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
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).
## Abbreviations and Definitions

### Abbreviations:

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DAVE™</td>
<td>Digital Application Virtual Engineer</td>
</tr>
<tr>
<td>APP</td>
<td>DAVE™ Application</td>
</tr>
<tr>
<td>API</td>
<td>Application Programming Interface</td>
</tr>
<tr>
<td>GUI</td>
<td>Graphical User Interface</td>
</tr>
<tr>
<td>MCU</td>
<td>Microcontroller Unit</td>
</tr>
<tr>
<td>SW</td>
<td>Software</td>
</tr>
<tr>
<td>HW</td>
<td>Hardware</td>
</tr>
<tr>
<td>LLD</td>
<td>Low Level Driver</td>
</tr>
<tr>
<td>I/O</td>
<td>Input/Output</td>
</tr>
<tr>
<td>CCU</td>
<td>Capture Compare Unit</td>
</tr>
</tbody>
</table>

### Definitions:

<table>
<thead>
<tr>
<th>Term</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Singleton</td>
<td>Only single instance of the APP is permitted</td>
</tr>
<tr>
<td>Sharable</td>
<td>Resource sharing with other APPs is permitted</td>
</tr>
<tr>
<td>initProvider</td>
<td>Provides the initialization routine</td>
</tr>
<tr>
<td>Physical connectivity</td>
<td>Hardware inter/intra peripheral (constant) signal connection</td>
</tr>
<tr>
<td>Conditional connectivity</td>
<td>Constrained hardware inter/intra peripheral signal connection</td>
</tr>
<tr>
<td>Aggregation</td>
<td>Indicates consumption of low level (dependent) DAVE™ APPs</td>
</tr>
</tbody>
</table>
The **TIMER APP** provides accurate hardware timer using one timer slice of CCU4 or CCU8 peripheral. And provides 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 microseconds).
4. Connection of timer interval event to other peripherals as a trigger source or for interrupt generation (using the DAVE HW Signal Connections)
The figure above shows the functional overview and use cases of the TIMER APP.

![Diagram of user_application (main.c) and TIMER APP](image)

**Figure 1**: Overview of 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
5. XMC4200 / XMC4100 Reference Manual
7. XMC1300 Reference Manual
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

<table>
<thead>
<tr>
<th>Signal Name</th>
<th>Input/Output</th>
<th>Availability</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>event_time_interval</td>
<td>Output</td>
<td>Conditional</td>
<td>Time interval event signal: Upon enabling &quot;Time interval event&quot; in GUI, signal is populated.</td>
</tr>
</tbody>
</table>

- Connect with an INTERRUPT APP to generate the interrupt for each time interval.
- Connect with
APPs Consumed:

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

**Table 2: APPs Consumed**

<table>
<thead>
<tr>
<th>APP Name</th>
<th>Consumption</th>
</tr>
</thead>
<tbody>
<tr>
<td>GLOBAL_CCU4</td>
<td>Conditionally consumed if &quot;CCU4&quot; is selected</td>
</tr>
<tr>
<td></td>
<td>in UI field &quot;Select timer module&quot;</td>
</tr>
</tbody>
</table>
GLOBAL_CCU8 Conditionally consumed if "CCU8" is selected in UI field "Select timer"
App Configuration Parameters

General Settings

Select timer module: CCU4

Timer Settings

Time interval [usec]: 1.0

Start after initialization

Figure 1: General Settings

Event Settings

Time interval event

Figure 2: Event Settings
## Enumerations

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>enum</code></td>
<td><code>TIMER_MODULE</code> { <code>TIMER_MODULE_CCU4</code> = 0U, <code>TIMER_MODULE_CCU8</code> }</td>
</tr>
<tr>
<td></td>
<td>The type identifies the CCU4 or CCU8 timer selected. <code>More...</code></td>
</tr>
<tr>
<td><code>enum</code></td>
<td><code>TIMER_STATUS</code> { <code>TIMER_STATUS_SUCCESS</code> = 0U, <code>TIMER_STATUS_FAILURE</code> }</td>
</tr>
<tr>
<td></td>
<td>status of the <code>TIMER APP</code> <code>More...</code></td>
</tr>
<tr>
<td><code>typedef</code></td>
<td><code>TIMER_MODULE</code> <code>TIMER_MODULE_t</code></td>
</tr>
<tr>
<td></td>
<td>The type identifies the CCU4 or CCU8 timer selected.</td>
</tr>
<tr>
<td><code>typedef</code></td>
<td><code>TIMER_STATUS</code> <code>TIMER_STATUS_t</code></td>
</tr>
<tr>
<td></td>
<td>status of the <code>TIMER APP</code></td>
</tr>
</tbody>
</table>
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`. 
<table>
<thead>
<tr>
<th>Home</th>
<th>Data Structures</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data structures</td>
<td></td>
</tr>
</tbody>
</table>
### Data Structures

