

# TIMER

Home

## Apps

Here is a list of all modules:

- License Terms and Copyright Information
  - Abbreviations and Definitions
  - Overview
  - Architecture Description
  - APP Configuration Parameters
  - Enumerations
  - Data structures
  - Methods
  - Usage
  - Release History
- 
- 
-

# TIMER

Home

## License Terms and Copyright Information

### License Terms and Copyright Information

Copyright (c) 2015, Infineon Technologies AG All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution. Neither the name of the copyright holders nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT

(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

To improve the quality of the software, users are encouraged to share modifications, enhancements or bug fixes with Infineon Technologies AG ([dave@infineon.com](mailto:dave@infineon.com)).

---

# TIMER

Home

## Abbreviations and Definitions

### Abbreviations and Definitions

Abbreviations:	
DAVE™	Digital Application Virtual Engineer
APP	DAVE™ Application
APP	DAVE Application
API	Application Programming Interface
GUI	Graphical User Interface
MCU	Microcontroller Unit
SW	Software
HW	Hardware
LLD	Low Level Driver
I/O	Input/Output
CCU	Capture Compare Unit

Definitions:	
Singleton	Only single instance of the APP is permitted
Sharable	Resource sharing with other APPs is permitted
initProvider	Provides the initialization routine
Physical connectivity	Hardware inter/intra peripheral (constant) signal connection
Conditional connectivity	Constrained hardware inter/intra peripheral signal connection
Aggregation	Indicates consumption of low level (dependent) DAVE™ APPs



# TIMER

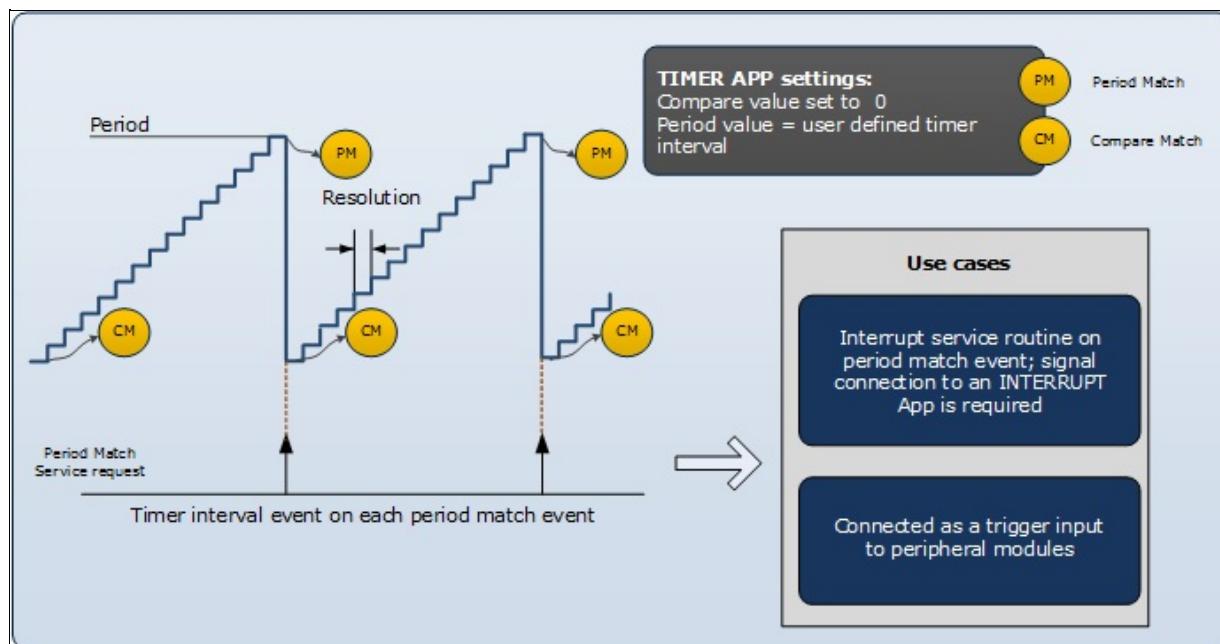
Home

## Overview

### Overview

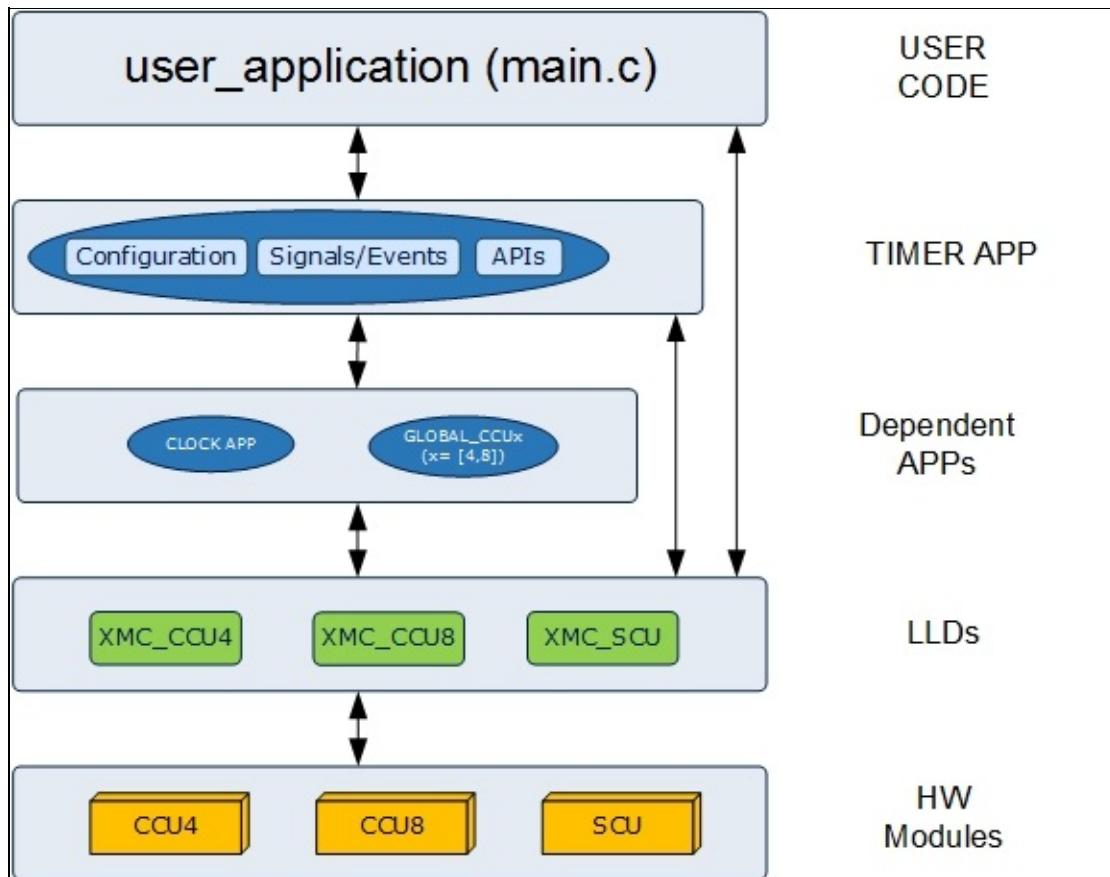
The **TIMER APP** provides accurate hardware timer using one timer slice of CCU4 or CCU8 peripheral. And provides provide the functionality below:

1. Continuous event generation for a given time interval.
2. Generates more accurate delay using Hardware timer.
3. Sets the desired time interval at runtime(in terms of micro-seconds).
4. Connection of timer interval event to other peripherals as a trigger source or for interrupt generation(using the DAVE HW Signal Connections)



**Figure 1 : Overview of TIMER APP**

The figure above shows the functional overview and use cases of the **TIMER APP**.



**Figure 2 : Hardware and Software connectivity of TIMER APP**

The figure above shows the layered architecture of the **TIMER APP** in DAVE™. XMC controllers provide the CCU4 or CCU8 module. Each module is having four 16-bit timers. The LLD layer provides abstraction for these hardware modules. The **TIMER APP** uses CCU4 or CCU8 and GPIO LLDs and other dependent APPs like GLOBAL\_CCUx(x = [4,8]) and CLOCK\_XMCx(x = [1,4]) for the functionality.

## Supported Devices

The APP supports below devices:

1. XMC4800/XMC4700 Series

2. XMC4500 Series
3. XMC4300 Series
4. XMC4400 Series
5. XMC4200 / XMC4100 Series
6. XMC1400 Series
7. XMC1300 Series
8. XMC1200 Series
9. XMC1100 Series

## **References**

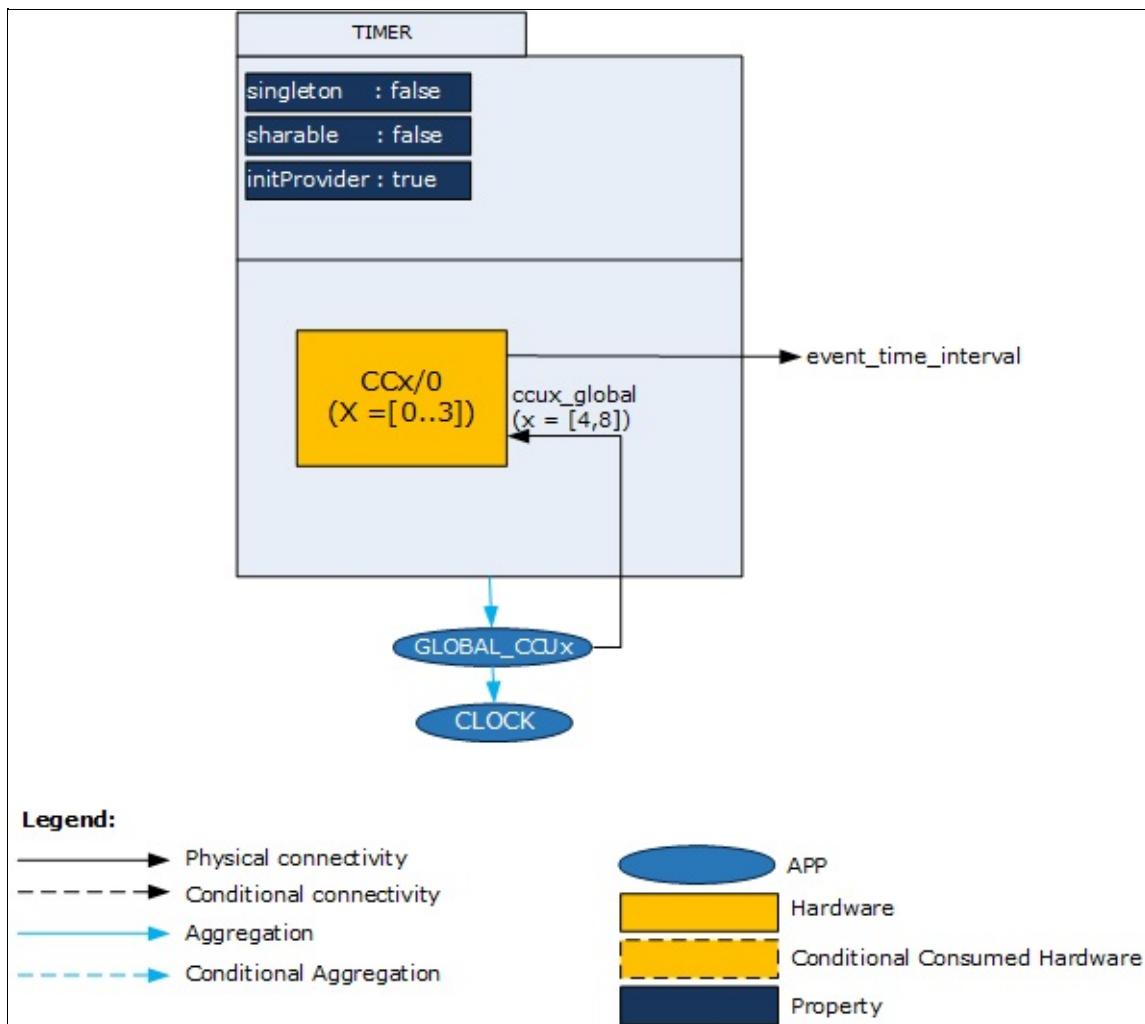
1. XMC4800 / XMC4700 Reference Manual
  2. XMC4500 Reference Manual
  3. XMC4400 Reference Manual
  4. XMC4300 Reference Manual
  5. XMC4200 / XMC4100 Reference Manual
  6. XMC1400 Reference Manual
  7. XMC1300 Reference Manual
  8. XMC1200 Reference Manual
  9. XMC1100 Reference Manual
- 
-

