9

Timing Data

9.1 Introduction

The timing data for the MC68020 version of **RTEMS** is provided along with the target dependent aspects concerning the gathering of the timing data. The hardware platform used to gather the times is described to give the reader a better understanding of each directive time provided. Also, provided is a description of the interrupt latency and the context switch times as they pertain to the MC68020 version of **RTEMS**.

9.2 Hardware Platform

All times reported except for the maximum period interrupts are disabled by **RTEMS** were measured using a Motorola MVME135 CPU board. The MVME135 is a 20Mhz board with one wait state dynamic memory and a MC68881 numeric coprocessor. The Zilog 8036 countdown timer on this board was used to measure elapsed time with a one-half microsecond resolution. All sources of hardware interrupts were disabled, although the interrupt level of the MC68020 allows all interrupts.

The maximum period interrupts are disabled was measured by summing the number of CPU cycles required by each assembly language instruction executed while interrupts were disabled. The worst case times of the MC68020 microprocessor were used for each instruction. Zero wait state memory was assumed. The total CPU cycles executed with interrupts disabled, including the instructions to disable and enable interrupts, was divided by 20 to simulate a 20Mhz MC68020. It should be noted that the worst case instruction times for the

MC68020 assume that the internal cache is disabled and that no instructions overlap.

9.3 Interrupt Latency

The maximum period with interrupts disabled within **RTEMS** is less than TBD microseconds including the instructions which disable and re-enable interrupts. The time required for the MC68020 to vector an interrupt and for the **RTEMS** entry overhead before invoking the user's interrupt handler are a total of 12 microseconds. These combine to yield a worst case interrupt latency of less than TBD + 12.0 microseconds at 20Mhz.

It should be noted again that the maximum period with interrupts disabled within **RTEMS** is hand-timed and based upon worst case (i.e. CPU cache disabled and no instruction overlap) times for a 20Mhz MC68020. The interrupt vector and entry overhead time was generated on an MVME135 benchmark platform using the Multiprocessing Communications registers to generate as the interrupt source.

9.4 Context Switch

The **RTEMS** processor context switch time is 15 microseconds on the MVME135 benchmark platform when no floating point context is saved or restored. Additional execution time is required when a **TASK_SWITCH** user extension is configured. The use of the **TASK_SWITCH** extension is application dependent. Thus, its execution time is not considered part of the raw context switch time.

Since **RTEMS** was designed specifically for embedded missile applications which are floating point intensive, the executive is optimized to avoid unnecessarily saving and restoring the state of the numeric coprocessor. The state of the numeric coprocessor is only saved when an **FLOATING_POINT** task is dispatched and that task was not the last task to utilize the coprocessor. In a system with only one **FLOATING_POINT** task, the state of the numeric coprocessor will never be saved or restored. When the first **FLOATING_POINT** task is dispatched, **RTEMS** does not need to save the current state of the numeric coprocessor.

The exact amount of time required to save and restore floating point context is dependent on whether an MC68881 or MC68882 is being used as well as the state of the numeric coprocessor. These numeric coprocessors define three operating states: initialized, idle, and busy. **RTEMS** places the coprocessor in the initialized state when a task is started or restarted. Once the task has utilized the coprocessor, it is in the idle state when floating point instructions are not executing and the busy state when floating point instructions are executing. The state of the coprocessor is task specific.

The following table summarizes the context switch times for the MVME135 benchmark platform:

Description	Microseconds
No Floating Point Contexts	15
Floating Point Contexts	
restore first FP task	22
save initialized, restore initialized	51
save idle, restore initialized	66
save idle, restore idle	66

9.5 Directive Times

This sections is divided into a number of subsections, each of which contains a table listing the execution times of that manager's directives.

9.6 Task Manager

Directive	Microseconds
TASK_CREATE	68
TASK_IDENT	80
TASK_START	53
TASK_RESTART	
calling task	61
suspended task -- returns to caller	66
blocked task -- returns to caller	99
ready task -- returns to caller	68
suspended task -- preempts caller	83
blocked task -- preempts caller	106
ready task -- preempts caller	99
TASK_DELETE	
calling task	84
suspended task	68
blocked task	73
ready task	72
TASK_SUSPEND	
calling task	51
returns to caller	36
TASK_RESUME	
task readied -- returns to caller	37
task readied -- preempts caller	58
TASK_SET_PRIORITY	
obtain current priority	24
returns to caller	58
preempts caller	82
TASK_MODE	
obtain current mode	13
no reschedule	12
reschedule -- returns to caller	18
reschedule -- preempts caller	40
TASK_GET_NOTE	27
TASK_SET_NOTE	28
TASK_WAKE_AFTER	
yield -- returns to caller	17
yield -- preempts caller	38
TASK_WAKE_WHEN	93

9.7 Interrupt Manager

It should be noted that the interrupt entry times include vectoring the interrupt handler.

