# MicroC/OS-II Chapter 6

中興資科所 楊智閔

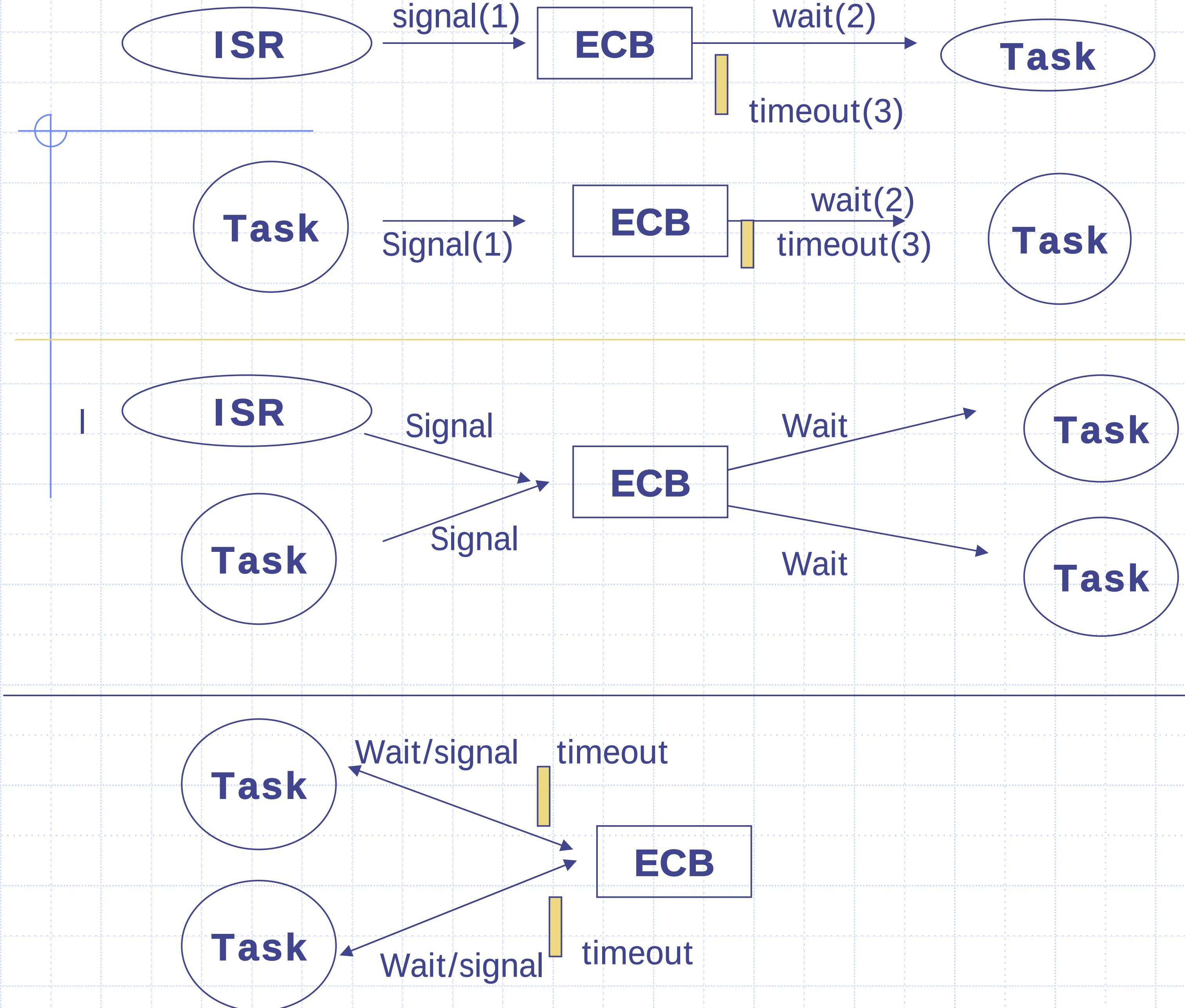
學號:79256029

# Chapter 6 Event Control Block

# Event Control Blocks

- Description:
- tasks and ISRs can interact with each other.
- A Task or an ISR Signals a Task  Through a Kernel Object