# TIMER

Home

## Architecture Description

### Architecture Description



**Figure 1 :** Architecture of **TIMER** APP

The figure above represents the internal software architecture of the **TIMER** APP. A **TIMER** APP instance exists in a DAVE™ project with fixed attributes as shown. Each instance of this APP configures one

CCU slice timer in the MCU. This in addition requires the consumption of the GLOBAL\_CCUX (x = [4,8]) and CLOCK APPS for its configuration and functioning. The **TIMER** APP also provides output signal for inter-peripheral connections.

An instantiated APP (after code generation) generates a specific data structure with the GUI configuration. The name of this data structure can be modified by changing the APP instance label (e.g. change label from default TIMER\_0 to GAME\_DELAY).

### Signals:

The following table presents the signals provided by the APP for connection. The signal gives the flexibility to configure and extend the connectivity to other APPs.

**Table 1:** APP I/O signals

Signal Name	Input/Output Availability	Description
event_time_interval	Output Conditional	<p>Time interval event signal: Upon enabling "Time interval event" in GUI, signal is populated.</p> <ul style="list-style-type: none"><li>• Connect the signal to an INTERRUPT APP to generate the interrupt function each time interval.</li><li>• Connect the signal to another APP to receive the time interval events.</li></ul>

other  
peripherals:  
ADC, ERL  
DMA, DAC  
POSIF) as  
trigger sou

### **APPs Consumed:**

The following table presents the APPs consumed to support the functionality:

**Table 2:** APPs Consumed

APP Name	Consumption
GLOBAL_CCU4	Conditionally consumed if "CCU4" is selected in UI field "Select timer"

**GLOBAL\_CCU8** Conditionally consumed  
if "CCU8" is selected in UI field "Select timer"

# TIMER

Home

## APP Configuration Parameters

### App Configuration Parameters

General Settings Event Settings

Select timer module: CCU4

Timer Settings

Time interval [usec]: 1.0

Start after initialization

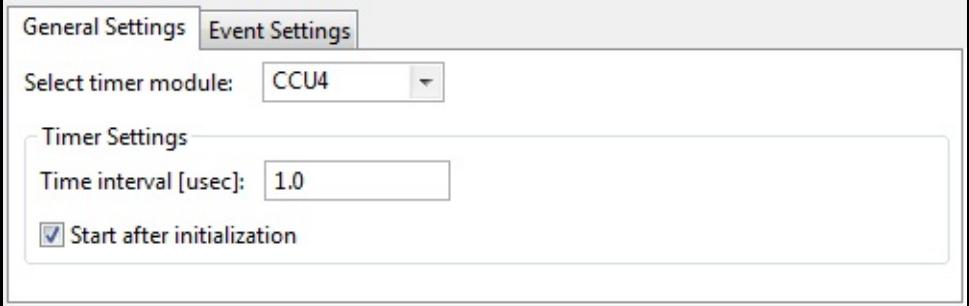


Figure 1: General Settings

General Settings Event Settings

Time interval event

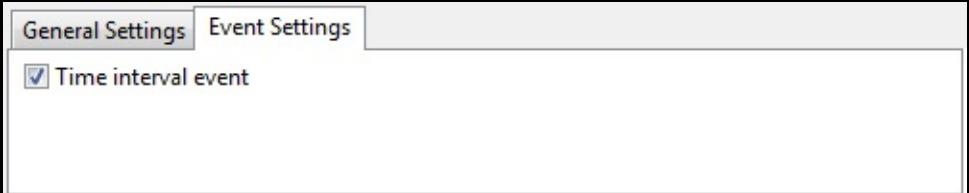


Figure 2: Event Settings



# TIMER

Home

## Enumerations

enum **TIMER\_MODULE** {  
    **TIMER\_MODULE\_CCU4** = 0U,  
    **TIMER\_MODULE\_CCU8** }  
The type identifies the CCU4 or  
CCU8 timer selected. [More...](#)

enum **TIMER\_STATUS** {  
    **TIMER\_STATUS\_SUCCESS** = 0U,  
    **TIMER\_STATUS\_FAILURE** }  
status of the **TIMER** APP [More...](#)

typedef enum **TIMER\_MODULE** **TIMER\_MODULE\_t**  
The type identifies the CCU4 or  
CCU8 timer selected.

typedef enum **TIMER\_STATUS** **TIMER\_STATUS\_t**  
status of the **TIMER** APP

## Enumeration Type Documentation

### enum TIMER\_MODULE

The type identifies the CCU4 or CCU8 timer selected.

#### Enumerator:

*TIMER\_MODULE\_CCU4* CCU4 is selected

*TIMER\_MODULE\_CCU8* CCU8 is selected

Definition at line [88](#) of file [TIMER.h](#).

### enum TIMER\_STATUS

status of the [TIMER APP](#)

#### Enumerator:

*TIMER\_STATUS\_SUCCESS* Status success

*TIMER\_STATUS\_FAILURE* Status failure

Definition at line [97](#) of file [TIMER.h](#).

# TIMER

Home

Data Structures

## Data structures

## Data Structures

---

struct **TIMER**

Initialization parameters of the **TIMER** APP.

[More...](#)

---

typedef struct **TIMER** **TIMER\_t**

Initialization parameters of the **TIMER** APP.

---



# TIMER

Home

## Methods

DAVE\_APP\_VERSION\_t **TIMER\_GetAppVersion** (void)

Get **TIMER** APP version.

**TIMER\_STATUS\_t** **TIMER\_Init** (**TIMER\_t** \*const handle\_ptr)

Initializes a **TIMER** with generated configuration.

**TIMER\_STATUS\_t** **TIMER\_Start** (**TIMER\_t** \*const handle\_ptr)

Starts the timer if the initialization of the APP is successful.

**TIMER\_STATUS\_t** **TIMER\_Stop** (**TIMER\_t** \*const handle\_ptr)

Stops the **TIMER**, if it is running.

uint32\_t **TIMER\_GetTime** (**TIMER\_t** \*const handle\_ptr)

Returns the current time in micro seconds by scaling with 100.

**TIMER\_STATUS\_t** **TIMER\_Clear** (**TIMER\_t** \*const handle\_ptr)

Clears the timer register.

bool **TIMER\_GetTimerStatus** (**TIMER\_t** \*const handle\_ptr)

Returns the running state of the timer.

**TIMER\_STATUS\_t** **TIMER\_SetTimeInterval** (**TIMER\_t** \*const handle\_ptr, uint32\_t time\_interval)

Set the new time interval for the event generation, by checking with the supported range.

bool **TIMER\_GetInterruptStatus** (**TIMER\_t** \*const handle\_ptr)

Indicates the occurrence of time interval event.

```
void TIMER_ClearEvent (TIMER_t *const  
handle_ptr)  
Clears the period match interrupt status of  
the given timer.
```

## Methods

---

## Function Documentation

**TIMER\_STATUS\_t TIMER\_Clear ( TIMER\_t \*const handle\_ptr )**

Clears the timer register.

**Parameters:**

**handle\_ptr** pointer to the **TIMER APP configuration**.

**Returns:**

**TIMER\_STATUS\_t**

**TIMER\_STATUS\_SUCCESS** : if clear is successful

**TIMER\_STATUS\_FAILURE** : if timer is not initialized and clear is requested

**Description:**

**TIMER\_Clear()** clears the timer register so that next cycle starts from reset value.

Example Usage:

```
#include <DAVE.h>
int main(void)
{
    DAVE_STATUS_t init_status;
    TIMER_STATUS_t timer_status;
    init_status = DAVE_Init();      // TIMER_Init(&TIMER_0) will be called from DAVE_Init()

    if(init_status == DAVE_STATUS_SUCCESS)
    {
        timer_status = TIMER_Start(&TIMER_0);
    }

    if (TIMER_GetTimerStatus(&TIMER_0))
```

```
{  
    timer_status = TIMER_Stop(&TIMER_0);  
}  
  
timer_status = TIMER_Clear(&TIMER_0);  
  
while(1)  
{  
}  
return 1;  
}
```

Definition at line [415](#) of file [TIMER.c](#).

References [TIMER::timer\\_module](#), [TIMER\\_MODULE\\_CCU4](#), [TIMER\\_MODULE\\_CCU8](#), [TIMER\\_STATUS\\_FAILURE](#), and [TIMER\\_STATUS\\_SUCCESS](#).

### **void [TIMER\\_ClearEvent](#)( [TIMER\\_t](#) \*const handle\_ptr )**

Clears the period match interrupt status of the given timer.

**Parameters:**

**handle\_ptr** pointer to the [TIMER APP configuration](#).

**Returns:**

None

**Description:**

For each occurrence of the time interval event, it has to be cleared through software only. So next event is considered as new.

### Example Usage:

```
#include <DAVE.h>
int main(void)
{
    DAVE_STATUS_t status;

    status = DAVE_Init(); // Initialization of DAVE APPs

    while(1U)
    {
    }
    return 1;
}

void Timetick_Handler(void)
{
    TIMER_ClearEvent(&TIMER_0);
}
```

Definition at line [350](#) of file [TIMER.c](#).

