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).
# SYSTIMER

## 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>SYSTIMER</td>
<td>System Timer</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>IO</td>
<td>Input Output</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 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.

![Software timer behaviour in One Shot mode.](image)

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.
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
5. XMC1400 Reference Manual
6. XMC1300 Reference Manual

**Limitations**

None
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.
### App Configuration Parameters

#### Figure 1: General Settings

<table>
<thead>
<tr>
<th>General Settings</th>
<th>Interrupt Settings</th>
</tr>
</thead>
<tbody>
<tr>
<td>SysTick timer period [us]:</td>
<td>1000</td>
</tr>
<tr>
<td>Number of software timers:</td>
<td>8</td>
</tr>
</tbody>
</table>

#### Figure 2: Interrupt Settings

<table>
<thead>
<tr>
<th>General Settings</th>
<th>Interrupt Settings</th>
</tr>
</thead>
<tbody>
<tr>
<td>SysTick Interrupt Settings</td>
<td></td>
</tr>
<tr>
<td>Preemption priority</td>
<td>63</td>
</tr>
<tr>
<td>Subpriority</td>
<td>0</td>
</tr>
</tbody>
</table>
## Enumerations

<table>
<thead>
<tr>
<th>Type</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>enum</code></td>
<td><code>SYSTIMER_STATUS</code> { <code>SYSTIMER_STATUS_SUCCESS</code>, <code>SYSTIMER_STATUS_FAILURE</code> } This enumeration indicates status <code>SYSTIMER</code>. More...</td>
</tr>
<tr>
<td><code>enum</code></td>
<td><code>SYSTIMER_STATE</code> { <code>SYSTIMER_STATE_NOT_INITIALIZED</code> = 0U, <code>SYSTIMER_STATE_RUNNING</code>, <code>SYSTIMER_STATE_STOPPED</code> } This enumeration defines possible timer state. More...</td>
</tr>
<tr>
<td><code>enum</code></td>
<td><code>SYSTIMER_MODE</code> { <code>SYSTIMER_MODE_ONE_SHOT</code>, <code>SYSTIMER_MODE_PERIODIC</code> } Enumeration values which describe timer types. More...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Type</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>typedef enum</code></td>
<td><code>SYSTIMER_STATUS</code> <code>SYSTIMER_STATUS_t</code> This enumeration indicates status <code>SYSTIMER</code>.</td>
</tr>
<tr>
<td><code>typedef enum</code></td>
<td><code>SYSTIMER_STATE</code> <code>SYSTIMER_STATE_t</code> This enumeration defines possible timer state.</td>
</tr>
<tr>
<td><code>typedef enum</code></td>
<td><code>SYSTIMER_MODE</code> <code>SYSTIMER_MODE_t</code> Enumeration values which describe timer types.</td>
</tr>
</tbody>
</table>
## Enumeration Type Documentation

### `enum SYSTIMER_MODE`

Enumeration values which describes timer types.

**Enumerator:**
- `SYSTIMER_MODE_ONE_SHOT`  
  timer type is one shot
- `SYSTIMER_MODE_PERIODIC`  
  timer type is periodic

Definition at line 119 of file `SYSTIMER.h`.

### `enum SYSTIMER_STATE`

This enumeration defines possible timer state.

**Enumerator:**
- `SYSTIMER_STATE_NOT_INITIALIZED`  
  The timer is in uninitialized state
- `SYSTIMER_STATE_RUNNING`  
  The timer is in running state
- `SYSTIMER_STATE_STOPPED`  
  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**

### Data structures

<table>
<thead>
<tr>
<th>typedef void(* SYSTIMER_CALLBACK_t)(void *args)</th>
</tr>
</thead>
<tbody>
<tr>
<td>timer callback function pointer</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>typedef struct SYSTIMER SYSTIMER_t</th>
</tr>
</thead>
<tbody>
<tr>
<td>This structure contains pointer which is used to hold CPU instance handle and variables for priority group.</td>
</tr>
</tbody>
</table>
## SYSTIMER

### Methods

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

**Methods**
Function Documentation

```c
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, infinitive 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 pow(2,24) * (1/fCPU) * 1E6, 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:**