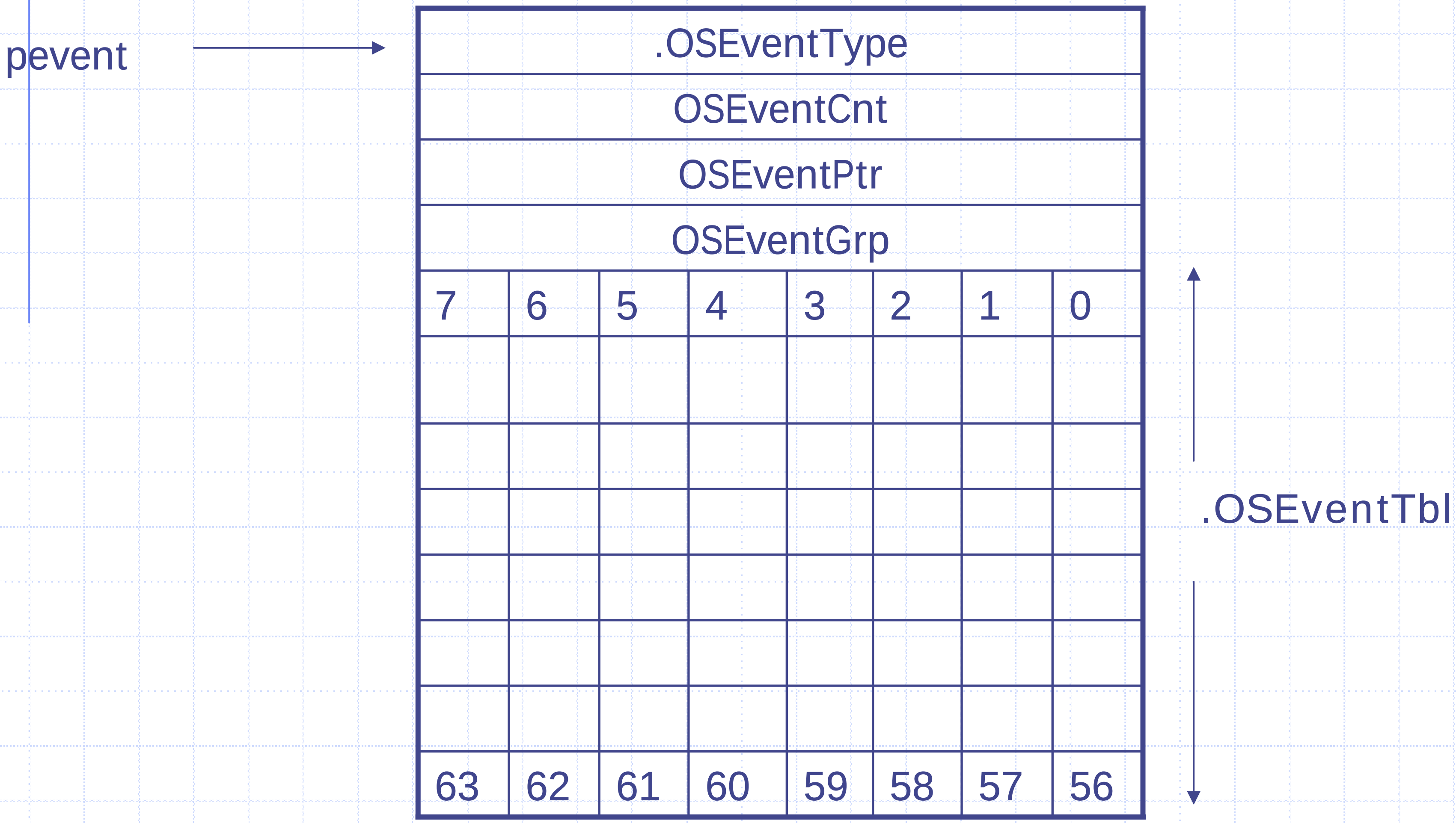
# Figure 6.1 Use of event control blocks



