/\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* uC/OS-II

\* The Real-Time Kernel

\* TIME MANAGEMENT

\*

\* (c) Copyright 1992-2009, Micrium, Weston, FL

\* All Rights Reserved

\*

\* File : OS\_TIME.C

\* By : Jean J. Labrosse

\* Version : V2.91

\*

\* LICENSING TERMS:

\* ---------------

\* uC/OS-II is provided in source form for FREE evaluation, for educational use or for peaceful research.

\* If you plan on using uC/OS-II in a commercial product you need to contact Micri祄 to properly license

\* its use in your product. We provide ALL the source code for your convenience and to help you experience

\* uC/OS-II. The fact that the source is provided does NOT mean that you can use it without paying a

\* licensing fee.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*/

#ifndef OS\_MASTER\_FILE

#include <ucos\_ii.h>

#endif

/\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* DELAY TASK 'n' TICKS

\*

\* Description: This function is called to delay execution of the currently running task until the

\* specified number of system ticks expires. This, of course, directly equates to delaying

\* the current task for some time to expire. No delay will result If the specified delay is

\* 0. If the specified delay is greater than 0 then, a context switch will result.

\*

\* Arguments : ticks is the time delay that the task will be suspended in number of clock 'ticks'.

\* Note that by specifying 0, the task will not be delayed.

\*

\* Returns : none

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*/

void OSTimeDly (INT32U ticks)

{

INT8U y;

#if OS\_CRITICAL\_METHOD == 3u /\* Allocate storage for CPU status register \*/

OS\_CPU\_SR cpu\_sr = 0u;

#endif

if (OSIntNesting > 0u) { /\* See if trying to call from an ISR \*/

return;

}

if (OSLockNesting > 0u) { /\* See if called with scheduler locked \*/

return;

}

if (ticks > 0u) { /\* 0 means no delay! \*/

OS\_ENTER\_CRITICAL();

y = OSTCBCur->OSTCBY; /\* Delay current task \*/

OSRdyTbl[y] &= (OS\_PRIO)~OSTCBCur->OSTCBBitX;

if (OSRdyTbl[y] == 0u) {

OSRdyGrp &= (OS\_PRIO)~OSTCBCur->OSTCBBitY;

}

OSTCBCur->OSTCBDly = ticks; /\* Load ticks in TCB \*/

OS\_EXIT\_CRITICAL();

OS\_Sched(); /\* Find next task to run! \*/

}

}

/\*$PAGE\*/

/\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* DELAY TASK FOR SPECIFIED TIME

\*

\* Description: This function is called to delay execution of the currently running task until some time

\* expires. This call allows you to specify the delay time in HOURS, MINUTES, SECONDS and

\* MILLISECONDS instead of ticks.

\*

\* Arguments : hours specifies the number of hours that the task will be delayed (max. is 255)

\* minutes specifies the number of minutes (max. 59)

\* seconds specifies the number of seconds (max. 59)

\* ms specifies the number of milliseconds (max. 999)

\*

\* Returns : OS\_ERR\_NONE

\* OS\_ERR\_TIME\_INVALID\_MINUTES

\* OS\_ERR\_TIME\_INVALID\_SECONDS

\* OS\_ERR\_TIME\_INVALID\_MS

\* OS\_ERR\_TIME\_ZERO\_DLY

\* OS\_ERR\_TIME\_DLY\_ISR

\*

\* Note(s) : The resolution on the milliseconds depends on the tick rate. For example, you can't do

\* a 10 mS delay if the ticker interrupts every 100 mS. In this case, the delay would be

\* set to 0. The actual delay is rounded to the nearest tick.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*/

#if OS\_TIME\_DLY\_HMSM\_EN > 0u

INT8U OSTimeDlyHMSM (INT8U hours,

INT8U minutes,

INT8U seconds,

INT16U ms)

{

INT32U ticks;

if (OSIntNesting > 0u) { /\* See if trying to call from an ISR \*/

return (OS\_ERR\_TIME\_DLY\_ISR);

}

if (OSLockNesting > 0u) { /\* See if called with scheduler locked \*/

return (OS\_ERR\_SCHED\_LOCKED);

}

#if OS\_ARG\_CHK\_EN > 0u

if (hours == 0u) {

if (minutes == 0u) {

if (seconds == 0u) {

if (ms == 0u) {

return (OS\_ERR\_TIME\_ZERO\_DLY);

}

}

}

}

if (minutes > 59u) {

return (OS\_ERR\_TIME\_INVALID\_MINUTES); /\* Validate arguments to be within range \*/

}

if (seconds > 59u) {

return (OS\_ERR\_TIME\_INVALID\_SECONDS);

}

if (ms > 999u) {

return (OS\_ERR\_TIME\_INVALID\_MS);

}

#endif

/\* Compute the total number of clock ticks required.. \*/

/\* .. (rounded to the nearest tick) \*/

ticks = ((INT32U)hours \* 3600uL + (INT32U)minutes \* 60uL + (INT32U)seconds) \* OS\_TICKS\_PER\_SEC

+ OS\_TICKS\_PER\_SEC \* ((INT32U)ms + 500uL / OS\_TICKS\_PER\_SEC) / 1000uL;

OSTimeDly(ticks);

return (OS\_ERR\_NONE);

}

#endif

/\*$PAGE\*/

/\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* RESUME A DELAYED TASK

\*

\* Description: This function is used resume a task that has been delayed through a call to either

\* OSTimeDly() or OSTimeDlyHMSM(). Note that you can call this function to resume a

\* task that is waiting for an event with timeout. This would make the task look

\* like a timeout occurred.