<table>
<thead>
<tr>
<th>Type</th>
<th>Definition</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>struct</td>
<td>TIMER</td>
<td>Initialization parameters of the TIMER APP.</td>
</tr>
<tr>
<td>typedef struct</td>
<td>TIMER TIMER_t</td>
<td>Initialization parameters of the TIMER APP.</td>
</tr>
</tbody>
</table>

More...
# TIMER

## Methods

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

```c
#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))
```
```c
{
    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.

```c
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:

```c
#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.

```c
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:

```c
#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:

```c
#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:

```c
#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:

```c
#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);
}

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:

```c
#include <DAVE.h>
int main(void)
{
    DAVE_STATUS_t init_status;
    init_status = DAVE_Init();  // TIMER_Init(&TIMER_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.
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:**
```c
#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
```
`#define TIMER_GPIO_PIN 0U       // Pin number
#define TIMER_500MS 500000*100U`

```c
volatile uint32_t count = 0U;          // count variable to change the time tick interval
uint32_t shadow_transfer_msk;          // This 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:

```c
#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);
    }
    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:**

<table>
<thead>
<tr>
<th>TIMER_STATUS_t</th>
</tr>
</thead>
<tbody>
<tr>
<td>TIMER_STATUS_SUCCESS : if timer is running and stop is successful</td>
</tr>
<tr>
<td>TIMER_STATUS_FAILURE : if timer is in idle state, and stop is called</td>
</tr>
</tbody>
</table>

**Description:**

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

**Example Usage:**

```c
#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_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.
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)**

```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, 50000000U);
    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

Release History

Release History

---
Data Structures

Here are the data structures with brief descriptions:

| TIMER      | Initialization parameters of the TIMER APP |
## 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

<table>
<thead>
<tr>
<th>Type</th>
<th>Variable</th>
</tr>
</thead>
<tbody>
<tr>
<td>uint32_t</td>
<td>time_interval_value_us</td>
</tr>
<tr>
<td>const uint32_t</td>
<td>timer_max_value_us</td>
</tr>
<tr>
<td>const uint32_t</td>
<td>timer_min_value_us</td>
</tr>
<tr>
<td>const uint32_t</td>
<td>shadow_mask</td>
</tr>
<tr>
<td>TIMER_MODULE_t const</td>
<td>timer_module</td>
</tr>
<tr>
<td>uint16_t</td>
<td>period_value</td>
</tr>
<tr>
<td>bool const</td>
<td>start_control</td>
</tr>
<tr>
<td>bool const</td>
<td>period_match_enable</td>
</tr>
</tbody>
</table>
### 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
**const uint32_t** `TIMER::timer_max_value_us`

Maximum timer value in micro seconds for the available clock

Definition at line 118 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 119 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 120 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**
### Data Structure Index

| T | T | T |
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

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

- **TIMER.c** [code]
- **TIMER.h** [code]
# TIMER

<table>
<thead>
<tr>
<th>Home</th>
<th>File List</th>
<th>Globals</th>
<th>Data Structures</th>
</tr>
</thead>
</table>

**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**.