# Listing 6.1   Event control block data structure

```
typedef structure {

    INT8U OSEventType                                      /* Event Type*/

    INT8U  OSEventGrp;                                      /*Group */

    Int16U OSEventCnt;                            /*Count ( When event is a semphore_*/

   void *OSEventPtr;                              /*ptr to message or queue structure*/

    INT8U OSEventTbl[OS_Event_TBL_SIZE];           /*wait list for event to occur*/
 }  OS_Event;
```

# OS_EVENT data structure



pevent

.OSEventType

OSEventCnt

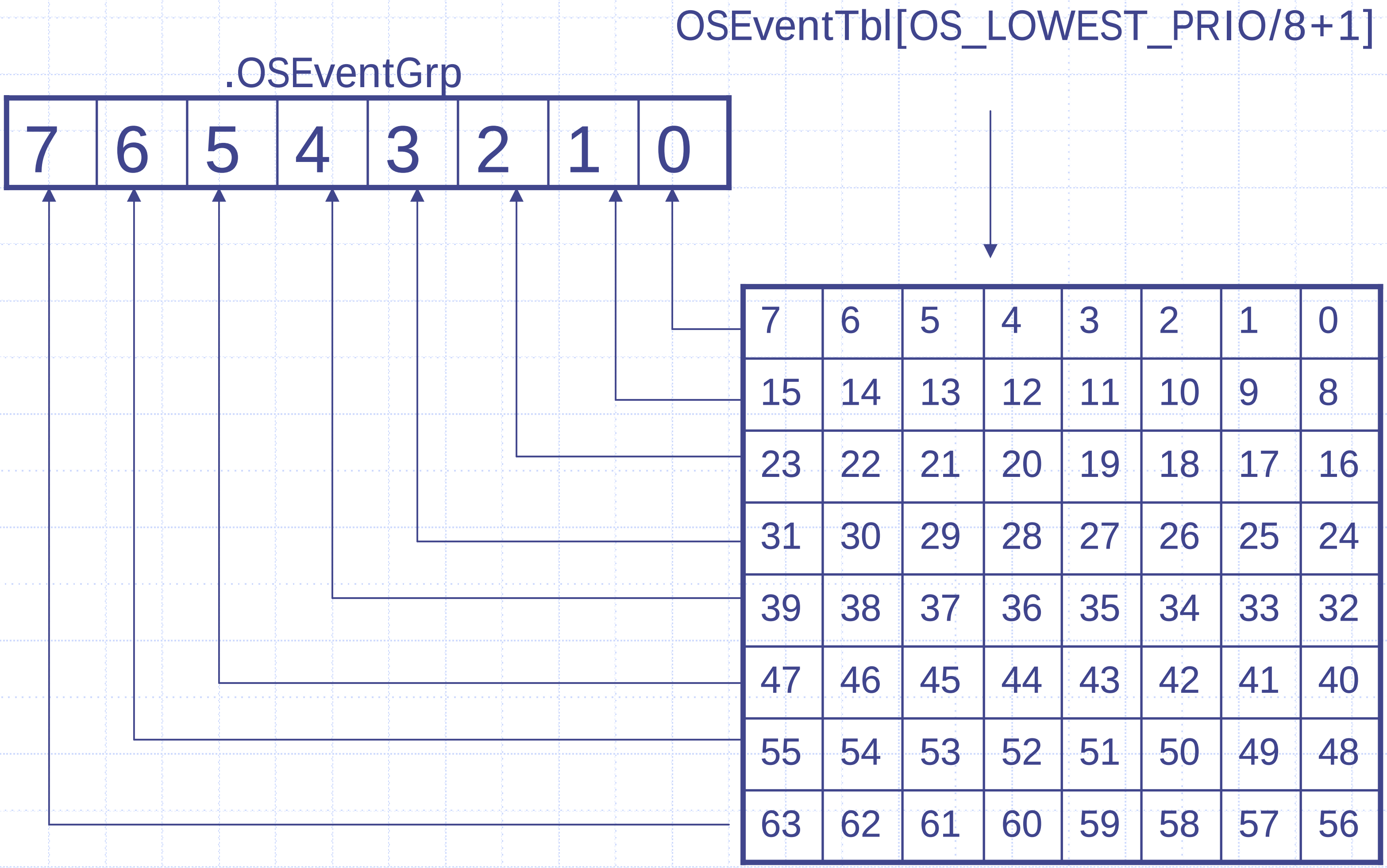
OSEventPtr

OSEventGrp

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| | | | | | | | |
| | | | | | | | |
| | | | | | | | |
| | | | | | | | |
| | | | | | | | |
| | | | | | | | |
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 |

.OSEventTbl

# Figure6.3
# Wait list for task waiting for an event to occur



| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
| 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 |
| 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 |
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 |

# 6.00 Placing a Task in the ECB Wait List

- Listing 6.2 Making a task wait for an event

算出是哪一個group,並加入自己是哪一個group

```
pevent=->OSEventGrp      |= OSMapTbl[prio >>3];

Pevent ->OSEventTbl[prio>>3]        |= OSMapTbl[prio & 0x07];
```

計算是哪一個columns

取出是哪一個Row,

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
| 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 |
| 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 |
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 |

# Table 6.1 Content of OSMapTbl[]

OSMapTbl[]

| Index | Bit Mask(Binary) |
|---|---|
| 0 | 00000001 |
| 1 | 00000010 |
| 2 | 00000100 |
| 3 | 00001000 |
| 4 | 00010000 |
| 5 | 00100000 |
| 6 | 01000000 |
| 7 | 10000000 |

# 6.01 Removing a Task from the ECB Wait List

- Listing 6.3 Removing a task form a wait list.

```
  if ((pevent -> OSEventTbl[prio >>3]  &= ~OSMapTbl[prio & 0x07]) ==0)
{
  pevent ->OSEventGrp  &= ~OSMapTbl[prio >> 3];
  }
```

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
| 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 |
| 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 |
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 |

