

SYSTIMER

[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](#)



SYSTIMER

[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).

SYSTIMER

Home

Abbreviations and Definitions

Abbreviations and Definitions

Abbreviations:	
DAVE™	Digital Application Virtual Engineer
APP	DAVE Application
API	Application Programming Interface
GUI	Graphical User Interface
SYSTIMER	System Timer
MCU	Microcontroller Unit
SW	Software
HW	Hardware
LLD	Low Level Driver
IO	Input Output

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

--

SYSTIMER

Home

Overview

Overview

The **SYSTIMER** APP uses the SysTick interrupt to call user functions periodically at a specified rate or after a given time period expires.

SYSTIMER APP supports following features:

1. **SYSTIMER** APP creates the software timers with **microseconds** resolution using hardware SysTick timer as a reference.
2. **SYSTIMER** APP can provides one shot and periodic software timer modes with user configurable time interval and supports maximum up to 16 software timers.
3. Each software timer can be created with associated user callback registration using **SYSTIMER_CreateTimer()** API.
4. The software timers callback is executed in the interrupt context.
5. Software timer will start running by calling **SYSTIMER_StartTimer()** API.
6. Software timer will be stopped by calling **SYSTIMER_StopTimer()** API.
7. Each software timer created has unique ID obtained from **SYSTIMER_CreateTimer()**. This ID can be used in other API's provided by this APP to reference the software timer.
8. Each software timer has a timer control block to describe the current status of the software timer. During run time, user can get the state of software timer using **SYSTIMER_GetTimerState()** API.
9. Software timer can be reconfigured with new time interval using **SYSTIMER_RestartTimer()** API.
10. All created software timers are managed through a timer list.
11. SysTick exception controls the timer list and if any of the timer is expired corresponding call back function is called and which is

- registered through **SYSTIMER_CreateTimer()** API.
12. Deletion of software timer after completion of usage using **SYSTIMER_DeleteTimer()** API.

Software timer mode of operation

1. **One shot:**

One shot software timer will cycle only once for the given time interval. Once the software timers period expires, the timer stops. The callback function is executed only once. The timer can be restarted manually by calling **SYSTIMER_RestartTimer()** API.

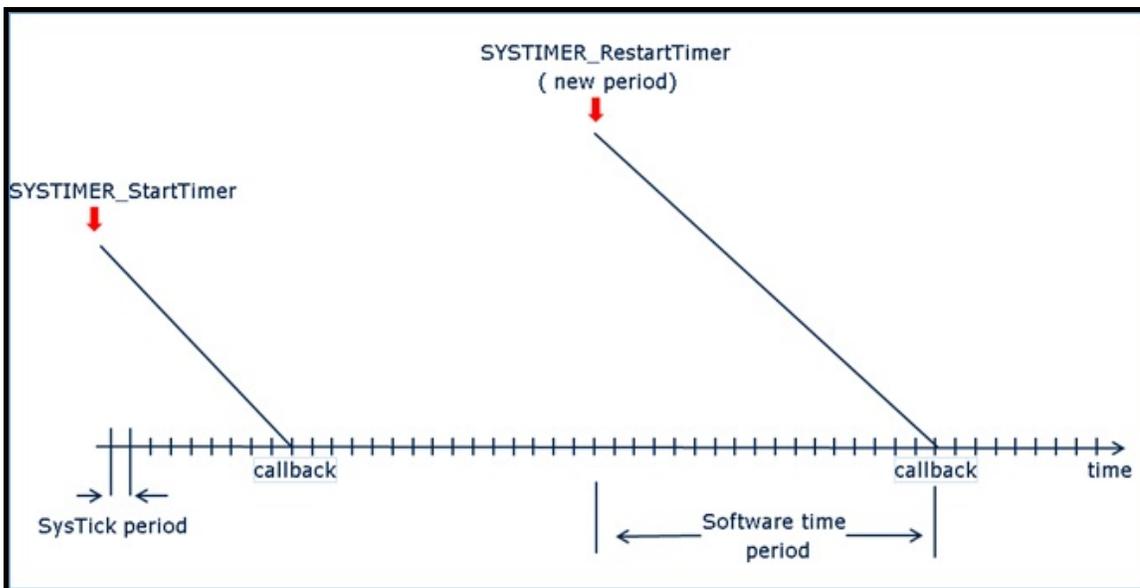


Figure 1 : Software timer behaviour in One Shot mode.

2. **Periodic:**

Periodic software timer will cycle continuously for the given time interval. Once the period expires, the callback is executed and the timer is restarted automatically.

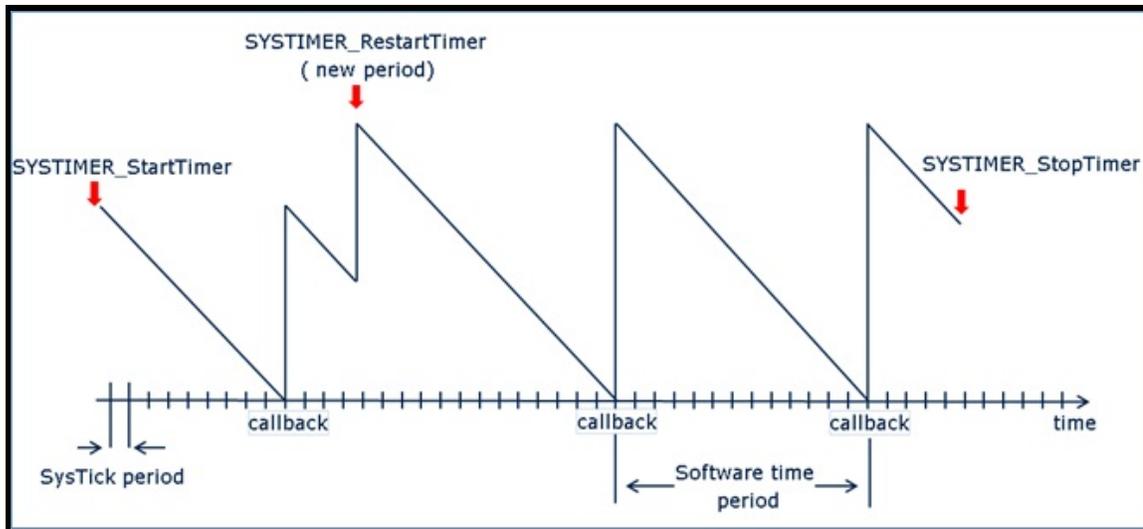


Figure 2 : Software timer behaviour in Periodic mode.

Note: All Software timers are using hardware SysTick timer for basic count. Hardware timer and software timers may not be in synchronization when software timer is started or restarted. This may cause first period executed by software timer will be less than expected. To overcome this limitation one extra period (corresponding count) is added for all software timers. This extra count will lead to, first period of every software timer will become greater than or equal to expected period value via API **SYSTIMER_CreateTimer()**. While all other periods(if asked for periodic software timers) will generate as expected.

Finite State Machine (FSM) for **SYSTIMER** States

The Figure 3 illustrates the FSM for **SYSTIMER** States and gives an outline of the software timer functionality with respect to user request transaction through **SYSTIMER** APP APIs.

- Creating a software timer using **SYSTIMER_CreateTimer()** API: All software timers are in UNINITIALIZED state after reset of device and creation of software timer by user using **SYSTIMER_CreateTimer** API changes the software timer states to STOPPED state.
- Running a software timer using **SYSTIMER_StartTimer()** API:

After creation of a software timer, user can start the software timer using `SYSTIMER_StartTimer` API and this changes the software timer states to `RUNNING` state.

- Stopping a software timer using `SYSTIMER_StopTimer()` API:
User can stop the software timer using `SYSTIMER_StopTimer` API and this changes the software timer states to `STOPPED` state.
- Restarting a software timer using `SYSTIMER_RestartTimer()` API when software timer is in `STOPPED` state:
User can restart the software timer (which is stopped previously) using `SYSTIMER_RestartTimer` API and this changes the software timer states to `RUNNING` state.
- Restarting a software timer using `SYSTIMER_RestartTimer()` API when software timer is in `RUNNING` state:
User can restart the software timer using `SYSTIMER_RestartTimer` API and this changes the software timer states to `RUNNING` state.
- Deleting a software timer using `SYSTIMER_DeleteTimer()` API when software timer is in `RUNNING` state:
User can delete the software timer using `SYSTIMER_DeleteTimer` API and this changes the software timer states to `UNINITIALIZED` state.
- Deleting a software timer using `SYSTIMER_DeleteTimer()` API when software timer is in `STOPPED` state:
User can delete the software timer (which is stopped previously) using `SYSTIMER_DeleteTimer` API and this changes the software timer states to `UNINITIALIZED` state.

