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 are 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 14 microseconds. These combine to yield a worst case interrupt latency of less than TBD + 14.0 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 30 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	30
Floating Point Contexts	
restore first FP task	68
save initialized, restore initialized	97
save idle, restore initialized	67
save idle, restore idle	67

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	113
TASK_IDENT	85
TASK_START	102
TASK_RESTART	
calling task	128
suspended task -- returns to caller	128
blocked task -- returns to caller	175
ready task -- returns to caller	130
suspended task -- preempts caller	163
blocked task -- preempts caller	195
ready task -- preempts caller	182
TASK_DELETE	
calling task	159
suspended task	122
blocked task	129
ready task	130
TASK_SUSPEND	
calling task	99
returns to caller	67
TASK_RESUME	
task readied -- returns to caller	67
task readied -- preempts caller	99
TASK_SET_PRIORITY	
obtain current priority	46
returns to caller	92
preempts caller	134
TASK_MODE	
obtain current mode	28
no reschedule	32
reschedule -- returns to caller	37
reschedule -- preempts caller	80
TASK_GET_NOTE	49
TASK_SET_NOTE	48
TASK_WAKE_AFTER	
yield -- returns to caller	21
yield -- preempts caller	65
TASK_WAKE_WHEN	161

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	14
returns to interrupted task	10
returns to preempting task	10
Interrupt Exit Overhead	
returns to nested interrupt	2
returns to interrupted task	10
returns to preempting task	63

9.8 Clock Manager

Directive	Microseconds
CLOCK_SET	107
CLOCK_GET	9
CLOCK_TICK	19

9.9 Timer Manager

Directive	Microseconds
TIMER_CREATE	38
TIMER_IDENT	81
TIMER_DELETE	
inactive	57
active	62
TIMER_FIRE_AFTER	
inactive	83
active	89
TIMER_FIRE_WHEN	
inactive	117
active	122
TIMER_RESET	
inactive	75
active	81
TIMER_CANCEL	
inactive	46
active	51

9.10 Semaphore Manager

Directive	Microseconds
SEMAPHORE_CREATE	63
SEMAPHORE_IDENT	81
SEMAPHORE_DELETE	90
SEMAPHORE_OBTAIN	
available	50
not available -- NO_WAIT	51
not available -- caller blocks	144
SEMAPHORE_RELEASE	
no waiting tasks	64
task readied -- returns to caller	90
task readied -- preempts caller	122

9.11 Message Manager

Directive	Microseconds
MESSAGE_QUEUE_CREATE	61
MESSAGE_QUEUE_IDENT	82
MESSAGE_QUEUE_DELETE	93
MESSAGE_QUEUE_SEND	
no waiting tasks	99
task readied -- returns to caller	95
task readied -- preempts caller	123
MESSAGE_QUEUE_URGENT	
no waiting tasks	99
task readied -- returns to caller	95
task readied -- preempts caller	123
MESSAGE_QUEUE_BROADCAST	
no waiting tasks	65
task readied -- returns to caller	114
task readied -- preempts caller	143
MESSAGE_QUEUE_RECEIVE	
available	73
not available -- NO_WAIT	57
not available -- caller blocks	151
MESSAGE_QUEUE_FLUSH	
no messages flushed	43
messages flushed	52

9.12 Event Manager

Directive	Microseconds
EVENT_SEND	
no task readied	46
task readied -- returns to caller	61
task readied -- preempts caller	121
EVENT_RECEIVE	
obtain current events	4
available	44
not available -- NO_WAIT	42
not available -- caller blocks	111

9.13 Signal Manager

Directive	Microseconds
SIGNAL_CATCH	21
SIGNAL_SEND	
returns to caller	56
signal to self	72
EXIT ASR OVERHEAD	
returns to calling task	41
returns to preempting task	75

9.14 Partition Manager

Directive	Microseconds
PARTITION_CREATE	92
PARTITION_IDENT	80
PARTITION_DELETE	63
PARTITION_GET_BUFFER	
available	58
not available	55
PARTITION_RETURN_BUFFER	66

9.15 Region Manager

Directive	Microseconds
REGION_CREATE	77
REGION_IDENT	81
REGION_DELETE	61
REGION_GET_SEGMENT	
available	69
not available -- NO_WAIT	66
not available -- caller blocks	159
REGION_RETURN_SEGMENT	
no waiting tasks	78
task readied -- returns to caller	157
task readied -- preempts caller	189

9.16 Dual-Ported Memory Manager

Directive	Microseconds
PORT_CREATE	39
PORT_IDENT	79
PORT_DELETE	44
PORT_INTERNAL_TO_EXTERNAL	34
PORT_EXTERNAL_TO_INTERNAL	32

9.17 I/O Manager

Directive	Microseconds
IO_INITIALIZE	17
IO_OPEN	17
IO_CLOSE	19
IO_READ	18
IO_WRITE	19
IO_CONTROL	17

9.18 Rate Monotonic Manager

Directive	Microseconds
RATE_MONOTONIC_CREATE	38
RATE_MONOTONIC_IDENT	81
RATE_MONOTONIC_CANCEL	49
RATE_MONOTONIC_DELETE	
active	62
inactive	57
RATE_MONOTONIC_PERIOD	
initiate period -- returns to caller	66
conclude period -- caller blocks	95
obtain status	40