# 6.02 Finding the Highest Priority Task Waiting on an ECB

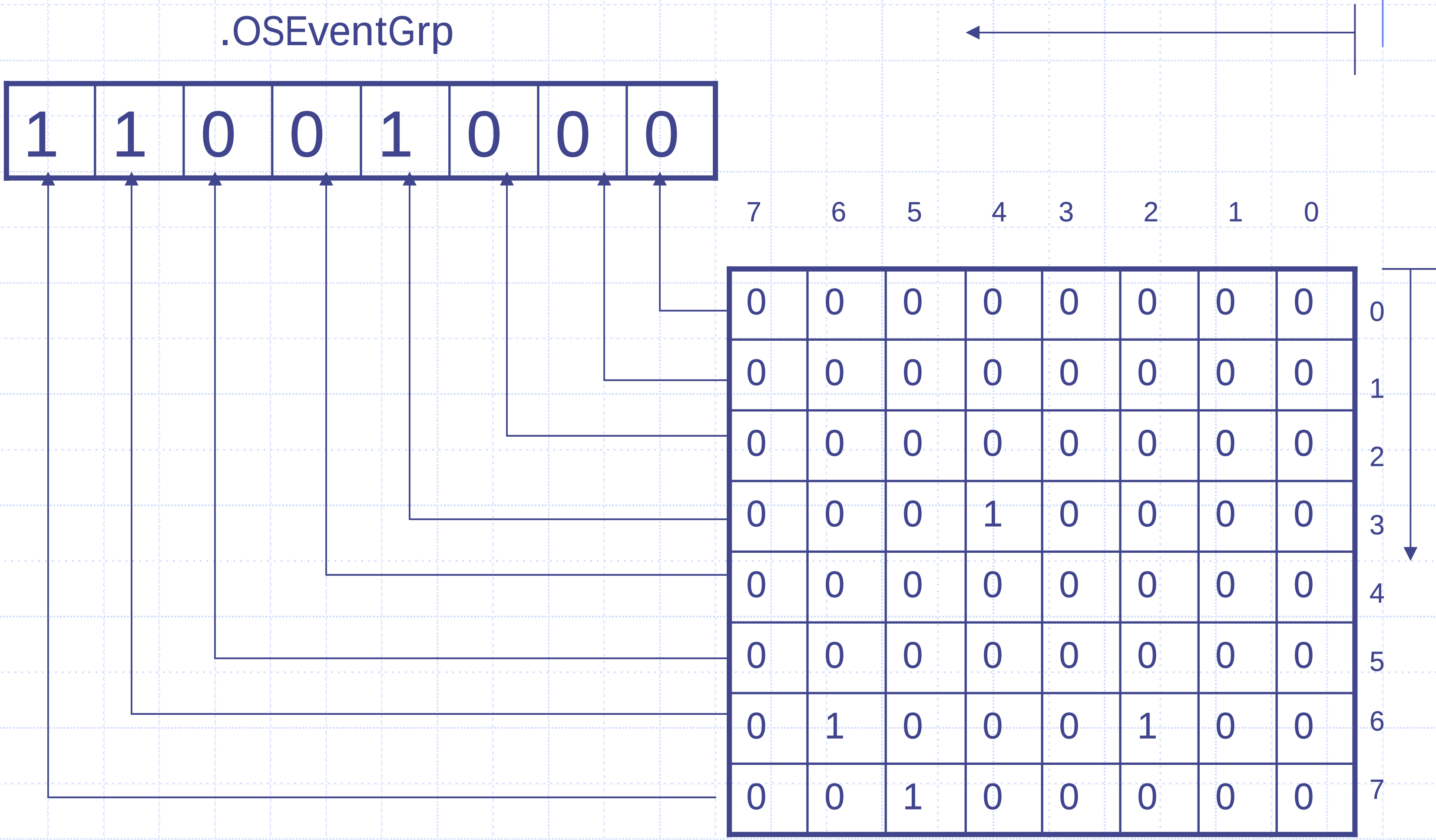
- Listing 6.4 Finding the highest priority task waiting for the event

```
y = OSUnMap[pevent->OSEventGrp];

x = OSUnMap[pevent->OSEventTbl[y]];

prio = ((y <<3) +x;
```

# Listing 6.5 OSUnMapTbl[]

```
INT*U const OSUnMapTbl[]={
0,0,1,0,2,0,1,0,3,0,1,0,2,0,1,0,    /* 0x00 to 0x0F */
4,0,1,0,2,0,1,0,3,0,1,0,2,0,1,0,    /* 0x10 to 0x1F */
5,0,1,0,2,0,1,0,3,0,1,0,2,0,1,0,    /* 0x20 to 0x2F */
4,0,1,0,2,0,1,0,3,0,1,0,2,0,1,0,    /* 0x30 to 0x3F */
6,0,1,0,2,0,1,0,3,0,1,0,2,0,1,0,    /* 0x40 to 0x4F */
