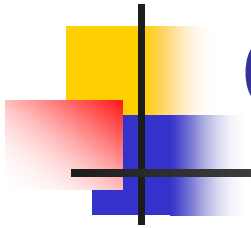
Chapter 11
Message Queue Management

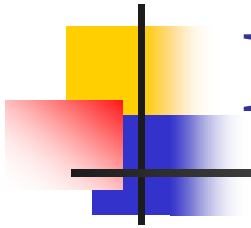
Yen-Ting Liu

2008/08/12



Outline

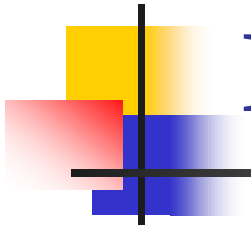
- Introduce
- Creating a Message Queue
- Deleting a Message Queue
- Waiting for a Message at a Queue
- Sending a Message to a Queue (FIFO)
- Sending a Message to a Queue (LIFO)
- Sending a Message to a Queue (Broadcast, FIFO or LIFO)
- Getting a Message Without Waiting
- Flushing a Queue
- Obtaining the status of a Queues
- Using a Message Queue When Reading Analog Inputs
- Using a Queue as a Counting Semaphore



Introduce

- Message queue is a μC/OS-II object
- Provide 9 services to access message queue

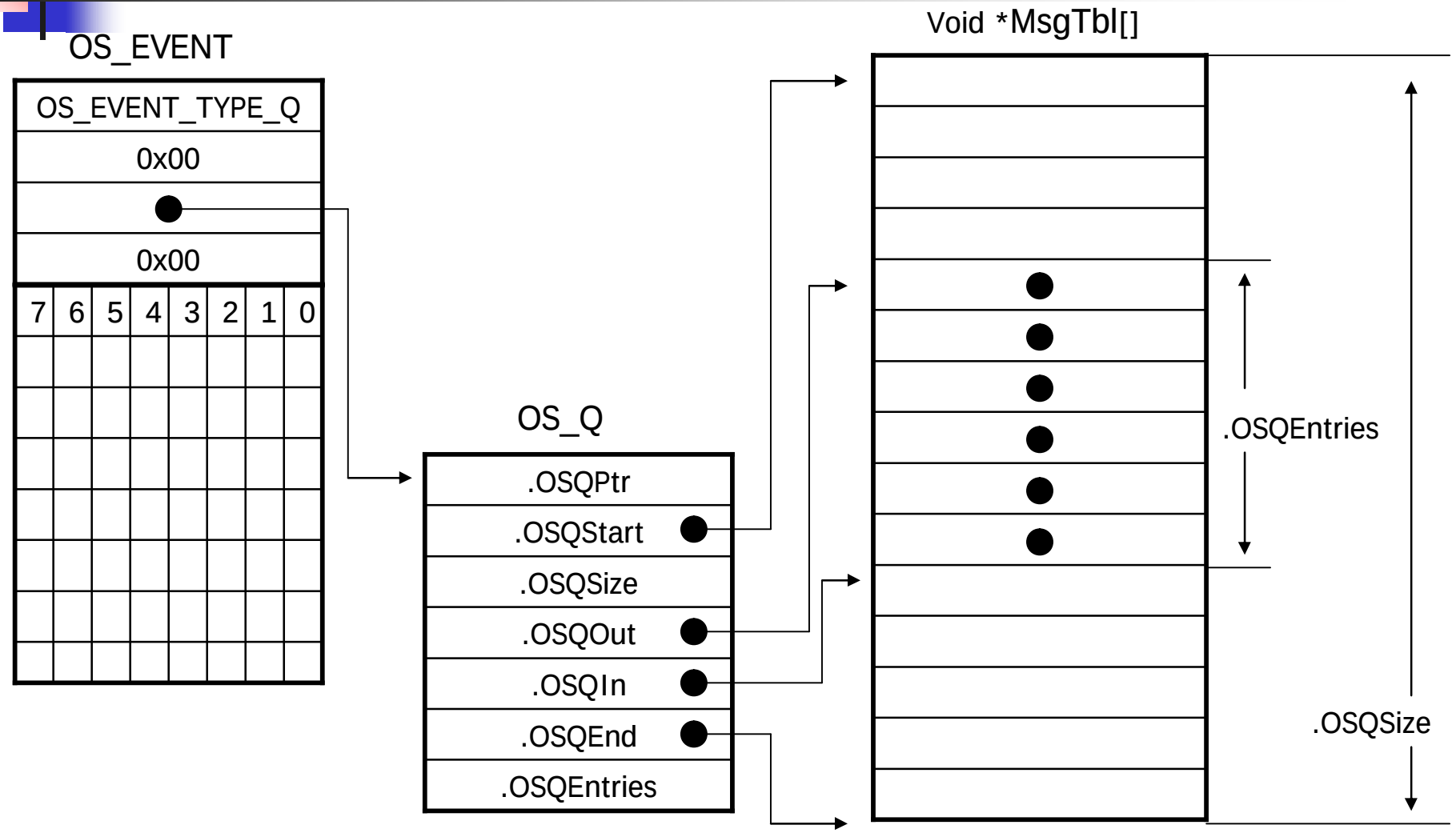
  OSQAccept()
  OSQCreate()
  OSQDel()
  OSQFlush()
  OSQPend()
  OSQPost()
  OSQPostFront()
  OSQPostOpt()
  OSQuery()