```c
#include "timer_conf.h" #include <DAVE_Common.h>
#include "timerExtern.h"
```
Data Structures

<table>
<thead>
<tr>
<th>struct TIMER</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initialization parameters of the TIMER APP. More...</td>
</tr>
</tbody>
</table>
typedef struct TIMER TIMER_t
Initialization parameters of the TIMER APP.
## Functions

<table>
<thead>
<tr>
<th>Type</th>
<th>Function Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DAVE_APP_VERSION_t</td>
<td>TIMER_GetAppVersion (void)</td>
<td>Get TIMER APP version.</td>
</tr>
<tr>
<td>TIMER_STATUS_t</td>
<td>TIMER_Init (TIMER_t *const handle_ptr)</td>
<td>Initializes a TIMER with generated configuration.</td>
</tr>
<tr>
<td>TIMER_STATUS_t</td>
<td>TIMER_Start (TIMER_t *const handle_ptr)</td>
<td>Starts the timer if the initialization of the APP is successful.</td>
</tr>
<tr>
<td>TIMER_STATUS_t</td>
<td>TIMER_Stop (TIMER_t *const handle_ptr)</td>
<td>Stops the TIMER, if it is running.</td>
</tr>
<tr>
<td>uint32_t</td>
<td>TIMER_GetTime (TIMER_t *const handle_ptr)</td>
<td>Returns the current time in microseconds by scaling with 100.</td>
</tr>
<tr>
<td>TIMER_STATUS_t</td>
<td>TIMER_Clear (TIMER_t *const handle_ptr)</td>
<td>Clears the timer register.</td>
</tr>
<tr>
<td>bool</td>
<td>TIMER_GetTimerStatus (TIMER_t *const handle_ptr)</td>
<td>Returns the running state of the timer.</td>
</tr>
<tr>
<td>TIMER_STATUS_t</td>
<td>TIMER_SetTimeInterval (TIMER_t *const handle_ptr, uint32_t time_interval)</td>
<td>Set the new time interval for the event generation, by checking with the supported range.</td>
</tr>
<tr>
<td></td>
<td>TIMER_GetInterruptStatus</td>
<td></td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>----------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td><code>bool (TIMER_t *const handle_ptr)</code></td>
<td>Indicates the occurrence of time interval event.</td>
<td></td>
</tr>
<tr>
<td><code>void TIMER_ClearEvent (TIMER_t *const handle_ptr)</code></td>
<td>Clears the period match interrupt status of the given timer.</td>
<td></td>
</tr>
</tbody>
</table>

### Enum Definitions

- **`enum TIMER_MODULE`**
  - `TIMER_MODULE_CCU4 = 0U`
  - `TIMER_MODULE_CCU8`
  The type identifies the CCU4 or CCU8 timer selected. 
  
- **`enum TIMER_STATUS`**
  - `TIMER_STATUS_SUCCESS = 0U`
  - `TIMER_STATUS_FAILURE`
  status of the `TIMER` APP

### Data Type Definitions

- **`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

Go to the source code of this file.
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

- 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_MODULE_t : TIMER.h
• TIMER_STATUS_t : TIMER.h
• TIMER_t : TIMER.h
**TIMER**

<table>
<thead>
<tr>
<th>Home</th>
<th>File List</th>
<th>Globals</th>
<th>All</th>
<th>Functions</th>
<th>Typedefs</th>
<th>Enumerations</th>
<th>Enumerator</th>
</tr>
</thead>
</table>

- TIMER_MODULE : `TIMER.h`
