Latency Measurements of FESA

19-22.12.2006/15.03.2007
Tobias Hoffmann

Abstract

With the release of FESA Version 2.9 an actual measurement of the timing behavior is advisable. As FESA 2.9 (on LynxOS) now provides the possibility to set a priority at startup of a FESA-executable, this behavior is also of interest. The influence of the optimization flags on the performance has been investigated. Within this document the internal processes such as kernel threads with very high priority are not treated.
The possibility to compile the RT part in a single-thread-executable (make rt) is also taken into account within the timing measurements.
In principle the same measurements can be done for a Linux driven RIO 3 for comparison purposes.

Setup and procedure:

- RIO3 CPU, 254MB
- LynxOS
- FESA 2.9
- CTRP Timing board
- VMOD-IO with VMOD-DOR piggy-back
- LeCroy 9314M Oszilloscope

Trigger-Generation:

An LTIM trigger was used to provide the same trigger on the LEMO output connector of the CTRP as on the interrupt level for the RT Action execution. The definition of the trigger was done in the LTIM FESA class. LEMO output #1 of the CTRP module was connected to the oszilloscope 50 Ohm input for a sharp pulse.

RT Action:

For this measurement a very simple FESA class was designed. No SERVER action, one instance using the predefined LTIM event and one RT Action. For the measurements different versions of this RTAction were compiled. Three versions for toggling the channels 1-3 of the

DOR module and a version as a separate RT thread using command “make rt”. The example source code is listed in box 1.

```
//
//  FESA framework                    June 2004.
//
// Use this code as a starting-point to develop your own equipment class

#include <acquisition.h>

// INPUT fields:
// OUTPUT fields:

extern "C" {
#include <drvrutil/dioaiolib.h>
#include <gm/moduletypes.h>
}



using namespace thoffman_ADemo;

acquisition::acquisition(const string& name, AbstractRTAction::RTActionConfig& rtActCfg) :
          RTAction<RTEvent, thoffman_ADemoGlobalStore, thoffman_ADemoDevice>(name, rtActCfg){}

void acquisition::execute(RTEvent * pEv){

  int ret, nibble;


  DioRead(IocVMODDOR,0,1,&nibble); //get bit state (usually 0 at start = level high)

  ret= DioWrite(IocVMODDOR, 0, 1, (nibble | 0x1)); //set to 1 (level low) or vv, for channel this is 0x2,
                                                   //  for 3 it is 0x4 and so on.

  ret= DioWrite(IocVMODDOR, 0, 1, (nibble & 0x1)); //set to 0 again (level high) or vv

}
```

Box 1: RT Action source code, part of the thoffman_ADemo FESA class.

Measurement:

At startup all output register bits of the VMOD-DOR are zero. This leads to a HIGH on all channels, when it is measured against a 1kOhm pull-up resistor connected with the 5V output of this module. The RT Action switches the level to LOW and back to HIGH. The pulse width corresponds to the rise- and fall-time (Fig.1) of this level and is not of importance within this measurement. At the beginning of the measurements the command DioChannelInit was applied in advance of the level setting. It turned out that this command had no effect. Its time consumption is mentioned in Fig. 1.

