

QNX V6.5 ON X86 ATOM

© Copyright Dedicated Systems Experts NV. All rights reserved, no part of the contents of this document may be reproduced or transmitted in any form or by any means without the written permission of Dedicated Systems Experts NV, Diepenbeemd 5, B-1650 Beersel, Belgium.

Authors: Luc Perneel (1, 2), Hasan Fayyad-Kazan (2) and Martin Timmerman (1, 2, 3)

1: Dedicated Systems Experts, 2: VUB-Brussels, 3: RMA-Brussels

Disclaimer

Although all care has been taken to obtain correct information and accurate test results, Dedicated Systems Experts, VUB-Brussels, RMA-Brussels and the authors cannot be liable for any incidental or consequential damages (including damages for loss of business, profits or the like) arising out of the use of the information provided in this report, even if these organisations and authors have been advised of the possibility of such damages.

<http://www.dedicated-systems.com>

E-mail: info@dedicated-systems.com

EVALUATION REPORT LICENSE

This is a legal agreement between you (the downloader of this document) and/or your company and the company DEDICATED SYSTEMS EXPERTS NV, Diepenbeemd 5, B-1650 Beersel, Belgium.
It is not possible to download this document without registering and accepting this agreement on-line.

1. **GRANT.** Subject to the provisions contained herein, Dedicated Systems Experts hereby grants you a non-exclusive license to use its accompanying proprietary evaluation report for projects where you or your company are involved as major contractor or subcontractor. You are not entitled to support or telephone assistance in connection with this license.
2. **PRODUCT.** Dedicated Systems Experts shall furnish the evaluation report to you electronically via Internet. This license does not grant you any right to any enhancement or update to the document.
3. **TITLE.** Title, ownership rights, and intellectual property rights in and to the document shall remain in Dedicated Systems Experts and/or its suppliers or evaluated product manufacturers. The copyright laws of Belgium and all international copyright treaties protect the documents.
4. **CONTENT.** Title, ownership rights, and an intellectual property right in and to the content accessed through the document is the property of the applicable content owner and may be protected by applicable copyright or other law. This License gives you no rights to such content.
5. **YOU CANNOT:**
 - You cannot, make (or allow anyone else make) copies, whether digital, printed, photographic or others, except for backup reasons. The number of copies should be limited to 2. The copies should be exact replicates of the original (in paper or electronic format) with all copyright notices and logos.
 - You cannot, place (or allow anyone else place) the evaluation report on an electronic board or other form of on line service without authorisation.
6. **INDEMNIFICATION.** You agree to indemnify and hold harmless Dedicated Systems Experts against any damages or liability of any kind arising from any use of this product other than the permitted uses specified in this agreement.
7. **DISCLAIMER OF WARRANTY.** All documents published by Dedicated Systems Experts on the World Wide Web Server or by any other means are provided "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT. This disclaimer of warranty constitutes an essential part of the agreement.
8. **LIMITATION OF LIABILITY.** Neither Dedicated Systems Experts nor any of its directors, employees, partners or agents shall, under any circumstances, be liable to any person for any special, incidental, indirect or consequential damages, including, without limitation, damages resulting from use of OR RELIANCE ON the INFORMATION presented, loss of profits or revenues or costs of replacement goods, even if informed in advance of the possibility of such damages.
9. **ACCURACY OF INFORMATION.** Every effort has been made to ensure the accuracy of the information presented herein. However Dedicated Systems Experts assumes no responsibility for the accuracy of the information. Product information is subject to change without notice. Changes, if any, will be incorporated in new editions of these publications. Dedicated Systems Experts may make improvements and/or changes in the products and/or the programs described in these publications at any time without notice. Mention of non-Dedicated Systems Experts products or services is for information purposes only and constitutes neither an endorsement nor a recommendation.
10. **JURISDICTION.** In case of any problems, the court of BRUSSELS-BELGIUM will have exclusive jurisdiction.

Agreed by downloading the document via the internet.

1	Document Intention.....	6
1.1	Purpose and scope	6
1.2	Document issue: the 2.9 framework	6
1.3	Related documents	7
2	Introduction	8
2.1	Overview	8
2.2	Evaluated (RTOS) product.....	8
2.2.1	Software	8
2.2.2	Hardware	8
3	Evaluation results summary.....	9
3.1	Positive points	9
3.2	Negative points	9
3.3	Ratings	9
4	Test Results.....	10
4.1	Calibration system test (CAL)	10
4.1.1	Tracing overhead (CAL-P-TRC).....	10
4.1.2	CPU power (CAL-P-CPU)	11
4.2	Clock tests (CLK)	12
4.2.1	Operating system clock setting (CLK-B-CFG)	12
4.2.2	Clock tick processing duration (CLK-P-DUR)	12
4.3	Thread tests (THR)	14
4.3.1	Thread creation behaviour (THR-B-NEW)	14
4.3.2	Round robin behaviour (THR-B-RR)	15
4.3.3	Thread switch latency between same priority threads (THR-P-SLS).....	15
4.3.4	Thread creation and deletion time (THR-P-NEW).....	17
4.4	Semaphore tests (SEM).....	21
4.4.1	Semaphore locking test mechanism (SEM-B-LCK)	21
4.4.2	Semaphore releasing mechanism (SEM-B-REL).....	22
4.4.3	Time needed to create and delete a semaphore (SEM-P-NEW).....	22
4.4.4	Test acquire-release timings: contention case (SEM-P-ARN)	24
4.4.5	Test acquire-release timings: contention case (SEM-P-ARC)	26
4.5	Mutex tests (MUT).....	29
4.5.1	Priority inversion avoidance mechanism (MUT-B-ARC)	29
4.5.2	Mutex acquire-release timings: contention case (MUT-P-ARC)	30
4.5.3	Mutex acquire-release timings: no-contention case (MUT-P-ARN)	31
4.6	Interrupt tests (IRQ)	33
4.6.1	Interrupt latency (IRQ_P_LAT).....	33
4.6.2	Interrupt dispatch latency (IRQ_P_DLT)	34
4.6.3	Interrupt to thread latency (IRQ_P_TLT).....	35
4.6.4	Maximum sustained interrupt frequency (IRQ_S_SUS).....	36
4.7	Memory tests.....	37

4.7.1	Memory leak test (MEM_B_LEK)	37
5	Appendix A: Vendor comments	38
6	Appendix B: Acronyms	39

DOCUMENT CHANGE LOG

Issue No.	Revised Issue Date	Para's / Pages Affected	Reason for Change
1	June 2, 2011	All	Initial draft
1.01	June 4, 2011	All	comments
2.0	July 27, 2011	All	Vendor revision draft
3.0	July 28, 2011	All	Vendor revision draft – patch included
3.10	Sept 7, 2011	All	Final report

1 Document Intention

1.1 Purpose and scope

This document presents the quantitative evaluation results of the QNX Neutrino operating system V6.5 employed on an x86 Atom based platform.

The layout of this report follows the one depicted in “The OS evaluation template” [Doc. 4]. The test specifications can be found in “The evaluation test report definition.” [Doc. 3]. See section 1.3 of this document for more detailed references. These documents have to be seen as an integral part of this report!

Due to the tightly coupling between these documents, the framework version of “The evaluation test report definition.” has to match the framework version of this evaluation report (which is 2.9). More information about the documents and tests versions together with their corresponding relation between both can be found in “The evaluation framework”, see [Doc. 1] in section 1.3 of this document.

The generic test code used to perform these tests can be downloaded on our website by using the link in the related documents section.

1.2 Document issue: the 2.9 framework

This document shows the test results in the scope of the evaluation framework 2.9.

1.3 Related documents

These are documents that are closely related to this document. They can all be downloaded using following link:

<http://www.dedicated-systems.com/encyc/buyersguide/rtos/evaluations>

- | | | |
|--------|---|---|
| Doc. 1 | <p>The evaluation framework
This document presents the evaluation framework. It also indicates which documents are available, and how their name giving, numbering and versioning are related. This document is the base document of the evaluation framework.
EVA-2.9-GEN-01</p> | <p>Issue: 1 Date: April 19, 2004</p> |
| Doc. 2 | <p>What is a good RTOS?
This document presents the criteria that Dedicated Systems Experts use to give an operating system the label "Real-Time". The evaluation tests are based upon the criteria defined in this document.
EVA-2.9-GEN-02</p> | |
| Doc. 3 | <p>The evaluation test report definition.
This document presents the different tests issued in this report together with the flowcharts and the generic pseudo code for each test. Test labels are all defined in this document.
EVA-2.9-GEN-03</p> | <p>Issue: 1 April 19, 2004</p> |
| Doc. 4 | <p>The OS evaluation template
This document presents the layout used for all reports in a certain framework.
EVA-2.9-GEN-04</p> | <p>Issue: 1 April 19, 2004</p> |
| Doc. 5 | <p>QNX v6.5, Theoretical evaluation
This document presents the qualitative discussion of the OS
EVA-2.9-OS-QNX-65</p> | <p>Issue: 1 May 20, 2011</p> |

2 Introduction

This chapter talks about the OS that we are going to test and evaluate, and the hardware on which the OS under testing will be employed.

2.1 Overview

QNX Software Systems Ltd was founded in 1980 and has been always focused on delivering solutions for the embedded systems market.

One of the main differences between QNX and other RTOS is the fact that QNX is built around the POSIX API standard. This has its advantages as a lot of code for Linux based platforms can be compiled and run on QNX Neutrino. However, bear in mind that we are discussing a real-time operating system here.

QNX Neutrino is based on true microkernel architecture with message-based inter-process communication. For instance, drivers are just applications with special privileges, and as such they cannot crash the kernel. The concept of kernel modules which is the case in Linux is not needed here, which makes QNX Neutrino a very stable product.

Furthermore, QNX Neutrino was initially built-up as a multi-processor capable operating system (both SMP and AMP). Nowadays, this is a very important asset in today's multi- and many-core business.

2.2 Evaluated (RTOS) product

2.2.1 Software

The operating system that we are going to evaluate is the QNX NEUTRINO RTOS v6.5.0 including patch 2530, from QNX Software Systems Ltd.

2.2.2 Hardware

The hardware that was used for executing our tests for the QNX Neutrino RTOS has the following characteristics:

- Motherboard: Advantech SOM-6760, PCI bus at 33MHz, using the System Controller Hub US15W.
- CPU: Intel Atom Z530 1.6GHz 133MHz Front Side Bus.
 - 32KByte L1 Instruction Cache,
 - 24KByte L1 Write Back Data Cache,
 - 512KByte 8-way L2 Cache (which can be reduced up to zero in some processor sleep states)
 - 1 core with hyper-threading support (however hyper threading was disabled during this test).
- RAM: 512MB DDR2
- VMETRO PCI exerciser in PCI slot 3 (PCI interrupt level D, local bus interrupt level 10)
- VMETRO PBT-315 PCI analyser in PCI slot 4.
- External and CPU internal cache was enabled during the tests.

3 Evaluation results summary

Following is a summary of the results of evaluating the QNX NEUTRINO RTOS v6.5.0, from QNX Software Systems Ltd.

3.1 Positive points








- Excellent architecture for a robust and distributed system.
- Very fast and predictable performance.
- Large number of board support packages (BSP) and drivers (the source for most of them is available for public) which can be easily downloaded.
- The availability of documentation which can be considered more than the average.
- Efficient and user friendly Integrated Development Environment (IDE)

3.2 Negative points

- Not all code is available in source code. Customers can apply for source access.

3.3 Ratings

For a description of the ratings, see [Doc. 3].

RTOS Architecture	0		10
OS Documentation	0		10
OS Configuration	0		10
Internet Components	0		10
Development Tools	0		10
BSPs	0		10
Support	0		10

4 Test Results

Test Results 0  10

QNX 6.5 has very good performance characteristics which guarantee the real-time behaviour.

