6

Clock Manager

6.1 Introduction

The **clock manager** provides support for time of day and other time related capabilities. The directives provided by the clock manager are:

Name	Directive Description
clock_set	Set system date and time
clock_get	Get system date and time information
clock_tick	Announce a clock tick

6.2 Background

6.2.1 Required Support

For the features provided by the clock manager to be utilized, periodic timer interrupts are required. Therefore, a real-time clock or hardware timer is necessary to create the timer interrupts. The **clock_tick** directive is normally called by the timer ISR to announce to **RTEMS** that a system clock tick has occurred. Elapsed time is measured in ticks. A tick is defined to be an integral number of microseconds which is specified by the user in the **Configuration Table**.

6.2.2 Time and Date Data Structures

The clock facilities of the clock manager operate upon calendar time. These directives utilize the following date and time structure for the native time and date format:

```
struct rtems_tod_control {
  rtems_unsigned32 year;   /* greater than 1987       */
  rtems_unsigned32 month;  /* 1 - 12                  */
  rtems_unsigned32 day;    /* 1 - 31                  */
  rtems_unsigned32 hour;   /* 0 - 23                  */
  rtems_unsigned32 minute; /* 0 - 59                  */
  rtems_unsigned32 second; /* 0 - 59                  */
  rtems_unsigned32 ticks;  /* elapsed between seconds */
};

typedef struct rtems_tod_control rtems_time_of_day;
```

The native date and time format is the only format supported when setting the system date and time using the **clock_get** directive. Some applications expect to operate on a "UNIX-style" date and time data structure. The **clock_get** directive can optionally return the current date and time in the following structure:

```
typedef struct {
  rtems_unsigned32 seconds;       /* seconds since RTEMS epoch*/
  rtems_unsigned32 microseconds;  /* since last second        */
} rtems_clock_time_value_control;
```

The **seconds** field in this structure is the number of seconds since the **RTEMS** epoch of January 1, 1988.

6.2.3 Timeslicing

Timeslicing is a task scheduling discipline in which tasks of equal priority are executed for a specific period of time before control of the CPU is passed to another task. It is also sometimes referred to as the **automatic round-robin** scheduling algorithm. The length of time allocated to each task is known as the **quantum** or **timeslice**.

The system's timeslice is defined as an integral number of ticks, and is specified in the **Configuration Table**. The timeslice is defined for the entire system of tasks, but timeslicing is enabled and disabled on a per task basis.

The **clock_tick** directive implements timeslicing by decrementing the running task's time-remaining counter when both timeslicing and preemption are enabled. If the task's timeslice has expired, then that task will be preempted if there exists a ready task of equal priority.

6.2.4 Delays

A **sleep timer** allows a task to delay for a given interval or up until a given time, and then wake and continue execution. This type of timer is created automatically by the **task_wake_after** and **task_wake_when** directives and, as a result, does not have an **RTEMS** ID. Once activated, a sleep timer cannot be explicitly deleted. Each task may activate one and only one sleep timer at a time.

6.2.5 Timeouts

Timeouts are a special type of timer automatically created when the timeout option is used on the **message_queue_receive**, **event_receive**, **semaphore_obtain** and **region_get_segment** directives. Each task may have one and only one timeout active at a time. When a timeout expires, it unblocks the task with a timeout status code.

6.3 Operations

6.3.1 Announcing a Tick

RTEMS provides the **clock_tick** directive which is called from the user's real-time clock ISR to inform **RTEMS** that a tick has elapsed. The tick frequency value, defined in microseconds, is a configuration parameter found in the **Configuration Table**. **RTEMS** divides one million microseconds (one second) by the number of microseconds per tick to determine the number of calls to the **clock_tick** directive per second. The frequency of **clock_tick** calls determines the resolution (granularity) for all time dependent **RTEMS** actions. For example, calling **clock_tick** ten times per second yields a higher resolution than calling **clock_tick** two times per second. The **clock_tick** directive is responsible for maintaining both calendar time and the dynamic set of timers.