4,0,1,0,2,0,1,0,3,0,1,0,2,0,1,0,    /* 0x50 to 0x5F */
5,0,1,0,2,0,1,0,3,0,1,0,2,0,1,0,    /* 0x60 to 0x6F */
4,0,1,0,2,0,1,0,3,0,1,0,2,0,1,0,    /* 0x70 to 0x7F */
7,0,1,0,2,0,1,0,3,0,1,0,2,0,1,0,    /* 0x80 to 0x8F */
4,0,1,0,2,0,1,0,3,0,1,0,2,0,1,0,    /* 0x90 to 0x9F */
5,0,1,0,2,0,1,0,3,0,1,0,2,0,1,0,    /* 0xA0 to 0xAF */
4,0,1,0,2,0,1,0,3,0,1,0,2,0,1,0,    /* 0xB0 to 0xBF */
6,0,1,0,2,0,1,0,3,0,1,0,2,0,1,0,    /* 0xC0 to 0xCF */
4,0,1,0,2,0,1,0,3,0,1,0,2,0,1,0,    /* 0xD0 to 0xDF*/
5,0,1,0,2,0,1,0,3,0,1,0,2,0,1,0,    /* 0xE0 to 0xBF */
4,0,1,0,2,0,1,0,3,0,1,0,2,0,1,0,    /* 0xF0 to 0xFF */};
```