On the Atom board we detected extra latency peaks around 11µs.

For this platform, we had to adapt our testing methodology a bit. The problem is the existence of the fast hardware and the different hardware FIFOs. On the old Pentium MMX platform, we can use directly PCI memory mapping to our test PCI device.

On this Atom based platform, the PCI chip and processor do write-postponing if the PCI device is memory-mapped (even using un-cached memory). Therefore, it is impossible to have accurate results using memory-mapping. For your information, the Intel chipset PCI Bridge cannot be programmed to disable write-postponing (which is something that is typically feasible on embedded PCI bridge chipsets).

The only solution is to map the PCI device as IO memory. Following the PCI specifications, the writes to IO memory may not be postponed.

There is however a couple of issues with this on x86 Intel based platforms:

- Total amount of IO memory can be only 64KB
- To access IO memory, special x86 IO instructions have to be used.

These special instructions make the writes to the PCI much slower. However, the delay is consistent enough to give us accurate enough results.

4.1 Calibration system test (CAL)

These tests are used to calibrate the tracing overhead compared with the processing power of the platform. This is important to understand the accuracy of the measurements done in scope of this report.

They are used also for measuring the processing power of the platform. This calibration permits comparison with the results on other platforms.

4.1.1 Tracing overhead (CAL-P-TRC)

This test calibrates the tracing system overhead. This is more hardware than OS related, but it is needed to correct the measured times. As on this platform we have to use special instructions to avoid postponed writes, the tracing overhead is much larger than on our default evaluation platform (Pentium MMX 200MHz).

In the rest of this document, the tracing overhead is subtracted from the results obtained.

Tracing accuracy depends here on the PCI clock (33MHz) and the special I/O instructions. As you can see in the table below, the measurement overhead is very high. However the overhead is stable. Therefore we believe that the measured results are accurate up to +/- 0.2 µseconds. Therefore the results shown in the tables are rounded to 0.1 microseconds.

4.1.1.1 Test results

Test	result
Number of samples	30000
Average tracing overhead	2.11 μ sec
minimum tracing overhead	2.04 μ sec
maximum tracing overhead	2.19 μ sec

4.1.2 CPU power (CAL-P-CPU)

This test will calibrate the CPU performance and the memory bandwidth of the used platform. This test is measured in different situations, from the situation where code and data are cached, until the situation where neither code nor data are cached. With such different situation tests, the effects of the cache can be calculated.

We have been seriously reworking this test lately. The CPU test uses only one data address; The non-cached version is about 172KB in size (instructions), while the cached version uses a loop (a bit unrolled to have a small loop overhead but so it fits in the L1 I-cache and it uses only two data words). The instruction cache test is done twice.

4.1.2.1 Test results

The results for our standard platform (Pentium MMX 200 MHz) are shown below:

Test	no cache	cached	cache effect
CPU test: first load.	884.3 us		
CPU test: ICache effect	872.0 us	136.9 us	6.2
MEM write test	394.4 us	392.3 us	1.0
MEM read test	661.9 us	407.0 us	1.7
Average caching effect (CPU and MEM)			3.0

On the Atom platform the same test gives following results:

Test	no cache	cached	cache effect
CPU test: first load.	46.9 us		
CPU test: ICache effect	37.3 us	19.0 us	2.0
MEM write test	18.0 us	13.8 us	1.3
MEM read test	23.2 us	13.8 us	1.7
Average caching effect (CPU and MEM)			1.7

The results show similar behaviour as on the Pentium MMX 200MHz: instruction loading is mostly impacted by the cache. Remark that we are mostly looking to the L1 cache impact. Also L2 cache misses have impact as can be seen on the first load test. However, the L2 cache impact is much harder to measure accurately.

If you compare both platforms, then you can see how the performance is improved in about 15 years!

If you compare this with an ARM cortex A8 at 1GHz, then the Atom is still much faster. Of course, the large cache and the much higher memory bandwidth compared with typical ARM based smartphone platform have a serious impact as well. When in cache, the Atom 1.6GHz is about 50% faster than the Cortex A8 running at 1GHz. However, when out of cache, the performance difference is more than a factor 2 in favour of the Atom.

4.2 Clock tests (CLK)

The clock test measures the time that an operating system needs to handle its clock interrupt. On the tested platform, the clock tick interrupt is set on the highest hardware interrupt level, interrupting any other thread or interrupt handler.

4.2.1 Operating system clock setting (CLK-B-CFG)

This test is done in order to examine the setting of the clock tick period in the operating system. This test shows the default clock timing as they are set by the OS.

For this test, the `nanosleep()` POSIX function call is used. Following POSIX, the delay should be based on the clock tick. The “nanosleep” function always pauses for at least its specified time, but however it can take up to one clock tick more than its specified time until the process becomes run-able again”.

As the OS cannot know when the next tick will occur, it will add the time of one tick and round it up to the higher tick. Using `nanosleep(0)` thus will wake up on next tick.

4.2.1.1 Test results

Test	result
Test succeeded	Yes
Tested clock period	1ms
Clock period adaptable	YES

4.2.2 Clock tick processing duration (CLK-P-DUR)

This test is done for examining the clock tick processing duration in the kernel. The test results are extremely important, as the clock interrupt will disturb all the other performed measurements.

The bottom line of these figures in section 4.2.2.2 represents the normal loop time of the test if no clock interrupt occurs during the test loop. The upper line is generated by the samples when a clock interrupt occurred during the loop. The difference between the two lines is the clock tick processing duration.

In absolute values, the 2.2 µsec clock tick impact is of course not that large on such fast platform. It is as well very stable as you can see in the diagrams below. Sometimes a delay of 10 µsec can be noticed; this is detected in different other tests as well.

The Z530 can achieve around 2GB/s memory bandwidth which mean that the L2 cache can be flushed in $\frac{1}{4}$ of a clock tick (512KB in 0.25ms). In memory intensive applications, it could happen that the clock interrupt is not in the cache; however on most embedded platforms, this will not be the case and thus the clock interrupt will always be present in the L2 cache. This is the reason why the overhead is stable.

This clock tick will impact all other real-time behaviour and measurements in the following tests.

4.2.2.1 Test results

Test	result
CLOCK_LOOP_COUNTER	5000
Normal busy loop time	15.8 μ s
Busy loop time with clock interrupt	18 μ s, worst case 26 μ s
Clock interrupt duration	2.2 μ s to 10 μ s

4.2.2.2 Diagrams

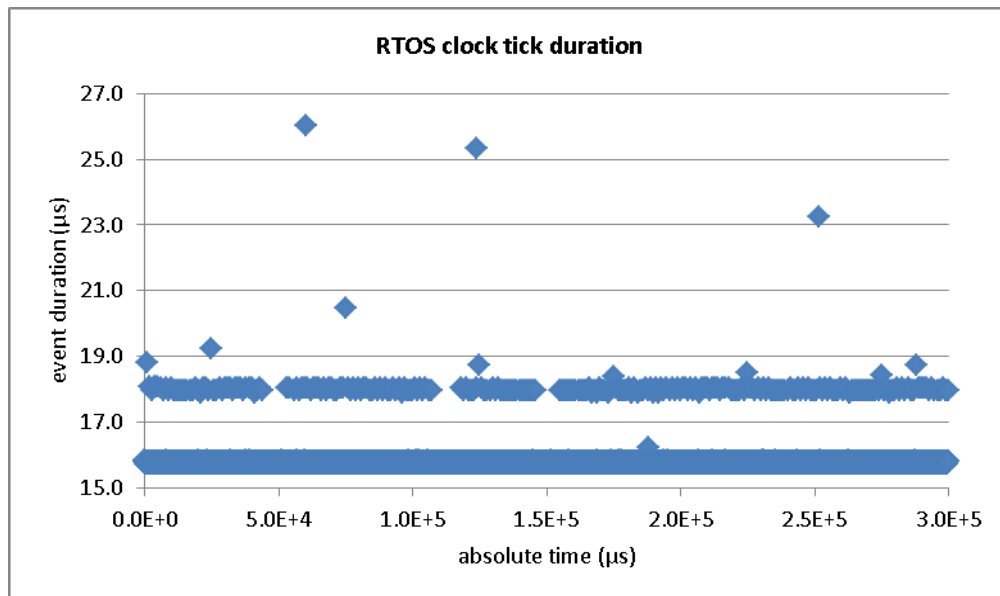


Figure 1: RTOS clock tick duration

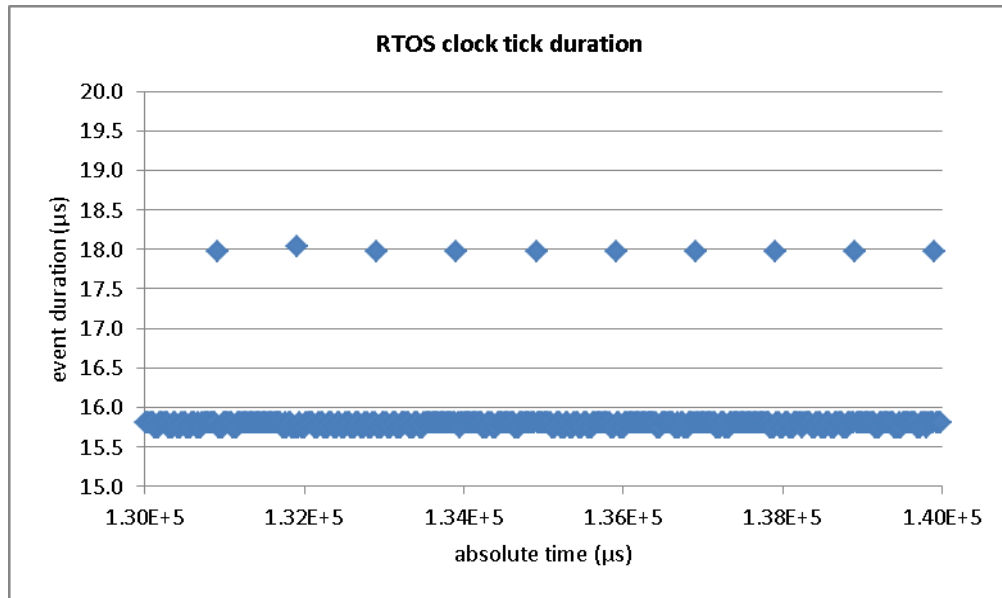


Figure 2: RTOS clock tick duration, detail.

4.3 Thread tests (THR)

These tests are used to measure the performance of the scheduler.

4.3.1 Thread creation behaviour (THR-B-NEW)

This test will examine the behaviour of creating threads. Does the operating system behave as it should be as long as it is considered being a real-time operating system? Following scenarios are tested:

- If a thread is created with a lower priority than the creating thread, then are we sure that it is not activated until the creating thread is finished?
- If a thread is created with the same priority as the creating thread, will it be put at the ready tail?
- When yielding after the creation in the above test, does the newly created thread becomes active?
- If a thread is created with a higher priority than the creating thread, is it then immediately activated?

This test succeeded without any problems.

4.3.1.1 Test results

Test	result
Test succeeded	YES
Lower priority not activated?	YES
Same priority at tail?	YES
Yielding works?	YES
Higher priority activated?	YES

4.3.2 Round robin behaviour (THR-B-RR)

This test checks if the scheduler uses a fair round robin mechanism when threads are having the same priority and all are in the ready-to-run state (and using the SCHED_RR scheduling policy)!

No problems were detected here. The round robin behaviour reschedules a thread each 4 clock ticks.

4.3.2.1 Test results

Test	result
Test succeeded	Yes
RR Time slice following this test	4 ms

4.3.3 Thread switch latency between same priority threads (THR-P-SLS)

This test measures the time to switch between threads of the same priority. Therefore, threads have to yield the processor voluntary for the other threads for using it.

