

TIMER_I960HX

The TIMER_I960HX module is a driver for the on-chip I960Hx counter/timer.

Process Information

Prototype Name	timer
Process Priority	driver-level
Process Name	should be “timer” if the standard timer interface library is used.

Module Options

TIMER_DEBUG	If this symbol is defined, the timer routine allows the default timer rate to be overridden, and the interrupt handler prints a message after a configurable number of intervals have elapsed. This is only used during initial development to calibrate a system.
-------------	--

Target File Definitions

TIMER_INT	The interrupt pin number of timer 0.
TIMER_INTVEC	The interrupt vector number of timer 0.
TIMER_TICKS2SEC	The number of timer ticks in a second, this is usually set to 1000.
TIMER_TCR0	The machine address for the Timer Count Register for timer 0.
TIMER_TMR0	The machine address for the Timer Mode Register for timer 0.
TIMER_TRR0	The machine address of the Timer Reload Register for timer 0.

Process Operation

The initialisation routine installs the interrupt handler and programs the interval for continuous running and auto-reloading with TIMER_TICKS2SECS interrupts per second. The module has a queue handler and a dummy main process (to define the “timer” queue name). and accepts the message defined in the *Timer* messageset:

TIMER messages with zero or negative elapsed timer are returned immediately by the queue handler, otherwise they are linked in to the timer queue in elapsed time order, with each interval being the ‘delta-ticks’ from the previous message. In the interrupt handler, one tick is subtracted from the message at the head of the queue on each timer interrupt, and the message(s) are returned to the sender when the counter reaches zero. Note that more than one message may be replied to within a single tick if the actual time is the same as a preceding message.

Shared Library Macros and Routines

The *sleep* and *timeout* routines can only be used in a ‘process context’ as they rely on inter-process messages.

sleep

```
void sleep(  
    int hz)
```

The *sleep* routine suspends the current process for *hz* timer ticks, by sending a timer message and waiting for the reply.

timeout

```
ptr timeout(  
    (void (*func))(caddr_t),  
    caddr_t se,  
    int hz)
```

The *timeout* routine requests that the routine *func* be called with argument *se* after at least *hz* clock ticks have elapsed. The routine places a *TIMEOUT* message on the timer queue. The returned value is an opaque ‘token’ that can be passed to the *untimeout* routine to cancel the timeout.

timer_tag

```
void timer_tag(  
    long *tick,  
    long *subtick)
```

The *timer_tag* routine sets the *tick* value to the current tick counter (the number of timer ticks since the system started) and the *subtick* value to the elapsed time within the current tick, normalised to 0..1000.

timer_tmhandler

```
void timer_tmhandler(  
    ROME_MESSAGE *mptr)
```

The *timer_tmhandler* routine **must** be used to handle replies to *TIMEOUT* messages if the messages were generated using the *timeout* routine. It calls the registered callback routine and frees the context information held internally within the library. The routine is suitable for including in a list of handlers passed to the *rome_generic_handler* routine, for *ROME_R_TIMEOUT* messages.

untimeout

```
void untimeout(  
    ptr token)
```

The *untimeout* routine prevents the routine associated with the call to *timeout* represented by the *token* from being called. This routine does not remove the message from the timer queue, so the reply must still be handled, but the routine will not be called.

Debug Support

The *timer_show_timerq* routine is callable from within the debugger, and will print the contents of the timer queue, listing the source process of each *TIMEOUT* message and the delta value from the previous message.