# Figure 6.4 Example of ECB wait list



.OSEventGrp

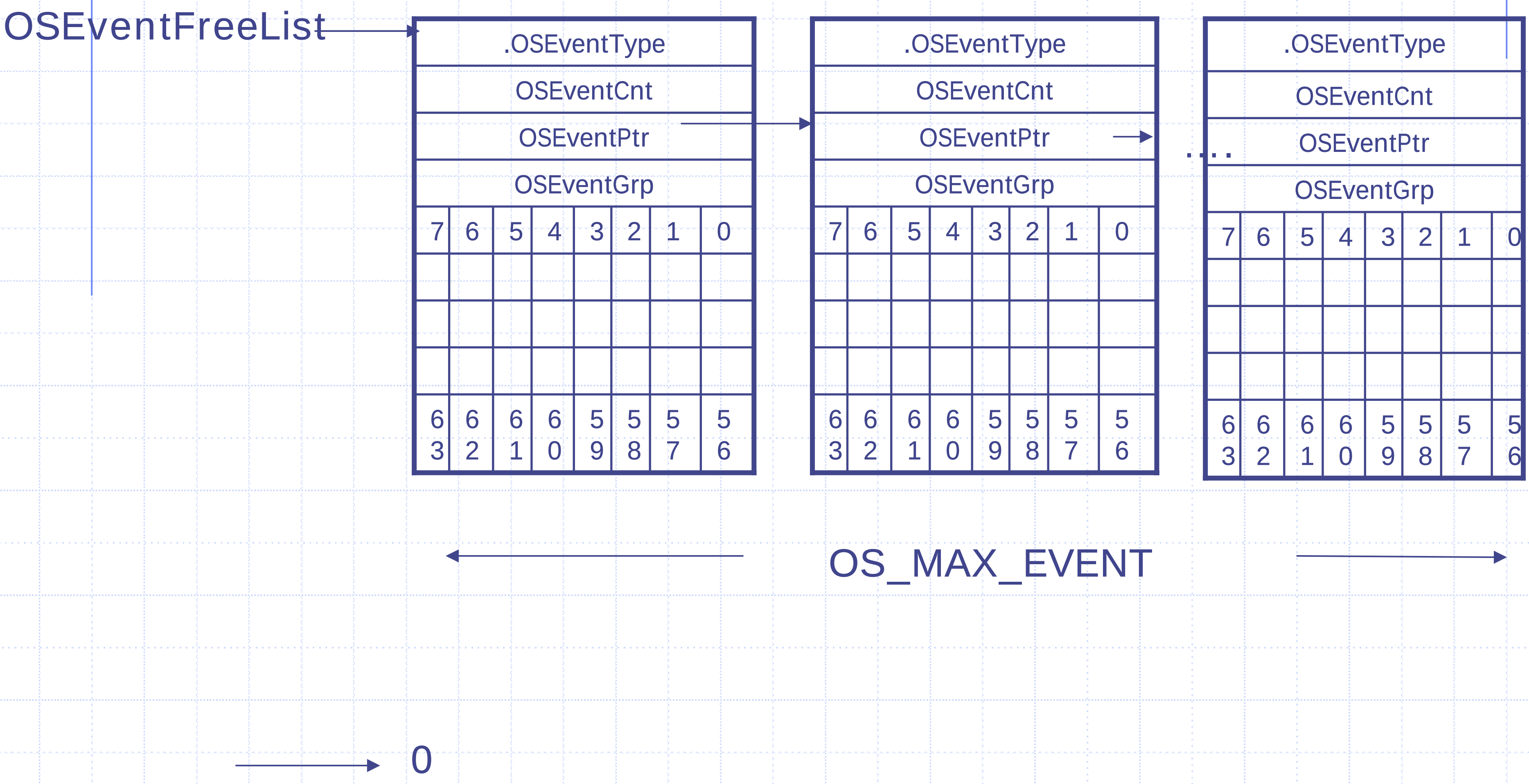
| 1 | 1 | 0 | 0 | 1 | 0 | 0 | 0 |
|---|---|---|---|---|---|---|---|