In this test, we use the SCHED_FIFO policy; otherwise it would be possible that a round-robin clock event occurs between the yield and the trace, so that the thread activation is not seen in the trace.

This test was performed several times, and each time using a higher number of threads in order to generate the worst case behaviour. If more threads are active, the caching effect will be obvious in a way that the thread context will not reside anymore in the L1 cache once we have enough threads (it will still be in the L2 cache anyhow, as the L2 cache is much larger).

Further you will see clearly the influence of clock interrupts (causing the higher values in the graphics).

Once there are enough running threads, the clock interrupt will always be un-cached (in L1) and thus for the 1000 thread tests, the clock interrupts always generate a delay of approximately 2 to 3 μ s. Remark that it is still in the L2 cache, so L1 cache misses has less impact.

A spike of around 11/12 μ s (see figures in section 4.3.3.2) was detected here and later on in other different tests. It is unclear what the cause of such spike was. On normal PC platforms having BIOS, we would suspect the BIOS via Service Management Interrupts. However, the board is booted directly with QNX, so there is no BIOS involved.

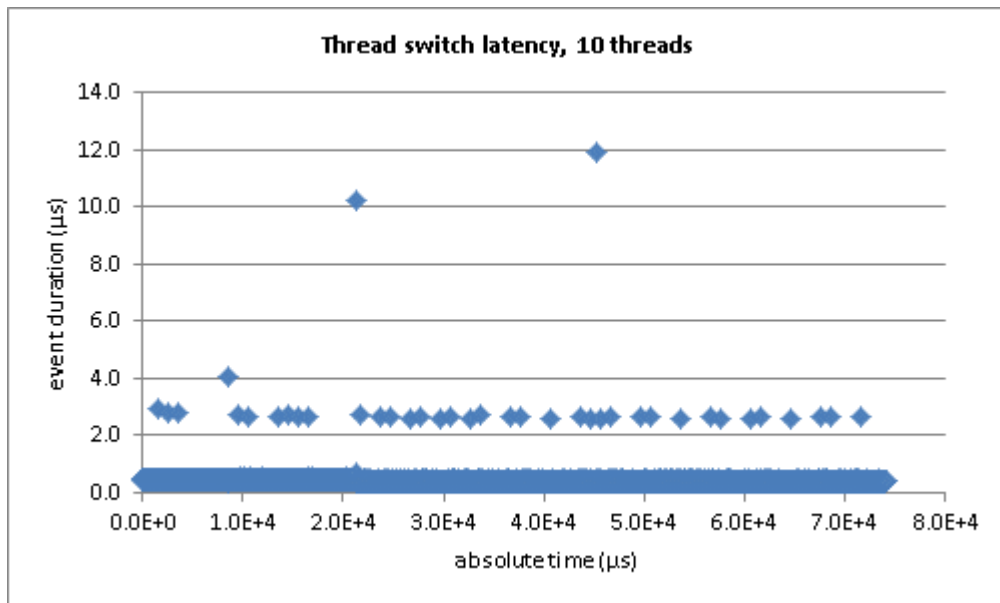
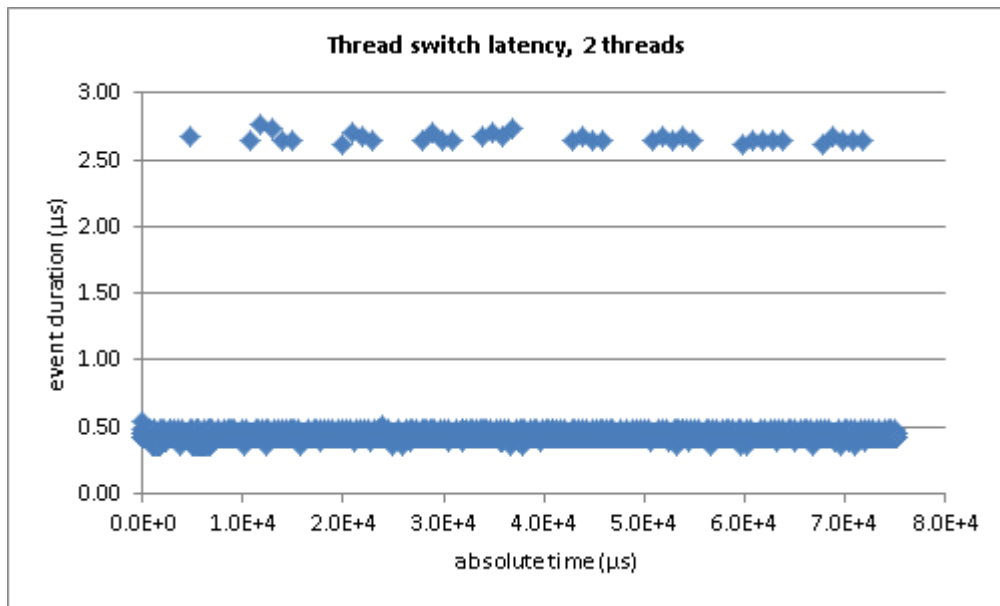
4.3.3.1 Test results

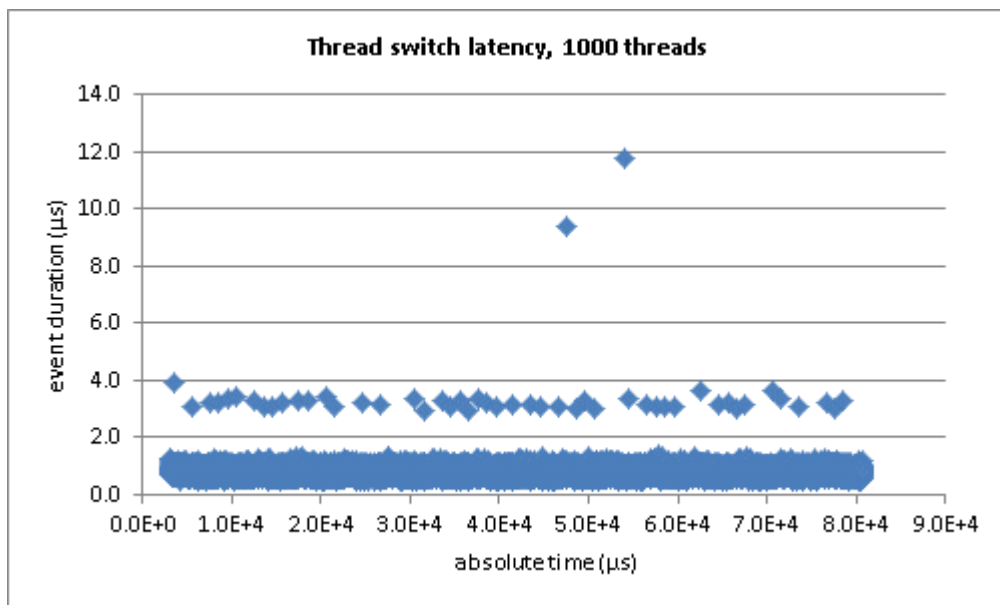
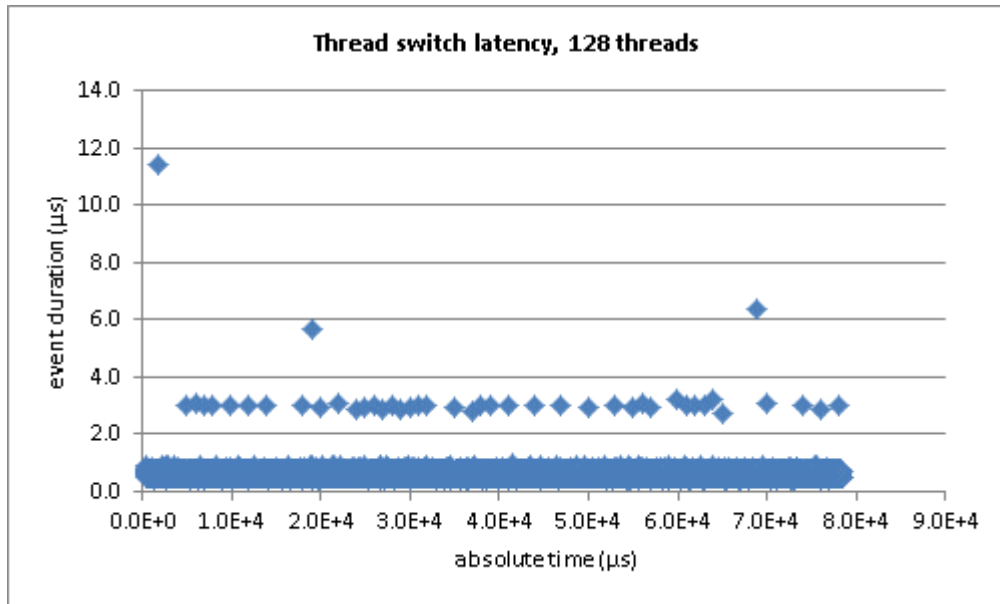
Test	result
Test succeeded	YES

Test	Sample qty	Avg	Max	Min
Thread switch latency, 2 threads	16383	0.4 μ s	2.8 μ s	0.4 μ s
Thread switch latency, 10 threads	16379	0.4 μ s	11.9 μ s	0.4 μ s

Test	Sample qty	Avg	Max	Min
Thread switch latency, 128 threads	16320	0.6 μ s	11.5 μ s	0.4 μ s
Thread switch latency, 1000 threads	15884	0.8 μ s	11.7 μ s	0.5 μ s

4.3.3.2 Diagrams





4.3.4 Thread creation and deletion time (THR-P-NEW)

This test examines the time for creating a thread, and the time for deleting a thread in different scenarios:

- Scenario 1 “never run”: The created thread has a lower priority than the creating thread and is deleted before it has any chance to run. No thread switch occurs in this test.
- Scenario 2 “run and terminate”: The created thread has a higher priority than the creating thread and will be activated. The created thread immediately terminates itself (thread does nothing).

- Scenario 3 “run and block”: The same as the previous scenario (scenario 2: run and terminate), but the created thread does not terminate (it lowers its priority when it is activated).

In the scenarios where the thread actually runs (2, 3), the creation time is the duration from the system call creating the thread to the time when the created thread is activated. For the “never run” scenario, the creation time is the duration of the system call.

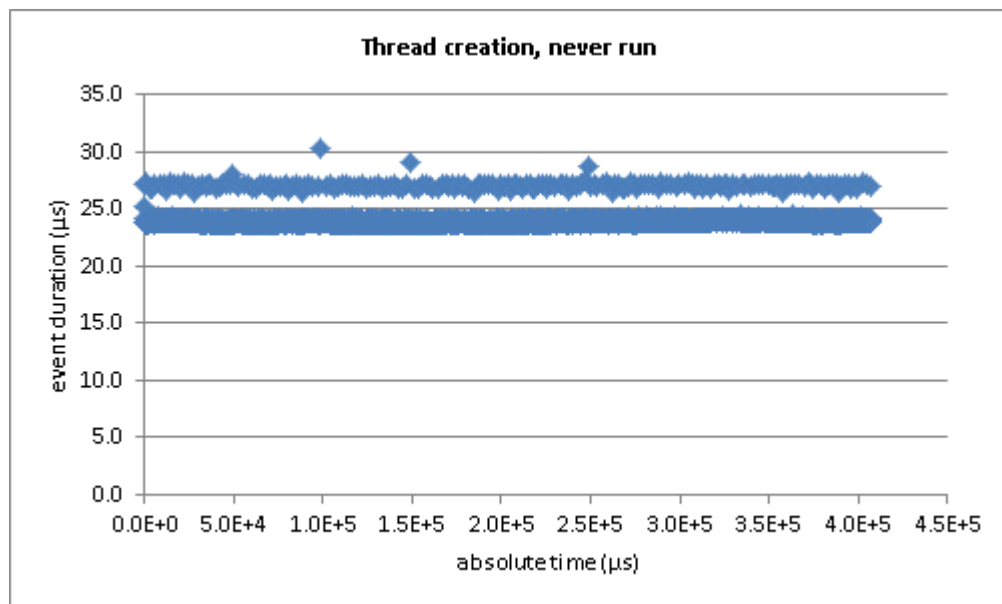
Clearly the performance is much better than on our reference platform (Pentium 200MHz MMX) by a factor 8 or even more.