Directive	Microseconds
Interrupt Entry Overhead	
returns to nested interrupt	12
returns to interrupted task	9
returns to preempting task	8
Interrupt Exit Overhead	
returns to nested interrupt	<1
returns to interrupted task	8
returns to preempting task	36

9.8 Clock Manager

Directive	Microseconds
CLOCK_SET	76
CLOCK_GET	2
CLOCK_TICK	14

9.9 Timer Manager

Directive	Microseconds
TIMER_CREATE	18
TIMER_IDENT	78
TIMER_DELETE	
inactive	36
active	40
TIMER_FIRE_AFTER	
inactive	54
active	58
TIMER_FIRE_WHEN	
inactive	85
active	85
TIMER_RESET	
inactive	51
active	54
TIMER_CANCEL	
inactive	27
active	31

9.10 Semaphore Manager

Directive	Microseconds
SEMAPHORE_CREATE	37
SEMAPHORE_IDENT	76
SEMAPHORE_DELETE	46
SEMAPHORE_OBTAIN	
available	33
not available -- NO_WAIT	33
not available -- caller blocks	90
SEMAPHORE_RELEASE	
no waiting tasks	36
task readied -- returns to caller	57
task readied -- preempts caller	76

9.11 Message Manager

Directive	Microseconds
MESSAGE_QUEUE_CREATE	37
MESSAGE_QUEUE_IDENT	76
MESSAGE_QUEUE_DELETE	48
MESSAGE_QUEUE_SEND	
no waiting tasks	58
task readied -- returns to caller	60
task readied -- preempts caller	80
MESSAGE_QUEUE_URGENT	
no waiting tasks	57
task readied -- returns to caller	60
task readied -- preempts caller	80
MESSAGE_QUEUE_BROADCAST	
no waiting tasks	39
task readied -- returns to caller	72
task readied -- preempts caller	92
MESSAGE_QUEUE_RECEIVE	
available	48
not available -- NO_WAIT	35
not available -- caller blocks	90
MESSAGE_QUEUE_FLUSH	
no messages flushed	25
messages flushed	35

9.12 Event Manager

Directive	Microseconds
EVENT_SEND	
no task readied	21
task readied -- returns to caller	51
task readied -- preempts caller	73
EVENT_RECEIVE	
obtain current events	1
available	26
not available -- NO_WAIT	24
not available -- caller blocks	63

9.13 Signal Manager

Directive	Microseconds
SIGNAL_CATCH	13
SIGNAL_SEND	
returns to caller	30
signal to self	34
EXIT ASR OVERHEAD	
returns to calling task	23
returns to preempting task	45

9.14 Partition Manager

Directive	Microseconds
PARTITION_CREATE	64
PARTITION_IDENT	76
PARTITION_DELETE	35
PARTITION_GET_BUFFER	
available	33
not available	30
PARTITION_RETURN_BUFFER	42

9.15 Region Manager

Directive	Microseconds
REGION_CREATE	50
REGION_IDENT	77
REGION_DELETE	36
REGION_GET_SEGMENT	
available	53
not available -- NO_WAIT	52
not available -- caller blocks	106
REGION_RETURN_SEGMENT	
no waiting tasks	59
task readied -- returns to caller	113
task readied -- preempts caller	134

9.16 Dual-Ported Memory Manager

Directive	Microseconds
PORT_CREATE	27
PORT_IDENT	76
PORT_DELETE	33
PORT_INTERNAL_TO_EXTERNAL	23
PORT_EXTERNAL_TO_INTERNAL	24

9.17 I/O Manager

Directive	Microseconds
IO_INITIALIZE	14
IO_OPEN	14
IO_CLOSE	12
IO_READ	14
IO_WRITE	14
IO_CONTROL	13

9.18 Rate Monotonic Manager

Directive	Microseconds

RATE_MONOTONIC_CREATE	25
RATE_MONOTONIC_IDENT	76
RATE_MONOTONIC_CANCEL	35
RATE_MONOTONIC_DELETE	
active	43
inactive	40
RATE_MONOTONIC_PERIOD	
initiate period -- returns to caller	43
conclude period -- caller blocks	65
obtain status	28