3.4.9 TM_TICK

NAME

tm_tick -- "Announce Tick"

SYNOPSIS

uint tm_tick ( )

DESCRIPTION

This call is used to inform the executive that a system clock tick has occurred. This information is used by the time manager to maintain correct calendar time, execute timeslicing, and decrement ticks from tasks which are currently being delayed or timing out. When a timeslice or timeout expires, the task is made ready.

RETURN VALUE

Tm_tick **always succeeds and returns 0.**

ERROR CONDITIONS

None.

NOTES

Can be called from within an ISR.

3.5 Interrupt Handling

Fast interrupt response and the ability to preempt from an Interrupt Service Routine (ISR) are important features of a real time executive.

In order to provide the fastest possible interrupt service mechanism, the executive will allow tasks and ISRs to directly claim interrupt vectors by writing directly to the vector table.

An ISR usually communicates with tasks within the system using RTEID directives. The directives which are callable from ISRs are identified in the **NOTES** section of each directive. Directives called from an ISR will always return immediately to the ISR, without going though the normal dispatch cycle. The postponed dispatch is required to complete the ISR before any tasks are dispatched.

The *i_return* directive provides the real-time exit mechanism for ISRs. Since an ISR can make a task other than the running task ready to run, i.e. by sending a message from the ISR, it becomes extremely important NOT to exit the ISR with the RTE instruction. This would return control to the running task at the time of the interrupt, which may not be the highest priority task ready to run. To ensure the highest priority task runs, all ISRs must exit using the *i_return* directive, which may cause the running task to be preempted.

The directives provided by the interrupt manager are:

Directive	Function
i_return	Return from Interrupt

3.5.1 I_RETURN

NAME

i_return -- "Return from Interrupt"

SYNOPSIS

void i_return ()

DESCRIPTION

The *i_return* directive will allow the executive to return control to the highest priority task in the system following the interrupt processing. The interrupt routine may have caused a task of higher priority than the task running at the time of interrupt, to become ready.

RETURN VALUE

None.

ERROR CONDITIONS

None.

NOTES

Can only be called from an ISR.

3.6 Fatal Errors

Occasionally, the executive, application or system software will detect an unrecoverable error condition. Such a condition is called a *fatal error* and normally halts execution on the local node. Such errors include checksum errors, not enough memory, etc.

The executive will provide a fatal error handler which is responsible for processing fatal errors. The exact manner in which fatal errors are processed is implementation dependent. For example, the executive may simply **STOP**, or it may pass control to a debugger or other user provided fatal error handling routine.

There are three sources for fatal errors:

1. the executive
2. system code
3. user application code

When the executive detects a fatal error, control is automatically passed to the fatal error handler. When system code or user application code detects a fatal error, the *k_fatal* directive should be used to pass control to the fatal error handler. The error code passed to the fatal error handler describes the type of fatal error.

Fatal errors only halt execution on the local node. Remote nodes are not directly affected.

The directive provided to report fatal errors is:

Directive	Function
k_fatal	Fatal Error

3.6.1 K_FATAL

NAME

k_fatal -- "Fatal Error"

SYNOPSIS

void k_fatal (errcode)

uint errcode; /* type of error to be reported */

DESCRIPTION

The *k_fatal* directive will allow the executive to halt execution of the system in a manner as described by the *errcode*. This directive does not return to the caller.

RETURN VALUE

None.

ERROR CONDITIONS

None.

NOTES

Can be called from within an ISR.