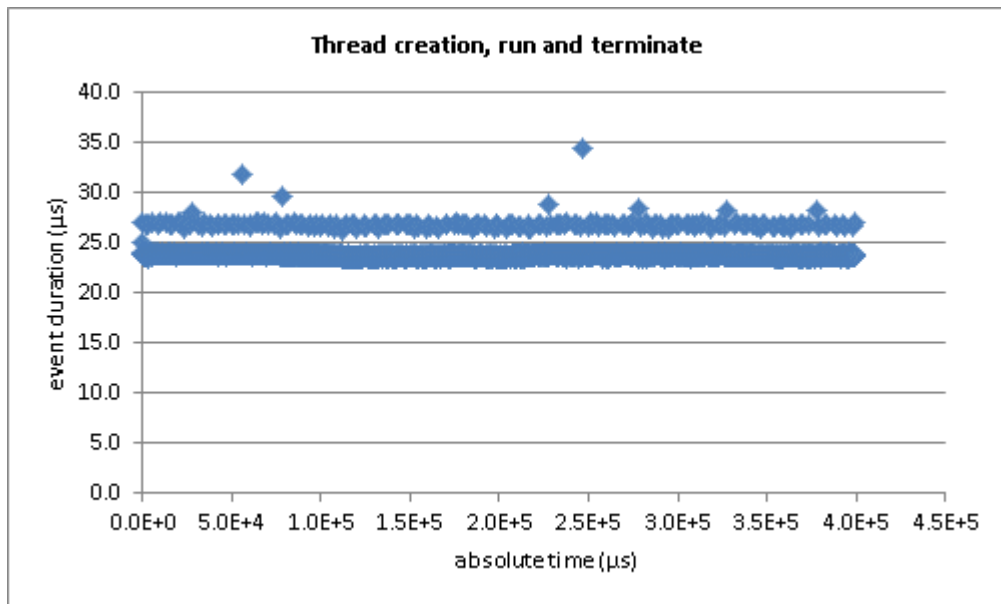
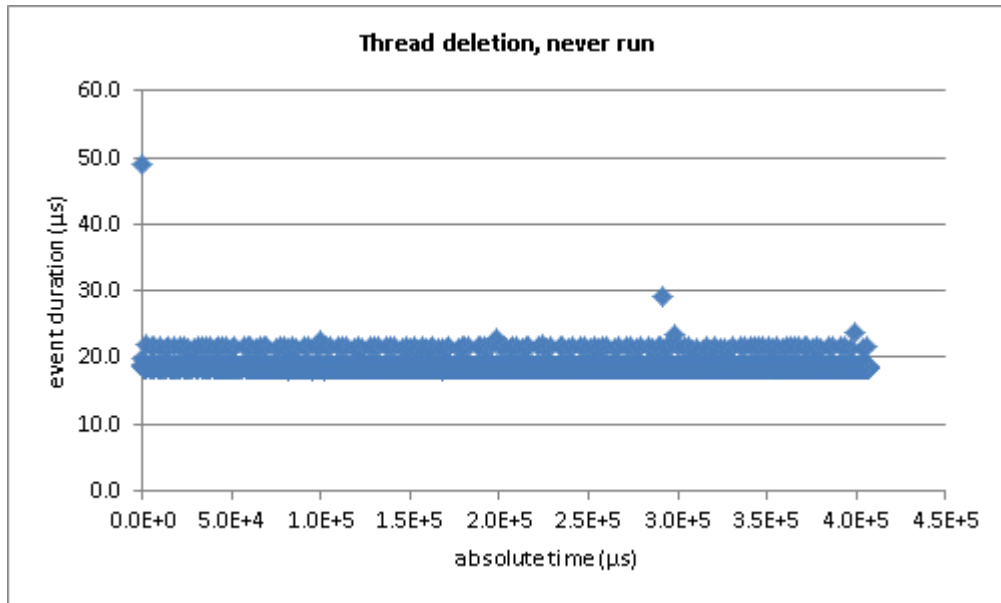
4.3.4.1 Test results

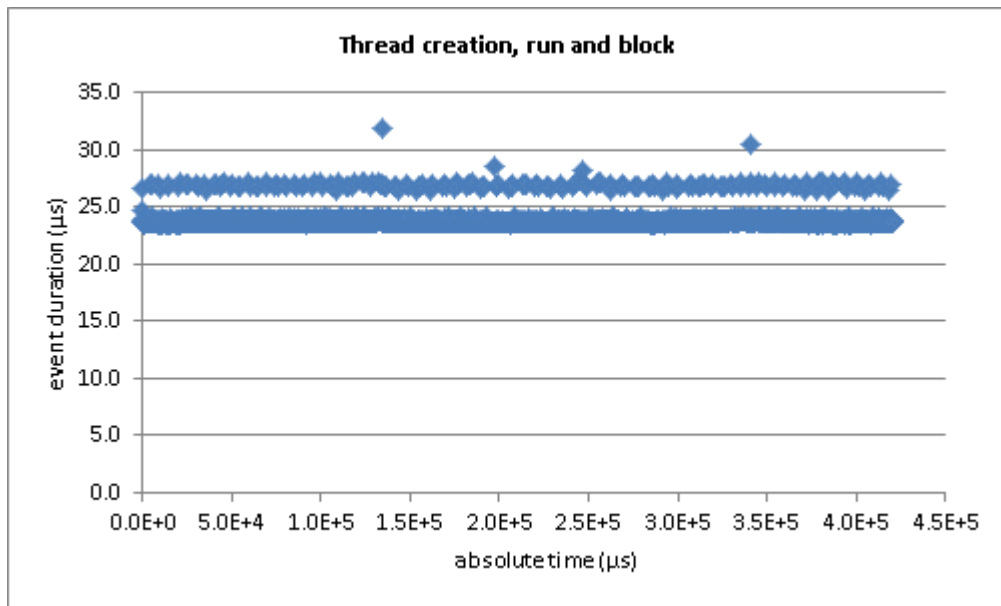
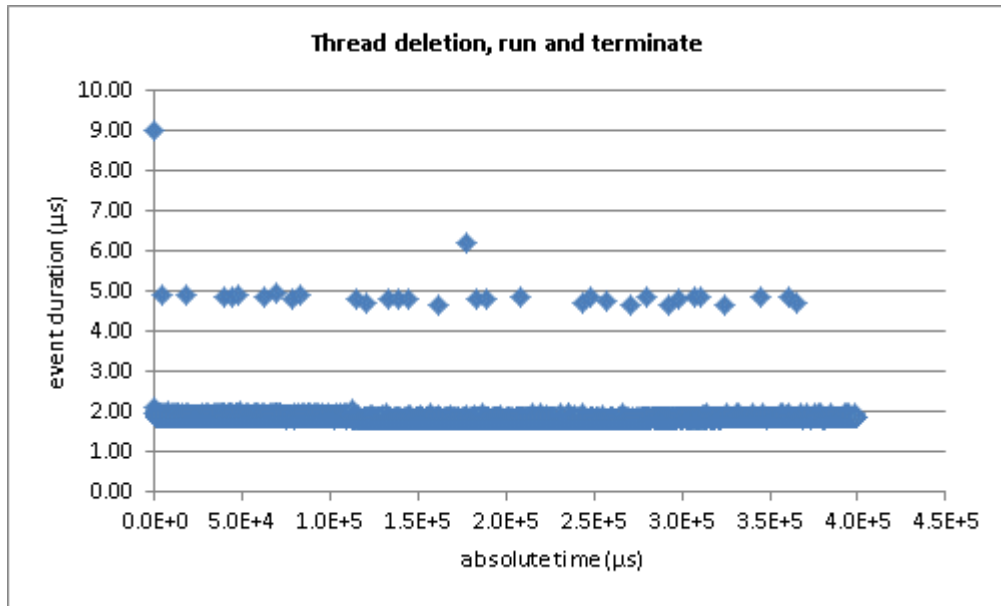
Test	result
Test succeeded	YES

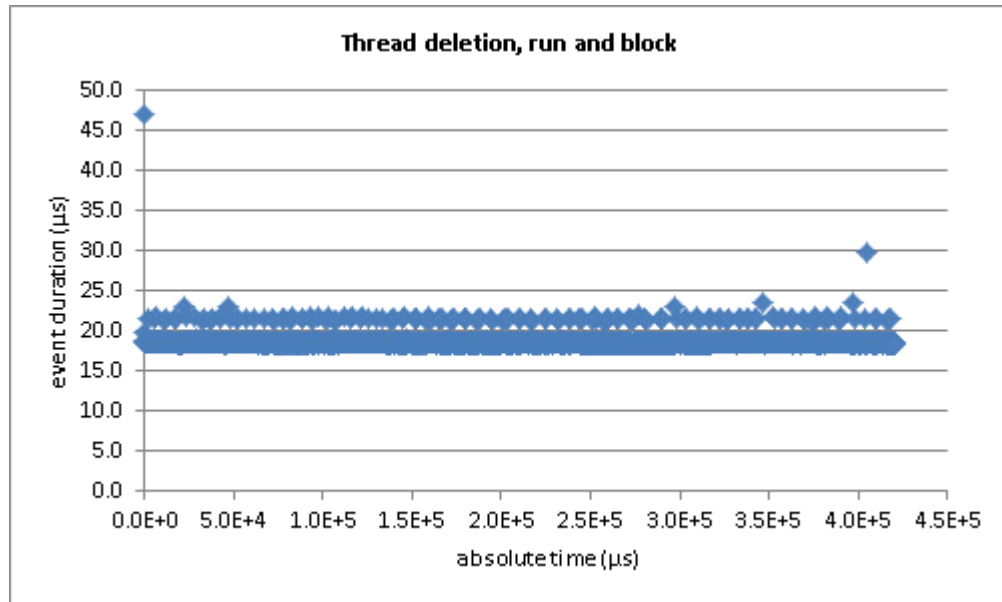
Test	Sample qty	Avg	Max	Min
Thread creation, never run	4096	24.0 μ s	30.3 μ s	23.8 μ s
Thread deletion, never run	4095	18.5 μ s	49.0 μ s	18.2 μ s
Thread creation, run and terminate	7500	23.8 μ s	34.5 μ s	23.4 μ s
Thread deletion, run and terminate	7500	1.9 μ s	9.0 μ s	1.8 μ s
Thread creation, run and block	7500	23.9 μ s	31.9 μ s	23.5 μ s
Thread deletion, run and block	7500	18.6 μ s	47.0 μ s	18.3 μ s

4.3.4.2 Diagrams









4.4 Semaphore tests (SEM)

This test examines the performance and the behaviour of the counting semaphore. The counting semaphore is a system object that can be used to synchronize threads.

4.4.1 Semaphore locking test mechanism (SEM-B-LCK)

In this test, we will experiment if the counting semaphore locking mechanism works as it is expected to do. The $P()$ call should block only when the count is zero. The $V()$ call should increment the semaphore counter. In the case where the semaphore counter is zero, the $V()$ call should cause a rescheduling by the OS: indeed blocked threads may become active.

The semaphore behaves correctly as a protection mechanism.

4.4.1.1 Test results

Test	result
Test succeeded	YES
Maximum semaphore value?	Limited by the "int" type
Rescheduling on free?	OK

4.4.2 Semaphore releasing mechanism (SEM-B-REL)

This test verifies that the highest priority thread being blocked on a semaphore will be released by the release operation. This should be independent of the order of the acquisitions taking place.

QNX passed this test.

