupling or sharing of resources has been avoided; instead, the ePWM is built up from smaller single channel modules with separate resources and that can operate together as required to form a system. This modular approach results in easy understand of its operation quickly.

J7 family device include six instances of ePWM. The EPWM module represents one complete PWM channel composed of two PWM outputs: EPWMxA and EPWMxB. A given EPWM module functionality can be extended with the so called High-Resolution Pulse Width modulator.

In the further description the letter x within a signal or module name is used to indicate a generic EPWM instance on a device. For example, output signals EPWMxA and EPWMxB refer to the output signals from the EPWMx instance.

There are 6 instances of the EPWM integrated in the device. Each of the Enhanced Pulse Width Modulator (EPWM) includes an Enhanced High Resolution Modulator (HRPWM). The high-resolution functionality is implemented only on the EPWMxA output. EPWMxB output has conventional PWM capabilities. At system level the EPWM0 through EPWM5 integration features are listed below:

- A 32-bit slave configuration port on the CBASS0 interconnect.
- A single functional clock from PLLCTRL.
- 2 hardware events per EPWM, that is a total of 12 events. From these events each EPWM:
- generates 2 interrupts to the device COMPUTE_CLUSTER0, PRU_ICSSG0_INTC, PRU_ICSSG1_INTC and MAIN2MCU_INTRTR_PLS
- A synchronization input/output daisy chain-like connection exists between the EPWM0 trough EPWM5.

The high-resolution pulse-width modulator (HRPWM) extends the time resolution capabilities of the conventionally derived digital pulse-width modulator (PWM). HRPWM is typically used when PWM resolution falls below ~9-10 bits. The key features of HRPWM are:

- Extended time resolution capability with regards to falling edge.
- Used in Duty cycle control methods.
- Finer time granularity control or edge positioning using extensions to the Compare A.
- Implemented using the A signal path of PWM, that is, on the EPWMxA output. EPWMxB output has conventional PWM capabilities.

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 EPWM channel at runtime besides the default configuration.
- The EPWM signal that can be generated is a square wave with variable duty cycle and period .

Design Identifier	Description
MCAL-7638 - SWS_Pwm_00065 : PWM General File Structure PUBLISHED	SWS_Pwm_00065 : PWM General File Structure
MCAL-7571 - SWS_Pwm_00088 : PWM APIs Reentrancy PUBLISHED	SWS_Pwm_00088 : PWM APIs Reentrancy
MCAL-7625 - SWS_Pwm_00070 : PWM Time Unit PUBLISHED	SWS_Pwm_00070 : PWM Time Unit
MCAL-7505 - SWS_Pwm_00106 : Pwm_ChannelType PUBLISHED	SWS_Pwm_00106 : Pwm_ChannelType

Design Identifier	Description
MCAL-7587 - SWS_Pwm_00107 : Pwm_PeriodType PUBLISHED	SWS_Pwm_00107 : Pwm_PeriodType
MCAL-7543 - SWS_Pwm_00108 : Pwm_OutputStateType PUBLISHED	SWS_Pwm_00108 : Pwm_OutputStateType
MCAL-7576 - SWS_Pwm_00109 : Pwm_EdgeNotificationType PUBLISHED	SWS_Pwm_00109 : Pwm_EdgeNotificationType
MCAL-7589 - SWS_Pwm_00110 : Pwm_ChannelClassType PUBLISHED	SWS_Pwm_00110 : Pwm_ChannelClassType
MCAL-7610 - SWS_Pwm_00116 : Pwm_Init API : Function calls before init PUBLISHED	SWS_Pwm_00116 : Pwm_Init API : Function calls before init
MCAL-7561 - SWS_Pwm_00089 : PWM Module integrity check PUBLISHED	SWS_Pwm_00089 : PWM Module integrity check

Design Identifier	Description
MCAL-7561 - SWS_Pwm_00089 : PWM Module integrity check PUBLISHED	SWS_Pwm_00089 : PWM Module integrity check
MCAL-7536 - SWS_Pwm_00104 : Dem_ReportErrorStatus API PUBLISHED	SWS_Pwm_00104 : Dem_ReportErrorStatus API
MCAL-7617 - SWS_Pwm_00001 : PWM Emulation PUBLISHED	SWS_Pwm_00001 : PWM Emulation
MCAL-7625 - SWS_Pwm_00070 : PWM Time Unit PUBLISHED	SWS_Pwm_00070 : PWM Time Unit
MCAL-7536 - 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)

- Specifically to EPWM IP, DeadBand, Trip Zone and PWM Chopper sub-modules are bypassed.

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 EPWM module is expected to be on before calling any PWM module API.
2. The EPWM driver as such doesn't perform any PRCM programming to get the functional clock.
3. For EPWM IP, the modules executes at the system clock which is typically 125Mhz.
4. The clock-source selection for EPWM is not performed by the PWM driver, other entities such as SBL, MCAL module MCU shall perform the same.

Design Identifier	Description
MCAL-7617 - 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 EPWM module. The EPWM module configurator shall refer to MCU clock source as listed in specification.

- EPWM module executes at the system clock which is typically 125Mhz.
- 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 EPWM line is an overflow event. If a match event occurs first,it does not toggle the EPWM line

3.7.1 3.7.1 EPWM

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
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 ePWM module is supported. **Software Platforms**
- Bare-Metal

3.9 **3.9 Dependencies** 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_PWM

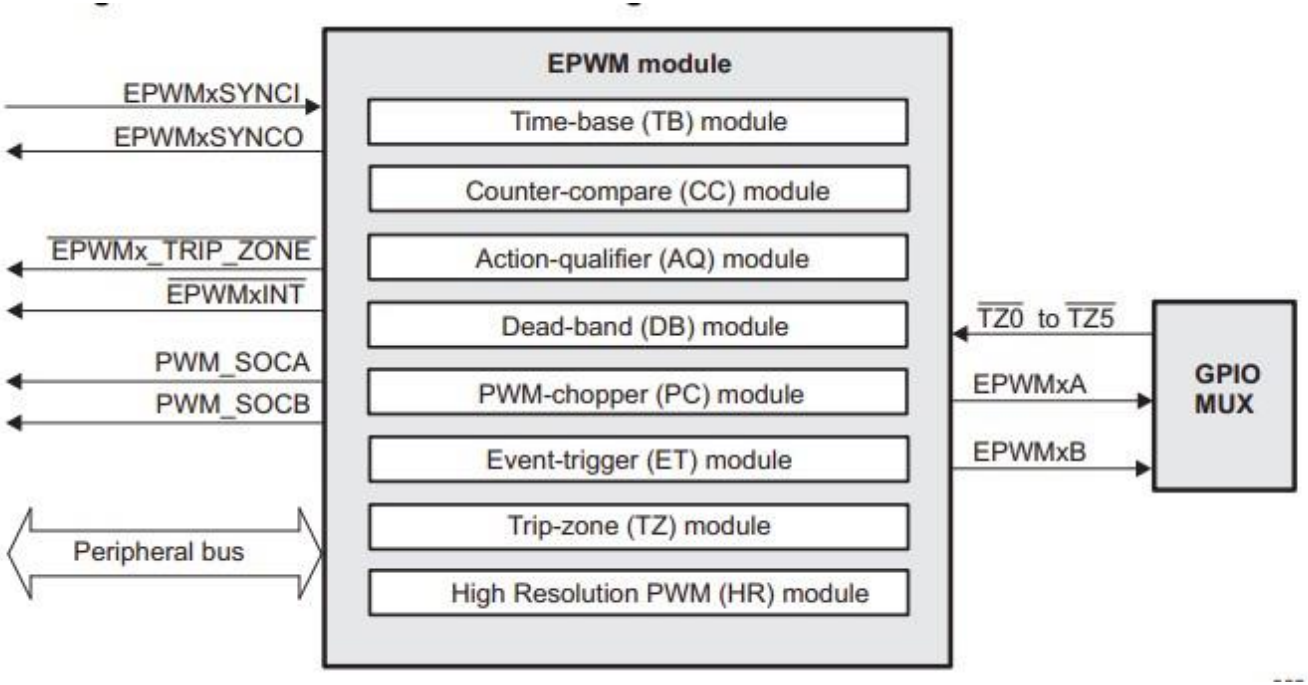
	Specification	Comment/Link
2	BSW General Requirements / Coding guidelines	Autosar and Coding guidelines for the Mcal drivers.
3	Software Product Specification (SPS)	Product Functional requirements.
4	Software Architecture	Mcal Software Architecture.

