IPP-Report

IPP Max-Planck-Institut für Plasmaphysik

Helmut Niedermeyer

Jitter Measurements on PCs with a Real Time Operating System

IPP 11/3

Oktober 1998

Max-Planck-Institut für Plasmaphysik

Jitter Measurements on PCs with a Real Time Operating System

H. Niedermeyer

Table of contents

Introduction

Real time systems are often defined by their „deterministic“ behaviour or predictable maximum response time to unexpected events. In order to get an impression of the usefulness of this concept for systems including high performance PCs measurements were performed on two different PCs running under VxWorks as a target normally connected to a host PC via TCP/IP. Triggering and timing measurements were performed with an add-on board on the target PC.

It has to be mentioned, that the tools used for the measurement, neither software nor hardware, can be considered as proven and the results for this reason should be considered as preliminary. The purpose of this report is to stimulate discussion and similar measurements.

Setup

Method of measurement

Triggering and delay measurements were performed with a bc635pci board from Datum Inc. inserted into the target PC. Four different boards of this type were used. A standard PCI version was used with a standard PC and a Compact PCI version was used with a Compact PCI computer (see below). Each of the two board types was used in an earlier version and a later, improved version. The earlier version had a few bugs not obviously interfering with the measurements. Unfortunately no other independent high precision timing system was available.

The bc635 board can generate periodic interrupts and houses a real time clock with a resolution of 100 ns and two independent registers to store the time. A task to be tested is triggered periodically by the Interrupt Service Routine (ISR). At the beginning of the process to be timed the time is transferred into the first of the registers (“time register”) by a write operation to a special memory address. At the end of the process the time is transferred into the second register (“event register”). For measuring the period of cycles registers 1 and 2 take the time alternately. Giving a binary semaphore indicates that data are available. A low priority task triggered by this semaphore reads the values from the registers into RAM, computes the delay and sends the results to another low priority task through a message queue for displaying the results on screen and / or storing them on a RAM disk in ASCII format. Visual output can be reduced to one character per value just to indicate activity or it can be turned off. Output was normally displayed through „Virtual IO“ in the target server window on the host.

The following lines of C code in the ISR demonstrate the time taking for computing the delay between successive ISR calls.

```
if (flag)    BC635_TIMEREQ;     /* Store time in "Time"-register of bc635 */
else         BC635_EVENTREQ;    /* Store time in "Event"-register of bc635 */
flag = !flag;
semGive(bc635_semB_period);
```

`BC635_TIMEREQ` and `BC635_EVENTREQ` are macros triggering the time transfer into the registers:

```
#define      BC635_TIMEREQ      (*bc635_timereq = 0x00)
#define      BC635_EVENTREQ     (*bc635_eventre = 0x00)
```

Operations of this kind are the only ones added to the task or the ISR to be investigated. The perturbation caused by the measurement therefore should be low. However, it turns out (see

below) that the low priority tasks necessary to process and store the data can cause appreciable jitter even to the start of ISRs.

Target systems

For most of the measurements a general purpose PC with a 200 MHz Pentium MMX processor on an ASUS TXP4-X motherboard was used. It was equipped with 64 MB SDRAM, a floppy, IDE Hard disk and CD ROM drive. In addition to the bc635 timing board the network interface Etherlink III ISA board was active. Other boards also installed in the PC like a sound board were not active. The bc635 used IRQ 9, the Ethernet board used IRQ 11. Other possible sources of interrupts like keyboard, mouse and serial ports were not used during the measurements and did not produce interrupts. Standard BIOS settings were used, the processor cache and the L2 cache were normally enabled.

A few measurements were performed with a Compact PCI system CC5 from OR Industrial Computers. This PC was equipped with a 166 MHz Pentium MMX processor, 32 MB DRAM, and an onboard ethernet interface. A Compact PCI version of the bc635 timing board was installed.

VxWorks version 5.3.1 with BSP version 1.1/1 (modified for the bc635pci hardware) was used. In most cases instrumentation for WindView was installed but normally not used. The following tasks were normally installed in the system during the tests.

Note that a priority of 0 is the highest, 255 the lowest possible.