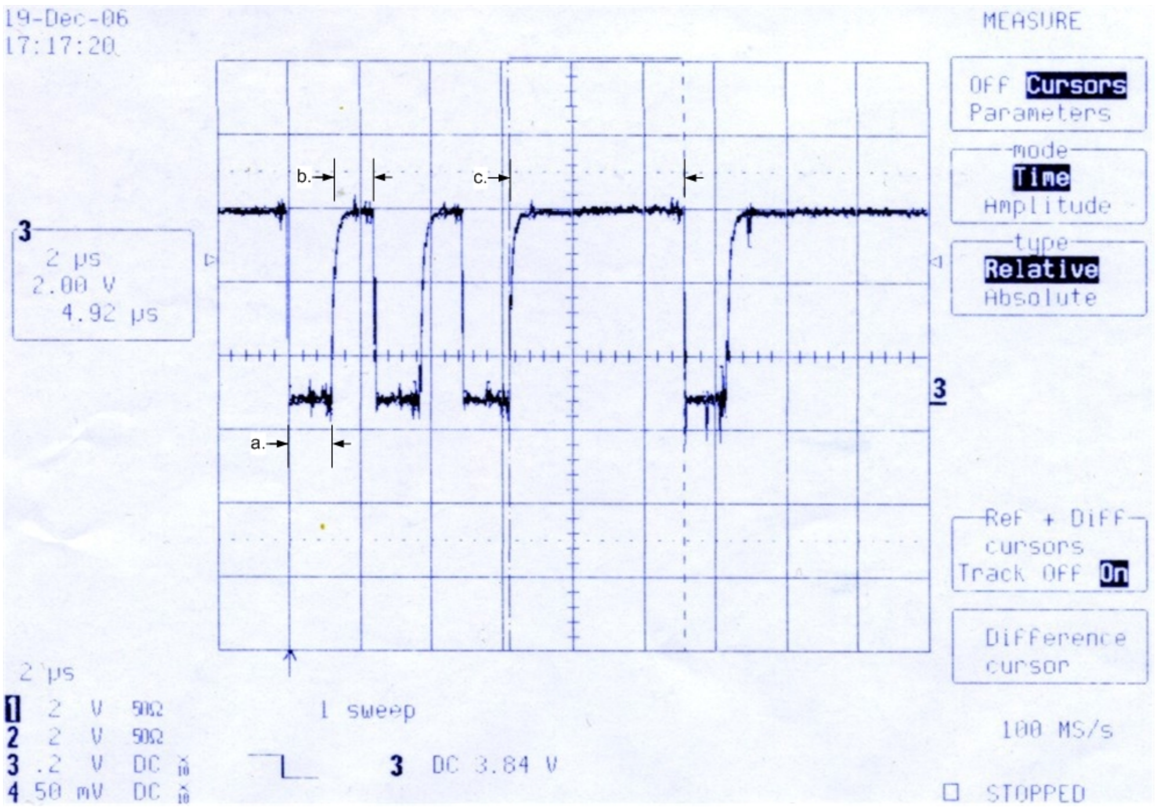

Figure 1: Latency times for read, write and init commands on VMOD-DOR. 3xoff/on with DioWrite followed by DioInit and finally one DioWrite off/on. Times a. and b. = 1.25μs, c. = 5.0 μs. Time c. consists of DioInit (3.75 μs) and DioWrite (1.25 μs).

Note: The VMOD-I/O with its possible piggy-backs is designed for single process operation and not for multi-threading. The access to the board is protected by one single semaphore flag which is observed by the OS. In case of two or more different threads operating at the same time on this board, the required action (hi/lo, read/write etc.) is delayed.
The thread-to-thread execution delay time for this module and this CPU setup was identified as 250±10μs for each thread. The response time of read/write actions on the board of approx. 1.25μs may be in this case neglected.

Note: At this time, a stable, reliable and reproducible latency time measurement regarding FESA can not be done. This is due to probable changes on source code (Timing, FESA and possibly FPGA firmwares etc.), which are relevant in this framework, which is always on enhancement.
E.g.: Measurements were undertaken to define the fastest response time from the occurring hardware trigger until the raising signal from the I/O module. This was evaluated being 560μs on one day and

430µs on the next day (after recompilation). This significant change of more then 20 % should show that all timing measurements have to be undertaken frequently. The most probable reason for this behavior is text output on the front-end, but also changes on source code may have effects.

As you would need to document all versions of every linked code, and in addition list up all running parallel processes on the used CPU, which obviously have also significant influence on the performance, these results presented here have to be interpreted as a direction and not as a final fact.