```c
#include <DAVE.h>
#define ONESEC 1000000U

void LED_Toggle_EverySec(void)
{
    // Add user code here
}

int main(void)
```


```c
{
    uint32_t TimerId;
    // ... Initializes APPs configuration ...
    DAVE_Init(); // SYSTIMER APP Initialized during DAVE Initialization
    // Create Software timer
    TimerId = (uint32_t)SYSTIMER_CreateTimer(ONES EC, SYSTIMER_MODE_PERIODIC,(void*)LED_Toggle_Every Sec,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:

```c
#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 operation
        }
    }
    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>
```c
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.

```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:**

```c
#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(ONES_EC, 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
  }
}
### 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:**

```c
#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(ONES
        EC, SYSTIMER_MODE_PERIODIC, (void*)LED_Toggle_Every
        Sec, 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
    }
}
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:

```c
#include <DAVE.h>
#define ONESEC 1000000U
#define NEW_INTERVAL (ONESEC * 10U)
void LED_Toggle_EverySec(void)
{
    // Add user code here
}
```
```c
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(ONES_EC, 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(TimerId);  // use case scenario 2
if (timer_state == SYSTIMER_STATE_RUNNING)
{
    status = SYSTIMER_StopTimer(TimerId);
    status = SYSTIMER_RestartTimer(TimerId, NEW_INTERVAL);
    // Add user code here
}
else if (timer_state == SYSTIMER_STATE_STOPPED)
{
    status = SYSTIMER_RestartTimer(TimerId, NEW_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(NEW_INTERVAL, SYSTIMER_MODE_PERIODIC, (void*)LED_Toggle_EverySec, NULL);
    status = SYSTIMER_StartTimer(TimerId);
    // Add user code here
}
else
{
    // timer ID Can not be zero
}
// ... infinite loop ...
while (1)
{
}
\texttt{return} (1);
}

Definition at line 593 of file \texttt{SYSTIMER.c}.

References \texttt{g_timer_tbl}.

\begin{verbatim}
SYSTIMER_STATUS_t SYSTIMER_Init(SYSTIMER_t * handle)

Initializes \texttt{SYSTIMER} APP.

\textbf{Parameters:}

\texttt{handle} Pointer pointing to \texttt{SYSTIMER} APP data structure.
Refer \texttt{SYSTIMER_t} for details.

\textbf{Returns:}

\texttt{SYSTIMER_STATUS_t} APP status. Refer \texttt{SYSTIMER_STATUS_t} for details.

\textbf{Description:}

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

\textbf{Example Usage:}

\begin{verbatim}
#include <DAVE.h>       //Declarations from DAVE Code Generation (includes SFR declaration)

int main(void)
{
   SYSTIMER_STATUS_t init_status;

   init_status = (SYSTIMER_STATUS_t)SYSTIME
\end{verbatim}
\end{verbatim}
R_Init(&SYSTIMER_0); // Initialization of SYS
TIMER APP

if (init_status == SYSTIMER_STATUS_SUCCESS)
{
    // Add application code here
    while(1)
    {
    }
}
else
{
    XMC_DEBUG("main: Application initialization 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 microsec
)

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

```c
#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(ONES EC, 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 restart operation
            }
        } else {
            // Error during software timer stop operation
        }
    } else {
        // Error during software timer start operation
    }
}
} 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:

```c
#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:

```c
#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 operation
    }
    else
    {
        // Error during software timer start operation
    }
    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().
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:

<table>
<thead>
<tr>
<th>General Settings</th>
<th>Interrupt Settings</th>
</tr>
</thead>
<tbody>
<tr>
<td>SysTick timer period [us]:</td>
<td>1000</td>
</tr>
<tr>
<td>Number of software timers:</td>
<td>1</td>
</tr>
</tbody>
</table>

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:

Step3: Pin Direction setting
Set Pin direction = Input/Output

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)**

```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 interval in order to generate software timer callback 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);
SYSTIMER

Release History

Release History
### SYSTIMER

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

---
SYSTIMER

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

<table>
<thead>
<tr>
<th>Type</th>
<th>Field</th>
</tr>
</thead>
<tbody>
<tr>
<td>bool</td>
<td>init_status</td>
</tr>
</tbody>
</table>
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

## Data Structure Index

<table>
<thead>
<tr>
<th>S</th>
<th>SYSTIMER</th>
<th>S</th>
</tr>
</thead>
</table>

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

- **init_status**: SYSTIMER


<table>
<thead>
<tr>
<th>Home</th>
<th>Data Structures</th>
<th>Data Structure Index</th>
<th>Data Fields</th>
</tr>
</thead>
<tbody>
<tr>
<td>All</td>
<td>Variables</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- init_status : SYSTIMER
# File List