Name	Priority	Comment
tEvtTask	0	event buffer task (for WindView)
tLogTask	0	used by VxWorks modules to log system messages
tExcTask	0	supports the VxWorks exception handling package
tWdbTask	3	services requests from the Tornado target server
tNetTask	50	handles task-level functions required by the VxWorks network
tFtpdTask	55	FTP server
u0	100	task to be tested (priority varied between 2 and 100)
takejob	200	reads time from bc635 registers and computes difference
recordjob	201	displays and record results depending on options

Host system

A general purpose PC running Tornado 1.0.1 under Windows NT 4.0 served as host. Normally the target server and the shell were running during the measurements, in one case also WindView was running.

Data evaluation

After a predetermined number of measurements had been taken the file on the RAM disk was closed and manually transferred to the host using FTP. With Mathematica a simple statistical evaluation (minimum value, maximum value, range, mean, standard deviation) was performed and a histogram was drawn. In cases of measuring periods each second value was negative due to the method of data taking. In this case the absolute value of all numbers was used for evaluation.

Note: The histograms show the logarithm of the number of samples in each of the 51 bins covering the full range between minimum and maximum value of the delays. For this reason the time scale varies between histograms. The vertical scale is the same in all histograms. However, different numbers of samples were used for different histograms. To the eye very rare samples appear to be more frequent, if the total number of samples is higher, because their total number is higher!

The list of figures below the histograms shows the number of samples in each of the bins.

Results

Execution time of a task

The time required for execution of the following loop in a task with priority 100 was measured. "loop" was 10000 (fig. 1) and 100000 (fig. 2) respectively. Frequency of the task trigger was 10 Hz, activity was indicated displaying single characters by "virtual IO" on the screen of the host PC.

```
FOREVER
        {
        semTake(bc635_semB_period, WAIT_FOREVER);

        BC635_TIMEREQ;              /* Store time in "Time"-register of bc635 */

                for (i=0;i<loop;i++)
                {
                s=s++;
                s=s--;
                }

        BC635_EVENTREQ;             /* Store time in "Event"-register of bc635 */
        semGive(semRecord);         /* Announce that data are ready */
        }
```

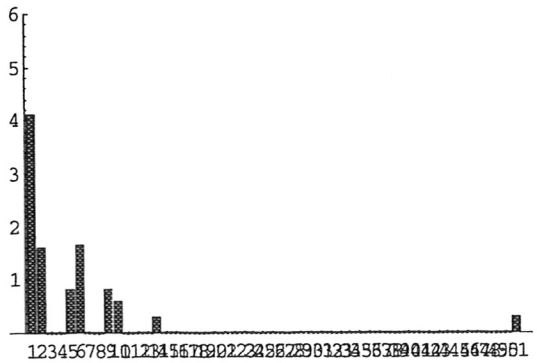
The elapsed time of 150 µs or 1.5 ms, respectively, is roughly proportional to the number of the loop variable. The standard deviation is just a few microseconds. This gives a wrong impression of the problems involved: a few runs suffer a delay ranging up to a value of 300 µs. Note that the number of samples for the longer loop is nearly 4 times higher than for the short loop. The relative frequency of long delays is about the same.

In the third example (fig. 3) everything not necessary has been removed from the system and the program to be tested had the highest priority possible. Interrupts from the system clock and priority = 0 tasks remained active. There are less samples with medium delay in the order of 50 – 100 µs, but still a number of samples with high delay.

Number of data: 13964

min = 0.000148 max = 0.0004443 interval = 0.0002963

mean = 0.000150264 standarddeviation = 3.38189×10^{-6}



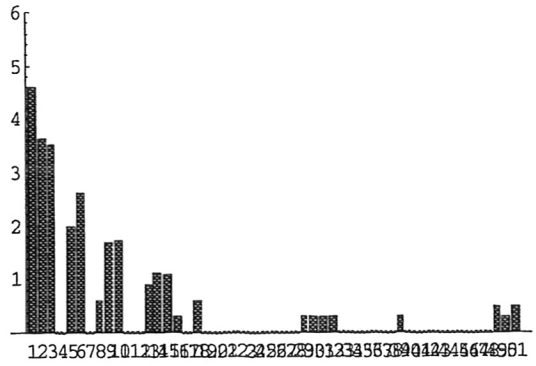
{13860, 42, 0, 0, 6, 45, 0, 0, 6, 3, 0, 0, 0, 1, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1}

Fig. 1 loop = 10 000, priority = 100, "virtual output" of activity indicator, instrumentation for WindView installed

