

# PWM\_CCU4

Home

Data Structures

Data Structure Index

Data Fields

## Data Structures

Here are the data structures with brief descriptions:

-  [PWM\\_CCU4\\_ConfigType](#) **Configuration parameters of the PWM\_CCU4 APP**
-  [PWM\\_CCU4\\_HandleTypeDef](#) **Initialization parameters of the PWM\_CCU4 APP**

# PWM\_CCU4

[Home](#)

[Data Structures](#)

[Data Structure Index](#)

[Data Fields](#)

[Data Fields](#)

## PWM\_CCU4\_ConfigType Struct Reference

[Data structures](#)

## Detailed Description

Configuration parameters of the PWM\_CCU4 APP.

Definition at line **171** of file **PWM\_CCU4.h**.

```
#include <PWM_CCU4.h>
```

## Data Fields

const bool **start\_cont**

const uint16\_t **period\_val**

const uint16\_t **compare\_n**

const bool **int\_per\_m**

const bool **int\_cmp\_n**

const bool **int\_cmp\_n**

const bool **int\_one\_m**

const bool **int\_e0**

const bool **int\_e1**

const bool **int\_e2**

const XMC\_CCU4\_SLICE\_SR\_ID\_t **sr\_per\_ma**

const XMC\_CCU4\_SLICE\_SR\_ID\_t **sr\_cmp\_ma**

const XMC\_CCU4\_SLICE\_SR\_ID\_t **sr\_cmp\_ma**

const XMC\_CCU4\_SLICE\_SR\_ID\_t **sr\_one\_ma**

const XMC\_CCU4\_SLICE\_SR\_ID\_t **sr\_e0**

const XMC\_CCU4\_SLICE\_SR\_ID\_t **sr\_e1**

```
const XMC_CCU4_SLICE_SR_ID_t sr_e2
```

```
const  
XMC_CCU4_SLICE_EVENT_CONFIG_t event0_co  
*const
```

```
const  
XMC_CCU4_SLICE_EVENT_CONFIG_t event1_co  
*const
```

```
const  
XMC_CCU4_SLICE_EVENT_CONFIG_t event2_co  
*const
```

```
const XMC_CCU4_SLICE_EVENT_t ext_start_c
```

```
const XMC_CCU4_SLICE_START_MODE_t ext_start_m
```

```
const XMC_CCU4_SLICE_EVENT_t ext_stop_c
```

```
const XMC_CCU4_SLICE_END_MODE_t ext_stop_m
```

```
const XMC_CCU4_SLICE_EVENT_t ext_count_c
```

```
const XMC_CCU4_SLICE_EVENT_t ext_gate_c
```

```
const XMC_CCU4_SLICE_EVENT_t ext_count_m
```

```
const XMC_CCU4_SLICE_EVENT_t ext_load_c
```

```
const XMC_CCU4_SLICE_EVENT_t ext_mod_c
```

```
const  
XMC_CCU4_SLICE_MODULATION_MODE_t ext_mod_m
```

const bool **ext\_mod\_s**

const XMC\_CCU4\_SLICE\_EVENT\_t **ext\_overrid**

const XMC\_CCU4\_SLICE\_EVENT\_t **ext\_overrid**

const bool **ext\_trap\_e**

const XMC\_CCU4\_SLICE\_EVENT\_t **ext\_trap\_e**

const bool **ext\_trap\_s**

const  
XMC\_CCU4\_SLICE\_TRAP\_EXIT\_MODE\_t **ext\_trap\_e**

const  
XMC\_CCU4\_MULTI\_CHANNEL\_SHADOW\_TRANSFER\_t **mcm\_shad**

const  
XMC\_CCU4\_SLICE\_SHADOW\_TRANSFER\_MODE\_t **shadow\_tr**

const uint32\_t **immediate**

const uint32\_t **automatic**

const bool **cascaded\_**

const  
XMC\_CCU4\_SLICE\_COMPARE\_CONFIG\_t **ccu4\_cc4\_**  
\*const

const bool **gpio\_ch\_o**

XMC\_GPIO\_PORT\_t \*const **gpio\_ch\_o**

const uint8\_t **gpio\_ch\_o**

const XMC\_GPIO\_CONFIG\_t \*const **gpio\_ch\_o**

GLOBAL\_CCU4\_t \*const **global\_ccu**

## Field Documentation

**const uint32\_t PWM\_CCU4\_ConfigType::automatic\_shadow\_transf**

Defines the registers that are enabled for automatic shadow transfer

Definition at line [229](#) of file [PWM\\_CCU4.h](#).

Referenced by [PWM\\_CCU4\\_Init\(\)](#).

**const bool PWM\_CCU4\_ConfigType::cascaded\_shadow\_txfr\_enab**

Enables cascade of shadow transfer in timer concatenate mode

Definition at line [230](#) of file [PWM\\_CCU4.h](#).

Referenced by [PWM\\_CCU4\\_Init\(\)](#).

**const XMC\_CCU4\_SLICE\_COMPARE\_CONFIG\_t\* const PWM\_CCU**

Points to the variable CCU4 timer initialization

Definition at line [233](#) of file [PWM\\_CCU4.h](#).

Referenced by [PWM\\_CCU4\\_Init\(\)](#), [PWM\\_CCU4\\_SetFreq\(\)](#), and [PWM\\_CCU4\\_SetFreqAndDutyCycle\(\)](#).

**const uint16\_t PWM\_CCU4\_ConfigType::compare\_value**

Channel 1 compare register value. Determines the duty cycle

Definition at line [175](#) of file [PWM\\_CCU4.h](#).

Referenced by [PWM\\_CCU4\\_Init\(\)](#).

**const XMC\_CCU4\_SLICE\_EVENT\_CONFIG\_t\* const PWM\_CCU4\_C**

Points to the variable containing event 0 configuration

Definition at line [193](#) of file [PWM\\_CCU4.h](#).

**const XMC\_CCU4\_SLICE\_EVENT\_CONFIG\_t\* const PWM\_CCU4\_C**

Points to the variable containing event 1 configuration

Definition at line [194](#) of file [PWM\\_CCU4.h](#).

**const XMC\_CCU4\_SLICE\_EVENT\_CONFIG\_t\* const PWM\_CCU4\_C**

Points to the variable containing event 2 configuration

Definition at line [195](#) of file [PWM\\_CCU4.h](#).

**const XMC\_CCU4\_SLICE\_EVENT\_t PWM\_CCU4\_ConfigType::ext\_c**

Defines to which event external count direction signal is connected

Definition at line [203](#) of file [PWM\\_CCU4.h](#).

**const XMC\_CCU4\_SLICE\_EVENT\_t PWM\_CCU4\_ConfigType::ext\_c**

Defines to which event external count signal is connected

Definition at line [207](#) of file [PWM\\_CCU4.h](#).

**const XMC\_CCU4\_SLICE\_EVENT\_t PWM\_CCU4\_ConfigType::ext\_c**

Defines to which event external gating signal is connected

Definition at line [205](#) of file [PWM\\_CCU4.h](#).

### **const XMC\_CCU4\_SLICE\_EVENT\_t PWM\_CCU4\_ConfigType::ext\_I**

Defines to which event external load signal is connected

Definition at line [209](#) of file [PWM\\_CCU4.h](#).

### **const XMC\_CCU4\_SLICE\_EVENT\_t PWM\_CCU4\_ConfigType::ext\_I**

Defines to which event external modulation signal is connected

Definition at line [211](#) of file [PWM\\_CCU4.h](#).

### **const XMC\_CCU4\_SLICE\_MODULATION\_MODE\_t PWM\_CCU4\_ConfigType::ext\_mod\_mode**

Defines mode of external modulation

Definition at line [212](#) of file [PWM\\_CCU4.h](#).

### **const bool PWM\_CCU4\_ConfigType::ext\_mod\_sync**

Defines mode of synchronization for external modulation

Definition at line [213](#) of file [PWM\\_CCU4.h](#).

### **const XMC\_CCU4\_SLICE\_EVENT\_t PWM\_CCU4\_ConfigType::ext\_c**

Defines to which event external edge override signal is connected

Definition at line [215](#) of file `PWM_CCU4.h`.

### `const XMC_CCU4_SLICE_EVENT_t PWM_CCU4_ConfigType::ext_c`

Defines to which event external level override signal is connected

Definition at line [217](#) of file `PWM_CCU4.h`.

### `const XMC_CCU4_SLICE_EVENT_t PWM_CCU4_ConfigType::ext_s`

Defines to which event external start signal is connected

Definition at line [197](#) of file `PWM_CCU4.h`.

Referenced by `PWM_CCU4_Start()`.

### `const XMC_CCU4_SLICE_START_MODE_t PWM_CCU4_ConfigTyp`

Defines mode of starting the timer

Definition at line [198](#) of file `PWM_CCU4.h`.

### `const XMC_CCU4_SLICE_EVENT_t PWM_CCU4_ConfigType::ext_s`

Defines to which event external stop signal is connected

Definition at line [200](#) of file `PWM_CCU4.h`.

### `const XMC_CCU4_SLICE_END_MODE_t PWM_CCU4_ConfigType::`

Defines mode of stopping the timer

Definition at line [201](#) of file `PWM_CCU4.h`.

**const bool PWM\_CCU4\_ConfigType::ext\_trap\_enable**

Enables the trap

Definition at line [219](#) of file [PWM\\_CCU4.h](#).

**const XMC\_CCU4\_SLICE\_EVENT\_t PWM\_CCU4\_ConfigType::ext\_t**

Defines to which event external trap signal is connected

Definition at line [220](#) of file [PWM\\_CCU4.h](#).

**const XMC\_CCU4\_SLICE\_TRAP\_EXIT\_MODE\_t PWM\_CCU4\_ConfigType::exit\_mode**

Defines mode of exiting trap state

Definition at line [222](#) of file [PWM\\_CCU4.h](#).

**const bool PWM\_CCU4\_ConfigType::ext\_trap\_sync**

Defines mode of synchronization

Definition at line [221](#) of file [PWM\\_CCU4.h](#).

**GLOBAL\_CCU4\_t\* const PWM\_CCU4\_ConfigType::global\_ccu4\_handle**

Points to GLOBAL\_CCU4 APP handle

Definition at line [240](#) of file [PWM\\_CCU4.h](#).

Referenced by [PWM\\_CCU4\\_Init\(\)](#).

**const XMC\_GPIO\_CONFIG\_t\* const PWM\_CCU4\_ConfigType::gpio**

Points to the variable containing GPIO configuration

Definition at line [238](#) of file [PWM\\_CCU4.h](#).

Referenced by [PWM\\_CCU4\\_Init\(\)](#).

**const bool PWM\_CCU4\_ConfigType::gpio\_ch\_out\_enable**

Enables GPIO initialization for channel 1 direct output

Definition at line [235](#) of file [PWM\\_CCU4.h](#).

Referenced by [PWM\\_CCU4\\_Init\(\)](#).

**const uint8\_t PWM\_CCU4\_ConfigType::gpio\_ch\_out\_pin**

Pin number in the selected PORT

Definition at line [237](#) of file [PWM\\_CCU4.h](#).

Referenced by [PWM\\_CCU4\\_Init\(\)](#).

**XMC\_GPIO\_PORT\_t\* const PWM\_CCU4\_ConfigType::gpio\_ch\_out**

Points to PORT BASE address

Definition at line [236](#) of file [PWM\\_CCU4.h](#).

Referenced by [PWM\\_CCU4\\_Init\(\)](#).

**const uint32\_t PWM\_CCU4\_ConfigType::immediate\_write**

Defines the registers that are enabled for immediate shadow transfer

Definition at line [228](#) of file [PWM\\_CCU4.h](#).

Referenced by [PWM\\_CCU4\\_Init\(\)](#).

### **const bool PWM\_CCU4\_ConfigType::int\_cmp\_match\_down**

Enables event service request generation when timer is counting down and equals to channel 1 compare register

Definition at line [179](#) of file [PWM\\_CCU4.h](#).

### **const bool PWM\_CCU4\_ConfigType::int\_cmp\_match\_up**

Enables event service request generation when timer is counting up and equals to channel 1 compare register

Definition at line [178](#) of file [PWM\\_CCU4.h](#).

### **const bool PWM\_CCU4\_ConfigType::int\_e0**

Enables event service request generation by external event 0 signal based on the trigger edge selection

Definition at line [181](#) of file [PWM\\_CCU4.h](#).

### **const bool PWM\_CCU4\_ConfigType::int\_e1**

Enables event service request generation by external event 1 signal based on the trigger edge selection

Definition at line [182](#) of file [PWM\\_CCU4.h](#).

### **const bool PWM\_CCU4\_ConfigType::int\_e2**

Enables event service request generation by external event 2 signal based on the trigger edge selection

Definition at line [183](#) of file [PWM\\_CCU4.h](#).

### **const bool PWM\_CCU4\_ConfigType::int\_one\_match\_down**

Enables event service request generation when timer is counting down and equals 1

Definition at line [180](#) of file [PWM\\_CCU4.h](#).

### **const bool PWM\_CCU4\_ConfigType::int\_per\_match**

Enables event service request generation when timer value equals to period register

Definition at line [177](#) of file [PWM\\_CCU4.h](#).

### **const XMC\_CCU4\_MULTI\_CHANNEL\_SHADOW\_TRANSFER\_t PWM**

Defines the mode of shadow transfer in multi channel mode operation

Definition at line [224](#) of file [PWM\\_CCU4.h](#).

Referenced by [PWM\\_CCU4\\_Init\(\)](#).

### **const uint16\_t PWM\_CCU4\_ConfigType::period\_value**

Period register value. Determines the frequency

Definition at line [174](#) of file `PWM_CCU4.h`.

Referenced by `PWM_CCU4_Init()`.

### `const XMC_CCU4_SLICE_SHADOW_TRANSFER_MODE_t PWM_CCU4_ConfigType::sr_cnr`

Defines the timer value(s) at which shadow transfer trigger is generated

Definition at line [227](#) of file `PWM_CCU4.h`.

Referenced by `PWM_CCU4_Init()`.

### `const XMC_CCU4_SLICE_SR_ID_t PWM_CCU4_ConfigType::sr_crn`

Service request node to which channel 1 compare match while timer counting down event is forwarded

Definition at line [187](#) of file `PWM_CCU4.h`.

### `const XMC_CCU4_SLICE_SR_ID_t PWM_CCU4_ConfigType::sr_cru`

Service request node to which channel 1 compare match while timer counting up event is forwarded

Definition at line [186](#) of file `PWM_CCU4.h`.

### `const XMC_CCU4_SLICE_SR_ID_t PWM_CCU4_ConfigType::sr_e0`

Service request node to which event 0 is forwarded

Definition at line [189](#) of file `PWM_CCU4.h`.

### **const XMC\_CCU4\_SLICE\_SR\_ID\_t PWM\_CCU4\_ConfigType::sr\_e1**

Service request node to which event 1 is forwarded

Definition at line [190](#) of file [PWM\\_CCU4.h](#).

### **const XMC\_CCU4\_SLICE\_SR\_ID\_t PWM\_CCU4\_ConfigType::sr\_e2**

Service request node to which event 2 is forwarded

Definition at line [191](#) of file [PWM\\_CCU4.h](#).

### **const XMC\_CCU4\_SLICE\_SR\_ID\_t PWM\_CCU4\_ConfigType::sr\_on**

Service request node to which timer one match event is forwarded

Definition at line [188](#) of file [PWM\\_CCU4.h](#).

### **const XMC\_CCU4\_SLICE\_SR\_ID\_t PWM\_CCU4\_ConfigType::sr\_pe**

Service request node to which period match event is forwarded

Definition at line [185](#) of file [PWM\\_CCU4.h](#).

### **const bool PWM\_CCU4\_ConfigType::start\_control**

Enables starting of timer after initialization

Definition at line [173](#) of file [PWM\\_CCU4.h](#).

Referenced by [PWM\\_CCU4\\_Init\(\)](#).

The documentation for this struct was generated from the following file:

- PWM\_CCU4.h
- 



# PWM\_CCU4

[Home](#)

[Data Structures](#)

[Data Structure Index](#)

[Data Fields](#)

[Data Fields](#)

## PWM\_CCU4\_HandleTypeDef Struct Reference

## Detailed Description

Initialization parameters of the PWM\_CCU4 APP.