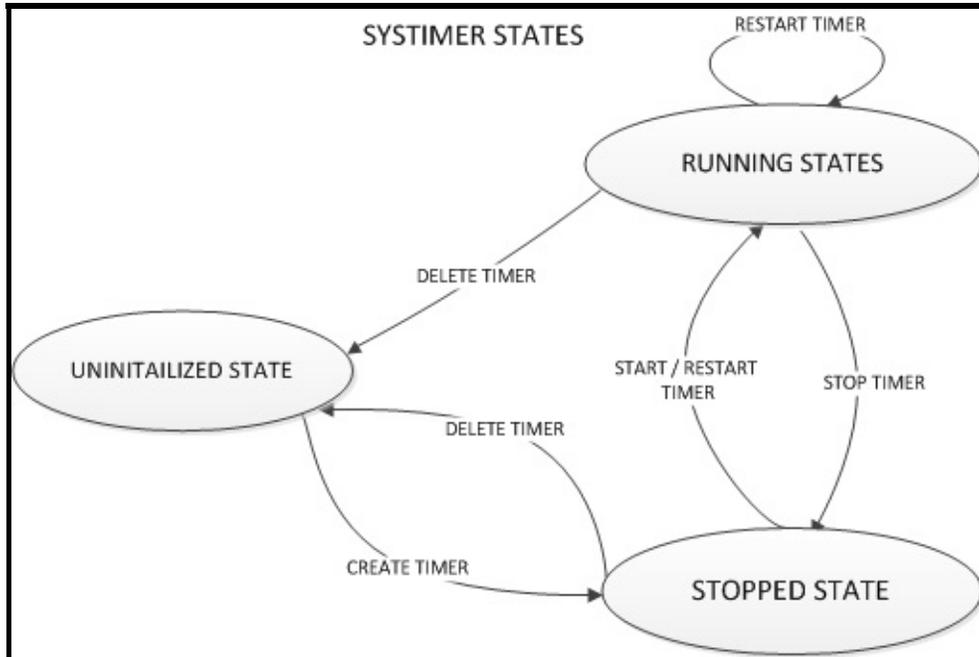


Figure 3 : FSM for SYSTIMER States

Hardware and Software connectivity of SYSTIMER APP

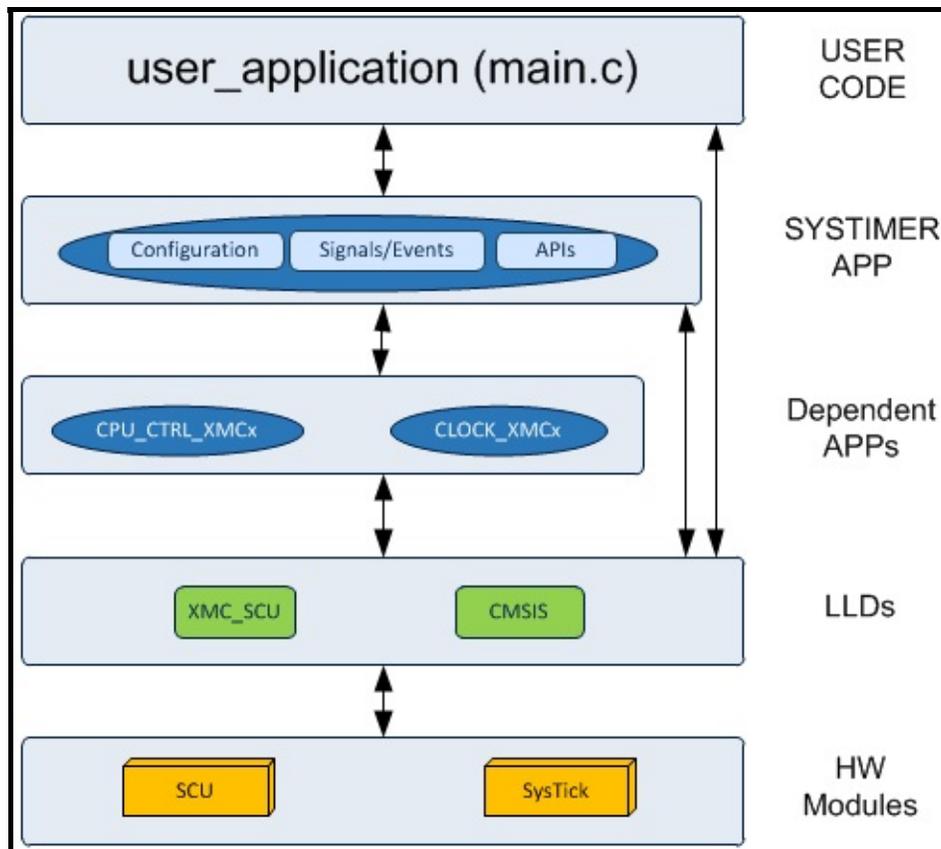


Figure 4 : Hardware and Software connectivity of **SYSTIMER APP.**

Figure 4 shows the APP structure in DAVE™. The APP configures the SysTick peripheral and clock using CLOCK_XMCxx APP. The LLD layer provides abstraction for SysTick hardware modules.

Supported Devices

The APP supports below devices:

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

References:

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

Limitations

None

SYSTIMER

Home

Architecture Description

Architecture Description

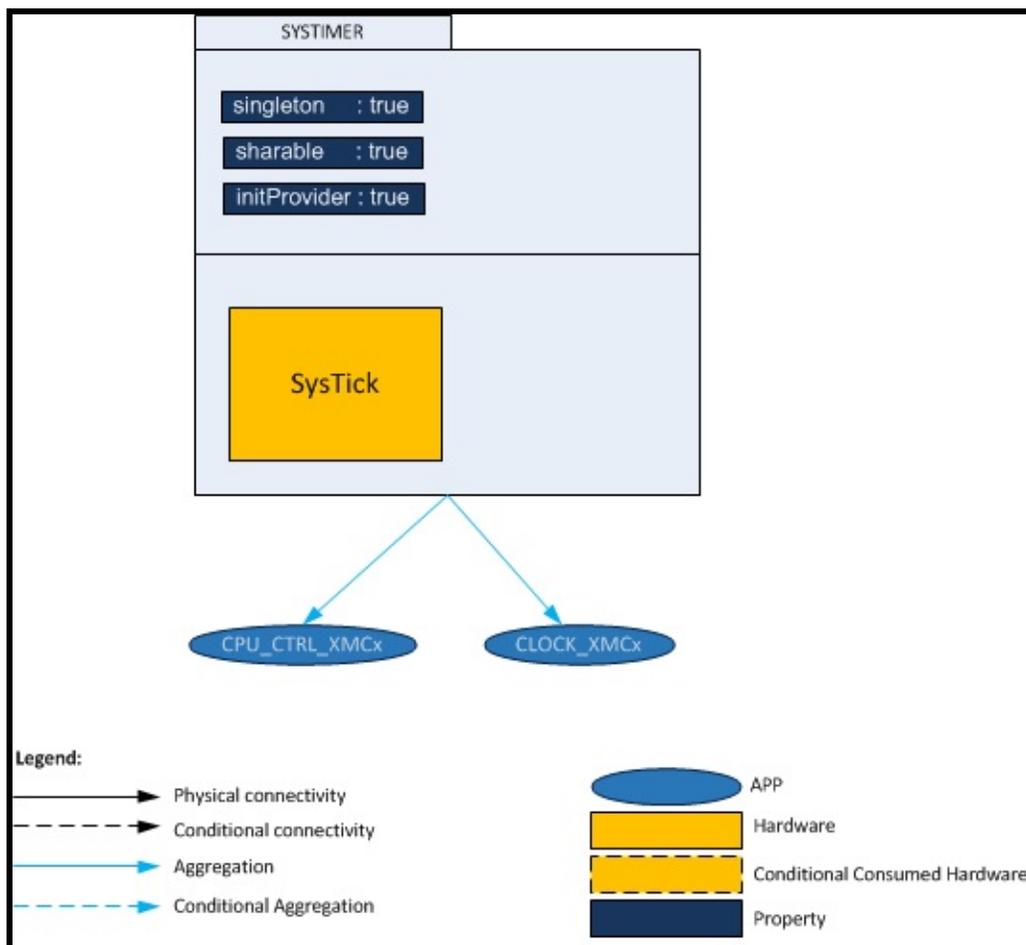


Figure 1 : Architecture of **SYSTIMER** APP

The above diagram represents the internal software architecture of the **SYSTIMER** APP. A **SYSTIMER** instance exists in a DAVE™ project with fixed attributes as shown. APP configures SysTick in the MCU. This in addition requires the consumption of the CPU_CTRL_XMCx (x = [1, 4]) APP for calculating preemption priority levels for SysTick

interrupt and CLOCK_XMCx (x = [1, 4]) APP for calculating SysTick interval value.

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 SYSTIMER_0 to SYSTIMER_CONFIG).

Signals

No signals are provided for external connection.



SYSTIMER

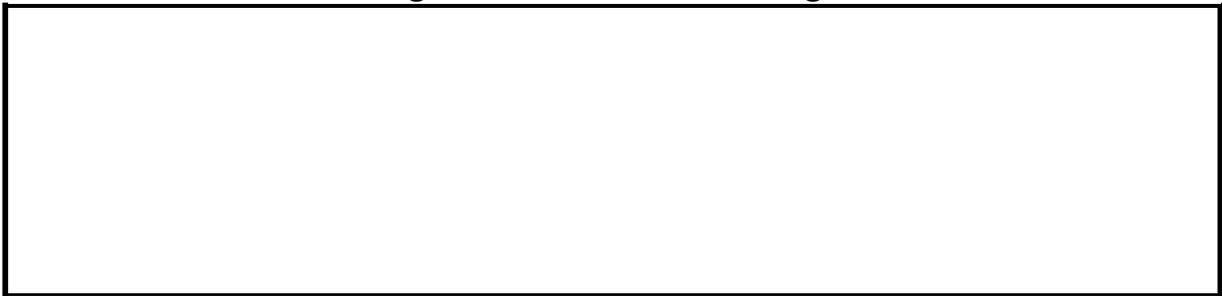
Home

APP Configuration Parameters

App Configuration Parameters

General Settings	Interrupt Settings
SysTick timer period [us]:	<input type="text" value="1000"/>
Number of software timers:	<input type="text" value="8"/>

Figure 1: General Settings



General Settings	Interrupt Settings		
SysTick Interrupt Settings			
Preemption priority	<input type="text" value="63"/>	Subpriority	<input type="text" value="0"/>

Figure 2: Interrupt Settings

--

--

SYSTIMER

[Home](#)

Enumerations

enum	SYSTIMER_STATUS { SYSTIMER_STATUS_SUCCESS SYSTIMER_STATUS_FAILURE }	This enumeration indicates status SYSTIMER . More...
enum	SYSTIMER_STATE { SYSTIMER_STATE_NOT_INITIAL = 0U, SYSTIMER_STATE_RUNNI SYSTIMER_STATE_STOPPED }	This enumeration defines possible timer state. More...
enum	SYSTIMER_MODE { SYSTIMER_MODE_ONE_SHOT : SYSTIMER_MODE_PERIODIC }	Enumeration values which describ timer types. More...
typedef enum	SYSTIMER_STATUS SYSTIMER_STATUS_t	This enumeration indicates status SYSTIMER .
typedef enum	SYSTIMER_STATE SYSTIMER_STATE_t	This enumeration defines possible timer state.
typedef enum	SYSTIMER_MODE SYSTIMER_MODE_t	Enumeration values which describ timer types.

Enumeration Type Documentation

enum `SYSTIMER_MODE`

Enumeration values which describes timer types.

Enumerator:

<code>SYSTIMER_MODE_ONE_SHOT</code>	timer type is one shot
<code>SYSTIMER_MODE_PERIODIC</code>	timer type is periodic

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

enum `SYSTIMER_STATE`

This enumeration defines possible timer state.

Enumerator:

<code>SYSTIMER_STATE_NOT_INITIALIZED</code>	The timer is in uninitialized state
<code>SYSTIMER_STATE_RUNNING</code>	The timer is in running state
<code>SYSTIMER_STATE_STOPPED</code>	The timer is in stop state

Definition at line [109](#) of file [SYSTIMER.h](#).

enum **SYSTIMER_STATUS**

This enumeration indicates status of **SYSTIMER**.

Enumerator:

SYSTIMER_STATUS_SUCCESS

Status Success if
initialization is successful

SYSTIMER_STATUS_FAILURE

Status Failure if initialization
is failed

Definition at line **100** of file **SYSTIMER.h**.