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

<table>
<thead>
<tr>
<th>File Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYSTIMER.c</td>
<td>[code]</td>
</tr>
<tr>
<td>SYSTIMER.h</td>
<td>[code]</td>
</tr>
<tr>
<td>Home</td>
<td></td>
</tr>
<tr>
<td>------</td>
<td></td>
</tr>
<tr>
<td>File List</td>
<td>Globals</td>
</tr>
<tr>
<td>SYSTIMER.c File Reference</td>
<td></td>
</tr>
</tbody>
</table>

Variables
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

<table>
<thead>
<tr>
<th>Function Type</th>
<th>Function Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DAVE_APP_VERSION_t</td>
<td>SYSTIMER_GetAppVersion ()</td>
<td>Get SYSTIMER APP version.</td>
</tr>
<tr>
<td>SYSTIMER_STATUS_t</td>
<td>SYSTIMER_Init (SYSTIMER_t *handle)</td>
<td>Initializes SYSTIMER APP.</td>
</tr>
<tr>
<td>uint32_t</td>
<td>SYSTIMER_CreateTimer (uint32_t period, SYSTIMER_MODE_t mode, SYSTIMER_CALLBACK_t callback, void *args)</td>
<td>Creates a new software timer.</td>
</tr>
<tr>
<td>SYSTIMER_STATUS_t</td>
<td>SYSTIMER_StartTimer (uint32_t id)</td>
<td>Starts the software timer.</td>
</tr>
<tr>
<td>SYSTIMER_STATUS_t</td>
<td>SYSTIMER_StopTimer (uint32_t id)</td>
<td>Stops the software timer.</td>
</tr>
<tr>
<td>SYSTIMER_STATUS_t</td>
<td>SYSTIMER_RestartTimer (uint32_t id, uint32_t microsec)</td>
<td>Function to modify the time interval and restart the timer for the new time interval.</td>
</tr>
<tr>
<td>SYSTIMER_STATUS_t</td>
<td>SYSTIMER_DeleteTimer (uint32_t id)</td>
<td>Deletes the software timer from the timer list.</td>
</tr>
<tr>
<td>uint32_t</td>
<td>SYSTIMER_GetTime (void)</td>
<td>Gives the current hardware SysTick time in microsecond since start of hardware SysTick timer.</td>
</tr>
<tr>
<td>uint32_t</td>
<td>SYSTIMER_GetTickCount (void)</td>
<td>Gives the SysTick count.</td>
</tr>
<tr>
<td>SYSTIMER_STATE_t</td>
<td>SYSTIMER_GetTimerState (uint32_t id)</td>
<td>Gives the current state of software timer.</td>
</tr>
<tr>
<td>Variables</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-----------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SYSTIMER_OBJECT_t  g_timer_tbl [SYSTIMER_CFG_MAX_TMR]</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
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

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 "systimerExtern.h"
**Data Structures**

<table>
<thead>
<tr>
<th>struct</th>
<th>SYSTIMER</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>This structure contains pointer which is used to hold CPU instance handle and variables for priority group.</td>
</tr>
</tbody>
</table>
## Typedefs