Number of data: 50000

min = 0.0014948 max = 0.0017948 interval = 0.0003

mean = 0.00149851 standarddeviation = 6.08362×10^{-6}



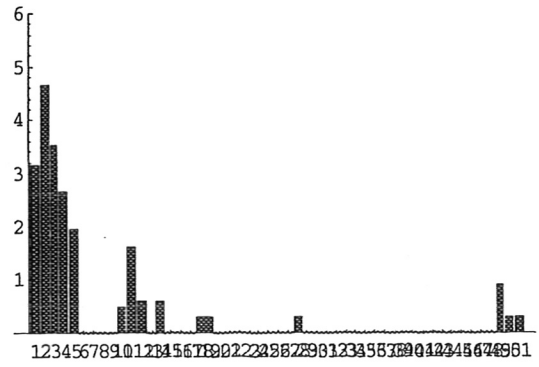
{41504, 4422, 3407, 0, 94, 422, 0, 3, 48, 55, 0, 0, 7, 13,
11, 1, 0, 3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0,
0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 1, 2}

Fig. 2 loop = 100 000, priority = 100, "virtual output" of activity indicator, instrumentation for WindView installed

Number of data: 50000

min = 0.0014907 max = 0.001752 interval = 0.0002613

mean = 0.00149713 standarddeviation = 4.2353×10^{-6}



{1389, 44623, 3380, 455, 93, 0, 0, 0, 0, 2, 40, 3, 0,
3, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 7, 1, 1}

Fig. 3 loop = 100 000, priority = 2, display of activity indicator on target screen, instrumentation for WindView not installed

Start of Interrupt Service Routine

The method applied does not allow to measure the latency of the start of ISRs relative to interrupt signals on the bus. Instead we measure the delay between two successive starts of an ISR when the interrupt signal is generated periodically. As long as large latencies are rare a large positive difference of the time measured and the average time approximates the latency for this particular interrupt. Negative differences occur if the ISR runs in time after a delayed run. The distribution should be symmetric to the period length as long as delayed runs alternate with undelayed runs. It has been verified by inspecting data for a few cases with long delay that these events are isolated.

In the first example (fig. 4) interrupts were generated with a frequency of 1 kHz. The activity indicator was displayed on the host screen through "virtual IO". Instrumentation for WindView was installed but not used. The loop was removed from the task triggered by the ISR and running with a priority of 100.

In the second example (fig. 5) virtual IO was still installed but there was no output generated. The frequency was reduced to 500 Hz.

In the third example (fig. 6) the frequency was further reduced to 100 Hz. However, in this case the task triggered had a priority of 2 and executed the loop 100 000 times. As shown above this takes 1.5 ms in the average and a maximum of 1.8 ms. Instrumentation for WindView was not installed, the indicator was displayed on the target screen. The system in this case had to perform no unnecessary activities (except perhaps the Wdb task).

The relatively large number in the order of 6% of samples with medium delay times up to about 60 µs in the first case must have been caused by low priority activities regularly going on when an interrupt occurs. In the second case there were less of these activities (no screen output) and more time for finishing the job before a new interrupt occurred. In the third case there was plenty of time to finish all jobs caused by the user.

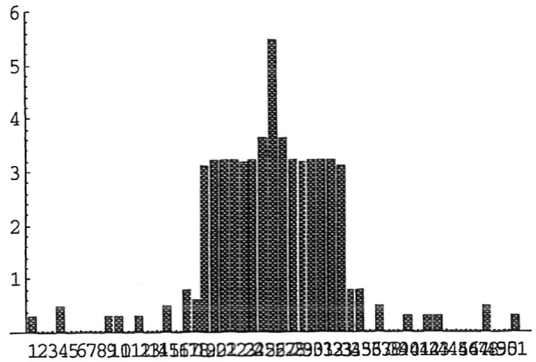
In all three cases there is a maximum delay of about 200 –250 µs. It has a probability in the order of 10^{-5} and must be caused by a system activity. If it is a high priority process taking this time we estimate from the probability that this task runs about every 20 seconds in the average.

The fourth example (fig. 7) compares to the first example (fig. 4). However, there was no output generated. This shows that the graphical output (together with related network activities) has caused the spreading over medium delay times in the first example.

Disabling the caches had no significant influence on the data with and without virtual output.

In order to find out the perturbation by WindView we compared two runs with same settings (trigger frequency 1 kHz, same small task with a priority of 100, output of activity indicator on the target screen) but one with instrumentation for WindView installed, the other one without. In both cases WindView was not open. It turns out that the small number of samples spreading out up to delays in the order of 200 µs visible in figs. 2, 4, 5, 7 is caused by the installation of WindView, probably the tEvtTask running with highest priority. Watching the system by means of WindView (probably the transfer of data to the host) causes additional delays in the range of a few 10 µs. See fig 8.