SYSTIMER

[Home](#)

Data structures

```
typedef void(* SYSTIMER_CALLBACK_t)(void *args)  
timer callback function pointer
```

```
typedef struct SYSTIMER SYSTIMER_t  
This structure contains pointer which is  
used to hold CPU instance handle and  
variables for priority group.
```

SYSTIMER

Home

Methods

SYSTIMER_STATUS_t	SYSTIMER_Init (SYSTIMER_t *handle) Initializes SYSTIMER APP.
uint32_t	SYSTIMER_CreateTimer (uint32_t period, SYSTIMER_MODE_t mode, SYSTIMER_CALLBACK_t callback, void *args) Creates a new software timer.
SYSTIMER_STATUS_t	SYSTIMER_StartTimer (uint32_t id) Starts the software timer.
SYSTIMER_STATUS_t	SYSTIMER_StopTimer (uint32_t id) Stops the software timer.
SYSTIMER_STATUS_t	SYSTIMER_RestartTimer (uint32_t id, uint32_t microsec) Function to modify the time interval and restart the timer for the new time interval.
SYSTIMER_STATUS_t	SYSTIMER_DeleteTimer (uint32_t id) Deletes the software timer from the timer list.
uint32_t	SYSTIMER_GetTime (void) Gives the current hardware SysTick time in microsecond since start of hardware SysTick timer.
uint32_t	SYSTIMER_GetTickCount (void) Gives the SysTick count.
SYSTIMER_STATE_t	SYSTIMER_GetTimerState (uint32_t id) Gives the current state of software timer.
DAVE_APP_VERSION_t	SYSTIMER_GetAppVersion (void)

Get **SYSTIMER** APP version.

Methods

Function Documentation

```
uint32_t SYSTIMER_CreateTimer ( uint32_t          period
                                SYSTIMER_MODE_t   mode
                                SYSTIMER_CALLBACK_t callback
                                void *            args
                                )
```

Creates a new software timer.

Parameters:

- period** timer period value in microseconds. Range: (SYSTIMER_TICK_PERIOD_US) to pow(2,32).
- mode** Mode of timer(ONE_SHOT/PERIODIC). Refer [SYSTIMER_MODE_t](#) for details.
- callback** Call back function of the timer(No Macros are allowed).
- args** Call back function parameter.

Returns:

uint32_t returns timer ID if timer created successfully otherwise returns 0 if timer creation failed. Range: 0 to 16, 0: Invalid timer ID, 1-16: Valid timer ID.

Description:

API for creating a new software timer instance. This also add created software timer to timer list.

Note :

1. This APP uses SysTick exception for controlling the timer list. Call back function registered through this function will be called in SysTick exception when the software timer is expired i.e the software timers callback is executed in the interrupt context.
2. Due to time at which software timer creation asked by user will not be in synchronize with Hardware SysTick timer, the count value used during creation of software timer will not

create starting/initial period same as expected value. It is decided to add one extra count(HW_TIMER_ADDITIONAL_CNT) with Software timer. Impact of this additional count(HW_TIMER_ADDITIONAL_CNT) is, first SW timer period(Initial one) is always equal to or more than expected/configured.

3. Callbacks are executed in round robin manner if more than one software timers are created with same period value. Last created software is having higher priority and its associated callback is executed first.

4. Avoid any call to wait, infinite while loop, blocking calls or creating software timer in ISR because their behavior could be corrupted when called from an ISR.

5. Software timers are based on 24-bit Hardware SysTick counters, so maximum counts can achieve is $\text{pow}(2,24) * (1/f\text{CPU}) * 1\text{E}6$, where fCPU is in hertz. Software timers are designed for times between microseconds and seconds. For longer times, application code need to ensure to take necessary action.

6. Software timer period value must be equal to SysTick Interval or integer multiple of a number with SysTick interval (i.e. SysTick Interval * n, where n is integer number, n can be 1,2,3,4... but n should not be fractional or float number). And also software timer period value should not be 0 or less than Hardware SysTick Interval.

Example Usage:

```
#include <DAVE.h>
#define ONESEC 1000000U
void LED_Toggle_EverySec(void)
{
    // Add user code here
}

int main(void)
```

```

{
    uint32_t TimerId;
    // ... Initializes APPs configuration ...
    DAVE_Init(); // SYSTIMER APP Initialized during DAVE Initialization
    // Create Software timer
    TimerId = (uint32_t)SYSTIMER_CreateTimer(ONESEC, SYSTIMER_MODE_PERIODIC, (void*)LED_Toggle_EverySec, NULL);
    if (TimerId != 0U)
    {
        //software timer is created successfully
        //Add user code here
    }
    else
    {
        // //software timer creation is failed
    }
    while (1)
    {
    }
    return (1);
}

```

Definition at line **390** of file **SYSTIMER.c**.

References **g_timer_tbl**, **SYSTIMER_MODE_ONE_SHOT**, **SYSTIMER_MODE_PERIODIC**, and **SYSTIMER_STATE_STOPPED**.

SYSTIMER_STATUS_t SYSTIMER_DeleteTimer (uint32_t id)

Deletes the software timer from the timer list.

Parameters:

id timer ID obtained from SYSTIMER_CreateTimer. Range : 1 to 16

Returns:

SYSTIMER_STATUS_t APP status. Refer [SYSTIMER_STATUS_t](#) for details.

Description:

API for deleting the created software timer instance from timer list.

Note : This API must be called after software timer is created using SYSTIMER_CreateTimer API with generated ID and enable XMC_ASSERT for better understanding of API behavioral in run time.

Example Usage:

```
#include <DAVE.h>
#define ONESEC 1000000U
void LED_Toggle_EverySec(void)
{
    // Add user code here
}

int main(void)
{
    uint32_t TimerId;
    SYSTIMER_STATUS_t status;
    // ... Initializes APPs configuration ...
    DAVE_Init(); // SYSTIMER APP Initialized during DAVE Initialization
    // Create Software timer
    TimerId = (uint32_t)SYSTIMER_CreateTimer(ONESEC, SYSTIMER_MODE_PERIODIC, (void*)LED_Toggle_EverySec, NULL);
    if (TimerId != 0U)
```

```

{
    //timer is created successfully, now start/
run software timer
    status = SYSTIMER_StartTimer(TimerId);
    if (status == SYSTIMER_STATUS_SUCCESS)
    {

        // User code

        status = SYSTIMER_StopTimer(TimerId);
        //User code

        if (status == SYSTIMER_STATUS_SUCCESS)
        {
            //User code

            status = SYSTIMER_DeleteTimer(TimerId);
            if (status == SYSTIMER_STATUS_SUCCESS)
            {
                // Software timer has deleted
            }
            else
            {
                // Error during software timer delete
operation
            }
        }
        else
        {
            // Error during software timer stop op
eration
        }
    }
    else
    {

```

```
        // Error during software timer start operation
    }
}
else
{
    // timer ID Can not be zero
}
// ... infinite loop ...
while (1)
{

}
return (1);
}
```

Definition at line **541** of file **SYSTIMER.c**.

References **g_timer_tbl**, **SYSTIMER_STATE_NOT_INITIALIZED**, **SYSTIMER_STATE_STOPPED**, **SYSTIMER_STATUS_FAILURE**, and **SYSTIMER_STATUS_SUCCESS**.

DAVE_APP_VERSION_t SYSTIMER_GetAppVersion (void)

Get **SYSTIMER** 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.

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

```

int main(void)
{
    DAVE_Init();
    DAVE_APP_VERSION_t systimer_version;
    systimer_version = SYSTIMER_GetAppVersion();
    if ((systimer_version.major == 4U) && (systimer
_version.minor == 1U))
    {
        // Add application code here
        while (1)
        {
        }
    }
    return(1);
}

```

Definition at line **332** of file **SYSTIMER.c**.

uint32_t SYSTIMER_GetTickCount (void)

Gives the SysTick count.

Returns:

uint32_t returns SysTick count. Range: 0 to pow(2,32).

Description:

API to get hardware SysTick counts since start of hardware SysTick timer.

Example Usage:

```

#include <DAVE.h>
#include <DAVE.h>
#define ONESEC 1000000U
void LED_Toggle_EverySec(void)

```

```

{
    // Add user code here
}

int main(void)
{
    uint32_t TimerId;
    uint32_t SysTick_Count;
    SYSTIMER_STATUS_t status;
    // ... Initializes APPs configuration ...
    DAVE_Init(); // SYSTIMER APP Initialized during DAVE Initialization
    // Create Software timer
    TimerId = (uint32_t)SYSTIMER_CreateTimer(ONESEC, SYSTIMER_MODE_PERIODIC, (void*)LED_Toggle_EverySec, NULL);
    if (TimerId != 0U)
    {
        //timer is created successfully, now start/run software timer
        status = SYSTIMER_StartTimer(TimerId);
        if (status == SYSTIMER_STATUS_SUCCESS)
        {
            // Add user code here

            SysTick_Count = SYSTIMER_GetTickCount();
            // Add user code here

        }
        else
        {
            // Error during software timer start operation
        }
    }
}

```

```
else
{
    // timer ID Can not be zero
}
// ... infinite loop ...
while (1)
{

}
return (1);
}
```

Definition at line **585** of file **SYSTIMER.c**.

uint32_t SYSTIMER_GetTime (void)

Gives the current hardware SysTick time in microsecond since start of hardware SysTick timer.

Returns:

uint32_t returns current SysTick time in microsecond. Range: (SYSTIMER_TICK_PERIOD_US) to pow(2,32).

Description:

API to get current hardware SysTick time in microsecond since start of hardware SysTick timer.

Example Usage:

```
#include <DAVE.h>
#define ONESEC 1000000U
void LED_Toggle_EverySec(void)
{
    // Add user code here
}
```

```

int main(void)
{
    uint32_t TimerId;
    uint32_t SysTick_Time;
    SYSTIMER_STATUS_t status;
    // ... Initializes APPs configuration ...
    DAVE_Init(); // SYSTIMER APP Initialized during DAVE Initialization
    // Create Software timer
    TimerId = (uint32_t)SYSTIMER_CreateTimer(ONESEC, SYSTIMER_MODE_PERIODIC, (void*)LED_Toggle_EverySec, NULL);
    if (TimerId != 0U)
    {
        //timer is created successfully, now start/run software timer
        status = SYSTIMER_StartTimer(TimerId);
        if (status == SYSTIMER_STATUS_SUCCESS)
        {
            // Add user code here

            SysTick_Time = SYSTIMER_GetTime();
            // Add user code here

        }
        else
        {
            // Error during software timer start operation
        }
    }
    else
    {
        // timer ID Can not be zero
    }
}

```

```
// ... infinite loop ...
while (1)
{

}
return (1);
}
```

Definition at line **577** of file **SYSTIMER.c**.

SYSTIMER_STATE_t SYSTIMER_GetTimerState (uint32_t id)

Gives the current state of software timer.

Parameters:

id timer ID obtained from SYSTIMER_CreateTimer. Range : 1 to 16

Returns:

SYSTIMER_STATE_t Software timer state. Refer **SYSTIMER_STATE_t** for details.

Description:

API to get current software timer state.