<table>
<thead>
<tr>
<th>Typedef</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>typedef void</strong> (<em>SYSTIMER_CALLBACK_t</em>)(void *args)</td>
<td>timer callback function pointer</td>
</tr>
<tr>
<td><strong>typedef struct</strong> SYSTIMER SYSTIMER_t</td>
<td>This structure contains pointer which is used to hold CPU instance handle and variables for priority group.</td>
</tr>
</tbody>
</table>
### Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DAVE_APP_VERSION_t</td>
<td>\texttt{SYSTIMER_GetAppVersion} (void)</td>
</tr>
<tr>
<td></td>
<td>Get \texttt{SYSTIMER} APP version.</td>
</tr>
<tr>
<td>SYSTIMER_STATUS_t</td>
<td>\texttt{SYSTIMER_Init} (\texttt{SYSTIMER_t *handle})</td>
</tr>
<tr>
<td></td>
<td>Initializes \texttt{SYSTIMER} APP.</td>
</tr>
<tr>
<td>uint32_t</td>
<td>\texttt{SYSTIMER_CreateTimer} (uint32_t \texttt{period}, \texttt{SYSTIMER_MODE_t mode}, \texttt{SYSTIMER_CALLBACK_t callback}, \texttt{void *args})</td>
</tr>
<tr>
<td></td>
<td>Creates a new software timer.</td>
</tr>
<tr>
<td>SYSTIMER_STATUS_t</td>
<td>\texttt{SYSTIMER_StartTimer} (uint32_t \texttt{id})</td>
</tr>
<tr>
<td></td>
<td>Starts the software timer.</td>
</tr>
<tr>
<td>SYSTIMER_STATUS_t</td>
<td>\texttt{SYSTIMER_StopTimer} (uint32_t \texttt{id})</td>
</tr>
<tr>
<td></td>
<td>Stops the software timer.</td>
</tr>
<tr>
<td>SYSTIMER_STATUS_t</td>
<td>\texttt{SYSTIMER_RestartTimer} (uint32_t \texttt{id}, uint32_t \texttt{microsec})</td>
</tr>
<tr>
<td></td>
<td>Function to modify the time interval and restart the timer for the new time interval.</td>
</tr>
<tr>
<td>SYSTIMER_STATUS_t</td>
<td>\texttt{SYSTIMER_DeleteTimer} (uint32_t \texttt{id})</td>
</tr>
<tr>
<td></td>
<td>Deletes the software timer from the timer list.</td>
</tr>
<tr>
<td>uint32_t</td>
<td>\texttt{SYSTIMER_GetTime} (void)</td>
</tr>
<tr>
<td></td>
<td>Gives the current hardware SysTick time in microsecond since start of hardware SysTick timer.</td>
</tr>
<tr>
<td>uint32_t</td>
<td>\texttt{SYSTIMER_GetTickCount} (void)</td>
</tr>
<tr>
<td></td>
<td>Gives the SysTick count.</td>
</tr>
<tr>
<td>SYSTIMER_STATE_t</td>
<td>\texttt{SYSTIMER_GetTimerState} (uint32_t \texttt{id})</td>
</tr>
<tr>
<td></td>
<td>Gives the current state of software timer.</td>
</tr>
</tbody>
</table>
enum SYSTIMER_STATUS { 
SYSTIMER_STATUS_SUCCESS 
SYSTIMER_STATUS_FAILURE 
} 
This enumeration indicates status SYSTIMER. More...

enum SYSTIMER_STATE { 
SYSTIMER_STATE_NOT_INITIALIZED = 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 describes 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 describes timer types.

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:

- **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

- 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
g_timer_tbl : SYSTIMER.c
SYSTIMER

- 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

- SYSTIMER_MODE : SYSTIMER.h
- SYSTIMER_STATE : SYSTIMER.h
- SYSTIMER_STATUS : SYSTIMER.h
SYSTIMER

- 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

SYSTIMER.h

Go to the documentation of this file.

00001
00067 /***********************************************************************************/
00068  *******************************************************************************
00069  *******************************************************************************
00069  * HEADER FILES
00070  ***********************************************************************************/
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 /***********************************************************************************/
00078  *******************************************************************************
00078  *******************************************************************************
00079  * MACROS
00080  ***********************************************************************************/
00080
00081 #if (!((XMC_LIB_MAJOR_VERSION == 2U) && \
00083   (XMC_LIB_MINOR_VERSION >= 0U) && \
00084   (XMC_LIB_PATCH_VERSION >= 0U)))
#error "SYSTIMER requires XMC Peripheral Library v2.0.0 or higher"

```c
typedef enum SYSTIMER_STATUS
{
    SYSTIMER_STATUS_SUCCESS = 0U,
    SYSTIMER_STATUS_FAILURE
} SYSTIMER_STATUS_t;

typedef enum SYSTIMER_STATE
{
    SYSTIMER_STATE_NOT_INITIALIZED = 0U,
    SYSTIMER_STATE_RUNNING,
    SYSTIMER_STATE_STOPPED
} SYSTIMER_STATE_t;

typedef enum SYSTIMER_MODE
{
    SYSTIMER_MODE_ONE_SHOT = 0U,
    SYSTIMER_MODE_PERIODIC
} SYSTIMER_MODE_t;
```

```c
/* DATA STRUCTURES */
```
typedef void (*SYSTIMER_CALLBACK_t)(void *args);

typedef struct SYSTIMER {
  bool init_status;
} SYSTIMER_t;

/* API Prototypes */

DAVE_APP_VERSION_t SYSTIMER_GetAppVersion(void);

SYSTIMER_STATUS_t SYSTIMER_Init(SYSTIMER_t *handle);