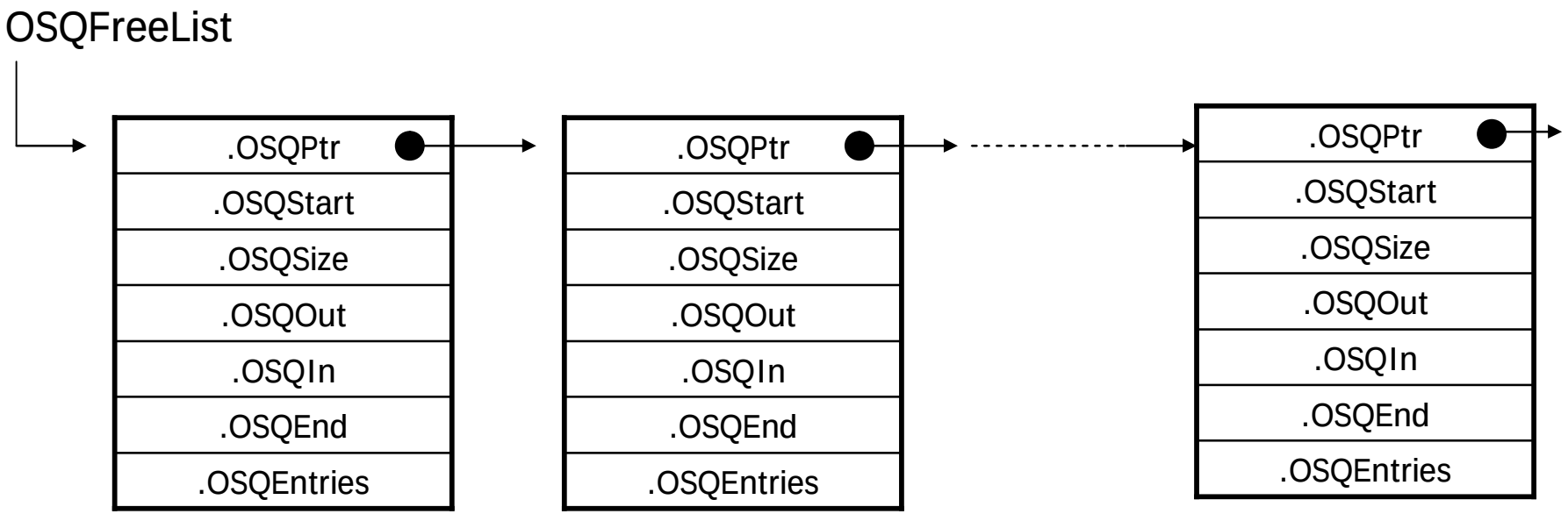
Introduce (cont.)

μC/OS-II Event Flag Service	Enabled when set 1 in OS_CFG.H
OSQAccetp()	OS_Q_ACCEPT_EN
OSQCreate()	
OSQDel()	OS_Q_DEL_EN
OSQFlush()	OS_Q_FLUSH_EN
OSQPend()	
OSQPost()	OS_Q_POST_EN
OSQPostFront()	OS_Q_POST_FRONT_EN
OSQPostOpt()	OS_Q_POST_OPT_EN
OSQQuery()	OS_Q_QUERY_EN