Example Usage:

```
#include <DAVE.h>
#define ONESEC 1000000U
#define NEW_INTERVAL (ONESEC * 10U)
void LED_Toggle_EverySec(void)
{
    // Add user code here
}
```

```

int main(void)
{
    uint32_t TimerId;
    SYSTIMER_STATUS_t status;
    SYSTIMER_STATE_t timer_state;
    // ... Initializes APPs configuration ...
    DAVE_Init(); // SYSTIMER APP Initialized during DAVE Initialization
    // Create Software timer
    TimerId = (uint32_t)SYSTIMER_CreateTimer(ONESEC, SYSTIMER_MODE_PERIODIC, (void*)LED_Toggle_EverySec, NULL);
    if (TimerId != 0U)
    {
        //timer is created successfully, now start/run software timer
        status = SYSTIMER_StartTimer(TimerId);
        timer_state = SYSTIMER_GetTimerState(TimerId); // use case scenario 1
        if (timer_state == SYSTIMER_STATE_RUNNING)
        {
            // software timer start operation is successful
            // Add user code here
        }
        else
        {
            // Error during software timer start operation
        }

        // Add user code here

        // user decided to change software interval, oops but user don't know the timer state
    }
}

```

```

    timer_state = SYSTIMER_GetTimerState(TimerI
d); // use case scenario 2
    if (timer_state == SYSTIMER_STATE_RUNNING)
    {
        status = SYSTIMER_StopTimer(TimerId);
        status = SYSTIMER_RestartTimer(TimerId,NE
W_INTERVAL);
        // Add user code here
    }
    else if (timer_state == SYSTIMER_STATE_STOP
PED)
    {
        status = SYSTIMER_RestartTimer(TimerId,NE
W_INTERVAL);
    }
    else if (timer_state == SYSTIMER_STATE_NOT_
INITIALIZED)
    {
        // user has already deleted this software
timer but need to recreate
        TimerId = (uint32_t)SYSTIMER_CreateTimer(N
EW_INTERVAL, SYSTIMER_MODE_PERIODIC, (void*)LED_Tog
gle_EverySec, NULL);
        status = SYSTIMER_StartTimer(TimerId);
        // Add user code here

    }
}
else
{
    // timer ID Can not be zero
}
// ... infinite loop ...
while (1)
{

}

```

```
    return (1);  
}
```

Definition at line **593** of file **SYSTIMER.c**.

References **g_timer_tbl**.

SYSTIMER_STATUS_t SYSTIMER_Init (SYSTIMER_t * handle)

Initializes **SYSTIMER** APP.

Parameters:

handle Pointer pointing to **SYSTIMER** APP data structure.
Refer **SYSTIMER_t** for details.

Returns:

SYSTIMER_STATUS_t APP status. Refer
SYSTIMER_STATUS_t for details.

Description:

Initializes the SysTick counter as per the SysTick interval specified by the user and start the SysTick counter. It also initializes global variables.

Example Usage:

```
#include <DAVE.h> //Declarations f  
rom DAVE Code Generation (includes SFR declar  
ation)  
  
int main(void)  
{  
    SYSTIMER_STATUS_t init_status;  
  
    init_status = (SYSTIMER_STATUS_t)SYSTIME
```

```

R_Init(&SYSTIMER_0); // Initialization of SYS
TIMER APP
    if (init_status == SYSTIMER_STATUS_SUCCE
SS)
    {
        // Add application code here
        while(1)
        {
        }
    }
    else
    {
        XMC_DEBUG("main: Application initializa
tion failed");
        while(1)
        {
        }
    }
    return (1);
}

```

Definition at line **346** of file **SYSTIMER.c**.

References **SYSTIMER::init_status**, **SYSTIMER_STATUS_FAILURE**, and **SYSTIMER_STATUS_SUCCESS**.

SYSTIMER_STATUS_t SYSTIMER_RestartTimer (uint32_t id, uint32_t microseconds)

Function to modify the time interval and restart the timer for the new time interval.

Parameters:

id ID of already created system timer. Range : 1 to 16

microsec new time interval. Range:
(SYSTIMER_TICK_PERIOD_US) to pow(2,32).

Returns:

SYSTIMER_STATUS_t APP status. Refer
SYSTIMER_STATUS_t for details.

Description:

API for restarting the created software timer instance with new time interval.

Note : This API must be called after software timer is created using SYSTIMER_CreateTimer API with generated ID and enable XMC_ASSERT for better understanding of API behavioral in run time.

Example Usage:

Demonstrate SYSTIMER_RestartTimer API

```
#include <DAVE.h>
#define ONESEC 1000000U
#define NEW_INTERVAL (ONESEC * 10U)
void LED_Toggle_EverySec(void)
{
    // Add user code here
}

int main(void)
{
    uint32_t TimerId;
    SYSTIMER_STATUS_t status;
    // ... Initializes APPs configuration ...
    DAVE_Init(); // SYSTIMER APP Initialized during DAVE Initialization
    // Create Software timer
    TimerId = (uint32_t)SYSTIMER_CreateTimer(ONESEC, SYSTIMER_MODE_PERIODIC, (void*)LED_Toggle_EverySec, NULL);
```

```

if (TimerId != 0U)
{
    //timer is created successfully
    // Start/Run Software timer
    status = SYSTIMER_StartTimer(TimerId);
    if (status == SYSTIMER_STATUS_SUCCESS)
    {

        // User code

        status = SYSTIMER_StopTimer(TimerId);
        //User code

        if (status == SYSTIMER_STATUS_SUCCESS)
        {
            //User code

            status = SYSTIMER_RestartTimer(TimerId,
NEW_INTERVAL);
            if (status == SYSTIMER_STATUS_SUCCESS)
            {
                // timer configured with the new time
interval and is running
            }
            else
            {
                // Error during software timer restar
t operation
            }
        }
        else
        {
            // Error during software timer stop op
eration
        }
    }
}

```

```

    }
    else
    {
        // Error during software timer start operation
    }
}
else
{
    // timer ID can not be zero
}
while (1)
{

}
return (1);
}

```

Definition at line **501** of file **SYSTIMER.c**.

References **g_timer_tbl**, **SYSTIMER_StartTimer()**, **SYSTIMER_STATE_NOT_INITIALIZED**, **SYSTIMER_STATE_STOPPED**, **SYSTIMER_STATUS_FAILURE**, **SYSTIMER_STATUS_SUCCESS**, and **SYSTIMER_StopTimer()**.

SYSTIMER_STATUS_t SYSTIMER_StartTimer (uint32_t id)

Starts the software timer.

Parameters:

id timer ID obtained from SYSTIMER_CreateTimer. Range : 1 to 16

Returns:

SYSTIMER_STATUS_t APP status. Refer **SYSTIMER_STATUS_t** for details.

Description:

API for starting a software timer instance.

Note : This API must be called after software timer is created using SYSTIMER_CreateTimer API with generated ID and enable XMC_ASSERT for better understanding of API behavioral in run time.

Example Usage:

```
#include <DAVE.h>
#define ONESEC 1000000U
void LED_Toggle_EverySec(void)
{
    // Add user code here
}

int main(void)
{
    uint32_t TimerId;
    SYSTIMER_STATUS_t status;
    // ... Initializes APPs configuration ...
    DAVE_Init(); // SYSTIMER APP Initialized during DAVE Initialization
    // Create Software timer
    TimerId = (uint32_t)SYSTIMER_CreateTimer(ONESEC, SYSTIMER_MODE_PERIODIC, (void*)LED_Toggle_EverySec, NULL);
    if (TimerId != 0U)
    {
        //timer is created successfully, now start/run software timer
        status = SYSTIMER_StartTimer(TimerId);
        if (status == SYSTIMER_STATUS_SUCCESS)
        {
```

```

        // Software timer is running
        // Add user code here
    }
    else
    {
        // Error during software timer start operation
    }
}
else
{
    // timer ID Can not be zero
}
// ... infinite loop ...
while (1)
{

}
return (1);
}

```

Definition at line **444** of file **SYSTIMER.c**.

References **g_timer_tbl**, **SYSTIMER_STATE_RUNNING**, **SYSTIMER_STATE_STOPPED**, **SYSTIMER_STATUS_FAILURE**, and **SYSTIMER_STATUS_SUCCESS**.

Referenced by **SYSTIMER_RestartTimer()**.

SYSTIMER_STATUS_t SYSTIMER_StopTimer (uint32_t id)

Stops the software timer.

Parameters:

id timer ID obtained from SYSTIMER_CreateTimer. Range : 1 to 16

Returns:

SYSTIMER_STATUS_t APP status. Refer [SYSTIMER_STATUS_t](#) for details.

Description:

API to stop created software timer instance.

Note : This API must be called after software timer is created using SYSTIMER_CreateTimer API with generated ID and enable XMC_ASSERT for better understanding of API behavioral in run time.

Example Usage:

```
#include <DAVE.h>
#define ONESEC 1000000U
void LED_Toggle_EverySec(void)
{
    // Add user code here
}

int main(void)
{
    uint32_t TimerId;
    SYSTIMER_STATUS_t status;
    // ... Initializes APPs configuration ...
    DAVE_Init(); // SYSTIMER APP Initialized during DAVE Initialization
    // Create Software timer
    TimerId = (uint32_t)SYSTIMER_CreateTimer(ONESEC, SYSTIMER_MODE_PERIODIC, (void*)LED_Toggle_EverySec, NULL);
    if (TimerId != 0U)
    {
        //timer is created successfully, now start/
```

```

run software timer
    status = SYSTIMER_StartTimer(TimerId);
    if (status == SYSTIMER_STATUS_SUCCESS)
    {
        // Software timer is running
        // Add user code here

        //stop the timer
        status = SYSTIMER_StopTimer(TimerId);
        if (status == SYSTIMER_STATUS_SUCCESS)
        {
            //Software timer has stopped
        }
        else
        {
            // Error during software timer stop op
eration
        }
    }
    else
    {
        // Error during software timer start oper
ation
    }
}
else
{
    // timer ID Can not be zero
}
// ... infinite loop ...
while (1)
{

}
return (1);

```

```
}
```

Definition at line **470** of file **SYSTIMER.c**.

References **g_timer_tbl**, **SYSTIMER_STATE_NOT_INITIALIZED**, **SYSTIMER_STATE_RUNNING**, **SYSTIMER_STATE_STOPPED**, **SYSTIMER_STATUS_FAILURE**, and **SYSTIMER_STATUS_SUCCESS**.

Referenced by **SYSTIMER_RestartTimer()**.

SYSTIMER

Home

Usage

Usage

This example demonstrates software timer callback registration and generation of software timer callback event at every one second time interval using **SYSTIMER** APP for XMC45 target device. Software timer callback event is indicated by LED toggling at every one second time interval.

Requirements

Boards Required: XMC4500 CPU Board.

Instantiate the required APPs

Drag an instance of **SYSTIMER** and DIGITAL_IO APPs. Update the fields in the GUI of these APPs with the following configuration.