\*

\* Arguments : prio specifies the priority of the task to resume

\*

\* Returns : OS\_ERR\_NONE Task has been resumed

\* OS\_ERR\_PRIO\_INVALID if the priority you specify is higher that the maximum allowed

\* (i.e. >= OS\_LOWEST\_PRIO)

\* OS\_ERR\_TIME\_NOT\_DLY Task is not waiting for time to expire

\* OS\_ERR\_TASK\_NOT\_EXIST The desired task has not been created or has been assigned to a Mutex.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*/

#if OS\_TIME\_DLY\_RESUME\_EN > 0u

INT8U OSTimeDlyResume (INT8U prio)

{

OS\_TCB \*ptcb;

#if OS\_CRITICAL\_METHOD == 3u /\* Storage for CPU status register \*/

OS\_CPU\_SR cpu\_sr = 0u;

#endif

if (prio >= OS\_LOWEST\_PRIO) {

return (OS\_ERR\_PRIO\_INVALID);

}

OS\_ENTER\_CRITICAL();

ptcb = OSTCBPrioTbl[prio]; /\* Make sure that task exist \*/

if (ptcb == (OS\_TCB \*)0) {

OS\_EXIT\_CRITICAL();

return (OS\_ERR\_TASK\_NOT\_EXIST); /\* The task does not exist \*/

}

if (ptcb == OS\_TCB\_RESERVED) {

OS\_EXIT\_CRITICAL();

return (OS\_ERR\_TASK\_NOT\_EXIST); /\* The task does not exist \*/

}

if (ptcb->OSTCBDly == 0u) { /\* See if task is delayed \*/

OS\_EXIT\_CRITICAL();

return (OS\_ERR\_TIME\_NOT\_DLY); /\* Indicate that task was not delayed \*/

}

ptcb->OSTCBDly = 0u; /\* Clear the time delay \*/

if ((ptcb->OSTCBStat & OS\_STAT\_PEND\_ANY) != OS\_STAT\_RDY) {

ptcb->OSTCBStat &= ~OS\_STAT\_PEND\_ANY; /\* Yes, Clear status flag \*/

ptcb->OSTCBStatPend = OS\_STAT\_PEND\_TO; /\* Indicate PEND timeout \*/

} else {

ptcb->OSTCBStatPend = OS\_STAT\_PEND\_OK;

}

if ((ptcb->OSTCBStat & OS\_STAT\_SUSPEND) == OS\_STAT\_RDY) { /\* Is task suspended? \*/

OSRdyGrp |= ptcb->OSTCBBitY; /\* No, Make ready \*/

OSRdyTbl[ptcb->OSTCBY] |= ptcb->OSTCBBitX;

OS\_EXIT\_CRITICAL();

OS\_Sched(); /\* See if this is new highest priority \*/

} else {

OS\_EXIT\_CRITICAL(); /\* Task may be suspended \*/

}

return (OS\_ERR\_NONE);

}

#endif

/\*$PAGE\*/

/\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* GET CURRENT SYSTEM TIME

\*

\* Description: This function is used by your application to obtain the current value of the 32-bit

\* counter which keeps track of the number of clock ticks.

\*

\* Arguments : none

\*

\* Returns : The current value of OSTime

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*/

#if OS\_TIME\_GET\_SET\_EN > 0u

INT32U OSTimeGet (void)

{

INT32U ticks;

#if OS\_CRITICAL\_METHOD == 3u /\* Allocate storage for CPU status register \*/

OS\_CPU\_SR cpu\_sr = 0u;

#endif

OS\_ENTER\_CRITICAL();

ticks = OSTime;

OS\_EXIT\_CRITICAL();

return (ticks);

}

#endif

/\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* SET SYSTEM CLOCK

\*

\* Description: This function sets the 32-bit counter which keeps track of the number of clock ticks.

\*

\* Arguments : ticks specifies the new value that OSTime needs to take.

\*

\* Returns : none

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*/

#if OS\_TIME\_GET\_SET\_EN > 0u

void OSTimeSet (INT32U ticks)

{

#if OS\_CRITICAL\_METHOD == 3u /\* Allocate storage for CPU status register \*/

OS\_CPU\_SR cpu\_sr = 0u;

#endif

OS\_ENTER\_CRITICAL();

OSTime = ticks;

OS\_EXIT\_CRITICAL();

}

#endif