Definition at line **247** of file **PWM\_CCU4.h**.

```
#include <PWM_CCU4.h>
```

## Data Fields

```
const PWM_CCU4_CONFIG_t *const config_ptr  
  
XMC_CCU4_MODULE_t *const ccu4_module_ptr  
  
XMC_CCU4_SLICE_t *const ccu4_slice_ptr  
  
const uint8_t kernel_number  
  
const uint8_t slice_number  
  
const uint32_t shadow_txfr_msk  
  
const uint32_t dither_shadow_txfr_msk  
  
const uint32_t prescaler_shadow_txfr_msk  
  
PWM_CCU4_STATE_t state  
  
uint32_t frequency_tclk  
  
uint32_t sym_duty
```

## Field Documentation

**XMC\_CCU4\_MODULE\_t\* const PWM\_CCU4\_HandleTypeDef::ccu4\_mo**

Points to CCU4 global register base address

Definition at line [250](#) of file [PWM\\_CCU4.h](#).

Referenced by [PWM\\_CCU4\\_Init\(\)](#), [PWM\\_CCU4\\_SetDither\(\)](#),  
[PWM\\_CCU4\\_SetDutyCycle\(\)](#), [PWM\\_CCU4\\_SetFreq\(\)](#),  
[PWM\\_CCU4\\_SetFreqAndDutyCycle\(\)](#), and [PWM\\_CCU4\\_Start\(\)](#).

**XMC\_CCU4\_SLICE\_t\* const PWM\_CCU4\_HandleTypeDef::ccu4\_slice\_**

Points to CCU4 slice register base address

Definition at line [251](#) of file [PWM\\_CCU4.h](#).

Referenced by [PWM\\_CCU4\\_ClearEvent\(\)](#),  
[PWM\\_CCU4\\_ClearTrap\(\)](#), [PWM\\_CCU4\\_GetInterruptStatus\(\)](#),  
[PWM\\_CCU4\\_GetTimerStatus\(\)](#), [PWM\\_CCU4\\_GetTimerValue\(\)](#),  
[PWM\\_CCU4\\_Init\(\)](#), [PWM\\_CCU4\\_SetDither\(\)](#),  
[PWM\\_CCU4\\_SetDutyCycle\(\)](#), [PWM\\_CCU4\\_SetFreq\(\)](#),  
[PWM\\_CCU4\\_SetFreqAndDutyCycle\(\)](#), [PWM\\_CCU4\\_Start\(\)](#), and  
[PWM\\_CCU4\\_Stop\(\)](#).

**const PWM\_CCU4\_CONFIG\_t\* const PWM\_CCU4\_HandleTypeDef::cor**

Points to the variable containing PWM\_CCU4 APP configuration

Definition at line [249](#) of file [PWM\\_CCU4.h](#).

Referenced by [PWM\\_CCU4\\_Init\(\)](#), [PWM\\_CCU4\\_SetFreq\(\)](#),  
[PWM\\_CCU4\\_SetFreqAndDutyCycle\(\)](#), and [PWM\\_CCU4\\_Start\(\)](#).

### **const uint32\_t PWM\_CCU4\_HandleTypeDef::dither\_shadow\_txfr\_msk**

Mask for enabling shadow transfer of dither registers

Definition at line [255](#) of file [PWM\\_CCU4.h](#).

Referenced by [PWM\\_CCU4\\_Init\(\)](#), and [PWM\\_CCU4\\_SetDither\(\)](#).

### **uint32\_t PWM\_CCU4\_HandleTypeDef::frequency\_tclk**

Defines the operating frequency of the CCU4 slice

Definition at line [259](#) of file [PWM\\_CCU4.h](#).

Referenced by [PWM\\_CCU4\\_Init\(\)](#), [PWM\\_CCU4\\_SetFreq\(\)](#), and [PWM\\_CCU4\\_SetFreqAndDutyCycle\(\)](#).

### **const uint8\_t PWM\_CCU4\_HandleTypeDef::kernel\_number**

CCU4 Kernel number

Definition at line [252](#) of file [PWM\\_CCU4.h](#).

### **const uint32\_t PWM\_CCU4\_HandleTypeDef::prescaler\_shadow\_txfr\_n**

Mask for enabling shadow transfer of floating prescaler registers

Definition at line [256](#) of file [PWM\\_CCU4.h](#).

### **const uint32\_t PWM\_CCU4\_HandleTypeDef::shadow\_txfr\_msk**

Mask for enabling shadow transfer of period and compare registers

Definition at line [254](#) of file `PWM_CCU4.h`.

Referenced by `PWM_CCU4_Init()`, `PWM_CCU4_SetDutyCycle()`,  
`PWM_CCU4_SetFreq()`, and `PWM_CCU4_SetFreqAndDutyCycle()`.

### `const uint8_t PWM_CCU4_HandleType::slice_number`

CCU4 slice number

Definition at line [253](#) of file `PWM_CCU4.h`.

Referenced by `PWM_CCU4_Start()`.

### `PWM_CCU4_STATE_t PWM_CCU4_HandleType::state`

Defines the current state of the PWM\_CCU4 APP

Definition at line [258](#) of file `PWM_CCU4.h`.

Referenced by `PWM_CCU4_Init()`, `PWM_CCU4_SetDutyCycle()`,  
`PWM_CCU4_SetFreq()`, `PWM_CCU4_SetFreqAndDutyCycle()`,  
`PWM_CCU4_Start()`, and `PWM_CCU4_Stop()`.

### `uint32_t PWM_CCU4_HandleType::sym_duty`

Defines the channel 1 duty cycle in symmetric mode

Definition at line [260](#) of file `PWM_CCU4.h`.

Referenced by `PWM_CCU4_SetDutyCycle()`,  
`PWM_CCU4_SetFreq()`, and `PWM_CCU4_SetFreqAndDutyCycle()`.

---

The documentation for this struct was generated from the following file:

- PWM\_CCU4.h
- 



# PWM\_CCU4

Home

Data Structures

Data Structure Index

Data Fields

## Data Structure Index

P

P

PWM\_CCU4\_HandleType

PWM\_CCU4\_ConfigType

P



# PWM\_CCU4

a	c	d	e	f	g	i	k	m	p	s
---	---	---	---	---	---	---	---	---	---	---

Here is a list of all documented struct and union fields with links to the struct/union documentation for each field:

- a -

- automatic\_shadow\_transfer : [PWM\\_CCU4\\_ConfigType](#)

- c -

- cascaded\_shadow\_txfr\_enable : [PWM\\_CCU4\\_ConfigType](#)
- ccu4\_cc4\_slice\_timer\_ptr : [PWM\\_CCU4\\_ConfigType](#)
- ccu4\_module\_ptr : [PWM\\_CCU4\\_HandleTypeDef](#)
- ccu4\_slice\_ptr : [PWM\\_CCU4\\_HandleTypeDef](#)
- compare\_value : [PWM\\_CCU4\\_ConfigType](#)
- config\_ptr : [PWM\\_CCU4\\_HandleTypeDef](#)

- d -

- dither\_shadow\_txfr\_msk : [PWM\\_CCU4\\_HandleTypeDef](#)

- e -

- event0\_config\_ptr : [PWM\\_CCU4\\_ConfigType](#)
- event1\_config\_ptr : [PWM\\_CCU4\\_ConfigType](#)
- event2\_config\_ptr : [PWM\\_CCU4\\_ConfigType](#)
- ext\_count\_dir\_event : [PWM\\_CCU4\\_ConfigType](#)
- ext\_count\_event : [PWM\\_CCU4\\_ConfigType](#)
- ext\_gate\_event : [PWM\\_CCU4\\_ConfigType](#)
- ext\_load\_event : [PWM\\_CCU4\\_ConfigType](#)

- ext\_mod\_event : **PWM\_CCU4\_ConfigType**
- ext\_mod\_mode : **PWM\_CCU4\_ConfigType**
- ext\_mod\_sync : **PWM\_CCU4\_ConfigType**
- ext\_override\_edge\_event : **PWM\_CCU4\_ConfigType**
- ext\_override\_level\_event : **PWM\_CCU4\_ConfigType**
- ext\_start\_event : **PWM\_CCU4\_ConfigType**
- ext\_start\_mode : **PWM\_CCU4\_ConfigType**
- ext\_stop\_event : **PWM\_CCU4\_ConfigType**
- ext\_stop\_mode : **PWM\_CCU4\_ConfigType**
- ext\_trap\_enable : **PWM\_CCU4\_ConfigType**
- ext\_trap\_event : **PWM\_CCU4\_ConfigType**
- ext\_trap\_exit : **PWM\_CCU4\_ConfigType**
- ext\_trap\_sync : **PWM\_CCU4\_ConfigType**

- f -

- frequency\_tclk : **PWM\_CCU4\_HandleTypeDef**

- g -

- global\_ccu4\_handle : **PWM\_CCU4\_ConfigType**
- gpio\_ch\_out\_config\_ptr : **PWM\_CCU4\_ConfigType**
- gpio\_ch\_out\_enable : **PWM\_CCU4\_ConfigType**
- gpio\_ch\_out\_pin : **PWM\_CCU4\_ConfigType**
- gpio\_ch\_out\_ptr : **PWM\_CCU4\_ConfigType**

- i -

- immediate\_write : **PWM\_CCU4\_ConfigType**
- int\_cmp\_match\_down : **PWM\_CCU4\_ConfigType**
- int\_cmp\_match\_up : **PWM\_CCU4\_ConfigType**
- int\_e0 : **PWM\_CCU4\_ConfigType**
- int\_e1 : **PWM\_CCU4\_ConfigType**
- int\_e2 : **PWM\_CCU4\_ConfigType**
- int\_one\_match\_down : **PWM\_CCU4\_ConfigType**
- int\_per\_match : **PWM\_CCU4\_ConfigType**

- k -

- kernel\_number : **PWM\_CCU4\_HandleTypeDef**

- m -

- mcm\_shadow\_txfr\_mode : **PWM\_CCU4\_ConfigType**

- p -

- period\_value : **PWM\_CCU4\_ConfigType**
- prescaler\_shadow\_txfr\_msk : **PWM\_CCU4\_HandleTypeDef**

- s -

- shadow\_transfer\_mode : **PWM\_CCU4\_ConfigType**
  - shadow\_txfr\_msk : **PWM\_CCU4\_HandleTypeDef**
  - slice\_number : **PWM\_CCU4\_HandleTypeDef**
  - sr\_cmp\_match\_down : **PWM\_CCU4\_ConfigType**
  - sr\_cmp\_match\_up : **PWM\_CCU4\_ConfigType**
  - sr\_e0 : **PWM\_CCU4\_ConfigType**
  - sr\_e1 : **PWM\_CCU4\_ConfigType**
  - sr\_e2 : **PWM\_CCU4\_ConfigType**
  - sr\_one\_match\_down : **PWM\_CCU4\_ConfigType**
  - sr\_per\_match : **PWM\_CCU4\_ConfigType**
  - start\_control : **PWM\_CCU4\_ConfigType**
  - state : **PWM\_CCU4\_HandleTypeDef**
  - sym\_duty : **PWM\_CCU4\_HandleTypeDef**
- 



# PWM\_CCU4

Data Fields										
All		Variables								
a	c	d	e	f	g	i	k	m	p	s

- a -

- automatic\_shadow\_transfer : **PWM\_CCU4\_ConfigType**

- c -

- cascaded\_shadow\_txfr\_enable : **PWM\_CCU4\_ConfigType**
- ccu4\_cc4\_slice\_timer\_ptr : **PWM\_CCU4\_ConfigType**
- ccu4\_module\_ptr : **PWM\_CCU4\_HandleTypeDef**
- ccu4\_slice\_ptr : **PWM\_CCU4\_HandleTypeDef**
- compare\_value : **PWM\_CCU4\_ConfigType**
- config\_ptr : **PWM\_CCU4\_HandleTypeDef**

- d -

- dither\_shadow\_txfr\_msk : **PWM\_CCU4\_HandleTypeDef**

- e -

- event0\_config\_ptr : **PWM\_CCU4\_ConfigType**
- event1\_config\_ptr : **PWM\_CCU4\_ConfigType**
- event2\_config\_ptr : **PWM\_CCU4\_ConfigType**
- ext\_count\_dir\_event : **PWM\_CCU4\_ConfigType**
- ext\_count\_event : **PWM\_CCU4\_ConfigType**
- ext\_gate\_event : **PWM\_CCU4\_ConfigType**
- ext\_load\_event : **PWM\_CCU4\_ConfigType**
- ext\_mod\_event : **PWM\_CCU4\_ConfigType**

- ext\_mod\_mode : **PWM\_CCU4\_ConfigType**
- ext\_mod\_sync : **PWM\_CCU4\_ConfigType**
- ext\_override\_edge\_event : **PWM\_CCU4\_ConfigType**
- ext\_override\_level\_event : **PWM\_CCU4\_ConfigType**
- ext\_start\_event : **PWM\_CCU4\_ConfigType**
- ext\_start\_mode : **PWM\_CCU4\_ConfigType**
- ext\_stop\_event : **PWM\_CCU4\_ConfigType**
- ext\_stop\_mode : **PWM\_CCU4\_ConfigType**
- ext\_trap\_enable : **PWM\_CCU4\_ConfigType**
- ext\_trap\_event : **PWM\_CCU4\_ConfigType**
- ext\_trap\_exit : **PWM\_CCU4\_ConfigType**
- ext\_trap\_sync : **PWM\_CCU4\_ConfigType**

- f -

- frequency\_tclk : **PWM\_CCU4\_HandleTypeDef**

- g -

- global\_ccu4\_handle : **PWM\_CCU4\_ConfigType**
- gpio\_ch\_out\_config\_ptr : **PWM\_CCU4\_ConfigType**
- gpio\_ch\_out\_enable : **PWM\_CCU4\_ConfigType**
- gpio\_ch\_out\_pin : **PWM\_CCU4\_ConfigType**
- gpio\_ch\_out\_ptr : **PWM\_CCU4\_ConfigType**

- i -

- immediate\_write : **PWM\_CCU4\_ConfigType**
- int\_cmp\_match\_down : **PWM\_CCU4\_ConfigType**
- int\_cmp\_match\_up : **PWM\_CCU4\_ConfigType**
- int\_e0 : **PWM\_CCU4\_ConfigType**
- int\_e1 : **PWM\_CCU4\_ConfigType**
- int\_e2 : **PWM\_CCU4\_ConfigType**
- int\_one\_match\_down : **PWM\_CCU4\_ConfigType**
- int\_per\_match : **PWM\_CCU4\_ConfigType**

- k -

- kernel\_number : **PWM\_CCU4\_HandleTypeDef**

- m -

- mcm\_shadow\_txfr\_mode : **PWM\_CCU4\_ConfigType**

- p -

- period\_value : **PWM\_CCU4\_ConfigType**
- prescaler\_shadow\_txfr\_msk : **PWM\_CCU4\_HandleTypeDef**

- s -

- shadow\_transfer\_mode : **PWM\_CCU4\_ConfigType**
  - shadow\_txfr\_msk : **PWM\_CCU4\_HandleTypeDef**
  - slice\_number : **PWM\_CCU4\_HandleTypeDef**
  - sr\_cmp\_match\_down : **PWM\_CCU4\_ConfigType**
  - sr\_cmp\_match\_up : **PWM\_CCU4\_ConfigType**
  - sr\_e0 : **PWM\_CCU4\_ConfigType**
  - sr\_e1 : **PWM\_CCU4\_ConfigType**
  - sr\_e2 : **PWM\_CCU4\_ConfigType**
  - sr\_one\_match\_down : **PWM\_CCU4\_ConfigType**
  - sr\_per\_match : **PWM\_CCU4\_ConfigType**
  - start\_control : **PWM\_CCU4\_ConfigType**
  - state : **PWM\_CCU4\_HandleTypeDef**
  - sym\_duty : **PWM\_CCU4\_HandleTypeDef**
- 



# PWM\_CCU4

Home

File List

Globals

## File List

Here is a list of all documented files with brief descriptions:

 [PWM\\_CCU4.c](#)

 [PWM\\_CCU4.h](#)

---

# PWM\_CCU4

<a href="#">Home</a>	
<a href="#">File List</a>	<a href="#">Globals</a>
<a href="#">Code</a>	<a href="#">Functions</a>

**PWM\_CCU4.c File Reference**

## Detailed Description

### Date

2016-10-28

NOTE: This file is generated by DAVE. Any manual modification done to this file will be lost when the code is regenerated.