Configure the APPs

SYSTIMER APP:

General Settings	Interrupt Settings
SysTick timer period [us]:	<input type="text" value="1000"/> 1
Number of software timers:	<input type="text" value="1"/> 2

Step1: Configure SysTick timer period value

Configure SysTick timer period [us] = 1000

Note: SysTick is reference to all software timers. SysTick events are generated for every one millisecond time interval.

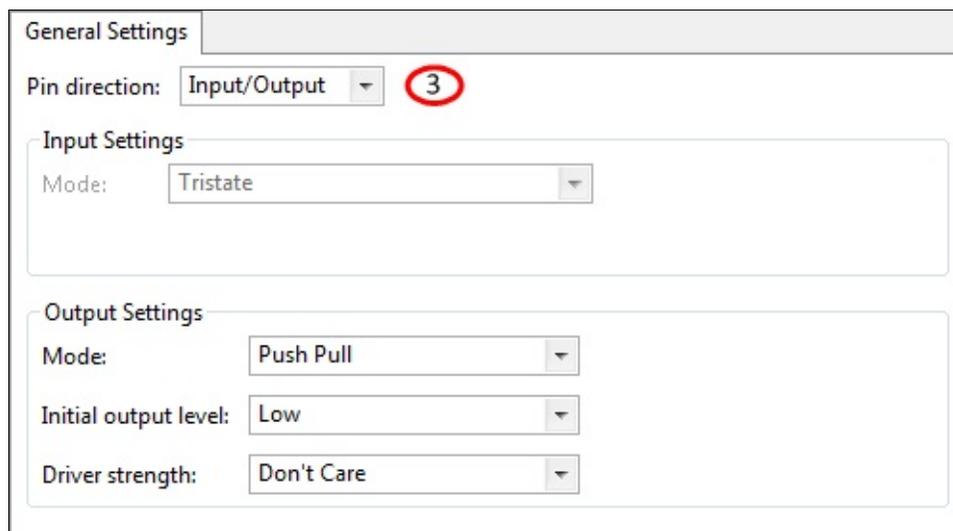
Step2: Configure required number of software timers

Configure Number of software timers = 1

Note: In main.c, need to call **SYSTIMER_CreateTimer()** API to configure the required time interval and user call back function.

SYSTIMER_StartTimer() API to start the software timer.

DIGITAL_IO APP:



General Settings

Pin direction: Input/Output **3**

Input Settings

Mode: Tristate

Output Settings

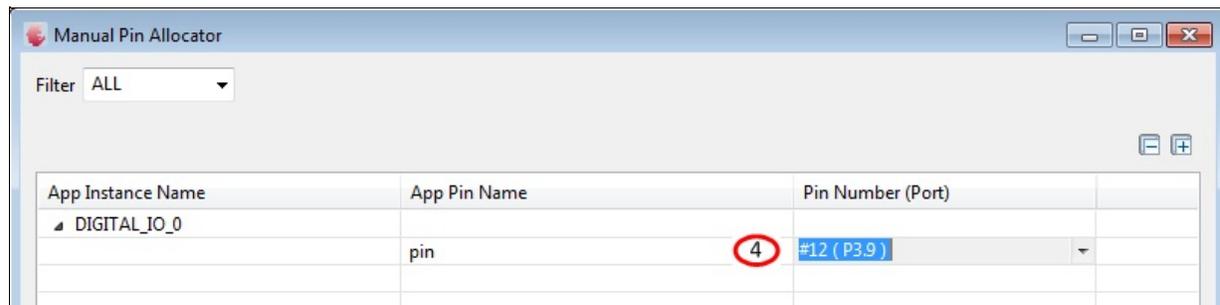
Mode: Push Pull

Initial output level: Low

Driver strength: Don't Care

Step3: Pin Direction setting

Set Pin direction = Input/Output



Manual Pin Allocator

Filter: ALL

App Instance Name	App Pin Name	Pin Number (Port)
└ DIGITAL_IO_0	pin 4	#12 (P3.9)

Step4: Pin Allocation

Allocate P3.9 for DIGITAL_IO

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>
#define ONESEC 1000000U

void LED_Toggle_EverySec(void)
{
    // LED Toggle for every second
    DIGITAL_IO_ToggleOutput(&DIGITAL_IO_0);
}
int main(void)
{
    uint32_t TimerId, status;
    // ... Initializes APPs configuration ...
    DAVE_Init(); // SYSTIMER APP Initialized during
    DAVE Initialization
    // Create Software Timer with one second time i
    nterval in order to generate software timer callba
    ck event at
    // every second
    TimerId = SYSTIMER_CreateTimer(ONESEC, SYSTIMER_
    MODE_PERIODIC, (void*)LED_Toggle_EverySec, NULL);
    if(TimerId != 0U)
    {
        //Timer is created successfully
        // Start/Run Software Timer
    }
}
```

```
status = SYSTIMER_StartTimer(TimerId);
if(status == SYSTIMER_STATUS_SUCCESS)
{
    // Timer is running
}
else
{
    // Error during software timer start operation
}
}
else
{
    // Timer ID Can not be zero
}
while(1)
{

}
return (1);
}
```

Build and Run the Project

Observation

- The LED connected to P3.9 of XMC45 device toggles at every second.



SYSTIMER

Home

Release History

Release History

--

--

SYSTIMER

Home

Data Structures

Data Structure Index

Data Fields

Data Structures

Here are the data structures with brief descriptions:

SYSTIMER

This structure contains pointer which is used to hold CPU instance handle and variables for priority group

--

SYSTIMER

[Home](#)

[Data Structures](#)

[Data Structure Index](#)

[Data Fields](#)

[Data Fields](#)

SYSTIMER Struct Reference

Detailed Description

This structure contains pointer which is used to hold CPU instance handle and variables for priority group.

Definition at line [146](#) of file [SYSTIMER.h](#).

```
#include <SYSTIMER.h>
```

Data Fields

bool **init_status**

Field Documentation

bool SYSTIMER::init_status

APP initialization status to ensure whether SYSTIMER_Init called or not

Definition at line **148** of file **SYSTIMER.h**.

Referenced by **SYSTIMER_Init()**.

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

- **SYSTIMER.h**



SYSTIMER

Home

Data Structures

Data Structure Index

Data Fields

Data Structure Index

S

S

SYSTIMER

S

SYSTIMER

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:

- `init_status` : [SYSTIMER](#)

--

SYSTIMER

Home			
Data Structures	Data Structure Index	Data Fields	
All	Variables		

- `init_status` : **SYSTIMER**

SYSTIMER

Home

File List

Globals

File List

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

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

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

--

SYSTIMER

[Home](#)

[File List](#)

[Globals](#)

[Variables](#)

SYSTIMER.c File Reference

Detailed Description

Date:

2015-10-08 This file is generated by DAVE, User modification to this file will be overwritten at the next code generation.

Definition in file **SYSTIMER.c**.

```
#include "systimer.h"
```

Functions

DAVE_APP_VERSION_t	SYSTIMER_GetAppVersion () Get SYSTIMER APP version.
SYSTIMER_STATUS_t	SYSTIMER_Init (SYSTIMER_t *handle) Initializes SYSTIMER APP.
uint32_t	SYSTIMER_CreateTimer (uint32_t period, SYSTIMER_MODE_t mode, SYSTIMER_CALLBACK_t callback, void *args) Creates a new software timer.
SYSTIMER_STATUS_t	SYSTIMER_StartTimer (uint32_t id) Starts the software timer.
SYSTIMER_STATUS_t	SYSTIMER_StopTimer (uint32_t id) Stops the software timer.
SYSTIMER_STATUS_t	SYSTIMER_RestartTimer (uint32_t id, uint32_t microsec) Function to modify the time interval and restart the timer for the new time interval.
SYSTIMER_STATUS_t	SYSTIMER_DeleteTimer (uint32_t id) Deletes the software timer from the timer list.
uint32_t	SYSTIMER_GetTime (void) Gives the current hardware SysTick time in microsecond since start of hardware SysTick timer.
uint32_t	SYSTIMER_GetTickCount (void) Gives the SysTick count.
SYSTIMER_STATE_t	SYSTIMER_GetTimerState (uint32_t id) Gives the current state of software timer.

Variables

SYSTIMER_OBJECT_t **g_timer_tbl** [SYSTIMER_CFG_MAX_TMR]

Variable Documentation

SYSTIMER_OBJECT_t g_timer_tbl[SYSTIMER_CFG_MAX_TMR]

Table which save timer control block.

Definition at line **104** of file **SYSTIMER.c**.

Referenced by **SYSTIMER_CreateTimer()**,
SYSTIMER_DeleteTimer(), **SYSTIMER_GetTimerState()**,
SYSTIMER_RestartTimer(), **SYSTIMER_StartTimer()**, and
SYSTIMER_StopTimer().

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



SYSTIMER

Home		
File List	Globals	

[Data Structures](#)

SYSTIMER.h File Reference

Detailed Description

Date:

2015-08-10

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 **SYSTIMER.h**.

```
#include "xmc_common.h" #include <DAVE_Common.h>
#include "systimer_conf.h"
#include "systimer_extern.h"
```

Data Structures

struct **SYSTIMER**

This structure contains pointer which is used to hold CPU instance handle and variables for priority group. [More...](#)

Typedefs

```
typedef void(* SYSTIMER_CALLBACK_t)(void *args)  
timer callback function pointer
```