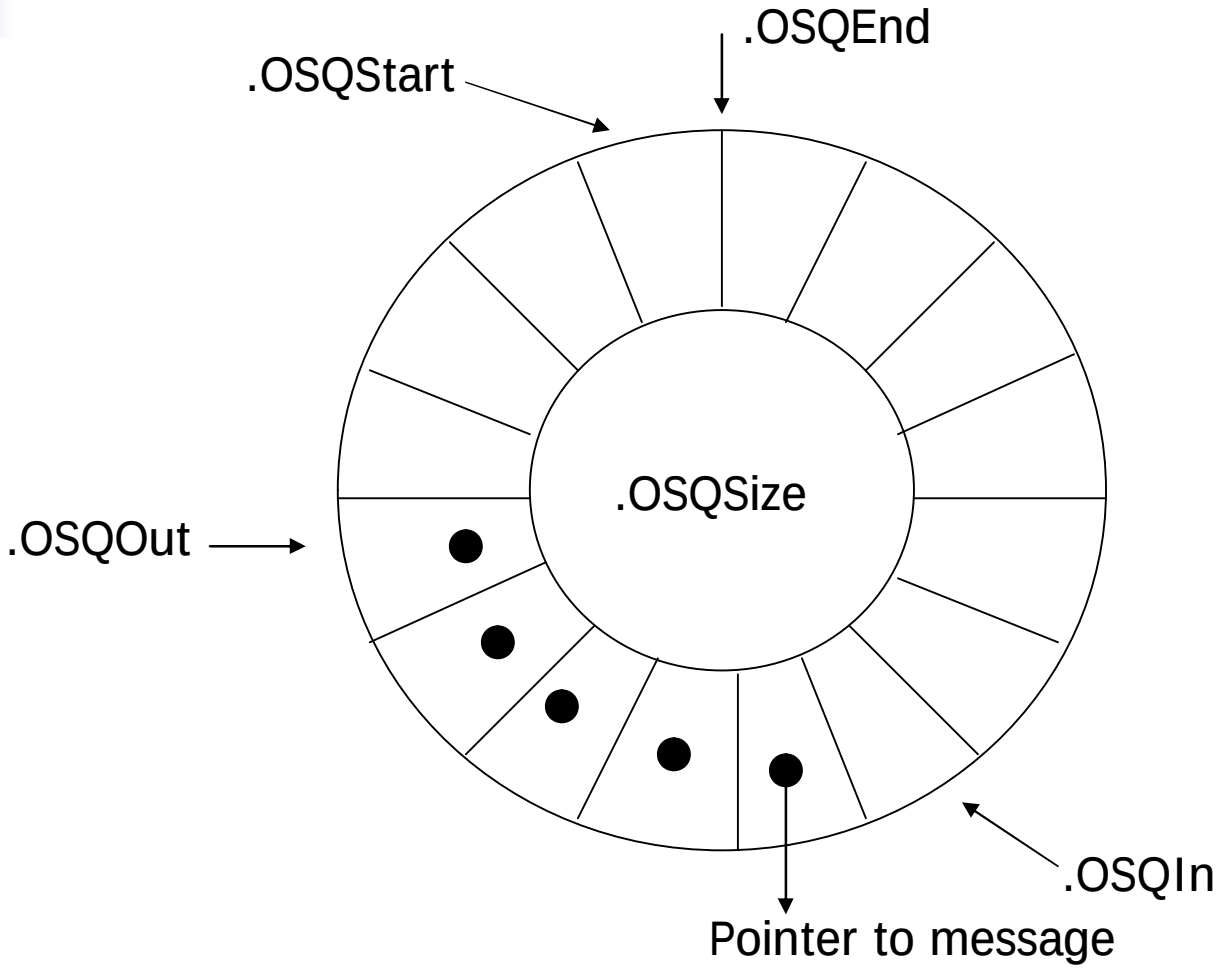
Introduce (cont.)



Introduce (cont.)



Introduce (cont.)



Creating a Message Queue OSQCreate()

```
OS_EVENT *OSQCreate (void **start, INT16U size)
{
#if OS_CRITICAL_METHOD == 3
    OS_CPU_SR cpu_sr;
#endif
    OS_EVENT *pevent;
    OS_Q *pq;
// not calling from an ISR
    if (OSIntNesting > 0) {
        return ((OS_EVENT *)0);
    }
    OS_ENTER_CRITICAL();
    pevent = OSEventFreeList;              // obtain an ECB from the free list of ECB
    if (OSEventFreeList != (OS_EVENT *)0) {
        OSEventFreeList = (OS_EVENT *)OSEventFreeList->OSEventPtr;
    }
    OS_EXIT_CRITICAL();
```

