Optimizing Test times

for LED Analyser-based Test Systems

Application note

Table of Contents

1. Introduction

In this application note an optimization will be perform, following the tips described in section "Test speed optimization tips".

The list of tips shown there are a result of a research performed using LED Analysers with different hardware versions and firmware versions, tested in multiple computers with a wide range of different conditions for long periods of time.

The components and general requirements for this experiment are described below:

- Computer Dell XPS: Windows10 Pro 64-bit Core i5-8400, 16Gb RAM
- 27 LEDs to be measured
- 1x Feasa 20-F (Fw F215)
- 1x Feasa 10-F (Fw F210)
- 27 x Optical Heads OH-3
- USB 3.0 cables and ports are used
- DLL 2.4.1
- Parameters required: Relative Intensity, CIE xy 1931, CCT

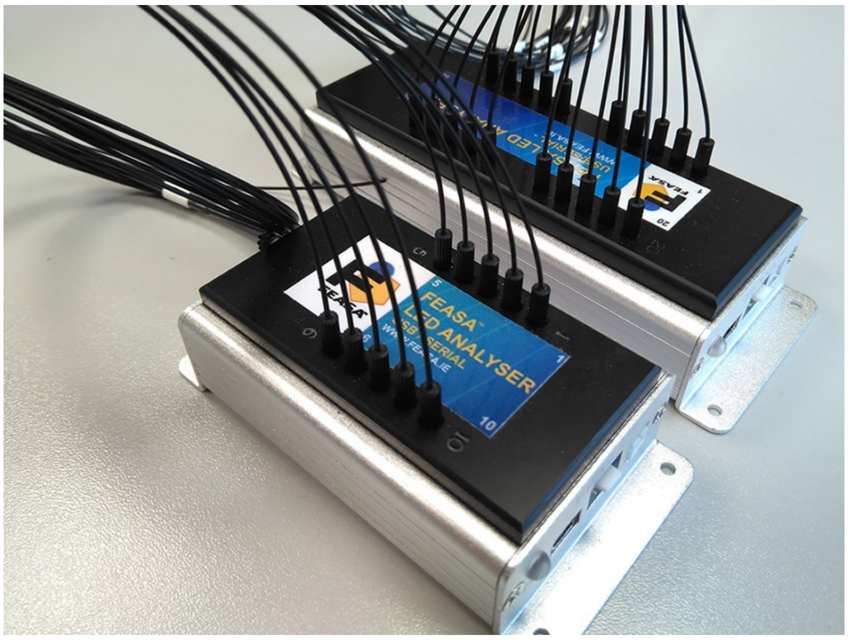


Image 1: Feasa LED Analysers used for the test: Feasa 10-F on the left and Feasa 20-F on the right.

Image 2: Feasa Optical Heads OH-3, used in the test.

In order to evaluate the long-term effects of test time, a 100 loop test will be performed. This will allow to extrapolate the effect to a real production environment.

Note: the test software used in this Application Note is provided along with this document with a ZIP file.

2. Experiment

2.1 Initial Setup

For the initial setup, both Analysers will be connected to the computer through USB using default settings: latency at 16ms and 57600 baud. Each LED Analyser will be connected using a different USB cable and port.

Optical heads OH-3 will be used at a distance of 8mm.