- TIMER_STATUS : `TIMER.h`
• TIMER_MODULE_CCU4 : TIMER.h
• TIMER_MODULE_CCU8 : TIMER.h
• TIMER_STATUS_FAILURE : TIMER.h
• TIMER_STATUS_SUCCESS : TIMER.h
#ifndef TIMER_H
#define TIMER_H
/
************ HEADER FILES ************

#include "timer_conf.h"

#ifdef TIMER_CCU4_USED
#include <GLOBAL_CCU4/global_ccu4.h>
#endif

#ifdef TIMER_CCU8_USED
#include <GLOBAL_CCU8/global_ccu8.h>
#endif

#include <DAVE_Common.h>
/

************ MACROS ************

*/

Go to the documentation of this file.
/*******************************************
* ENUMS
*******************************************/

typedef enum TIMER_MODULE {
   TIMER_MODULE_CCU4 = 0U,
   TIMER_MODULE_CCU8
} TIMER_MODULE_t;

typedef enum TIMER_STATUS {
   TIMER_STATUS_SUCCESS = 0U,
   TIMER_STATUS_FAILURE
} TIMER_STATUS_t;

/*******************************************
* DATA STRUCTURES
*******************************************/

typedef struct TIMER {
   uint32_t time_interval_value_us;
   const uint32_t timer_max_value_us;
   const uint32_t timer_min_value_us;
   const uint8_t shadow_mask;
   #ifdef TIMER_CCU4_USED
   GLOBAL_CCU4_t* const global_ccu4_handler;
   XMC_CCU4_SLICE_t* const ccu4_slice_ptr;
   const uint8_t ccu4_slice_number; /* Timer being used */
}
XMC_CCU4_SLICE_COMPARE_CONFIG_t* const ccu4_slice_config_ptr;
const XMC_CCU4_SLICE_SR_ID_t const ccu4_period_match_node;
#endif
#ifdef TIMER_CCU8_USED
GLOBAL_CCU8_t* const global_ccu8_handler;
const XMC_CCU8_SLICE_t* const ccu8_slice_ptr;
const uint8_t ccu8_slice_number; /* Timer being used */
XMC_CCU8_SLICE_COMPARE_CONFIG_t* const ccu8_slice_config_ptr;
XMC_CCU8_SLICE_SR_ID_t const ccu8_period_match_node;
#endif
TIMER_MODULE_t const timer_module;
uint16_t period_value;
bool const start_control;
bool const period_match_enable;
bool initialized; /* flag to indicate the initialization state of the APP instance */
} TIMER_t;

/* API Prototypes */
#ifdef __cplusplus
extern "C" {
#endif

DAVE_APP_VERSION_t TIMER_GetAppVersion(void);

#ifdef __cplusplus
}
#endif
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 "timerExtern.h" /* Included to access the APP Handles at Main.c */
00592
00593 #ifdef __cplusplus
00594 }
00595 #endif
00596
00597 #endif /* TIMER_H */
TIMER.c

00001
00060 /***************************************************/
00061 * HEADER FILES
00062 ***************************************************/
00063 #include "timer.h"
00064
00065 /***************************************************/
00066 * MACROS
00067 ***************************************************/
00068 #define TIMER_CMP_100_DUTY ((uint16_t)0) /* Compare value for 100% duty cycle */
00069 #define TIMER_RESOLUTION_SEC_TO_MICRO (100000000U) /* Convert the resolution from sec to usec, by dividing with the scale factor */
00070 #define TIMER_PRESCALER_MAX (15U) /* Maximum prescaler values allowed */
00071 #define TIMER_PERIOD_16BIT_MAX (0xFF FFU) /* Maximum period value */