6.3.2 Setting the Time

The **clock_set** directive allows a task or an ISR to set the date and time maintained by **RTEMS**. If setting the date and time causes any outstanding timers to pass their deadline, then the expired timers will be fired during the invocation of the **clock_set** directive.

6.3.3 Obtaining the Time

The **clock_get** directive allows a task or an ISR to obtain the current date and time or date and time related information. The current date and time can be returned in

either native or UNIX-style format. Additionally, the application can obtain date and time related information such as the number of seconds since the **RTEMS** epoch, the number of ticks since the executive was initialized, and the number of ticks per second. The information returned by the **clock_get** directive is dependent on the option selected by the caller. The following options are available:

Option	Description
CLOCK_GET_TOD	obtain native style date and time
CLOCK_GET_TIME_VALUE	obtain UNIX-style date and time
CLOCK_GET_TICKS_SINCE_BOOT	obtain number of ticks since **RTEMS** was initialized
CLOCK_GET_SECONDS_SINCE_EPOCH	obtain number of seconds since **RTEMS** epoch
CLOCK_GET_TICKS_PER_SECOND	obtain number of clock ticks per second

Calendar time operations will return an error code if invoked before the date and time have been set.

6.4 Directives

This section details the clock manager's directives. A subsection is dedicated to each of this manager's directives and describes the calling sequence, related constants, usage, and status codes.

6.4.1 CLOCK_SET - Set system date and time

CALLING SEQUENCE:

```
rtems_status_code rtems_clock_set(
  rtems_time_of_day *time_buffer
);
```

DIRECTIVE STATUS CODES:

SUCCESSFUL	date and time set successfully
INVALID_TIME_OF_DAY	invalid time of day

DESCRIPTION:

This directive sets the system date and time. The date, time, and ticks in the **time_buffer** structure are all range-checked, and an error is returned if any one is out of its valid range.

NOTES:

Years before 1988 are invalid.

The system date and time are based on the configured tick rate (number of microseconds in a tick).

Setting the time forward may cause a higher priority task, blocked waiting on a specific time, to be made ready. In this case, the calling task will be preempted after the next clock tick.

Re-initializing **RTEMS** causes the system date and time to be reset to an uninitialized state. Another call to **clock_set** is required to re-initialize the system date and time to application specific specifications.

6.4.2 CLOCK_GET - Get system date and time information

CALLING SEQUENCE:

```
rtems_status_code rtems_clock_get(
  rtems_clock_get_options  option,
  void                    *time_buffer
);
```

DIRECTIVE STATUS CODES:

SUCCESSFUL	current time obtained successfully
NOT_DEFINED	system date and time is not set

DESCRIPTION:

This directive obtains the system date and time. If the caller is attempting to obtain the date and time (i.e. **option** is set to either **CLOCK_GET_SECONDS_SINCE_EPOCH**, **CLOCK_GET_TOD**, or **CLOCK_GET_TIME_VALUE)** and the date and time has not been set with a previous call to **clock_set**, then the **NOT_DEFINED** status code is returned. The caller can always obtain the number of ticks per second (**option** is **CLOCK_GET_TICKS_PER_SECOND**) and the number of ticks since the executive was initialized **option** is **CLOCK_GET_TICKS_SINCE_BOOT**).

NOTES:

This directive is callable from an ISR.

This directive will not cause the running task to be preempted. Re-initializing **RTEMS** causes the system date and time to be reset to an uninitialized state. Another call to **clock_set** is required to re-initialize the system date and time to application specific specifications.

6.4.3 CLOCK_TICK - Announce a clock tick

CALLING SEQUENCE:

```
rtems_status_code rtems_clock_tick( void );
```

DIRECTIVE STATUS CODES:

SUCCESSFUL current time obtained successfully

DESCRIPTION:

This directive announces to **RTEMS** that a system clock tick has occurred. The directive is usually called from the timer interrupt **ISR** of the local processor. This directive maintains the system date and time, decrements timers for delayed tasks, timeouts, rate monotonic periods, and implements timeslicing.

NOTES:

This directive is typically called from an ISR.

The **microseconds_per_tick** and **ticks_per_timeslice** parameters in the **Configuration Table** contain the number of microseconds per tick and number of ticks per timeslice, respectively.