Creating a Message Queue OSQCreate() (cont.)

```
if (pevent != (OS_EVENT *)0) {          /* if ECB available
    OS_ENTER_CRITICAL();                   allocate a queue control block from
    pq = OSQFreeList;                      free list of queue control blocks */
    if (pq != (OS_Q *)0) {
        OSQFreeList = OSQFreeList->OSQPtr;
        OS_EXIT_CRITICAL();
        pq->OSQStart = start;
        pq->OSQEnd = &start[size];
        pq->OSQIn = start;
        pq->OSQOut = start;
        pq->OSQSize = size;
        pq->OSQEntries = 0;
        pevent->OSEventType = OS_EVENT_TYPE_Q;
        pevent->OSEventCnt = 0;
        pevent->OSEventPtr = pq;
        OS_EventWaitListInit(pevent);
    }
```

Creating a Message Queue OSQCreate() (cont.)

```
        else {                          // queue control block is not available
                pevent->OSEventPtr = (void *)OSEventFreeList;
                OSEventFreeList = pevent;
                OS_EXIT_CRITICAL();
                pevent = (OS_EVENT *)0;
        }
    }
    return (pevent);
}
```

Deleting a Message Queue OSQDel()

```
OS_EVENT *OSQDel (OS_EVENT *pevent, INT8U opt, INT8U *err)
{
#if OS_CRITICAL_METHOD == 3
    OS_CPU_SR cpu_sr;
#endif
    BOOLEAN tasks_waiting;
    OS_Q *pq;
// not calling fomr an ISR
    if (OSIntNesting > 0) {
        *err = OS_ERR_DEL_ISR;
        return ((OS_EVENT *)0);
    }
#if OS_ARG_CHK_EN > 0                    // validate pevent and ECB
    if(pevent == (OS_EVENT *)0) {
        *err = OS_ERR_PREVENT_NULL;
        return (pevent);
    }
```

Deleting a Message Queue OSQDel() (cont.)

```
    if (pevent->OSEventType != OS_EVENT_TYPE_Q) {
            *err = OS_ERR_EVENT_TYPE;
            return (pevent) ;
    }
#endif
     OS_ENTER_CRITICAL();
    if (pevent->OSEventGrp != 0x00) {                    // task waiting the queue ?
            tasks_waiting = TRUE;
    } else {
            tasks_waiting = FALSE;
    }
    switch (opt) {
            case OS_DEL_NO_PEND;     // delete the queue only if no tasks are pending
              if (tasks_waiting == FALSE) {
                      pq = (OS_Q *)pevent->OSEventPtr;   /* return the queue control
                      pq->OSQPtr = OSQFreeList;              block to the free list */
```

Deleting a Message Queue OSQDel() (cont.)

```
            OSQFreeList = pq;
            pevent->OSEventType = OS_EVENT_TYPE_UNUSED;
            pevent->OSEventPtr = OSEventFreeList;       /* return the ECB to
            OSEventFreeList = pevent;                      the freelist */
            OS_EXIT_CRITICAL();
            *err = OS_NO_ERR;
            return ((OS_EVENT *)0);
        } else {                                    // return an error code
            OS_EXIT_CRITICAL();
            *err = OS_ERR_TASK_WAITING;
            return (pevent);
        }
    case OS_DEL_ALWAYS:                 // always delete
        while (pevent->OSEventGrp != 0x00) {                /* all tasks waiting on
            OS_EventTaskRdy(pevent, (void *)0, OS_STAT_Q);    the queue are readied */
        }
```

Deleting a Message Queue OSQDel() (cont.)

```
            pq = (OS_Q *)pevent->OSEventPtr;          /* return queue control block
            pq->OSQPtr = OSQFreeList;                    and ECB */
            OSQFreeList = pq;
            pevent->OSEventType = OS_EVENT_TYPE_UNUSED;
            pevent->OSEventPtr = OSEventFreeList;
            OSEventFreeList = pevent;
            OS_EXIT_CRITICAL();
            if (tasks_waiting == TRUE) {
                OS_Sched();
            }
            *err = OS_NO_ERR;
            return ((OS_EVENT *)0);
        default:
            OS_EXIT_CRITICAL();
            *err = OS_ERR_INVALID_OPT;
            return (pevent);
        }
```

Waiting for a Message at a Queue OSQPend()

```
void *OSQPend (OS_EVENT *pevent, INT16U timeout, INT8U *err)
{
#if OS_CRITICAL_METHOD == 3 /* Allocate storage for CPU status register */
    OS_CPU_SR cpu_sr;
#endif
    void *msg;
    OS_Q *pq;

    if (OSIntNesting > 0) {
        *err = OS_ERR_DEL_ISR;
        return ((void *)0);
    }
#if OS_ARG_CHK_EN > 0
    if(pevent == (OS_EVENT *)0) {
        *err = OS_ERR_PREVENT_NULL;
        return ((void *)0);
    }
```