References [TIMER::timer\\_module](#), [TIMER\\_MODULE\\_CCU4](#), and [TIMER\\_MODULE\\_CCU8](#).

### **DAVE\_APP\_VERSION\_t TIMER\_GetAppVersion ( void )**

Get [TIMER](#) APP version.

#### **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_STATUS_t status;
    DAVE_APP_VERSION_t app_version;

    status = DAVE_Init();           // TIMER_Init() is called from DAVE_Init()

    app_version = TIMER_GetAppVersion();

    if (app_version.major != 4U)
    {
        // Probably, not the right version.
    }

    while(1U)
    {
    }
    return 1;
}
```

Definition at line **97** of file **TIMER.c**.

**bool TIMER\_GetInterruptStatus ( TIMER\_t \*const handle\_ptr )**

Indicates the occurrence of time interval event.

**Parameters:**

**handle\_ptr** pointer to the **TIMER APP** configuration.

**Returns:**

bool  
true : if event set  
false : if event is not set

**Description:**

The status returned, can be utilized to generate the delay function.

Example Usage:

```
#include <DAVE.h>
#define TIMER_DELAY_MUL_FACTOR 100000U // Converts micro seconds to milli seconds with multiplication factor for
                                         // TIMER_
GetInterruptStatus().
void TIMER_Delay(uint32_t);
int main(void)
{
    DAVE_STATUS_t init_status;
    TIMER_STATUS_t status;
    uint32_t delay_val; // delay value in terms milli seconds

    init_status = DAVE_Init();      // TIMER_Init(&TIMER_0) will be called from DAVE_Init()

    TIMER_ClearEvent(&TIMER_0);

    if(init_status == DAVE_STATUS_SUCCESS)
    {
        delay_val = 1000; // 1000 milli seconds

        TIMER_Delay(delay_val);
```

```
}

while(1)
{
}

return 1;
}

void TIMER_Delay(uint32_t delay_val)
{
    uint32_t delay_cnt;

    delay_cnt = delay_val * TIMER_DELAY_MUL_FACTOR
;

    TIMER_SetTimeInterval(&TIMER_0,delay_cnt);

    TIMER_Start(&TIMER_0);

    while(!TIMER_GetInterruptStatus(&TIMER_0));

    TIMER_Stop(&TIMER_0);
}
```

Definition at line [324](#) of file [TIMER.c](#).

References [TIMER::timer\\_module](#), [TIMER\\_MODULE\\_CCU4](#), and [TIMER\\_MODULE\\_CCU8](#).

---

**uint32\_t [TIMER\\_GetTime](#) ([TIMER\\_t](#) \*const handle\_ptr )**

Returns the current time in micro seconds by scaling with 100.

**Parameters:**

**handle\_ptr** pointer to the **TIMER APP** configuration.

**Returns:**

uint32\_t  
time in microseconds

**Description:**

By using prescaler and frequency and timer register value, this API calculates the current time in micro seconds. Then the value is scaled with 100, before returning.

**Example Usage:**

```
#include <DAVE.h>
int main(void)
{
    DAVE_STATUS_t init_status;
    TIMER_STATUS_t timer_status;
    uint32_t elapsed_time;
    init_status = DAVE_Init();      // TIMER_Init(&TIMER_0) will be called from DAVE_Init()

    if(init_status == DAVE_STATUS_SUCCESS)
    {
        timer_status = TIMER_Start(&TIMER_0);
    }

    timer_status = TIMER_Stop(&TIMER_0);

    elapsed_time = TIMER_GetTime(&TIMER_0);

    while(1)
    {
    }
    return 1;
}
```

Definition at line 374 of file **TIMER.c**.

References **TIMER::timer\_module**, **TIMER\_MODULE\_CCU4**, and **TIMER\_MODULE\_CCU8**.

## **bool TIMER\_GetTimerStatus ( TIMER\_t \*const handle\_ptr )**

Returns the running state of the timer.

### **Parameters:**

**handle\_ptr** pointer to the **TIMER APP** configuration.

### **Returns:**

**bool**  
true : if the timer is running  
false : if the timer is not running

### **Description:**

**TIMER\_GetTimerStatus()** reads the run bit of the timer to indicate the actual state of the **TIMER**.

Example Usage:

```
#include <DAVE.h>
int main(void)
{
    DAVE_STATUS_t init_status;
    TIMER_STATUS_t timer_status;
    init_status = DAVE_Init();      // TIMER_Init(&TIMER_0) will be called from DAVE_Init()

    if(init_status == DAVE_STATUS_SUCCESS)
    {
        timer_status = TIMER_Start(&TIMER_0);
```

```
}

if (TIMER_GetTimerStatus(&TIMER_0))
{
    while(TIMER_GetTimerStatus(&TIMER_0));

    timer_status = TIMER_Stop(&TIMER_0);
}
while(1)
{
}
return 1;
}
```

Definition at line [219](#) of file **TIMER.c**.

References **TIMER::timer\_module**, **TIMER\_MODULE\_CCU4**, and **TIMER\_MODULE\_CCU8**.

Referenced by **TIMER\_SetTimeInterval()**, and **TIMER\_Stop()**.

## **TIMER\_STATUS\_t TIMER\_Init( TIMER\_t \*const handle\_ptr )**

Initializes a **TIMER** with generated configuration.

### **Parameters:**

**handle\_ptr** pointer to the **TIMER APP** configuration.

### **Returns:**

**TIMER\_STATUS\_t**

**TIMER\_STATUS\_SUCCESS** : if initialization is successful

**TIMER\_STATUS\_FAILURE** : if initialization is failed

### **Description:**

- Enable the clock for the slice and invoke the LLD API with generated configuration handle.
- Load the Period, Compare and Prescaler shadow registers with the generated values and enable the shadow transfer request. This loads the values into the actual registers and start the **TIMER** based on the configuration.
- If "Start after initialization" is not enabled, **TIMER\_Start()** can be invoked to start the timer.

Example Usage:

```
#include <DAVE.h>
int main(void)
{
    DAVE_STATUS_t init_status;
    init_status = DAVE_Init();      // TIMER_Init(&TI
    MER_0) will be called from DAVE_Init()

    while(1)
    {
    }
    return 1;
}
```

Definition at line **112** of file **TIMER.c**.

References **TIMER::timer\_module**, **TIMER\_MODULE\_CCU4**, **TIMER\_MODULE\_CCU8**, and **TIMER\_STATUS\_SUCCESS**.

---

**TIMER\_STATUS\_t TIMER\_SetTimeInterval ( **TIMER\_t** \*const handle  
  uint32\_t time\_ir  
  )**

---

Set the new time interval for the event generation, by checking with the supported range.

**Parameters:**

**handle\_ptr** pointer to the **TIMER** APP configuration.

**time\_interval** new time interval value in micro seconds.

**Returns:**

**TIMER\_STATUS\_t**

**TIMER\_STATUS\_SUCCESS** : Setting new time interval value is successful

**TIMER\_STATUS\_FAILURE** : New time value is not in range of supported time value

Timer is in running condition

**Description:**

Based on the timer interval, prescaler value is calculated for the CCU timer. By using this prescaler and time interval values Period value is calculated. The period value is updated into the shadow register and shadow transfer request is enabled. Timer has to be stopped before updating the time interval.

**Note:**

Input time interval value has to be scaled by 100 to the actual required value.

e.g. : required timer interval value = 30.45 micro seconds

Input value to the API =  $30.45 * 100 = 3045$

**Example Usage:**

```
#include <DAVE.h>
#include <xmc_gpio.h> // GPIO
LLD header, this contains the interface for Port
functionality
#define TIMER_GPIO_PORT XMC_GPIO_PORT0 // PORT0
```

```

Address
#define TIMER_GPIO_PIN  0U                                // Pin number
#define TIMER_500MS 500000*1000

volatile uint32_t count = 0U;                          // count
variable to change the time tick interval
uint32_t shadow_transfer_msk;                      // This is
is to generate the slice specific shadow transfer
mask

const XMC_GPIO_CONFIG_t GPIO_0_config =
{
    .mode = XMC_GPIO_MODE_OUTPUT_PUSH_PULL,
    .output_level = XMC_GPIO_OUTPUT_LEVEL_LOW,
};

int main(void)
{
    DAVE_STATUS_t status;

    XMC_GPIO_Init(TIMER_GPIO_PORT, TIMER_GPIO_PIN,
&GPIO_0_config);

    status = DAVE_Init();                                // Initialization of DAVE APPs

    while(1U)
    {
    }
    return 1;
}

void Timetick_Handler(void)
{
    count++;
}

```

```
TIMER_ClearEvent(&TIMER_0);

XMC_GPIO_ToggleOutput(TIMER_GPIO_PORT, TIMER_GPIO_PIN);

if(count > 10)
{
    count = 0U;
    TIMER_Stop(&TIMER_0);
    status = TIMER_SetTimeInterval(&TIMER_0, TIME_R_500MS);
    if (status == TIMER_STATUS_SUCCESS)
    {
        TIMER_Start(&TIMER_0);
    }
}
```

Definition at line 250 of file **TIMER.c**.

References **TIMER::period\_value**, **TIMER\_GetTimerStatus()**, **TIMER::timer\_max\_value\_us**, **TIMER::timer\_min\_value\_us**, **TIMER::timer\_module**, **TIMER\_MODULE\_CCU4**, **TIMER\_MODULE\_CCU8**, **TIMER\_STATUS\_FAILURE**, and **TIMER\_STATUS\_SUCCESS**.

### **TIMER\_STATUS\_t TIMER\_Start ( **TIMER\_t** \*const handle\_ptr )**

Starts the timer if the initialization of the APP is successful.

#### **Parameters:**

**handle\_ptr** pointer to the **TIMER** APP configuration.

#### **Returns:**

`TIMER_STATUS_t`  
`TIMER_STATUS_SUCCESS` : if timer start is successful  
`TIMER_STATUS_FAILURE` : if timer start is failed

### Description:

If "Start after initialization" is not enabled, `TIMER_Start()` can be invoked to start the timer. `TIMER_Stop()` can be used to stop the Timer. No need to reconfigure the timer to start again.

### Example Usage:

```
#include <DAVE.h>
int main(void)
{
    DAVE_STATUS_t init_status;
    TIMER_STATUS_t timer_status;
    init_status = DAVE_Init();      // TIMER_Init(&TI
MER_0) will be called from DAVE_Init()

    if(init_status == DAVE_STATUS_SUCCESS)
    {
        timer_status = TIMER_Start(&TIMER_0);
    }
    while(1)
    {
    }
    return 1;
}
```

Definition at line [145](#) of file `TIMER.c`.

References `TIMER::timer_module`, `TIMER_MODULE_CCU4`, `TIMER_MODULE_CCU8`, `TIMER_STATUS_FAILURE`, and `TIMER_STATUS_SUCCESS`.

## **TIMER\_STATUS\_t TIMER\_Stop ( TIMER\_t \*const handle\_ptr )**

Stops the **TIMER**, if it is running.

### **Parameters:**

**handle\_ptr** pointer to the **TIMER APP configuration**.

### **Returns:**

**TIMER\_STATUS\_t**

**TIMER\_STATUS\_SUCCESS** : if timer is running and stop is successful

**TIMER\_STATUS\_FAILURE** : if timer is in idle state, and stop is called

### **Description:**

Clears the Timer run bit to stop. No further event is generated.

### **Example Usage:**