```c
#define TIMER_PERIOD_MIN (0x1U) /* Minimum period value */
#define TIMER_CLK_SCALE_FACTOR (32U) /* Scale factor used during calculation of the "TIMER_CLK_CONST_SCALED" */

/*******************************************
**************************************************
**************************
* LOCAL DATA
**************************************************
*************************/

/*******************************************
**************************************************
**************************
* LOCAL ROUTINES
**************************************************
*************************/

#ifdef TIMER_CCU4_USED
 TIMER_STATUS_t TIMER_CCU4_lInit(TIMER_t* const handle_ptr);
 void TIMER_CCU4_lShadowTransfer(TIMER_t* const handle_ptr);
#endif

#ifdef TIMER_CCU8_USED
 TIMER_STATUS_t TIMER_CCU8_lInit(TIMER_t* const handle_ptr);
 void TIMER_CCU8_lShadowTransfer(TIMER_t* const handle_ptr);
#endif
```

/* Returns the version of the TIMER APP. */

DAVE_APP_VERSION_t TIMER_GetAppVersion(void)
{
    DAVE_APP_VERSION_t version;
    version.major = TIMER_MAJOR_VERSION;
    version.minor = TIMER_MINOR_VERSION;
    version.patch = TIMER_PATCH_VERSION;

    return version;
}

/* This function initializes a TIMER APP based on user configuration. */

TIMER_STATUS_t TIMER_Init(TIMER_t* const handle_ptr)
{
    TIMER_STATUS_t status;
    XMC_ASSERT("TIMER_Init:handle_ptr NULL", (handle_ptr != NULL));
    status = TIMER_STATUS_SUCCESS;
    /* Check for APP instance is initialized or not */
    if (false == handle_ptr->initialized)
    {
        #ifdef TIMER_CCU4_USED

if (TIMER_MODULE_CCU4 == handle_ptr->timer_module) {
   /* Configure CCU4 timer for the required time tick settings */
   status = TIMER_CCU4_lInit(handle_ptr);
}
#endif

#ifdef TIMER_CCU8_USED
if (TIMER_MODULE_CCU8 == handle_ptr->timer_module) {
   /* Configure CCU8 timer for the required time tick settings */
   status = TIMER_CCU8_lInit(handle_ptr);
}
#endif
}
return (status);

/* This function starts the timer to generate the events for the specified time_interval value */

TIMER_STATUS_t TIMER_Start(TIMER_t *const handle_ptr) {
   TIMER_STATUS_t status;
   XMC_ASSERT("TIMER_Start:handle_ptr NULL", (handle_ptr != NULL));
   /* Check for APP instance is initialized or not */
if (true == handle_ptr->initialized)
{
  ifdef TIMER_CCU4_USED
  if (TIMER_MODULE_CCU4 == handle_ptr->timer_module)
  {
    /* Start the timer manually */
    XMC_CCU4_SLICE_StartTimer(handle_ptr->ccu4_slice_ptr);
  }
  endif
  ifdef TIMER_CCU8_USED
  if (TIMER_MODULE_CCU8 == handle_ptr->timer_module)
  {
    /* Start the timer manually */
    XMC_CCU8_SLICE_StartTimer(handle_ptr->ccu8_slice_ptr);
  }
  endif
  status = TIMER_STATUS_SUCCESS;
}
else
{
  status = TIMER_STATUS_FAILURE;
}
return (status);

/*
 * This function stops and clears the timer
 */

TIMER_STATUS_t TIMER_Stop(TIMER_t *const handle_ptr)
{

TIMER_STATUS_t status;

XMC_ASSERT("TIMER_Stop:handle_ptr NULL", (handle_ptr != NULL));

/* Check whether timer is initialized and in running state */
if ((TIMER_GetTimerStatus(handle_ptr)) && (true == handle_ptr->initialized))
{
  #ifdef TIMER_CCU4_USED
    if (TIMER_MODULE_CCU4 == handle_ptr->timer_module)
    {
      /* Stops the timer */
      XMC_CCU4_SLICE_StopTimer(handle_ptr->ccu4_slice_ptr);
    }
  #endif

  #ifdef TIMER_CCU8_USED
    if (TIMER_MODULE_CCU8 == handle_ptr->timer_module)
    {
      /* Stops the timer */
      XMC_CCU8_SLICE_StopTimer(handle_ptr->ccu8_slice_ptr);
    }
  #endif

  status = TIMER_STATUS_SUCCESS;
}
else
{
  status = TIMER_STATUS_FAILURE;
}

return (status);
bool TIMER_GetTimerStatus(TIMER_t *const handle_ptr) {
    bool status;
    XMC_ASSERT("TIMER_GetTimerStatus:handle_ptr NULL", (handle_ptr != NULL));
    status = false;
    #ifdef TIMER_CCU4_USED
    if (TIMER_MODULE_CCU4 == handle_ptr->timer_module) {