Waiting for a Message at a Queue OSQPend() (cont.)

```
    if (pevent->OSEventType != OS_EVENT_TYPE_Q) {
        *err = OS_ERR_EVENT_TYPE;
        return ((void *)0);
    }
#endif
    OS_ENTER_CRITICAL();
    pq = (OS_Q *)pevent->OSEventPtr;
    if (pq->OSQEntries > 0) {          // message is available when OSQEntries > 0
        msg = *pq->OSQOut++;           /* store message and
        pq->OSQEntries--;                                   move OSQout point */
        if (pq->OSQOut == pq->OSQEnd) {
            pq->OSQOut = pq->OSQStart;
        }
        OS_EXIT_CRITICAL();
        *err = OS_NO_ERR;
        return (msg);
    }
```

Waiting for a Message at a Queue OSQPend() (cont.)

```
// message queue is empty
    OSTCBCur->OSTCBStat |= OS_STAT_Q;   // set the status flag in the task's TCB
    OSTCBCur->OSTCBDly = timeout;       // set timeout
    OS_EventTaskWait(pevent);
    OS_EXIT_CRITICAL();
    OS_Sched();                         // call scheduler
    OS_ENTER_CRITICAL();
    msg = OSTCBCur->OSTCBMsg;
    if (msg != (void *)0) {             // receive message
        OSTCBCur->OSTCBMsg = (void *)0;
        OSTCBCur->OSTCBStat = OS_STAT_RDY;
        OSTCBCur->OSTCBEventPtr = (OS_EVENT *)0;
        OS_EXIT_CRITICAL();
        *err = OS_NO_ERR;
        return (msg);
    }
```

Waiting for a Message at a Queue OSQPend() (cont.)

```
    OS_EventTO(pevent);                    // timeout
    OS_EXIT_CRITICAL();
    *err = OS_TIMEOUT;
    return ((void *)0);
}
```

Sending a Message to a Queue (FIFO) OSQPost()

```
INT8U OSQPost (OS_EVENT *pevent, void *msg)
{
#if OS_CRITICAL_METHOD == 3
    OS_CPU_SR cpu_sr;
#endif
    OS_Q *pq;
#if OS_ARG_CHK_EN > 0                    //check pevent, message and ECB
    if (pevent == (OS_EVENT *)0) {
        return (OS_ERR_PEVENT_NULL);
    }
    if (msg == (void *)0) {
        return (OS_ERR_POST_NULL_PTR);
    }
    if (pevent->OSEventType != OS_EVENT_TYPE_Q) {
        return (OS_ERR_EVENT_TYPE);
    }
#endif
```

Sending a Message to a Queue (FIFO) OSQPost() (cont.)

```
    OS_ENTER_CRITICAL();
    if (pevent->OSEventGrp != 0x00) {          // task wait message ?
        OS_EventTaskRdy(pevent, msg, OS_STAT_Q);
        OS_EXIT_CRITICAL();
        OS_Sched();
        return (OS_NO_ERR);
    }  // if no task wait message , store message in the message queue
    pq = (OS_Q *)pevent->OSEventPtr;
    if (pq->OSQEntries >= pq->OSQSize) {          // message queue is full ?
        OS_EXIT_CRITICAL();
        return (OS_Q_FULL);
    }  // message queue is not full
    *pq->OSQIn++ = msg;
    pq->OSQEntries++;
    if (pq->OSQIn == pq->OSQEnd) {
        pq->OSQIn = pq->OSQStart;
    }
    OS_EXIT_CRITICAL();
    return (OS_NO_ERR);
}
```

Sending a Message to a Queue (LIFO) OSQFront()

```
INT8U OSQPost (OS_EVENT *pevent, void *msg)
{
#if OS_CRITICAL_METHOD == 3
    OS_CPU_SR cpu_sr;
#endif
    OS_Q *pq;
#if OS_ARG_CHK_EN > 0                    //check pevent, message and ECB
    if (pevent == (OS_EVENT *)0) {
            return (OS_ERR_PEVENT_NULL);
    }
    if (msg == (void *)0) {
            return (OS_ERR_POST_NULL_PTR);
    }
    if (pevent->OSEventType != OS_EVENT_TYPE_Q) {
            return (OS_ERR_EVENT_TYPE);
    }
#endif
```