```
#include <DAVE.h>
int main(void)
{
    DAVE_STATUS_t init_status;
    TIMER_STATUS_t timer_status;
    init_status = DAVE_Init();      // TIMER_Init(&TI
    MER_0) will be called from DAVE_Init()

    if(init_status == DAVE_STATUS_SUCCESS)
    {
        timer_status = TIMER_Start(&TIMER_0);
    }

    if (timer_status == TIMER_STATUS_SUCCESS)
    {
```

```
    while(TIMER_GetInterruptStatus(&TIMER_0));

    timer_status = TIMER_Stop(&TIMER_0);
}
while(1)
{
}
return 1;
}
```

Definition at line [182](#) of file **TIMER.c**.

References [TIMER\\_GetTimerStatus\(\)](#), [TIMER::timer\\_module](#), [TIMER\\_MODULE\\_CCU4](#), [TIMER\\_MODULE\\_CCU8](#), [TIMER\\_STATUS\\_FAILURE](#), and [TIMER\\_STATUS\\_SUCCESS](#).

---

# TIMER

Home

## Usage

### Usage

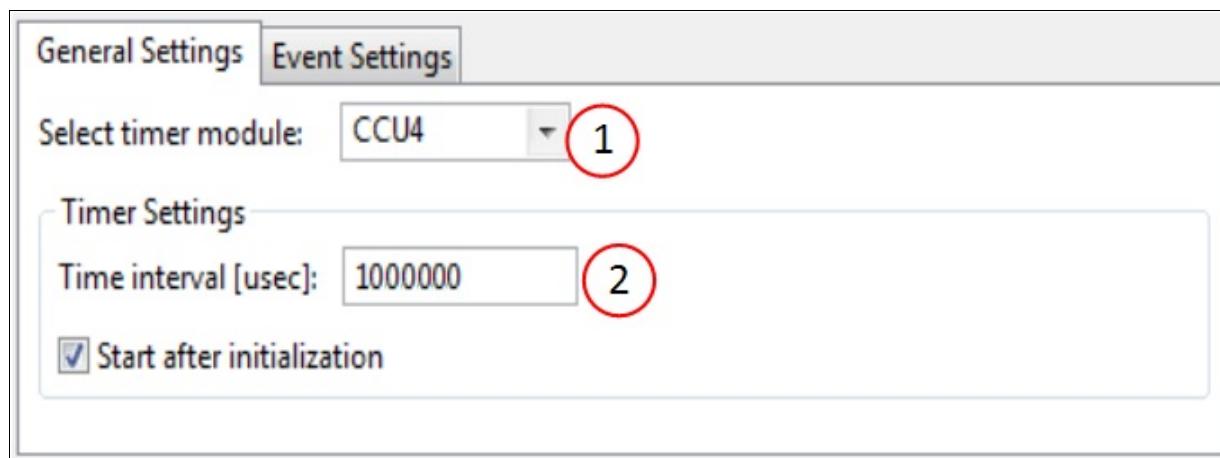
This example demonstrates the generation of events for the defined time intervals using **TIMER APP** considering CCU4 HW timer module. Initially the APP is configured to generate the events for every one second. And after getting the 10 events(10 seconds), time interval is updated in ISR with 0.5 second. Generation of events is indicated by toggling an LED.

### Instantiate the required APPs

Drag an instance of **TIMER**, **INTERRUPT** and **DIGITAL\_IO** APPs. Update the fields in the GUI of these APPs with the following configuration.

### Configure the APPs

#### TIMER APP:



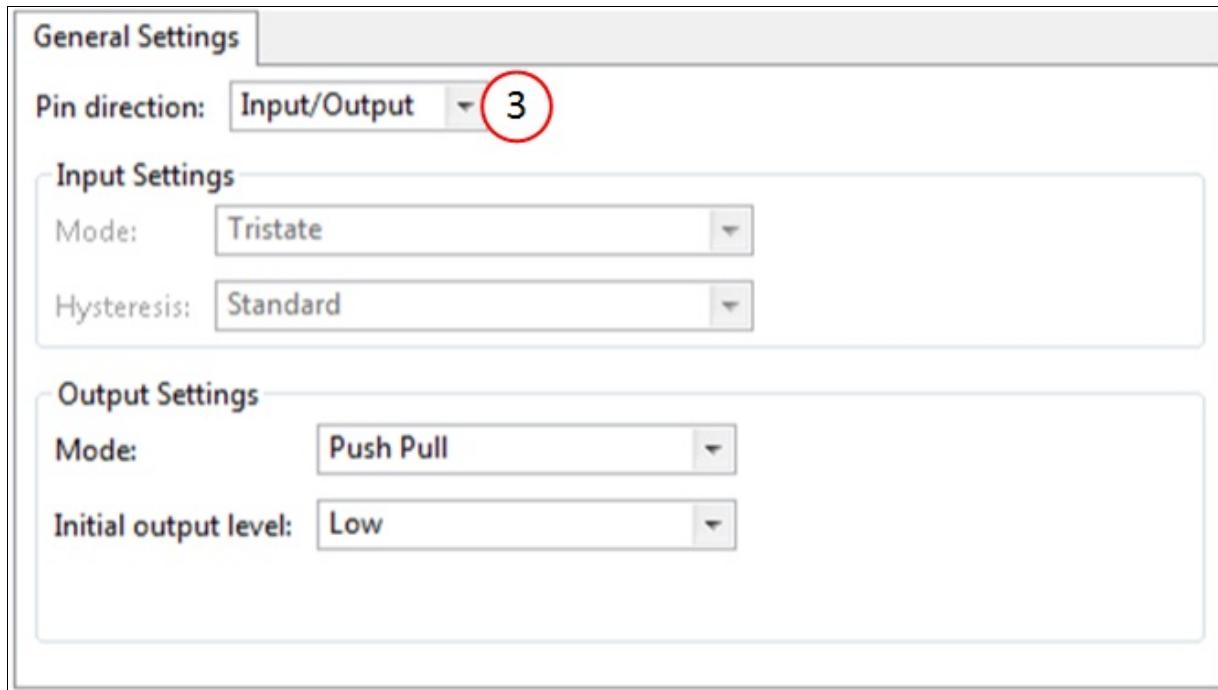
1. Select CCU4 module

**Note:** If all the timers in the CCU4 module are consumed,

manually switch one of the existing instances of APPs to CCU8 and try again.

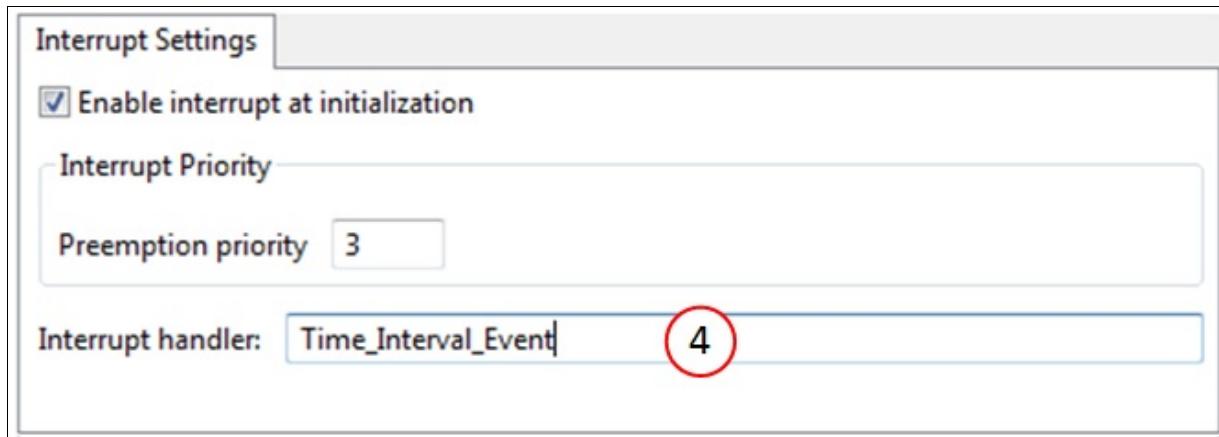
2. Set the time interval to 1sec
- Time interval [usec]: 1000000

#### DIGITAL\_IO APP:



3. Set pin direction to output by choosing - Pin direction : Input/Output

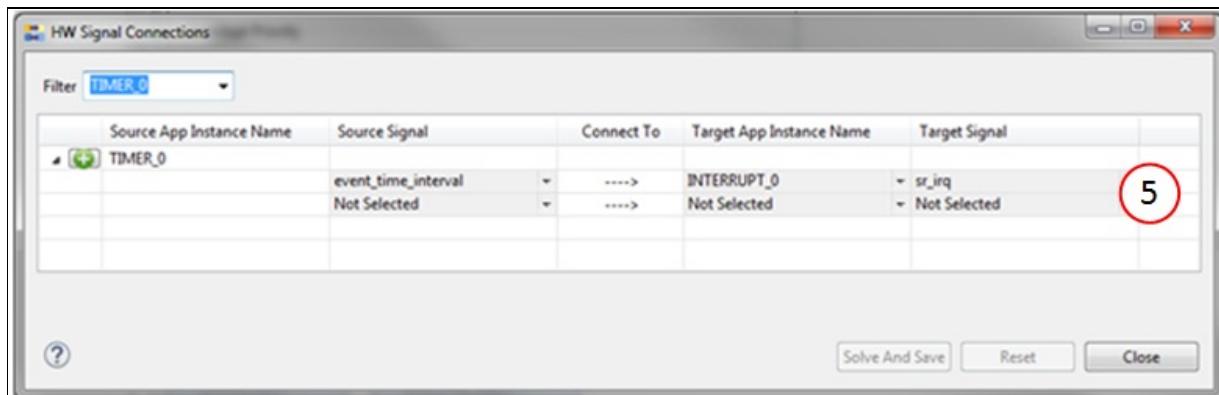
#### INTERRUPT APP:



4. Enter an appropriate ISR handle name for the Time interval event.  
e.g. Time\_Interval\_Event  
**Note :** This ISR shall be defined by user application (main.c)

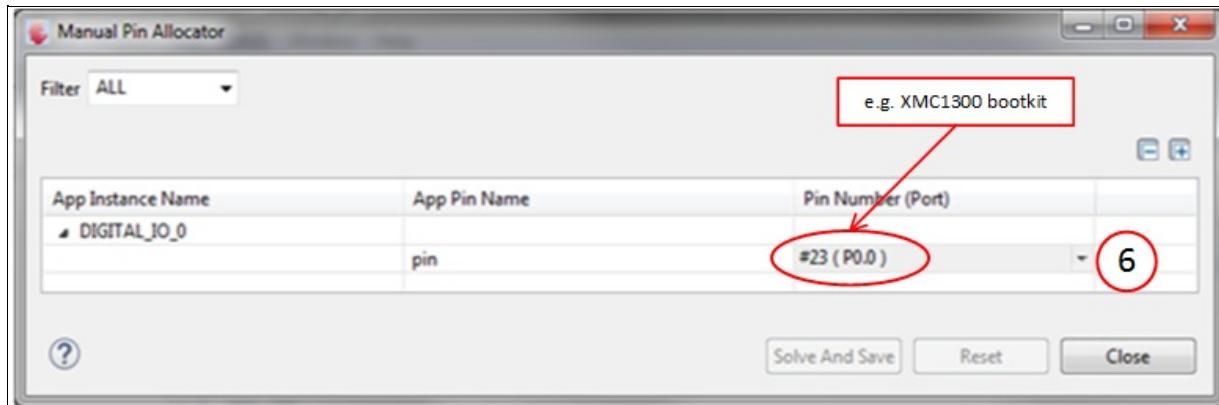
## Signal Connection

Establish a HW signal connection between the **TIMER** and the **INTERRUPT APP** to ensure **TIMER** events goes to **INTERRUPT**.



5. Connect TIMER\_0/event\_time\_interval -> INTERRUPT\_0/sr\_irq to ensure assigning ISR node to the time\_interval event.

## Manual pin allocation



## 6. Select the pin to be toggled (on-board LED)

**Note:** The pin number is specific to the development board chosen to run this example. The pin shown in the image above may not be available on every XMC boot kit. Ensure that a proper pin is selected according to the board.

## Generate code

Files are generated here: `<project\_name>/Dave/Generated/' ('project\_name' is the name chosen by the user during project creation). APP instance definitions and APIs are generated only after code generation.

- **Note:** Code must be explicitly generated for every change in the GUI configuration.  
**Important:** Any manual modification to APP specific files will be overwritten by a subsequent code generation operation.

## Sample Application (main.c)

