**January 22, 1988 Real Time Executive Interface Definition**

**3.2.3 Q\_URGENT**

**NAME**

**q\_urgent \_ “Place an Urgent Message at the Head of a Message Queue"**

**SYNOPSIS**

**#include <message.h>
uint q\_urgent ( qid, buffer )**

**uint qid; /\* message queue id returned from q\_create or q\_indent \*/
long (\*buffer) [4]; /\* pointer to message buffer \*/**

**DESCRIPTION**

**The *q\_urgent* directive sends a message to the queue identified by the *qid*. This call is the same as the *q\_send* call, except, if there are other messages at the queue, this message is put at the head of the queue.**

**If a task is already waiting at the queue, the message is copied to that task’s indicated receiving buffer. The task is then made ready. If there is no task waiting; the message is copied to a system which is then placed at the head c ' the message queue.**

**Once sent, the task’s message area may be reused immediately. A message is fixed length, 16bytes.**

**The message queue may exist on the local processor or any remote processor in a multiprocessor configuration, as long as the queue was created with the GLOBAL flags value set (see *q\_create*),**

**RETURN VALUE**

**If the *q\_urgcnt* directive successfully sent a message, then 0 is returned.
If the call was not successful, an error code is returned.**

**ERROR CONDITIONS**

**Message *qid* in invalid.
Out of system message buffers.
Message queue at maximum count. -
ISR cannot reference remote node.**

**NOTES**

**Can be called from within an ISR, except when the queue was not created from the local node.**

**May cause a preempt if a task waiting at the message queue has a higher priority than the running task, and the preempt mode is in effect. A preempt will not occur if a task waiting exists on**

**Real Time Executive Interface Definition January 22, 1988**

**a remote processor in a multiprocessor configuration.**

**January 22, 1988 Real Time Executive Interface Definition**

**8.2.10 Q\_BROADCAST**

**NAME**

**q\_broadcast -- “Broadcast N Identical Messages to a Message Queue”**

**SYNOPSIS**

**#include <message.h>
uint q\_broadcast ( qid, buffer, &count )**

**uint qid; /\* message queue id returned from q\_create or q\_ident \*/
long (\*buffer) [4]; /\* pointer to message buffer \*/
uint count; /\* number of tasks made ready - returned by this call \*/**

**DESCRIPTION**

**The *q\_broadcast* directive sends as many messages as necessary to make ready all tasks waiting on the queue identified by the *qid*. The number of tasks readied is returned to the caller in *count*.**

**Once sent, the task’s message buffer may be reused immediately.**

**The message queue may exist on the local processor or any remote processor in a multiprocessor configuration, as long as the queue was created with the GLOBAL flags value set (see *q\_create*)**

**RETURN VALUE**

**If the *q\_broadcast* directive succeeds, the *count* is filled in with the number of tasks readied, and O is returned.**

**II the call was not successful, an error code is returned.**

**ERROR CONDITIONS**

**Message *qid* is invalid.**

**ISR cannot reference remote node.**

**NOTES**

**Can be called from within an ISR, except when the queue was not created from the local node.**

**May came a preempt if a task waiting at the message queue has a higher priority than the running tank, and the preempt mode is in effect. A preempt will not occur if a task waiting exists on a remote processor in a multiprocessor configuration.**

**Real Time Executive Interface Definition**

**3.2.11 Q\_RECEIVE**

**NAME**

**q\_receive -- “Receive a Message from a Message Queue”**

**SYNOPSIS**

**#include <message.h>
uint q\_receive ( qid, buffer, flags, timeout )**

**uint qid; /\* message queue id returned from q\_create or q\_ident \*/
long (\*buffer) [4] /\* pointer to message buffer \*/
uint flags; /\* options \*/
uint timeout /\* number of ticks to wait \*/
 /\* 0 indicates wait forever \*/**

**DESCRIPTION**

**The *q\_receive* directive allows a task to request a message from the message queue identified by qid.**

**If there is a message at the message queue, it is copied into the requester’s buffer.**

**If there is no message at the message queue, then the NOWAIT flag determines what to do. If the NOWAIT flags value is set, the task returns immediately with -1 and the no message at queue error number. If the NOWAIT flags value is clear, the task is put on a wait list for the message queue, according the queue's attributes [FIFO or priority).**

**The *timeout* field is used to determine how long to wait. A zero in the *timeout* field indicates no timeout -- wait forever. A non-zero entry in the timeout field indicates that the task will run after that many ticks, if a message has not been received, or before if a message is received.**

**When *q\_receive* is called from an ISR, the no wait option is forced by the executive. Thus there will be no waiting for a message. An error will be returned if there is no message.**

**The message queue may exist on the local processor or any remote processor in a multiprocessor configuration, as long as the queue was created with the GLOBAL flags value set [see *q\_create*].**

**RETURN VALUE**

**If the *q\_receive* directive succeeds, then 0 is returned.**

**If the call was not successful, an error code is returned.**

**January 22, 1988 Real Time Executive Interface Definition**

**ERROR CONDITIONS**

**Message *qid* is invalid.**

**No message at queue ( if no wait is selected ).**

**Message queue deleted.**

**Timed out with no message ( if wait and timeout is selected )**

**ISR cannot reference remote node.**

**NOTES**

**Can he called from within an ISR, except when the queue was not created from the local node. The executive will force the options to no wait.**