Definition in file [PWM\\_CCU4.c](#).

```
#include "pwm_ccu4.h"
```

## Functions

DAVE\_APP\_VERSION\_t **PWM\_CCU4\_GetAppVersion** (void)  
Retrieves the version of the PWM\_CCU4 APP. [More...](#)

PWM\_CCU4\_STATUS\_t **PWM\_CCU4\_Init** (PWM\_CCU4\_t \*handle\_ptr)  
Initializes the PWM\_CCU4 APP. [More...](#)

PWM\_CCU4\_STATUS\_t **PWM\_CCU4\_Start** (PWM\_CCU4\_t \*handle\_ptr)  
Start the selected CCU4 slice. [More...](#)

PWM\_CCU4\_STATUS\_t **PWM\_CCU4\_Stop** (PWM\_CCU4\_t \*handle\_ptr)  
Stop the selected CCU4 slice. [More...](#)

uint32\_t **PWM\_CCU4\_GetTimerValue** (PWM\_CCU4\_t \*handle\_ptr)  
Returns the timer value. [More...](#)

bool **PWM\_CCU4\_GetTimerStatus** (PWM\_CCU4\_t \*handle\_ptr)  
Returns the timer status. [More...](#)

PWM\_CCU4\_STATUS\_t **PWM\_CCU4\_SetFreq** (PWM\_CCU4\_t \*handle\_ptr, uint32\_t pwm\_freq\_hz)  
Sets the PWM frequency. [More...](#)

PWM\_CCU4\_STATUS\_t **PWM\_CCU4\_SetDutyCycle** (PWM\_CCU4\_t \*handle\_ptr, uint32\_t duty\_cycle)

Sets the duty cycle of PWM. More...

**PWM\_CCU4\_SetFreqAndDutyCycle**

**PWM\_CCU4\_STATUS\_t** (**PWM\_CCU4\_t** \*handle\_ptr, uint32\_t pwm\_freq\_hz, uint32\_t duty)

Sets the frequency duty cycle of PWM.

More...

**PWM\_CCU4\_SetDither (PWM\_CCU4\_t**

void \*handle\_ptr, bool dither\_period, bool dither\_comp, uint8\_t dither\_value)

Sets the dither value for period , duty or both. More...

**PWM\_CCU4\_ClearTrap (PWM\_CCU4\_t**

void \*handle\_ptr)

Clears the trap event. More...

**PWM\_CCU4\_GetInterruptStatus**

bool (**PWM\_CCU4\_t** \*handle\_ptr, XMC\_CCU4\_SLICE\_IRQ\_ID\_t pwm\_interrupt)

Returns the interrupt status. More...

**PWM\_CCU4\_ClearEvent (PWM\_CCU4\_t**

void \*handle\_ptr, XMC\_CCU4\_SLICE\_IRQ\_ID\_t pwm\_interrupt)

Acknowledges the interrupt. More...

## Function Documentation

```
void PWM_CCU4_ClearEvent( PWM_CCU4_t *const handle,
                           XMC_CCU4_SLICE_IRQ_ID_t pwm_
                           )
```

Acknowledges the interrupt.

### Parameters

**handle\_ptr** Pointer to PWM\_CCU4\_t structure containing APP parameters.  
**pwm\_interrupt** interrupt ID

### Returns

#### Description:

Clears the interrupt status flag, provided the interrupt condition no longer exists.

Example Usage:

```
#include <DAVE.h>
int main(void)
{
    DAVE_Init();
    PWM_CCU4_ClearEvent(&PWM_CCU4_0,XMC_CCU4_SLICE_IRQ_ID);
    PWM_CCU4_ClearEvent(&PWM_CCU4_0,XMC_CCU4_SLICE_IRQ_ID);
    while(1);
    return 0;
}
```

Definition at line 586 of file [PWM\\_CCU4.c](#).

References [PWM\\_CCU4\\_HandleType::ccu4\\_slice\\_ptr](#).

**void PWM\_CCU4\_ClearTrap ( PWM\_CCU4\_t \*const handle\_ptr )**

Clears the trap event.

**Parameters**

**handle\_ptr** Pointer to PWM\_CCU4\_t structure containing APP parameters.

**Returns**

void

**Description:**

Clears the trap event provided the trap condition no longer exists.

Example Usage:

```
#include <DAVE.h>
int main(void)
{
    DAVE_Init();
    PWM_CCU4_ClearTrap(&PWM_CCU4_0);
    while(1);
    return 0;
}
```

Definition at line [563](#) of file [PWM\\_CCU4.c](#).

References [PWM\\_CCU4\\_HandleTypeDef::ccu4\\_slice\\_ptr](#).

**bool PWM\_CCU4\_GetInterruptStatus ( PWM\_CCU4\_t \*const XMC\_CCU4\_SLICE\_IRQ\_ID\_ )**

Returns the interrupt status.

## Parameters

**handle\_ptr** Pointer to PWM\_CCU4\_t structure containing APP parameters.  
**pwm\_interrupt** interrupt ID

## Returns

bool

## Description:

Returns true if the interrupt flag is set, else returns false.

Example Usage:

```
#include <DAVE.h>
int main(void)
{
    bool status;
    DAVE_Init();
    // Returns period match interrupt status.
    status =
        PWM_CCU4_GetInterruptStatus(&PWM_CCU4_0,XMC_CCU4_SLICE_
    // Returns channel compare match interrupt status.
    status =
        PWM_CCU4_GetInterruptStatus(&PWM_CCU4_0,XMC_CCU4_SLICE_
    while(1);
    return 0;
}
```

Definition at line 575 of file [PWM\\_CCU4.c](#).

References [PWM\\_CCU4\\_HandleTypeDef::ccu4\\_slice\\_ptr](#).

---

**bool PWM\_CCU4\_GetTimerStatus ( PWM\_CCU4\_t \*const handle\_p**

Returns the timer status.

## Parameters

**handle\_ptr** Pointer to PWM\_CCU4\_t structure containing APP parameters.

**Returns**

XMC\_CCU4\_STATUS\_t

**Description:**

Returns true if the timer is running else returns false.

Example Usage:

```
#include <DAVE.h>
int main(void)
{
    bool status;
    DAVE_Init();
    status = PWM_CCU4_GetTimerStatus(&PWM_CCU4_0);
    while(1);
    return 0;
}
```

Definition at line 393 of file [PWM\\_CCU4.c](#).

References [PWM\\_CCU4\\_HandleTypeDef::ccu4\\_slice\\_ptr](#).

---

**uint32\_t PWM\_CCU4\_GetTimerValue ( PWM\_CCU4\_t \*const handle**

Returns the timer value.

**Parameters**

**handle\_ptr** Pointer to PWM\_CCU4\_t structure containing APP parameters.

**Returns**

uint32\_t

**Description:**

Returns the timer value if the APP is initialized.

Example Usage:

```
#include <DAVE.h>
int main(void)
{
    uint32_t timer ;
    DAVE_Init();
    timer = PWM_CCU4_GetTimerValue(&PWM_CCU4_0);
    while(1);
    return 0;
}
```

Definition at line 383 of file [PWM\\_CCU4.c](#).

References [PWM\\_CCU4\\_HandleTypeDef::ccu4\\_slice\\_ptr](#).

[PWM\\_CCU4\\_STATUS\\_t PWM\\_CCU4\\_Init \( PWM\\_CCU4\\_t \\*const handle\\_ptr \)](#)

Initializes the PWM\_CCU4 APP.

#### Parameters

**handle\_ptr** Pointer to PWM\_CCU4\_t structure containing APP parameters.

#### Returns

PWM\_CCU4\_STATUS\_t

#### Description:

Configures the CCU4 slice registers with the selected PWM\_CCU4 parameters. The slice is configured in PWM generation mode.

Example Usage:

```
#include <DAVE.h>
```

```
int main(void)
{
    DAVE_Init(); //PWM_CCU4_Init() is called by DAVE_Init().
    while(1);
    return 0;
}
```

Definition at line 105 of file [PWM\\_CCU4.c](#).

## References

[PWM\\_CCU4\\_ConfigType::automatic\\_shadow\\_transfer](#),  
[PWM\\_CCU4\\_ConfigType::cascaded\\_shadow\\_txfr\\_enable](#),  
[PWM\\_CCU4\\_ConfigType::ccu4\\_cc4\\_slice\\_timer\\_ptr](#),  
[PWM\\_CCU4\\_HandleTypeDef::ccu4\\_module\\_ptr](#),  
[PWM\\_CCU4\\_HandleTypeDef::ccu4\\_slice\\_ptr](#),  
[PWM\\_CCU4\\_ConfigType::compare\\_value](#),  
[PWM\\_CCU4\\_HandleTypeDef::config\\_ptr](#),  
[PWM\\_CCU4\\_HandleTypeDef::dither\\_shadow\\_txfr\\_msk](#),  
[PWM\\_CCU4\\_HandleTypeDef::frequency\\_tclk](#),  
[PWM\\_CCU4\\_ConfigType::global\\_ccu4\\_handle](#),  
[PWM\\_CCU4\\_ConfigType::gpio\\_ch\\_out\\_config\\_ptr](#),  
[PWM\\_CCU4\\_ConfigType::gpio\\_ch\\_out\\_enable](#),  
[PWM\\_CCU4\\_ConfigType::gpio\\_ch\\_out\\_pin](#),  
[PWM\\_CCU4\\_ConfigType::gpio\\_ch\\_out\\_ptr](#),  
[PWM\\_CCU4\\_ConfigType::immediate\\_write](#),  
[PWM\\_CCU4\\_ConfigType::mcm\\_shadow\\_txfr\\_mode](#),  
[PWM\\_CCU4\\_ConfigType::period\\_value](#), [PWM\\_CCU4\\_Start\(\)](#),  
[PWM\\_CCU4\\_STATE\\_INITIALIZED](#),  
[PWM\\_CCU4\\_STATE\\_UNINITIALIZED](#),  
[PWM\\_CCU4\\_STATUS\\_ALREADY\\_INITIALIZED](#),  
[PWM\\_CCU4\\_STATUS\\_FAILURE](#),  
[PWM\\_CCU4\\_STATUS\\_SUCCESS](#),  
[PWM\\_CCU4\\_ConfigType::shadow\\_transfer\\_mode](#),  
[PWM\\_CCU4\\_HandleTypeDef::shadow\\_txfr\\_msk](#),  
[PWM\\_CCU4\\_ConfigType::start\\_control](#), and  
[PWM\\_CCU4\\_HandleTypeDef::state](#).

```
void PWM_CCU4_SetDither ( PWM_CCU4_t *const handle_ptr,  
                           bool          dither_period,  
                           bool          dither_comp,  
                           uint8_t       dither_value  
                         )
```

Sets the dither value for period , duty or both.

#### Parameters

**handle\_ptr** Pointer to PWM\_CCU4\_t structure containing APP parameters.  
**dither\_period** apply dither to period  
**dither\_comp** apply dither to compare  
**dither\_value** dither value

#### Returns

void

#### Description:

Sets the dither value for period , duty or both.

dither\_value: is the dither value.

dither\_period: when true, dither is applied to period.

dither\_comp: when true, dither is applied to compare.

Example Usage:

```
#include <DAVE.h>  
int main(void)  
{  
    DAVE_Init();  
    PWM_CCU4_SetDither(&PWM_CCU4_0,(bool) true, (bool)true, 10);  
    while(1);  
    return 0;  
}
```

Definition at line **550** of file [PWM\\_CCU4.c](#).

References `PWM_CCU4_HandleType::ccu4_module_ptr`,  
`PWM_CCU4_HandleType::ccu4_slice_ptr`, and  
`PWM_CCU4_HandleType::dither_shadow_txfr_msk`.

```
PWM_CCU4_STATUS_t PWM_CCU4_SetDutyCycle ( PWM_CCU4_t
                                            uint32_t
                                            )
```

Sets the duty cycle of PWM.

#### Parameters

**handle\_ptr** Pointer to `PWM_CCU4_t` structure containing APP parameters.  
**duty\_cycle** channel duty cycle `uint32_t`

#### Returns

`PWM_CCU4_STATUS_t`

#### Description:

Sets the PWM duty. The APP should not be in "PWM\_CCU4\_UNINITIALIZED" state. Duty is scaled by 100. The condition [duty < 100%] should be met. Returns `PWM_CCU4_STATUS_SUCCESS` if operation update is success.

Example Usage:

```
#include <DAVE.h>
int main(void)
{
    PWM_CCU4_STATUS_t status;
    DAVE_Init();
    // sets the channel duty to 40%.
    status = PWM_CCU4_SetDutyCycle(&PWM_CCU4_0, 4000);
    while(1);
    return 0;
```

```
}
```

Definition at line [458](#) of file [PWM\\_CCU4.c](#).

References [PWM\\_CCU4\\_HandleTypeDef::ccu4\\_module\\_ptr](#),  
[PWM\\_CCU4\\_HandleTypeDef::ccu4\\_slice\\_ptr](#),  
[PWM\\_CCU4\\_STATE\\_UNINITIALIZED](#),  
[PWM\\_CCU4\\_STATUS\\_FAILURE](#),  
[PWM\\_CCU4\\_STATUS\\_SUCCESS](#),  
[PWM\\_CCU4\\_HandleTypeDef::shadow\\_txfr\\_msk](#),  
[PWM\\_CCU4\\_HandleTypeDef::state](#), and  
[PWM\\_CCU4\\_HandleTypeDef::sym\\_duty](#).

```
PWM_CCU4_STATUS_t PWM_CCU4_SetFreq ( PWM_CCU4_t *cons
                                         uint32_t
                                         )
```

Sets the PWM frequency.

#### Parameters

**handle\_ptr** Pointer to PWM\_CCU4\_t structure containing APP parameters.  
**pwm\_freq\_hz** value in Hz (uint32\_t)

#### Returns

PWM\_CCU4\_STATUS\_t

#### Description:

Sets the PWM frequency of PWM generation. The APP should not be in "PWM\_CCU4\_UNINITIALIZED" state. Returns PWM\_CCU4\_STATUS\_SUCCESS if frequency update is success.

Example Usage:

```
#include <DAVE.h>
```

```
int main(void)
{
    PWM_CCU4_STATUS_t status;
    DAVE_Init();
    status = PWM_CCU4_SetFreq(&PWM_CCU4_0, 100000);
    while(1);
    return 0;
}
```

Definition at line [405](#) of file [PWM\\_CCU4.c](#).

References [PWM\\_CCU4\\_ConfigType::ccu4\\_cc4\\_slice\\_timer\\_ptr](#),  
[PWM\\_CCU4\\_HandleTypeDef::ccu4\\_module\\_ptr](#),  
[PWM\\_CCU4\\_HandleTypeDef::ccu4\\_slice\\_ptr](#),  
[PWM\\_CCU4\\_HandleTypeDef::config\\_ptr](#),  
[PWM\\_CCU4\\_HandleTypeDef::frequency\\_tclk](#),  
[PWM\\_CCU4\\_STATE\\_UNINITIALIZED](#),  
[PWM\\_CCU4\\_STATUS\\_FAILURE](#),  
[PWM\\_CCU4\\_STATUS\\_SUCCESS](#),  
[PWM\\_CCU4\\_HandleTypeDef::shadow\\_txfr\\_msk](#),  
[PWM\\_CCU4\\_HandleTypeDef::state](#), and  
[PWM\\_CCU4\\_HandleTypeDef::sym\\_duty](#).

**PWM\_CCU4\_STATUS\_t PWM\_CCU4\_SetFreqAndDutyCycle ( [PWM\\_HandleTypeDef](#) handle, [uint32\\_t](#) pwm\_freq\_hz, [uint32\\_t](#) duty )**

Sets the frequency duty cycle of PWM.