4.4.2.1 Test results

Test	result
Test succeeded	YES

4.4.3 Time needed to create and delete a semaphore (SEM-P-NEW)

This test is done to get an insight about the time needed to create a semaphore and the time to delete it. The deletion time is checked in two cases:

- The semaphore is used between the creation and deletion.
- The semaphore is NOT used between the creation and deletion.

Remark that although we do not use “named” semaphores, there seems to be a system call required to create/delete a semaphore.

At start up, there is a peak (Diagrams of section 4.4.3.2), probably by caching.

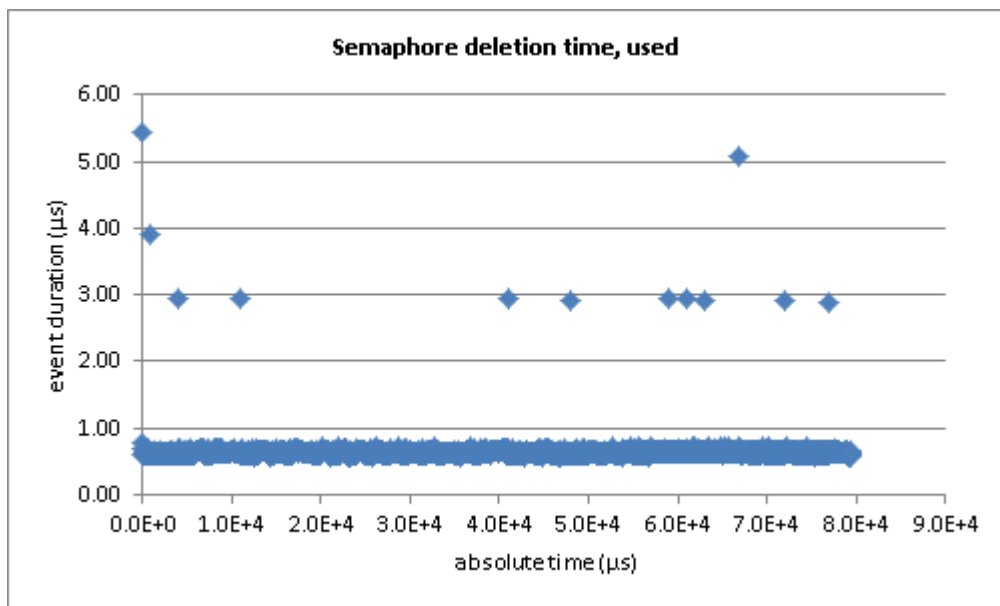
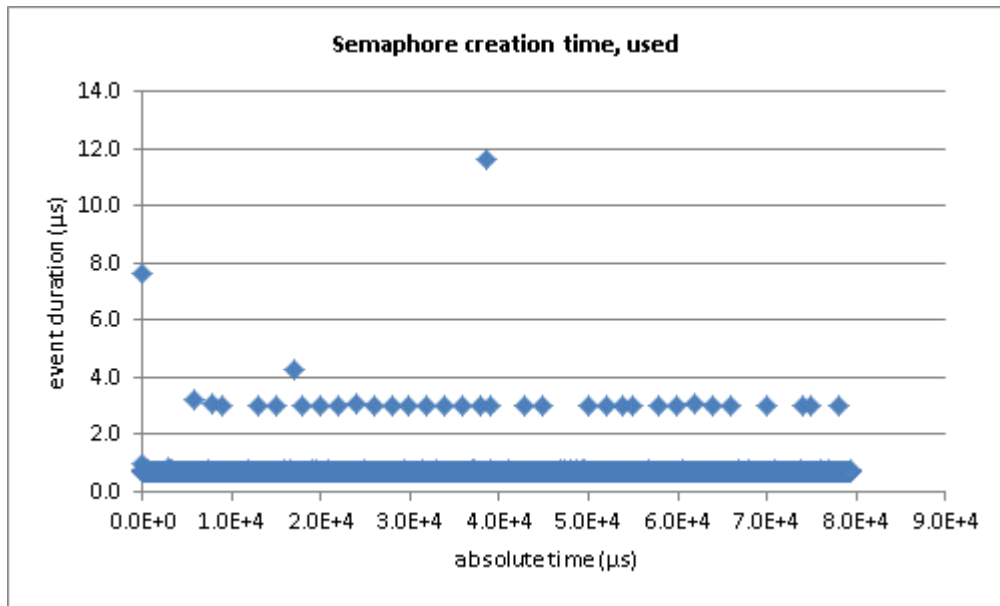
As usual, the clock tick interrupt is seen. You will also find a peak of 11µs which is found back in multiple tests.

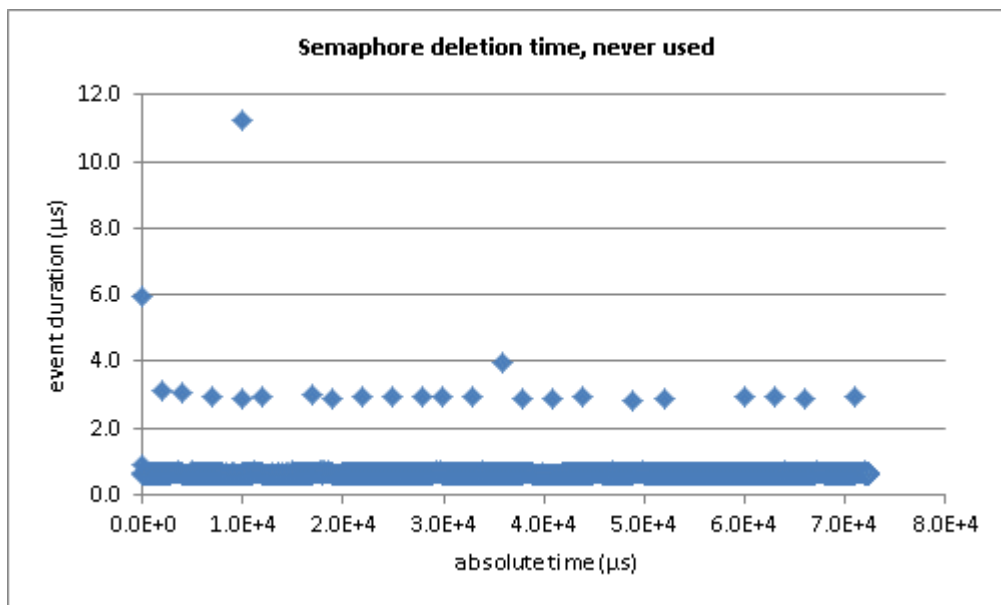
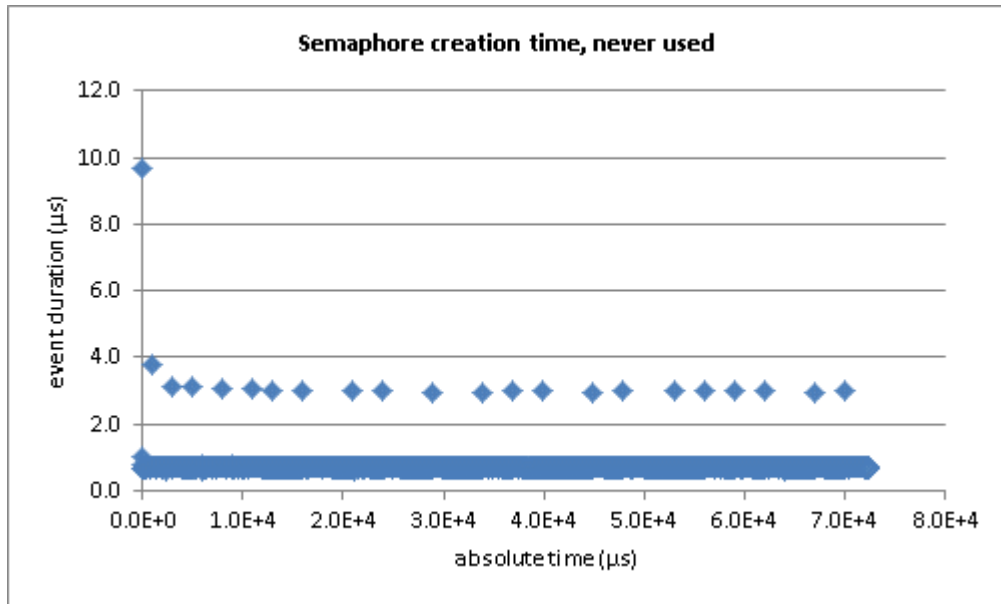
4.4.3.1 Test results

Test	result
Test succeeded	YES

Test	Sample qty	Avg	Max	Min
Semaphore creation time, used	7500	0.7 µs	11.7 µs	0.7 µs
Semaphore deletion time, used	7500	0.6 µs	5.4 µs	0.6 µs
Semaphore creation time, never used	7500	0.7 µs	9.7 µs	0.6 µs
Semaphore deletion time, never used	7500	0.6 µs	12.2 µs	0.6 µs

4.4.3.2 Diagrams





4.4.4 Test acquire-release timings: contention case (SEM-P-ARN)

Here we test the acquisition and release time in the non contention case. As in this test case the semaphore does not neither block nor cause any rescheduling (thread switch), the duration of the call should be short.

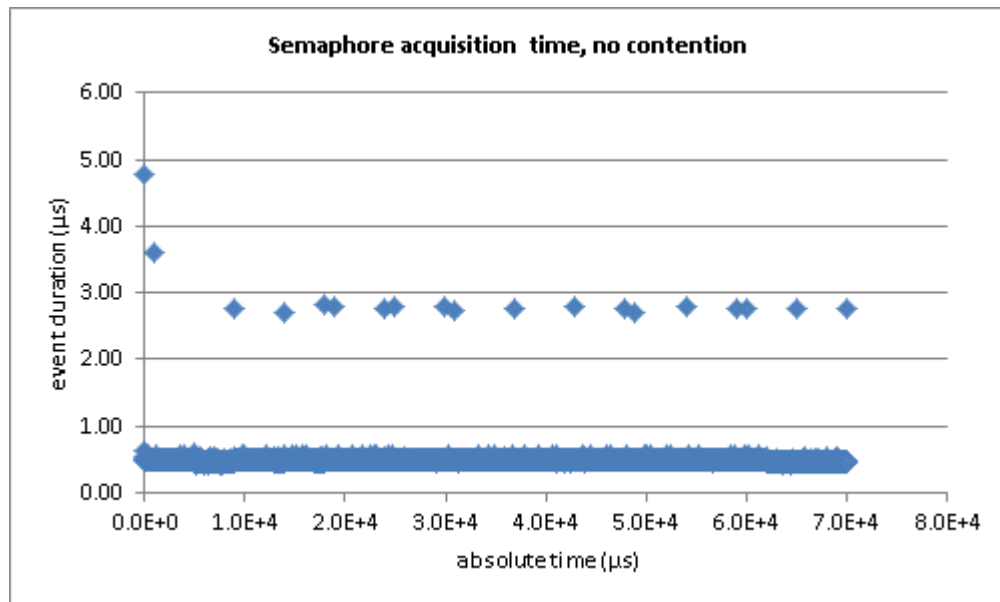
The clock tick is always present.

