Chapter 9
Event Flag Management

79256030 張景和
s9256030@cs.nchu.edu.tw
中興大學資科所
系統與網路實驗室

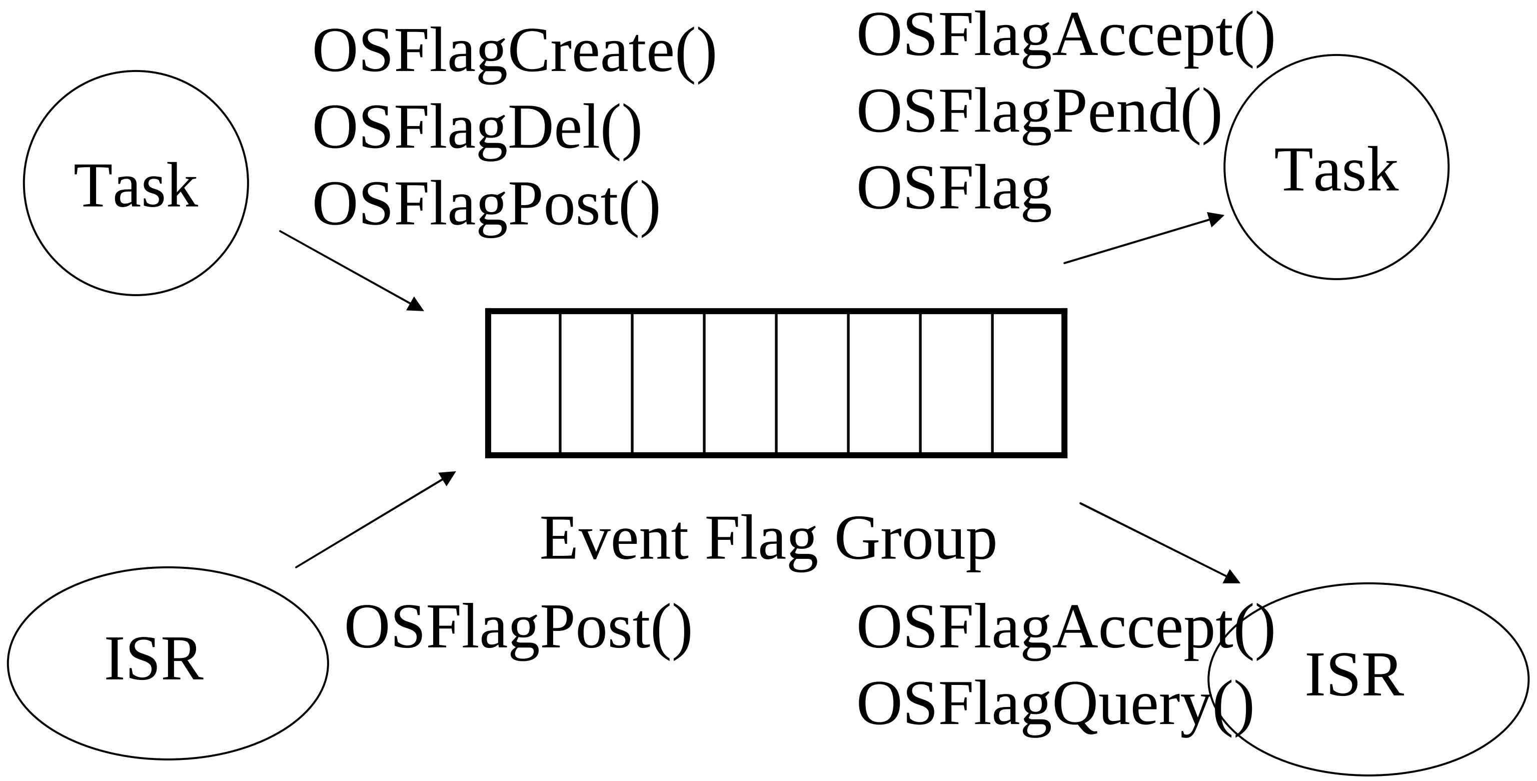
μC/OS-II event flags consist of two elements:

- ❖ A series of bits (either 8,16,32) used to hold the current state of the events in the group .
- ❖ A list of tasks waiting for a combination of these bits to be either set (1) or cleared (0).