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

SYSTIMER_STATUS_t SYSTIMER_StartTimer(uint32_t id);
SYSTIMER_STATUS_t SYSTIMER_StopTimer(uint32_t id);

SYSTIMER_STATUS_t SYSTIMER_RestartTimer(uint32_t id, uint32_t microsec);

SYSTIMER_STATUS_t SYSTIMER_DeleteTimer(uint32_t id);

uint32_t SYSTIMER_GetTime(void);

uint32_t SYSTIMER_GetTickCount(void);

SYSTIMER_STATE_t SYSTIMER_GetTimerState(uint32_t id);

/* Support for C++ codebase */
#endif __cplusplus

/* Inclusion of APP extern file */
#include "systimerExtern.h"

#endif /* SYSTIMER_H */
SYSTIMER.c

Go to the documentation of this file.

00001
00071  /*****************************************************************************
00072  ******************************************************************************
00073  ***************************************************************************/
00074
00075  /* Included to access APP data structure, functions & enumerations */
00076  #include "systimer.h"
00077
00078  /*****************************************************************************
00079  ******************************************************************************
00080  ***************************************************************************/
00081
00082  #define HW_TIMER_ADDITIONAL_CNT (1U)
00083
00084  /*****************************************************************************
00085  ******************************************************************************
00086  ***************************************************************************/
00087
00088  /* LOCAL DATA */
/* SYSTIMER_OBJECT structure acts as the timer control block */

typedef struct SYSTIMER_OBJECT {
  struct SYSTIMER_OBJECT *next;
  struct SYSTIMER_OBJECT *prev;
  SYSTIMER_CALLBACK_t callback;
  SYSTIMER_MODE_t mode;
  SYSTIMER_STATE_t state;
  void *args;
  uint32_t id;
  uint32_t count;
  uint32_t reload;
  bool delete_swtmr;
} SYSTIMER_OBJECT_t;

SYSTIMER_OBJECT_t g_timer_tbl[SYSTIMER_CFG_MAX_TMR];

/* The header of the timer Control list. */
SYSTIMER_OBJECT_t *g_timer_list = NULL;

/* Timer ID tracker */
uint32_t g_timer_tracker = 0U;

/* SysTick counter */
volatile uint32_t g_systick_count = 0U;

/* LOCAL ROUTINES */
This function is called to insert a timer into the timer list.

This function is called to remove a timer from the timer list.

Handler function called from SysTick event handler.

SysTick handler which is the main interrupt service routine to service the system timer's configured

void SysTick_Handler(void);

/* API IMPLEMENTATION

*/
00143 /*
00144  * This function is called to insert a timer
00145  * into the timer list.
00146 */
00147 static void SYSTIMER_lInsertTimerList(uint32_t tbl_index)
00148 {
00149     SYSTIMER_OBJECT_t *object_ptr;
00150     int32_t delta_ticks;
00151     int32_t timer_count;
00152     bool found_flag = false;
00153     /* Get timer time */
00154     timer_count = (int32_t)g_timer_tbl[tbl_index].count;
00155     /* Check if Timer list is NULL */
00156     if (NULL == g_timer_list)
00157     {
00158         /* Set this as first Timer */
00159         g_timer_list = &g_timer_tbl[tbl_index];
00160     }
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)
/* Check If head item */
if (NULL != object_ptr->prev)
{
    /* If Insert to list */
object_ptr->prev->next = &g_timer_tbl[tbl_index];
goct_ptr->prev = g_timer_tbl[tbl_index].prev = object_ptr->prev;
goct_ptr = g_timer_tbl[tbl_index].next = object_ptr->prev = &g_timer_tbl[tbl_index];
}
else
{
    /* Set Timer as first item */
g_timer_tbl[tbl_index].next = g_timer_list;
goct_list->prev = &g_timer_tbl[tbl_index];
goct_list = &g_timer_tbl[tbl_index];
}
goct_tbl[tbl_index].count = g_timer_tbl[tbl_index].next->count + (uint32_t)delta_ticks;
goct_tbl[tbl_index].next->count -= g_timer_tbl[tbl_index].count;
found_flag = true;
}
/* Check for last item in list */
else
{
    if ((delta_ticks > 0) && (NULL == object_ptr->next))
    {