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 2 |
| 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 3 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 4 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 5 |
| 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 6 |
| 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 7 |

# 6.02 List of Free ECBs

- The number of ECBs to allocate depends on the number of semaphores, mutual exclusion semaphores, mailbox, and queues needed for the application.
- The number of ECBs is established by the #define OS_MAX_EVENTS, which is found in OS_CFG.H
- When OSInit() is called link in a singly link list—the list of free ECBs.
- When a Semaphore, mutex mailbox , or queue is created, an ECB is removed from this list and initialized.
- ECBs can be returned to the list of free ECBs by invoking the OS???Del() functions for semaphore, mutex, mailbox, or queue services.

# Figure 6.5 List of free ECBs



OSEventFreeList

.OSEventType
OSEventCnt
OSEventPtr
OSEventGrp
7 6 5 4 3 2 1 0
63 62 61 60 59 58 57 56

.OSEventType
OSEventCnt
OSEventPtr
OSEventGrp
7 6 5 4 3 2 1 0
63 62 61 60 59 58 57 56

. . . .

.OSEventType
OSEventCnt
OSEventPtr
OSEventGrp
7 6 5 4 3 2 1 0
63 62 61 60 59 58 57 56

OS_MAX_EVENT

0

# Four common operations on ECBs

- Initialize an ECB. OS_EventWaitListInit()
- Make a task ready. OS_EventTaskRdy()
- Make a task wait for an event. OS_EventWait()
- Make a task ready because a timeout occurred while waiting for an event. OS_EventTO()

# 6.04 Initializing an ECB, [OS_EventWaitListInit()]

- OS_EventWaitListInit() is a function called when a semaphore, mutex, message mailbox, or message queue is created
- All that is accomplish by OS_EventWaitListInit() is to indicate that no task is waiting on the ECB.
- OS_EventWaitListInit() is passed a pointer to an event control block, which is assigned when the semaphore, mutex, message mailbox, or message queue is created.

# Listing 6.6 Initializing the wait list

```
Void  OS_EventWaitListInit (OS_EVENT *pevent)
{
    INT8U  *ptbl;

 pevent->OSEventGrp = 0x00;                          /* No task waiting onevent */

    ptbl  = &pevent->OSEventTbl[0];

#If OS_EVENT_TBL_SIZE > 0
    *ptbl++            = 0x00;
#endif

#If OS_EVENT_TBL_SIZE > 1
    *ptbl++            = 0x00;
#endif

#If OS_EVENT_TBL_SIZE > 2
    *ptbl++            = 0x00;
 #endif
```

