ch5
Time Management

Date : 2007/08/02

Speaker:Ming-Shyong Tsai

Clock Tick

- Periodic interrupt
  - Keep track of time delays and timeouts
  - clock ticks : 10 ~ 100 Hz
- five services deal with time issues:

	Enable when set to 1 in OS_CFG.h
– OSTimeDly( )	
– OSTimeDlyHMSM( )	*OS_TIME_DLY_HMSM_EN*
– OSTimeDlyResume( )	*OS_TIME_DLY_RESUME_EN*
– OSTimeGet( )	*OS_TIME_DLY_GET_SET_EN*
– OSTimeSet( )	*OS_TIME_DLY_GET_SET_EN*

OSTimeDly(1/3)

```
void OSTimeDly (INT16U ticks)
{
    if (ticks > 0) {
        OS_ENTER_CRITICAL();

        if ((OSRdyTbl[OSTCBCur->OSTCBY]
            &= ~OSTCBCur->OSTCBBitX) == 0)
                OSRdyGrp &= ~OSTCBCur->OSTCBBitY;
        }
        OSTCBCur->OSTCBDly = ticks;
        OS_EXIT_CRITICAL();
        OSSched();
    }
}
```

OSTimeDly(2/3)

- **OSRdyTbl [ ]**
  - Table of tasks which are ready to run
- OSTCBY
  - Index into ready table corresponding to task priority
- OSTCBBitX
  - Bit mask to access bit position in ready table
- **OSRdyGrp**
  - Ready list group
- OSTCBBitY
  - Bit mask to access bit position in ready group

OSTimeDly(3/3)