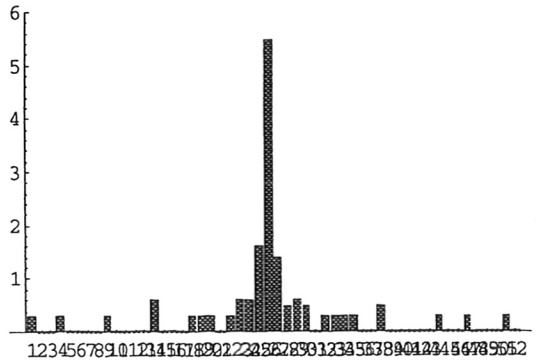
```
Number of data: 330000
min = 0.0007934    max = 0.0012067    interval = 0.0004133
mean = 0.001  standarddeviation = 9.38957×10^-6
```



```
{1, 0, 0, 2, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 2, 0, 5, 3, 1343, 1628, 1646,
 1666, 1585, 1730, 4425, 301988, 4368, 1725, 1595, 1653, 1637, 1645,
 1332, 5, 5, 0, 2, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 2, 0, 0, 1}
```

Fig. 4 Interrupt frequency 1 kHz, virtual IO, instrumentation for WindView installed but not used. Very short task with priority of 100.

```
Number of data: 300000
min = 0.0017847    max = 0.0022159    interval = 0.0004312
mean = 0.002  standarddeviation = 1.59855×10^-6
```



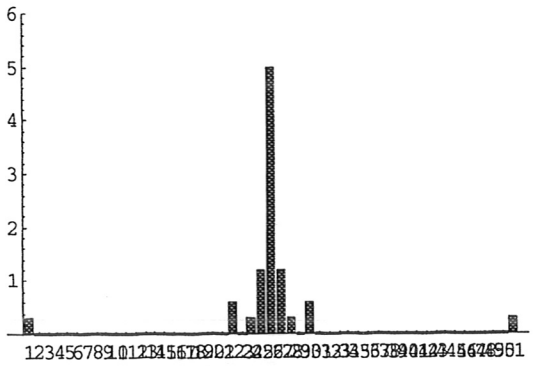
```
{1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 3, 0, 0, 0, 1, 1, 1, 0,
 1, 3, 3, 41, 299903, 24, 2, 3, 2, 0, 1, 1, 1, 1, 0, 0, 2, 0,
 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0}
```

Fig. 5 Interrupt frequency 500 Hz, virtual IO installed but without any output, instrumentation for WindView installed but not used. Very short task with priority of 100.

Number of data: 100000

min = 0.0075568 max = 0.0124432 interval = 0.0048864

mean = 0.01 standarddeviation = 0.0000115798



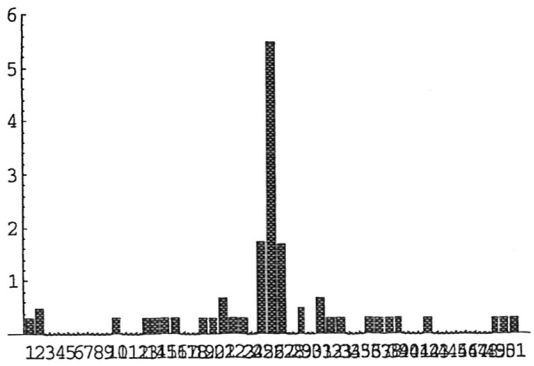
{1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 3, 0, 1, 15, 99960, 15, 1, 0, 3, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1}

Fig. 6 Interrupt frequency 100 Hz,
indicator displayed on target screen,
instrumentation for WindView not installed.
Loop task(1.5 ms average) triggered, priority = 2.