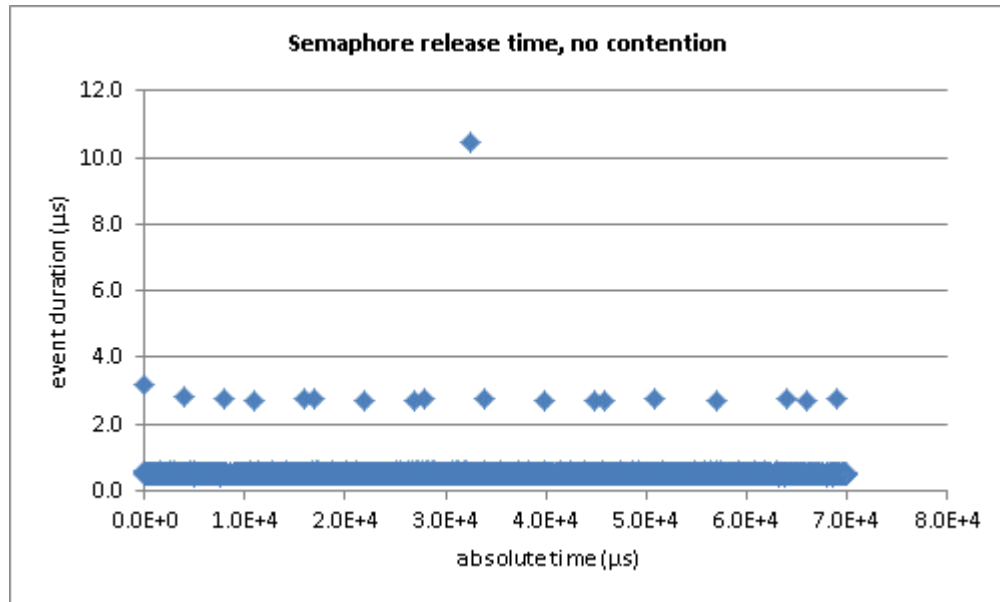
4.4.4.1 Test results

Test	result
Test succeeded	YES

Test	Sample qty	Avg	Max	Min
Semaphore acquisition time, no contention	6993	0.5 μ s	4.8 μ s	0.4 μ s
Semaphore release time, no contention	6993	0.5 μ s	10.4 μ s	0.4 μ s

4.4.4.2 Diagrams





4.4.5 Test acquire-release timings: contention case (SEM-P-ARC)

This is performed to test the time needed to acquire and release a semaphore, depending on the number of threads blocked on the semaphore. It measures the time in the contention case when the acquisition and release system call causes a rescheduling to occur.

The aim of this test is to verify whether the number of blocked threads has an impact on these timings or not. So this will answer the question: “how much time the operating system needs to find out the next thread to schedule”.

As each thread has a different priority, the question is how these pending thread priorities on a semaphore are handled. To have a more clear view on our test, you can take a look on the expanded diagrams during a small time frame (e.g. one test loop):

- We create 128 threads with different priorities. The creating thread has a lower priority than the threads being created.
- When the thread starts execution, it tries to acquire the semaphore; but as it is taken, the thread stops and the kernel switch back to the creating thread. The time from the acquisition try (which fails) until the creating thread is activated again is called here the “acquisition time”. Thus, this time includes the thread switch time.
Thread creation takes some time, so the time between each measurement point is large compared with most other tests.
- After the last thread is created and is blocked on the semaphore, the creating thread starts to release the semaphore and this is the same number of times as there are blocked threads.
- We start timing at the moment the semaphore is released which in turn will activate the pending thread with the highest priority, which will stop the timing (thus again the thread switch time is included).

Now, the most important part of this test is to see if the number of threads pending on a semaphore has an impact on release times. Clearly, it doesn't, so this is good.

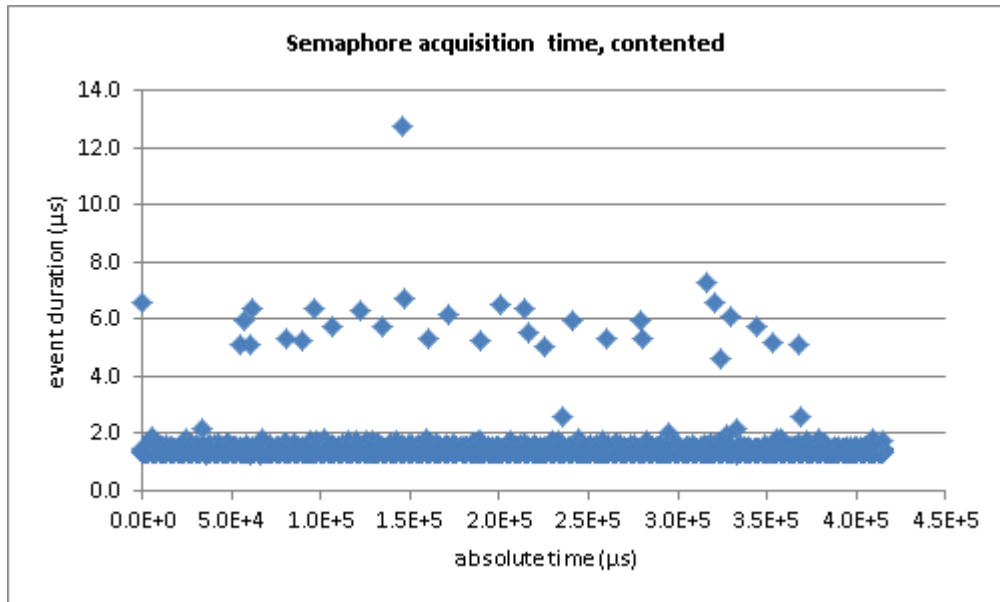
As usual, we find the clock tick back in these diagrams.

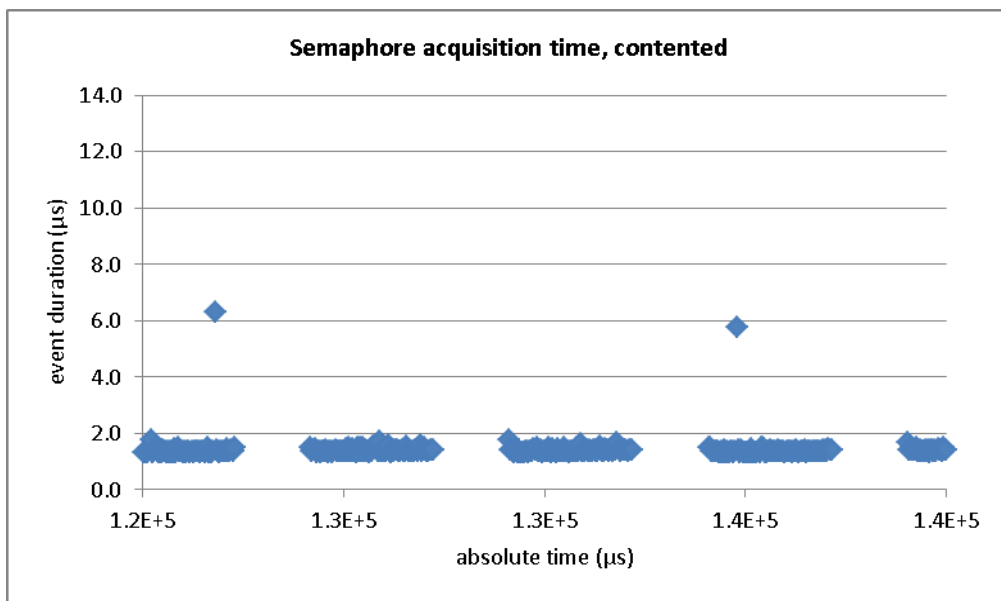
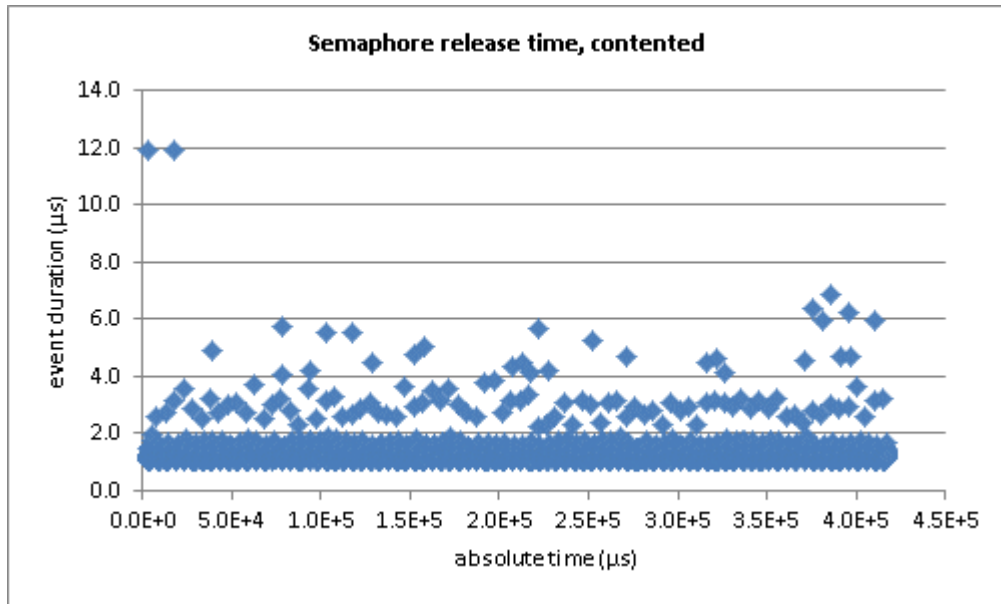
4.4.5.1 Test results

Test	result
Test succeeded	YES
Max number of threads pending	128

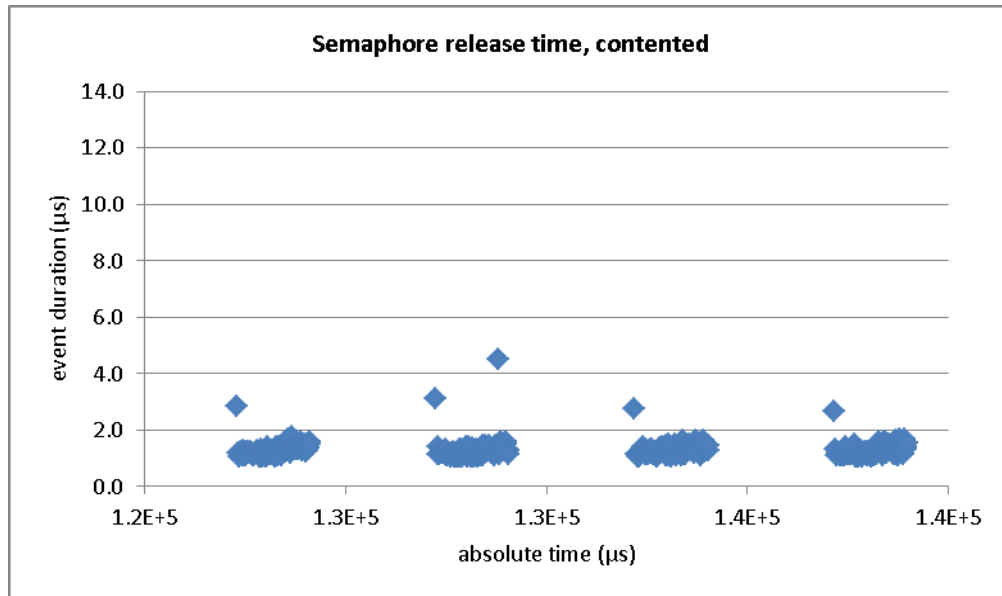
Test	Sample qty	Avg	Max	Min
Semaphore acquisition time, contented	1021	1.4 μ s	12.8 μ s	1.3 μ s
Semaphore release time, contented	1021	1.3 μ s	12.0 μ s	1.0 μ s

4.4.5.2 Diagrams





Detail of acquisition test



Detail of release part: done in same test run as diagram above

4.5 Mutex tests (MUT)

Here we are going to test the performance and behaviour of the mutual exclusive semaphore.

Although the mutual exclusive semaphore (further called mutex) is mostly explained as being the same as a counting semaphore where the count is one, this is not true. A mutex has a totally different behaviour than semaphores. Mutexes have the concept of "lock owner", and thus they can be used for preventing priority inversions. This is something that cannot be done by semaphores. Therefore it is a bad idea to use semaphores as a critical section protection mechanism.