        /* Returns the current status of the timer */
        status = XMC_CCU4_SLICE_IsTimerRunning(handle_ptr->ccu4_slice_ptr);
    }
    #endif
    #ifdef TIMER_CCU8_USED
    if (TIMER_MODULE_CCU8 == handle_ptr->timer_module) {
        /* Returns the current status of the timer */
        status = XMC_CCU8_SLICE_IsTimerRunning(handle_ptr->ccu8_slice_ptr);
    }
    #endif
}
/* This function changes the PWM period which in turn changes the time tick interval value by checking that the given time tick value is within supported range. */

TIMER_STATUS_t TIMER_SetTimeInterval(TIMER_t *const handle_ptr, uint32_t time_interval)
{
    TIMER_STATUS_t status;
    uint32_t lfrequency;
    uint32_t lprescaler;
    
    XMC_ASSERT("TIMER_SetTimeInterval:handle_ptr NULL", (handle_ptr != NULL));
    status = TIMER_STATUS_FAILURE;
    
    if (false == TIMER_GetTimerStatus(handle_ptr))
    {
        /* check for time_interval range */
        if ((time_interval >= handle_ptr->timer_min_value_us) && (time_interval <= handle_ptr->timer_max_value_us))
        {
            /* Initialize the prescaler */
            lprescaler = 0U;
            while (time_interval > (handle_ptr->timer_max_value_us >> (TIMER_PRESCALER_MAX - lprescaler)))
            {
            }
lprescaler++;

#ifdef TIMER_CCU4_USED

if (TIMER_MODULE_CCU4 == handle_ptr->timer_module) {
    lfrequency = handle_ptr->global_ccu4_handler->module_frequency;
    handle_ptr->ccu4_slice_config_ptr->prescaler_initval = lprescaler;

    /* Calculate the period register for the required time_interval value */
    handle_ptr->period_value = (uint16_t)((((uint64_t)time_interval * lfrequency) >> handle_ptr->ccu4_slice_config_ptr->prescaler_initval) /
        TIMER_RESOLUTION_SEC_TO_MICRO);

    /* Actual timer period values is Period_reg_val+1U */
    if (handle_ptr->period_value > TIMER_PERIOD_MIN) {
        (handle_ptr->period_value)--;
    }

    /* Update the prescaler */
    XMC_CCU4_SLICE_SetPrescaler(handle_ptr->ccu4_slice_ptr, handle_ptr->ccu4_slice_config_ptr->prescaler_initval);

    /* update period, compare and prescaler values */
    TIMER_CCU4_lShadowTransfer(handle_ptr);

    /* Update the status */
    status = TIMER_STATUS_SUCCESS;
}

#ifndef TIMER_CCU8_USED
if (TIMER_MODULE_CCU8 == handle_ptr->timer_module)
{
   handle_ptr->ccu8_slice_config_ptr->prescaler_initval = lprescaler;
   lfrequency = handle_ptr->global_ccu8_handler->module_frequency;

   /* Calculate the period register for the required time_interval value */
   handle_ptr->period_value = (uint16_t)(((uint64_t)time_interval * lfrequency) >>
               handle_ptr->ccu8_slice_config_ptr->prescaler_inival)
               / TIMER_RESOLUTION_SEC_TO_MICRO);

   /* Actual timer period values is Period_reg_val+1U */
   if (handle_ptr->period_value > TIMER_PERIOD_MIN)
   {
      (handle_ptr->period_value)--;
   }

   /* Update the prescaler */
   XMC_CCU8_SLICE_SetPrescaler(handle_ptr->ccu8_slice_ptr, handle_ptr->ccu8_slice_config_ptr->prescaler_initval);

   /* update period, compare and prescaler values */
   TIMER_CCU8_lShadowTransfer(handle_ptr);

   /* Update the status */
   status = TIMER_STATUS_SUCCESS;
}
}
bool TIMER_GetInterruptStatus(TIMER_t * const handle_ptr) {
    bool status;
    XMC_ASSERT("TIMER_GetInterruptStatus:handle_ptr NULL", (handle_ptr != NULL));
    status = false;
    #ifdef TIMER_CCU4_USED
    if (TIMER_MODULE_CCU4 == handle_ptr->timer_module) {
        /* Reads the interrupt status */
        status = XMC_CCU4_SLICE_GetEvent(handle_ptr->ccu4_slice_ptr, XMC_CCU4_SLICE_IRQ_ID_PERIOD_MATCH);
    }
    #endif
    #ifdef TIMER_CCU8_USED
    if (TIMER_MODULE_CCU8 == handle_ptr->timer_module) {
        /* Reads the interrupt status */
        status = XMC_CCU8_SLICE_GetEvent(handle_ptr->ccu8_slice_ptr, XMC_CCU8_SLICE_IRQ_ID_PERIOD_MATCH);