Number of data: 300000

min = 0.0008092 max = 0.0011912 interval = 0.000382

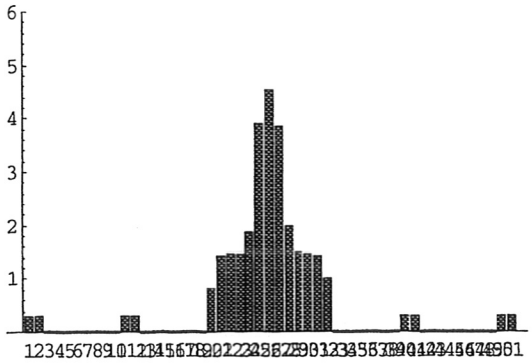
mean = 0.001 standarddeviation = 1.07405×10^{-6}



{1, 2, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 1, 1, 0, 0, 1,
1, 4, 1, 1, 0, 55, 299863, 50, 0, 2, 0, 4, 1, 1, 0, 0,
1, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1}

Fig. 7 Interrupt frequency 1 kHz,
virtual IO installed, no output,
instrumentation for WindView installed but not used.
Very short triggered task with priority of 100.

```
Number of data: 50000
min = 0.000806   max = 0.0011939   interval = 0.0003879
mean = 0.001  standarddeviation = 4.47341×10^-6
```




```
{1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 6, 26, 29,
 28, 77, 7728, 34260, 7642, 100, 31, 29, 27, 9, 0, 0, 0, 0, 0, 0,
 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1}
```

Fig. 8 Interrupt frequency 1 kHz, output on target screen, instrumentation for WindView installed and used. Very short triggered task with priority of 100.

Start of triggered task

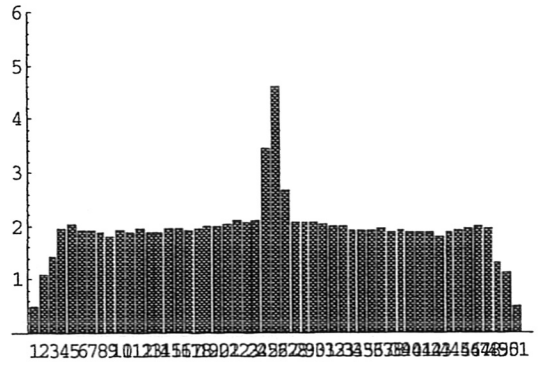
Similar to the measurement of the jitter of the ISR start we measured the jitter of a task start triggered by the ISR. In this case the jitter of the context switch adds – probably in a correlated way – to the jitter of the ISR start. This means that delays cannot simply be added algebraically, because, for example, rare periodic events causing long delays count only once

In the first example (fig. 9) the loop task (loop = 10000) is triggered at a frequency of 1 kHz with a priority of 100. Instrumentation for WindView is installed but not used. The indicator output goes as virtual IO to the host via Ethernet. About 93 % of the runs have a delay smaller than 8 µs, about 4 % spread out uniformly up to a delay of 390 µs. (note that samples with delays shorter than the average are not counted).

In the second example (fig. 10) the priority of the task has been increased to a value of 2. In contrast to the first example the network task and the wdB task now have a lower priority than the task to be tested. The loop has been removed in order to reduce the total load on the CPU and to give more time to the lower priority tasks. Now 97 % have a delay smaller than 8 µs, 3 % spread out uniformly up to 70 µs, a few samples are delayed up to 400 µs.

Additionally turning off any output in the third example (fig. 11) steepens the distribution of the bulk of samples dramatically. The maximum delay of about 250 µs is smaller than that in the other examples. Because of the small number of samples with high delay this might be insignificant.

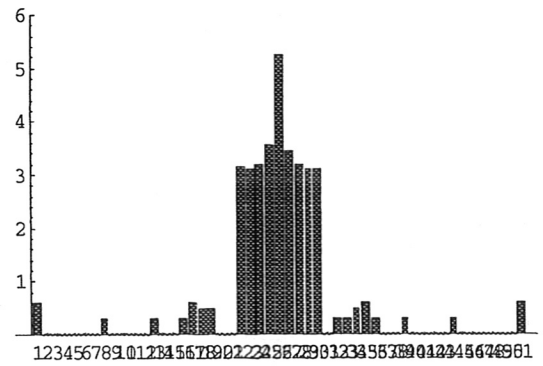
```
Number of data: 50000
min = 0.0006134    max = 0.0013918    interval = 0.0007784
mean = 0.001  standarddeviation = 0.0000564091
```




```
{2, 11, 26, 91, 110, 83, 79, 75, 65, 81, 75, 87, 75, 74, 90,
 90, 79, 87, 96, 99, 107, 124, 111, 128, 2790, 42873, 477,
 119, 113, 119, 103, 100, 99, 79, 84, 85, 86, 78, 85, 76,
 78, 74, 65, 75, 84, 86, 101, 93, 19, 12, 2}
```

Fig. 9 Interrupt frequency 1 kHz,
task with loop = 10 000, priority = 100, virtual IO,
instrumentation for WindView installed but not used..