#### Parameters

- |                    |  |
|--------------------|--|
| <b>handle_ptr</b>  | Pointer to PWM_CCU4_t structure containing APP parameters. |
| <b>pwm_freq_hz</b> | value in Hz ( <a href="#">uint32_t</a> )                   |
| <b>duty</b>        | channel duty   |

## Returns

PWM\_CCU4\_STATUS\_t

## Description:

Sets the frequency and duty of PWM. The APP should not be in "PWM\_CCU4\_UNINITIALIZED" state. Duty is scaled by 100.

The condition [duty < 100%] should be met.

Returns PWM\_CCU4\_STATUS\_SUCCESS if operation update is success.

Example Usage:

```
#include <DAVE.h>
int main(void)
{
    PWM_CCU4_STATUS_t status;
    DAVE_Init();
    // Sets freq = 100000, channel duty = 40%
    status = PWM_CCU4_SetFreqAndDutyCycle(&PWM_CCU4_0,
    100000, 4000);
    while(1);
    return 0;
}
```

Definition at line 495 of file [PWM\\_CCU4.c](#).

References [PWM\\_CCU4\\_ConfigType::ccu4\\_cc4\\_slice\\_timer\\_ptr](#),  
[PWM\\_CCU4\\_HandleType::ccu4\\_module\\_ptr](#),  
[PWM\\_CCU4\\_HandleType::ccu4\\_slice\\_ptr](#),  
[PWM\\_CCU4\\_HandleType::config\\_ptr](#),  
[PWM\\_CCU4\\_HandleType::frequency\\_tclk](#),  
[PWM\\_CCU4\\_STATE\\_UNINITIALIZED](#),  
[PWM\\_CCU4\\_STATUS\\_FAILURE](#),  
[PWM\\_CCU4\\_STATUS\\_SUCCESS](#),  
[PWM\\_CCU4\\_HandleType::shadow\\_txfr\\_msk](#),  
[PWM\\_CCU4\\_HandleType::state](#), and  
[PWM\\_CCU4\\_HandleType::sym\\_duty](#).

## **PWM\_CCU4\_STATUS\_t PWM\_CCU4\_Start( PWM\_CCU4\_t \*const h**

Start the selected CCU4 slice.

### **Parameters**

**handle\_ptr** Pointer to PWM\_CCU4\_t structure containing APP parameters.

### **Returns**

PWM\_CCU4\_STATUS\_t

### **Description:**

Starts the selected CCU4 slice for PWM generation. Returns PWM\_CCU4\_STATUS\_SUCCESS if the PWM\_CCU4 APP state is in "PWM\_CCU4\_STATE\_INITIALIZED" or "PWM\_CCU4\_STOPPED" else returns PWM\_CCU4\_STATUS\_FAILURE.

**PWM\_CCU4\_Start()** is needed to be called if "Start during initialization" is unchecked to start PWM generation, else its called by DAVE\_Init();

Example Usage:

```
#include <DAVE.h>
int main(void)
{
    DAVE_Init();
    //This API needs to be called if "Start during initialization" is
    //unchecked
    PWM_CCU4_Start(&PWM_CCU4_0);
    while(1);
    return 0;
}
```

Definition at line **339** of file **PWM\_CCU4.c**.

References **PWM\_CCU4\_HandleType::ccu4\_module\_ptr**,

`PWM_CCU4_HandleType::ccu4_slice_ptr`,  
`PWM_CCU4_HandleType::config_ptr`,  
`PWM_CCU4_ConfigType::ext_start_event`,  
`PWM_CCU4_STATE_INITIALIZED`,  
`PWM_CCU4_STATE_RUNNING`, `PWM_CCU4_STATE_STOPPED`,  
`PWM_CCU4_STATUS_FAILURE`,  
`PWM_CCU4_STATUS_SUCCESS`,  
`PWM_CCU4_HandleType::slice_number`, and  
`PWM_CCU4_HandleType::state`.

Referenced by `PWM_CCU4_Init()`.

## `PWM_CCU4_STATUS_t PWM_CCU4_Stop ( PWM_CCU4_t *const h`

Stop the selected CCU4 slice.

### Parameters

`handle_ptr` Pointer to `PWM_CCU4_t` structure containing APP parameters.

### Returns

`PWM_CCU4_STATUS_t`

### Description:

Stops the selected CCU4 slice from PWM generation. Returns `PWM_CCU4_STATUS_SUCCESS` if the `PWM_CCU4` APP state is not "`PWM_CCU4_STATE_UNINITIALIZED`" else returns `PWM_CCU4_STATUS_FAILURE`.

Example Usage:

```
#include <DAVE.h>
int main(void)
{
    DAVE_Init();
    PWM_CCU4_Stop(&PWM_CCU4_0);
    while(1);
```

```
return 0;  
}
```

Definition at line 363 of file [PWM\\_CCU4.c](#).

References [PWM\\_CCU4\\_HandleTypeDef::ccu4\\_slice\\_ptr](#),  
[PWM\\_CCU4\\_STATE\\_STOPPED](#),  
[PWM\\_CCU4\\_STATE\\_UNINITIALIZED](#),  
[PWM\\_CCU4\\_STATUS\\_FAILURE](#),  
[PWM\\_CCU4\\_STATUS\\_SUCCESS](#), and  
[PWM\\_CCU4\\_HandleTypeDef::state](#).

---

Go to the source code of this file.

---

# PWM\_CCU4

<a href="#">Home</a>	
<a href="#">File List</a>	<a href="#">Globals</a>
<a href="#">Code</a>	

[Data Structures](#)

## PWM\_CCU4.h File Reference

## Detailed Description

### Date

2016-03-21

NOTE: This file is generated by DAVE. Any manual modification done to this file will be lost when the code is regenerated.

Definition in file [PWM\\_CCU4.h](#).

```
#include <xmc_gpio.h> #include "pwm_ccu4_conf.h"  
#include "pwm_ccu4_extern.h"
```

## Data Structures

---

struct **PWM\_CCU4\_ConfigType**

Configuration parameters of the PWM\_CCU4 APP. [More...](#)

---

struct **PWM\_CCU4\_HandleTypeDef**

Initialization parameters of the PWM\_CCU4 APP. [More...](#)

## TypeDefs

typedef struct **PWM\_CCU4\_ConfigType** **PWM\_CCU4\_CONFIG\_t**

Configuration parameters  
of the PWM\_CCU4 APP.

---

typedef struct **PWM\_CCU4\_HandleTypeDef** **PWM\_CCU4\_t**

Initialization parameters  
of the PWM\_CCU4 APP.

## Functions

DAVE_APP_VERSION_t	<b>PWM_CCU4_GetAppVersion</b> (void *const handle_ptr)	Retrieves the version of the PWM APP. More...
<b>PWM_CCU4_STATUS_t</b>	<b>PWM_CCU4_Init</b> ( <b>PWM_CCU4_HANDLE_PTR</b> handle_ptr)	Initializes the PWM_CCU4 APP.
<b>PWM_CCU4_STATUS_t</b>	<b>PWM_CCU4_Start</b> ( <b>PWM_CCU4_HANDLE_PTR</b> handle_ptr)	Start the selected CCU4 slice. More...
<b>PWM_CCU4_STATUS_t</b>	<b>PWM_CCU4_Stop</b> ( <b>PWM_CCU4_HANDLE_PTR</b> handle_ptr)	Stop the selected CCU4 slice. More...
uint32_t	<b>PWM_CCU4_GetTimerValue</b> ( <b>PWM_CCU4_HANDLE_PTR</b> *const handle_ptr)	Returns the timer value. More...
bool	<b>PWM_CCU4_GetTimerStatus</b> ( <b>PWM_CCU4_HANDLE_PTR</b> *const handle_ptr)	Returns the timer status. More...
<b>PWM_CCU4_STATUS_t</b>	<b>PWM_CCU4_SetFreq</b> ( <b>PWM_CCU4_HANDLE_PTR</b> handle_ptr, uint32_t pwm_freq_hz)	Sets the PWM frequency. More...
<b>PWM_CCU4_STATUS_t</b>	<b>PWM_CCU4_SetDutyCycle</b> ( <b>PWM_CCU4_HANDLE_PTR</b> *const handle_ptr, uint32_t duty_cycle)	Sets the duty cycle of PWM. More...

**PWM\_CCU4\_STATUS\_t** **PWM\_CCU4\_SetFreqAndDutyCycle** (PWM\_CCU4\_t \*const handle\_ptr, uint32\_t pwm\_freq\_hz, uint32\_t duty)  
Sets the frequency duty cycle of the PWM slice.

void **PWM\_CCU4\_SetDither** (PWM\_CCU4\_t \*const handle\_ptr, bool dither\_period, bool dither\_polarity, uint8\_t dither\_value)  
Sets the dither value for period , polarity and value.

void **PWM\_CCU4\_ClearTrap** (PWM\_CCU4\_t \*const handle\_ptr)  
Clears the trap event. More...

bool **PWM\_CCU4\_GetInterruptStatus** (PWM\_CCU4\_t \*const handle\_ptr, XMC\_CCU4\_SLICE\_IRQ\_ID\_t p)  
Returns the interrupt status. More...

void **PWM\_CCU4\_ClearEvent** (PWM\_CCU4\_t \*const handle\_ptr, XMC\_CCU4\_SLICE\_INTERRUPT pwm\_interrupt)  
Acknowledges the interrupt. More...

enum **PWM\_CCU4\_STATUS** {  
 PWM\_CCU4\_STATUS\_SUCCESS = 0,  
 PWM\_CCU4\_STATUS\_FAILURE,  
 PWM\_CCU4\_STATUS\_ALREADY\_SET,  
 PWM\_CCU4\_STATUS\_INVALID\_HANDLE  
} = 2U }  
The type identifies the APP status.

**PWM\_CCU4\_STATE** {  
 PWM\_CCU4\_STATE\_UNINITIALI

enum **PWM\_CCU4\_STATE\_INITIALIZI**  
**PWM\_CCU4\_STATE\_RUNNING**  
**PWM\_CCU4\_STATE\_STOPPED**  
The type identifies APP state. Mc

**typedef enum PWM\_CCU4\_STATUS PWM\_CCU4\_STATUS\_t**  
The type identifies the APP status.

**typedef enum PWM\_CCU4\_STATE PWM\_CCU4\_STATE\_t**  
The type identifies APP state.

Go to the source code of this file.

---

# PWM\_CCU4

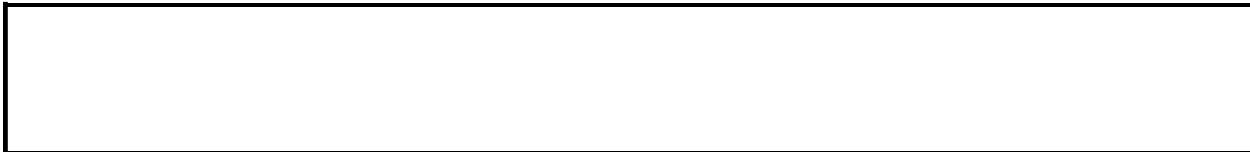
The screenshot shows a software interface for documenting code. At the top is a navigation bar with tabs: 'Home', 'File List', and 'Globals'. The 'Globals' tab is active and has its own sub-navigation with tabs: 'All', 'Functions', 'Typedefs', 'Enumerations', and 'Enumerator'. Below the navigation bar is a search bar containing the letter 'p'. The main area of the interface is currently empty, indicating no results for the search.

Here is a list of all documented functions, variables, defines, enums, and typedefs with links to the documentation:

- p -

- PWM\_CCU4\_ClearEvent() : [PWM\\_CCU4.c](#) , [PWM\\_CCU4.h](#)
- PWM\_CCU4\_ClearTrap() : [PWM\\_CCU4.h](#) , [PWM\\_CCU4.c](#)
- PWM\_CCU4\_CONFIG\_t : [PWM\\_CCU4.h](#)
- PWM\_CCU4\_GetAppVersion() : [PWM\\_CCU4.c](#) , [PWM\\_CCU4.h](#)
- PWM\_CCU4\_GetInterruptStatus() : [PWM\\_CCU4.c](#) , [PWM\\_CCU4.h](#)
- PWM\_CCU4\_GetTimerStatus() : [PWM\\_CCU4.c](#) , [PWM\\_CCU4.h](#)
- PWM\_CCU4\_GetTimerValue() : [PWM\\_CCU4.c](#) , [PWM\\_CCU4.h](#)
- PWM\_CCU4\_Init() : [PWM\\_CCU4.c](#) , [PWM\\_CCU4.h](#)
- PWM\_CCU4\_SetDither() : [PWM\\_CCU4.c](#) , [PWM\\_CCU4.h](#)
- PWM\_CCU4\_SetDutyCycle() : [PWM\\_CCU4.c](#) , [PWM\\_CCU4.h](#)
- PWM\_CCU4\_SetFreq() : [PWM\\_CCU4.c](#) , [PWM\\_CCU4.h](#)
- PWM\_CCU4\_SetFreqAndDutyCycle() : [PWM\\_CCU4.h](#) , [PWM\\_CCU4.c](#)
- PWM\_CCU4\_Start() : [PWM\\_CCU4.c](#) , [PWM\\_CCU4.h](#)
- PWM\_CCU4\_STATE : [PWM\\_CCU4.h](#)
- PWM\_CCU4\_STATE\_INITIALIZED : [PWM\\_CCU4.h](#)
- PWM\_CCU4\_STATE\_RUNNING : [PWM\\_CCU4.h](#)
- PWM\_CCU4\_STATE\_STOPPED : [PWM\\_CCU4.h](#)
- PWM\_CCU4\_STATE\_t : [PWM\\_CCU4.h](#)
- PWM\_CCU4\_STATE\_UNINITIALIZED : [PWM\\_CCU4.h](#)
- PWM\_CCU4\_STATUS : [PWM\\_CCU4.h](#)
- PWM\_CCU4\_STATUS\_ALREADY\_INITIALIZED : [PWM\\_CCU4.h](#)

- PWM\_CCU4\_STATUS\_FAILURE : [PWM\\_CCU4.h](#)
  - PWM\_CCU4\_STATUS\_SUCCESS : [PWM\\_CCU4.h](#)
  - PWM\_CCU4\_STATUS\_t : [PWM\\_CCU4.h](#)
  - PWM\_CCU4\_Stop() : [PWM\\_CCU4.h](#) , [PWM\\_CCU4.c](#)
  - PWM\_CCU4\_t : [PWM\\_CCU4.h](#)
- 



# PWM\_CCU4

Home					
File List	Globals				
All	Functions	Typedefs	Enumerations	Enumerator	

- PWM\_CCU4\_ClearEvent() : [PWM\\_CCU4.c](#) , [PWM\\_CCU4.h](#)
  - PWM\_CCU4\_ClearTrap() : [PWM\\_CCU4.h](#) , [PWM\\_CCU4.c](#)
  - PWM\_CCU4\_GetAppVersion() : [PWM\\_CCU4.c](#) , [PWM\\_CCU4.h](#)
  - PWM\_CCU4\_GetInterruptStatus() : [PWM\\_CCU4.h](#) , [PWM\\_CCU4.c](#)
  - PWM\_CCU4\_GetTimerStatus() : [PWM\\_CCU4.c](#) , [PWM\\_CCU4.h](#)
  - PWM\_CCU4\_GetTimerValue() : [PWM\\_CCU4.c](#) , [PWM\\_CCU4.h](#)
  - PWM\_CCU4\_Init() : [PWM\\_CCU4.c](#) , [PWM\\_CCU4.h](#)
  - PWM\_CCU4\_SetDither() : [PWM\\_CCU4.h](#) , [PWM\\_CCU4.c](#)
  - PWM\_CCU4\_SetDutyCycle() : [PWM\\_CCU4.c](#) , [PWM\\_CCU4.h](#)
  - PWM\_CCU4\_SetFreq() : [PWM\\_CCU4.c](#) , [PWM\\_CCU4.h](#)
  - PWM\_CCU4\_SetFreqAndDutyCycle() : [PWM\\_CCU4.c](#) , [PWM\\_CCU4.h](#)
  - PWM\_CCU4\_Start() : [PWM\\_CCU4.c](#) , [PWM\\_CCU4.h](#)
  - PWM\_CCU4\_Stop() : [PWM\\_CCU4.h](#) , [PWM\\_CCU4.c](#)
-

# PWM\_CCU4

Home				
File List	Globals			
All	Functions	Typedefs	Enumerations	Enumerator

- PWM\_CCU4\_CONFIG\_t : [PWM\\_CCU4.h](#)
- PWM\_CCU4\_STATE\_t : [PWM\\_CCU4.h](#)
- PWM\_CCU4\_STATUS\_t : [PWM\\_CCU4.h](#)
- PWM\_CCU4\_t : [PWM\\_CCU4.h](#)

# PWM\_CCU4

The screenshot shows a software interface with a navigation bar at the top. The bar has several tabs: "Home", "File List", "Globals", "All", "Functions", "Typedefs", "Enumerations", and "Enumerator". The "Globals" tab is currently selected, indicated by a dark blue background. Below the navigation bar, there is a large, empty rectangular area, likely a content pane.