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 EPWM



EPWM Submodules

EPWM Module in J721E/J7200 consist of submodules:

Time-base (TB)

- Scale the time-base clock (TBCLK) relative to the system clock (FICLK).
- Configure the PWM time-base counter (TBCNT) frequency or period.
- Time-base counter mode selection:
    a. count-up mode: used for asymmetric PWM
    b. count-down mode: used for asymmetric PWM
    c. count-up-and-down mode: used for symmetric PWM
- Configure the time-base phase relative to another EPWM module.
- Synchronize the time-base counter between modules through hardware or software.
- Configure the direction (up or down) of the time-base counter after a synchronization event.
- Configure how the time-base counter will behave when the device is halted by an emulator.
- Specify the source for the synchronization output of the EPWM module:
    a. Synchronization input signal
    b. Time-base counter equal to zero
    c. Time-base counter equal to counter-compare B (CMPB)
    d. No output synchronization signal generated. **Counter-compare (CC)**
- Specify the PWM duty cycle for output EPWMxA and/or output EPWMxB
- Specify the time at which switching events occur on the EPWMxA or EPWMxB output

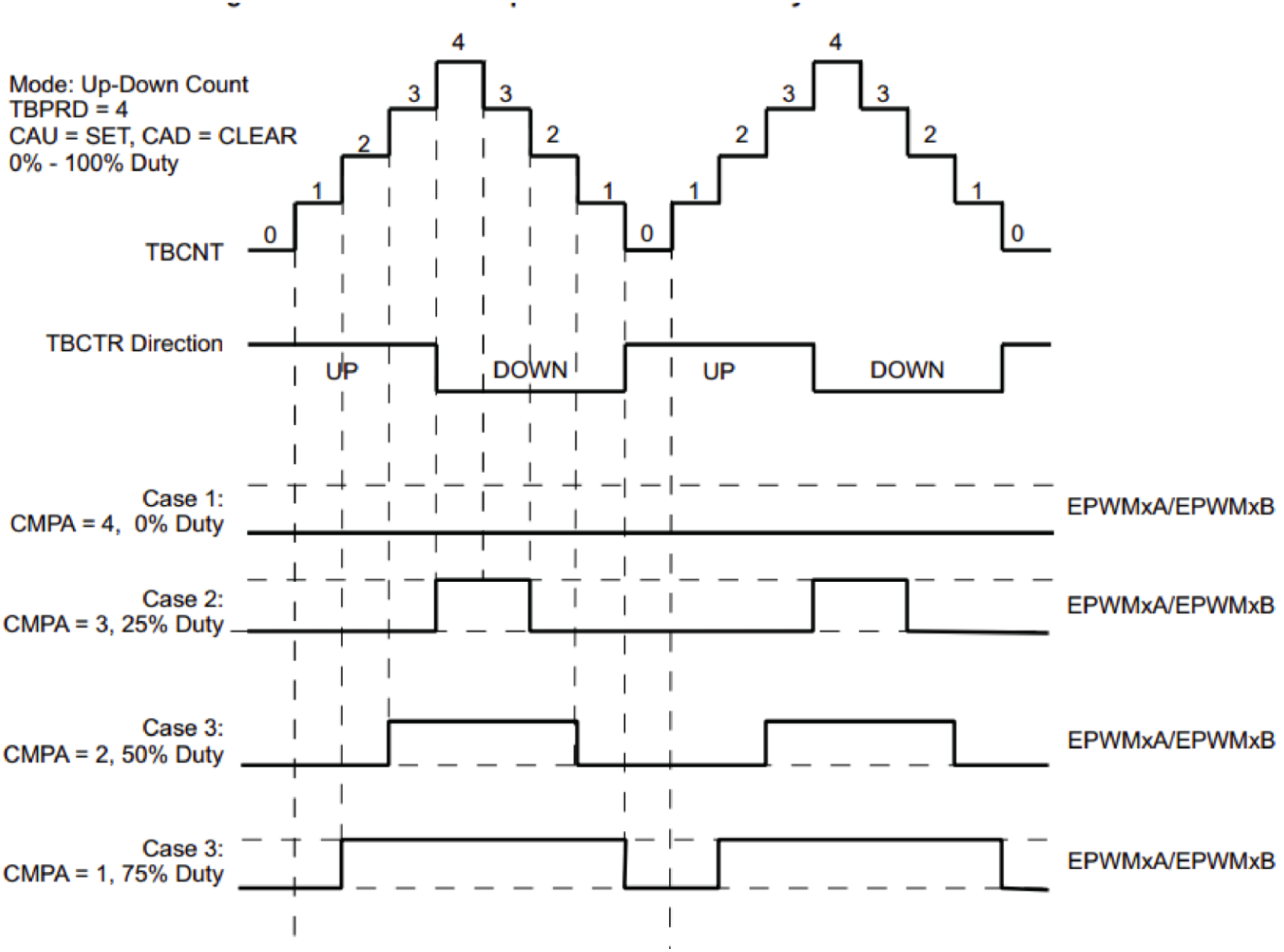
Action-qualifier (AQ)

- Controls how the two outputs EPWMxA and EPWMxB behave when a particular event occurs.
- Possible actions are: Set High, Clear Low, Toggle and Do nothing.

- Specify the type of action taken when a time-base or counter-compare submodule event occurs: a. No action taken
    b. Output EPWMxA and/or EPWMxB switched high
    c. Output EPWMxA and/or EPWMxB switched low
    d. Output EPWMxA and/or EPWMxB toggled
- Force the PWM output state through software control
- Configure and control the PWM dead-band through software