    }
    #endif
    return (status);
}
#ifndef

return (status);
}

/*
 * This function clears the period match interrupt status of the given timer.
 */

void TIMER_ClearEvent(TIMER_t *const handle_ptr)
{
    XMC_ASSERT("TIME_CCU_AcknowledgeInterrupt: handle_ptr NULL", (handle_ptr != NULL));

    if (TIMER_MODULE_CCU4 == handle_ptr->timer_module)
    {
        /* clears the timer event(period match interrupt) */
        XMC_CCU4_SLICE_ClearEvent(handle_ptr->ccu4_slice_ptr, XMC_CCU4_SLICE_IRQ_ID_PERIOD_MATCH);
    }

    if (TIMER_MODULE_CCU8 == handle_ptr->timer_module)
    {
        /* clears the timer event(period match interrupt) */
        XMC_CCU8_SLICE_ClearEvent(handle_ptr->ccu8_slice_ptr, XMC_CCU8_SLICE_IRQ_ID_PERIOD_MATCH);
    }

} #endif

00371
This function returns the current time value

uint32_t TIMER_GetTime(TIMER_t *const handle_ptr)
{
    uint32_t ltimer_val;
    uint32_t lprescaler;
    uint32_t ltime_val;

    XMC_ASSERT("TIMER_GetTimerStatus:handle_ptr NULL", (handle_ptr != NULL));
    ltime_val = 0U;

    #ifdef TIMER_CCU4_USED
    if (TIMER_MODULE_CCU4 == handle_ptr->timer_module)
    {
        /* Added one to according to the edge aligned mode */
        ltimer_val = (uint32_t)XMC_CCU4_SLICE_GetTimerValue(handle_ptr->ccu4_slice_ptr) + 1U;
        lprescaler = handle_ptr->ccu4_slice_config_ptr->prescaler_initval;

        /* calculate the time value in micro seconds and scaled with 100 */
        ltime_val = (uint32_t)((uint64_t)((uint64_t)ltimer_val * (uint64_t)TIMER_CLK_CONST_SCALED) >> \n        (TIMER_CLK_SCALE_FACTOR - lprescaler));
    }
    #endif

    #ifdef TIMER_CCU8_USED
    \n    \n    #endif
}
if (TIMER_MODULE_CCU8 == handle_ptr->timer_module)
{
    /* Added one to according to the edge aligned mode */
    ltimer_val = (uint32_t)XMC_CCU8_SLICE_GetTimerValue(handle_ptr->ccu8_slice_ptr) + 1U;
    lprescaler = handle_ptr->ccu8_slice_config_ptr->prescaler_initval;

    /* calculate the time value in micro seconds and scaled with 100 */
    ltime_val = ((uint64_t)((uint64_t)ltimer_val * (uint64_t)TIMER_CLK_CONST_SCALED) >>
                  (TIMER_CLK_SCALE_FACTOR - lprescaler));
}
#endif
return ltime_val;

TIMER_STATUS_t TIMER_Clear(TIMER_t *const handle_ptr)
{
    TIMER_STATUS_t status;
    XMC_ASSERT("TIMER_Clear:handle_ptr NULL", (handle_ptr != NULL));

    /* Check for APP instance is initialized or not */
    if (true == handle_ptr->initialized)
{  
    #ifdef TIMER_CCU4_USED  
    if (TIMER_MODULE_CCU4 == handle_ptr->timer_module)  
    {  
        /* Clear the timer register */  
        XMC_CCU4_SLICE_ClearTimer(handle_ptr->ccu4_slice_ptr);  
    }  
    #endif  

    #ifdef TIMER_CCU8_USED  
    if (TIMER_MODULE_CCU8 == handle_ptr->timer_module)  
    {  
        /* Clear the timer register */  
        XMC_CCU8_SLICE_ClearTimer(handle_ptr->ccu8_slice_ptr);  
    }  
    #endif  

    status = TIMER_STATUS_SUCCESS;  
}

else  
{  
    status = TIMER_STATUS_FAILURE;  
}

return (status);  
}

/**************************************************************************  
* PRIVATE FUNCTION DEFINITIONS ******************************************  
**************************************************************************/*

#ifdef TIMER_CCU4_USED  
/*  
* This function configures timer ccu4 timer with required time tick value  
*/  

#pragma pack(push, 1)  
#pragma pack(pop)  

XMC_CCU4_SLICE_ConfigTimer(handle_ptr->ccu4_slice_ptr, time_tick_value, timer_id);

#pragma pack(pop)
TIMER_STATUS_t TIMER_CCU4_lInit(TIMER_t* const handle_ptr)
{
    TIMER_STATUS_t status;
    /* Initialize the global registers */
    status = (TIMER_STATUS_t)GLOBAL_CCU4_Init(handle_ptr->global_ccu4_handler);
    /* Enable the clock for selected timer */