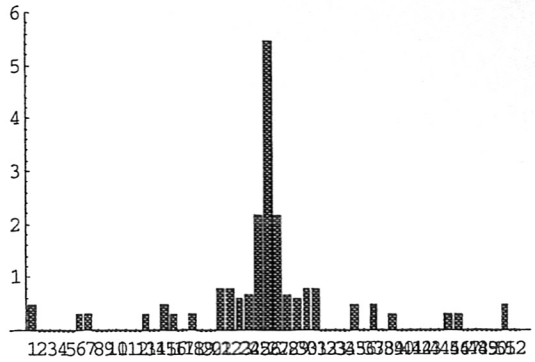
```
Number of data: 200000
min = 0.0005991    max = 0.0014032    interval = 0.0008041
mean = 0.001  standarddeviation = 0.0000104506
```




```
{3, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 3, 2, 2,
 0, 0, 1434, 1281, 1600, 3581, 185291, 2730, 1497, 1290,
 1270, 0, 1, 1, 2, 3, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 3}
```

Fig. 10 Interrupt frequency 1 kHz,
task without loop, priority = 2, virtual IO,
instrumentation for WindView installed but not used..

```
Number of data: 300000
min = 0.0007536   max = 0.0012466   interval = 0.000493
mean = 0.001   standarddeviation = 1.4359 10^-6
```



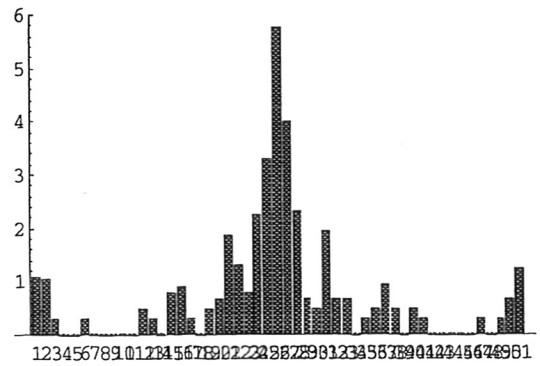
```
{2, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 2, 1, 0, 1, 0, 0,
 5, 5, 3, 4, 151, 299649, 148, 4, 3, 5, 5, 0, 0, 0, 2, 0,
 2, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 2, 0}
```

Fig. 11 Interrupt frequency 1 kHz,
task without loop, priority = 2, no output
instrumentation for WindView installed but not used..

End of triggered task – standard PC

In the last case we measure the jitter of the time between the end of successive runs of the triggered loop task.

```
Number of data: 600000
min = 0.000747   max = 0.0012512   interval =
 0.0005042
mean = 0.001   standarddeviation = 2.82468×10^-6
```



```
{11, 10, 1, 0, 0, 1, 0, 0, 0, 0, 0, 2, 1, 0, 5, 7, 1, 0, 2,
 4, 75, 19, 5, 183, 2048, 586743, 10528, 208, 4, 2, 93,
 4, 4, 0, 1, 2, 8, 2, 0, 2, 1, 0, 0, 0, 0, 0, 1, 0, 1, 4, 17}
```

Fig. 12 End of task: interrupt frequency 1 kHz,
loop = 20000 (0.3 ms), priority 2,
display on target screen,
no instrumentation for WindView

The loop parameter was 20 000, the average execution time of the loop was about 0.3 ms, the repetition frequency was 1 kHz, priority was 2. Any unnecessary system activities were avoided. The activity indicator was displayed on the screen, instrumentation for WindView was not installed. 600000 samples were taken to improve statistics. Fig 12 shows the result.

These measurements were repeated with the new bc635pci board. Further components of the VxWorks system, for example the target agent, were removed. In another run the trigger frequency was slightly changed to 1025 Hz. In no case a significant change of the distribution, especially no change of the number of data with large delay was detected.

Triggering with the system clock at a frequency of 60 Hz increased the standard deviation of the distribution significantly from a value of about 3 µs in the previous cases to 4.2 µs. This is due to a broadening of the central peak. The tail of the distribution was not significantly changed.

