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.

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 *eventin* 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 elects 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.

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 ISR.

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

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 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.

ERROR CONDITIONS

None.

NOTES

Cannot be called from within an ISR.

Will not cause a preempt.