- PWM\_CCU4\_STATE : [PWM\\_CCU4.h](#)
- PWM\_CCU4\_STATUS : [PWM\\_CCU4.h](#)

# PWM\_CCU4

Home					
File List	Globals				
All	Functions	Typedefs	Enumerations	Enumerator	

- PWM\_CCU4\_STATE\_INITIALIZED : [PWM\\_CCU4.h](#)
  - PWM\_CCU4\_STATE\_RUNNING : [PWM\\_CCU4.h](#)
  - PWM\_CCU4\_STATE\_STOPPED : [PWM\\_CCU4.h](#)
  - PWM\_CCU4\_STATE\_UNINITIALIZED : [PWM\\_CCU4.h](#)
  - PWM\_CCU4\_STATUS\_ALREADY\_INITIALIZED : [PWM\\_CCU4.h](#)
  - PWM\_CCU4\_STATUS\_FAILURE : [PWM\\_CCU4.h](#)
  - PWM\_CCU4\_STATUS\_SUCCESS : [PWM\\_CCU4.h](#)
-

# PWM\_CCU4

Home

Data Structures

## Data structures

## Data Structures

---

struct **PWM\_CCU4\_ConfigType**  
Configuration parameters  
of the PWM\_CCU4 APP.  
[More...](#)

typedef struct **PWM\_CCU4\_ConfigType** **PWM\_CCU4\_CONFIG\_t**  
Configuration parameters  
of the PWM\_CCU4 APP.

typedef struct **PWM\_CCU4\_HandleTypeDef** **PWM\_CCU4\_t**  
Initialization parameters of  
the PWM\_CCU4 APP.

## **Detailed Description**

---



# PWM\_CCU4

<a href="#">Home</a>	
<a href="#">File List</a>	<a href="#">Globals</a>
<a href="#">Code</a>	

## PWM\_CCU4.h

[Go to the documentation of this file.](#)

```
1
67 #ifndef PWM_CCU4_H_
68 #define PWM_CCU4_H_
69
70 /*****
71 * HEADER FILES
72 ****
73 #include <xmc_gpio.h>
74 #include "pwm_ccu4_conf.h"
75
76 #if (!(XMC_LIB_MAJOR_VERSION == 2U) && \
77 (XMC_LIB_MINOR_VERSION >= 0U) && \
78 (XMC_LIB_PATCH_VERSION >= 0U)))
79 #error "PWM_CCU4 requires XMC Peripheral Library v2.0.0 or
higher"
80 #endif
81
82 ****
83 * MACROS
84 ****
85
86 #define PWM_CCU4_MAX_TIMER_COUNT (65535U)
87
88 #define PWM_CCU4_DUTY_FULL_SCALE (10000U) /*100% *
100*/
```

```
89 #define PWM_CCU4_DUTY_SCALE (100U) /*100*/
90
91 #define PWM_CCU4_SYM_DUTY_MAX (10000U) /*duty Max*/
92 #define PWM_CCU4_SYM_DUTY_MIN (0U) /*duty Min*/
93
94 ****
95 * ENUMS
96 ****
97
106 typedef enum PWM_CCU4_STATUS
107 {
111  PWM_CCU4_STATUS_SUCCESS = 0U,
112
116  PWM_CCU4_STATUS_FAILURE = 1U,
117
121  PWM_CCU4_STATUS_ALREADY_INITIALIZED = 2U
122 } PWM_CCU4_STATUS_t;
123
124
128 typedef enum PWM_CCU4_STATE
129 {
135  PWM_CCU4_STATE_UNINITIALIZED = 0U,
136
141  PWM_CCU4_STATE_INITIALIZED = 1U,
142
147  PWM_CCU4_STATE_RUNNING = 2U,
148
153  PWM_CCU4_STATE_STOPPED = 3U
154
155 } PWM_CCU4_STATE_t;
156
161 ****
162 * DATA STRUCTURES
163 ****
171 typedef struct PWM_CCU4_ConfigType
172 {
173  const bool start_control;
```

```
174 const uint16_t period_value;
175 const uint16_t compare_value;
177 const bool int_per_match;
178 const bool int_cmp_match_up;
179 const bool int_cmp_match_down;
180 const bool int_one_match_down;
181 const bool int_e0;
182 const bool int_e1;
183 const bool int_e2;
185 const XMC_CCU4_SLICE_SR_ID_t sr_per_match;
186 const XMC_CCU4_SLICE_SR_ID_t sr_cmp_match_up;
187 const XMC_CCU4_SLICE_SR_ID_t sr_cmp_match_down;
188 const XMC_CCU4_SLICE_SR_ID_t sr_one_match_down;
189 const XMC_CCU4_SLICE_SR_ID_t sr_e0;
190 const XMC_CCU4_SLICE_SR_ID_t sr_e1;
191 const XMC_CCU4_SLICE_SR_ID_t sr_e2;
193 const XMC_CCU4_SLICE_EVENT_CONFIG_t *const
event0_config_ptr;
194 const XMC_CCU4_SLICE_EVENT_CONFIG_t *const
event1_config_ptr;
195 const XMC_CCU4_SLICE_EVENT_CONFIG_t *const
event2_config_ptr;
197 const XMC_CCU4_SLICE_EVENT_t ext_start_event;
198 const XMC_CCU4_SLICE_START_MODE_t ext_start_mode;
200 const XMC_CCU4_SLICE_EVENT_t ext_stop_event;
201 const XMC_CCU4_SLICE_END_MODE_t ext_stop_mode;
203 const XMC_CCU4_SLICE_EVENT_t ext_count_dir_event;
205 const XMC_CCU4_SLICE_EVENT_t ext_gate_event;
207 const XMC_CCU4_SLICE_EVENT_t ext_count_event;
209 const XMC_CCU4_SLICE_EVENT_t ext_load_event;
211 const XMC_CCU4_SLICE_EVENT_t ext_mod_event;
212 const XMC_CCU4_SLICE_MODULATION_MODE_t
ext_mod_mode;
213 const bool ext_mod_sync;
215 const XMC_CCU4_SLICE_EVENT_t ext_override_edge_event;
217 const XMC_CCU4_SLICE_EVENT_t ext_override_level_event;
219 const bool ext_trap_enable;
```

```

220 const XMC_CCU4_SLICE_EVENT_t ext_trap_event;
221 const bool ext_trap_sync;
222 const XMC_CCU4_SLICE_TRAP_EXIT_MODE_t ext_trap_exit;
224 const XMC_CCU4_MULTI_CHANNEL_SHADOW_TRANSFER_t
mcm_shadow_txfr_mode;
226 #if (UC_SERIES == XMC14)/*below feature available in XMC14xx
devices */
227 const XMC_CCU4_SLICE_SHADOW_TRANSFER_MODE_t
shadow_transfer_mode;
228 const uint32_t immediate_write;
229 const uint32_t automatic_shadow_transfer;
230 const bool cascaded_shadow_txfr_enable;
231 #endif
232
233 const XMC_CCU4_SLICE_COMPARE_CONFIG_t *const
ccu4_cc4_slice_timer_ptr;
235 const bool gpio_ch_out_enable;
236 XMC_GPIO_PORT_t *const gpio_ch_out_ptr;
237 const uint8_t gpio_ch_out_pin;
238 const XMC_GPIO_CONFIG_t *const gpio_ch_out_config_ptr;
240 GLOBAL_CCU4_t *const global_ccu4_handle;
242 } PWM_CCU4_CONFIG_t;
243
247 typedef struct PWM_CCU4_HandleType
248 {
249 const PWM_CCU4_CONFIG_t *const config_ptr;
250 XMC_CCU4_MODULE_t *const ccu4_module_ptr;
251 XMC_CCU4_SLICE_t *const ccu4_slice_ptr;
252 const uint8_t kernel_number;
253 const uint8_t slice_number;
254 const uint32_t shadow_txfr_msk;
255 const uint32_t dither_shadow_txfr_msk;
256 const uint32_t prescaler_shadow_txfr_msk;
258 PWM_CCU4_STATE_t state;
259 uint32_t frequency_tclk;
260 uint32_t sym_duty;
261 } PWM_CCU4_t;

```

```
262
272 ****
273 * API Prototypes
274 ****
275 /* Support for C++ */
276 #ifdef __cplusplus
277 extern "C" {
278 #endif
279
302 DAVE_APP_VERSION_t PWM_CCU4_GetAppVersion(void);
303
325 PWM_CCU4_STATUS_t PWM_CCU4_Init(PWM_CCU4_t* const
handle_ptr);
326
352 PWM_CCU4_STATUS_t PWM_CCU4_Start(PWM_CCU4_t*
const handle_ptr);
353
375 PWM_CCU4_STATUS_t PWM_CCU4_Stop(PWM_CCU4_t* const
handle_ptr);
376
398 uint32_t PWM_CCU4_GetTimerValue(PWM_CCU4_t* const
handle_ptr);
399
421 bool PWM_CCU4_GetTimerStatus(PWM_CCU4_t* const
handle_ptr);
422
446 PWM_CCU4_STATUS_t PWM_CCU4_SetFreq(PWM_CCU4_t*
const handle_ptr, uint32_t pwm_freq_hz);
447
473 PWM_CCU4_STATUS_t
PWM_CCU4_SetDutyCycle(PWM_CCU4_t* const handle_ptr, uint32_t
duty_cycle);
474
475
502 PWM_CCU4_STATUS_t
PWM_CCU4_SetFreqAndDutyCycle(PWM_CCU4_t* const handle_ptr,
```

```
uint32_t pwm_freq_hz, uint32_t duty);  
503  
530 void PWM_CCU4_SetDither(PWM_CCU4_t* const handle_ptr,  
bool dither_period, bool dither_comp, uint8_t dither_value);  
531  
552 void PWM_CCU4_ClearTrap(PWM_CCU4_t* const handle_ptr);  
553  
579 bool PWM_CCU4_GetInterruptStatus(PWM_CCU4_t* const  
handle_ptr, XMC_CCU4_SLICE_IRQ_ID_t pwm_interrupt);  
580  
603 void PWM_CCU4_ClearEvent(PWM_CCU4_t* const handle_ptr,  
XMC_CCU4_SLICE_IRQ_ID_t pwm_interrupt);  
604  
605  
606 #include "pwm_ccu4_extern.h"  
607  
611 #ifdef __cplusplus  
612 }  
613 #endif  
614  
615 #endif /* PWM_CCU4_H */  
616
```

---

# PWM\_CCU4

Home | File List | Globals | Code >

## PWM\_CCU4.c

Go to the documentation of this file.

```
1
65 ****
66 * HEADER FILES
67 ****
68 #include "pwm_ccu4.h"
69
70 ****
71 * MACROS
72 ****
73
74 ****
75 * LOCAL DATA
76 ****
77
78 ****
79 * LOCAL ROUTINES
80 ****
81
82 /* Initialize the App Interrupts */
83 static void PWM_CCU4_llInit_Interrupt(PWM_CCU4_t* handle_ptr);
84
85 /* Initialize the App events and configurations */
```

```
86 static void PWM_CCU4_IConfigure_Events(PWM_CCU4_t*
handle_ptr);
87
88 /*****
89 * API IMPLEMENTATION
90 ****
91
92 /* API to retrieve App version info */
93 DAVE_APP_VERSION_t PWM_CCU4_GetAppVersion(void)
94 {
95 DAVE_APP_VERSION_t version;
96
97 version.major = PWM_CCU4_MAJOR_VERSION;
98 version.minor = PWM_CCU4_MINOR_VERSION;
99 version.patch = PWM_CCU4_PATCH_VERSION;
100
101 return version;
102 }
103
104 /* This function initializes the app */
105 PWM_CCU4_STATUS_t PWM_CCU4_Init(PWM_CCU4_t*
handle_ptr)
106 {
107 PWM_CCU4_STATUS_t status;
108 GLOBAL_CCU4_STATUS_t status_ccu4_global;
109 uint32_t frequency_module;
110 uint32_t prescalar;
111
112 status = PWM_CCU4_STATUS_FAILURE;
113 status_ccu4_global = GLOBAL_CCU4_STATUS_FAILURE;
114 XMC_ASSERT("PWM_CCU4_Init:handle_ptr is NULL",
(handle_ptr != NULL));
115
116 if (PWM_CCU4_STATE_UNINITIALIZED == handle_ptr->state)
117 {
118 /* Initialize consumed Apps */
```

```
119 status_ccu4_global = GLOBAL_CCU4_Init(handle_ptr->config_ptr->global_ccu4_handle);
120
121 /* Initialize CCU4x_CC4y slice */
122 if (GLOBAL_CCU4_STATUS_SUCCESS == status_ccu4_global)
123 {
124 XMC_DEBUG("PWM_CCU4_Init:Initilizing slice");
125
126 /* Configure CCU4x_CC4y slice as timer */
127 XMC_CCU4_SLICE_CompareInit(handle_ptr->ccu4_slice_ptr,
handle_ptr->config_ptr->ccu4_cc4_slice_timer_ptr);
128 /* Set period match value of the timer */
129 XMC_CCU4_SLICE_SetTimerPeriodMatch(handle_ptr-
>ccu4_slice_ptr, handle_ptr->config_ptr->period_value);
130
131 /* Set timer compare match value for channel 1 */
132 XMC_CCU4_SLICE_SetTimerCompareMatch(handle_ptr-
>ccu4_slice_ptr, (uint16_t) handle_ptr->config_ptr->compare_value);
133
134 if (1U == handle_ptr->config_ptr->ccu4_cc4_slice_timer_ptr-
>mcm_enable)
135 {
136 XMC_CCU4_SetMultiChannelShadowTransferMode(handle_ptr-
>ccu4_module_ptr,
137 (uint32_t) handle_ptr->config_ptr->mcm_shadow_txfr_mode);
138 }
139
140 #if (UC_SERIES == XMC14) /*below feature available in XMC14xx
devices */
141 XMC_CCU4_SLICE_SetShadowTransferMode(handle_ptr-
>ccu4_slice_ptr, handle_ptr->config_ptr->shadow_transfer_mode);
142
XMC_CCU4_SLICE_WriteImmediateAfterShadowTransfer(handle_ptr-
>ccu4_slice_ptr,
143 handle_ptr->config_ptr->immediate_write);
144
XMC_CCU4_SLICE_EnableAutomaticShadowTransferRequest(handle_
```

```

>ccu4_slice_ptr,
145 handle_ptr->config_ptr->automatic_shadow_transfer);
146 if((bool)true == handle_ptr->config_ptr-
>cascaded_shadow_txfr_enable)
147 {
148
XMC_CCU4_SLICE_EnableCascadedShadowTransfer(handle_ptr-
>ccu4_slice_ptr);
149 }
150 #endif
151
152 /* Transfer value from shadow timer registers to actual timer
registers */
153 XMC_CCU4_EnableShadowTransfer(handle_ptr-
>ccu4_module_ptr, handle_ptr->shadow_txfr_msk);
154 XMC_CCU4_EnableShadowTransfer(handle_ptr-
>ccu4_module_ptr, handle_ptr->dither_shadow_txfr_msk);
155
156 /* Configure events */
157 PWM_CCU4_IConfigure_Events(handle_ptr);
158
159 /* Enable the interrupts */
160 PWM_CCU4_IInit_Interrupt(handle_ptr);
161
162 /*Initializes the GPIO*/
163 if ((bool) true == handle_ptr->config_ptr->gpio_ch_out_enable)
164 {
165 XMC_GPIO_Init(handle_ptr->config_ptr->gpio_ch_out_ptr,
handle_ptr->config_ptr->gpio_ch_out_pin,
166 handle_ptr->config_ptr->gpio_ch_out_config_ptr);
167 }
168
169 frequency_module = handle_ptr->config_ptr-
>global_ccu4_handle->module_frequency;
170 prescalar = (uint32_t) handle_ptr->config_ptr-
>ccu4_cc4_slice_timer_ptr->prescaler_initval;
171 frequency_module = frequency_module / ((uint32_t) 1 <<