```
#include <DAVE.h>

uint32_t event_count;

int main(void)
{
```

```

DAVE_STATUS_t status;

status = DAVE_Init();
if (status == DAVE_STATUS_FAILURE)
{
    XMC_DEBUG(("DAVE Apps initialization failed with status %d\n", status));
    while (1U)
    {
    }
}

while (1U)
{
}

return 1;
}

// User defined ISR time interval event
// Toggles GPIO on each time tick:
// Initial time interval: 1 second. Update time
// Interval to 0.5 seconds after 10 seconds.
//
void Time_Interval_Event(void)
{
    /* Acknowledge Period Match interrupt generated
on TIMER_CCU_1 */
    TIMER_ClearEvent(&TIMER_0);
    DIGITAL_IO_ToggleOutput(&DIGITAL_IO_0);

    //
    // Increment event count and update time interval to 0.5
    // seconds after 10 seconds
    //
}

```

```
event_count++;
if (event_count == 10U)
{
    if(TIMER_Stop(&TIMER_0) == 0U)
    {
        TIMER_SetTimeInterval(&TIMER_0, 500000000U);
        TIMER_Start(&TIMER_0);
    }
}
```

## Build and Run the Project

### Observation

- For the first 10 seconds LED is toggled for every second, and then for every 0.5 seconds.
- 

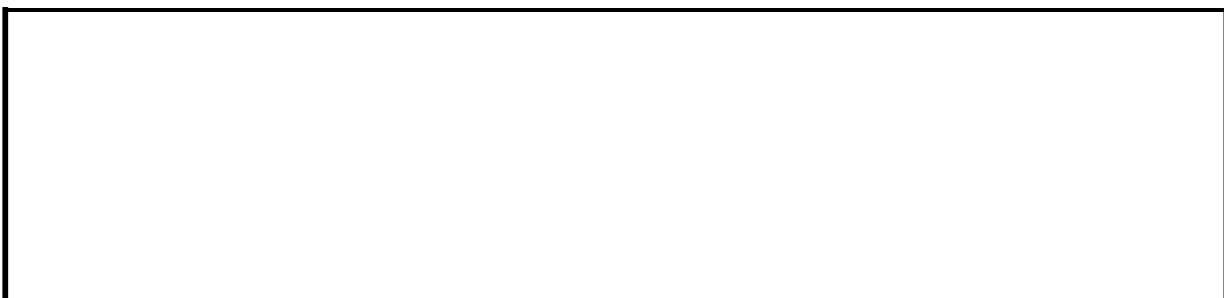


# TIMER

Home

## Release History

### Release History



# TIMER

[Home](#)

[Data Structures](#)

[Data Structure Index](#)

[Data Fields](#)

## Data Structures

Here are the data structures with brief descriptions:

[TIMER](#)

Initialization parameters of the [TIMER APP](#)

# TIMER

[Home](#)

[Data Structures](#)

[Data Structure Index](#)

[Data Fields](#)

[Data Fields](#)

## TIMER Struct Reference

[Data structures](#)

---

## Detailed Description

Initialization parameters of the **TIMER** APP.

Definition at line **116** of file **TIMER.h**.

```
#include <TIMER.h>
```

## Data Fields

uint32_t	time_interval_value_us
const uint32_t	timer_max_value_us
const uint32_t	timer_min_value_us
const uint32_t	shadow_mask
TIMER_MODULE_t const	timer_module
uint16_t	period_value
bool const	start_control
bool const	period_match_enable

## Field Documentation

**bool const TIMER::period\_match\_enable**

Indicate the generation of period match event

Definition at line [141](#) of file **TIMER.h**.

**uint16\_t TIMER::period\_value**

Period value to be loaded into timer for the corresponding time tick

Definition at line [139](#) of file **TIMER.h**.

Referenced by **TIMER\_SetTimeInterval()**.

**const uint32\_t TIMER::shadow\_mask**

shadow transfer mask for the selected timer

Definition at line [121](#) of file **TIMER.h**.

**bool const TIMER::start\_control**

Indicate whether to start the APP during initialization itself

Definition at line [140](#) of file **TIMER.h**.

**uint32\_t TIMER::time\_interval\_value\_us**

Timer interval value for which event is being generated

Definition at line [118](#) of file **TIMER.h**.

### **const uint32\_t TIMER::timer\_max\_value\_us**

Maximum timer value in micro seconds for the available clock

Definition at line [119](#) of file **TIMER.h**.

Referenced by **TIMER\_SetTimeInterval()**.

### **const uint32\_t TIMER::timer\_min\_value\_us**

Minimum timer value in micro seconds for the available clock

Definition at line [120](#) of file **TIMER.h**.

Referenced by **TIMER\_SetTimeInterval()**.

### **TIMER\_MODULE\_t const TIMER::timer\_module**

Indicate which timer module is being used from CCU4 and CCU8

Definition at line [138](#) of file **TIMER.h**.

Referenced by **TIMER\_Clear()**, **TIMER\_ClearEvent()**,  
**TIMER\_GetInterruptStatus()**, **TIMER\_GetTime()**,  
**TIMER\_GetTimerStatus()**, **TIMER\_Init()**, **TIMER\_SetTimeInterval()**,  
**TIMER\_Start()**, and **TIMER\_Stop()**.

---

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

- **TIMER.h**
-



# TIMER

Home

Data Structures

Data Structure Index

Data Fields

## Data Structure Index

T

T

TIMER

T



# TIMER

Home		
Data Structures	Data Structure Index	Data Fields
All	Variables	

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

- period\_match\_enable : **TIMER**
  - period\_value : **TIMER**
  - shadow\_mask : **TIMER**
  - start\_control : **TIMER**
  - time\_interval\_value\_us : **TIMER**
  - timer\_max\_value\_us : **TIMER**
  - timer\_min\_value\_us : **TIMER**
  - timer\_module : **TIMER**
- 
-

# TIMER

Home		
Data Structures	Data Structure Index	Data Fields
All	Variables	

- period\_match\_enable : **TIMER**
  - period\_value : **TIMER**
  - shadow\_mask : **TIMER**
  - start\_control : **TIMER**
  - time\_interval\_value\_us : **TIMER**
  - timer\_max\_value\_us : **TIMER**
  - timer\_min\_value\_us : **TIMER**
  - timer\_module : **TIMER**
-

# TIMER

[Home](#)

[File List](#)

[Globals](#)

## File List

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

[TIMER.c \[code\]](#)

[TIMER.h \[code\]](#)

# TIMER

[Home](#)

[File List](#)

[Globals](#)

[Data Structures](#)

## TIMER.h File Reference

## Detailed Description

**Date:**

2015-12-18

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 **TIMER.h**.

```
#include "timer_conf.h" #include <DAVE_Common.h>
#include "timer_extern.h"
```

## Data Structures

struct **TIMER**

Initialization parameters of the **TIMER** APP. [More...](#)

## TypeDefs

---