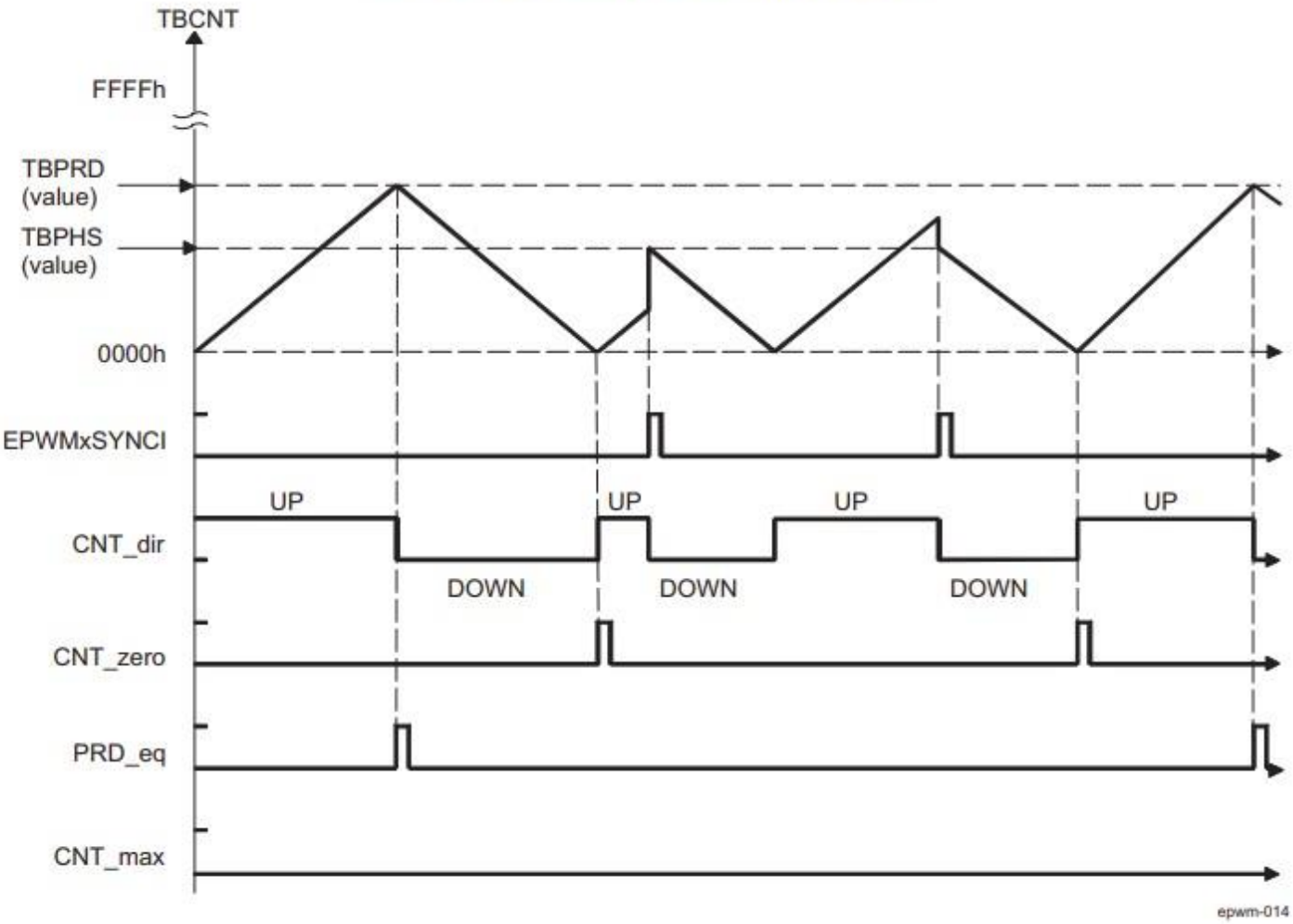
Event-trigger (ET)

- Enable the EPWM events that will trigger an interrupt.
- Specify the rate at which events cause triggers (every occurrence or every second or third occurrence)
- Poll, set, or clear event flags
- Manages the events generated by the time-base submodule and the counter-compare submodule to generate an aggregated interrupt request.
- An event can be be of the following:
    a. Time-base counter equal to 0.
    b. Time-base counter equal to period.
    c. Time-base counter equal to the compare A or B register when timer is incr/decr **High-Resolution PWM (HRPWM)**
- Enable extended time resolution capabilities with regards to falling edge.
- Configure finer time granularity control or edge positioning.
- Once the EPWM has been configured to provide conventional PWM of a given frequency and polarity, the HRPWM is configured by programming the control HR control register.
- Typical low-frequency PWM operations (below 250kHz) may not require the HRPWM capabilities.
- HRPWM is based on micro edge positioner (MEP) technology. MEP logic is capable of positioning an edge finely by sub-diving one coarse system clock of a conventional PWM generator. This driver assumes a MEP value of 180 ps.


Mode: Up-Down Count
TBPRD = 4
CAU = SET, CAD = CLEAR
0% - 100% Duty
TBCNT
TBCTR Direction
UP
DOWN
UP
DOWN
Case 1:
CMPA = 4, 0% Duty
Case 2:
CMPA = 3, 25% Duty
Case 3:
CMPA = 2, 50% Duty
Case 3:
CMPA = 1, 75% Duty
EPWMxA/EPWMxB
EPWMxA/EPWMxB
EPWMxA/EPWMxB
EPWMxA/EPWMxB

EPWM Waveform : Duty Cycle vs Compare Period Registers

Count Down on Synchronization Event



EPWM Waveform : Up/Down Counter Example

4.2 4.2 Dynamic Behavior

4.2.1 4.2.1 States for EPWM

- Before Pwm_Init() - PWM_STATUS_UNINIT
- After Pwm_Init() - PWM_STATUS_INIT
- After Pwm_DeInit() - PWM_STATUS_UNINIT

4.3 4.3 Time Unit Ticks

Refer 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-7625 - SWS_Pwm_00070 : PWM Time Unit PUBLISHED	SWS_Pwm_00070 : PWM Time Unit

4.4 4.4 Duty Cycle Resolution and Scaling

Refer 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-7544 - SWS_Pwm_00058 : PWM Duty cycle Parameter Width PUBLISHED	SWS_Pwm_00058 : PWM Duty cycle Parameter Width
MCAL-7547 - SWS_Pwm_00059 : PWM Duty Cycle Scaling Scheme PUBLISHED	SWS_Pwm_00059 : PWM Duty Cycle Scaling Scheme