Sending a Message to a Queue (LIFO) OSQFront() (cont).

```
OS_ENTER_CRITICAL();
if (pevent->OSEventGrp != 0x00) {          // task wait message ?
        OS_EventTaskRdy(pevent, msg, OS_STAT_Q);
        OS_EXIT_CRITICAL();
        OS_Sched();
        return (OS_NO_ERR);
}  // if no task wait message , store message in the message queue
pq = (OS_Q *)pevent->OSEventPtr;
if (pq->OSQEntries >= pq->OSQSize) {          // message queue is full ?
        OS_EXIT_CRITICAL();
        return (OS_Q_FULL);
}  // message queue is not full
if (pq->OSQOut == pq->OSQStart) {
        pq->OSQOut = pq->OSQEnd;
}
```

Sending a Message to a Queue (LIFO) OSQFront() (cont).

```
    pq->OSQOut--;
    *pq->OSQOut = msg;
    pq->OSQEntries++;
    OS_EXIT_CRITICAL();
    return (OS_NO_ERR);
}
```

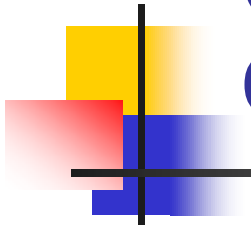
Sending a Message to a Queue (Broadcast, FIFO or LIFO) OSQPostOpt()

```
INT8U OSQPostOpt (OS_EVENT *pevent, void *msg, INT8U opt)
{
#if OS_CRITICAL_METHOD == 3
    OS_CPU_SR cpu_sr;
#endif
    OS_Q *pq;
#if OS_ARG_CHK_EN > 0
    if (pevent == (OS_EVENT *)0) {
        return (OS_ERR_PEVENT_NULL);
    }
    if (msg == (void *)0) {
        return (OS_ERR_POST_NULL_PTR);
    }
    if (pevent->OSEventType != OS_EVENT_TYPE_Q) {
        return (OS_ERR_EVENT_TYPE);
    }
#endif
```

Sending a Message to a Queue (Broadcast, FIFO or LIFO) OSQPostOpt() (cont.)

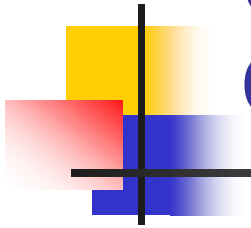
```
OS_ENTER_CRITICAL();
if (pevent->OSEventGrp != 0x00) {                  // task wait message ?
        if ((opt & OS_POST_OPT_BROADCAST) != 0x00) {     // opt set broadcast
                while (pevent->OSEventGrp != 0x00) {
                        OS_EventTaskRdy(pevent, msg, OS_STAT_Q);
                }
        } else {        // only highest priority
                OS_EventTaskRdy(pevent, msg, OS_STAT_Q);
        }
        OS_EXIT_CRITICAL();
        OS_Sched();
        return (OS_NO_ERR);
}
```

Sending a Message to a Queue (Broadcast, FIFO or LIFO) OSQPostOpt() (cont.)



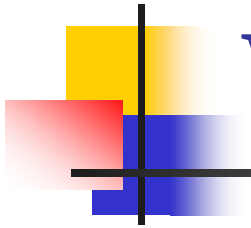
```
pq = (OS_Q *)pevent->OSEventPtr;
if (pq->OSQEntries >= pq->OSQSize) {        // message queue full ?
        OS_EXIT_CRITICAL();
        return (OS_Q_FULL);
}
if ((opt & OS_POST_OPT_FRONT) != 0x00) {    // opt set front (LIFO)
        if (pq->OSQOut == pq->OSQStart) {
                pq->OSQOut = pq->OSQEnd;
        }
        pq->OSQOut--;
        *pq->OSQOut = msg;
        } else {                // FIFO
                *pq->OSQIn++ = msg;
                if (pq->OSQIn == pq->OSQEnd) {
                pq->OSQIn = pq->OSQStart;
        }
}
```

Sending a Message to a Queue (Broadcast, FIFO or LIFO) OSQPostOpt() (cont.)



```
    pq->OSQEntries++;
    OS_EXIT_CRITICAL();
    return (OS_NO_ERR);
}
```

Getting a Message Without Waiting OSQAccept()



```
void *OSQAccept (OS_EVENT *pevent)
{
#if OS_CRITICAL_METHOD == 3
    OS_CPU_SR cpu_sr;
#endif
    void *msg;
    OS_Q *pq;
#if OS_ARG_CHK_EN > 0
    if (pevent == (OS_EVENT *)0) {
        return ((void *)0);
    }
    if (pevent->OSEventType != OS_EVENT_TYPE_Q) {
        return ((void *)0);
    }
#endif
```