Event flag configuration constant in OS_CFG.H

μC/OS-II Event Flag Service	Enableed when set 1 in OS_CFG.H
OSFlagAccept()	OS_FLAG_ACCEPT_EN
OSFlagCreate()	
OSFlagDel()	OS_FLAG_DEL_EN
OSFlagPenf()	
OSFlagPost()	
OSFlagQuery()	OS_FLAG_QUERY_EN

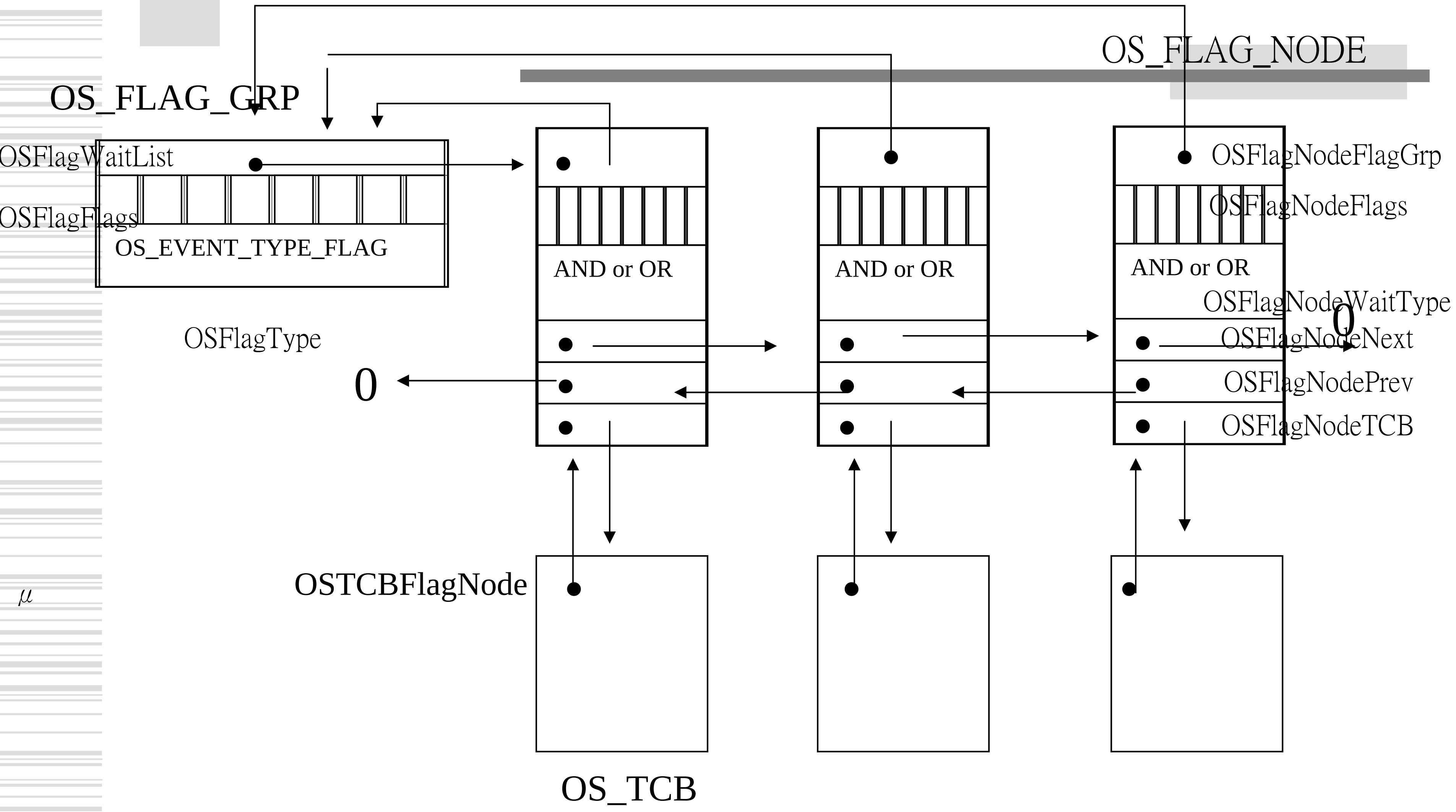
μ C/OS-II event flag services



Event Flag Group data structure

```
typedef struct {                        /* Event Flag Group                                   */
    INT8U        OSFlagType;            /* Should be set to OS_EVENT_TYPE_FLAG                */
    void        *OSFlagWaitList;        /* Pointer to first NODE of task waiting on event flag */
    OS_FLAGS     OSFlagFlags;           /* 8, 16 or 32 bit flags                              */
} OS_FLAG_GRP;
```

Relations between event flag group,event flag nodes, and TCBs



μ

Event flag group node data structure

```
typedef struct {                          /* Event Flag Wait List Node                      */
    void        *OSFlagNodeNext;          /* Pointer to next     NODE in wait list          */
    void        *OSFlagNodePrev;          /* Pointer to previous NODE in wait list          */
    void        *OSFlagNodeTCB;           /* Pointer to TCB of waiting task                 */
    void        *OSFlagNodeFlagGrp;       /* Pointer to Event Flag Group                    */
    OS_FLAGS     OSFlagNodeFlags;         /* Event flag to wait on                          */
    INT8U        OSFlagNodeWaitType;      /* Type of wait:                                  */
                                     /*      OS_FLAG_WAIT_AND                               */
                                     /*      OS_FLAG_WAIT_ALL                               */
                                     /*      OS_FLAG_WAIT_OR                                */
                                     /*      OS_FLAG_WAIT_ANY                               */
} OS_FLAG_NODE;
```