**The requesting task may be blocked if there is no message available, and the wait option is selected.**

**Real Time Executive Interface Definition January 22, 1988**

**3.2.12 EV\_SEND**

**NAME**

**ev\_send -- “Send Event to a Task”**

**SYNOPSIS**

**uint ev\_send ( tid, event )**

**uint tid; /\* task id as returned by t\_create or t\_ident \*/
uint event; /\* event set \*/**

**DESCRIPTION**

**The *ev\_send* directive sends an event to a task. The *event* field describes the set of events the task wishes to send. Thirty-two events are available. Sixteen are available as *system* events and sixteen are available as *user* events.**

**The task identified by the *tid* may exist on the local processor or any remote processor in a multiprocessor configuration, as long as the task was created with the GLOBAL flags value set (see *t\_create*).**

**Events sent to tasks not waiting for an event are left pending.**

**RETURN VALUE**

**If the *ev\_send* directive succeeds, then 0 is returned.**

**If the call was not successful, an error code is returned.**

**ERROR CONDITIONS**

**Invalid *tid*.**

**ISR cannot reference remote node.**

**NOTES**

**Can be called from within an ISR, except when the task was not created from the local node.**

**May cause a preempt if the task waiting for the event has a higher priority than the running task, and the preempt mode is in effect. A preempt will not occur if the task waiting exists on a remote processor in a multiprocessor configuration.**

**January 22, 1988 Real Time Executive Interface Definition**

**3.2.13 EV\_RECEIVE**

**NAME**

**ev\_receive -- “Receive Event”**

**SYNOPSIS**

**uint ev\_receive ( eventin, flags, timeout, &eventout )**

**uint eventin; /\* input event condition \*/
uint flags; /\* options \*/
uint timeout; /\* number of ticks to wait \*/
 /\* 0 indicates wait forever \*/
uint eventout; /\* output events - returned by this call \*/**

**The flags values are:**

**NOWAIT set if the task is to return immediately
 clear if the task is to wait for event condition
ANY set return when any one
 of the indicated events has occurred
 clear return when all
 of the indicated events have occurred**

**DESCRIPTION**

**The *ev\_receive* directive allows a task to receive an event condition. The event condition to receive is a set of events specified in the *euentin* field.**

**The task may elect to wait for the event condition, or return immediately by setting the NOWAIT value in the flags field. The task may elect to receive all of the events, or receive any one of them by setting the ANY value in the flags field.**

**When pending events satisfy the event condition, the events are cleared and the task will remain running. Otherwise, if the task elects to wait, the task will become blocked. The task will be made ready to run when the event condition is satisfied by new events, or the timeout condition is met.**

**When pending events do not satisfy the event condition, and the task elect: not to wait, the task returns immediately with -1 and the no event available error number.**

**If the eventin field is 0, *ev\_receive* will return the pending events, but the events will remain pending.**

**The *timeout* field is used to determine how long to wait. A zero in the *timeout* field indicates no timeout -- wait forever. A non-zero entry in the *timeout* field indicates that the task will run after that many ticks, if the event condition is not satisfied, or before if the event condition is satisfied. .**

**Real Time Executive Interface Definition January 22, 1988**

**RETURN VALUE**

**If the *ev\_receive* directive succeeds, *eventout* is filled in with the output events, and 0 is returned.**

**If the call was not successful, an error code is returned.**

**ERROR CONDITIONS**

**Event not satisfied ( if no wait is selected ).**

**Timed out with no event ( if wait and timeout is selected )**

**NOTES**

**Cannot be called from within an ESR.**

**The requesting task may be blocked if the event condition is not satisfied, and the wait option is selected.**

**January 22, 1968 Real Time Executive Interface Definition**

**3.2.14 AS\_CATCH**

**NAME**

**as\_catch -- “Catch Signals"**

**SYNOPSIS**

**uint as\_catch ( asraddr, mode )**

**ptf asraddr; /\* address of Asynchronous Signal Routine (asr) \*/
 /\* 0 indicates asr is invalid
uint mode; /\* mode value for asr \*/**

**The *mode* value is defined as follows:**

**NOPREEMPT set to disable preempting
 clear to enable preempting
TSLICE set to enable timeslicing
 clear to disable timeslicing
DISASR set to disable asr processing
 clear to enable asr processing
SUPV set to execute in supervisor mode
 clear to execute in user mode
LEVEL interrupt level when SUPV is set**

**DESCRIPTION**

**The *as\_catch* directive allows a task to specify what action to take when catching signals.**

**The asr address is established when *as\_catch* is called with a non-zero address in the *asraddr* field. Zero is not a valid asr address. The asr is invalidated when *as\_catch* is called with the *asraddr* field equal zero. Asynchronous signal processing will be discontinued until re-enabled with a valid asr address in another *as\_catch* call.**

**When a signal is caught, the task is not unblocked. Signals are latched until the task becomes the running task, at which time the task is dispatched to its asr. The task will execute the asr according to the values specified in the *mode* field. The signal condition will be passed to the task, along with the task’s current PC and mode, on the task’s stack in a signal stack frame. The signal condition contains all of the signals which have been received since the last time the task was executing.**

**The asr is responsible for saving and restoring all registers it uses.**

**The *as\_return* directive must be executed to return the task to its previous dispatch address.**

**Only one asr per task is allowed.**

**RETURN VALUE**

**The *as\_catch* directive always succeeds, and returns 0.**

**Real Time Executive Interface Definition January 22, 1988**

**ERROR CONDITIONS**

**None.**

**NOTES**

**Cannot be called from within an ISR.**

**Will not cause a preempt.**