```
typedef struct SYSTIMER SYSTIMER_t  
This structure contains pointer which is  
used to hold CPU instance handle and  
variables for priority group.
```

Functions

DAVE_APP_VERSION_t	SYSTIMER_GetAppVersion (void) Get SYSTIMER APP version.
SYSTIMER_STATUS_t	SYSTIMER_Init (SYSTIMER_t *h) Initializes SYSTIMER APP.
uint32_t	SYSTIMER_CreateTimer (uint32_t period, SYSTIMER_MODE_t mode, SYSTIMER_CALLBACK_t callback, void *args) Creates a new software timer.
SYSTIMER_STATUS_t	SYSTIMER_StartTimer (uint32_t id) Starts the software timer.
SYSTIMER_STATUS_t	SYSTIMER_StopTimer (uint32_t id) Stops the software timer.
SYSTIMER_STATUS_t	SYSTIMER_RestartTimer (uint32_t id, uint32_t microsec) Function to modify the time interval and restart the timer for the new time interval.
SYSTIMER_STATUS_t	SYSTIMER_DeleteTimer (uint32_t id) Deletes the software timer from the timer list.
uint32_t	SYSTIMER_GetTime (void) Gives the current hardware SysTick time in microsecond since start of hardware SysTick timer.
uint32_t	SYSTIMER_GetTickCount (void) Gives the SysTick count.
SYSTIMER_STATE_t	SYSTIMER_GetTimerState (uint32_t id) Gives the current state of software timer.

timer.

enum **SYSTIMER_STATUS** {
SYSTIMER_STATUS_SUCCESS
SYSTIMER_STATUS_FAILURE }
This enumeration indicates status **SYSTIMER**. More...

enum **SYSTIMER_STATE** {
SYSTIMER_STATE_NOT_INITIAL
= 0U, **SYSTIMER_STATE_RUNNING**,
SYSTIMER_STATE_STOPPED }
This enumeration defines possible timer state. More...

enum **SYSTIMER_MODE** {
SYSTIMER_MODE_ONE_SHOT :
SYSTIMER_MODE_PERIODIC }
Enumeration values which describe timer types. More...

typedef enum **SYSTIMER_STATUS** **SYSTIMER_STATUS_t**
This enumeration indicates status **SYSTIMER**.

typedef enum **SYSTIMER_STATE** **SYSTIMER_STATE_t**
This enumeration defines possible timer state.

typedef enum **SYSTIMER_MODE** **SYSTIMER_MODE_t**
Enumeration values which describe timer types.

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



SYSTIMER

Home					
File List		Globals			
All	Functions	Variables	Typedefs	Enumerations	Enumerator
g	s				

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

- g -

- [g_timer_tbl](#) : [SYSTIMER.c](#)

- s -

- [SYSTIMER_CALLBACK_t](#) : [SYSTIMER.h](#)
- [SYSTIMER_CreateTimer\(\)](#) : [SYSTIMER.c](#) , [SYSTIMER.h](#)
- [SYSTIMER_DeleteTimer\(\)](#) : [SYSTIMER.c](#) , [SYSTIMER.h](#)
- [SYSTIMER_GetAppVersion\(\)](#) : [SYSTIMER.c](#) , [SYSTIMER.h](#)
- [SYSTIMER_GetTickCount\(\)](#) : [SYSTIMER.c](#) , [SYSTIMER.h](#)
- [SYSTIMER_GetTime\(\)](#) : [SYSTIMER.c](#) , [SYSTIMER.h](#)
- [SYSTIMER_GetTimerState\(\)](#) : [SYSTIMER.c](#) , [SYSTIMER.h](#)
- [SYSTIMER_Init\(\)](#) : [SYSTIMER.c](#) , [SYSTIMER.h](#)
- [SYSTIMER_MODE](#) : [SYSTIMER.h](#)
- [SYSTIMER_MODE_ONE_SHOT](#) : [SYSTIMER.h](#)
- [SYSTIMER_MODE_PERIODIC](#) : [SYSTIMER.h](#)
- [SYSTIMER_MODE_t](#) : [SYSTIMER.h](#)
- [SYSTIMER_RestartTimer\(\)](#) : [SYSTIMER.c](#) , [SYSTIMER.h](#)
- [SYSTIMER_StartTimer\(\)](#) : [SYSTIMER.c](#) , [SYSTIMER.h](#)
- [SYSTIMER_STATE](#) : [SYSTIMER.h](#)
- [SYSTIMER_STATE_NOT_INITIALIZED](#) : [SYSTIMER.h](#)
- [SYSTIMER_STATE_RUNNING](#) : [SYSTIMER.h](#)
- [SYSTIMER_STATE_STOPPED](#) : [SYSTIMER.h](#)
- [SYSTIMER_STATE_t](#) : [SYSTIMER.h](#)
- [SYSTIMER_STATUS](#) : [SYSTIMER.h](#)

- SYSTIMER_STATUS_FAILURE : **SYSTIMER.h**
 - SYSTIMER_STATUS_SUCCESS : **SYSTIMER.h**
 - SYSTIMER_STATUS_t : **SYSTIMER.h**
 - SYSTIMER_StopTimer() : **SYSTIMER.h** , **SYSTIMER.c**
 - SYSTIMER_t : **SYSTIMER.h**
-
-

SYSTIMER

Home					
File List	Globals				
All	Functions	Variables	Typedefs	Enumerations	Enumerator

- SYSTIMER_CreateTimer() : **SYSTIMER.c** , **SYSTIMER.h**
- SYSTIMER_DeleteTimer() : **SYSTIMER.h** , **SYSTIMER.c**
- SYSTIMER_GetAppVersion() : **SYSTIMER.c** , **SYSTIMER.h**
- SYSTIMER_GetTickCount() : **SYSTIMER.h** , **SYSTIMER.c**
- SYSTIMER_GetTime() : **SYSTIMER.c** , **SYSTIMER.h**
- SYSTIMER_GetTimerState() : **SYSTIMER.c** , **SYSTIMER.h**
- SYSTIMER_Init() : **SYSTIMER.c** , **SYSTIMER.h**
- SYSTIMER_RestartTimer() : **SYSTIMER.c** , **SYSTIMER.h**
- SYSTIMER_StartTimer() : **SYSTIMER.h** , **SYSTIMER.c**
- SYSTIMER_StopTimer() : **SYSTIMER.h** , **SYSTIMER.c**



SYSTIMER

Home					
File List	Globals				
All	Functions	Variables	Typedefs	Enumerations	Enumerator

- `g_timer_tbl` : **SYSTIMER.c**



SYSTIMER

Home					
File List	Globals				
All	Functions	Variables	Typedefs	Enumerations	Enumerator

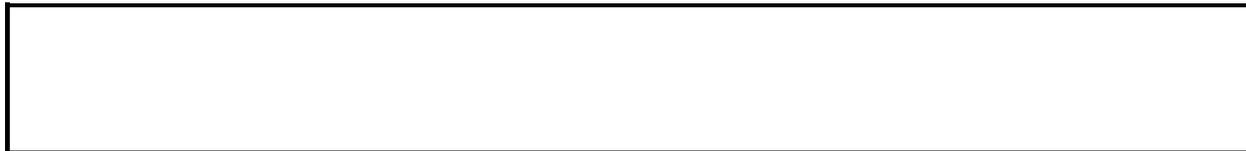
- SYSTIMER_CALLBACK_t: [SYSTIMER.h](#)
- SYSTIMER_MODE_t: [SYSTIMER.h](#)
- SYSTIMER_STATE_t: [SYSTIMER.h](#)
- SYSTIMER_STATUS_t: [SYSTIMER.h](#)
- SYSTIMER_t: [SYSTIMER.h](#)



SYSTIMER

Home					
File List	Globals				
All	Functions	Variables	Typedefs	Enumerations	Enumerator

- SYSTIMER_MODE : [SYSTIMER.h](#)
- SYSTIMER_STATE : [SYSTIMER.h](#)
- SYSTIMER_STATUS : [SYSTIMER.h](#)



SYSTIMER

Home					
File List	Globals				
All	Functions	Variables	Typedefs	Enumerations	Enumerator

- SYSTIMER_MODE_ONE_SHOT : [SYSTIMER.h](#)
- SYSTIMER_MODE_PERIODIC : [SYSTIMER.h](#)
- SYSTIMER_STATE_NOT_INITIALIZED : [SYSTIMER.h](#)
- SYSTIMER_STATE_RUNNING : [SYSTIMER.h](#)
- SYSTIMER_STATE_STOPPED : [SYSTIMER.h](#)
- SYSTIMER_STATUS_FAILURE : [SYSTIMER.h](#)
- SYSTIMER_STATUS_SUCCESS : [SYSTIMER.h](#)



SYSTIMER

Home	
File List	Globals

SYSTIMER.h

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

```
00001
00067 /*****
*****
*****
00068  * HEADER FILES
00069  *****/
00070
00071 #ifndef SYSTIMER_H
00072 #define SYSTIMER_H
00073
00074 #include "xmc_common.h"
00075 #include <DAVE_Common.h>
00076 #include "systimer_conf.h"
00077
00078 /*****
*****
*****
00079  * MACROS
00080  *****/
00081
00082 #if (!(XMC_LIB_MAJOR_VERSION == 2U) && \
00083      (XMC_LIB_MINOR_VERSION >= 0U) && \
00084      (XMC_LIB_PATCH_VERSION >= 0U))
```

```

00085 #error "SYSTIMER requires XMC Peripheral Lib
rary v2.0.0 or higher"
00086 #endif
00087
00088 /*****
*****
*****
00089  * ENUMS
00090  ****
*****
*****
***** /
00091
00100 typedef enum SYSTIMER_STATUS
00101 {
00102     SYSTIMER_STATUS_SUCCESS = 0U,
00103     SYSTIMER_STATUS_FAILURE
00104 } SYSTIMER_STATUS_t;
00105
00109 typedef enum SYSTIMER_STATE
00110 {
00111     SYSTIMER_STATE_NOT_INITIALIZED = 0U,
00112     SYSTIMER_STATE_RUNNING,
00113     SYSTIMER_STATE_STOPPED
00114 } SYSTIMER_STATE_t;
00115
00119 typedef enum SYSTIMER_MODE
00120 {
00121     SYSTIMER_MODE_ONE_SHOT = 0U,
00122     SYSTIMER_MODE_PERIODIC
00123 } SYSTIMER_MODE_t;
00124
00129 /*****
*****
*****
00130  * DATA STRUCTURES
00131  ****
*****
*****

```

```

*****/
00140 typedef void (*SYSTIMER_CALLBACK_t)(void *ar
gs);
00141
00146 typedef struct SYSTIMER
00147 {
00148     bool init_status;
00149 } SYSTIMER_t;
00154 /*****
*****
*****
00155     * API Prototypes
00156     *****/
00157
00158 /* Support for C++ codebase */
00159 #ifdef __cplusplus
00160 extern "C" {
00161 #endif
00162
00195 DAVE_APP_VERSION_t SYSTIMER_GetAppVersion(vo
id);
00196
00233 SYSTIMER_STATUS_t SYSTIMER_Init(SYSTIMER_t
*handle);
00234
00302 uint32_t SYSTIMER_CreateTimer
00303 (
00304     uint32_t period,
00305     SYSTIMER_MODE_t mode,
00306     SYSTIMER_CALLBACK_t callback,
00307     void *args
00308 );
00309
00364 SYSTIMER_STATUS_t SYSTIMER_StartTimer(uint32
_t id);

```

```
00365
00434 SYSTIMER_STATUS_t SYSTIMER_StopTimer(uint32_
t id);
00435
00518 SYSTIMER_STATUS_t SYSTIMER_RestartTimer(uint
32_t id, uint32_t microsec);
00519
00600 SYSTIMER_STATUS_t SYSTIMER_DeleteTimer(uint3
2_t id);
00601
00659 uint32_t SYSTIMER_GetTime(void);
00660
00721 uint32_t SYSTIMER_GetTickCount(void);
00722
00803 SYSTIMER_STATE_t SYSTIMER_GetTimerState(uint
32_t id);
00804
00809 /* Support for C++ codebase */
00810 #ifdef __cplusplus
00811 }
00812 #endif
00813
00814 /* Inclusion of APP extern file */
00815 #include "systimer_extern.h"
00816
00817 #endif /* SYSTIMER_H */
```



SYSTIMER

Home	
File List	Globals

SYSTIMER.c

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