Creating an Event Flag Group,OSFlagCreate()

This function is called to create an event flag group

```
OS_FLAG_GRP  *OSFlagCreate (OS_FLAGS flags, INT8U *err)
{
#if OS_CRITICAL_METHOD == 3                    /* Allocate storage for CPU status register   */
    OS_CPU_SR    cpu_sr;
#endif
    OS_FLAG_GRP *pgrp;


    if (OSIntNesting > 0) {                    /* See if called from ISR ...                 */
        *err = OS_ERR_CREATE_ISR;              /* ... can't CREATE from an ISR               */
        return ((OS_FLAG_GRP *)0);
    }
```

Creating an Event Flag Group,OSFlagCreate() (Cont.)

```
OS_ENTER_CRITICAL();
    pgrp = OSFlagFreeList;                          /* Get next free event flag                    */
    if (pgrp != (OS_FLAG_GRP *)0) {                  /* See if we have event flag groups available     */
                                                 /* Adjust free list                            */
        OSFlagFreeList       = (OS_FLAG_GRP *)OSFlagFreeList->OSFlagWaitList;
        pgrp->OSFlagType      = OS_EVENT_TYPE_FLAG;  /* Set to event flag group type                 */
        pgrp->OSFlagFlags     = flags;              /* Set to desired initial value                 */
        pgrp->OSFlagWaitList = (void *)0;           /* Clear list of tasks waiting on flags          */
        OS_EXIT_CRITICAL();
        *err                 = OS_NO_ERR;
    } else {
        OS_EXIT_CRITICAL();
        *err                 = OS_FLAG_GRP_DEPLETED;
    }
    return (pgrp);                                  /* Return pointer to event flag group           */
}
```

Deleting an Event Flag Group.OSFlagDel()

This function deletes an event flag group and readies all tasks pending on the event flag group

```
OS_FLAG_GRP  *OSFlagDel (OS_FLAG_GRP *pgrp, INT8U opt, INT8U *err)
{
#if OS_CRITICAL_METHOD == 3                          /* Allocate storage for CPU status register */
    OS_CPU_SR     cpu_sr;
#endif
    BOOLEAN       tasks_waiting;
    OS_FLAG_NODE *pnode;


    if (OSIntNesting > 0) {                          /* See if called from ISR ...              */
        *err = OS_ERR_DEL_ISR;                           /* ... can't DELETE from an ISR            */
        return (pgrp);
    }
```

Deleting an Event Flag Group.OSFlagDel()(Cont.)

```
#if OS_ARG_CHK_EN > 0
    if (pgrp == (OS_FLAG_GRP *)0) {                    /* Validate 'pgrp'                    */
        *err = OS_FLAG_INVALID_PGRP;
        return (pgrp);
    }
    if (pgrp->OSFlagType != OS_EVENT_TYPE_FLAG) {          /* Validate event group type   */
        *err = OS_ERR_EVENT_TYPE;
        return (pgrp);
    }
#endif
    OS_ENTER_CRITICAL();
    if (pgrp->OSFlagWaitList != (void *)0) {              /* See if any tasks waiting on event flags  */
        tasks_waiting = TRUE;                        /* Yes                                */
    } else {
        tasks_waiting = FALSE;                       /* No                                 */
    }
```

Deleting an Event Flag Group.OSFlagDel()(Cont.)

```
switch (opt) {
    case OS_DEL_NO_PEND:                          /* Delete group if no task waiting        */
        if (tasks_waiting == FALSE) {
            pgrp->OSFlagType     = OS_EVENT_TYPE_UNUSED;
            pgrp->OSFlagWaitList = (void *)OSFlagFreeList; /* Return group to free list        */
            OSFlagFreeList       = pgrp;
            OS_EXIT_CRITICAL();
            *err                 = OS_NO_ERR;
            return ((OS_FLAG_GRP *)0);            /* Event Flag Group has been deleted      */
        } else {
            OS_EXIT_CRITICAL();
            *err                 = OS_ERR_TASK_WAITING;
            return (pgrp);
        }
```

Deleting an Event Flag Group.OSFlagDel()(Cont.)

```
case OS_DEL_ALWAYS:                              /* Always delete the event flag group      */
        pnode = (OS_FLAG_NODE *)pgrp->OSFlagWaitList;
        while (pnode != (OS_FLAG_NODE *)0) {          /* Ready ALL tasks waiting for flags   */
            OS_FlagTaskRdy(pnode, (OS_FLAGS)0);
            pnode = (OS_FLAG_NODE *)pnode->OSFlagNodeNext;
        }
        pgrp->OSFlagType     = OS_EVENT_TYPE_UNUSED;
        pgrp->OSFlagWaitList = (void *)OSFlagFreeList;/* Return group to free list          */
        OSFlagFreeList       = pgrp;
        OS_EXIT_CRITICAL();
        if (tasks_waiting == TRUE) {                  /* Reschedule only if task(s) were waiting  */
            OS_Sched();                               /* Find highest priority task ready to run  */
        }
        *err = OS_NO_ERR;
        return ((OS_FLAG_GRP *)0);                    /* Event Flag Group has been deleted       */
```