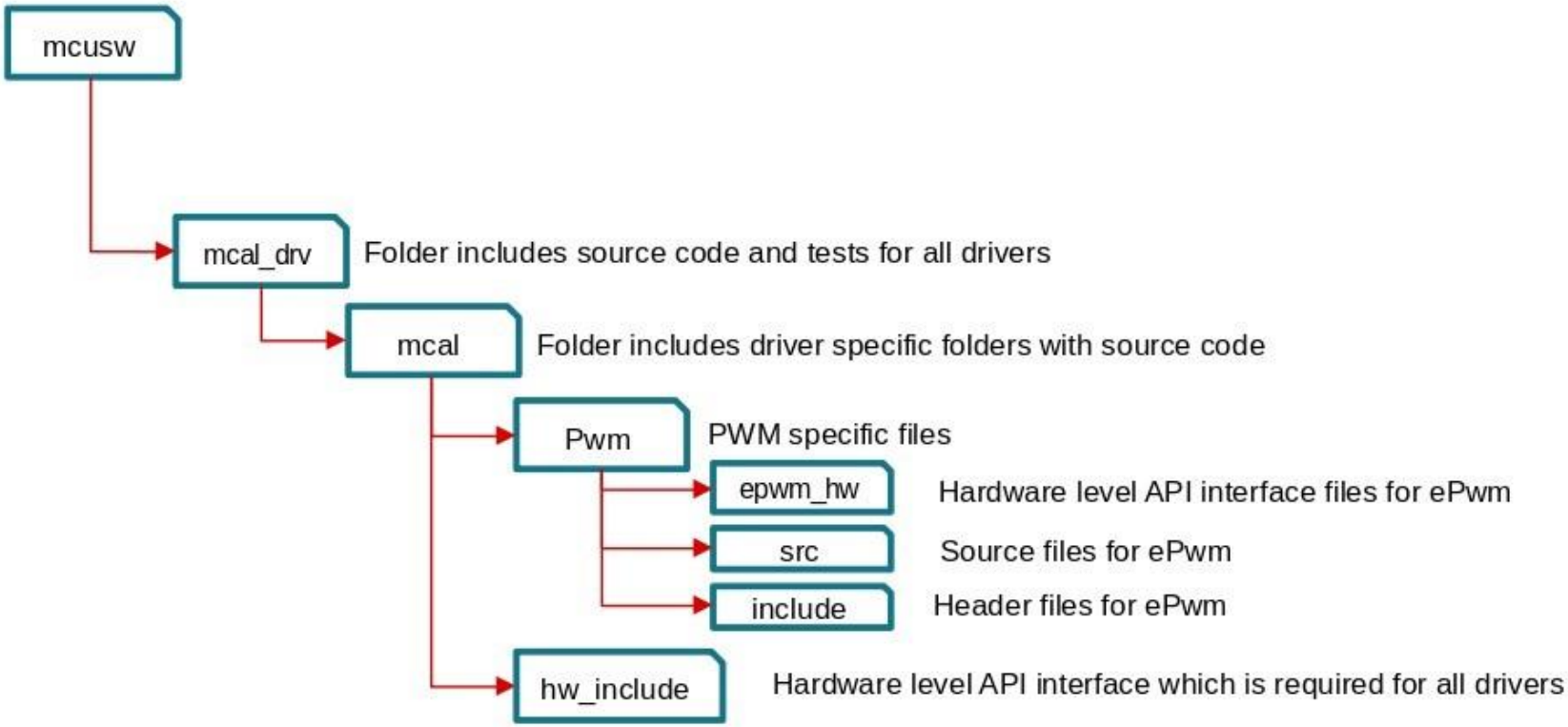
4.5 4.5 Directory Structure

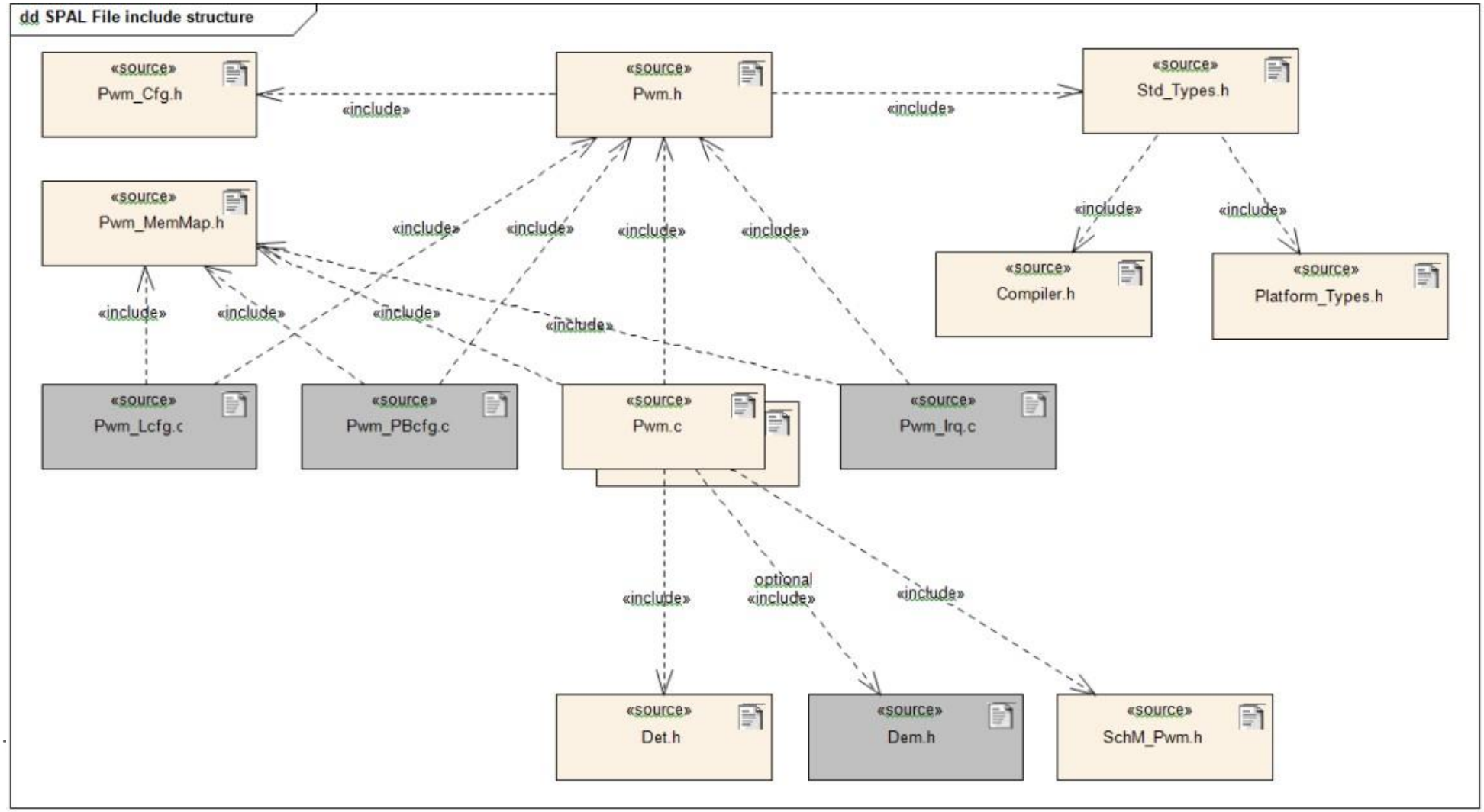
The directory structure is as depicted in figures below, the source files can be categorized under “Driver Implementation” and “Example Application” **Driver**

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.