```
00001
00071 /* ****
****
****
00072 *  HEADER FILES
00073 *  ****
****
**** */
00074
00075 /* Included to access APP data structure, fu
nctions & enumerations */
00076 #include "systimer.h"
00077
00078 /* ****
****
****
00079 *  MACROS
00080 *  ****
****
**** */
00081
00082 #define HW_TIMER_ADDITIONAL_CNT (1U)
00083
00084 /* ****
****
****
00085 *  LOCAL DATA
```

```

00086  ****
00087  ****
00088  ****
00089  ****
00090  ****
00091  ****
00092  ****
00093  ****
00094  ****
00095  ****
00096  ****
00097  ****
00098  ****
00099  ****
00100  ****
00101  ****
00102  ****
00103  ****
00104  ****
00105  ****
00106  ****
00107  ****
00108  ****
00109  ****
00110  ****
00111  ****
00112  ****
00113  ****
00114  ****
00115  ****
00116  ****
00117  ****

```

```

*****
*****/
00118
00119 /*
00120  * This function is called to insert a timer
    into the timer list.
00121  */
00122 static void SYSTIMER_lInsertTimerList(uint32
_t tbl_index);
00123
00124 /*
00125  * This function is called to remove a timer
    from the timer list.
00126  */
00127 static void SYSTIMER_lRemoveTimerList(uint32
_t tbl_index);
00128
00129 /*
00130  * Handler function called from SysTick eve
nt handler.
00131  */
00132 static void SYSTIMER_lTimerHandler(void);
00133
00134 /*
00135  * SysTick handler which is the main interr
upt service routine to service the
00136  * system timer's configured
00137  */
00138 void SysTick_Handler(void);
00139
00140 /*****
*****
*****
00141  * API IMPLEMENTATION
00142  *****/
*****
*****/

```

```

00143 /*
00144  * This function is called to insert a timer
    into the timer list.
00145  */
00146 static void SYSTIMER_lInsertTimerList(uint32
_t tbl_index)
00147 {
00148     SYSTIMER_OBJECT_t *object_ptr;
00149     int32_t delta_ticks;
00150     int32_t timer_count;
00151     bool found_flag = false;
00152     /* Get timer time */
00153     timer_count = (int32_t)g_timer_tbl[tbl_index].count;
00154     /* Check if Timer list is NULL */
00155     if (NULL == g_timer_list)
00156     {
00157         /* Set this as first Timer */
00158         g_timer_list = &g_timer_tbl[tbl_index];
00159     }
00160     /* If not, find the correct place, and insert the
specified timer */
00161     else
00162     {
00163         object_ptr = g_timer_list;
00164         /* Get timer tick */
00165         delta_ticks = timer_count;
00166         /* Find correct place for inserting the
timer */
00167         while ((NULL != object_ptr) && (false ==
found_flag))
00168         {
00169             /* Get timer Count Difference */
00170             delta_ticks -= (int32_t)object_ptr->count;
00171             /* Check for delta ticks < 0 */
00172             if (delta_ticks <= 0)

```

```

00173     {
00174         /* Check If head item */
00175         if (NULL != object_ptr->prev)
00176         {
00177             /* If Insert to list */
00178             object_ptr->prev->next = &g_timer_
tbl[tbl_index];
00179             g_timer_tbl[tbl_index].prev = obje
ct_ptr->prev;
00180             g_timer_tbl[tbl_index].next = obje
ct_ptr;
00181             object_ptr->prev = &g_timer_tbl[tb
l_index];
00182         }
00183         else
00184         {
00185             /* Set Timer as first item */
00186             g_timer_tbl[tbl_index].next = g_ti
mer_list;
00187             g_timer_list->prev = &g_timer_tbl[
tbl_index];
00188             g_timer_list = &g_timer_tbl[tbl_in
dex];
00189         }
00190         g_timer_tbl[tbl_index].count = g_tim
er_tbl[tbl_index].next->count + (uint32_t)delta_ti
cks;
00191         g_timer_tbl[tbl_index].next->count
 -= g_timer_tbl[tbl_index].count;
00192         found_flag = true;
00193     }
00194     /* Check for last item in list */
00195     else
00196     {
00197         if ((delta_ticks > 0) && (NULL == ob
ject_ptr->next))
00198         {

```

```

00199         /* Yes, insert into */
00200         g_timer_tbl[tbl_index].prev = obje
ct_ptr;
00201         object_ptr->next = &g_timer_tbl[tb
l_index];
00202         g_timer_tbl[tbl_index].count = (ui
nt32_t)delta_ticks;
00203         found_flag = true;
00204     }
00205 }
00206     /* Get the next item in timer list */
00207     object_ptr = object_ptr->next;
00208 }
00209 }
00210 }
00211
00212 /*
00213  * This function is called to remove a timer
from the timer list.
00214  */
00215 static void SYSTIMER_lRemoveTimerList(uint32
_t tbl_index)
00216 {
00217     SYSTIMER_OBJECT_t *object_ptr;
00218     object_ptr = &g_timer_tbl[tbl_index];
00219     /* Check whether only one timer available
*/
00220     if ((NULL == object_ptr->prev) && (NULL ==
object_ptr->next ))
00221     {
00222         /* set timer list as NULL */
00223         g_timer_list = NULL;
00224     }
00225     /* Check if the first item in timer list */

00226     else if (NULL == object_ptr->prev)
00227     {

```

```

00228     /* Remove timer from list, and reset timer list */
00229     g_timer_list = object_ptr->next;
00230     g_timer_list->prev = NULL;
00231     g_timer_list->count += object_ptr->count
;
00232     object_ptr->next = NULL;
00233 }
00234 /* Check if the last item in timer list */
00235 else if (NULL == object_ptr->next)
00236 {
00237     /* Remove timer from list */
00238     object_ptr->prev->next = NULL;
00239     object_ptr->prev = NULL;
00240 }
00241 else
00242 {
00243     /* Remove timer from list */
00244     object_ptr->prev->next = object_ptr->next;
00245     object_ptr->next->prev = object_ptr->prev;
00246     object_ptr->next->count += object_ptr->count;
00247     object_ptr->next = NULL;
00248     object_ptr->prev = NULL;
00249 }
00250 }
00251
00252 /*
00253 * Handler function called from SysTick event handler.
00254 */
00255 static void SYSTIMER_lTimerHandler(void)
00256 {
00257     SYSTIMER_OBJECT_t *object_ptr;
00258     /* Get first item of timer list */

```

```

00259     object_ptr = g_timer_list;
00260     while ((NULL != object_ptr) && (0U == object_ptr->count))
00261     {
00262         if (true == object_ptr->delete_swtmr)
00263         {
00264             /* Yes, remove this timer from timer list */
00265             SYSTIMER_lRemoveTimerList((uint32_t)object_ptr->id);
00266             /* Set timer status as SYSTIMER_STATE_NOT_INITIALIZED */
00267             object_ptr->state = SYSTIMER_STATE_NOT_INITIALIZED;
00268             /* Release resource which are hold by this timer */
00269             g_timer_tracker &= ~(1U << object_ptr->id);
00270         }
00271         /* Check whether timer is a one shot timer */
00272         else if (SYSTIMER_MODE_ONE_SHOT == object_ptr->mode)
00273         {
00274             /* Yes, remove this timer from timer list */
00275             SYSTIMER_lRemoveTimerList((uint32_t)object_ptr->id);
00276             /* Set timer status as SYSTIMER_STATE_STOPPED */
00277             object_ptr->state = SYSTIMER_STATE_STOPPED;
00278             /* Call timer callback function */
00279             (object_ptr->callback)(object_ptr->args);
00280         }
00281         /* Check whether timer is periodic timer

```

```

    */
00282     else if (SYSTIMER_MODE_PERIODIC == objec
t_ptr->mode)
00283     {
00284         /* Yes, remove this timer from timer l
ist */
00285         SYSTIMER_lRemoveTimerList((uint32_t)ob
ject_ptr->id);
00286         /* Reset timer tick */
00287         object_ptr->count = object_ptr->reload
;
00288         /* Insert timer into timer list */
00289         SYSTIMER_lInsertTimerList((uint32_t)ob
ject_ptr->id);
00290         /* Call timer callback function */
00291         (object_ptr->callback)(object_ptr->arg
s);
00292     }
00293     else
00294     {
00295         break;
00296     }
00297     /* Get first item of timer list */
00298     object_ptr = g_timer_list;
00299 }
00300 }
00301
00302 /*
00303  * SysTick Event Handler.
00304  */
00305 void SysTick_Handler(void)
00306 {
00307     SYSTIMER_OBJECT_t *object_ptr;
00308     object_ptr = g_timer_list;
00309     g_systick_count++;
00310
00311     if (NULL != object_ptr)

```

```

00312     {
00313         if (object_ptr->count > 1UL)
00314         {
00315             object_ptr->count--;
00316         }
00317         else
00318         {
00319             object_ptr->count = 0U;
00320             SYSTIMER_1TimerHandler();
00321         }
00322     }
00323 }
00324
00329 /*
00330  * Function to retrieve the version of the S
00331  * YSTIMER APP.
00332  */
00332 DAVE_APP_VERSION_t SYSTIMER_GetAppVersion()
00333 {
00334     DAVE_APP_VERSION_t version;
00335
00336     version.major = (uint8_t)SYSTIMER_MAJOR_VE
00337     RSION;
00337     version.minor = (uint8_t)SYSTIMER_MINOR_VE
00338     RSION;
00338     version.patch = (uint8_t)SYSTIMER_PATCH_VE
00339     RSION;
00339
00340     return (version);
00341 }
00342
00343 /*
00344  * Initialization function which initializes
00345  * the SYSTIMER APP, configures SysTick timer and Sy
00346  * sTick exception.
00345  */
00346 SYSTIMER_STATUS_t SYSTIMER_Init(SYSTIMER_t *

```

```

handle)
00347 {
00348     SYSTIMER_STATUS_t status = SYSTIMER_STATUS
_SUCCESS;
00349
00350     XMC_ASSERT("SYSTIMER_Init: SYSTIMER APP ha
ndle pointer uninitialized", (handle != NULL));
00351
00352     /* Check APP initialization status to ensu
re whether SYSTIMER_Init called or not, initialize
SYSTIMER if
00353     * SYSTIMER_Init called first time.
00354     */
00355     if (false == handle->init_status)
00356     {
00357         /* Initialize the header of the list */
00358         g_timer_list = NULL;
00359         /* Initialize SysTick timer */
00360         status = (SYSTIMER_STATUS_t)SysTick_Conf
ig((uint32_t)(SYSTIMER_SYSTICK_CLOCK * SYSTIMER_TI
CK_PERIOD));
00361
00362         if (SYSTIMER_STATUS_FAILURE == status)
00363         {
00364             XMC_DEBUG("SYSTIMER_Init: Timer reload
value out of range");
00365         }
00366         else
00367         {
00368             #if (UC_FAMILY == XMC4)
00369                 /* setting of First SW Timer period is
always and subpriority value for XMC4000 devices
*/
00370                 NVIC_SetPriority(SysTick_IRQn, NVIC_En
codePriority(
00371                 NVIC_GetPriorityGrouping(), SYSTIMER_P
RRIORITY, SYSTIMER_SUBPRIORITY));

```