Deleting an Event Flag Group.OSFlagDel()(Cont.)

```
default:
          OS_EXIT_CRITICAL();
          *err = OS_ERR_INVALID_OPT;
          return (pgrp);
   }
}
```

Waiting for Event(s) of an Event Flag Group,OSFlagPend()

This function is called to wait for a combination of bits to be set in an event flag group.Your application can wait for ANY bit to be set or ALL bits to be set.

```
OS_FLAGS  OSFlagPend (OS_FLAG_GRP *pgrp, OS_FLAGS flags, INT8U wait_type,
INT16U timeout, INT8U *err)
{
#if OS_CRITICAL_METHOD == 3              /* Allocate storage for CPU status register */
    OS_CPU_SR      cpu_sr;
#endif
    OS_FLAG_NODE  node;
    OS_FLAGS      flags_cur;
    OS_FLAGS      flags_rdy;
    BOOLEAN       consume;
```

Waiting for Event(s) of an Event Flag Group,OSFlagPend() (Cont.)

```
if (OSIntNesting > 0) {                    /* See if called from ISR ...      */
        *err = OS_ERR_PEND_ISR;    /* ... can't PEND from an ISR   */
        return ((OS_FLAGS)0);
    }
#if OS_ARG_CHK_EN > 0
    if (pgrp == (OS_FLAG_GRP *)0)        /* Validate 'pgrp'            */
        *err = OS_FLAG_INVALID_PGRP;
        return ((OS_FLAGS)0);
    }
    if (pgrp->OSFlagType != OS_EVENT_TYPE_FLAG) {
*err = OS_ERR_EVENT_TYPE;
        return ((OS_FLAGS)0);
    }
#endif
```

Waiting for Event(s) of an Event Flag Group,OSFlagPend() (Cont.)

```
if (wait_type & OS_FLAG_CONSUME)    /* See if we need to consume the flags */
    wait_type &= ~OS_FLAG_CONSUME;
    consume    = TRUE;
} else {
    consume    = FALSE;
}
OS_ENTER_CRITICAL();
switch (wait_type) {
    case OS_FLAG_WAIT_SET_ALL:                          /* See if all required flags are set   */
        flags_rdy = pgrp->OSFlagFlags & flags;      /* Extract only the bits we want       */
        if (flags_rdy == flags) {                   /* Must match ALL the bits that we want     */
            if (consume == TRUE) {                  /* See if we need to consume the flags      */
                pgrp->OSFlagFlags &= ~flags_rdy;   /* Clear ONLY the flags that we wanted */
            }
            flags_cur = pgrp->OSFlagFlags;         /* Will return the state of the group       */
            OS_EXIT_CRITICAL();                    /* Yes, condition met, return to caller     */
            *err      = OS_NO_ERR;
            return (flags_cur);
```

Waiting for Event(s) of an Event Flag Group,OSFlagPend() (Cont.)

```
} else {                                    /* Block task until events occur or timeout */
            OS_FlagBlock(pgrp, &node, flags, wait_type, timeout);
            OS_EXIT_CRITICAL();
        }
        break;

    case OS_FLAG_WAIT_SET_ANY:
        flags_rdy = pgrp->OSFlagFlags & flags;      /* Extract only the bits we want         */
        if (flags_rdy != (OS_FLAGS)0) {             /* See if any flag set                   */
            if (consume == TRUE) {                  /* See if we need to consume the flags   */
                pgrp->OSFlagFlags &= ~flags_rdy;    /* Clear ONLY the flags that we got     */
            }
            flags_cur = pgrp->OSFlagFlags;          /* Will return the state of the group    */
            OS_EXIT_CRITICAL();                     /* Yes, condition met, return to caller  */
            *err      = OS_NO_ERR;
            return (flags_cur);
        }
```

Waiting for Event(s) of an Event Flag Group,OSFlagPend() (Cont.)

```
else {                                  /* Block task until events occur or timeout */
            OS_FlagBlock(pgrp, &node, flags, wait_type, timeout);
            OS_EXIT_CRITICAL();
         }
         break;
#if OS_FLAG_WAIT_CLR_EN > 0
      case OS_FLAG_WAIT_CLR_ALL:                 /* See if all required flags are cleared    */
         flags_rdy = ~pgrp->OSFlagFlags & flags;    /* Extract only the bits we want            */
         if (flags_rdy == flags) {                /* Must match ALL the bits that we want     */
            if (consume == TRUE) {                /* See if we need to consume the flags      */
               pgrp->OSFlagFlags |= flags_rdy;    /* Set ONLY the flags that we wanted        */
            }
            flags_cur = pgrp->OSFlagFlags;        /* Will return the state of the group       */
            OS_EXIT_CRITICAL();                   /* Yes, condition met, return to caller     */
            *err      = OS_NO_ERR;
            return (flags_cur);
         }
```