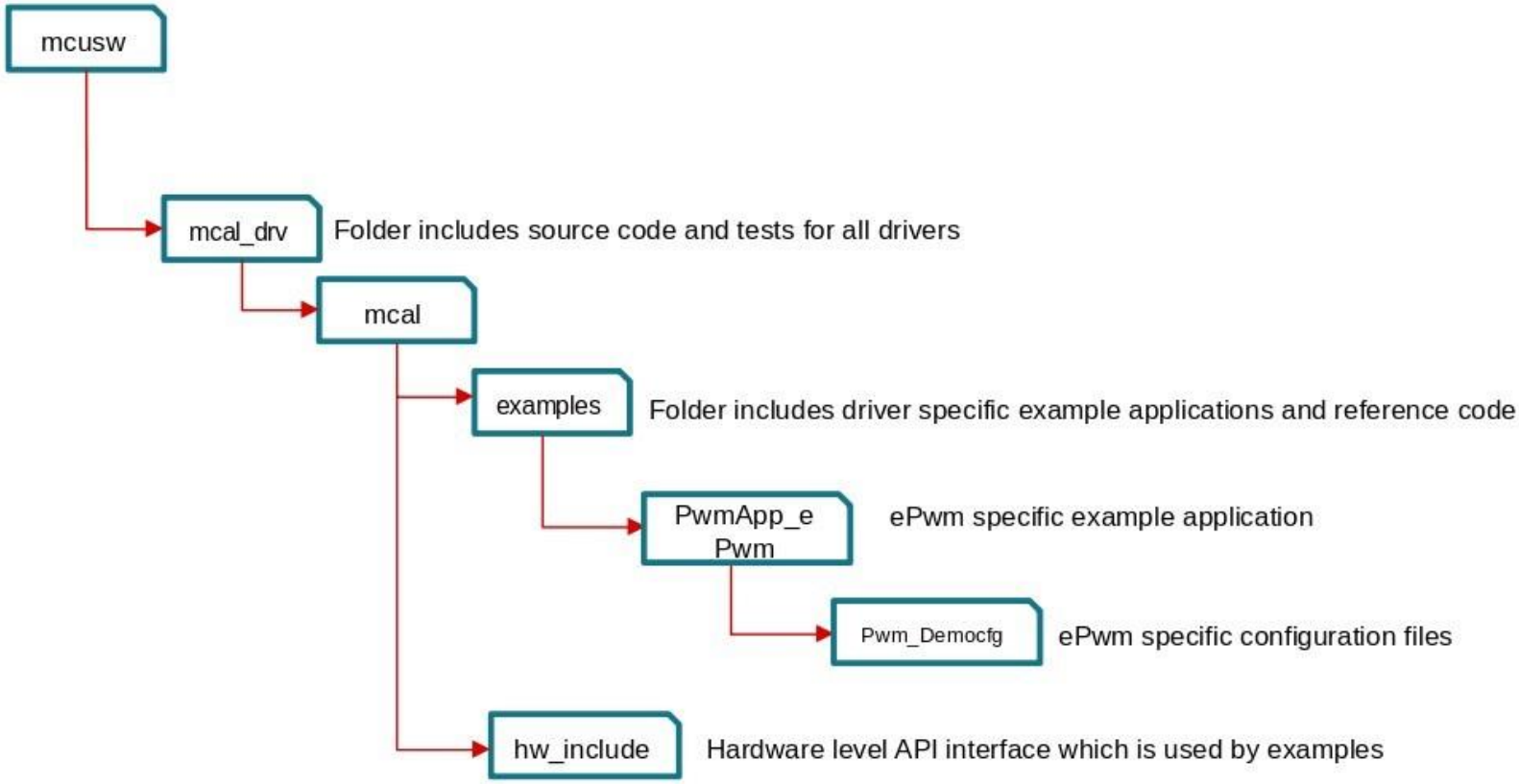
- EPWM specific API are present in Pwm_Ehrpwm.c.
- hw_include : Shall be used by all drivers..


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 ePwm
src
Source files for ePwm
include
Header files for ePwm
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
examples
Folder includes driver specific example applications and reference code
PwmApp_ePwm
ePwm specific example application
Pwm_Democfg
ePwm specific configuration files
hw_include
Hardware level API interface which is used by examples

Design Identifier	Description
MCAL-7638 - SWS_Pwm_00065 : PWM General File Structure PUBLISHED	SWS_Pwm_00065 : PWM General File Structure
MCAL-7613 - SWS_Pwm_60075 : PWM File Structure PBCfg.h PUBLISHED	SWS_Pwm_60075 : PWM File Structure PBCfg.h
MCAL-7572 - SWS_Pwm_50075 : PWM File Structure Det.h PUBLISHED	SWS_Pwm_50075 : PWM File Structure Det.h
MCAL-7551 - SWS_Pwm_10075 : PWM File Structure Cfg.h PUBLISHED	SWS_Pwm_10075 : PWM File Structure Cfg.h

Design Identifier	Description
MCAL-7639 - SWS_Pwm_70075 : PWM File Structure Pwm_Irq.h PUBLISHED	SWS_Pwm_70075 : PWM File Structure Pwm_Irq.h
MCAL-7570 - 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.

Design Identifier	Description
MCAL-7578 - ECUC_Pwm_00004 : PwmGeneral PUBLISHED	ECUC_Pwm_00004 : PwmGeneral

Design Identifier	Description
MCAL-7588 - ECUC_Pwm_00131 : PwmDevErrorDetect PUBLISHED	ECUC_Pwm_00131 : PwmDevErrorDetect
MCAL-7567 - ECUC_Pwm_00132 : PwmDutycycleUpdatedEndperiod PUBLISHED	ECUC_Pwm_00132 : PwmDutycycleUpdatedEndperiod
MCAL-7608 - ECUC_Pwm_00139 : PwmIndex PUBLISHED	ECUC_Pwm_00139 : PwmIndex
MCAL-7584 - ECUC_Pwm_00133 : PwmNotificationSupported PUBLISHED	ECUC_Pwm_00133 : PwmNotificationSupported
MCAL-7599 - ECUC_Pwm_00134 : PwmPeriodUpdatedEndperiod PUBLISHED	ECUC_Pwm_00134 : PwmPeriodUpdatedEndperiod
MCAL-7633 - ECUC_Pwm_00027 : PwmChannel PUBLISHED	ECUC_Pwm_00027 : PwmChannel

Design Identifier	Description
MCAL-7597 - ECUC_Pwm_00136 : PwmChannelClass PUBLISHED	ECUC_Pwm_00136 : PwmChannelClass
MCAL-7566 - ECUC_Pwm_00137 : PwmChannelId PUBLISHED	ECUC_Pwm_00137 : PwmChannelId
MCAL-7643 - ECUC_Pwm_00138 : PwmDutycycleDefault PUBLISHED	ECUC_Pwm_00138 : PwmDutycycleDefault
MCAL-7553 - ECUC_Pwm_00122 : PwmIdleState PUBLISHED	ECUC_Pwm_00122 : PwmIdleState
MCAL-7531 - ECUC_Pwm_00123 : PwmNotification PUBLISHED	ECUC_Pwm_00123 : PwmNotification
MCAL-7519 - ECUC_Pwm_00124 : PwmPeriodDefault PUBLISHED	ECUC_Pwm_00124 : PwmPeriodDefault

Design Identifier	Description
MCAL-7591 - ECUC_Pwm_00125 : PwmPolarity PUBLISHED	ECUC_Pwm_00125 : PwmPolarity
MCAL-7522 - ECUC_Pwm_00140 : PwmChannelConfigSet PUBLISHED	ECUC_Pwm_00140 : PwmChannelConfigSet
MCAL-7557 - ECUC_Pwm_00126 : PwmConfigurationOfOptApiServices PUBLISHED	ECUC_Pwm_00126 : PwmConfigurationOfOptApiServices
MCAL-7541 - ECUC_Pwm_00128 : PwmSetDutyCycle PUBLISHED	ECUC_Pwm_00128 : PwmSetDutyCycle
MCAL-7506 - ECUC_Pwm_00129 : PwmSetOutputToIdle PUBLISHED	ECUC_Pwm_00129 : PwmSetOutputToIdle

Design Identifier	Description
MCAL-7520 - ECUC_Pwm_00130 : PwmSetPeriodAndDuty PUBLISHED	ECUC_Pwm_00130 : PwmSetPeriodAndDuty
MCAL-7573 - ECUC_Pwm_00135 : PwmVersionInfoApi PUBLISHED	ECUC_Pwm_00135 : PwmVersionInfoApi
MCAL-7621 - 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-7545 - TI Specific ECUC :PwmFunctionalClock PUBLISHED	TI Specific ECUC :PwmFunctionalClock
MCAL-7502 - TI Specific ECUC : PwmClkPrescaler PUBLISHED	TI Specific ECUC : PwmClkPrescaler
MCAL-7620 - TI Specific ECUC : PwmHSClkPrescaler PUBLISHED	TI Specific ECUC : PwmHSClkPrescaler