```

```

prescalar);
172 handle_ptr->frequency_tclk = frequency_module;
173
174 handle_ptr->state = PWM_CCU4_STATE_INITIALIZED;
175 status = PWM_CCU4_STATUS_SUCCESS;
176
177 /* Start the PWM generation if start at initialization is enabled */
178 if ((bool) true == handle_ptr->config_ptr->start_control)
179 {
180 status = PWM_CCU4_Start(handle_ptr);
181 }
182 }
183 else
184 {
185 handle_ptr->state = PWM_CCU4_STATE_UNINITIALIZED;
186 }
187
188 }
189 else
190 {
191 status = PWM_CCU4_STATUS_ALREADY_INITIALIZED;
192
XMC_DEBUG("PWM_CCU4_Init:PWM_CCU4_STATUS_ALREADY_INIT");
193 }
194
195 return (status);
196 } /* end of PWM_CCU4_Init() api */
197
198 static void PWM_CCU4_lInit_Interrupt(PWM_CCU4_t* handle_ptr)
199 {
200
201 /* Enable events. Bind event to corresponding service request
node.Enable Interrupts. The user may choose to
202 disable the interrupts by LLD calls. */
203 if ((bool) true == handle_ptr->config_ptr->int_per_match)
204 {
205 XMC_DEBUG("PWM_CCU4_Init: Interrupt period match enable");

```

```
206 XMC_CCU4_SLICE_SetInterruptNode(handle_ptr->ccu4_slice_ptr, XMC_CCU4_SLICE_IRQ_ID_PERIOD_MATCH,
207 handle_ptr->config_ptr->sr_per_match);
208 XMC_CCU4_SLICE_EnableEvent(handle_ptr->ccu4_slice_ptr, XMC_CCU4_SLICE_IRQ_ID_PERIOD_MATCH);
209 }
210
211 if ((bool) true == handle_ptr->config_ptr->int_cmp_match_up)
212 {
213 XMC_DEBUG("PWM_CCU4_Init: Interrupt compare match up enable");
214 XMC_CCU4_SLICE_SetInterruptNode(handle_ptr->ccu4_slice_ptr,
XMC_CCU4_SLICE_IRQ_ID_COMPARE_MATCH_UP,
215 handle_ptr->config_ptr->sr_cmp_match_up);
216 XMC_CCU4_SLICE_EnableEvent(handle_ptr->ccu4_slice_ptr, XMC_CCU4_SLICE_IRQ_ID_COMPARE_MATCH_UP);
217 }
218
219 if ((bool) true == handle_ptr->config_ptr->int_cmp_match_down)
220 {
221 XMC_DEBUG("PWM_CCU4_Init: Interrupt compare match down enable");
222 XMC_CCU4_SLICE_SetInterruptNode(handle_ptr->ccu4_slice_ptr,
XMC_CCU4_SLICE_IRQ_ID_COMPARE_MATCH_DOWN,
223 handle_ptr->config_ptr->sr_cmp_match_down);
224 XMC_CCU4_SLICE_EnableEvent(handle_ptr->ccu4_slice_ptr, XMC_CCU4_SLICE_IRQ_ID_COMPARE_MATCH_DOWN);
225 }
226
227 if ((bool) true == handle_ptr->config_ptr->int_one_match_down)
228 {
229 XMC_DEBUG("PWM_CCU4_Init: Interrupt one match enable");
230 XMC_CCU4_SLICE_SetInterruptNode(handle_ptr->ccu4_slice_ptr, XMC_CCU4_SLICE_IRQ_ID_ONE_MATCH,
231 handle_ptr->config_ptr->sr_one_match_down);
```

```
232 XMC_CCU4_SLICE_EnableEvent(handle_ptr->ccu4_slice_ptr,
XMC_CCU4_SLICE_IRQ_ID_ONE_MATCH);
233 }
234
235 if ((bool) true == handle_ptr->config_ptr->int_e0)
236 {
237 XMC_DEBUG("PWM_CCU4_Init: Interrupt event 0 enable");
238 XMC_CCU4_SLICE_SetInterruptNode(handle_ptr-
>ccu4_slice_ptr, XMC_CCU4_SLICE_IRQ_ID_EVENT0,
239 handle_ptr->config_ptr->sr_e0);
240 XMC_CCU4_SLICE_EnableEvent(handle_ptr->ccu4_slice_ptr,
XMC_CCU4_SLICE_IRQ_ID_EVENT0);
241 }
242
243 if ((bool) true == handle_ptr->config_ptr->int_e1)
244 {
245 XMC_DEBUG("PWM_CCU4_Init: Interrupt event 1 enable");
246 XMC_CCU4_SLICE_SetInterruptNode(handle_ptr-
>ccu4_slice_ptr, XMC_CCU4_SLICE_IRQ_ID_EVENT1,
247 handle_ptr->config_ptr->sr_e1);
248 XMC_CCU4_SLICE_EnableEvent(handle_ptr->ccu4_slice_ptr,
XMC_CCU4_SLICE_IRQ_ID_EVENT1);
249 }
250
251 if ((bool) true == handle_ptr->config_ptr->int_e2)
252 {
253 XMC_DEBUG("PWM_CCU4_Init: Interrupt event 2 enable");
254 XMC_CCU4_SLICE_SetInterruptNode(handle_ptr-
>ccu4_slice_ptr, XMC_CCU4_SLICE_IRQ_ID_EVENT2,
255 handle_ptr->config_ptr->sr_e2);
256 XMC_CCU4_SLICE_EnableEvent(handle_ptr->ccu4_slice_ptr,
XMC_CCU4_SLICE_IRQ_ID_EVENT2);
257 }
258 }
259
260 static void PWM_CCU4_IConfigure_Events(PWM_CCU4_t*
handle_ptr)
```

```
261 {
262
263 /* Configure slice to a external event 0 */
264 XMC_CCU4_SLICE_ConfigureEvent(handle_ptr->ccu4_slice_ptr,
XMC_CCU4_SLICE_EVENT_0,
265 handle_ptr->config_ptr->event0_config_ptr);
266
267 /* Configure slice to a external event 1 */
268 XMC_CCU4_SLICE_ConfigureEvent(handle_ptr->ccu4_slice_ptr,
XMC_CCU4_SLICE_EVENT_1,
269 handle_ptr->config_ptr->event1_config_ptr);
270
271 /* Configure slice to a external event 2 */
272 XMC_CCU4_SLICE_ConfigureEvent(handle_ptr->ccu4_slice_ptr,
XMC_CCU4_SLICE_EVENT_2,
273 handle_ptr->config_ptr->event2_config_ptr);
274
275 /* External signal controls start of the timer */
276 if (XMC_CCU4_SLICE_EVENT_NONE != handle_ptr->config_ptr-
>ext_start_event)
277 {
278 XMC_CCU4_SLICE_StartConfig(handle_ptr->ccu4_slice_ptr,
handle_ptr->config_ptr->ext_start_event,
279 handle_ptr->config_ptr->ext_start_mode);
280 }
281
282 /* External signal can stop the timer */
283 if (XMC_CCU4_SLICE_EVENT_NONE != handle_ptr->config_ptr-
>ext_stop_event)
284 {
285 XMC_CCU4_SLICE_StopConfig(handle_ptr->ccu4_slice_ptr,
handle_ptr->config_ptr->ext_stop_event,
286 handle_ptr->config_ptr->ext_stop_mode);
287 }
288
289 /* External signal can change the timer counting direction */
290 if (XMC_CCU4_SLICE_EVENT_NONE != handle_ptr->config_ptr-
```

```

>ext_count_dir_event)
291 {
292 XMC_CCU4_SLICE_DirectionConfig(handle_ptr->ccu4_slice_ptr,
handle_ptr->config_ptr->ext_count_dir_event);
293 }
294 /* External signal can stop the timer and the timer value remains
same */
295 if (XMC_CCU4_SLICE_EVENT_NONE != handle_ptr->config_ptr-
>ext_gate_event)
296 {
297 XMC_CCU4_SLICE_GateConfig(handle_ptr->ccu4_slice_ptr,
handle_ptr->config_ptr->ext_gate_event);
298 }
299 /* Timer increments on external signal */
300 if (XMC_CCU4_SLICE_EVENT_NONE != handle_ptr->config_ptr-
>ext_count_event)
301 {
302 XMC_CCU4_SLICE_CountConfig(handle_ptr->ccu4_slice_ptr,
handle_ptr->config_ptr->ext_count_event);
303 }
304 /* Timer gets loaded with compare register value or period register
value on external signal */
305 if (XMC_CCU4_SLICE_EVENT_NONE != handle_ptr->config_ptr-
>ext_load_event)
306 {
307 XMC_CCU4_SLICE_LoadConfig(handle_ptr->ccu4_slice_ptr,
handle_ptr->config_ptr->ext_load_event);
308 }
309 /* External signal PWM signal (ST bit) output gets modulated by
external signal */
310 if (XMC_CCU4_SLICE_EVENT_NONE != handle_ptr->config_ptr-
>ext_mod_event)
311 {
312 XMC_CCU4_SLICE_ModulationConfig(handle_ptr-
>ccu4_slice_ptr, handle_ptr->config_ptr->ext_mod_event,
313 handle_ptr->config_ptr->ext_mod_mode, handle_ptr->config_ptr-
>ext_mod_sync);

```

```

314 }
315
316 /* PWM signal (ST bit) output gets modulated by external signal */
317 if (XMC_CCU4_SLICE_EVENT_2 == handle_ptr->config_ptr-
>ext_trap_event)
318 {
319 XMC_CCU4_SLICE_TrapConfig(handle_ptr->ccu4_slice_ptr,
handle_ptr->config_ptr->ext_trap_exit,
320 handle_ptr->config_ptr->ext_trap_sync);
321
322 if ((bool) true == handle_ptr->config_ptr->ext_trap_enable)
323 {
324 XMC_CCU4_SLICE_EnableTrap(handle_ptr->ccu4_slice_ptr);
325 }
326 }
327 if ((XMC_CCU4_SLICE_EVENT_1 == handle_ptr->config_ptr-
>ext_override_edge_event) && (XMC_CCU4_SLICE_EVENT_2
328 == handle_ptr->config_ptr->ext_override_level_event))
329 {
330
XMC_CCU4_SLICE_ConfigureStatusBitOverrideEvent(handle_ptr-
>ccu4_slice_ptr,
331 handle_ptr->config_ptr->event1_config_ptr,
332 handle_ptr->config_ptr->event2_config_ptr);
333 XMC_CCU4_SLICE_StatusBitOverrideConfig(handle_ptr-
>ccu4_slice_ptr);
334 }
335
336 }
337 ****
338 /*Starts the CCU4_CC4 slice. This needs to be called even if
external start is configured.*/
339 PWM_CCU4_STATUS_t PWM_CCU4_Start(PWM_CCU4_t*
handle_ptr)
340 {
341 PWM_CCU4_STATUS_t status;
342

```

```

343 status = PWM_CCU4_STATUS_FAILURE;
344 XMC_ASSERT("PWM_CCU4_Start:handle_ptr NULL",
(handle_ptr != NULL));
345 if ((PWM_CCU4_STATE_INITIALIZED == handle_ptr->state) ||
(PWM_CCU4_STATE_STOPPED == handle_ptr->state))
346 {
347 /* clear IDLE mode for the slice; Start timer */
348 XMC_CCU4_EnableClock(handle_ptr->ccu4_module_ptr,
handle_ptr->slice_number);
349
350 if (XMC_CCU4_SLICE_EVENT_NONE == handle_ptr->config_ptr-
>ext_start_event)
351 {
352 XMC_CCU4_SLICE_StartTimer(handle_ptr->ccu4_slice_ptr);
353 }
354
355 handle_ptr->state = PWM_CCU4_STATE_RUNNING;
356 status = PWM_CCU4_STATUS_SUCCESS;
357 XMC_DEBUG("PWM_CCU4_Start:start PWM");
358 }
359 return (status);
360 } /* end of PWM_CCU4_Start() api */
361 ****
362 /*Stops the CCU4_CC4 slice. */
363 PWM_CCU4_STATUS_t PWM_CCU4_Stop(PWM_CCU4_t*
handle_ptr)
364 {
365 PWM_CCU4_STATUS_t status;
366
367 status = PWM_CCU4_STATUS_FAILURE;
368 XMC_ASSERT("PWM_CCU4_Stop:handle_ptr NULL",
(handle_ptr != NULL));
369 if (PWM_CCU4_STATE_UNINITIALIZED != handle_ptr->state)
370 {
371 XMC_CCU4_SLICE_StopTimer(handle_ptr->ccu4_slice_ptr);
372 XMC_CCU4_SLICE_ClearTimer(handle_ptr->ccu4_slice_ptr);
373

```

```

374 handle_ptr->state = PWM_CCU4_STATE_STOPPED;
375 status = PWM_CCU4_STATUS_SUCCESS;
376 XMC_DEBUG("PWM_CCU4_Stop:stop PWM");
377 }
378 return (status);
379
380 } /* end of PWM_CCU4_Stop() api */
381 ****
382 /*Gets the timer value of CCU4_CC4 slice. */
383 uint32_t PWM_CCU4_GetTimerValue(PWM_CCU4_t* handle_ptr)
384 {
385 uint32_t timer_value;
386 XMC_ASSERT("PWM_CCU4_GetTimerValue:handle_ptr NULL",
(handle_ptr != NULL));
387 timer_value = (uint32_t)
XMC_CCU4_SLICE_GetTimerValue(handle_ptr->ccu4_slice_ptr);
388 XMC_DEBUG("PWM_CCU4_GetTimerValue:timer value");
389 return (timer_value);
390 }/* end of PWM_CCU4_GetTimerValue() api */
391 ****
392 /*Gets the status of CCU4_CC4 slice. */
393 bool PWM_CCU4_GetTimerStatus(PWM_CCU4_t* handle_ptr)
394 {
395 bool status_timer;
396 XMC_ASSERT("PWM_CCU4_GetTimerStatus:handle_ptr NULL",
(handle_ptr != NULL));
397 status_timer = XMC_CCU4_SLICE_IsTimerRunning(handle_ptr-
>ccu4_slice_ptr);
398 return (status_timer);
399
400 }/* end of PWM_CCU4_GetStatus() api */
401
402 ****
403
404 /*Sets the frequency for CCU4_CC4 slice. */
405 PWM_CCU4_STATUS_t PWM_CCU4_SetFreq(PWM_CCU4_t*
handle_ptr, uint32_t pwm_freq_hz)

```

```

406 {
407     PWM_CCU4_STATUS_t status;
408     uint32_t frequency_tclk;
409     uint32_t period;
410     uint32_t duty;
411     uint16_t compare;
412
413     status = PWM_CCU4_STATUS_FAILURE;
414     frequency_tclk = 0U;
415     XMC_ASSERT("PWM_CCU4_SetFreq:handle_ptr NULL",
416     (handle_ptr != NULL));
416     if (PWM_CCU4_STATE_UNINITIALIZED != handle_ptr->state)
417     {
418         if (0U == pwm_freq_hz)
419         {
420             XMC_DEBUG("PWM_CCU4_SetFreq:cannot set frequency 0Hz");
421         }
422     else
423     {
424         frequency_tclk = handle_ptr->frequency_tclk;
425         period = frequency_tclk / pwm_freq_hz;
426
427         if ((uint32_t) XMC_CCU4_SLICE_TIMER_COUNT_MODE_CA ==
428             handle_ptr->config_ptr->ccu4_cc4_slice_timer_ptr->timer_mode)
428     {
429         period = period >> 1U; /*divide by 2*/
430     }
431
432     if ((period != 0U) && (period <=
433         PWM_CCU4_MAX_TIMER_COUNT))
433     {
434         /*Calculate the current duty cycle in no-timer concatenation
435 mode*/
435         duty = handle_ptr->sym_duty;
436
437         duty = (PWM_CCU4_DUTY_FULL_SCALE - duty);
438         duty = duty * period;

