FreeRTOS
Event Groups

Norman McEntire
norman.mcentire@gmail.com

Textbook Reference

- Mastering the FreeRTOS Real Time Kernel by Richard Barry

  - Chapter 8: Event Groups

Topics

- 8.1 Intro and Scope
- 8.2 Characteristics of Event Groups
- 8.3 Event Management Using Event Groups
- 8.4 Task Synchronization Using an Event Group

Event Groups is Optional

- To include Event Groups in your project, include this file in your project
  - event_groups.c

8.1 Intro and Scope

- Real-time systems must take actions in response to events
    - We have already covered:
        - Semaphores
        - Queues
- The above allow a task to:
    - Wait in the blocked state for a single event to occur
    - Unblock a single task when the event occurs
        - The task unblocked is the highest priority task waiting for the event

How Event Groups are Different from Queues and Semaphores

- Event Groups are useful for synchronizing multiple tasks - they can broadcast
- Event groups allow a task to wait in the blocked state for a combination of one or more events to occur
- Event groups unblock all the tasks that were waiting for the same event (or combination of events)

Unique properties of Event Groups Make them useful

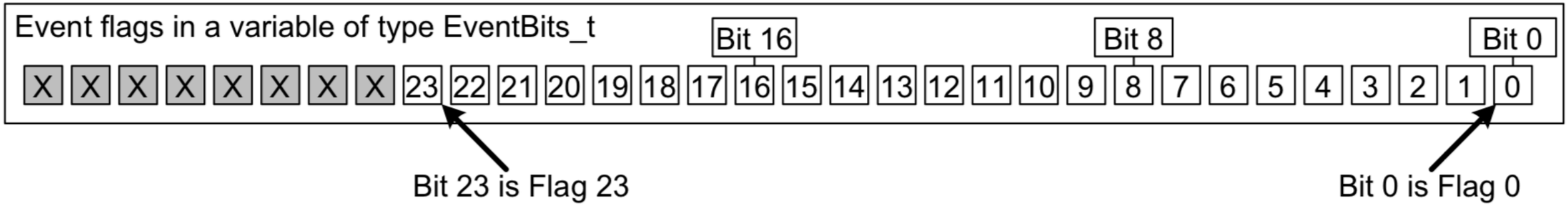
- Event Groups useful for synchronizing multiple tasks…
  - For broadcasting events to more than one task…
    - For allowing a task to wait in the Blocked state for any one of a set of events to occur
      - For allowing a task to wait in the Blocked state for multiple actions to complete

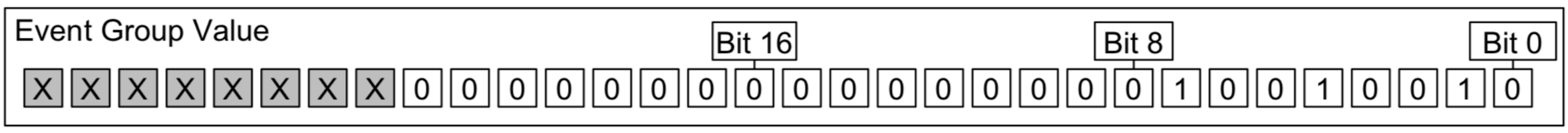
8.2 Characteristics of Event Groups

- Note: Event Flag and Event Bit mean the same thing
- An event flag is a Boolean - 0 or 1
  - An Event Group is a set of these event flags
- **EventBits_t**
  - If configUSE_16_BIT_TICKS is 1, then Event Group contains **8 usable bits**
  - If configUSE_16_BIT_TICKS is 0, then Event Group is **24 usable bits**

Event Flags and Event Bits Mean the Same Thing

configUSE_16_BIT_TICKS is 0 (24-bits for Event Group)





0x92

It is up to you to define the meanings of the Event Bits

- Example
  - Bit 0 - “message has been received from network”
  - Bit 1 - “message ready to be sent to network”
  - Bit 2 - “Cancel the current network connection”

Access by Multiple Tasks

- Event Groups are objects in their own right
  - Can be accessed by other Tasks
  - Can be accessed by ISRs
- Any number of Tasks and ISRs can set bits in the Event Group

Example of Using Event Groups: TCP/IP

- A TCP socket must respond to different events
  - Accept Events
  - Bind Events
  - Read Events
  - Close Events
- The state of a FreeRTOS+TCP socket is held in a structure called FreeRTOS_Socket_t
  - Inside the structure is an Event Group
- Advantage: The TCP socket code can block on the Event Group - waiting for one or more of the events to occur

8.3 Event Management Using Event Groups

- We cover these APIs in the following slides
  - xEventGroupCreate()
    xEventGroupCreateStatic()
  - xEventGroupSetBits()
    xEventGroupSetBitsFromISR()
  - xEventGroupWaitBits()

xEventGroupCreate()

- EventGroupHandle_t xEventGroupCreate(void)
- To allocate dynamically at run-time
- If returns NULL, then could not be created

xEventGroupCreateStatic()