Waiting for Event(s) of an Event Flag Group,OSFlagPend() (Cont.)

```
else {                                   /* Block task until events occur or timeout */
            OS_FlagBlock(pgrp, &node, flags, wait_type, timeout);
            OS_EXIT_CRITICAL();
         }
         break;

      case OS_FLAG_WAIT_CLR_ANY:
         flags_rdy = ~pgrp->OSFlagFlags & flags;      /* Extract only the bits we want          */
         if (flags_rdy != (OS_FLAGS)0) {              /* See if any flag cleared               */
            if (consume == TRUE) {                    /* See if we need to consume the flags    */
               pgrp->OSFlagFlags |= flags_rdy;        /* Set ONLY the flags that we got         */
            }
            flags_cur = pgrp->OSFlagFlags;            /* Will return the state of the group     */
            OS_EXIT_CRITICAL();                       /* Yes, condition met, return to caller   */
            *err      = OS_NO_ERR;
            return (flags_cur);
         }
```

Waiting for Event(s) of an Event Flag Group,OSFlagPend() (Cont.)

```
else {                                    /* Block task until events occur or timeout */
              OS_FlagBlock(pgrp, &node, flags, wait_type, timeout);
              OS_EXIT_CRITICAL();
          }
          break;
#endif

      default:
          OS_EXIT_CRITICAL();
          flags_cur = (OS_FLAGS)0;
          *err      = OS_FLAG_ERR_WAIT_TYPE;
          return (flags_cur);
   }
   OS_Sched();                                      /* Find next HPT ready to run          */
   OS_ENTER_CRITICAL();
```

Waiting for Event(s) of an Event Flag Group,OSFlagPend() (Cont.)

```
if (OSTCBCur->OSTCBStat & OS_STAT_FLAG) {        /* Have we timed-out */
    OS_FlagUnlink(&node);
    OSTCBCur->OSTCBStat = OS_STAT_RDY;           /* Yes, make task ready-to-run     */
    OS_EXIT_CRITICAL();
    flags_cur          = (OS_FLAGS)0;
    *err               = OS_TIMEOUT;             /* Indicate that we timed-out waiting      */
  } else {
    if (consume == TRUE) {                       /* See if we need to consume the flags     */
      switch (wait_type) {
        case OS_FLAG_WAIT_SET_ALL:
        case OS_FLAG_WAIT_SET_ANY:               /* Clear ONLY the flags we got         */
            pgrp->OSFlagFlags &= ~OSTCBCur->OSTCBFlagsRdy;
            break;
```

Waiting for Event(s) of an Event Flag Group,OSFlagPend() (Cont.)

```
#if OS_FLAG_WAIT_CLR_EN > 0
            case OS_FLAG_WAIT_CLR_ALL:
            case OS_FLAG_WAIT_CLR_ANY:          /* Set   ONLY the flags we got      */
                pgrp->OSFlagFlags |= OSTCBCur->OSTCBFlagsRdy;
                break;
#endif
         }
      }
      flags_cur = pgrp->OSFlagFlags;
      OS_EXIT_CRITICAL();
      *err      = OS_NO_ERR;                    /* Event(s) must have occurred         */
   }
   return (flags_cur);
}
```

Adding a task to the event flag group wait list,OS_FlagBlock()

This function is internal to uC/OS-II and is used to put a task to sleep until the desired event flag bit(s) are set .

```
static  void  OS_FlagBlock (OS_FLAG_GRP *pgrp, OS_FLAG_NODE *pnode,
 OS_FLAGS flags, INT8U wait_type, INT16U timeout)
{
    OS_FLAG_NODE  *pnode_next;



    OSTCBCur->OSTCBStat      |= OS_STAT_FLAG;
    OSTCBCur->OSTCBDly       = timeout;              /* Store timeout in task's TCB           */
#if OS_TASK_DEL_EN > 0
    OSTCBCur->OSTCBFlagNode   = pnode;               /* TCB to link to node                   */
#endif
    pnode->OSFlagNodeFlags    = flags;               /* Save the flags that we need to wait for  */
    pnode->OSFlagNodeWaitType = wait_type;           /* Save the type of wait we are doing */
    pnode->OSFlagNodeTCB      = (void *)OSTCBCur;    /* Link to task's TCB                    */
    pnode->OSFlagNodeNext     = pgrp->OSFlagWaitList;
```