Getting a Message Without Waiting OSQAccept() (cont.)

```
    OS_ENTER_CRITICAL();
    pq = (OS_Q *)pevent->OSEventPtr;
    if (pq->OSQEntries > 0) {
        msg = *pq->OSQOut++;
        pq->OSQEntries--;
        if (pq->OSQOut == pq->OSQEnd) {
            pq->OSQOut = pq->OSQStart;
        }
    } else {
        msg = (void *)0;
    }
    OS_EXIT_CRITICAL();
    return (msg);
}
```

Flushing a Queue
OSQFlush()

```
INT8U OSQFlush (OS_EVENT *pevent)
{
#if OS_CRITICAL_METHOD == 3
    OS_CPU_SR cpu_sr;
#endif

    OS_Q *pq;
#if OS_ARG_CHK_EN > 0
    if (pevent == (OS_EVENT *)0) {
        return (OS_ERR_PEVENT_NULL);
    }
    if (pevent->OSEventType != OS_EVENT_TYPE_Q) {
        return (OS_ERR_EVENT_TYPE);
    }
#endif
```

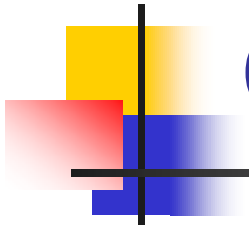
Flushing a Queue
OSQFlush() (cont.)

```
    OS_ENTER_CRITICAL();
    pq = (OS_Q *)pevent->OSEventPtr;
    pq->OSQIn = pq->OSQStart;
    pq->OSQOut = pq->OSQStart;
    pq->OSQEntries = 0;
    OS_EXIT_CRITICAL();
}
```

Obtaining the status of a Queues OSQQuery()

```
INT8U OSQQuery (OS_EVENT *pevent, OS_Q_DATA *pdata)
{
#if OS_CRITICAL_METHOD == 3
    OS_CPU_SR cpu_sr;
#endif
    OS_Q *pq;
    INT8U *psrc;
    INT8U *pdest;
#if OS_ARG_CHK_EN > 0
    if (pevent == (OS_EVENT *)0) {
        return (OS_ERR_PEVENT_NULL);
    }
    if (pevent->OSEventType != OS_EVENT_TYPE_Q) {
        return (OS_ERR_EVENT_TYPE);
    }
#endif
```

Obtaining the status of a Queues OSQQuery() (cont.)



```
    OS_ENTER_CRITICAL();
    pdata->OSEventGrp = pevent->OSEventGrp;      // copy waiting list
    psrc = &pevent->OSEventTbl[0];
    pdest = &pdata->OSEventTbl[0];
#if OS_EVENT_TBL_SIZE > 0
    *pdest++ = *psrc++;
#endif
#if OS_EVENT_TBL_SIZE > 1
    *pdest++ = *psrc++;
#endif
#if OS_EVENT_TBL_SIZE > 2
    *pdest++ = *psrc++;
#endif
#if OS_EVENT_TBL_SIZE > 3
    *pdest++ = *psrc++;
#endif
```