Samples with a delay larger than 200 µs occur independently of the trigger frequency with a period of about 30.7 s. In cases with high trigger frequency mostly bursts with a few delayed samples alternating with undelayed samples appear, in a few cases, however, no delayed samples occurred when we expected them. With 60 Hz trigger the same fraction of samples shows this long delay and the delayed samples are on the 30.7 s grid. However, only a few grid points show strongly delayed samples and bursts are never observed, resulting in the same total number of delayed samples with strongly increased recording time of 10 000s for 600 000 samples.

End of triggered task – Compact PCI system

Measurements of the jitter of the end of triggered tasks as reported in the previous chapter

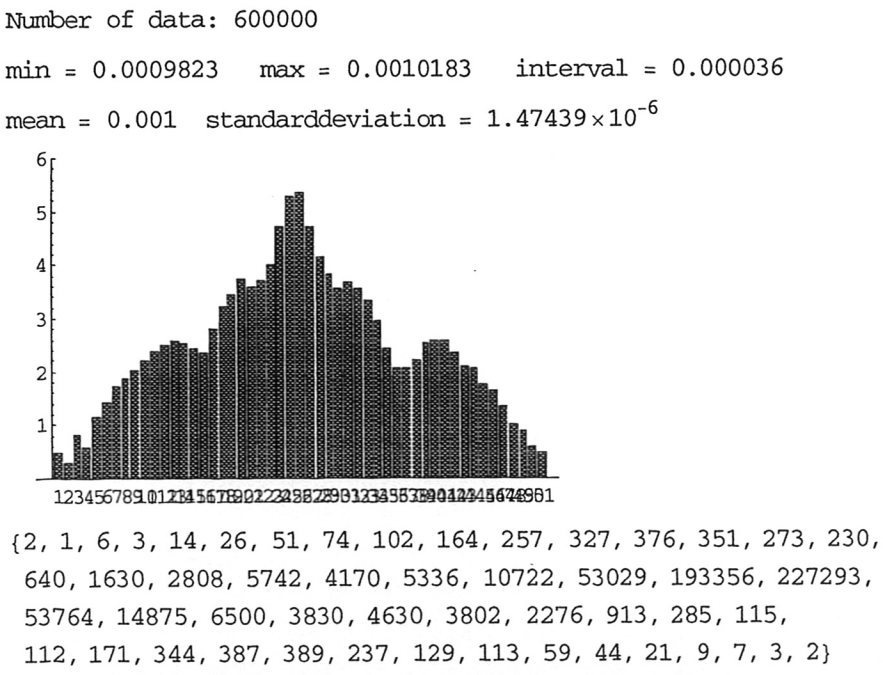


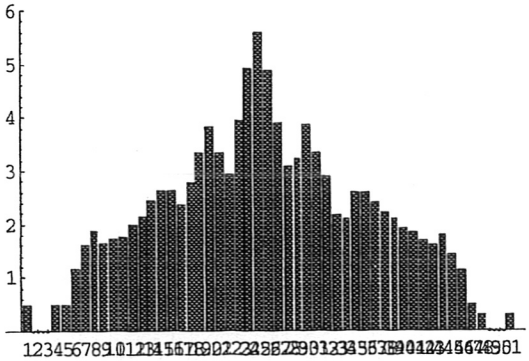
Fig. 13 End of task: CPCI system, new bc635 board, interrupt frequency 1 kHz, loop = 20000 , priority 2, display on target screen, no instrumentation for WindView

were repeated with the Compact PCI computer from OR. Fig. 13 shows a histogram

comparable to that shown in fig. 12. Note that the time scale is different. This system exhibits a standard deviation about half as large as the standard PC.

Eliminating unnecessary system components, for example the tWdb task, reduces the standard deviation, but not the maximum delay (fig.14).

```
Number of data: 600000
min = 0.0009813   max = 0.0010202   interval = 0.0000389
mean = 0.001  standarddeviation = 1.16903×10^-6
```




```
{2, 0, 0, 2, 2, 14, 40, 74, 43, 52, 57, 98, 137, 277, 416, 419,
 234, 622, 2099, 6716, 2185, 855, 8621, 81019, 395923,
 77275, 7798, 1249, 1668, 7092, 2283, 787, 145, 127, 406,
 374, 252, 164, 123, 83, 71, 47, 42, 64, 27, 12, 2, 1, 0, 0, 1}
```

Fig. 14 End of task: CPCI system, new bc635 board, interrupt frequency 1 kHz, loop = 20000 , priority 2, no display of system activity, minimized operating system, no Wdb agent

Samples with extremely long delay were not observed with this bc635 board. With the older version of the board, however, in each of two runs with 600 000 samples one bad sample was found, one being about 100 µs too early. A spurious interrupt seems to be the only plausible explanation for this observation.

