WIND RIVER

VxWorks® Benchmark Data Sheet

VxWorks 6.8
600 MHz Wind River SBC8560 Reference Board

Introduction

This datasheet presents benchmark results for a PowerPC MPC8560 Wind River SBC8560 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

- wrSbc8560
- 600 MHz MPC8560
- 32 KB L1 I-cache
- 32 KB L1 D-cache
- 256 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 (µs), 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

WIND RIVER

Kernel Mode Timings

	Default Build µs	Maximal Performance µs
Key Measurements		
Interrupt response	0.53	0.53
Task response	2.39	2.23
Task Context Switches		
Task blocked on unavailable event	1.41	1.28
Task unblocked by incoming event	1.51	1.40
Task blocked by unavailable message	1.53	1.32
Task unblocked by incoming message	2.41	2.22
Task blocked by unavailable semaphore	1.40	1.26
Task unblocked by available semaphore	1.34	1.17
Semaphores		
semCreate		
Binary	3.79	3.13
Counting	3.78	3.14
Mutual Exclusion	3.77	3.12
semDelete		
Binary	3.46	2.51
Counting	3.46	2.51
Mutual Exclusion	3.46	2.50
semGive		
Tasks in queue		
Binary	0.61	0.56
Binary inline	0.46	0.44
Counting	0.61	0.55
Mutual Exclusion	0.77	0.73
Mutual Exclusion inline	0.62	0.61
No tasks in queue		
Binary	0.23	0.19
Binary inline	0.09	0.09
Counting	0.23	0.20
Mutual Exclusion	0.32	0.28
Mutual Exclusion inline	0.21	0.19
semTake		
Semaphore available		
Binary	0.24	0.19
Binary inline	0.16	0.12
Counting	0.24	0.19
Mutual Exclusion	0.27	0.22
Mutual Exclusion inline	0.11	0.11
Semaphore unavailable		
Binary	0.29	0.25
Binary inline	0.20	0.18
Counting	0.30	0.25
Mutual Exclusion	0.26	0.20
Mutual Exclusion inline	0.20	0.18

	Default Build µs	Maximal Performance µs
Reader-Writer semaphores		
semCreate	4.86	4.24
semDelete	3.80	2.90
semGive		
No tasks in queue		
acquired for writing, no tasks pending	0.35	0.31
acquired for reading, no tasks pending	0.36	0.32
Tasks in queue		
writer in queue; acquired for writing	0.79	0.73
writer in queue; acquired for reading	0.35	0.31
reader in queue; acquired for writing	0.36	0.32
semTake		
Semaphore available		
caller owns semaphore for writing	0.31	0.26
caller owns semaphore for reading	0.29	0.25
Semaphore unavailable		
caller owns semaphore for writing	0.28	0.23
caller owns semaphore for reading	0.24	0.20
Tasks		
taskInit	15.08	13.30
taskActivate	0.47	0.46
taskLock	0.05	0.02
taskUnlock	0.08	0.07
taskResume		
Ready task	0.35	0.33
Pended task	0.35	0.33
Suspended task	0.47	0.45
Delayed task	0.35	0.33
taskSuspend		
Ready task	0.43	0.40
Pended task	0.31	0.29
Suspended task	0.30	0.29
Delayed task	0.31	0.29
taskPrioritySet		
Ready task	0.68	0.67
Pended task	0.46	0.44
taskSpawn	19.54	16.62
taskDelete	16.68	14.16
Message Queues		
msgQReceive		
Message available	0.67	0.62
Message unavailable	0.50	0.45
msgQSend		
No tasks pending	0.86	0.80
Tasks pending	1.23	1.14
Queue full	0.53	0.49
msgQCreate	5.64	4.84
msgQDelete	3.53	2.77

	Default Build µs	Maximal Performance µs
Events		
Send self	0.19	0.18
Receive available	0.22	0.22
Receive unavailable	0.29	0.29
Task send wanted	0.67	0.64
Task send unwanted	0.20	0.19
Watchdogs		
wdCreate	2.70	2.23
wdDelete		
Timer started	3.27	2.34
Timer not started	3.17	2.25
wdStart		
No timer in queue	0.58	0.53
Timer in queue	0.66	0.60
wdCancel	0.45	0.38

User Mode Timings

	Default Build µs
Key Measurements	
System call overhead	0.87
Context Switches	
Within an RTP	
Task blocked on unavailable event	3.56
Task unblocked by incoming event	3.35
Task blocked by unavailable message	3.84
Task unblocked by incoming message	4.76
Task blocked by unavailable semaphore	3.54
Task unblocked by available semaphore	3.45
RTP-to-RTP	
Task blocked on unavailable event	3.50
Task unblocked by incoming event	4.12
Task blocked by unavailable message	3.87
Task unblocked by incoming message	4.71
Task blocked by unavailable semaphore	3.50
Task unblocked by available semaphore	3.70
Semaphores	
semCreate	
Binary	9.02
Counting	9.02
Mutual Exclusion	8.75
semDelete	
Binary	12.17
Counting	12.64
Mutual Exclusion	12.33
semGive	
Tasks in queue	
Binary	2.06
Counting	2.07
Mutual Exclusion	4.74
No tasks in queue	
Binary	1.62
Counting	1.63
Mutual Exclusion	0.45
semTake	
Semaphore available	
Binary	1.63
Counting	1.63
Mutual Exclusion	0.43
Semaphore unavailable	
Binary	1.84
Counting	1.85
Mutual Exclusion	0.43

	Default Build µs
Tasks	
taskSpawn	156.03
taskDelete	139.77
taskResume	
Ready task	1.84
Pended task	1.84
Suspended task	1.97
Delayed task	1.84
taskSuspend	
Ready task	2.00
Pended task	1.91
Suspended task	1.86
Delayed task	1.88
Events	
Send self	1.14
Receive available	1.41
Receive unavailable	1.65
Task send wanted	2.04
Task send unwanted	1.54
Message Queues	
msgQCreate	7.91
msgQDelete	7.09
msgQReceive	
Message available	2.29
Message unavailable	2.27
msgQSend	
No tasks pending	2.50
Task pending	2.84
Queue full	2.31

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.