```
typedef struct TIMER_TIMER_t
```

```
    Initialization parameters of the TIMER APP.
```

## Functions

DAVE_APP_VERSION_t	<b>TIMER_GetAppVersion</b> (void) Get <b>TIMER</b> APP version.
<b>TIMER_STATUS_t</b>	<b>TIMER_Init</b> ( <b>TIMER_t</b> *const handle_ptr) Initializes a <b>TIMER</b> with generated configuration.
<b>TIMER_STATUS_t</b>	<b>TIMER_Start</b> ( <b>TIMER_t</b> *const handle_ptr) Starts the timer if the initialization of the APP is successful.
<b>TIMER_STATUS_t</b>	<b>TIMER_Stop</b> ( <b>TIMER_t</b> *const handle_ptr) Stops the <b>TIMER</b> , if it is running.
uint32_t	<b>TIMER_GetTime</b> ( <b>TIMER_t</b> *const handle_ptr) Returns the current time in micro seconds by scaling with 100.
<b>TIMER_STATUS_t</b>	<b>TIMER_Clear</b> ( <b>TIMER_t</b> *const handle_ptr) Clears the timer register.
bool	<b>TIMER_GetTimerStatus</b> ( <b>TIMER_t</b> *const handle_ptr) Returns the running state of the timer.
<b>TIMER_STATUS_t</b>	<b>TIMER_SetTimeInterval</b> ( <b>TIMER_t</b> *const handle_ptr, uint32_t time_interval) Set the new time interval for the event generation, by checking with the supported range.
	<b>TIMER_GetInterruptStatus</b>

	bool ( <b>TIMER_t</b> *const handle_ptr)	Indicates the occurrence of time interval event.
	void <b>TIMER_ClearEvent</b> ( <b>TIMER_t</b> *const handle_ptr)	Clears the period match interrupt status of the given timer.
enum	<b>TIMER_MODULE</b> { <b>TIMER_MODULE_CCU4</b> = 0U, <b>TIMER_MODULE_CCU8</b> }	The type identifies the CCU4 or CCU8 timer selected. <a href="#">More...</a>
enum	<b>TIMER_STATUS</b> { <b>TIMER_STATUS_SUCCESS</b> = 0U, <b>TIMER_STATUS_FAILURE</b> }	status of the <b>TIMER</b> APP <a href="#">More...</a>
typedef enum <b>TIMER_MODULE</b>	<b>TIMER_MODULE_t</b>	The type identifies the CCU4 or CCU8 timer selected.
typedef enum <b>TIMER_STATUS</b>	<b>TIMER_STATUS_t</b>	status of the <b>TIMER</b> APP

Go to the source code of this file.

---

# TIMER

<a href="#">Home</a>					
<a href="#">File List</a>	<a href="#">Globals</a>				
<a href="#">All</a>	<a href="#">Functions</a>	<a href="#">Typedefs</a>	<a href="#">Enumerations</a>	<a href="#">Enumerator</a>	

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

- `TIMER_Clear()` : [TIMER.h](#)
- `TIMER_ClearEvent()` : [TIMER.h](#)
- `TIMER_GetAppVersion()` : [TIMER.h](#)
- `TIMER_GetInterruptStatus()` : [TIMER.h](#)
- `TIMER_GetTime()` : [TIMER.h](#)
- `TIMER_GetTimerStatus()` : [TIMER.h](#)
- `TIMER_Init()` : [TIMER.h](#)
- `TIMER_MODULE` : [TIMER.h](#)
- `TIMER_MODULE_CCU4` : [TIMER.h](#)
- `TIMER_MODULE_CCU8` : [TIMER.h](#)
- `TIMER_MODULE_t` : [TIMER.h](#)
- `TIMER_SetTimeInterval()` : [TIMER.h](#)
- `TIMER_Start()` : [TIMER.h](#)
- `TIMER_STATUS` : [TIMER.h](#)
- `TIMER_STATUS_FAILURE` : [TIMER.h](#)
- `TIMER_STATUS_SUCCESS` : [TIMER.h](#)
- `TIMER_STATUS_t` : [TIMER.h](#)
- `TIMER_Stop()` : [TIMER.h](#)
- `TIMER_t` : [TIMER.h](#)

# TIMER

Home					
File List	Globals				
All	Functions	Typedefs	Enumerations	Enumerator	

- TIMER\_Clear() : **TIMER.h**
  - TIMER\_ClearEvent() : **TIMER.h**
  - TIMER\_GetAppVersion() : **TIMER.h**
  - TIMER\_GetInterruptStatus() : **TIMER.h**
  - TIMER\_GetTime() : **TIMER.h**
  - TIMER\_GetTimerStatus() : **TIMER.h**
  - TIMER\_Init() : **TIMER.h**
  - TIMER\_SetTimeInterval() : **TIMER.h**
  - TIMER\_Start() : **TIMER.h**
  - TIMER\_Stop() : **TIMER.h**
- 
-

# TIMER

Home					
File List		Globals			
All	Functions	TypeDefs	Enumerations	Enumerator	

- TIMER\_MODULE\_t : [TIMER.h](#)
  - TIMER\_STATUS\_t : [TIMER.h](#)
  - TIMER\_t : [TIMER.h](#)
- 
-

# TIMER

Home					
File List		Globals			
All	Functions	Typedefs	Enumerations	Enumerator	

- TIMER\_MODULE : [TIMER.h](#)
  - TIMER\_STATUS : [TIMER.h](#)
-

# TIMER

Home					
File List		Globals			
All	Functions	Typedefs	Enumerations	Enumerator	

- TIMER\_MODULE\_CCU4 : [TIMER.h](#)
  - TIMER\_MODULE\_CCU8 : [TIMER.h](#)
  - TIMER\_STATUS\_FAILURE : [TIMER.h](#)
  - TIMER\_STATUS\_SUCCESS : [TIMER.h](#)
-

# TIMER

[Home](#)

[File List](#)

[Globals](#)

## TIMER.h

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

```
00001
00058 #ifndef TIMER_H
00059 #define TIMER_H
00060 /*****
* ****
* ****
* ****
* ****
00061 * HEADER FILES
00062 ****
* ****
**** */
00063 #include "timer_conf.h"
00064 #ifdef TIMER_CCU4_USED
00065 #include <GLOBAL_CCU4/global_ccu4.h>
00066 #endif
00067 #ifdef TIMER_CCU8_USED
00068 #include <GLOBAL_CCU8/global_ccu8.h>
00069 #endif
00070 #include <DAVE_Common.h>
00071
00072 /*****
* ****
* ****
* ****
* ****
00073 * MACROS
00074 ****
* ****
**** */
00075
```

```
00076
00077 /*****
00078 ****
00079 * ENUMS
00080 ****
00081 ****
00082 ****
00083 ****
00084 ****
00085 ****
00086 ****
00087 ****
00088 typedef enum TIMER_MODULE
00089 {
00090     TIMER_MODULE_CCU4 = 0U,
00091     TIMER_MODULE_CCU8
00092 } TIMER_MODULE_t;
00093
00094
00095
00096
00097 typedef enum TIMER_STATUS{
00098     TIMER_STATUS_SUCCESS = 0U,
00099     TIMER_STATUS_FAILURE
00100 } TIMER_STATUS_t;
00101
00102
00103 /*****
00104 ****
00105 ****
00106 * DATA STRUCTURES
00107 ****
00108 ****
00109 ****
00110 ****
00111 ****
00112 ****
00113 ****
00114 ****
00115 ****
00116 typedef struct TIMER
00117 {
00118     uint32_t time_interval_value_us;
00119     const uint32_t timer_max_value_us;
00120     const uint32_t timer_min_value_us;
00121     const uint32_t shadow_mask;
00122 #ifdef TIMER_CCU4_USED
00123     GLOBAL_CCU4_t* const global_ccu4_handler;
00124     XMC_CCU4_SLICE_t* const ccu4_slice_ptr;
00125     const uint8_t ccu4_slice_number; /* Timer
being used */
```

```
00126     XMC_CCU4_SLICE_COMPARE_CONFIG_t* const ccu
4_slice_config_ptr;
00128     XMC_CCU4_SLICE_SR_ID_t  const ccu4_period_
match_node;
00129 #endif
00130 #ifdef TIMER_CCU8_USED
00131     GLOBAL_CCU8_t* const global_ccu8_handler;
00132     XMC_CCU8_SLICE_t* const ccu8_slice_ptr;
00133     const uint8_t ccu8_slice_number; /* Timer
being used */
00134     XMC_CCU8_SLICE_COMPARE_CONFIG_t* const ccu
8_slice_config_ptr;
00136     XMC_CCU8_SLICE_SR_ID_t const ccu8_period_m
atch_node;
00137 #endif
00138     TIMER_MODULE_t const timer_module;
00139     uint16_t period_value;
00140     bool const start_control;
00141     bool const period_match_enable;
00142     bool initialized; /* flag to indicate the
initialization state of the APP instance */
00143 } TIMER_t;
00144
00148 /*****
***** */
00149 * API Prototypes
00150 *****/
00151 #ifdef __cplusplus
00152 extern "C" {
00153 #endif
00154
00190 DAVE_APP_VERSION_t TIMER_GetAppVersion(void)
;
00191
```

```
00223 TIMER_STATUS_t TIMER_Init(TIMER_t *const handle_ptr);
00224
00258 TIMER_STATUS_t TIMER_Start(TIMER_t *const handle_ptr);
00259
00300 TIMER_STATUS_t TIMER_Stop(TIMER_t *const handle_ptr);
00301
00339 uint32_t TIMER_GetTime(TIMER_t *const handle_ptr);
00340
00380 TIMER_STATUS_t TIMER_Clear(TIMER_t *const handle_ptr);
00381
00421 bool TIMER_GetTimerStatus(TIMER_t *const handle_ptr);
00422
00495 TIMER_STATUS_t TIMER_SetTimeInterval(TIMER_t *const handle_ptr, uint32_t time_interval);
00496
00554 bool TIMER_GetInterruptStatus(TIMER_t * const handle_ptr);
00555
00586 void TIMER_ClearEvent(TIMER_t *const handle_ptr);
00587
00591 #include "timer_extern.h" /* Included to access the APP Handles at Main.c */
00592
00593 #ifdef __cplusplus
00594 }
00595 #endif
00596
00597 #endif /* TIMER_H */
```



# TIMER

Home

File List

Globals

## TIMER.c

```
00001
00060 /*****
***** * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
***** * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
***** * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
00061 * HEADER FILES
00062 ****
***** * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
***** * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
***** * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * /
00063 #include "timer.h"
00064
00065 /*****
***** * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
***** * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
***** * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
00066 * MACROS
00067 ****
***** * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
***** * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
***** * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * /
00068 #define TIMER_CMP_100_DUTY ((uin
t16_t)0) /* Compare value for 100% duty cycle */
00069 #define TIMER_RESOLUTION_SEC_TO_MICRO (1000
00000U) /* Convert the resolution from sec to use
c, by dividing with the \
00070
scale factor */
00071 #define TIMER_PRESCALER_MAX (15U)
/* Maximum prescaler values allowed */
00072 #define TIMER_PERIOD_16BIT_MAX (0xFF
FFU) /* Maximum period value */
```

```
00073 #define TIMER_PERIOD_MIN          (0x1U
)      /* Minimum period value */
00074 #define TIMER_CLK_SCALE_FACTOR    (32U)
      /* Scale factor used during calculation of t
he "TIMER_CLK_CONST_SCALED" */
00075
00076 /*****
***** * LOCAL DATA
00078 *****
***** */
00079
00080 /*****
***** * LOCAL ROUTINES
00082 *****
***** */
00083 #ifdef TIMER_CCU4_USED
00084 TIMER_STATUS_t TIMER_CCU4_lInit(TIMER_t* con
st handle_ptr);
00085 void TIMER_CCU4_lShadowTransfer(TIMER_t* con
st handle_ptr);
00086 #endif
00087
00088 #ifdef TIMER_CCU8_USED
00089 TIMER_STATUS_t TIMER_CCU8_lInit(TIMER_t* con
st handle_ptr);
00090 void TIMER_CCU8_lShadowTransfer(TIMER_t* con
st handle_ptr);
00091 #endif
00092
00093 /*****
```