In scope of the framework, this test will look into detail of a mutex system object that avoids priority inversion.

4.5.1 Priority inversion avoidance mechanism (MUT-B-ARC)

This test will determine if the system call under testing prevents the priority inversion case. Therefore the test will artificially create a priority inversion.

Priority inversion behaves as expected.

4.5.1.1 Test results

Test	result
Priority inversion avoidance system call present	Yes
System call used	pthread_mutex_lock
Test succeeded	YES

Test	result
Priority inversion avoided	YES
Mechanism used if any?	pthread_mutexattr_setprotocol: PTHREAD_PRIO_INHERIT

4.5.2 Mutex acquire-release timings: contention case (MUT-P-ARC)

This is the same test as above, but performed in a loop. In this case, the time is measured to acquire and release the mutex in the priority inversion case.

Remark that the acquisition enforces a thread switch. The acquiring thread is blocked and the one having the lock is released. The time is measured from the request for the mutex acquisition to the lower priority thread having the lock being activated.

Before the release, an intermediate priority level thread is activated (between the low priority one having the lock and the high priority one asking the lock). Due to the priority inheritance, this thread does not start to run (the low priority thread having the lock inherited the high priority of the thread asking the lock).

The release time is measured from the release call to the thread requesting the mutex being activated, so it also includes a thread switch.

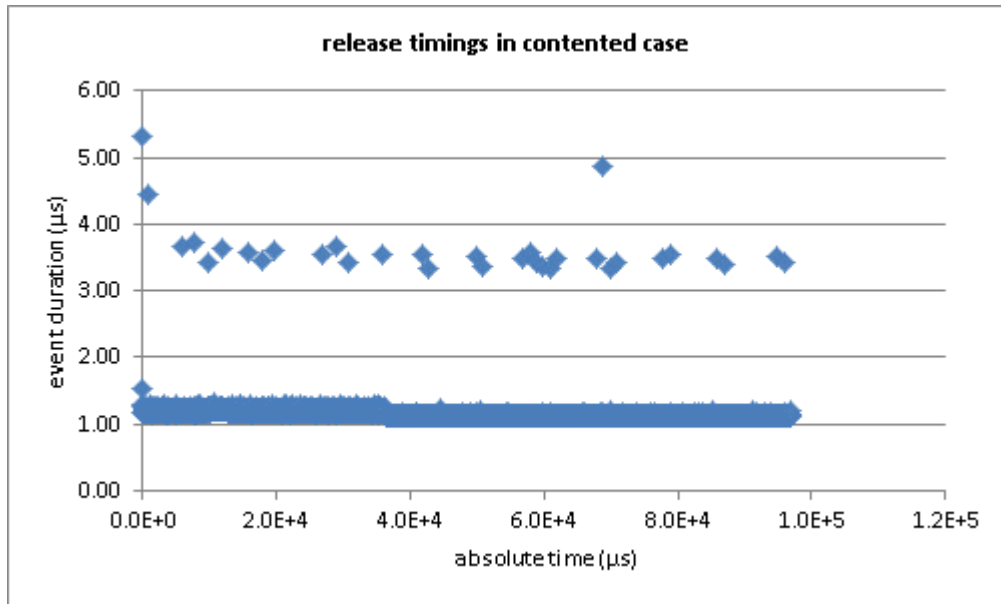
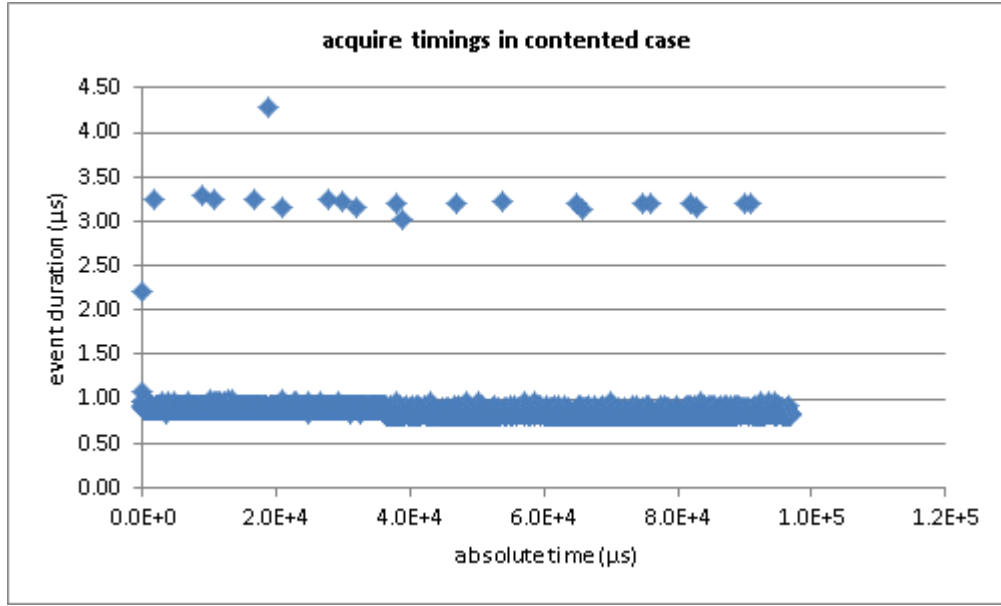
As usual, the clock ticks can be clearly seen. Also the one 12 μ s spike shows up again.

4.5.2.1 Test results

Test	result
Test succeeded	Yes

Test	Sample qty	Avg	Max	Min
Mutex acquisition time, contention	7500	0.9 μ s	4.2 μ s	0.8 μ s
Mutex release time, contention	7500	1.2 μ s	5.3 μ s	1.1 μ s

4.5.2.2 Diagrams:



4.5.3 Mutex acquire-release timings: no-contention case (MUT-P-ARN)

This test measures the overhead of using a lock when it is not locked by another thread. Good designed software will use non-contended locks most of the time and only in some rare cases the lock will be taken by another thread.

Therefore, it is important that the non-contention case should be fast. Remark that this is only possible if the CPU supports some type of atomic instruction, so that no system call is needed when no contention is detected. Clearly this is the case for QNX; probably no system call is issued. Anyhow the time it requires starts to be too small to measure.

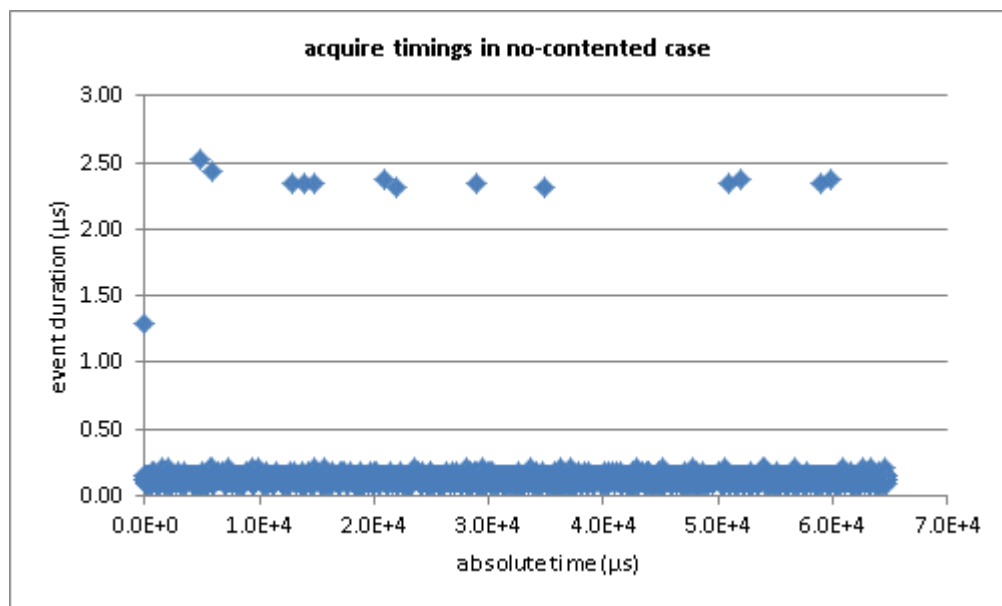
As in all diagrams, the clock tick shows up again.

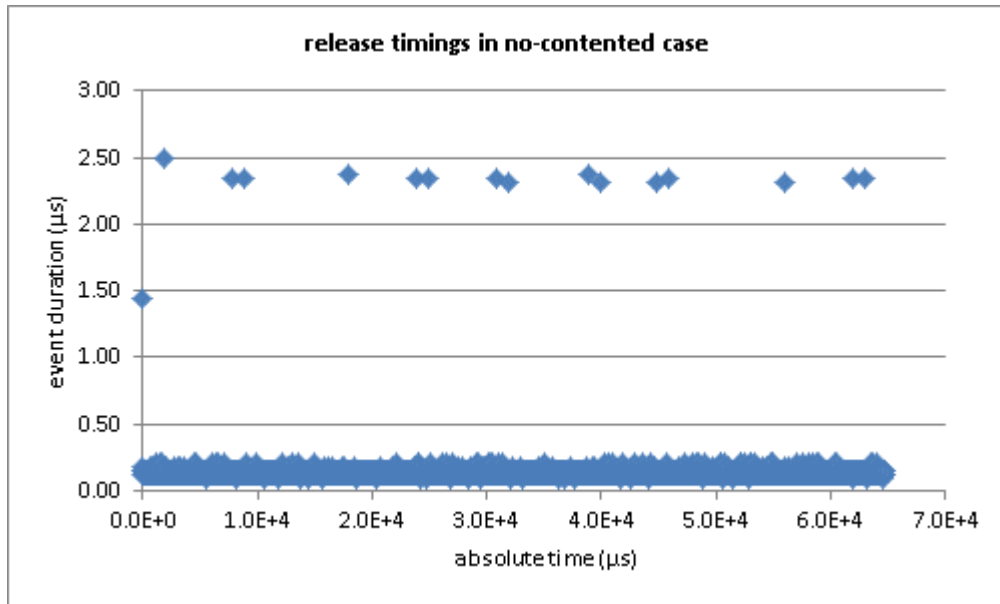
4.5.3.1 Test results

Test	result
Test succeeded	Yes

Test	Sample qty	Avg	Max	Min
Semaphore acquisition time, no contention	7500	0.1 μ s	2.5 μ s	<0.1 μ s
Semaphore release time, no contention	7500	0.1 μ s	2.5 μ s	<0.1 μ s

4.5.3.2 Diagrams:





4.6 Interrupt tests (IRQ)

The performance of the interrupt handling in the operating system and hardware is tested here.

In a real-time system, interrupt handling is a major part of the system. Indeed, such systems are typically event driven.

For these tests, our standard tracing system is adapted. Interrupts are generated by a plugged-in PCI related card (can be PMC/PCI or CPCI). This card has a complete independent processor on board, with custom-made software. As such, we can guarantee that an independent interrupt source is not synchronised in any way with the platform under test.