Design Identifier	Description
MCAL-7605 - TI Specific ECUC : PwmDeviceVariant PUBLISHED	TI Specific ECUC : PwmDeviceVariant
MCAL-7510 - TI Specific ECUC : PwmDefaultOSCounterId PUBLISHED	TI Specific ECUC : PwmDefaultOSCounterId
MCAL-7585 - TI Specific ECUC: PwmOsCounterRef PUBLISHED	TI Specific ECUC: PwmOsCounterRef
MCAL-7623 - TI Specific ECUC: PwmTimeoutDuration PUBLISHED	TI Specific ECUC: PwmTimeoutDuration
MCAL-7601 - TI Specific ECUC : PwmTypeofInterruptFunction PUBLISHED	TI Specific ECUC : PwmTypeofInterruptFunction

4.6.1.1 4.6.1.1 EPWM specific parameters:

Parameter	Usage comment
PwmOutputChSelect	PWM Output Channel select EPWMxA or EPWMxB or Both. Both Outputs available with same duty cycle, period and polarity.
PwmHSClkPrescaler	This parameter allows the selection of High Speed pre-scalar value. The High Speed prescaler stage is clocked with the pwm clock and acts as a clock divider for the time-base clock. High-Speed Time-base Clock Prescale Bits.
PwmEnableHighRes	This parameter will be a switch to enable or disable high resolution (HRPWM) capability of EPWM module.

Standard Config which will be used to EHRPWM mode of operation.

Parameter	Usage comment
PwmIndex	Specifies the InstanceId of this module instance.

- 0 - EPWM will be used (Default)

Output Channel Structure:

- EpwmOuputCh - to choose between the channel A, B or both.

- EPWM_OUTPUT_CH_A - Output channel A
- EPWM_OUTPUT_CH_B - Output channel B
- EPWM_OUTPUT_CH_BOTH_A_AND_B - Both Output channel A and B

4.6.2 4.6.2 Variant Support

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

Design Identifier	Description
MCAL-7629 - 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*	*Type of Error*
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*	*Type of Error*
API Pwm_SetPowerState is called while the PWM module is still in use	PWM_E_NOT_DISENGAGED	0x16

Design Identifier	Description
MCAL-7555 - SWS_Pwm_10002 : PWM_Init API: Developement Error Detection PWM_E_PARAM_CONFIG PUBLISHED	SWS_Pwm_10002 : PWM_Init API: Development Error Detection PWM_E_PARAM_CONFIG

Design Identifier	Description
MCAL-7611 - SWS_Pwm_20002 : PWM Without Init : Developement Error Detection PWM_E_UNINIT PUBLISHED	SWS_Pwm_20002 : PWM Without Init : Development Error Detection PWM_E_UNINIT
MCAL-7523 - SWS_Pwm_30002 : PWM Developement Error Detection PWM_E_PARAM_CHANNEL PUBLISHED	SWS_Pwm_30002 : PWM Development Error Detection PWM_E_PARAM_CHANNEL

Design Identifier	Description
MCAL-7616 - SWS_Pwm_40002 : PWM Developement Error Detection PWM_E_PERIOD_UNCHANGEABLE PUBLISHED	SWS_Pwm_40002 : PWM Development Error Detection PWM_E_PERIOD_UNCHANGEABLE
MCAL-7574 - SWS_Pwm_50002 : PWM_Init : Developement Error Detection PWM_E_ALREADY_INITIALIZED PUBLISHED	SWS_Pwm_50002 : PWM_Init : Development Error Detection PWM_E_ALREADY_INITIALIZED

Design Identifier	Description
MCAL-7548 - SWS_Pwm_00151 : Pwm_GetVersionInfo API : Developement Error Detection PWM_E_PARAM_POINTER PUBLISHED	SWS_Pwm_00151 : Pwm_GetVersionInfo API : Development Error Detection PWM_E_PARAM_POINTER
MCAL-7527 - SWS_Pwm_00201 : Pwm Development Error Types PUBLISHED	SWS_Pwm_00201 : Pwm Development Error Types
MCAL-7529 - 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 5.1.8 Pwm_epwmOutputCh_t

used for EPWM outputs in a single epwm channel

5.1.9 5.1.9 Pwm_ChannelConfigType

Structure represents a Pwm channel configuration

5.1.10 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
pwmTbCtl	EPWM	uint16	0 to 0xFFFF	Time base Control register
pwmTbPhs	EPWM	uint16	0 to 0xFFFF	Time base Counter Phase register
pwmTbCnt	EPWM	uint16	0 to 0xFFFF	Time base Counter Register

Design Identifier	Description
MCAL-7505 - SWS_Pwm_00106 : Pwm_ChannelType PUBLISHED	SWS_Pwm_00106 : Pwm_ChannelType
MCAL-7587 - SWS_Pwm_00107 : Pwm_PeriodType PUBLISHED	SWS_Pwm_00107 : Pwm_PeriodType
MCAL-7543 - SWS_Pwm_00108 : Pwm_OutputStateType PUBLISHED	SWS_Pwm_00108 : Pwm_OutputStateType

Design Identifier	Description
MCAL-7576 - SWS_Pwm_00109 : Pwm_EdgeNotificationType PUBLISHED	SWS_Pwm_00109 : Pwm_EdgeNotificationType
MCAL-7589 - SWS_Pwm_00110 : Pwm_ChannelClassType PUBLISHED	SWS_Pwm_00110 : Pwm_ChannelClassType
MCAL-7535 - SWS_Pwm_00111 : Pwm_ConfigType Structure PUBLISHED	SWS_Pwm_00111 : Pwm_ConfigType Structure

Design Identifier	Description
MCAL-7533 - SWS_Pwm_00061 : Pwm_ConfigType PUBLISHED	SWS_Pwm_00061 : Pwm_ConfigType
MCAL-7590 - 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

Parameter	Description	Possible Value ranges	Unit of Value	Default Value	Variant
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 EPWM Module

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

- **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.
- **Information redundancy techniques:** Application can implement this using GPIO inputs used as interrupts. The interrupts can be timestamped to provide a check on the pulse widths.
- **Monitoring by eCAP:** Application can use external eCAP module to monitor the PWM outputs.
- **PWM4 and PWM5. Periodic software readback of static configuration registers and writen configuration registers**

EPWM driver provides the API Pwm_RegisterReadback() to read the static configuration registers and written configuration registers, The system integrator shall use the API to periodically read the static config registers. A snapshot of the values shall be taken after initializing the config registers and stored by the application. Periodically, the application can use the API to get the config values at runtime and compare.

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-7619 - 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-7610 - SWS_Pwm_00116 : Pwm_Init API : Function calls before init PUBLISHED	SWS_Pwm_00116 : Pwm_Init API : Function calls before init
MCAL-7552 - 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-7568 - SWS_Pwm_00121 : Pwm_Init API : Re-Initialization PUBLISHED	SWS_Pwm_00121 : Pwm_Init API : Re-Initialization
MCAL-7581 - SWS_Pwm_00007 : Pwm_Init API Variable configuration PUBLISHED	SWS_Pwm_00007 : Pwm_Init API Variable configuration
MCAL-7579 - SWS_Pwm_00062 : Pwm_Init API Resource configuration PUBLISHED	SWS_Pwm_00062 : Pwm_Init API Resource configuration

Design Identifier	Description
MCAL-7628 - SWS_Pwm_10009 : Pwm_Init API Channel configuration PUBLISHED	SWS_Pwm_10009 : Pwm_Init API Channel configuration
MCAL-7528 - SWS_Pwm_00052 : Pwm_Init API Disable Notification PUBLISHED	SWS_Pwm_00052 : Pwm_Init API Disable Notification
MCAL-7565 - SWS_Pwm_00093 : Pwm_Init API Call during runtime PUBLISHED	SWS_Pwm_00093 : Pwm_Init API Call during runtime
MCAL-7631 - 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-7635 - 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