HW-trigger to I/O reaction – Latency-time

The complete reaction time starting from the HW-trigger (LTIM event) until the I/O signal is lowered is determined as **430µs** (see Fig.2). This includes the complete FESA activity plus the hardware correlated influences. The priority settings on LynxOs up to 255 can not improve that value. The measurement was done after a fresh reboot without significant parallel users except LTIM Fesa class.

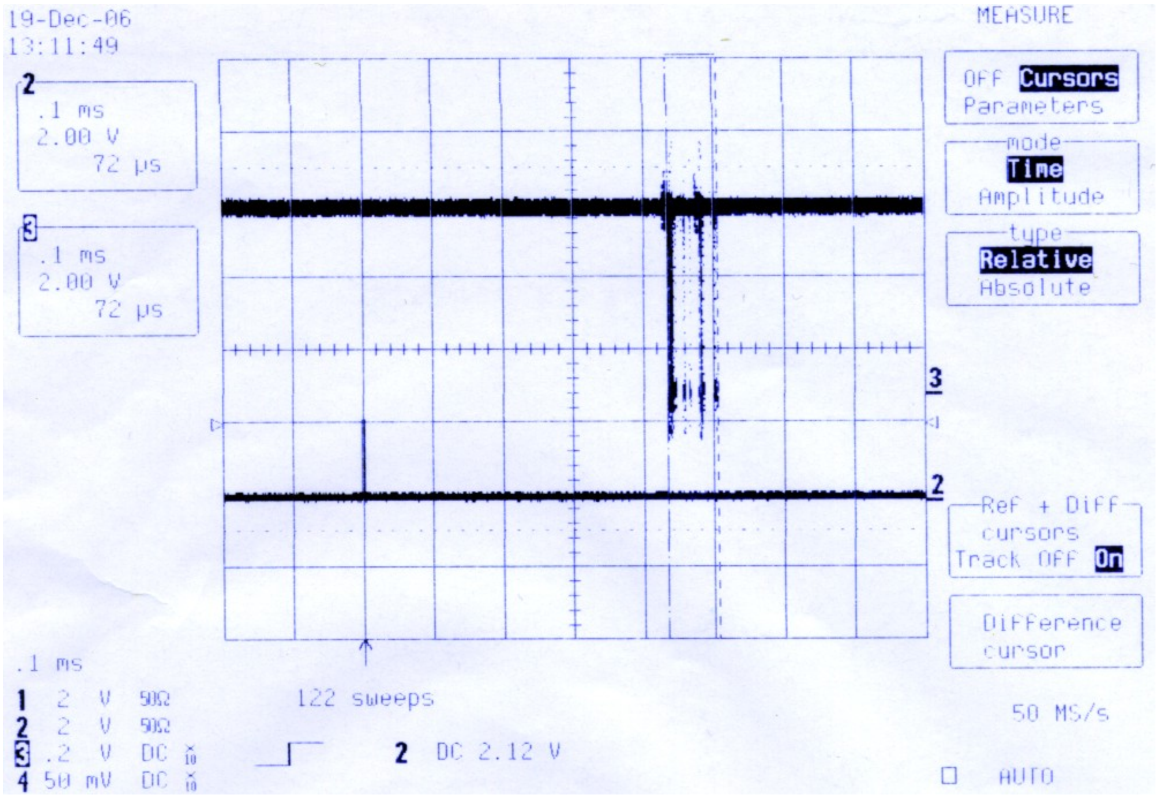

Figure 2: Response time to a hardware trigger is 430µs plus CPU systematic delays.

In any case of executed measurements typical delay fractions (approx. 20µs steps) can be seen. This is supposed to be a scheduler characteristic and is originated by the OS-kernel and not by FESA. The average time frame for this delay distribution is 75µs. It can not be called a jitter as it is not randomly distributed.

Also compiling the Fesa class in a separate RT thread using “make rt” and starting this executable with different priorities has no influence on the response time of 430µs.