```

```

439 duty = duty / PWM_CCU4_DUTY_FULL_SCALE;
440
441 compare = (uint16_t) duty;
442
443 XMC_CCU4_SLICE_SetTimerPeriodMatch(handle_ptr-
>ccu4_slice_ptr, (uint16_t)(period - 1U));
444 XMC_CCU4_SLICE_SetTimerCompareMatch(handle_ptr-
>ccu4_slice_ptr, compare);
445 XMC_CCU4_EnableShadowTransfer(handle_ptr-
>ccu4_module_ptr, handle_ptr->shadow_txfr_msk);
446 XMC_DEBUG("PWM_CCU4_SetFreq:frequency set");
447 status = PWM_CCU4_STATUS_SUCCESS;
448 }
449 }
450 }
451 return (status);
452
453 } /* end of PWM_CCU4_SetFreqSymmetric() api */
454
455 ****
456
457 /*Sets the duty cycle (uint32_t) for CCU4_CC4 slice. */
458 PWM_CCU4_STATUS_t
459 PWM_CCU4_SetDutyCycle(PWM_CCU4_t* handle_ptr, uint32_t
duty_cycle)
460 {
461 PWM_CCU4_STATUS_t status;
462 uint32_t period;
463 uint32_t compare;
464
465 status = PWM_CCU4_STATUS_FAILURE;
466 XMC_ASSERT("PWM_CCU4_SetDutyCycle:handle_ptr NULL",
(handle_ptr != NULL));
467 if (PWM_CCU4_STATE_UNINITIALIZED != handle_ptr->state)
468 {
469 /* duty cycle has to be in between 0 and 100 */
470 if ((duty_cycle > PWM_CCU4_SYM_DUTY_MAX))

```

```

470 {
471 XMC_DEBUG("PWM_CCU4_SetDutyCycle:Cannot set duty cycle
> 100%");
472 }
473 else
474 {
475 period = (uint32_t)
XMC_CCU4_SLICE_GetTimerPeriodMatch(handle_ptr-
>ccu4_slice_ptr) + 1U;
476
477 /* Duty Cycle(symmetric) = (PR-CR1)+1 / period */
478 compare = ((period * (PWM_CCU4_DUTY_FULL_SCALE -
duty_cycle)) / PWM_CCU4_DUTY_FULL_SCALE);
479
480 XMC_CCU4_SLICE_SetTimerCompareMatch(handle_ptr-
>ccu4_slice_ptr, (uint16_t) compare);
481 XMC_CCU4_EnableShadowTransfer(handle_ptr-
>ccu4_module_ptr, handle_ptr->shadow_txfr_msk);
482
483 handle_ptr->sym_duty = duty_cycle;
484
485 XMC_DEBUG("PWM_CCU4_SetDutyCycle:dutycycle set");
486 status = PWM_CCU4_STATUS_SUCCESS;
487 }
488 }
489 return (status);
490 } /* end of PWM_CCU4_SetDutyCycle() api */
491
492 ****
493
494 /*Sets the frequency and duty cycle for CCU4_CC4 slice
Symmetric Mode. */
495 PWM_CCU4_STATUS_t
PWM_CCU4_SetFreqAndDutyCycle(PWM_CCU4_t* handle_ptr,
uint32_t pwm_freq_hz, uint32_t duty)
496 {
497

```

```
498 PWM_CCU4_STATUS_t status;
499 uint32_t frequency_tclk;
500 uint32_t period;
501 uint32_t compare;
502
503 status = PWM_CCU4_STATUS_FAILURE;
504 frequency_tclk = 0U;
505 XMC_ASSERT("PWM_CCU4_SetFreqAndDutyCycle:handle_ptr
NULL", (handle_ptr != NULL));
506 if (PWM_CCU4_STATE_UNINITIALIZED != handle_ptr->state)
507 {
508 if (0U == pwm_freq_hz)
509 {
510
XMC_DEBUG("PWM_CCU4_SetFreqAndDutyCycleSymmetric:cannot
set frequency 0Hz");
511 }
512 else if (duty > PWM_CCU4_SYM_DUTY_MAX)
513 {
514 XMC_DEBUG("PWM_CCU4_SetFreqAndDutyCycle:duty >
100%");
515 }
516 else
517 {
518 frequency_tclk = handle_ptr->frequency_tclk;
519 period = frequency_tclk / pwm_freq_hz;
520
521 if ((uint32_t) XMC_CCU4_SLICE_TIMER_COUNT_MODE_CA ==
handle_ptr->config_ptr->ccu4_cc4_slice_timer_ptr->timer_mode)
522 {
523 period = period >> 1U; /*divide by 2*/
524 }
525
526 if ((period != 0U) && (period <=
PWM_CCU4_MAX_TIMER_COUNT))
527 {
528 /*Calculate the current duty cycle in no-timer concatenation
```

```
mode*/  
529 compare = ((period * (PWM_CCU4_DUTY_FULL_SCALE - duty))  
/ PWM_CCU4_DUTY_FULL_SCALE);  
530  
531 XMC_CCU4_SLICE_SetTimerPeriodMatch(handle_ptr-  
>ccu4_slice_ptr, (uint16_t)(period - 1U));  
532 XMC_CCU4_SLICE_SetTimerCompareMatch(handle_ptr-  
>ccu4_slice_ptr, (uint16_t) compare);  
533  
534 XMC_CCU4_EnableShadowTransfer(handle_ptr-  
>ccu4_module_ptr, handle_ptr->shadow_txfr_msk);  
535  
536 handle_ptr->sym_duty = duty;  
537  
538 XMC_DEBUG("PWM_CCU4_SetFreqAndDutyCycle:frequency  
set");  
539 status = PWM_CCU4_STATUS_SUCCESS;  
540 }  
541 }  
542 }  
543 return (status);  
544  
545 /* end of PWM_CCU4_SetFreqAndDutyCycle() api */  
546  
547 ****  
548  
549 /*Sets the dither value, enables the dither. */  
550 void PWM_CCU4_SetDither(PWM_CCU4_t* handle_ptr, bool  
dither_period, bool dither_comp, uint8_t dither_value)  
551 {  
552  
553 XMC_ASSERT("PWM_CCU4_SetDither:handle_ptr NULL",  
(handle_ptr != NULL));  
554 XMC_CCU4_SLICE_EnableDithering(handle_ptr->ccu4_slice_ptr,  
dither_period, dither_comp, dither_value);  
555 XMC_CCU4_EnableShadowTransfer(handle_ptr-  
>ccu4_module_ptr, handle_ptr->dither_shadow_txfr_msk);
```

```
556 XMC_DEBUG("PWM_CCU4_SetDither:dither compare value  
set");  
557  
558 }/* end of PWM_CCU4_SetDither() api */  
559  
560 /*****  
561  
562 /*exits trap condition if trap signal is inactive */  
563 void PWM_CCU4_ClearTrap(PWM_CCU4_t* handle_ptr)  
564 {  
565  
566 XMC_ASSERT("PWM_CCU4_ClearTrap:handle_ptr NULL",  
(handle_ptr != NULL));  
567 XMC_CCU4_SLICE_ClearEvent(handle_ptr->ccu4_slice_ptr,  
XMC_CCU4_SLICE_IRQ_ID_EVENT2);  
568 XMC_DEBUG("PWM_CCU4_ClearTrap:trap event cleared");  
569  
570 }/* end of PWM_CCU4_ClearTrap() api */  
571  
572 /*****  
573  
574 /*Gets the interrupt status of CCU4_CC4 slice. */  
575 bool PWM_CCU4_GetInterruptStatus(PWM_CCU4_t* handle_ptr,  
XMC_CCU4_SLICE_IRQ_ID_t pwm_interrupt)  
576 {  
577 bool status = (bool) false;  
578 XMC_ASSERT("PWM_CCU4_GetInterruptStatus:handle_ptr  
NULL", (handle_ptr != NULL));  
579 status = XMC_CCU4_SLICE_GetEvent(handle_ptr-  
>ccu4_slice_ptr, pwm_interrupt);  
580 return (status);  
581 }/* end of PWM_CCU4_GetInterruptStatus() api */  
582  
583 /*****  
584  
585 /*Acknowledges the interrupt of CCU4_CC4 slice. */  
586 void PWM_CCU4_ClearEvent(PWM_CCU4_t* handle_ptr,
```

```
XMC_CCU4_SLICE_IRQ_ID_t pwm_interrupt)
587 {
588 XMC_ASSERT("PWM_CCU4_ClearEvent:handle_ptr NULL",
589 (handle_ptr != NULL));
590 XMC_DEBUG("PWM_CCU4_ClearEvent:Acknowledge Interrupt");
591 } /* end of PWM_CCU4_ClearEvent() api */
592
593 /* end of CCU4 function definitions */
594
```

---

# PWM\_CCU4

Home

# Enumerations

```
enum PWM_CCU4_STATUS {
    PWM_CCU4_STATUS_SUCCESS
    PWM_CCU4_STATUS_FAILURE
    PWM_CCU4_STATUS_ALREADY_SET
} = 2U }
```

```
enum PWM_CCU4_STATE {
    PWM_CCU4_STATE_UNINITIALI
    PWM_CCU4_STATE_INITIALIZI
    PWM_CCU4_STATE_RUNNING
    PWM_CCU4_STATE_STOPPED
}
```

The type identifies APP state. Mc

**typedef enum PWM\_CCU4\_STATUS PWM\_CCU4\_STATUS\_t**  
The type identifies the APP status.

**typedef enum PWM\_CCU4\_STATE PWM\_CCU4\_STATE\_t**  
The type identifies APP state.

## **Detailed Description**

## Enumeration Type Documentation

### enum PWM\_CCU4\_STATE

The type identifies APP state.

Enumerator	Description
PWM_CCU4_STATE_UNINITIALIZED	default state after power on reset PWM_CCU4 APP is in uninitialized mode. The corresponding CCU4 timer is not configured. PWM pulses is not generated.
PWM_CCU4_STATE_INITIALIZED	PWM_CCU4 APP is in initialized mode. The corresponding CCU4 timer is configured. The corresponding CCU4 timer is not started(not running).
PWM_CCU4_STATE_RUNNING	PWM_CCU4 APP is in running mode. The corresponding CCU4 timer is running. Trigger signal for any of the configured Interrupt or service request in the CCU4 timer is triggered.
PWM_CCU4_STATE_STOPPED	PWM_CCU4 APP is in stopped mode. The

corresponding CCU4 timer is stopped. Trigger signal for any of the configured Interrupt or service request in the CCU4 timer is not triggered.

Definition at line [128](#) of file **PWM\_CCU4.h**.

### **enum PWM\_CCU4\_STATUS**

The type identifies the APP status.

Enumerator	
<i>PWM_CCU4_STATUS_SUCCESS</i>	STATUS SUCCESS
<i>PWM_CCU4_STATUS_FAILURE</i>	STATUS FAILURE
<i>PWM_CCU4_STATUS_ALREADY_INITIALIZED</i>	STATUS ALREADY INITIALIZED

Definition at line [106](#) of file **PWM\_CCU4.h**.

# PWM\_CCU4

Home

Code >

## Code Directory Reference

## Files

---

file [PWM\\_CCU4.c](#) [code]

---

file [PWM\\_CCU4.h](#) [code]

---



# PWM\_CCU4

Home

## Methods

DAVE\_APP\_VERSION\_t **PWM\_CCU4\_GetAppVersion** (void)  
Retrieves the version of the PWM\_CCU4 APP. [More...](#)

PWM\_CCU4\_STATUS\_t **PWM\_CCU4\_Init** (PWM\_CCU4\_t \*const handle\_ptr)  
Initializes the PWM\_CCU4 APP. [More...](#)

PWM\_CCU4\_STATUS\_t **PWM\_CCU4\_Start** (PWM\_CCU4\_t \*const handle\_ptr)  
Start the selected CCU4 slice. [More...](#)

PWM\_CCU4\_STATUS\_t **PWM\_CCU4\_Stop** (PWM\_CCU4\_t \*const handle\_ptr)  
Stop the selected CCU4 slice. [More...](#)

uint32\_t **PWM\_CCU4\_GetTimerValue**  
(PWM\_CCU4\_t \*const handle\_ptr)  
Returns the timer value. [More...](#)

bool **PWM\_CCU4\_GetTimerStatus**  
(PWM\_CCU4\_t \*const handle\_ptr)  
Returns the timer status. [More...](#)

PWM\_CCU4\_STATUS\_t **PWM\_CCU4\_SetFreq** (PWM\_CCU4\_t  
\*const handle\_ptr, uint32\_t pwm\_freq\_hz)  
Sets the PWM frequency. [More...](#)

**PWM\_CCU4\_SetDutyCycle**  
**PWM\_CCU4\_STATUS\_t** (**PWM\_CCU4\_t** \*const handle\_ptr, uint32\_t duty\_cycle)  
Sets the duty cycle of PWM. [More...](#)

---

**PWM\_CCU4\_SetFreqAndDutyCycle**  
**PWM\_CCU4\_STATUS\_t** (**PWM\_CCU4\_t** \*const handle\_ptr, uint32\_t pwm\_freq\_hz, uint32\_t duty)  
Sets the frequency duty cycle of PWM.  
[More...](#)

---

**PWM\_CCU4\_SetDither** (**PWM\_CCU4\_t**  
void \*const handle\_ptr, bool dither\_period, bool  
dither\_comp, uint8\_t dither\_value)  
Sets the dither value for period , duty or  
both. [More...](#)

---

void **PWM\_CCU4\_ClearTrap** (**PWM\_CCU4\_t**  
\*const handle\_ptr)  
Clears the trap event. [More...](#)

---

bool **PWM\_CCU4\_GetInterruptStatus**  
(**PWM\_CCU4\_t** \*const handle\_ptr,  
XMC\_CCU4\_SLICE\_IRQ\_ID\_t  
pwm\_interrupt)  
Returns the interrupt status. [More...](#)

---

void **PWM\_CCU4\_ClearEvent** (**PWM\_CCU4\_t**  
\*const handle\_ptr,  
XMC\_CCU4\_SLICE\_IRQ\_ID\_t  
pwm\_interrupt)  
Acknowledges the interrupt. [More...](#)

## **Detailed Description**

### **Methods**

## Function Documentation

```
void PWM_CCU4_ClearEvent( PWM_CCU4_t *const handle,
                           XMC_CCU4_SLICE_IRQ_ID_t pwm_
                           )
```

Acknowledges the interrupt.

### Parameters

**handle\_ptr** Pointer to PWM\_CCU4\_t structure containing APP parameters.  
**pwm\_interrupt** interrupt ID

### Returns

#### Description:

Clears the interrupt status flag, provided the interrupt condition no longer exists.

Example Usage:

```
#include <DAVE.h>
int main(void)
{
    DAVE_Init();
    PWM_CCU4_ClearEvent(&PWM_CCU4_0,XMC_CCU4_SLICE_IRQ_ID);
    PWM_CCU4_ClearEvent(&PWM_CCU4_0,XMC_CCU4_SLICE_IRQ_ID);
    while(1);
    return 0;
}
```

Definition at line 586 of file [PWM\\_CCU4.c](#).

References [PWM\\_CCU4\\_HandleType::ccu4\\_slice\\_ptr](#).

## **void PWM\_CCU4\_ClearTrap ( PWM\_CCU4\_t \*const handle\_ptr )**

Clears the trap event.

### **Parameters**

**handle\_ptr** Pointer to PWM\_CCU4\_t structure containing APP parameters.

### **Returns**

void

### **Description:**

Clears the trap event provided the trap condition no longer exists.

Example Usage:

```
#include <DAVE.h>
int main(void)
{
    DAVE_Init();
    PWM_CCU4_ClearTrap(&PWM_CCU4_0);
    while(1);
    return 0;
}
```

Definition at line [563](#) of file [PWM\\_CCU4.c](#).

References [PWM\\_CCU4\\_HandleType::ccu4\\_slice\\_ptr](#).