/* Yes, insert into */
g_timer_tbl[tbl_index].prev = object_ptr;
object_ptr->next = &g_timer_tbl[tbl_index];
g_timer_tbl[tbl_index].count = (uint32_t)delta_ticks;
found_flag = true;
}
}
/* Get the next item in timer list */
object_ptr = object_ptr->next;
}
}
/*
This function is called to remove a timer from the timer list.
*/
static void SYSTIMER_lRemoveTimerList(uint32_t tbl_index)
{
SYSTIMER_OBJECT_t *object_ptr;
object_ptr = &g_timer_tbl[tbl_index];
/* Check whether only one timer available */
if ((NULL == object_ptr->prev) && (NULL == object_ptr->next))
{
/* set timer list as NULL */
g_timer_list = NULL;
}
/* Check if the first item in timer list */
else if (NULL == object_ptr->prev)
{

/* Remove timer from list, and reset timer list */
g_timer_list = object_ptr->next;
g_timer_list->prev = NULL;
g_timer_list->count += object_ptr->count;
object_ptr->next = NULL;
}

/* Check if the last item in timer list */
else if (NULL == object_ptr->next)
{
    /* Remove timer from list */
    object_ptr->prev->next = NULL;
    object_ptr->prev = NULL;
}
else
{
    /* Remove timer from list */
    object_ptr->prev->next = object_ptr->next;
    object_ptr->next->prev = object_ptr->prev;
    object_ptr->next->count += object_ptr->count;
    object_ptr->next = NULL;
    object_ptr->prev = NULL;
}

/* Handler function called from SysTick event handler. */
static void SYSTIMER_lTimerHandler(void)
{
    SYSTIMER_OBJECT_t *object_ptr;

    /* Get first item of timer list */
object_ptr = g_timer_list;
while ((NULL != object_ptr) && (0U == object_ptr->count))
{
  if (true == object_ptr->delete_swtrm)
  {
    /* Yes, remove this timer from timer list */
    SYSTIMER_lRemoveTimerList((uint32_t)object_ptr->id);
    /* Set timer status as SYSTIMER_STATE_NOT_INITIALIZED */
    object_ptr->state = SYSTIMER_STATE_NOT_INITIALIZED;
    /* Release resource which are hold by this timer */
    g_timer_tracker &= ~(1U << object_ptr->id);
  }
  /* Check whether timer is a one shot timer */
  else if (SYSTIMER_MODE_ONE_SHOT == object_ptr->mode)
  {
    /* Yes, remove this timer from timer list */
    SYSTIMER_lRemoveTimerList((uint32_t)object_ptr->id);
    /* Set timer status as SYSTIMER_STATE_STOPPED */
    object_ptr->state = SYSTIMER_STATE_STOPPED;
    /* Call timer callback function */
    (object_ptr->callback)(object_ptr->args);
  }
  /* Check whether timer is periodic timer */
else if (SYSTIMER_MODE_PERIODIC == object_ptr->mode) {
    /* Yes, remove this timer from timer list */
    SYSTIMER_lRemoveTimerList((uint32_t)object_ptr->id);
    /* Reset timer tick */
    object_ptr->count = object_ptr->reload;
    /* Insert timer into timer list */
    SYSTIMER_lInsertTimerList((uint32_t)object_ptr->id);
    /* Call timer callback function */
    (object_ptr->callback)(object_ptr->args);
} else {
    break;
}
/* Get first item of timer list */
object_ptr = g_timer_list;

/* SysTick Event Handler. */
void SysTick_Handler(void)
{
    SYSTIMER_OBJECT_t *object_ptr;
    object_ptr = g_timer_list;
    g_systick_count++;
    if (NULL != object_ptr)
if (object_ptr->count > 1UL) {
    object_ptr->count--;
} else {
    object_ptr->count = 0U;
    SYSTIMER_lTimerHandler();
}

/*
 * Function to retrieve the version of the SYSTIMER APP.
 */
DAVE_APP_VERSION_t SYSTIMER_GetAppVersion()
{
    DAVE_APP_VERSION_t version;
    version.major = (uint8_t)SYSTIMER_MAJOR_VERSION;
    version.minor = (uint8_t)SYSTIMER_MINOR_VERSION;
    version.patch = (uint8_t)SYSTIMER_PATCH_VERSION;
    return (version);
}

/*
 * Initialization function which initializes the SYSTIMER APP, configures SysTick timer and SysTick exception.
 */
SYSTIMER_STATUS_t SYSTIMER_Init(SYSTIMER_t *
SYSTIMER_STATUS_t status = SYSTIMER_STATUS_SUCCESS;