Adding a task to the event flag group wait list,OS_FlagBlock() (Cont.)

```
pnode->OSFlagNodePrev     = (void *)0;
   pnode->OSFlagNodeFlagGrp  = (void *)pgrp;        /* Link to Event Flag Group           */
   pnode_next                = (OS_FLAG_NODE *)pgrp->OSFlagWaitList;
   if (pnode_next != (void *)0) {                  /* Is this the first NODE to insert?         */
       pnode_next->OSFlagNodePrev = pnode;          /* No, link in doubly linked list          */

   pgrp->OSFlagWaitList = (void *)pnode;
                                         /* Suspend current task until flag(s) received   */
   if ((OSRdyTbl[OSTCBCur->OSTCBY] &= ~OSTCBCur->OSTCBBitX) == 0) {
       OSRdyGrp &= ~OSTCBCur->OSTCBBitY;
   }
}
```

Setting or Clearing Event(s) in an Event Flag group,OSFlagPost()

This function is called to set or clear some bits in an event flag group. The bits to set or clear are specified by a 'bit mask'.

```
OS_FLAGS  OSFlagPost (OS_FLAG_GRP *pgrp, OS_FLAGS flags, INT8U opt, INT8U *err)
{
#if OS_CRITICAL_METHOD == 3          /* Allocate storage for CPU status register       */
   OS_CPU_SR     cpu_sr;
#endif
   OS_FLAG_NODE *pnode;
   BOOLEAN       sched;
   OS_FLAGS      flags_cur;
   OS_FLAGS      flags_rdy;


#if OS_ARG_CHK_EN > 0
   if (pgrp == (OS_FLAG_GRP *)0) {              /* Validate 'pgrp'                          */
       *err = OS_FLAG_INVALID_PGRP;
       return ((OS_FLAGS)0);
   }
```

Setting or Clearing Event(s) in an Event Flag group,OSFlagPost() (Cont.)

```
 if (pgrp->OSFlagType != OS_EVENT_TYPE_FLAG) {
        *err = OS_ERR_EVENT_TYPE;
        return ((OS_FLAGS)0);
    }
#endif
/*$PAGE*/ ,
    OS_ENTER_CRITICAL();
    switch (opt) {
        case OS_FLAG_CLR:
             pgrp->OSFlagFlags &= ~flags;            /* Clear the flags specified in the group         */
             break;

        case OS_FLAG_SET:
             pgrp->OSFlagFlags |=  flags;            /* Set   the flags specified in the group         */
             break;
```

Setting or Clearing Event(s) in an Event Flag group,OSFlagPost() (Cont.)

```
default:
        OS_EXIT_CRITICAL();                    /* INVALID option                          */
        *err = OS_FLAG_INVALID_OPT;
        return ((OS_FLAGS)0);
  }
  sched = FALSE;                       /* Indicate that we don't need rescheduling      */
  pnode = (OS_FLAG_NODE *)pgrp->OSFlagWaitList;
  while (pnode != (OS_FLAG_NODE *)0) {    /*Go through all tasks waiting on event flag(s)*
     switch (pnode->OSFlagNodeWaitType) {
        case OS_FLAG_WAIT_SET_ALL:      /*See if all req. flags are set for current node */
           flags_rdy = pgrp->OSFlagFlags & pnode->OSFlagNodeFlags;
           if (flags_rdy == pnode->OSFlagNodeFlags) {
              if (OS_FlagTaskRdy(pnode, flags_rdy) == TRUE) {
                 sched = TRUE;                    /* When done we will reschedule   */
              }
           }
           break;
```

Setting or Clearing Event(s) in an Event Flag group,OSFlagPost() (Cont.)

```
        case OS_FLAG_WAIT_SET_ANY:              /* See if any flag set                  */
            flags_rdy = pgrp->OSFlagFlags & pnode->OSFlagNodeFlags;
            if (flags_rdy != (OS_FLAGS)0) {
                if (OS_FlagTaskRdy(pnode, flags_rdy) == TRUE) {
                    sched = TRUE;                       /* When done we will reschedule   */
                }
            }
            break;

#if OS_FLAG_WAIT_CLR_EN > 0
        case OS_FLAG_WAIT_CLR_ALL:     /* See if all req. flags are set for current node */
            flags_rdy = ~pgrp->OSFlagFlags & pnode->OSFlagNodeFlags;
            if (flags_rdy == pnode->OSFlagNodeFlags) {
                if (OS_FlagTaskRdy(pnode, flags_rdy) == TRUE) {
                   sched = TRUE;                       /* When done we will reschedule   */
                }
            }
            break;
```

Setting or Clearing Event(s) in an Event Flag group,OSFlagPost() (Cont.)

```
case OS_FLAG_WAIT_CLR_ANY:              /* See if any flag set                        */
            flags_rdy = ~pgrp->OSFlagFlags & pnode->OSFlagNodeFlags;
            if (flags_rdy != (OS_FLAGS)0) {
                if (OS_FlagTaskRdy(pnode, flags_rdy) == TRUE) {
                    sched = TRUE;                       /* When done we will reschedule   */
                }
            }
            break;
#endif
        }
        pnode = (OS_FLAG_NODE *)pnode->OSFlagNodeNext;
    }
    OS_EXIT_CRITICAL();
    if (sched == TRUE) {
        OS_Sched();
    }
    OS_ENTER_CRITICAL();
    flags_cur = pgrp->OSFlagFlags;
    OS_EXIT_CRITICAL();
    *err      = OS_NO_ERR;
    return (flags_cur);
}
```