## **DAVE\_APP\_VERSION\_t PWM\_CCU4\_GetAppVersion ( void )**

Retrieves the version of the PWM\_CCU4 APP.

### **Parameters**

**None**

**Returns**

DAVE\_APP\_VERSION\_t APP version information (major, minor and patch number)

**Description:**

The function can be used to check application software compatibility with a specific version of the APP.

\*Example Usage:

```
#include <DAVE.h>
int main(void)
{
    DAVE_APP_VERSION_t version;
    version = PWM_CCU4_GetAppVersion();
    while(1);
    return 0;
}
```

Definition at line [93](#) of file [PWM\\_CCU4.c](#).

```
bool PWM_CCU4_GetInterruptStatus ( PWM_CCU4_t *const
                                    XMC_CCU4_SLICE_IRQ_ID_
                                    )
```

Returns the interrupt status.

**Parameters**

**handle\_ptr** Pointer to PWM\_CCU4\_t structure containing APP parameters.  
**pwm\_interrupt** interrupt ID

**Returns**

bool

**Description:**

Returns true if the interrupt flag is set, else returns false.

Example Usage:

```
#include <DAVE.h>
int main(void)
{
    bool status;
    DAVE_Init();
    // Returns period match interrupt status.
    status =
        PWM_CCU4_GetInterruptStatus(&PWM_CCU4_0,XMC_CCU4_SLICE_
    // Returns channel compare match interrupt status.
    status =
        PWM_CCU4_GetInterruptStatus(&PWM_CCU4_0,XMC_CCU4_SLICE_
    while(1);
    return 0;
}
```

Definition at line [575](#) of file [PWM\\_CCU4.c](#).

References [PWM\\_CCU4\\_HandleTypeDef::ccu4\\_slice\\_ptr](#).

**bool PWM\_CCU4\_GetTimerStatus ( PWM\_CCU4\_t \*const handle\_p**

Returns the timer status.

#### Parameters

**handle\_ptr** Pointer to PWM\_CCU4\_t structure containing APP parameters.

#### Returns

XMC\_CCU4\_STATUS\_t

#### Description:

Returns true is the timer is running else returns false.

Example Usage:

```
#include <DAVE.h>
int main(void)
{
    bool status;
    DAVE_Init();
    status = PWM_CCU4_GetTimerStatus(&PWM_CCU4_0);
    while(1);
    return 0;
}
```

Definition at line [393](#) of file [PWM\\_CCU4.c](#).

References [PWM\\_CCU4\\_HandleTypeDef::ccu4\\_slice\\_ptr](#).

**uint32\_t PWM\_CCU4\_GetTimerValue ( [PWM\\_CCU4\\_t](#) \*const handle )**

Returns the timer value.

#### Parameters

**handle\_ptr** Pointer to PWM\_CCU4\_t structure containing APP parameters.

#### Returns

uint32\_t

#### Description:

Returns the timer value if the APP is initialized.

Example Usage:

```
#include <DAVE.h>
int main(void)
{
    uint32_t timer ;
    DAVE_Init();
```

```
timer = PWM_CCU4_GetTimerValue(&PWM_CCU4_0);
while(1);
return 0;
}
```

Definition at line **383** of file [PWM\\_CCU4.c](#).

References [PWM\\_CCU4\\_HandleTypeDef::ccu4\\_slice\\_ptr](#).

**PWM\_CCU4\_STATUS\_t PWM\_CCU4\_Init ( PWM\_CCU4\_t \*const handle\_ptr )**

Initializes the PWM\_CCU4 APP.

#### Parameters

**handle\_ptr** Pointer to PWM\_CCU4\_t structure containing APP parameters.

#### Returns

PWM\_CCU4\_STATUS\_t

#### Description:

Configures the CCU4 slice registers with the selected PWM\_CCU4 parameters. The slice is configured in PWM generation mode.

Example Usage:

```
#include <DAVE.h>
int main(void)
{
    DAVE_Init(); //PWM_CCU4_Init() is called by DAVE_Init().
    while(1);
    return 0;
}
```

Definition at line **105** of file [PWM\\_CCU4.c](#).

## References

PWM\_CCU4\_ConfigType::automatic\_shadow\_transfer,  
PWM\_CCU4\_ConfigType::cascaded\_shadow\_txfr\_enable,  
PWM\_CCU4\_ConfigType::ccu4\_cc4\_slice\_timer\_ptr,  
PWM\_CCU4\_HandleType::ccu4\_module\_ptr,  
PWM\_CCU4\_HandleType::ccu4\_slice\_ptr,  
PWM\_CCU4\_ConfigType::compare\_value,  
PWM\_CCU4\_HandleType::config\_ptr,  
PWM\_CCU4\_HandleType::dither\_shadow\_txfr\_msk,  
PWM\_CCU4\_HandleType::frequency\_tclk,  
PWM\_CCU4\_ConfigType::global\_ccu4\_handle,  
PWM\_CCU4\_ConfigType::gpio\_ch\_out\_config\_ptr,  
PWM\_CCU4\_ConfigType::gpio\_ch\_out\_enable,  
PWM\_CCU4\_ConfigType::gpio\_ch\_out\_pin,  
PWM\_CCU4\_ConfigType::gpio\_ch\_out\_ptr,  
PWM\_CCU4\_ConfigType::immediate\_write,  
PWM\_CCU4\_ConfigType::mcm\_shadow\_txfr\_mode,  
PWM\_CCU4\_ConfigType::period\_value, PWM\_CCU4\_Start(),  
PWM\_CCU4\_STATE\_INITIALIZED,  
PWM\_CCU4\_STATE\_UNINITIALIZED,  
PWM\_CCU4\_STATUS\_ALREADY\_INITIALIZED,  
PWM\_CCU4\_STATUS\_FAILURE,  
PWM\_CCU4\_STATUS\_SUCCESS,  
PWM\_CCU4\_ConfigType::shadow\_transfer\_mode,  
PWM\_CCU4\_HandleType::shadow\_txfr\_msk,  
PWM\_CCU4\_ConfigType::start\_control, and  
PWM\_CCU4\_HandleType::state.

```
void PWM_CCU4_SetDither( PWM_CCU4_t *const handle_ptr,
                           bool dither_period,
                           bool dither_comp,
                           uint8_t dither_value
                         )
```

Sets the dither value for period , duty or both.

## Parameters

- handle\_ptr** Pointer to PWM\_CCU4\_t structure containing APP parameters.
- dither\_period** apply dither to period
- dither\_comp** apply dither to compare
- dither\_value** dither value

## Returns

void

## Description:

Sets the dither value for period , duty or both.

dither\_value: is the dither value.

dither\_period: when true, dither is applied to period.

dither\_comp: when true, dither is applied to compare.

Example Usage:

```
#include <DAVE.h>
int main(void)
{
    DAVE_Init();
    PWM_CCU4_SetDither(&PWM_CCU4_0,(bool) true, (bool)true, 10);
    while(1);
    return 0;
}
```

Definition at line **550** of file **PWM\_CCU4.c**.

References [PWM\\_CCU4\\_HandleTypeDef::ccu4\\_module\\_ptr](#),  
[PWM\\_CCU4\\_HandleTypeDef::ccu4\\_slice\\_ptr](#), and  
[PWM\\_CCU4\\_HandleTypeDef::dither\\_shadow\\_txfr\\_msk](#).

**PWM\_CCU4\_STATUS\_t PWM\_CCU4\_SetDutyCycle (**[\*\*PWM\\_CCU4\\_t\*\*](#)  
[\*\*uint32\\_t\*\*](#)  
**)**

Sets the duty cycle of PWM.

## Parameters

**handle\_ptr** Pointer to PWM\_CCU4\_t structure containing APP parameters.  
**duty\_cycle** channel duty cycle uint32\_t

## Returns

PWM\_CCU4\_STATUS\_t

## Description:

Sets the PWM duty. The APP should not be in "PWM\_CCU4\_UNINITIALIZED" state. Duty is scaled by 100. The condition [duty < 100%] should be met. Returns PWM\_CCU4\_STATUS\_SUCCESS if operation update is success.

Example Usage:

```
#include <DAVE.h>
int main(void)
{
    PWM_CCU4_STATUS_t status;
    DAVE_Init();
    // sets the channel duty to 40%.
    status = PWM_CCU4_SetDutyCycle(&PWM_CCU4_0, 4000);
    while(1);
    return 0;
}
```

Definition at line 458 of file [PWM\\_CCU4.c](#).

References [PWM\\_CCU4\\_HandleTypeDef::ccu4\\_module\\_ptr](#),  
[PWM\\_CCU4\\_HandleTypeDef::ccu4\\_slice\\_ptr](#),  
[PWM\\_CCU4\\_STATE\\_UNINITIALIZED](#),  
[PWM\\_CCU4\\_STATUS\\_FAILURE](#),  
[PWM\\_CCU4\\_STATUS\\_SUCCESS](#),  
[PWM\\_CCU4\\_HandleTypeDef::shadow\\_txfr\\_msk](#),

`PWM_CCU4_HandleTypeDef::state`, and  
`PWM_CCU4_HandleTypeDef::sym_duty`.

```
PWM_CCU4_STATUS_t PWM_CCU4_SetFreq ( PWM_CCU4_t *cons
                                         uint32_t
                                         )
```

Sets the PWM frequency.

#### Parameters

**handle\_ptr** Pointer to `PWM_CCU4_t` structure containing APP parameters.  
**pwm\_freq\_hz** value in Hz (`uint32_t`)

#### Returns

`PWM_CCU4_STATUS_t`

#### Description:

Sets the PWM frequency of PWM generation. The APP should not be in "PWM\_CCU4\_UNINITIALIZED" state. Returns `PWM_CCU4_STATUS_SUCCESS` if frequency update is success.

Example Usage:

```
#include <DAVE.h>
int main(void)
{
    PWM_CCU4_STATUS_t status;
    DAVE_Init();
    status = PWM_CCU4_SetFreq(&PWM_CCU4_0, 100000);
    while(1);
    return 0;
}
```

Definition at line 405 of file `PWM_CCU4.c`.

References `PWM_CCU4_ConfigType::ccu4_cc4_slice_timer_ptr`,  
`PWM_CCU4_HandleTypeDef::ccu4_module_ptr`,  
`PWM_CCU4_HandleTypeDef::ccu4_slice_ptr`,  
`PWM_CCU4_HandleTypeDef::config_ptr`,  
`PWM_CCU4_HandleTypeDef::frequency_tclk`,  
`PWM_CCU4_STATE_UNINITIALIZED`,  
`PWM_CCU4_STATUS_FAILURE`,  
`PWM_CCU4_STATUS_SUCCESS`,  
`PWM_CCU4_HandleTypeDef::shadow_txfr_msk`,  
`PWM_CCU4_HandleTypeDef::state`, and  
`PWM_CCU4_HandleTypeDef::sym_duty`.

```
PWM_CCU4_STATUS_t PWM_CCU4_SetFreqAndDutyCycle ( PWM_CCU4_CONFIG_t config, uint32_t freq, uint32_t dutyCycle )
```

Sets the frequency duty cycle of PWM.

## Parameters

**handle\_ptr** Pointer to PWM\_CCU4\_t structure containing APP parameters.  
**pwm\_freq\_hz** value in Hz (uint32\_t)  
**duty** channel duty

## Returns

## PWM CCU4 STATUS t

## Description:

Sets the frequency and duty of PWM. The APP should not be in "PWM\_CCU4\_UNINITIALIZED" state. Duty is scaled by 100. The condition [duty < 100%] should be met. Returns PWM\_CCU4\_STATUS\_SUCCESS if operation update is success.

## Example Usage:

```
#include <DAVE.h>
int main(void)
{
    PWM_CCU4_STATUS_t status;
    DAVE_Init();
    // Sets freq = 100000, channel duty = 40%
    status = PWM_CCU4_SetFreqAndDutyCycle(&PWM_CCU4_0,
    100000, 4000);
    while(1);
    return 0;
}
```

Definition at line [495](#) of file [PWM\\_CCU4.c](#).

References [PWM\\_CCU4\\_ConfigType::ccu4\\_cc4\\_slice\\_timer\\_ptr](#),  
[PWM\\_CCU4\\_HandleType::ccu4\\_module\\_ptr](#),  
[PWM\\_CCU4\\_HandleType::ccu4\\_slice\\_ptr](#),  
[PWM\\_CCU4\\_HandleType::config\\_ptr](#),  
[PWM\\_CCU4\\_HandleType::frequency\\_tclk](#),  
[PWM\\_CCU4\\_STATE\\_UNINITIALIZED](#),  
[PWM\\_CCU4\\_STATUS\\_FAILURE](#),  
[PWM\\_CCU4\\_STATUS\\_SUCCESS](#),  
[PWM\\_CCU4\\_HandleType::shadow\\_txfr\\_msk](#),  
[PWM\\_CCU4\\_HandleType::state](#), and  
[PWM\\_CCU4\\_HandleType::sym\\_duty](#).

## PWM\_CCU4\_STATUS\_t PWM\_CCU4\_Start ( PWM\_CCU4\_t \*const h

Start the selected CCU4 slice.

### Parameters

**handle\_ptr** Pointer to PWM\_CCU4\_t structure containing APP parameters.

### Returns

PWM\_CCU4\_STATUS\_t

### Description:

Starts the selected CCU4 slice for PWM generation. Returns PWM\_CCU4\_STATUS\_SUCCESS if the PWM\_CCU4 APP state is in "PWM\_CCU4\_STATE\_INITIALIZED" or "PWM\_CCU4\_STOPPED" else returns PWM\_CCU4\_STATUS\_FAILURE.

**PWM\_CCU4\_Start()** is needed to be called if "Start during initialization" is unchecked to start PWM generation, else its called by DAVE\_Init();

### Example Usage:

```
#include <DAVE.h>
int main(void)
{
    DAVE_Init();
    //This API needs to be called if "Start during initialization" is
    //unchecked
    PWM_CCU4_Start(&PWM_CCU4_0);
    while(1);
    return 0;
}
```

Definition at line 339 of file **PWM\_CCU4.c**.

References **PWM\_CCU4\_HandleType::ccu4\_module\_ptr**,  
**PWM\_CCU4\_HandleType::ccu4\_slice\_ptr**,  
**PWM\_CCU4\_HandleType::config\_ptr**,  
**PWM\_CCU4\_ConfigType::ext\_start\_event**,  
**PWM\_CCU4\_STATE\_INITIALIZED**,  
**PWM\_CCU4\_STATE\_RUNNING**, **PWM\_CCU4\_STATE\_STOPPED**,  
**PWM\_CCU4\_STATUS\_FAILURE**,  
**PWM\_CCU4\_STATUS\_SUCCESS**,  
**PWM\_CCU4\_HandleType::slice\_number**, and  
**PWM\_CCU4\_HandleType::state**.

Referenced by **PWM\_CCU4\_Init()**.

**PWM\_CCU4\_STATUS\_t PWM\_CCU4\_Stop ( PWM\_CCU4\_t \*const h**

Stop the selected CCU4 slice.

### Parameters

**handle\_ptr** Pointer to PWM\_CCU4\_t structure containing APP parameters.

### Returns

PWM\_CCU4\_STATUS\_t

### Description:

Stops the selected CCU4 slice from PWM generation. Returns PWM\_CCU4\_STATUS\_SUCCESS if the PWM\_CCU4 APP state is not "PWM\_CCU4\_STATE\_UNINITIALIZED" else returns PWM\_CCU4\_STATUS\_FAILURE.

Example Usage:

```
#include <DAVE.h>
int main(void)
{
    DAVE_Init();
    PWM_CCU4_Stop(&PWM_CCU4_0);
    while(1);
    return 0;
}
```

Definition at line 363 of file **PWM\_CCU4.c**.

References **PWM\_CCU4\_HandleTypeDef::ccu4\_slice\_ptr**,  
**PWM\_CCU4\_STATE\_STOPPED**,  
**PWM\_CCU4\_STATE\_UNINITIALIZED**,  
**PWM\_CCU4\_STATUS\_FAILURE**,  
**PWM\_CCU4\_STATUS\_SUCCESS**, and  
**PWM\_CCU4\_HandleTypeDef::state**.