```
*****
00094 * API IMPLEMENTATION
00095 ****
*****,
00096 /* Returns the version of the TIMER APP.
00097 */
00098 DAVE_APP_VERSION_t TIMER_GetAppVersion(void)
00099 {
00100     DAVE_APP_VERSION_t version;
00101
00102     version.major = TIMER_MAJOR_VERSION;
00103     version.minor = TIMER_MINOR_VERSION;
00104     version.patch = TIMER_PATCH_VERSION;
00105
00106     return version;
00107 }
00108
00109 /*
00110 * This function initializes a TIMER APP based on user configuration.
00111 *
00112 */
00113 TIMER_STATUS_t TIMER_Init(TIMER_t* const handle_ptr)
00114 {
00115     TIMER_STATUS_t status;
00116
00117     XMC_ASSERT("TIMER_Init:handle_ptr NULL" ,
00118 (handle_ptr != NULL));
00119
00120     status = TIMER_STATUS_SUCCESS;
00121     /* Check for APP instance is initialized or not */
00122     if (false == handle_ptr->initialized)
00123     {
00124 #ifdef TIMER_CCU4_USED
```

```
00124     if (TIMER_MODULE_CCU4 == handle_ptr->timer_module)
00125     {
00126         /* Configure CCU4 timer for the required time tick settings */
00127         status = TIMER_CCU4_lInit(handle_ptr);
00128     }
00129 #endif
00130
00131 #ifdef TIMER_CCU8_USED
00132     if (TIMER_MODULE_CCU8 == handle_ptr->timer_module)
00133     {
00134         /* Configure CCU8 timer for the required time tick settings */
00135         status = TIMER_CCU8_lInit(handle_ptr);
00136     }
00137 #endif
00138 }
00139
00140     return (status);
00141 }
00142
00143 /*
00144 * This function starts the timer to generate the events for the specified time_interval value
00145 */
00146 TIMER_STATUS_t TIMER_Start(TIMER_t *const handle_ptr)
00147 {
00148     TIMER_STATUS_t status;
00149
00150     XMC_ASSERT("TIMER_Start:handle_ptr NULL" ,
00151                (handle_ptr != NULL));
00152
00153     /* Check for APP instance is initialized or not */
00154 }
```

```
00153     if (true == handle_ptr->initialized)
00154     {
00155 #ifdef TIMER_CCU4_USED
00156         if (TIMER_MODULE_CCU4 == handle_ptr->timer_module)
00157         {
00158             /* Start the timer manually */
00159             XMC_CCU4_SLICE_StartTimer(handle_ptr->
00160 ccu4_slice_ptr);
00160         }
00161 #endif
00162
00163 #ifdef TIMER_CCU8_USED
00164         if (TIMER_MODULE_CCU8 == handle_ptr->timer_module)
00165         {
00166             /* Start the timer manually */
00167             XMC_CCU8_SLICE_StartTimer(handle_ptr->
00168 ccu8_slice_ptr);
00168         }
00169 #endif
00170         status = TIMER_STATUS_SUCCESS;
00171     }
00172 else
00173 {
00174     status = TIMER_STATUS_FAILURE;
00175 }
00176
00177     return (status);
00178 }
00179
00180 /*
00181 * This function stops and clears the timer
00182 */
00183 TIMER_STATUS_t TIMER_Stop(TIMER_t *const handle_ptr)
00184 {
```

```
00185     TIMER_STATUS_t status;
00186
00187     XMC_ASSERT("TIMER_Stop:handle_ptr NULL" ,
00188     (handle_ptr != NULL));
00189
00190     /* Check whether timer is initialized and
00191     in running state */
00192     if ((TIMER_GetTimerStatus(handle_ptr)) && (
00193         true == handle_ptr->initialized))
00194     {
00195         /* Stops the timer */
00196         XMC_CCU4_SLICE_StopTimer(handle_ptr->c
00197         cu4_slice_ptr);
00198     }
00199 #endif
00200
00201 #ifdef TIMER_CCU8_USED
00202     if (TIMER_MODULE_CCU8 == handle_ptr->tim
00203     er_module)
00204     {
00205         /* Stops the timer */
00206         XMC_CCU8_SLICE_StopTimer(handle_ptr->c
00207         cu8_slice_ptr);
00208     }
00209     else
00210     {
00211         status = TIMER_STATUS_FAILURE;
00212     }
00213
00214     return (status);
```

```
00215 }
00216
00217 /*
00218 * This function returns the status of the t
imer
00219 */
00220 bool TIMER_GetTimerStatus(TIMER_t *const ha
ndle_ptr)
00221 {
00222     bool status;
00223
00224     XMC_ASSERT("TIMER_GetTimerStatus:handle_pt
r NULL" , (handle_ptr != NULL));
00225
00226     status = false;
00227
00228 #ifdef TIMER_CCU4_USED
00229     if (TIMER_MODULE_CCU4 == handle_ptr->timer
_module)
00230     {
00231         /* Returns the current status of the tim
er */
00232         status = XMC_CCU4_SLICE_IsTimerRunning(h
andle_ptr->ccu4_slice_ptr);
00233     }
00234 #endif
00235
00236 #ifdef TIMER_CCU8_USED
00237     if (TIMER_MODULE_CCU8 == handle_ptr->timer
_module)
00238     {
00239         /* Returns the current status of the tim
er */
00240         status = XMC_CCU8_SLICE_IsTimerRunning(h
andle_ptr->ccu8_slice_ptr);
00241     }
00242 #endif
```

```
00243
00244     return (status);
00245 }
00246
00247 /*
00248  * This function changes the PWM period which in turn changes the time tick interval value by checking that
00249  * the given time tick value is within supported range.
00250 */
00251 TIMER_STATUS_t TIMER_SetTimeInterval(TIMER_t
00252     *const handle_ptr, uint32_t time_interval)
00253 {
00254     TIMER_STATUS_t status;
00255     uint32_t lfrequency;
00256     uint32_t lprescaler;
00257
00258     XMC_ASSERT("TIMER_SetTimeInterval:handle_ptr NULL" , (handle_ptr != NULL));
00259     status = TIMER_STATUS_FAILURE;
00260
00261     if (false == TIMER_GetTimerStatus(handle_ptr))
00262     {
00263         /* check for time_interval range */
00264         if ((time_interval >= handle_ptr->timer_min_value_us) && (time_interval <= handle_ptr->timer_max_value_us))
00265         {
00266             /* Initialize the prescaler */
00267             lprescaler = 0U;
00268             while (time_interval > (handle_ptr->timer_max_value_us >> (TIMER_PRESCALER_MAX - lprescaler)))
00269             {
```

```

00270             lprescaler++;
00271         }
00272 #ifdef TIMER_CCU4_USED
00273     if (TIMER_MODULE_CCU4 == handle_ptr->t
00274     {
00275         lfrequency = handle_ptr->global_ccu4
00276         _handler->module_frequency;
00277         handle_ptr->ccu4_slice_config_ptr->p
00278         rescaler_initval = lprescaler;
00279         /* Calculate the period register for
00280         the required time_interval value */
00281         handle_ptr->period_value = (uint16_t
00282 )(((uint64_t)time_interval * lfrequency) >> \
00283
00284         handle_ptr->ccu4_slice_config_ptr->prescaler_in
00285         itval) / \
00286
00287         TIMER_RESOLUTION_SEC_TO_MICRO);
00288         /* Actual timer period values is Per
00289         iod_reg_val+1U */
00290         if (handle_ptr->period_value > TIMER
00291         _PERIOD_MIN)
00292         {
00293             (handle_ptr->period_value)--;
00294         }
00295         /* Update the prescaler */
00296         XMC_CCU4_SLICE_SetPrescaler(handle_p
00297         tr->ccu4_slice_ptr, handle_ptr->ccu4_slice_config_
00298         ptr->prescaler_initval);
00299         /* update period, compare and presca
00300         ler values */
00301         TIMER_CCU4_lShadowTransfer(handle_pt
00302         r);
00303         /* Update the status */
00304         status = TIMER_STATUS_SUCCESS;
00305     }

```

```
00293 #endif
00294
00295 #ifdef TIMER_CCU8_USED
00296     if (TIMER_MODULE_CCU8 == handle_ptr->t
00297         imer_module)
00298     {
00299         handle_ptr->ccu8_slice_config_ptr->p
00300         rescaler_initval = lprescaler;
00301         lfrequency = handle_ptr->global_ccu8
00302         _handler->module_frequency;
00303         /* Calculate the period register for
00304             the required time_interval value */
00305         handle_ptr->period_value = (uint16_t
00306 )(((uint64_t)time_interval * lfrequency) >> \
00307
00308         handle_ptr->ccu8_slice_config_ptr->prescaler_in
00309         itval) / \
00310
00311         TIMER_RESOLUTION_SEC_TO_MICRO);
00312         /* Actual timer period values is Per
00313             iod_reg_val+1U */
00314         if (handle_ptr->period_value > TIMER
00315             _PERIOD_MIN)
00316         {
00317             (handle_ptr->period_value)--;
00318         }
00319         /* Update the prescaler */
00320         XMC_CCU8_SLICE_SetPrescaler(handle_p
00321         tr->ccu8_slice_ptr, handle_ptr->ccu8_slice_config_
00322         ptr->prescaler_initval);
00323         /* update period, compare and presca
00324             ler values */
00325         TIMER_CCU8_lShadowTransfer(handle_pt
00326         r);
00327         /* Update the status */
00328         status = TIMER_STATUS_SUCCESS;
00329     }
```

```
00316 #endif
00317     }
00318 }
00319     return (status);
00320 }
00321
00322 /*
00323 * This function reads the timer event(period match interrupt) status of the given timer
00324 */
00325 bool TIMER_GetInterruptStatus(TIMER_t * const
00326 handle_ptr)
00327 {
00328     bool status;
00329     XMC_ASSERT("TIMER_GetInterruptStatus:handle_ptr NULL" , (handle_ptr != NULL));
00330     status = false;
00331 #ifdef TIMER_CCU4_USED
00332     if (TIMER_MODULE_CCU4 == handle_ptr->timer
00333 _module)
00334     {
00335         /* Reads the interrupt status */
00336         status = XMC_CCU4_SLICE_GetEvent(handle_
00337 ptr->ccu4_slice_ptr, XMC_CCU4_SLICE_IRQ_ID_PERIOD_
00338 MATCH);
00339     }
00340 #endif
00341
00342 #ifdef TIMER_CCU8_USED
00343     if (TIMER_MODULE_CCU8 == handle_ptr->timer
00344 _module)
00345     {
00346         /* Reads the interrupt status */
00347         status = XMC_CCU8_SLICE_GetEvent(handle_
00348 ptr->ccu8_slice_ptr, XMC_CCU8_SLICE_IRQ_ID_PERIOD_
00349 MATCH);
00350     }
```