Design Identifier	Description
MCAL-7598 - 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-7600 - SWS_Pwm_00010: Pwm_DeInit API PUBLISHED	SWS_Pwm_00010: Pwm_DeInit API
MCAL-7559 - SWS_Pwm_00011: Pwm_DeInit API Output Signal State PUBLISHED	SWS_Pwm_00011: Pwm_DeInit API Output Signal State
MCAL-7637 - SWS_Pwm_00012: Pwm_DeInit API Disable Interrupts and Notifications PUBLISHED	SWS_Pwm_00012: Pwm_DeInit API Disable Interrupts and Notifications

Design Identifier	Description
MCAL-7634 - SWS_Pwm_10080: Pwm_DeInit API Pre Compile time configurable PUBLISHED	SWS_Pwm_10080: Pwm_DeInit API Pre Compile time configurable
MCAL-7632 - SWS_Pwm_20080: Pwm_DeInit API Configurable by PwmDeInitApi PUBLISHED	SWS_Pwm_20080: Pwm_DeInit API Configurable by PwmDeInitApi
MCAL-7606 - SWS_Pwm_00117 : PWM Module : Development error PWM_E_UNINIT. PUBLISHED	SWS_Pwm_00117 : PWM Module : Development error PWM_E_UNINIT.
MCAL-7595 - SWS_Pwm_10051 : PWM Module Development error detection and reporting PUBLISHED	SWS_Pwm_10051 : PWM Module Development error detection and reporting
MCAL-7609 - 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-7636 - SWS_Pwm_00013: Pwm_SetDutyCycle API PUBLISHED	SWS_Pwm_00013: Pwm_SetDutyCycle API
MCAL-7509 - 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-7511 - SWS_Pwm_00016: Pwm_SetDutyCycle API Output signal Modulation PUBLISHED	SWS_Pwm_00016: Pwm_SetDutyCycle API Output signal Modulation
MCAL-7596 - SWS_Pwm_00017: Pwm_SetDutyCycle API Update Duty Cycle PUBLISHED	SWS_Pwm_00017: Pwm_SetDutyCycle API Update Duty Cycle
MCAL-7544 - SWS_Pwm_00058 : PWM Duty cycle Parameter Width PUBLISHED	SWS_Pwm_00058 : PWM Duty cycle Parameter Width

Design Identifier	Description
MCAL-7547 - SWS_Pwm_00059 : PWM Duty Cycle Scaling Scheme PUBLISHED	SWS_Pwm_00059 : PWM Duty Cycle Scaling Scheme
MCAL-7530 - SWS_Pwm_00018: Pwm_SetDutyCycle API Output Signal Spike PUBLISHED	SWS_Pwm_00018: Pwm_SetDutyCycle API Output Signal Spike
MCAL-7606 - SWS_Pwm_00117 : PWM Module : Development error PWM_E_UNINIT. PUBLISHED	SWS_Pwm_00117 : PWM Module : Development error PWM_E_UNINIT.
MCAL-7602 - SWS_Pwm_00047 : PWM Module : Development error PWM_E_PARAM_CHANNEL PUBLISHED	SWS_Pwm_00047 : PWM Module : Development error PWM_E_PARAM_CHANNEL
MCAL-7595 - SWS_Pwm_10051 : PWM Module Development error detection and reporting PUBLISHED	SWS_Pwm_10051 : PWM Module Development error detection and reporting
MCAL-7609 - 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-7642 - SWS_Pwm_10082: Pwm_SetDutyCycle API Pre Compile time configurable PUBLISHED	SWS_Pwm_10082: Pwm_SetDutyCycle API Pre Compile time configurable
MCAL-7537 - 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-7563 - SWS_Pwm_00019: Pwm_SetPeriodAndDuty API PUBLISHED	SWS_Pwm_00019: Pwm_SetPeriodAndDuty API
MCAL-7532 - SWS_Pwm_00076: Pwm_SetPeriodAndDuty API Period Update PUBLISHED	SWS_Pwm_00076: Pwm_SetPeriodAndDuty API Period Update

Design Identifier	Description
MCAL-7512 - SWS_Pwm_00020: Pwm_SetPeriodAndDuty API Output Signal Spike PUBLISHED	SWS_Pwm_00020: Pwm_SetPeriodAndDuty API Output Signal Spike
MCAL-7544 - SWS_Pwm_00058 : PWM Duty cycle Parameter Width PUBLISHED	SWS_Pwm_00058 : PWM Duty cycle Parameter Width
MCAL-7547 - SWS_Pwm_00059 : PWM Duty Cycle Scaling Scheme PUBLISHED	SWS_Pwm_00059 : PWM Duty Cycle Scaling Scheme
MCAL-7606 - SWS_Pwm_00117 : PWM Module : Development error PWM_E_UNINIT. PUBLISHED	SWS_Pwm_00117 : PWM Module : Development error PWM_E_UNINIT.
MCAL-7526 - SWS_Pwm_00045 : PWM Module : Development error PWM_E_PERIOD_UNCHANGEABLE PUBLISHED	SWS_Pwm_00045 : PWM Module : Development error PWM_E_PERIOD_UNCHANGEABLE
MCAL-7602 - SWS_Pwm_00047 : PWM Module : Development error PWM_E_PARAM_CHANNEL PUBLISHED	SWS_Pwm_00047 : PWM Module : Development error PWM_E_PARAM_CHANNEL

Design Identifier	Description
MCAL-7595 - SWS_Pwm_10051 : PWM Module Development error detection and reporting PUBLISHED	SWS_Pwm_10051 : PWM Module Development error detection and reporting
MCAL-7609 - SWS_Pwm_20051 : PWM Module Development erro r detection and skipping functionality PUBLISHED	SWS_Pwm_20051 : PWM Module Development error detection and skipping functionality
MCAL-7560 - SWS_Pwm_00041: Pwm_SetPeriodAndDuty API Channel PUBLISHED	SWS_Pwm_00041: Pwm_SetPeriodAndDuty API Channel
MCAL-7615 - SWS_Pwm_10083: Pwm_SetPeriodAndDuty API Pre Compile Time Configurable PUBLISHED	SWS_Pwm_10083: Pwm_SetPeriodAndDuty API Pre Compile Time Configurable
MCAL-7575 - SWS_Pwm_20083: Pwm_SetPeriodAndDuty API Configurable through PwmSetPeriodAndDuty PUBLISHED	SWS_Pwm_20083: Pwm_SetPeriodAndDuty API Configurable through PwmSetPeriodAndDuty
MCAL-7582 - 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-7582 - SWS_Pwm_00150: Pwm_SetPeriodAndDuty API : Period set as zero PUBLISHED	SWS_Pwm_00150: Pwm_SetPeriodAndDuty API : Period set as zero
MCAL-7626 - SWS_Pwm_00021 : Pwm_SetOutputToIdle API PUBLISHED	SWS_Pwm_00021 : Pwm_SetOutputToIdle API
MCAL-7515 - SWS_Pwm_10084 : Pwm_SetOutputToIdle API Pre Compile Time Configurable PUBLISHED	SWS_Pwm_10084 : Pwm_SetOutputToIdle API Pre Compile Time Configurable
MCAL-7542 - SWS_Pwm_20084 : Pwm_SetOutputToIdle API Configurable through PwmSetOutputToIdle PUBLISHED	SWS_Pwm_20084 : Pwm_SetOutputToIdle API Configurable through PwmSetOutputToIdle