Influence of PRIO on two parallel processes

After a fresh reboot of the CPU two equal processes were started with different PRIO settings. Process 1 was switching channel 1 on the DOR board, process 2 the channel2. As this module suffers from a strong crosstalk on the outputs, the results can be observed on one oszilloscope channel, only the pulse height is different, which is not essential for this timing measurement.
Process 1 was always started first.

No.	**Prio** Process 1	**Prio** Process 2	**t_{min} [µs]** Process 1	**t_{max} [µs]** Process 2	**t_{min} [µs]** Process 1	**t_{max} [µs]** Process 2
1	17	17	1026	1116	1252	1348
2	50	17	588	684	1620	1900+
3	17	50	1620	1900+	604	688
4	100	50	596	688	1220	1352
5	50	50	1004	1120	1248	1368
6	100	100	1004	1152	1252	1392
7	17	100	1640	1940	590	710

Table 1: 2 Processes running at different PRIO settings.

As a first obvious outcome one can see, that setting two parallel processes to the same priority (grayed lines in table 1) has no positive effect for both of them. They both run delayed compared to the maximum response time. The second one in addition is delayed by the DIO access (approx. 250µs). This could also be confirmed for three different processes at same priority.

Setting a higher priority to one of two parallel processes has a significant influence on the response time. The one with higher priority is executed close to the shortest possible response time (at time of this measurement this was 560µs). The second process instead is furthermore delayed.

Comparing line 2 and 4 of table 1 shows, that setting a higher priority (more than standard 17) to both processes, leads to a decrease of the response time for the lower priority process. It was found out, that the step size of the priority setting has no influence. That means that there is no visible difference between setting 100/50 or 100/99. The response times are equal in both cases like in line 4.

A measurement with 3 parallel processes (Fig.3) shows the same results. The processes with higher priority than 17 were shifted to faster response times; the process with priority 17 instead was additionally delayed. That means, priority setting delays others processes significantly.