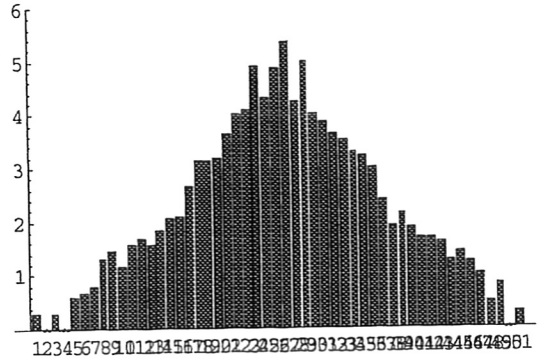
In order to increase the chance for random events to perturb the measurement with the new bc635 board the measurement was repeated with a trigger frequency of 50 Hz and a numerical task taking a factor of 20 more time so that the run took more than three hours. The result is shown in fig. 15.

The standard deviation is about tripled, but no bad samples were observed.

Number of data: 600000

min = 0.0199606 max = 0.0200374 interval = 0.0000768

mean = 0.02 standarddeviation = 3.43484×10^{-6}



{1, 0, 1, 0, 3, 4, 5, 20, 28, 14, 36, 50, 37, 70, 118, 125, 463, 1479, 1472,
1546, 4187, 10482, 12424, 84344, 21276, 74770, 240321, 17121, 98143,
10611, 7487, 4407, 3425, 2046, 1739, 982, 250, 85, 136, 78, 48, 48,
40, 18, 25, 17, 9, 2, 6, 0, 1}

Fig. 15 End of task: CPCI system, new bc635 board,
interrupt frequency 1 kHz,
loop = 400000 , priority 2,
no display of system activity,
minimized operating system, no Wdb agent.

Conclusions

We had set up real time systems for periodic operation based on two types of a PC. Because of random delays both systems show a jitter depending on task parameters and the configuration of the operating system. The jitter of the standard PC has an unexpected distribution function with a probability of a few times 10^{-5} for the delay reaching a value in the order of 250 µs.

A few effects responsible for the jitter could be identified, but by far not all. The process causing the extremely long delay could not be identified. It is not even clear whether it is located in the computer board, in the timing board, in the operating system or in the application software.

Operating systems like VxWorks offer many mechanisms for influencing the delay by executing operations with different urgency in different tasks, choosing proper priority levels and suppressing unnecessary system activities. In spite of this the total system behaves in an unexpected unfavourable way.

The operating system performs highest priority activities which a priori cannot be controlled by the user, for example activities related to the system clock, other activities cannot be controlled because of insufficient knowledge about them.

It seems to be impossible to predict the maximum delay or the probability distribution of delays. General purpose computers with high performance processors, real time operating systems with many features like VxWorks and many add-on boards are so complex that in practice nobody can be sure to consider all possible effects. In this sense high level real time

operating systems do not produce another quality of computer systems compared to general purpose operating systems but offer more degrees of freedom for tailoring the system and speeding up urgent operations. This is not a special property of VxWorks, therefore this statement should not be misinterpreted as an opinion about this operating system

The measurement of response times based on software tools and system monitoring tools like WindView interferes with the process to be monitored. Measurements with external equipment like bus analyzers should be preferred in critical cases.

The probability density distribution of response times must be determined by statistical methods from a very large number of samples (at least $10^5 - 10^6$) because very rare cases of extremely long delay are possible, and just effects of these few cases could sum up to large errors over a long time or cause a disaster in a practical application. Even test runs with a duration of weeks cannot guarantee that the worst case has been observed. A consequence of this is that all code running on high performance standard computers under full-featured operating systems must tolerate some probability of failure to finish within the time limit foreseen. For example, the accuracy of clocks based on counting interrupts is reduced if only a few ticks are missed.

The concept of a real time system as a "deterministic" system with guaranteed response to unexpected events within a certain time limit seems to be an illusion if very complex components are used. Instead, response within the given limit with sufficiently high probability is a realistic alternative even for cases with hard real time requirements provided the software takes the few cases of failure into account and the probability of a failure is definitely low enough to be tolerated in the special application.

A careful selection of computer hardware is necessary in order to come closest to a "deterministic" system.