Make a waiting task ready to run,OS_FlagTaskRdy()

This function is internal to uC/OS-II and is used to make a task ready-to-run because the desired event flag bits have been set.

```
static  BOOLEAN  OS_FlagTaskRdy (OS_FLAG_NODE *pnode, OS_FLAGS flags_rdy)
{
    OS_TCB   *ptcb;
    BOOLEAN   sched;
    ptcb                = (OS_TCB *)pnode->OSFlagNodeTCB;  /* Point to TCB of waiting task   */
    ptcb->OSTCBDly      = 0;
    ptcb->OSTCBFlagsRdy = flags_rdy;
    ptcb->OSTCBStat    &= ~OS_STAT_FLAG;
    if (ptcb->OSTCBStat == OS_STAT_RDY) {                /* Put task into ready list         */
        OSRdyGrp               |= ptcb->OSTCBBitY;
        OSRdyTbl[ptcb->OSTCBY] |= ptcb->OSTCBBitX;
        sched                   = TRUE;
    } else {
        sched                   = FALSE;
    }
    OS_FlagUnlink(pnode);
    return (sched);
}
```

Unlinking an OS_FLAG_NODE,OS_Flagunlink()

This function is internal to uC/OS-II and is used to unlink an event flag node from a list of tasks waiting for the event flag.

```
void  OS_FlagUnlink (OS_FLAG_NODE *pnode)
{
#if OS_TASK_DEL_EN > 0
    OS_TCB       *ptcb;
#endif
    OS_FLAG_GRP  *pgrp;
    OS_FLAG_NODE *pnode_prev;
    OS_FLAG_NODE *pnode_next;
    pnode_prev = (OS_FLAG_NODE *)pnode->OSFlagNodePrev;
    pnode_next = (OS_FLAG_NODE *)pnode->OSFlagNodeNext;
    if (pnode_prev == (OS_FLAG_NODE *)0) {                    /* Is it first node in wait list?      *
        pgrp                = (OS_FLAG_GRP *)pnode->OSFlagNodeFlagGrp;
        pgrp->OSFlagWaitList = (void *)pnode_next;            /*  Update list for new 1st node   */
        if (pnode_next != (OS_FLAG_NODE *)0) {
            pnode_next->OSFlagNodePrev = (OS_FLAG_NODE *)0;
        }
    }
```

Unlinking an OS_FLAG_NODE,OS_Flagunlink() (Cont.)

```
else {                                          /* No,  A node somewhere in the list   */
      pnode_prev->OSFlagNodeNext = pnode_next;         /*  Link around the node to unlink */
      if (pnode_next != (OS_FLAG_NODE *)0) {          /*  Was this the LAST node?        */
         pnode_next->OSFlagNodePrev = pnode_prev;       /*  No, Link around current node   *
      }
   }
#if OS_TASK_DEL_EN > 0
   ptcb             = (OS_TCB *)pnode->OSFlagNodeTCB;
   ptcb->OSTCBFlagNode = (OS_FLAG_NODE *)0;
#endif
}
#endif
```

Looking for Event(s) of an Event Flag Group,OSFlagAccept()

This function is called to check the status of a combination of bits to be set or cleared in an event flag group.Your application can check for ANY bit to be set/cleared or ALL bits to be set/cleared.

```
OS_FLAGS  OSFlagAccept (OS_FLAG_GRP *pgrp, OS_FLAGS flags, INT8U wait_type,
 INT8U *err)
{
#if OS_CRITICAL_METHOD == 3                        /* Allocate storage for CPU status register */
    OS_CPU_SR     cpu_sr;
#endif
    OS_FLAGS      flags_cur;
    OS_FLAGS      flags_rdy;
    BOOLEAN       consume;

#if OS_ARG_CHK_EN > 0
    if (pgrp == (OS_FLAG_GRP *)0) {                /* Validate 'pgrp'                  */
        *err = OS_FLAG_INVALID_PGRP;
        return ((OS_FLAGS)0);
    }
    if (pgrp->OSFlagType != OS_EVENT_TYPE_FLAG) {        /* Validate event block type    */
        *err = OS_ERR_EVENT_TYPE;
        return ((OS_FLAGS)0);
    }
```