- EventGroupHandle_t xEventGroupCreateStatic()
- To allocate statically at compile time
- If returns NULL, then could not be created

xEventGroupSetBits()

- EventBits_t
  xEventGroupSetBits(
      EventGroupHandle_t xEventGroup,
      const EventBits_t uxBitsToSet)

- Notes
  - uxBitsToSet will do an bitwise OR of the bits
  - Returned value is the bits that are set

xEventGroupSetBitsFromISR ()

- BaseType_t xEventGroupSetBitsFromISR(
  EventGroupHandle_t xEventGroup,
  const EventBits_t uxBitsToSet,
  BaseType_t *pxHigherPriorityTaskWoken)

- Notes
  - uxBitsToSet will do an bitwise OR of the bits
  - Returned value either pdPASS or pdFALSE

xEventGroupWaitBits

- EventBits_t xEventGroupWaitBits(
  const EventGroupHandle_t xEventGroup,
  const EventBits_t uxBitsToWaitFor,
  const BaseType_t xClearOnExit,
  const BaseType_t xWaitForAllBits,
  TickType_t xTicksToWait)

xEventGroupWaitBits Example 1

- Existing Event Group Value: 0000
- uxBitsToWaitFor: 0101
- xWaitForAllBits: pdFALSE
- Results
  - Calling task will enter the Blocked state
  - Will leave the blocked state when either bit 0 or bit 2 is set

xEventGroupWaitBits Example 2

- Existing Event Group Value: 0100
- uxBitsToWaitFor: 0101
- xWaitForAllBits: pdTRUE
- Results
  - Calling task will enter the Blocked state
  - Will leave the blocked state when BOTH bit 0 AND bit 2 is set

xEventGroupWaitBits Example 3

- Existing Event Group Value: 0100
- uxBitsToWaitFor: 0110
- xWaitForAllBits: pdFALSE
- Results
  - Calling task will NOT enter the Blocked state because at least one of the bits (bit 2) is set

xEventGroupWaitBits Example 4

- Existing Event Group Value: 0100
- uxBitsToWaitFor: 0110
- xWaitForAllBits: pdTRUE
- Results
  - Calling task will enter the Blocked state
  - The task will leave the Blocked state when both bits 1 and bits 2 are set

Code Demo - Overview
Experimenting with Event Groups

- What this code demo will show
  - How to create an event group
  - How to set bits in an event group from an ISR
  - How to set bits in an event group from a task
  - How to block on an event group

Code Demo - Part 1
Experimenting with Event Groups

- #define mainFIRST_TASK_BIT (1UL << 0UL)
- #define mainSECOND_TASK_BIT (1UL << 1UL)
- #define mainISR_BIT (1UL << 2UL)

Code Demo - Part 2
Experimenting with Event Groups

- static void **vEventBitSettingTask**(void *pvPararms) {

```
    const TickType_t xDelay200ms =
      pdMS_TO_TICKS( 200UL );
    const TickType xDontBlock = 0;
    for ( ;; ) {
        vTaskDelay( xDelay200ms );
        vPrintString( “Bit setting task - setting bit 0\r\n”);
        xEventGroupSetBits( xEventGroup, mainFIRST_TASK_BIT);
        vTaskDelay( xDelay200ms );
        vPrintString(“Bit setting task - setting bit 1\r\n”);
        xEventGroupSetBits( xEventGroup,
mainSECOND_TASK_BIT);
    }
}
```

Code Demo - Part 3
Experimenting with Event Groups

- static uint32_t **ulEventBitSettingISR**(void) {
  static const char *pcString = “ISR Bit Setting\r\n”);
  BaseType_t xHigherPriorityTaskWoken = pdFALSE;

// Use pending function call since cannot print from ISR
xTimerPendFunctionCallFromISR(vPrintStringFromDaemonTask,
(void *) pcString, 0, &xHigherPriorityTaskWoken);

xEventGroupSetBitsFromISR(xEventGroup, mainISR_BIT,
&xHigherPriorityTaskWoken);

portYIELD_FROM_ISR(xHigherPriorityTaskWoken);
  }
}

Code Demo - Part 4
Experimenting with Event Groups

- static void **vEventBitReadingTask**(void *pvParams) {
  EventBits_t xEventGroupValue;
  const EventBits_t xBitsToWaitFor = (
   mainFIRST_TASK_BIT |
   mainSECOND_TASK_BIT |
   mainISR_BIT)
   for (;;) {
      xEventGroupValue = xEventGroupWaitBits(
      xEventGroup, xBitsToWaitFor,
      pdTRUE, //Clear bits on exit
      pdFALSE, // Do not wait for all bits
      postMAX_DELAY);
    if ((xEventGroupValue & mainFIRST_TASK_BIT) != 0) {
      vPrintString(“Bit 0 was set\r\n”);
    }
    If ((xEventGroupValue & mainSECOND_TASK_BIT) != 0) {
       vPrintString(“Bit 1 was set\r\n’);
    }
    If ((xEventGroupValue & mainISR_BIT) != 0) {
       vPrintString(“Bit 2 was set\r\n”);
    }
  }
}