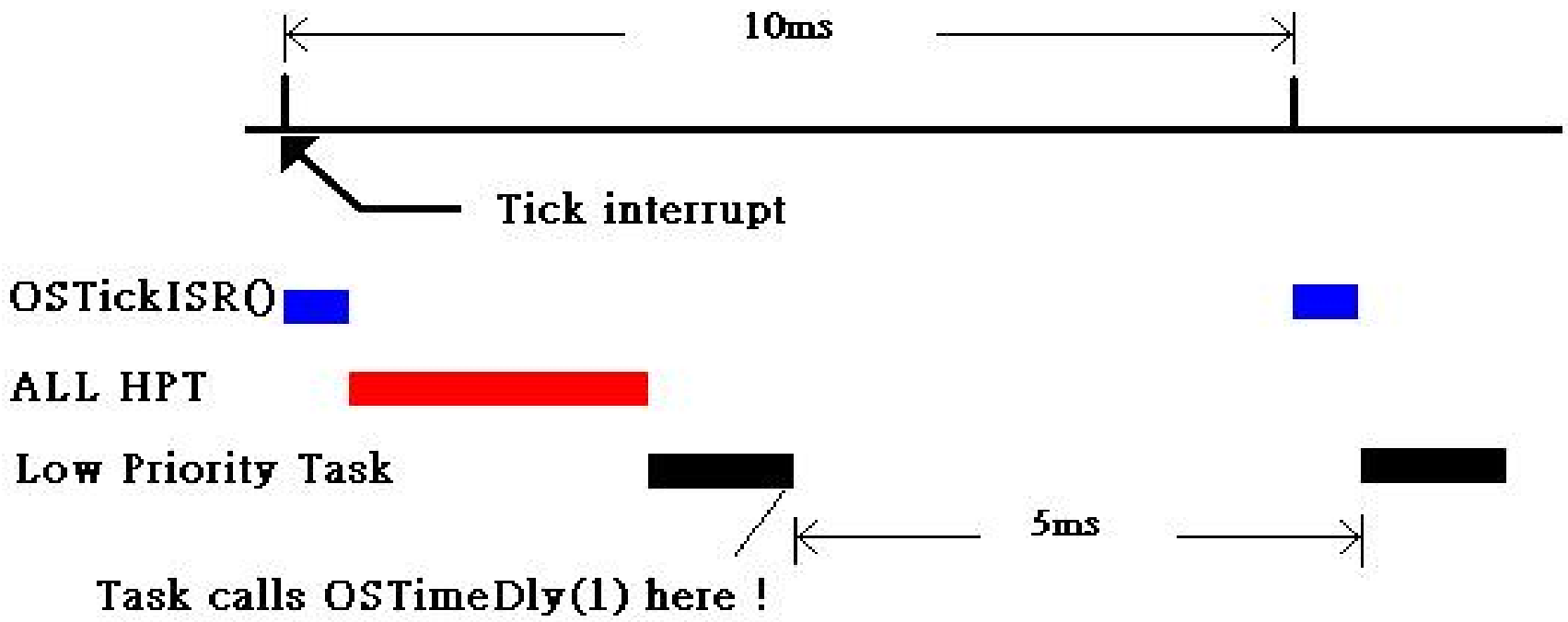

Figure 1. Delay resolution

OSTimeDlyHMSM

```
INT8U OSTimeDlyHMSM (INT8U hours,
      INT8U minutes, INT8U seconds, INT16U
      milli)
{
      INT32U  ticks;
      INT16U  loops;

      if (hours > 0 || minutes > 0 || seconds > 0
      || milli > 0) {
         if (minutes > 59) {
         return(OS_TIME_INVALID_MINUTES);
      }
      if (seconds > 59) {
       return(OS_TIME_INVALID_SECONDS);
      }
        If (milli > 999) {
            return (OS_TIME_INVALID_MILLI);
      }
```

```
ticks = (INT32U) hours * 3600L *
      OS_TICKS_PER_SEC
     + (INT32U) minutes * 60L *
      OS_TICKS_PER_SEC
      + (INT32U) seconds *
      OS_TICKS_PER_SEC
      + OS_TICKS_PER_SEC *
      ( (INT32U) milli  + 500L /
      OS_TICKS_PER_SEC  )   / 1000L;
      loops = ticks / 65536L;
      ticks = ticks % 65536L;
      OSTimeDly(ticks);
      while (loops > 0) {
                OSTimeDly(32768);
                OSTimeDly(32768);
                loops--;
      }
      return (OS_NO_ERR);
      } else {
                return (OS_TIME_ZERO_DLY);
      }
}
```

OSTimeDlyResume(1/2)

- Delayed tasks
  - Timeout
  - Be awaked by another tasks
  - OSTimeDlyResume( ) can also resume a task waiting for an event
- The task need to satisfy
  - a valid priority
  - existence
  - If the task exists ,It is waiting for delayed time to expire
  - Not suspended

  Then the task can be placed in ready list
- A task maybe delays itself by semaphore, mutex, event flag, mailbox, queue . And It can resume itself by the same way but requires more RAM for the ECB(event control block)

OSTimeDlyResume(2/2)

```
INT8U OSTimeDlyResume (INT8U prio)
{
    OS_TCB *ptcb;

    if (prio >= OS_LOWEST_PRIO) {
    return (OS_PRIO_INVALID);
    }

    OS_ENTER_CRITICAL();

    ptcb = (OS_TCB *)OSTCBPrioTbl[prio];
    if (ptcb != (OS_TCB *)0) {
      if (ptcb->OSTCBDly != 0) {
        ptcb->OSTCBDly = 0;

        if (!(ptcb->OSTCBStat
              &OS_STAT_SUSPEND)) {
        OSRdyGrp   |= ptcb->OSTCBBitY;
        OSRdyTbl[ptcb->OSTCBY] |=
              ptcb->OSTCBBitX;
         OS_EXIT_CRITICAL();
        OSSched();
        } else {
              OS_EXIT_CRITICAL();
        }
        return (OS_NO_ERR);
      } else {
        OS_EXIT_CRITICAL();
        return (OS_TIME_NOT_DLY);
        }
      } else {
        OS_EXIT_CRITICAL();
        return (OS_TASK_NOT_EXIST);
       }
}
```

System time, OSTimeGet and OSTimeSet(1/2)

- µC/ OSII : 32 bit-counter
  - Start at zero
  - increment itself when clock tick occurs
  - Multitasking by calling OSStart()
  - Obtain the current value by OSTimeGet()
  - Change the value by OSTimeSet()
- When accessing OSTime, interrupts are disabled
  - the most 8-bit processor ,32-bit counter
    - counter ++
    - copy a 32-bit value
  - atomic instruction

System time, OSTimeGet and OSTimeSet(2/2)

```
INT32U  OSTimeGet (void)
{
#if OS_CRITICAL_METHOD == 3
    OS_CPU_SR  cpu_sr;
#endif
   INT32U     ticks;


   OS_ENTER_CRITICAL();
   ticks = OSTime;
   OS_EXIT_CRITICAL();
   return (ticks);
}
```

```
void  OSTimeSet (INT32U ticks)
{
#if OS_CRITICAL_METHOD == 3
    OS_CPU_SR  cpu_sr;
#endif


   OS_ENTER_CRITICAL();
   OSTime = ticks;
   OS_EXIT_CRITICAL();
}
```