Looking for Event(s) of an Event Flag Group,OSFlagAccept() (Cont.)

```
#endif
    if (wait_type & OS_FLAG_CONSUME) {              /* See if we need to consume the flags      */
        wait_type &= ~OS_FLAG_CONSUME;
        consume    = TRUE;
    } else {
        consume    = FALSE;
    }
/*$PAGE*/ ,
    *err = OS_NO_ERR;                              /* Assume NO error until proven otherwise.  */
    OS_ENTER_CRITICAL();
    switch (wait_type) {
        case OS_FLAG_WAIT_SET_ALL:                         /* See if all required flags are set        */
             flags_rdy = pgrp->OSFlagFlags & flags;      /* Extract only the bits we want            */
             if (flags_rdy == flags) {                 /* Must match ALL the bits that we want     */
                 if (consume == TRUE) {                /* See if we need to consume the flags      */
                     pgrp->OSFlagFlags &= ~flags_rdy;   /* Clear ONLY the flags that we wanted      */
                 }
             }
```

Looking for Event(s) of an Event Flag Group,OSFlagAccept() (Cont.)

```
else {
        *err  = OS_FLAG_ERR_NOT_RDY;
      }
      flags_cur = pgrp->OSFlagFlags;              /* Will return the state of the group       */
      OS_EXIT_CRITICAL();
      break;

   case OS_FLAG_WAIT_SET_ANY:
      flags_rdy = pgrp->OSFlagFlags & flags;       /* Extract only the bits we want             */
      if (flags_rdy != (OS_FLAGS)0) {              /* See if any flag set                  */
        if (consume == TRUE) {                     /* See if we need to consume the flags      */
          pgrp->OSFlagFlags &= ~flags_rdy;       /* Clear ONLY the flags that we got         */
        }
      } else {
        *err  = OS_FLAG_ERR_NOT_RDY;
      }
```

Looking for Event(s) of an Event Flag Group,OSFlagAccept() (Cont.)

```
 flags_cur = pgrp->OSFlagFlags;                /* Will return the state of the group       */
            OS_EXIT_CRITICAL();
            break;

#if OS_FLAG_WAIT_CLR_EN > 0
       case OS_FLAG_WAIT_CLR_ALL:                       /* See if all required flags are cleared    */
            flags_rdy = ~pgrp->OSFlagFlags & flags;      /* Extract only the bits we want            */
            if (flags_rdy == flags) {                   /* Must match ALL the bits that we want     */
                if (consume == TRUE) {                  /* See if we need to consume the flags      */
                    pgrp->OSFlagFlags |= flags_rdy;      /* Set ONLY the flags that we wanted        */
                }
            } else {
                *err  = OS_FLAG_ERR_NOT_RDY;
            }
            flags_cur = pgrp->OSFlagFlags;               /* Will return the state of the group       */
            OS_EXIT_CRITICAL();
            break;
```

Looking for Event(s) of an Event Flag Group,OSFlagAccept() (Cont.)

```
    case OS_FLAG_WAIT_CLR_ANY:
        flags_rdy = ~pgrp->OSFlagFlags & flags;      /* Extract only the bits we want           */
        if (flags_rdy != (OS_FLAGS)0) {               /* See if any flag cleared                 */
            if (consume == TRUE) {                    /* See if we need to consume the flags     */
                pgrp->OSFlagFlags |= flags_rdy;       /* Set ONLY the flags that we got          */
            }
        } else {
            *err  = OS_FLAG_ERR_NOT_RDY;
        }
        flags_cur = pgrp->OSFlagFlags;                /* Will return the state of the group      */
        OS_EXIT_CRITICAL();
        break;
#endif
```

Looking for Event(s) of an Event Flag Group,OSFlagAccept() (Cont.)

```
default:
             OS_EXIT_CRITICAL();
             flags_cur = (OS_FLAGS)0;
             *err      = OS_FLAG_ERR_WAIT_TYPE;
             break;
    }
    return (flags_cur);
}
#endif
```

Querying an Event Flag Group,OSFlagQuery()

This function is used to check the value of the event flag group.

```
OS_FLAGS  OSFlagQuery (OS_FLAG_GRP *pgrp, INT8U *err)
{
#if OS_CRITICAL_METHOD == 3              /* Allocate storage for CPU status register    */
    OS_CPU_SR  cpu_sr;
#endif
    OS_FLAGS   flags;


#if OS_ARG_CHK_EN > 0
    if (pgrp == (OS_FLAG_GRP *)0) {              /* Validate 'pgrp'                          */
        *err = OS_FLAG_INVALID_PGRP;
        return ((OS_FLAGS)0);
    }
    if (pgrp->OSFlagType != OS_EVENT_TYPE_FLAG) { /* Validate event block type        */
        *err = OS_ERR_EVENT_TYPE;
        return ((OS_FLAGS)0);
    }
#endif
```

Querying an Event Flag Group,OSFlagQuery(),(Cont.)

```
OS_ENTER_CRITICAL();
    flags = pgrp->OSFlagFlags;
    OS_EXIT_CRITICAL();
    *err = OS_NO_ERR;
    return (flags);                          /* Return the current value of the event flags       */
}
#endif
```