

VxWorks® Benchmark Data Sheet

VxWorks 6.8

300 MHz Wind River SBCPowerQUICCII

Introduction

This datasheet presents benchmark results for a PowerPC PPC603 Wind River SBCPowerQUICCII target board running VxWorks 6.8. For definitions of the individual benchmarks in this datasheet, please refer to the Glossary at the end of this report.

Target Board Configuration

- wrSbcPowerQuiccll
- 300 MHz PPC603
- 16 KB L1 I-cache
- 16 KB L1 D-cache
- MPC2605 external L2 cache

Results

The benchmark results presented in this datasheet consist of interrupt and task response times, context switch times, and various kernel and user-mode routine timings.

VxWorks 6, like VxWorks 5 before it, fully meets the needs of devices requiring hard real-time performance. Of course, new features in VxWorks 6, such as MMU-based memory protection, have necessitated low-level changes to the RTOS. The performance of the individual OS routines listed in this datasheet may have been affected, but may not in general be relevant to application steady-state performance.

For example, although object creation and deletion routines have become slower, they are usually used in initialization and shutdown procedures and are not generally used where hard real-time functionality is required.

While the benchmark numbers reported in this datasheet are provided as a guideline to VxWorks 6 performance, how the performance of these routines affects overall application performance depends in large part on the behavior of the specific application.

General Timing Information

The measurements in this benchmark are obtained

using a VxWorks timestamp driver.

All times are given in microseconds (μ s), unless otherwise noted.

Maximal Performance Profile

The Maximal Performance Profile is a build of the VxWorks runtime source, intended to approximate deployed kernel-mode-only VxWorks systems. This profile builds the kernel without support for the following:

- Real-Time Processes (RTP)
- System-Viewer Instrumentation
- Kernel object ID validation checks
- Kernel object ownership information
- Kernel object class lists

Kernel Mode Timings

page 2

	Default Build µs	Maximal Performance µs
Key Measurements		
Interrupt response	3.54	3.20
Task response	8.08	6.28
Task Context Switches		
Task blocked on unavailable event	3.96	2.55
Task unblocked by incoming event	4.07	2.69
Task blocked by unavailable message	4.69	2.72
Task unblocked by incoming message	6.37	4.41
Task blocked by unavailable semaphore	4.54	2.50
Task unblocked by available semaphore	3.99	2.17
Semaphores		
semCreate		
Binary	10.45	6.29
Counting	10.05	6.22
Mutual Exclusion	9.84	6.57
semDelete		
Binary	10.73	5.87
Counting	10.73	6.11
Mutual Exclusion	10.79	6.11
semGive		
Tasks in queue		
Binary	1.33	1.03
Binary inline	1.16	0.87
Counting	1.22	1.04
Mutual Exclusion	1.86	1.49
Mutual Exclusion inline	1.39	1.30
No tasks in queue		
Binary	0.44	0.33
Binary inline	0.14	0.15
Counting	0.45	0.34
Mutual Exclusion	0.66	0.54
Mutual Exclusion inline	0.25	0.25
semTake		
Semaphore available		
Binary	0.44	0.34
Binary inline	0.12	0.14
Counting	0.45	0.34
Mutual Exclusion	0.55	0.42
Mutual Exclusion inline	0.19	0.21
Semaphore unavailable		
Binary	0.58	0.44
Binary inline	0.22	0.20
Counting	0.58	0.44
Mutual Exclusion	0.49	0.36
Mutual Exclusion inline	0.22	0.21

page 2

	Default Build μs	Maximal Performance μs
Reader-Writer semaphores		
semCreate	11.29	9.16
semDelete	11.28	6.53
semGive		
No tasks in queue		
acquired for writing, no tasks pending	0.76	0.63
acquired for reading, no tasks pending	0.84	0.71
Tasks in queue		
writer in queue; acquired for writing	1.71	1.63
writer in queue; acquired for reading	0.78	0.64
reader in queue; acquired for writing	0.77	0.64
semTake		
Semaphore available		
caller owns semaphore for writing	0.66	0.54
caller owns semaphore for reading	0.64	0.54
Semaphore unavailable		
caller owns semaphore for writing	0.59	0.46
caller owns semaphore for reading	0.52	0.42
Tasks		
taskInit	57.93	40.42
taskActivate	1.11	0.96
taskLock	0.08	0.04
taskUnlock	0.19	0.20
taskResume		
Ready task	0.64	0.61
Pended task	0.64	0.61
Suspended task	0.91	0.84
Delayed task	0.63	0.61
taskSuspend		
Ready task	0.92	0.87
Pended task	0.65	0.58
Suspended task	0.62	0.58
Delayed task	0.62	0.59
taskPrioritySet		
Ready task	1.45	1.40
Pended task	0.92	0.85
taskSpawn	81.46	55.60
taskDelete	53.48	40.10
Message Queues		
msgQReceive		
Message available	1.51	1.35
Message unavailable	1.17	1.01
msgQSend		
No tasks pending	1.81	1.68
Tasks pending	2.65	2.38
Queue full	1.27	1.15
msgQCreate	12.37	9.91
msgQDelete	8.27	5.94