    XMC_CCU4_EnableClock(handle_ptr->global_ccu4_handler->module_ptr, handle_ptr->ccu4_slice_number);
    /* Configure the timer with required settings */
    XMC_CCU4_SLICE_CompareInit(handle_ptr->ccu4_slice_ptr, handle_ptr->ccu4_slice_config_ptr);
    /* programs the timer period and compare register according to time interval value and do the shadow transfer */
    TIMER_CCU4_lShadowTransfer(handle_ptr);
    #ifdef TIMER_INTERRUPT
    if (true == handle_ptr->period_match_enable)
    {
        /* Binds a period match event to an NVIC node */
        XMC_CCU4_SLICE_SetInterruptNode(handle_ptr->ccu4_slice_ptr, XMC_CCU4_SLICE_IRQ_ID_PERIOD_MATCH, handle_ptr->ccu4_period_match_node);
        /* Enables a timer(period match) event */
        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->ccu4_slice_ptr);
00489  }
00490
00491  return (status);
00492 }
00493
00494  /*
00495  * This function configures timer period and compare values and triggers the shadow transfer operation
00496  */
00497  void TIMER_CCU4_lShadowTransfer(TIMER_t* const handle_ptr)
00498  {
00499    /* programs the timer period register according 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 */
XMC_CCU4_SLICE_SetTimerCompareMatch(handle_ptr->ccu4_slice_ptr, TIMER_CMP_100_DUTY);

/* Transfers value from shadow timer registers to actual timer registers */
XMC_CCU4_EnableShadowTransfer(handle_ptr->global_ccu4_handler->module_ptr, handle_ptr->shadow_mask);

#endif

#endif TIMER_CCU8_USED

/* This function configures timer ccu8 timer with required time tick value */

TIMER_STATUS_t TIMER_CCU8_lInit(TIMER_t* const handle_ptr) {
    TIMER_STATUS_t status;
    /* Initialize the global registers */
    status = (TIMER_STATUS_t)GLOBAL_CCU8_Init(handle_ptr->global_ccu8_handler);
    /* Enable the clock for selected timer */
    XMC_CCU8_EnableClock(handle_ptr->global_ccu8_handler->module_ptr, handle_ptr->ccu8_slice_number);
    /* Configure the timer with required settings */
    XMC_CCU8_SLICE_CompareInit(handle_ptr->ccu8_slice_ptr, handle_ptr->ccu8_slice_config_ptr);
    /* programs the timer period and compare register according to time interval value and do the shadow transfer */
    TIMER_CCU8_lShadowTransfer(handle_ptr);
#endif TIMER_INTERRUPT
if (true == handle_ptr->period_match_enable)
{
/* Binds a period match event to an NVIC node */
XMC_CCU8_SLICE_SetInterruptNode(handle_ptr->ccu8_slice_ptr, XMC_CCU8_SLICE_IRQ_ID_PERIOD_MATCH,
handle_ptr->ccu8_period_match_node);
/* Enables a timer(period match) event */
XMC_CCU8_SLICE_EnableEvent(handle_ptr->ccu8_slice_ptr, XMC_CCU8_SLICE_IRQ_ID_PERIOD_MATCH);
}
#endif
/* Clears the timer register */
XMC_CCU8_SLICE_ClearTimer(handle_ptr->ccu8_slice_ptr);
/* update the initialization flag as true for particular instance*/
handle_ptr->initialized = true;
/* Check whether the start of the timer is enabled during initialization or not */
if (handle_ptr->start_control == true)
{
/* Start the timer */
XMC_CCU8_SLICE_StartTimer(handle_ptr->ccu8_slice_ptr);
}
return (status);
This function configures timer period and compare values and triggers the shadow transfer operation.

```c
void TIMER_CCU8_lShadowTransfer(TIMER_t* const handle_ptr)
{
    /* programs the timer period register according to time interval value */
    XMC_CCU8_SLICE_SetTimerPeriodMatch(handle_ptr->ccu8_slice_ptr, handle_ptr->period_value);
    /* programs the timer compare register for 50% duty cycle in compare channel 1*/
    XMC_CCU8_SLICE_SetTimerCompareMatch(handle_ptr->ccu8_slice_ptr,
                                          XMC_CCU8_SLICE_COMPARE_CHANNEL_1,
                                          TIMER_CMP_100_DUTY);
    /* Transfers value from shadow timer registers to actual timer registers */
    XMC_CCU8_EnableShadowTransfer(handle_ptr->global_ccu8_handler->module_ptr, handle_ptr->shadow_mask);
}
```

```c
#endif
```