HighIntegritySystems

WITTENSTEIN

Event Flags and Event Groups

Event Flags and Event Groups provide an efficient and flexible method of handling Task synchronization and event management within a Real Time Operating System environment.

Event Flags

An Event Flag is a Boolean value that represents the status of a single event within an Event Group. For example, an Event Flag could be defined to mean “A message has been received and is ready for processing” when it is set to 1, and “there are no messages waiting to be processed” when it is set to 0. An Event Flag is a single bit, like a Binary Semaphore.

The single Event Flag can be used to communicate the occurrence of one event to a task. A task can be blocked (not consuming any CPU time) waiting for a single Event Flag.

Event Groups

An Event Group is a special type of variable constructed from Event Flags. For example, the Event Flag at position 0 could be defined as “A message has been received and is ready for processing” when it is set to 1, and the Event Flag at position 5 could be defined to mean “The application has queued a message that is ready to be sent to a network” when it is set. Similarly, the other Flags within the Group can be defined, or left undefined.

Event Group API

An RTOS will provide an API for the manipulation of the Event Group and the individual bits. The API contains functions to enable the programmer to create and remove Event Groups, to reset individual Event Flags to zero, and to clear all the Event Flags in a group.

There is a function that sets individual Event Flags in the Event Group to represent events that have occurred. It does this by updating the existing value of the Event Group by ORing the value passed by the function with the current state of the Event Group bits.

Setting Event Flags from an Interrupt Service Routine, or ISR, is performed by sending messages from the ISR to a kernel Task, which can reduce the need for extended critical sections within the code. Any Task or ISR that knows of the existence of an Event Group can set Event Flags in it, and multiple Tasks can set Event Flags and read Event Flags from the same Event Group.

If a Task has to wait on more than one Event Flag becoming set before unblocking, then the test can be based on either a bitwise OR or a bitwise AND test on the value of the Event Group.

Generally the API would contain the following function types:

- xEventGroupCreate
- xEventGroupDelete
- xEventGroupGetBits
- xEventGroupSetBits
- xEventGroupClearBits
- eEventGroupWaitBits

Uses of Event Groups

The Event Group is important because it can be used for a number of different purposes:

- Tasks can be blocked (not consuming any CPU time) waiting for a single Event Flag, or on combinations of Event Flags within an Event Group.
- An Event Group can replace a number of separate Binary Semaphores, reducing the amount of processing and memory required. This allows Tasks to wait on a combination of events, rather than a single event. For example, the Task could unblock when a single, a combination, or all Event Flags are set within the Event Group.
- An Event Group can be used to synchronise Tasks by making the Tasks wait in the Blocked state until all the other Tasks taking part in the synchronisation have also reached their synchronisation point. This can be signalled by the setting of individual bits.

WITTENSTEIN high integrity systems
Americas: +1 408 625 4712
ROTW: +44 1275 395 600

email: sales@highintegritysystems.com
web: www.highintegritysystems.com

More Efficient Task Communication

For Task to Task communication and synchronisation Event Flags may provide a more efficient mechanism than a Binary Semaphore; however, this is not always the case:

- If there is a one-to-one relationship, for example, if only one Task blocks waiting for the Semaphore, then Task Notification should provide the most efficient solution;
- If many Tasks are blocked waiting for a single Semaphore, then an Event Flag should provide the most efficient solution;
- Semaphores are generally more efficient when the interaction is between ISR and Tasks.

Event Group data type

Event Groups are variables of a specific data type, with size depending on the implementation. Figure 1 shows a 24-bit Event Group which has three bits, or Event Flags, in use, and one bit (bit 2) is set. An Event Group has a value, in this case binary 100 or octal or hex 4.

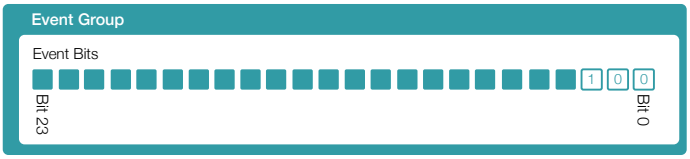

Figure 1. *An Event Group with three bits*

In Figure 2, the first eight bits are used, and bits 2,4 and 6 are set, giving the binary value 01010100 (octal 124 , hex 54).

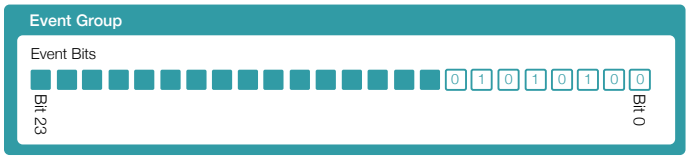

Figure 2. *A Event Group with eight bits*

What to lookout for when using Event Groups

Enabling a Task to wait on multiple events can introduce indeterminacy into the operations, since there is no control over the sequence in which events can occur, and Tasks using the same Event Group can set the bits in any time order.

Some RTOS have inbuilt precautions to help to guard against this, but the developer must be aware of the dangers. These dangers can include “race conditions”- when the program has to carry out actions in the right order to perform them correctly. However, this cannot be enforced, and it cannot be known how many Tasks are blocked or how many event bits will need to be tested when an event bit is set.

Event Flags and SAFERTOS®

SAFE**RTOS**® implements Event Groups, but the code is designed to ensure that use of Event Groups cannot lead to non-deterministic behaviour. This includes built-in code to ensure that setting, testing and clearing of bits appears to calling code to be atomic.

Like FreeRTOS, SAFE**RTOS** also implements the Task notification data type, which can be used as an alternative to implement Event Flags in certain cases with a saving in RAM and an increase in speed.

When using SAFE**RTOS** the top 8 bits are reserved and cannot be used.

Figure 3. *An Event Group within SAFE**RTOS***

WITTENSTEIN high **integrity** systems

Worldwide Sales and Support
Americas: +1 408 625 4712
ROTW: +44 1275 395 600
Email: sales@high**integrity**systems.com