# Listing 6.6 Initializing the wait list

```
#if OS_EVENT_TBL_SIZE > 3
    *ptbl++                  = 0x00;
#endif

#if OS_EVENT_TBL_SIZE > 4
    *ptbl++                  = 0x00;
#endif

#if OS_EVENT_TBL_SIZE > 5
    *ptbl++                  = 0x00;
#endif

#if OS_EVENT_TBL_SIZE > 6
    *ptbl++                  = 0x00;
#endif

#if OS_EVENT_TBL_SIZE > 7
    *ptbl                    = 0x00;
#endif
   }
```

# 6.05 Making a Task Ready , [OS_EventTasjRdy()]

- This function is called by Post functions for a semaphore, a mutex, a message mailbox, or a message queue when ECB is signaled and the highest priority task wait in on the ECB needs to be made ready on run.
- OS_EventTaskRdy() removes the highest priority task (HPT) from the wait list of the ECB and makes this task ready to run

# Listing 6.7  Making a task ready to run

```
INT8U  OS_EventTaskRdy (OS_EVENT *pevent, void *msg, INT8U msk)
{
    OS_TCB *ptcb;
    INT8U   x;
    INT8U   y;
    INT8U   bitx;
    INT8U   bity;
    INT8U   prio;


    y    = OSUnMapTbl[pevent->OSEventGrp];
    bity = OSMapTbl[y];
    x    = OSUnMapTbl[pevent->OSEventTbl[y]];
    bitx = OSMapTbl[x];
    prio = (INT8U)((y << 3) + x);

    if ((pevent->OSEventTbl[y] &= ~bitx) == 0x00) {
        pevent->OSEventGrp &= ~bity;
    }
```

# Listing 6.7  Making a task ready to run

```
ptcb                   =  OSTCBPrioTbl[prio];

ptcb->OSTCBDly         =  0; ptcb->OSTCBEventPtr  = (OS_EVENT *)0;

#if ((OS_Q_EN > 0) && (OS_MAX_QS > 0)) || (OS_MBOX_EN > 0)
    ptcb->OSTCBMsg         = msg;

#else
    msg                    = msg;
#endif

    ptcb->OSTCBStat     &= ~msk;
if (ptcb->OSTCBStat == OS_STAT_RDY) {

OSRdyGrp        |=  bity;

OSRdyTbl[y]     |=  bitx;
    }
    return (prio);
}
```

# 6.06 Making a Task Wait for an Event

- This function is called by Pend functions for a semaphore, a mutex, a message mailbox, or a message queue when a task must wait on an ECB
- OS_EventTaskWait() removes the current task frome ready list and places it in the wait list of the ECB.

# Listing 6.8  Making a task wait on an ECB

```
void  OS_EventTaskWait (OS_EVENT *pevent)
{
    OSTCBCur->OSTCBEventPtr = pevent;

    if ((OSRdyTbl[OSTCBCur->OSTCBY] &= ~OSTCBCur->OSTCBBitX) == 0x00) {
  OSRdyGrp &= ~OSTCBCur->OSTCBBitY;
    }

   pevent->OSEventTbl[OSTCBCur->OSTCBY] |=
   OSTCBCur->OSTCBBitX;

    pevent->OSEventGrp  |= OSTCBCur->OSTCBBitY;
}
```

# 6.07 Making a Task Ready Because of a Timeout

- This function is called by Pend functions for a semaphore, a mutex, a message mailbox, or a message queue when OSTimeTick() has readied a task to run, which means that the ECB was not signaled within the specified timeout period.

# Listing 6.9  Making a task ready because of a timeout

```
void  OS_EventTO (OS_EVENT *pevent)
{
     if ((pevent->OSEventTbl[OSTCBCur->OSTCBY] &= ~OSTCBCur->OSTCBBitX) ==
    0x00)
    {
          pevent->OSEventGrp &= ~OSTCBCur->OSTCBBitY;
     }
     OSTCBCur->OSTCBStat      = OS_STAT_RDY; OSTCBCur->OSTCBEventPtr =
    (OS_EVENT *)0;
}
```