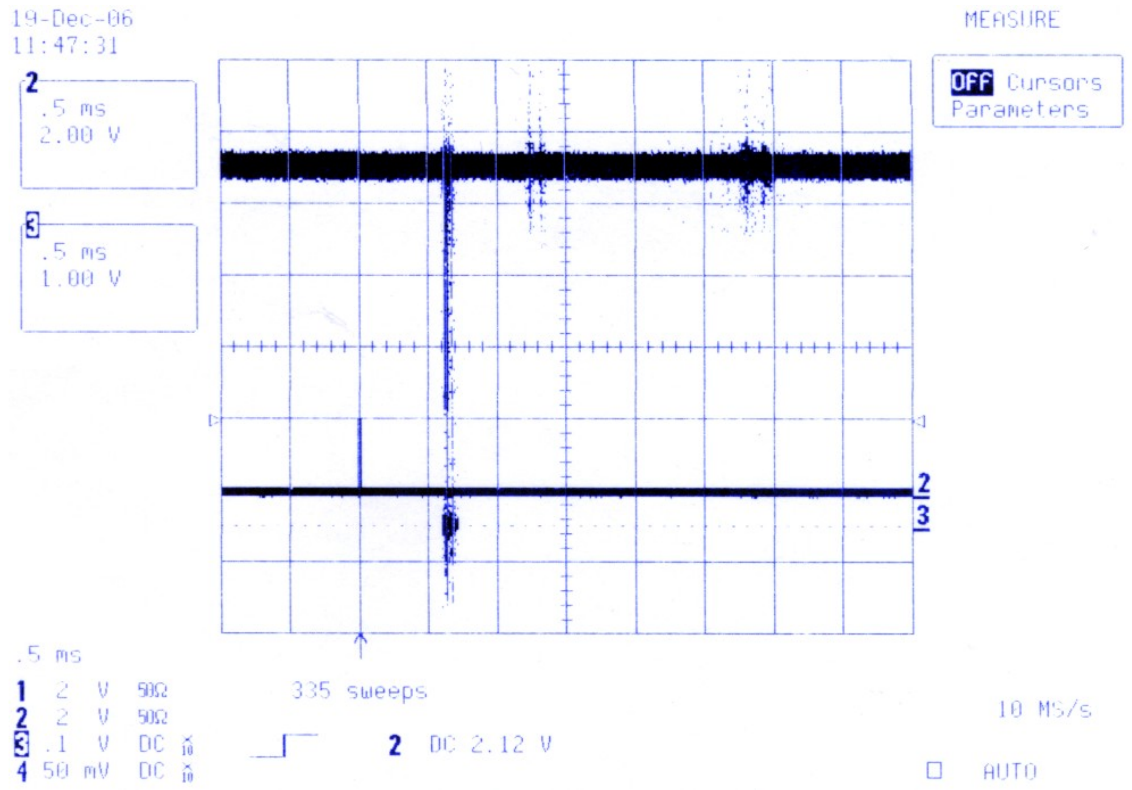

Figure 3: Three parallel processes with priority settings 50/17/25. The second started process is the one at the most right (prio 17).

Influence of gcc optimization flags

The idea is to improve latency time by using the gcc optimization flags while compile time. There are the types -O2, -O3, and –Os which were tested with FESA. Setting the flag to the FESA-Equipment class only, had no effect. Compiling the complete FESA framework plus the equipment class decreased the latency time by 25%. The results are presented in Fig. 4, 5, and 6. The executable FESA classes were started with priority setting to 50.