4.6.1 Interrupt latency (IRQ_P_LAT)

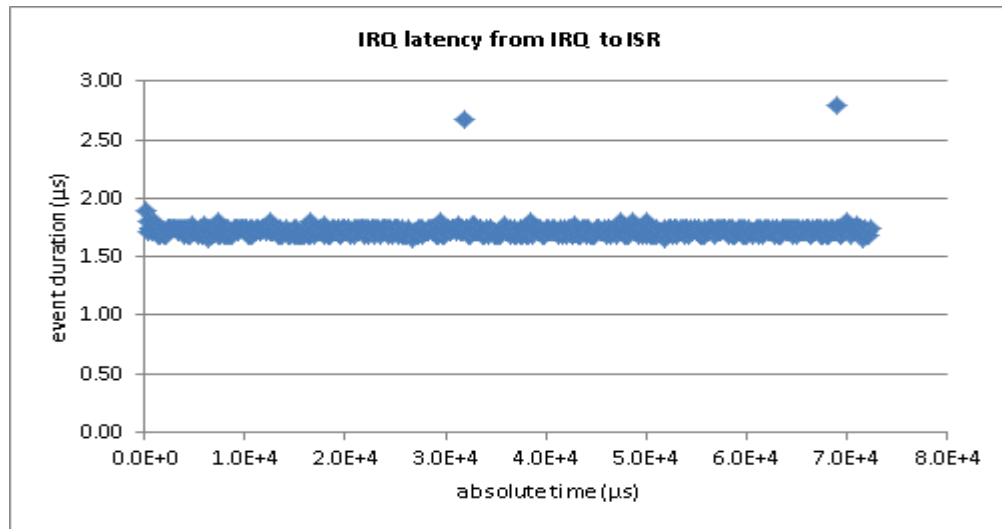
This test measures the time it takes to switch from a running thread to an interrupt handler. The time is measured from the moment the running thread is interrupted until the interrupt routine. Thus it does not include the hardware latency.

Although in all tests the Atom is about 5 to 10 times faster than the Pentium MMX 200MHz, this is not the case for interrupt latency. There is always some hardware latency that did not improve that much as the processor speed.

4.6.1.1 Test results

Test	Sample qty	Avg	Max	Min
Interrupt dispatch latency	576	1.7 µs	2.8 µs	1.7 µs

4.6.1.2 Diagrams



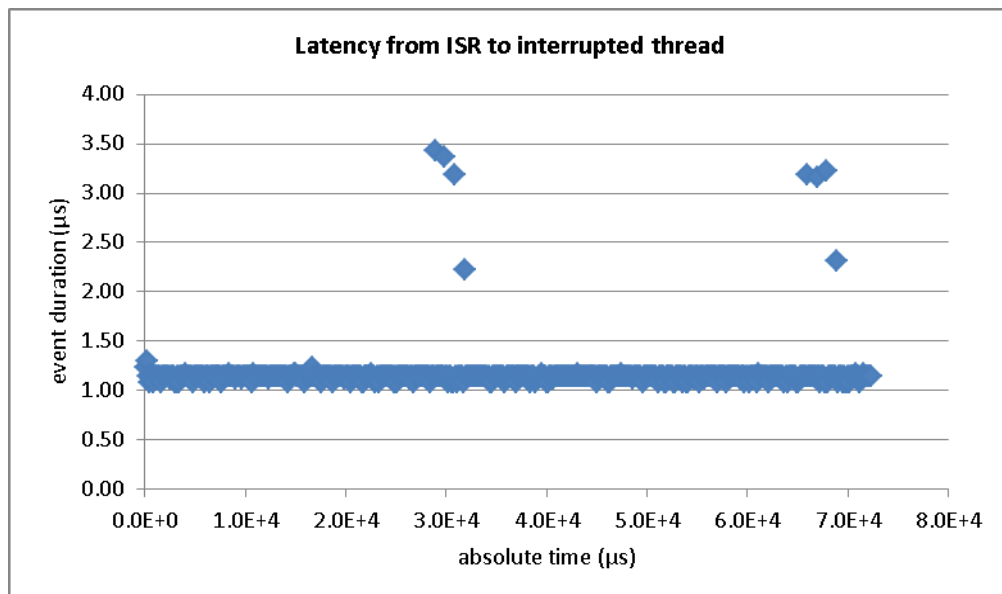
4.6.2 Interrupt dispatch latency (IRQ_P_DLT)

This test measures the time it takes to switch from the interrupt handler back to the interrupted thread. Clock tick is clearly seen again.

4.6.2.1 Test results

Test	Sample qty	Avg	Max	Min
Dispatch latency from interrupt handler	7384	1.2 µs	3.4 µs	1.1 µs

4.6.2.2 Diagrams



4.6.3 Interrupt to thread latency (IRQ_P_TLT)

4.6.3.1 Test results

This test measures the time it takes to switch from the interrupt handler to the thread that is activated from the interrupt handler.

This test is done by allowing the interrupt handler to emit an event which releases a blocked thread. This blocking thread has the highest priority in the system. There is also a low priority thread looping. So the measurement takes the time from the interrupt handler to the blocked thread (as a consequence this includes a thread switch).

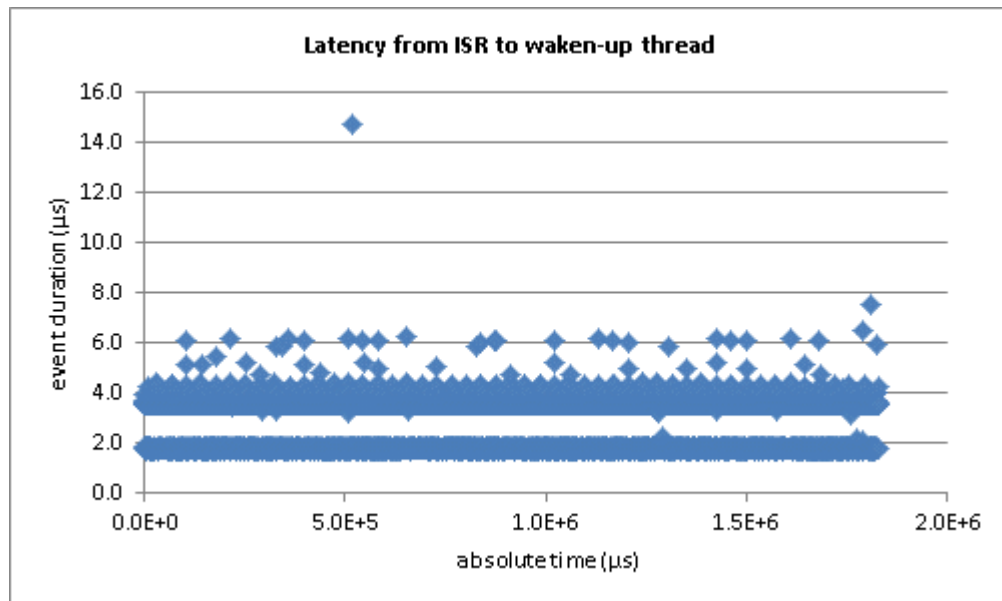
Remark that due to the usage of I/O mapping and thus the slow x86 I/O instructions (pci bus access), we detect some stepping in the results: an idle thread continuously writes I/O traces on the bus which take around 2us. So it is only once on the 2us that the interrupt can occur (it can not interrupt the pending I/O cycle on the PCI bus). This is why you see the 2us step in the zoomed diagram extract.

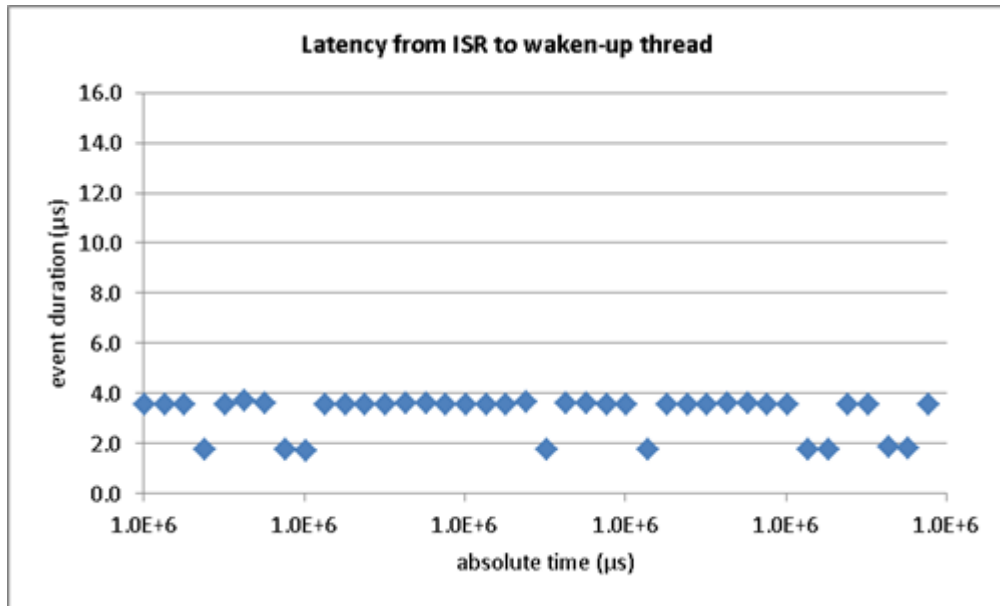
For sustained interrupt tests, we do not use this background thread.

4.6.3.2 Test results

Test	Sample qty	Avg	Max	Min
Latency from ISR to waken-up thread	10920	3.2 μ s	14.7 μ s	1.7 μ s

4.6.3.3 Diagrams





Zoomed in extract of previous diagram

4.6.4 Maximum sustained interrupt frequency (IRQ_S_SUS)

This test measures the probability that an interrupt is missed. Is the interrupt handling duration stable and predictable?

The test is done on three levels:

- 1000 interrupts, initial phase: a fast test just to see where we have to start searching.
- 1 000 000 interrupts, second phase based on the results from the first phase. This test still takes less than a minute and gives already accurate results.
- 1 000 000 000 interrupts, takes more than 24 hours: to verify stability, therefore we cannot run a lot of tests, especially when it comes to large interrupt latencies.

It is a good exercise to compare these results on the Atom with the ones on the Pentium 200 MHz MMX. Although the platform is about 5 to 10 times faster in general, the interrupt latency is only a factor 2 better:

- A lot of latencies are caused by hardware latencies as well; this is not reduced by a faster processor.
- Faster processors have deeper pipelines that need to be cleaned up on an interrupt which takes also an extra time.

Further, we noticed an unexpected latency peak of around 11µs on different locations during the test. This can be also found in the longer sustained interrupt frequency test, where we need around 15µs in order not to lose any interrupt (which is about the half compared with the Pentium MMX 200MHz).

It is unclear for us where this extra delay comes from. We did the tests with only a basic system (no network, no USB ...) so that besides the clock interrupt (and the serial line, but this is not used during the test run) we do not expect any interrupt. But still the peak is in.

4.6.4.1 Test results

Interrupt period	#interrupts generated	#interrupts serviced	#interrupts lost
2.6 μ s	1 000	887	113
3.1 μ s	1 000	994	6
3.4 μ s	1 000	1 000	0
12 μ s	1 000 000	999 981	19
12.5 μ s	1 000 000	1 000 000	0
12.5 μ s	1 000 000 000	999 999 816	184
15 μ s	1 000 000 000	1 000 000 000	0

4.7 Memory tests

This test examines the memory leaks of OS.

4.7.1 Memory leak test (MEM_B_LEK)

This test continuously create/remove objects in the operating system (threads, semaphores, mutexes ...).

Test	result
Test succeeded	YES
Test duration (how long we let the endless loop run)	>10h
Number of main test loops done	> 50 000

5 Appendix A: Vendor comments

All vendor comments were integrated within the documents as there were no disagreements.

6 Appendix B: Acronyms

Acronym	Explanation
API	Application Programmers Interface: calls used to call code from a library or system.
BSP	Board Support Package: all code and device drivers to get the OS running on a certain board
DSP	Digital Signal Processor
FIFO	First In First Out: a queuing rule
GPOS	General Purpose Operating System
GUI	Graphical User Interface
IDE	Integrated Development Environment (GUI tool used to develop and debug applications)
IRQ	Interrupt Request
ISR	Interrupt Servicing Routine
MMU	Memory Management Unit
OS	Operating System
PCI	Peripheral Component Interconnect: bus to connect devices, used in all PCs!
PIC	Programmable Interrupt Controller
PMC	PCI Mezzanine Card
PrPMC	Processor PMC: a PMC with the processor
RTOS	Real-Time Operating System
SDK	Software Development Kit
SoC	System on a Chip