Design Identifier	Description
MCAL-7604 - 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-7546 - SWS_Pwm_20086 : Pwm_SetOutputToIdle API : Re-activation through Pwm_SetDutyCycle API PUBLISHED	SWS_Pwm_20086 : Pwm_SetOutputToIdle API : Re-activation through Pwm_SetDutyCycle API
MCAL-7641 - SWS_Pwm_00119 : Pwm_SetOutputToIdle API : Re-activation for fixed period type channels through Pwm_SetDutyCycle 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-7539 - SWS_Pwm_00023 : Pwm_DisableNotification API PUBLISHED	SWS_Pwm_00023 : Pwm_DisableNotification API
MCAL-7627 - SWS_Pwm_10112 : Pwm_DisableNotification API Pre-compile time configurable PUBLISHED	SWS_Pwm_10112 : Pwm_DisableNotification API Pre-compile time configurable
MCAL-7594 - 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-7606 - SWS_Pwm_00117 : PWM Module : Development error PWM_E_UNINIT. PUBLISHED	SWS_Pwm_00117 : PWM Module : Development error PWM_E_UNINIT.
MCAL-7602 - SWS_Pwm_00047 : PWM Module : Development error PWM_E_PARAM_CHANNEL PUBLISHED	SWS_Pwm_00047 : PWM Module : Development error PWM_E_PARAM_CHANNEL

Design Identifier	Description
MCAL-7595 - SWS_Pwm_10051 : PWM Module Development error detection and reporting PUBLISHED	SWS_Pwm_10051 : PWM Module Development error detection and reporting
MCAL-7609 - SWS_Pwm_20051 : PWM Module Development erro r detection and skipping functionality PUBLISHED	SWS_Pwm_20051 : PWM Module Development error detection and skipping functionality
MCAL-7619 - 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-7586 - SWS_Pwm_00024 : Pwm_EnableNotification PUBLISHED	SWS_Pwm_00024 : Pwm_EnableNotification
MCAL-7606 - SWS_Pwm_00117 : PWM Module : Development error PWM_E_UNINIT. PUBLISHED	SWS_Pwm_00117 : PWM Module : Development error PWM_E_UNINIT.
MCAL-7554 - SWS_Pwm_00081 : Pwm_EnableNotification Cancel Pending interrupts PUBLISHED	SWS_Pwm_00081 : Pwm_EnableNotification Cancel Pending interrupts
MCAL-7593 - SWS_Pwm_10113 : Pwm_EnableNotification : Pre-compile time configurable PUBLISHED	SWS_Pwm_10113 : Pwm_EnableNotification : Pre-compile time configurable
MCAL-7624 - 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

Design Identifier	Description
MCAL-7602 - SWS_Pwm_00047 : PWM Module : Development error PWM_E_PARAM_CHANNEL PUBLISHED	SWS_Pwm_00047 : PWM Module : Development error PWM_E_PARAM_CHANNEL
MCAL-7595 - SWS_Pwm_10051 : PWM Module Development error detection and reporting PUBLISHED	SWS_Pwm_10051 : PWM Module Development error detection and reporting
MCAL-7609 - SWS_Pwm_20051 : PWM Module Development erro r detection and skipping functionality PUBLISHED	SWS_Pwm_20051 : PWM Module Development error 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-7606 - 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-7540 - 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 Integration of EPWM

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 EPWM driver implementation and interface should be simple and user friendly. The user should be able to choose which one to use.	Have EPWM as default selected : When all EPWM channels are used start using the GPT Channels **Advantages:** • Can use different channels at runtime. • GPTimer will be kept free and would be available for other applications. **Disadvantages:** • Incompatible with configurator model as Channel ID cannot change at runtime. • User not given the flexibility, and would make the design more complicated.	Based on EPWM/GPT up,	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).	

9.2 9.2 EHRPWM internal step size

HRPWM is based on micro edge positioner (MEP) technology. MEP logic is capable of positioning an edge finely by sub-diving one coarse system clock of a conventional PWM generator.

No.	Decision Criteria	Alternatives	Selected alternative	Rationale	Trade-offs
1	The EPWM driver implementation should avoid any failure risks.	The driver should assume the MEP to be 180ps for all scenarios. **Advantages:** • This will avoid any failures that come with varied MEP values if user is given freedom to choose. • Will make the driver more stable. **Disadvantages:** • Driver will only support fixed MEP value.	Based on micro edge positioner (MEP)	Option 1 is chosen. Typically MEP scale factor is calibrated in hardware. However, the current hardware does not have an in-built calibration/diagnostic module, so different MEP values cannot be analyzed at run-time. Keeping a fixed MEP value which is suggested by datasheet (180ps) would make the driver functionality simple and stable.	None
		The driver should give the user to select the MEP step size **Advantages:** • User has freedom to choose the MEP value. **Disadvantages:** • Will make driver complicated and prone to user error.			

9.3 **9.3 Support each EPWM Output as a PWM Channel**

The PWM module instance is composed of two PWM outputs: EPWMxA and EPWMxB. It is possible to consider each output as one channel. In this way we can support 12 PWM outputs in case of J721E/J7200 simultaneously.

N o.	**Decision Criteria**	Link each HW PWM instance as 1 individual PWM Channel **Advantages:**	Based on EPWMxA and EPWMxB	**Rationale**	**Trade-offs**
1	The EPWM driver implementation and interface should be easily controllable. **Alternatives Selected alternative**	• It is possible to link only one output of each instance as one individual PWM Channel. • Each PWM output is independent and frequency, duty cycle and polarity can be controlled independently. **Disadvantages:** • In this way we can support only 6 PWM outputs in case of TI SoCs simultaneously. • Interrupt can be generated only for one of the event output either EPWMxA or EPWMxB since each instance has only one interrupt line.		Typically in PWM use cases, it is recommended to have independent outputs so that you can control frequency, duty cycle and polarity can be independently. Even AUTOSAR provides API's to change frequency independently for each channel. So option 1 is decided for implementation.	None

No.	Decision Criteria	Alternatives	Selected alternative	Rationale	Trade-offs
		Link each HW PWM instance output(EPWMxA or EPWMxB) as 1 individual PWM Channel **Advantages:** • Each instance consists of two outputs(EPWMxA and EPWMxB) and can be considered as individual PWM Channels. • Each instance outputs can have a different duty cycle and polarity. • In this way we can support 12 PWM outputs in case of TI SoCs simultaneously. **Disadvantages:** • Each instance outputs are not independent. If frequency is changed for channel(EPWMxA) then it will be applied for 2nd channel also(EPWMxB). • Interrupt can be generated only for one of the event output either EPWMxA or EPWMxB since each instance has only one interrupt line. Other hardware/driver specific decisions taken are listed below.			

No.	Decision Criteria	Alternatives	Selected alternative	Rationale	Trade-offs
		• If the default period is configured to zero during driver initialization, then the output shall be zero (zero percent duty-cycle). • Always UpDown Counter mode is selected to achieve symmetrical PWM output. • Interrupt notification is generated only for one of the event output either EPWMxA or EPWMxB, whichever is selected, since each instance has only one interrupt line. • If interface parameter outputCh is selected as EPWM_OUTPUT_CH_BOTH_A_AND_B then dual outputs of each instance will be available. i.e same EPWMxA output is also available in EPWMxB signal with same configured frequency, duty cycle and polarity. Interrupt notification is generated only for EPWMxA output if this option is selected.			

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 behavior 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