XMC_ASSERT("SYSTIMER_Init: SYSTIMER APP handle pointer uninitialized", (handle != NULL));

/* Check APP initialization status to ensure whether SYSTIMER_Init called or not, initialize SYSTIMER if */
if (false == handle->init_status)
{
    /* Initialize the header of the list */
g_timer_list = NULL;

    /* Initialize SysTick timer */
    status = (SYSTIMER_STATUS_t)SysTick_Config((uint32_t)(SYSTIMER_SYSTICK_CLOCK * SYSTIMER_TICK_PERIOD));

    if (SYSTIMER_STATUS_FAILURE == status)
    {
        XMC_DEBUG("SYSTIMER_Init: Timer reload value out of range");
    }
    else
    {
        #if (UC_FAMILY == XMC4)
        /* setting of First SW Timer period is always and subpriority value for XMC4000 devices */
        NVIC_SetPriority(SysTick_IRQn, NVIC_EncodePriority(NVIC_GetPriorityGrouping(), SYSTIMER_PRIORITY, SYSTIMER_SUBPRIORITY));
        #endif
#elif (UC_FAMILY == XMC1)

/* setting of priority value for XMC10 devices */

NVIC_SetPriority(SysTick_IRQn, SYSTIMER_PRIORITY);
#endif

g_timer_tracker = 0U;

/* Update the Initialization status of the SYSTIMER APP instance */

handle->init_status = true;

status = SYSTIMER_STATUS_SUCCESS;

}
}

return (status);

}

/* API for creating a new software Timer in stance. */

uint32_t SYSTIMER_CreateTimer(

uint32_t period,
SYSTIMER_MODE_t mode,
SYSTIMER_CALLBACK_t callback,
void *args

)
{
uint32_t id = 0U;
uint32_t count = 0U;
uint32_t period_ratio = 0U;

XMC_ASSERT("SYSTIMER_CreateTimer: Timer creation failure due to invalid period value",

((period >= SYSTIMER_TICK_PERIOD _US) && (period > 0U) && (period <= 0xFFFFFFFFFU)))
; 00403   XMC_ASSERT("SYSTIMER_CreateTimer: Timer creation failure due to invalid timer mode",
00404        ((SYSTIMER_MODE_ONE_SHOT == mode)
00405          || (SYSTIMER_MODE_PERIODIC == mode)));  
00405   XMC_ASSERT("SYSTIMER_CreateTimer: Can not create software without user callback", (NULL != c
00406          allback));

00406   if (period < SYSTIMER_TICK_PERIOD_US)
00407       { id = 0U;
00408       }
00409   else
00410       {
00411          for (count = 0U; count < SYSTIMER_CFG_MAX_TMR; count++)
00412              {
00413                /* Check for free timer ID */
00414                if (0U == (g_timer_tracker & (1U << count)))
00415                    {
00416                       /* If yes, assign ID to this timer */
00417                       g_timer_tracker |= (1U << count);
00418                       /* Initialize the timer as per input values */
00419                       g_timer_tbl[count].id = count;
00420                       g_timer_tbl[count].mode = mode;
00421                       g_timer_tbl[count].state = SYSTIMER_STATE_STOPPED;
00422                       period_ratio = (uint32_t)(period / SYSTIMER_TICK_PERIOD_US);
00423                       g_timer_tbl[count].count = (period_ratio + HW_TIMER_ADDITIONAL_CNT);
00424                       g_timer_tbl[count].reload = period_ratio;
00425                     }
00427          g_timer_tbl[count].callback = callback;
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  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      XMC_ASSERT("SYSTIMER_StartTimer: Failure i
00449      n timer restart operation due to invalid timer ID" ,
00450          ((id <= SYSTIMER_CFG_MAX_TMR) &&
00451          (id > 0U)));  
00452      XMC_ASSERT("SYSTIMER_StartTimer: Error dur
00453      ing start of software timer", (0U != (g_timer_tracker & (1U << (id - 1U)))));
00454  /* Check if timer is running */
00455      if (SYSTIMER_STATE_STOPPED == g_timer_tbl[
00456       id - 1U].state)
00457          {  

g_timer_tbl[id - 1U].count = (g_timer_tbl[id - 1U].reload + HW_TIMER_ADDITIONAL_CNT);

/* set timer status as SYSTIMER_STATE_RUNNING */
g_timer_tbl[id - 1U].state = SYSTIMER_STATE_RUNNING;