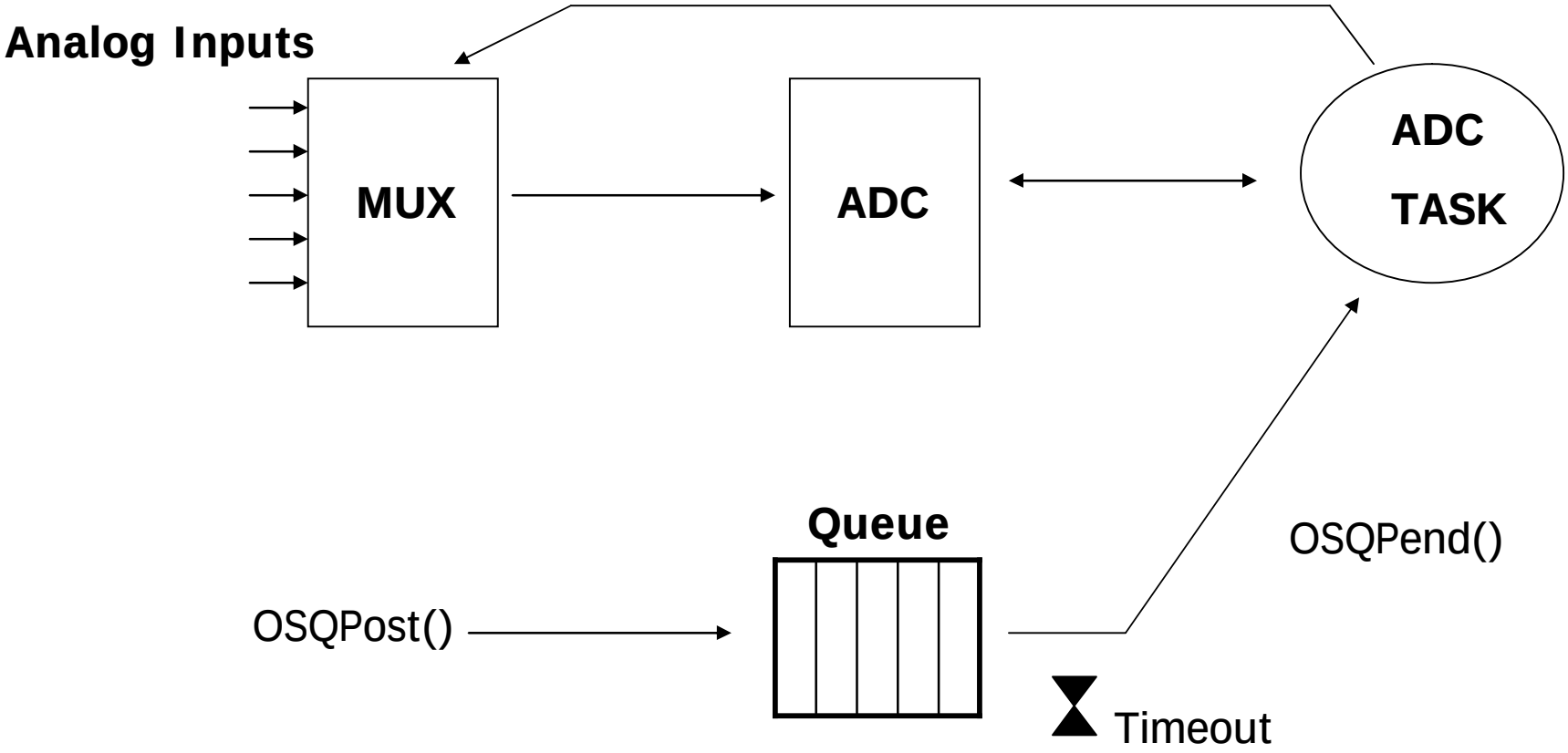
Obtaining the status of a Queues OSQQuery() (cont.)

```
#if OS_EVENT_TBL_SIZE > 4
    *pdest++ = *psrc++;
#endif
#if OS_EVENT_TBL_SIZE > 5
    *pdest++ = *psrc++;
#endif
#if OS_EVENT_TBL_SIZE > 6
    *pdest++ = *psrc++;
#endif
#if OS_EVENT_TBL_SIZE > 7
    *pdest = *psrc;
#endif
    pq = (OS_Q *)pevent->OSEventPtr;
    if (pq->OSQEntries > 0) {                    // copy message
        pdata->OSMsg = *pq->OSQOut;
    } else {
        pdata->OSMsg = (void *)0;
    }
```

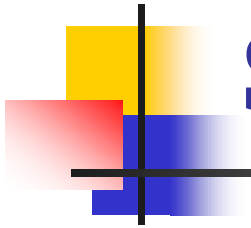
Obtaining the status of a Queues OSQQuery() (cont.)

```
    pdata->OSNMsgs = pq->OSQEntries;          // number of entries
    pdata->OSQSize = pq->OSQSize;             // queue size
    OS_EXIT_CRITICAL();
    return (OS_NO_ERR);
}
```

Using a Message Queue When Reading Analog Inputs



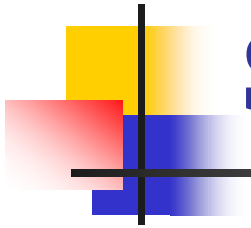
Using a Queue as a Counting Semaphore



```
OS_EVENT *QSem
Void *QMsgTbl [N_RESOURCES]

void main (void)
{
    OSInit();
    .
    QSem = OSQCreate(&QMsgTbl[0], N_RESOURCES);
    for (i = 0; i < N_RESOURCES; i++) {
            OSQPost(Qsem, (void *)1);
    }
    .
    OSTaskCreate(Task1, .., .., ..);
    .
    OSStart();
}
```

Using a Queue as a Counting Semaphore (cont.)



```
void Task1 (void *pdata)
{
    INT8U err;

    for (;;) {
        OSQPend(&QSem, 0, &err);.   // obtain access to resource
        ..                          // task has semaphore, access resource
        OSMQPost(QSem, (void )1);   // release access to resource
    }
}
```

The End