```

00372 #elif (UC_FAMILY == XMC1)
00373     /* setting of priority value for XMC10
00 devices */
00374     NVIC_SetPriority(SysTick_IRQn, SYSTIME
R_PRIORITY);
00375 #endif
00376     g_timer_tracker = 0U;
00377     /* Update the Initialization status of
the SYSTIMER APP instance */
00378     handle->init_status = true;
00379     status = SYSTIMER_STATUS_SUCCESS;
00380     }
00381 }
00382
00383 return (status);
00384 }
00385
00386 /*
00387 * API for creating a new software Timer in
stance.
00388 */
00389 uint32_t SYSTIMER_CreateTimer
00390 (
00391     uint32_t period,
00392     SYSTIMER_MODE_t mode,
00393     SYSTIMER_CALLBACK_t callback,
00394     void *args
00395 )
00396 {
00397     uint32_t id = 0U;
00398     uint32_t count = 0U;
00399     uint32_t period_ratio = 0U;
00400
00401     XMC_ASSERT("SYSTIMER_CreateTimer: Timer cr
eation failure due to invalid period value",
00402         ((period >= SYSTIMER_TICK_PERIOD
_US) && (period > 0U) && (period <= 0xFFFFFFFFU)))

```

```

;
00403   XMC_ASSERT("SYSTIMER_CreateTimer: Timer cr
eation failure due to invalid timer mode",
00404           ((SYSTIMER_MODE_ONE_SHOT == mode
) || (SYSTIMER_MODE_PERIODIC == mode)));
00405   XMC_ASSERT("SYSTIMER_CreateTimer: Can not
create software without user callback", (NULL != c
allback));
00406
00407   if (period < SYSTIMER_TICK_PERIOD_US)
00408   {
00409       id = 0U;
00410   }
00411   else
00412   {
00413       for (count = 0U; count < SYSTIMER_CFG_MA
X_TMR; count++)
00414       {
00415           /* Check for free timer ID */
00416           if (0U == (g_timer_tracker & (1U << co
unt)))
00417           {
00418               /* If yes, assign ID to this timer */
00419
00420               g_timer_tracker |= (1U << count);
00421               /* Initialize the timer as per input
values */
00422               g_timer_tbl[count].id      = count;
00423               g_timer_tbl[count].mode    = mode;
00424               g_timer_tbl[count].state   = SYSTIMER
_STATE_STOPPED;
00425               period_ratio = (uint32_t)(period / S
YSTIMER_TICK_PERIOD_US);
00426               g_timer_tbl[count].count  = (period_
ratio + HW_TIMER_ADDITIONAL_CNT);
00427               g_timer_tbl[count].reload = period_
ratio;

```

```

00427         g_timer_tbl[count].callback = callba
ck;
00428         g_timer_tbl[count].args = args;
00429         g_timer_tbl[count].prev     = NULL;
00430         g_timer_tbl[count].next     = NULL;
00431         id = count + 1U;
00432         break;
00433     }
00434 }
00435
00436 }
00437
00438 return (id);
00439 }
00440
00441 /*
00442  * API to start the software timer.
00443  */
00444 SYSTIMER_STATUS_t SYSTIMER_StartTimer(uint32
_t id)
00445 {
00446     SYSTIMER_STATUS_t status;
00447     status = SYSTIMER_STATUS_FAILURE;
00448
00449     XMC_ASSERT("SYSTIMER_StartTimer: Failure i
n timer restart operation due to invalid timer ID"
,
00450               ((id <= SYSTIMER_CFG_MAX_TMR) &&
(id > 0U)));
00451     XMC_ASSERT("SYSTIMER_StartTimer: Error dur
ing start of software timer", (0U != (g_timer_trac
ker & (1U << (id - 1U)))));
00452
00453     /* Check if timer is running */
00454     if (SYSTIMER_STATE_STOPPED == g_timer_tbl[
id - 1U].state)
00455     {

```

```

00456     g_timer_tbl[id - 1U].count = (g_timer_tbl
[id - 1U].reload + HW_TIMER_ADDITIONAL_CNT);
00457     /* set timer status as SYSTIMER_STATE_RU
NNING */
00458     g_timer_tbl[id - 1U].state = SYSTIMER_ST
ATE_RUNNING;
00459     /* Insert this timer into timer list */
00460     SYSTIMER_lInsertTimerList((id - 1U));
00461     status = SYSTIMER_STATUS_SUCCESS;
00462 }
00463
00464     return (status);
00465 }
00466
00467 /*
00468  * API to stop the software timer.
00469  */
00470 SYSTIMER_STATUS_t SYSTIMER_StopTimer(uint32_
t id)
00471 {
00472     SYSTIMER_STATUS_t status;
00473     status = SYSTIMER_STATUS_SUCCESS;
00474
00475     XMC_ASSERT("SYSTIMER_StopTimer: Failure in
timer restart operation due to invalid timer ID",
00476               ((id <= SYSTIMER_CFG_MAX_TMR) &&
(id > 0U)));
00477     XMC_ASSERT("SYSTIMER_StopTimer: Error duri
ng stop of software timer", (0U != (g_timer_tracke
r & (1U << (id - 1U)))));
00478
00479     if (SYSTIMER_STATE_NOT_INITIALIZED == g_ti
mer_tbl[id - 1U].state)
00480     {
00481         status = SYSTIMER_STATUS_FAILURE;
00482     }
00483     else

```

```

00484     {
00485         /* Check whether Timer is in Stop state
*/
00486         if (SYSTIMER_STATE_RUNNING == g_timer_tbl
[id - 1U].state)
00487         {
00488             /* remove Timer from node list */
00489             SYSTIMER_removeTimerList(id - 1U);
00490             /* Set timer status as SYSTIMER_STATE_
STOPPED */
00491             g_timer_tbl[id - 1U].state = SYSTIMER_
STATE_STOPPED;
00492         }
00493     }
00494
00495     return (status);
00496 }
00497
00498 /*
00499  * API to reinitialize the time interval an
d to start the timer.
00500  */
00501 SYSTIMER_STATUS_t SYSTIMER_RestartTimer(uint
32_t id, uint32_t microsec)
00502 {
00503     uint32_t period_ratio = 0U;
00504     SYSTIMER_STATUS_t status;
00505     status = SYSTIMER_STATUS_SUCCESS;
00506
00507     XMC_ASSERT("SYSTIMER_RestartTimer: Failure
in timer restart operation due to invalid timer I
D",
00508               ((id <= SYSTIMER_CFG_MAX_TMR) &&
(id > 0U)));
00509     XMC_ASSERT("SYSTIMER_RestartTimer: Error d
uring restart of software timer", (0U != (g_timer_
tracker & (1U << (id - 1U))));

```

```

00510     XMC_ASSERT("SYSTIMER_RestartTimer: Can not
          restart timer due to invalid period value",
00511                (microsec >= SYSTIMER_TICK_PERIOD_US) && (microsec > 0U));
00512
00513
00514     if (SYSTIMER_STATE_NOT_INITIALIZED == g_timer_tbl[id - 1U].state)
00515     {
00516         status = SYSTIMER_STATUS_FAILURE;
00517     }
00518     else
00519     {
00520         /* check whether timer is in run state */
00521         if( SYSTIMER_STATE_STOPPED != g_timer_tbl[id - 1U].state)
00522         {
00523             /* Stop the timer */
00524             status = SYSTIMER_StopTimer(id);
00525         }
00526         if (SYSTIMER_STATUS_SUCCESS == status)
00527         {
00528             period_ratio = (uint32_t)(microsec / SYSTIMER_TICK_PERIOD_US);
00529             g_timer_tbl[id - 1U].reload = period_ratio;
00530             /* Start the timer */
00531             status = SYSTIMER_StartTimer(id)
;
00532         }
00533     }
00534
00535     return (status);
00536 }
00537

```

```

00538 /*
00539  *   Function to delete the Timer instance.
00540  */
00541 SYSTIMER_STATUS_t SYSTIMER_DeleteTimer(uint3
2_t id)
00542 {
00543     SYSTIMER_STATUS_t status;
00544     status = SYSTIMER_STATUS_SUCCESS;
00545
00546     XMC_ASSERT("SYSTIMER_DeleteTimer: Failure
in timer restart operation due to invalid timer ID"
,
00547               ((id <= SYSTIMER_CFG_MAX_TMR) &&
(id > 0U)));
00548     XMC_ASSERT("SYSTIMER_DeleteTimer: Error du
ring deletion of software timer", (0U != (g_timer_
tracker & (1U << (id - 1U)))));
00549
00550     /* Check whether Timer is in delete state
*/
00551     if (SYSTIMER_STATE_NOT_INITIALIZED == g_ti
mer_tbl[id - 1U].state)
00552     {
00553         status = SYSTIMER_STATUS_FAILURE;
00554     }
00555     else
00556     {
00557         if (SYSTIMER_STATE_STOPPED == g_timer_tbl
[id - 1U].state)
00558         {
00559             /* Set timer status as SYSTIMER_STATE_
NOT_INITIALIZED */
00560             g_timer_tbl[id - 1U].state = SYSTIMER_
STATE_NOT_INITIALIZED;
00561             /* Release resource which are hold by
this timer */
00562             g_timer_tracker &= ~(1U << (id - 1U));

```

```

00563     }
00564     else
00565     {
00566         /* Yes, remove this timer from timer list during ISR execution */
00567         g_timer_tbl[id - 1U].delete_swtmr = true;
00568     }
00569 }
00570
00571 return (status);
00572 }
00573
00574 /*
00575  * API to get the current SysTick time in microsecond.
00576  */
00577 uint32_t SYSTIMER_GetTime(void)
00578 {
00579     return (g_systick_count * SYSTIMER_TICK_PERIOD_US);
00580 }
00581
00582 /*
00583  * API to get the SysTick count.
00584  */
00585 uint32_t SYSTIMER_GetTickCount(void)
00586 {
00587     return (g_systick_count);
00588 }
00589
00590 /*
00591  * API to get the current state of software timer.
00592  */
00593 SYSTIMER_STATE_t SYSTIMER_GetTimerState(uint32_t id)

```

```
00594 {  
00595     return (g_timer_tbl[id - 1U].state);  
00596 }
```