Code Demo - Part 5
Experimenting with Event Groups

- int main(void) {
  xEventGroup = xEventGroupCreate();
  xTaskCreate(vEventBitSettingTask,”Bit Setter”, 1000, NULL, 1, NULL);
  xTaskCreate(vEventBitReadingTask,”Bit Reader”, 1000, NULL, 2, NULL);
  xTaskCreate(vInterruptGenerator,”Int Gen”, 1000, NULL, 3, NULL);
  vPortSetInterruptHandler(mainINTERRUPT_NUMBER, ulEventBitSettingISR)
  vTaskStartScheduler();
  for(;;); //Should never get here!
  return 0;
}

8.4 Task Synchronization Using an Event Group - Part 1

- Sometimes two or more tasks must synchronize with each other
- Example #1
  - Task A receives and event and then delegates work to Tasks B, C, and D
  - Task A cannot continue until Tasks B, C, and D complete their work

8.4 Task Synchronization Using an Event Group - Part 2

- Example #2
    - FreeRTOS+TCP socket shared between RxTask and TxTask
    - It is not safe to close the socket until BOTH of the tasks are done with the socket
        - Hence, if either task wants to close the socket, it must first notify the other task, then wait for other task to stop using the socket before proceeding

Code Demo - Part 1

```
• void SocketTxTask(void pvParams) {
    xSocket_t xSocket;
    unit32_t ulTxCount = 0UL;
    for ( ;; ) {
        xSocket = FreeRTOS_socket(…);
        FreeRTOS_connect(…);
        xQueueSend(xSocketPassingQueue, &xSocket, portMAX_DELAY);
        for ( ulTxCount = 0; ulTxCount < 1000; ulTxCount++) {
            if (FreeRTOS_send(xSocket, … ) < 0) {
                break; //unexpected error
            }
        }
    }
    TxTaskWantsToCloseSocket();
    xEventGroupSync(…);
    FreeRTOS_shutdown(xSocket, …);
    WaitForSocketToDisconnect();
    FreeRTOS_closesocket(xSocket);
}
```

Code Demo - Part 2

- void **SocketRxTask**(void pvParams) {
  xSocket_t xSocket;
  for (;;) {
  xQueueReceive(xSocketPassingQueue), xSocket, portMAX_DELAY);
  while (TxTaskWantsToCloseSocket() == pdFALSE) {
  FreeRTOS_recv(xSocket, …);
  ProcessReceiveData();
  }
  xEventGroupSync(…)
  }

xEventGroupSync()

- EventBits_t xEventGroupSync(EventGroupHandle_t xEventGroup, const EventBits_t uxBitsToSet, const EventBits_t uxBitsToWaitFor, TickType_t xTicksToWait)
-

Code Demo - Part 1

- #define mainFIRST_TASK_BIT (1UL << OUL)
- #define mainSECOND_TASK_BIT (1UL << 1UL)
- #define mainTHIRD_TASK_BIT (1UL << 2UL)
- EventGroupHandle_t xEventGroup;

Code Demo - Part 2

- static void **vSyncingTask**(void *pvParams) {
  const TickType_t xMaxDelay = pdMS_TO_TICKS(4000UL);
  const TickType_t xMinDelay = pdMS_TO_TICKS(2000UL);
  TickType_t xDelayTime;
  EventBits_t uxThisTasksSyncBit;
  const EventBits_t uxAllSyncBits(mainFIRST_TASK_BIT | mainSECOND_TASK_BIT | mainTHIRD_TASK_BIT);
  // Three tasks are created - each has different bit
  uxThisTasksSyncBit = (EventBits_t) pvParams);
  for (;;) {
    xDelayTime = (rand() % xMaxDelay) + xMinDelay;
    vTaskDelay(xDelayTime);
    vPrintTwoStrings(pcTaskGetTaskName(NULL),"reached sync point");
    xEventGroupSync(xEventGroup, uxThisTasksSyncBit, uxAllSyncBits, portMAX_DELAY)
    vPrintTwoStrings(pcTaskGetTaskName(NULL), "exited sync point");
  }
}

Code Demo - Part 3

- int main(void) {
    xEventGroup = xEventGroupCreate();

    xTaskCreate(vSyncingTask, “Task 1”, 1000, mainFIRST_TASK_BIT, 1, NULL);
    xTaskCreate(vSyncingTask, “Task 2”, 1000, mainSECOND_TASK_BIT, 1, NULL);
    xTaskCreate(vSynchingTask, “Task 3”, 1000, mainTHIRD_TASK_BIT, 1, NULL);

    vTaskStartScheduler();

    for (;;); //Should never get here
    return 0;
}

Summary

- 8.1 Intro and Scope
- 8.2 Characteristics of Event Groups
- 8.3 Event Management Using Event Groups
- 8.4 Task Synchronization Using an Event Group