/* Insert this timer into timer list */
SYSTIMER_lInsertTimerList((id - 1U));
status = SYSTIMER_STATUS_SUCCESS;
}

return (status);

SYSTIMER_STATUS_t SYSTIMER_StopTimer(uint32_t id)
{
SYSTIMER_STATUS_t status;
status = SYSTIMER_STATUS_SUCCESS;

XMC_ASSERT("SYSTIMER_StopTimer: Failure in timer restart operation due to invalid timer ID", ((id <= SYSTIMER_CFG_MAX_TMR) && (id > 0U)));
XMC_ASSERT("SYSTIMER_StopTimer: Error during stop of software timer", (0U != (g_timer_tracker & (1U <= (id - 1U)))));

if (SYSTIMER_STATE_NOT_INITIALIZED == g_timer_tbl[id - 1U].state)
{
    status = SYSTIMER_STATUS_FAILURE;
}
else
/* Check whether Timer is in Stop state */
if (SYSTIMER_STATE_RUNNING == g_timer_tbl[id - 1U].state) {
    /* remove Timer from node list */
    SYSTIMER_lRemoveTimerList(id - 1U);
    /* Set timer status as SYSTIMER_STATE_STOPPED */
    g_timer_tbl[id - 1U].state = SYSTIMER_STATE_STOPPED;
}
return (status);

/* API to reinitialize the time interval and to start the timer. */
SYSTIMER_STATUS_t SYSTIMER_RestartTimer(uint32_t id, uint32_t microsec) {
    uint32_t period_ratio = 0U;
    SYSTIMER_STATUS_t status;
    status = SYSTIMER_STATUS_SUCCESS;
    XMC_ASSERT("SYSTIMER_RestartTimer: Failure in timer restart operation due to invalid timer ID",
               ((id <= SYSTIMER_CFG_MAX_TMR) && (id > 0U)));
    XMC_ASSERT("SYSTIMER_RestartTimer: Error during restart of software timer", (0U != (g_timer_tracker & (1U << (id - 1U)))));
XMC_ASSERT("SYSTIMER_RestartTime: Can not restart timer due to invalid period value",
(microsec >= SYSTIMER_TICK_PERIOD_US) && (microsec > 0U));

if (SYSTIMER_STATE_NOT_INITIALIZED == g_timer_tbl[id - 1U].state)
{
    status = SYSTIMER_STATUS_FAILURE;
}
else
{
    /* check whether timer is in run state */
    if (SYSTIMER_STATE_STOPPED != g_timer_tbl[id - 1U].state)
    {
        /* Stop the timer */
        status = SYSTIMER_StopTimer(id);
    }
    if (SYSTIMER_STATUS_SUCCESS == status)
    {
        period_ratio = (uint32_t)(microsec / SYSTIMER_TICK_PERIOD_US);
        g_timer_tbl[id - 1U].reload = period_ratio;
        /* Start the timer */
        status = SYSTIMER_StartTimer(id);
    }
}
return (status);
/* Function to delete the Timer instance. */

SYSTIMER_STATUS_t SYSTIMER_DeleteTimer(uint32_t id)
{
    SYSTIMER_STATUS_t status;
    status = SYSTIMER_STATUS_SUCCESS;
    XMC_ASSERT("SYSTIMER_DeleteTimer: Failure in timer restart operation due to invalid timer ID",
        ((id <= SYSTIMER_CFG_MAX_TMR) && (id > 0U)));
    XMC_ASSERT("SYSTIMER_DeleteTimer: Error during deletion of software timer", (0U != (g_timer_tracker & (1U << (id - 1U)))));

    /* Check whether Timer is in delete state */
    if (SYSTIMER_STATE_NOT_INITIALIZED == g_timer_tbl[id - 1U].state)
    {
        status = SYSTIMER_STATUS_FAILURE;
    }
    else
    {
        if (SYSTIMER_STATE_STOPPED == g_timer_tbl[id - 1U].state)
        {
            /* Set timer status as SYSTIMER_STATE_NOT_INITIALIZED */
            g_timer_tbl[id - 1U].state = SYSTIMER_STATE_NOT_INITIALIZED;
            /* Release resource which are hold by this timer */
            g_timer_tracker &= ~(1U << (id - 1U));
        }
    }
}
else {
  /* Yes, remove this timer from timer list during ISR execution */
  g_timer_tbl[id - 1U].delete_swtmr = true;
}

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