	Default Build	Maximal Performance
	µs	µs
Events		
Send self	0.38	0.35
Receive available	0.44	0.45
Receive unavailable	0.62	0.52
Task send wanted	1.39	1.34
Task send unwanted	0.41	0.38
Watchdogs		
wdCreate	6.81	4.35
wdDelete		
Timer started	9.43	5.44
Timer not started	9.06	5.23
wdStart		
No timer in queue	1.15	1.01
Timer in queue	1.32	1.15
wdCancel	0.86	0.70

User Mode Timings

page 5

Default Build

μs

Key Measurements

System call overhead	6.74
----------------------	------

Context Switches

Within an RTP

Task blocked on unavailable event	23.81
Task unblocked by incoming event	21.14
Task blocked by unavailable message	25.76
Task unblocked by incoming message	32.23
Task blocked by unavailable semaphore	23.94
Task unblocked by available semaphore	23.93

RTP-to-RTP

Task blocked on unavailable event	26.27
Task unblocked by incoming event	22.20
Task blocked by unavailable message	24.83
Task unblocked by incoming message	36.35
Task blocked by unavailable semaphore	24.84
Task unblocked by available semaphore	25.58

Semaphores

semCreate

Binary	74.76
Counting	74.76
Mutual Exclusion	75.73

semDelete

Binary	88.97
Counting	91.91
Mutual Exclusion	91.92

semGive

Tasks in queue	
Binary	14.32
Counting	13.63
Mutual Exclusion	37.29
No tasks in queue	
Binary	11.35
Counting	11.06
Mutual Exclusion	1.18

semTake

Semaphore available	
Binary	11.02
Counting	10.99
Mutual Exclusion	1.03
Semaphore unavailable	
Binary	11.53
Counting	11.52
Mutual Exclusion	1.12

page 5

Tasks	
taskSpawn	620.29
taskDelete	538.30
taskResume	
Ready task	10.76
Pended task	10.81
Suspended task	12.05
Delayed task	10.77
taskSuspend	
Ready task	11.54
Pended task	10.53
Suspended task	10.40
Delayed task	10.39
Events	
Send self	9.91
Receive available	11.06
Receive unavailable	12.47
Task send wanted	13.51
Task send unwanted	11.78
Message Queues	
msgQCreate	39.37
msgQDelete	31.33
msgQReceive	
Message available	12.96
Message unavailable	14.06
msgQSend	
No tasks pending	13.38
Task pending	17.05
Queue full	15.78

Glossary of Benchmark Metrics

Context Switching

Task context switch time

Task blocked on unavailable event

Task context switch time caused by the outgoing task executing eventReceive() on an unavailable event.

Task unblocked by incoming event

Task context switch time caused by the outgoing task executing eventSend() to the incoming higher priority task.

Task blocked by unavailable message

Task context switch time caused by the outgoing task executing msgQReceive() on an unavailable message queue.

Task unblocked by incoming message

Task context switch time caused by the outgoing task executing msgQSend() to the incoming higher priority task.

Task blocked by unavailable semaphore

Task context switch time caused by the outgoing task executing semTake() on an unavailable semaphore.

Task unblocked by available semaphore

Task context switch time caused by the outgoing task executing semGive() to the incoming higher priority task.

Events

Send self

Task to send event to self

Receive available

Receive event that is already available (does not block)

Receive unavailable

Receive unavailable event with no wait (does not block)

Task send wanted

Send an event wanted by another lower-priority task

Task send unwanted

Send an event that no task is currently waiting for

Interrupt Response

The time from when an interrupt is raised until the user-installed handler starts execution. This time includes: saving the interrupted task context, setting up the interrupt context, and identifying the source of the interrupt by the interrupt controller driver, plus the overhead of the timer driver used as the auxiliary clock.

Task response

The task response is the time elapsed from when an interrupt is raised until a task resumes execution after completion of the interrupt handling. There are two types task response times provided. In the first case, there is no rescheduling, and the interrupted task resumes execution after the interrupt handling completed. In the second case, the interrupt handler causes rescheduling by giving a binary semaphore that unblocks a different task than the task that was interrupted.