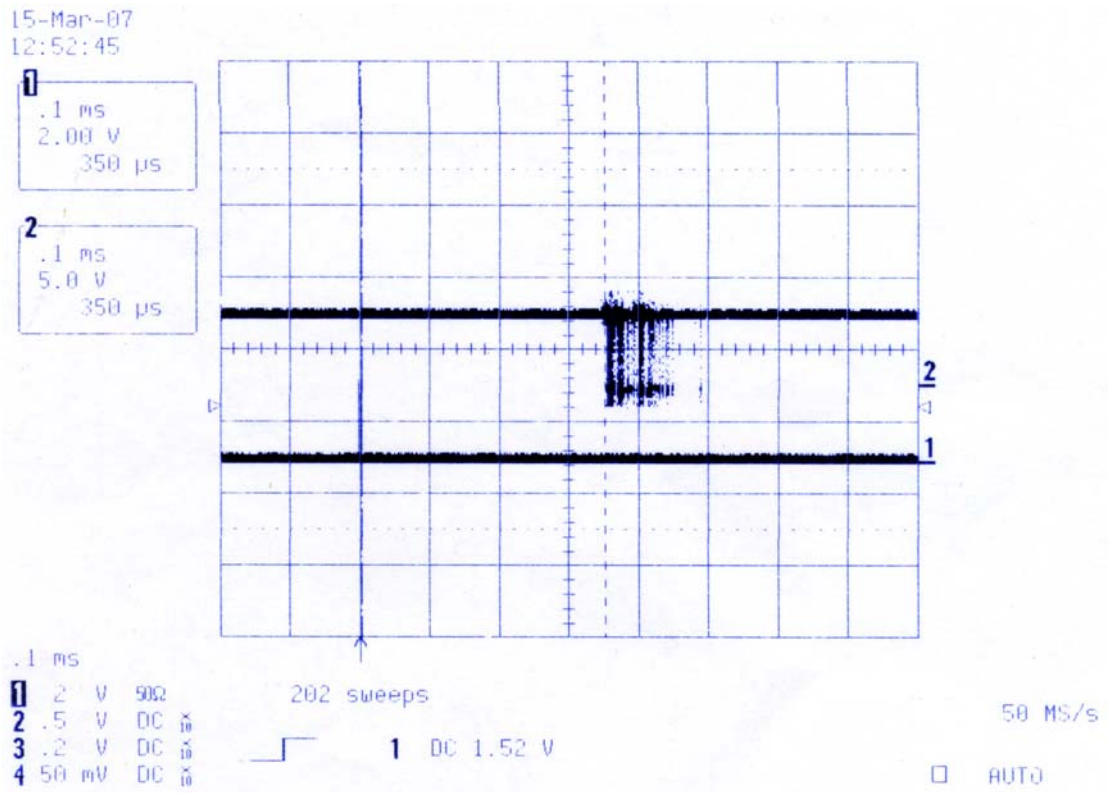

Figure 4: Measurement with Compile-Flag –O2 and priority setting 50. The shortest latency time is 350µs (without flag it was 430 µs).

The latency time did not change on setting the flag –O3 instead of –O2 as can be seen in Fig. 5. The flag –Os which is usallly used for size-optimization did not reduce the size, but had the best effect as shown in Fig. 6 with a (best) latency time of 338µs. Nevertheless other secondary influences on stability of FESA were not investigated. Also the third-party libraries e.g. from the timing section were not compiled with –O flags. This test should be performed to gain probably another improvement.

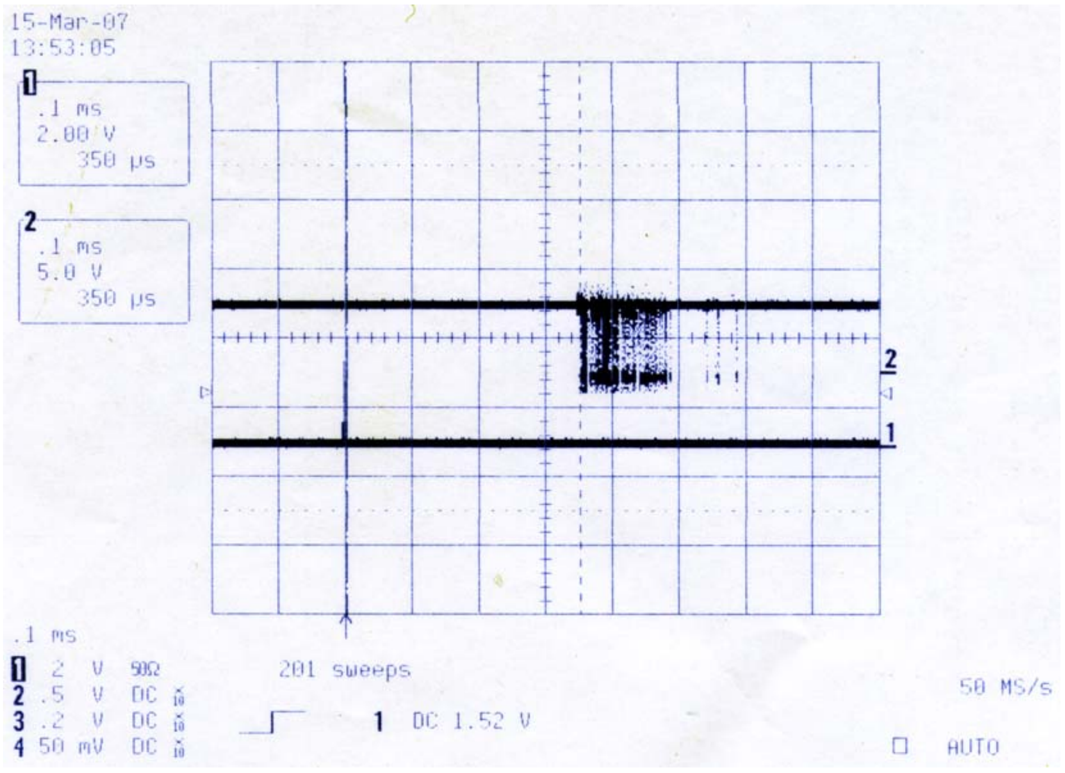

Figure 5: Measurement with Compile-Flag –O3 and priority setting 50. No difference to –O2 can be seen.