```
00344 #endif
00345     return (status);
00346 }
00347 /*
00348 * This function clears the period match int
00349 * interrupt status of the given timer.
00350 */
00351 void TIMER_ClearEvent(TIMER_t *const handle_
ptr)
00352 {
00353     XMC_ASSERT("TIME_CCU_AcknowledgeInterrupt:
handle_ptr NULL" , (handle_ptr != NULL));
00354
00355 #ifdef TIMER_CCU4_USED
00356     if (TIMER_MODULE_CCU4 == handle_ptr->timer
_module)
00357     {
00358         /* clears the timer event(period match i
ninterrupt) */
00359         XMC_CCU4_SLICE_ClearEvent(handle_ptr->cc
u4_slice_ptr, XMC_CCU4_SLICE_IRQ_ID_PERIOD_MATCH);
00360     }
00361 #endif
00362
00363 #ifdef TIMER_CCU8_USED
00364     if (TIMER_MODULE_CCU8 == handle_ptr->timer
_module)
00365     {
00366         /* clears the timer event(period match i
ninterrupt) */
00367         XMC_CCU8_SLICE_ClearEvent(handle_ptr->cc
u8_slice_ptr, XMC_CCU8_SLICE_IRQ_ID_PERIOD_MATCH);
00368     }
00369 #endif
00370 }
00371
```

```
00372 /*
00373  * This function returns the current time value
00374 */
00375 uint32_t TIMER_GetTime(TIMER_t *const handle_ptr)
00376 {
00377     uint32_t ltimer_val;
00378     uint32_t lprescaler;
00379     uint32_t ltime_val;
00380
00381     XMC_ASSERT("TIMER_GetTimerStatus:handle_ptr == NULL", (handle_ptr != NULL));
00382     ltime_val = 0U;
00383
00384 #ifdef TIMER_CCU4_USED
00385     if (TIMER_MODULE_CCU4 == handle_ptr->timer_module)
00386     {
00387         /* Added one to according to the edge aligned mode */
00388         ltimer_val = (uint32_t)XMC_CCU4_SLICE_GetTimerValue(handle_ptr->ccu4_slice_ptr) + 1U;
00389         lprescaler = handle_ptr->ccu4_slice_config_ptr->prescaler_initval;
00390
00391         /* calculate the time value in micro seconds and scaled with 100 */
00392         ltime_val = (uint32_t)((uint64_t)((uint64_t)ltimer_val * (uint64_t)TIMER_CLK_CONST_SCALED)
00393             >> \
00394             (TIMER_CLK_SCALE_FACTOR - lprescaler));
00395     }
00396
00397 #ifdef TIMER_CCU8_USED
```

```
00398     if (TIMER_MODULE_CCU8 == handle_ptr->timer
00399     {
00400         /* Added one to according to the edge al
00401         iigned mode */
00402         ltimer_val = (uint32_t)XMC_CCU8_SLICE_Ge
00403         tTimerValue(handle_ptr->ccu8_slice_ptr) + 1U;
00404         lprescaler = handle_ptr->ccu8_slice_conf
00405         ig_ptr->prescaler_initval;
00406
00407         /* calculate the time value in micro sec
00408         onds and scaled with 100 */
00409         ltime_val = (uint32_t)((uint64_t)((uint6
00410         4_t)ltimer_val * (uint64_t)TIMER_CLK_CONST_SCALED)
00411         >> \
00412                         (TIMER_CLK_SCALE_
00413             FACTOR - lprescaler));
00414     }
00415 #endif
00416
00417     return ltime_val;
00418 }
00419
00420 /*
00421     * Clear the timer
00422 */
00423 TIMER_STATUS_t TIMER_Clear(TIMER_t *const ha
00424 ndle_ptr)
00425 {
00426     TIMER_STATUS_t status;
00427
00428     XMC_ASSERT("TIMER_Clear:handle_ptr NULL" ,
00429     (handle_ptr != NULL));
00430
00431     /* Check for APP instance is initialized o
00432     r not */
00433     if (true == handle_ptr->initialized)
```

```
00424  {
00425 #ifdef TIMER_CCU4_USED
00426     if (TIMER_MODULE_CCU4 == handle_ptr->timer_module)
00427     {
00428         /* Clear the timer register */
00429         XMC_CCU4_SLICE_ClearTimer(handle_ptr->
ccu4_slice_ptr);
00430     }
00431 #endif
00432
00433 #ifdef TIMER_CCU8_USED
00434     if (TIMER_MODULE_CCU8 == handle_ptr->timer_module)
00435     {
00436         /* Clear the timer register */
00437         XMC_CCU8_SLICE_ClearTimer(handle_ptr->
ccu8_slice_ptr);
00438     }
00439 #endif
00440     status = TIMER_STATUS_SUCCESS;
00441 }
00442 else
00443 {
00444     status = TIMER_STATUS_FAILURE;
00445 }
00446
00447 return (status);
00448 }
00449
00450 ****
* PRIVATE FUNCTION DEFINITIONS ****
*****
00451 #ifdef TIMER_CCU4_USED
00452 /*
00453 * This function configures timer ccu4 timer
with required time tick value
```

```
00454  */
00455 TIMER_STATUS_t TIMER_CCU4_lInit(TIMER_t* const handle_ptr)
00456 {
00457     TIMER_STATUS_t status;
00458     /* Initialize the global registers */
00459     status = (TIMER_STATUS_t)GLOBAL_CCU4_Init(
00460         handle_ptr->global_ccu4_handler);
00461     /* Enable the clock for selected timer */
00462     XMC_CCU4_EnableClock(handle_ptr->global_ccu4_handler->module_ptr, handle_ptr->ccu4_slice_number);
00463     /* Configure the timer with required settings */
00464     XMC_CCU4_SLICE_CompareInit(handle_ptr->ccu4_slice_ptr, handle_ptr->ccu4_slice_config_ptr);
00465     /* programs the timer period and compare register according to time interval value and do the shadow transfer */
00466     TIMER_CCU4_lShadowTransfer(handle_ptr);
00467
00468 #ifdef TIMER_INTERRUPT
00469     if (true == handle_ptr->period_match_enable)
00470     {
00471         /* Binds a period match event to an NVIC node */
00472         XMC_CCU4_SLICE_SetInterruptNode(handle_ptr->ccu4_slice_ptr, XMC_CCU4_SLICE_IRQ_ID_PERIOD_MATCH,
00473                                         handle_ptr->ccu4_period_match_node);
00474         /* Enables a timer(period match) event */
00475         XMC_CCU4_SLICE_EnableEvent(handle_ptr->ccu4_slice_ptr, XMC_CCU4_SLICE_IRQ_ID_PERIOD_MATCH)
```

```
 ;
00476 }
00477 #endif
00478 /* Clears the timer register */
00479 XMC_CCU4_SLICE_ClearTimer(handle_ptr->ccu4
_slice_ptr);
00480
00481 /* update the initialization flag as true
for particular instance*/
00482 handle_ptr->initialized = true;
00483
00484 /* Check whether the start of the timer is
enabled during initialization or not */
00485 if (handle_ptr->start_control == true)
00486 {
00487 /* Start the timer */
00488 XMC_CCU4_SLICE_StartTimer(handle_ptr->cc
u4_slice_ptr);
00489 }
00490
00491 return (status);
00492 }
00493
00494 /*
00495 * This function configures timer period and
compare values and triggers the shadow transfer o
peration
00496 */
00497 void TIMER_CCU4_lShadowTransfer(TIMER_t* con
st handle_ptr)
00498 {
00499 /* programs the timer period register acco
rding to time interval value */
00500 XMC_CCU4_SLICE_SetTimerPeriodMatch(handle_
ptr->ccu4_slice_ptr, handle_ptr->period_value);
00501 /* programs the timer compare register for
50% duty cycle */
```

```
00502     XMC_CCU4_SLICE_SetTimerCompareMatch(handle
00503         _ptr->ccu4_slice_ptr, TIMER_CMP_100_DUTY);
00503     /* Transfers value from shadow timer registers to actual timer registers */
00504     XMC_CCU4_EnableShadowTransfer(handle_ptr->
00504         global_ccu4_handler->module_ptr, handle_ptr->shadow_mask);
00505 }
00506 #endif
00507
00508 #ifdef TIMER_CCU8_USED
00509 /*
00510     * This function configures timer ccu8 timer
00511     * with required time tick value
00511 */
00512 TIMER_STATUS_t TIMER_CCU8_lInit(TIMER_t* const handle_ptr)
00513 {
00514     TIMER_STATUS_t status;
00515     /* Initialize the global registers */
00516     status = (TIMER_STATUS_t)GLOBAL_CCU8_Init(
00516         handle_ptr->global_ccu8_handler);
00517
00518     /* Enable the clock for selected timer */
00519     XMC_CCU8_EnableClock(handle_ptr->global_ccu8_handler->module_ptr, handle_ptr->ccu8_slice_number);
00520     /* Configure the timer with required settings */
00521     XMC_CCU8_SLICE_CompareInit(handle_ptr->ccu8_slice_ptr, handle_ptr->ccu8_slice_config_ptr);
00522     /* programs the timer period and compare register according to time interval value and do the shadow transfer */
00523     TIMER_CCU8_lShadowTransfer(handle_ptr);
00524
00525 #ifdef TIMER_INTERRUPT
```

```
00526     if (true == handle_ptr->period_match_enable
00527     )
00528     {
00529         /* Binds a period match event to an NVIC
00530          node */
00531         XMC_CCU8_SLICE_SetInterruptNode(handle_p
00532             tr->ccu8_slice_ptr, XMC_CCU8_SLICE_IRQ_ID_PERIOD_M
00533             ATCH,
00534             handle_p
00535             tr->ccu8_period_match_node);
00536         /* Enables a timer(period match) event
00537          */
00538         XMC_CCU8_SLICE_EnableEvent(handle_ptr->c
00539             cu8_slice_ptr, XMC_CCU8_SLICE_IRQ_ID_PERIOD_MATCH)
00540         ;
00541     }
00542 #endif
00543     /* Clears the timer register */
00544     XMC_CCU8_SLICE_ClearTimer(handle_ptr->ccu8
00545             _slice_ptr);
00546
00547     /* update the initialization flag as true
00548      for particular instance*/
00549     handle_ptr->initialized = true;
00550
00551     /* Check whether the start of the timer is
00552      enabled during initialization or not */
00553     if (handle_ptr->start_control == true)
00554     {
00555         /* Start the timer */
00556         XMC_CCU8_SLICE_StartTimer(handle_ptr->cc
00557             u8_slice_ptr);
00558     }
00559
00560     return (status);
00561 }
```

```
00551 /*
00552  * This function configures timer period and
00553   compare values and triggers the shadow transfer o
00554   peration
00555 */
00556 /* programs the timer period register acco
00557   rding to time interval value */
00558 XMC_CCU8_SLICE_SetTimerPeriodMatch(handle_
00559   _ptr->ccu8_slice_ptr, handle_ptr->period_value);
00560 /* programs the timer compare register for
00561   50% duty cycle in compare channel 1*/
00562 XMC_CCU8_SLICE_SetTimerCompareMatch(handle_
00563   _ptr->ccu8_slice_ptr,
00564
00565 #endif
```