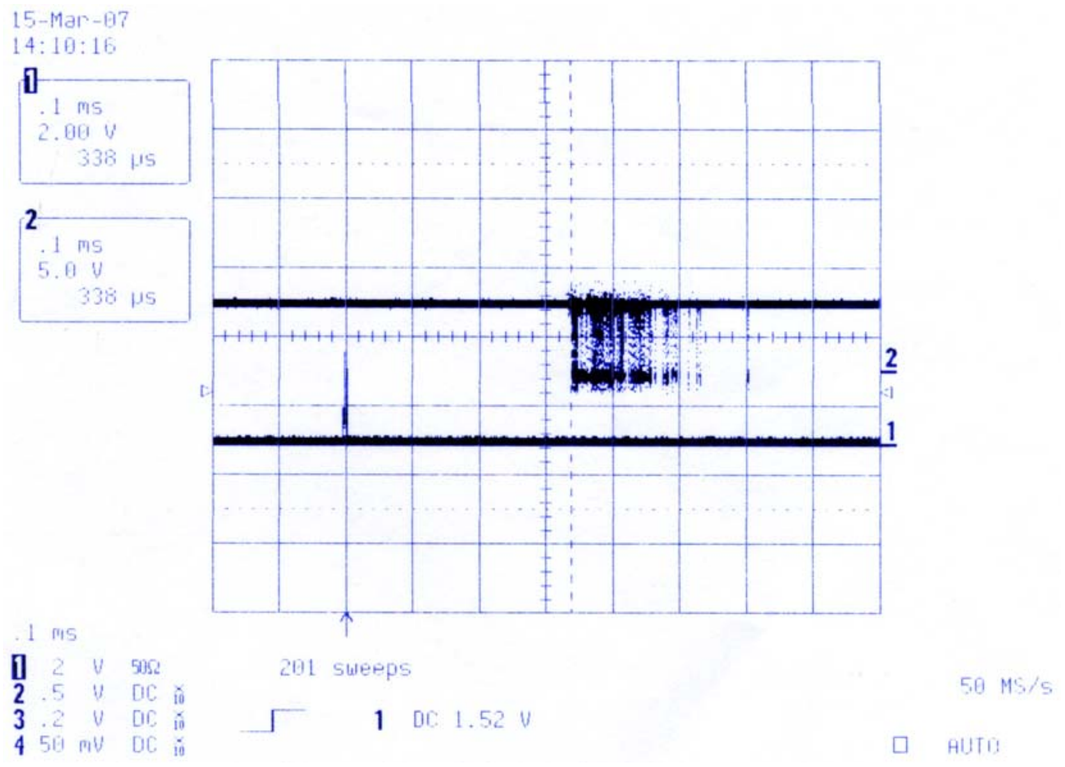

Figure 6: Measurement with Compile-Flag –Os and priority setting 50. The shortest latency time is 338µs, the best value within the complete test measurements.

Following Makefiles where changed for the mentioned optimization measurements:

Framework:

- CMW
- CORE
- DATASTORE
- EXCEPTION
- INTERFACE
- LOGGING
- PERSISTENCY
- PLC
- RECORDER
- RT
- SECURITY
- SORTING
- SYNCHRONIZATION
- UTILITY

FESA Equipment class:

- GENERATED CODE
- RT

Addendum

After reboot following processes were running, this could have influence on the scheduling and the timing-behavior.

```
last pid:   117;  load averages:  0.01,  0.00,  0.00
16:50:21
47 threads:    1 running, 3 ready, 43
waiting

Memory: 254M total, 33M user, 10M kernel, 210M
free


PID   USERNAME          TID  PRI  TEXT     STK    DATA   STATE   TIME   CPU     COMMAND
   0  root                0    0  0K       0K     0K     ready   3:24   0.00%   nullpr
   0  root                8  100  0K       8K     0K     wait    0:00   0.00%   amd
  20  root                9   18  65K      52K    68K    wait    0:00   0.00%   unfsio
   0  root                2  100  0K       8K     0K     wait    0:00   0.00%   bsd_netisr
  63  root               37   17  1154K    48K    152K   wait    0:00   0.00%   sshd
  22  root               10   18  65K      52K    67K    wait    0:00   0.00%   unfsio
  24  root               11   18  65K      52K    67K    wait    0:00   0.00%   unfsio
  26  root               12   18  65K      56K    73K    wait    0:00   0.00%   unfsio
  35  root               28   25  4175K    56K    312K   wait    0:00   0.00%   LTIM_M
  83  root               18   25  4119K    56K    308K   wait    0:00   0.00%   proz1.ppc4
  33  root               27   33  248K     44K    104K   wait    0:00   0.00%   dtmrt_ls
  86  thoffman           46   17  386K     52K    175K   wait    0:00   0.00%   tcsh
  29  root               26   17  516K     44K    92K    wait    0:00   0.00%   bash
  80  root               36   17  516K     44K    94K    wait    0:00   0.00%   bash
 -83  root               41   17  4119K    44K    308K   ready   0:00   0.00%
  28  root               25   17  154K     40K    55K    wait    0:00   0.00%   telnetd
  78  root                7   17  154K     40K    55K    wait    0:00   0.00%   telnetd
 -14  root               23   99  438K     44K    150K   wait    0:00   0.00%
  87  thoffman           45   17  96K      36K    326K   run     0:00   0.00%   top
  14  root               22  100  438K     40K    150K   wait    0:00   0.00%   get_tgm_tim
 -83  root               40   17  4119K    36K    308K   wait    0:00   0.00%
 107  root               20   19  255K     44K    60K    wait    0:00   0.00%   sysReporter
  79  root               17   17  15K      36K    10K    wait    0:00   0.00%   syncer
   0  root                4   17  0K       4K     0K     wait    0:00   0.00%   TX
  98  root               19   17  283K     40K    146K   wait    0:00   0.00%   xntpd
  31  root               15   17  155K     44K    61K    wait    0:00   0.00%   inetd
   0  root                1   17  0K       8K     0K     ready   0:00   0.00%   CALLOUT
 -35  root               34   25  4175K    40K    312K   wait    0:00   0.00%
 -35  root               29   25  4175K    36K    312K   wait    0:00   0.00%
 -35  root               31   25  4175K    36K    312K   wait    0:00   0.00%
 -35  root               32   25  4175K    36K    312K   wait    0:00   0.00%
  34  root               24   25  516K     44K    62K    wait    0:00   0.00%   sh
 -35  root               30   25  4175K    36K    312K   wait    0:00   0.00%
  13  root               21  100  516K     44K    63K    wait    0:00   0.00%   sh
 -35  root               33   25  4175K    36K    312K   wait    0:00   0.00%
 -83  root               42   25  4119K    36K    308K   wait    0:00   0.00%
 -83  root               43   25  4119K    40K    308K   wait    0:00   0.00%
 -83  root               44   19  4119K    36K    308K   wait    0:00   0.00%
 117  root               16   19  68K      36K    23K    wait    0:00   0.00%   errlocal
 -35  root               35   19  4175K    36K    312K   wait    0:00   0.00%
   0  root               13   18  0K       8K     0K     wait    0:00   0.00%   nfssync
  76  root                6   17  41K      36K    27K    wait    0:00   0.00%   login
   0  root               14   18  0K       8K     0K     wait    0:00   0.00%   nfsaio
 -83  root               38   17  4119K    36K    308K   wait    0:00   0.00%
 -83  root               39   17  4119K    36K    308K   wait    0:00   0.00%
   0  root                5   17  0K       4K     0K     wait    0:00   0.00%   RX
   1  root                3   16  28K      36K    11K    wait    0:00   0.00%   init
```