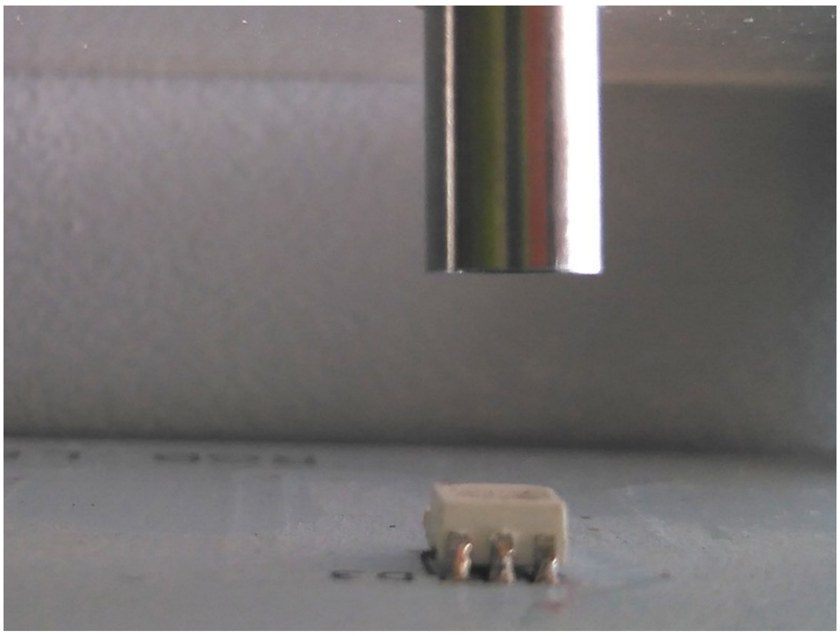
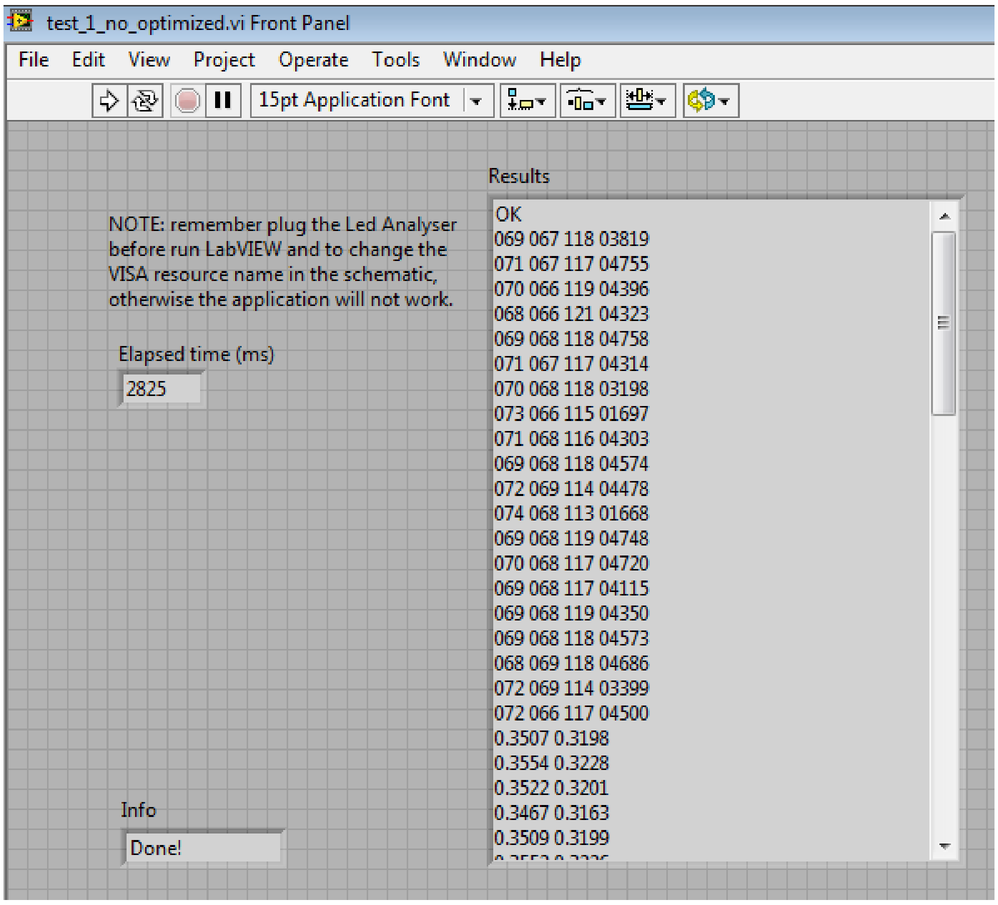


Image 3: Optical Head OH-3 placed on top of LED, 8mm gap.

The test software interface will be developed using LabVIEW 12.0 and ports will be accessed through VISA library.

Finally, Auto Capture will be performed and data will be downloaded using GETRGBI, GETxy and GETCCT, for each one of the fibers (addressed individually). A test schema of Open port, capture, download data and close port is used, for each individual port, in a sequential manner.


test_1_no_optimized.vi Front Panel
File Edit View Project Operate Tools Window Help
15pt Application Font

NOTE: remember plug the Led Analyser before run LabVIEW and to change the VISA resource name in the schematic, otherwise the application will not work.

Elapsed time (ms)
2825

Info
Done!

Results

```
OK
069 067 118 03819
071 067 117 04755
070 066 119 04396
068 066 121 04323
069 068 118 04758
071 067 117 04314
070 068 118 03198
073 066 115 01697
071 068 116 04303
069 068 118 04574
072 069 114 04478
074 068 113 01668
069 068 119 04748
070 068 117 04720
069 068 117 04115
069 068 119 04350
069 068 118 04573
068 069 118 04686
072 069 114 03399
072 066 117 04500
0.3507 0.3198
0.3554 0.3228
0.3522 0.3201
0.3467 0.3163
0.3509 0.3199
```

Image 4: LabVIEW test program

As can be seen in the picture above, the average test time for this 100-loop test is 2825ms.

In order to evaluate the effect of using the DLL, a new test has been performed using it, throwing a result of 2868ms.

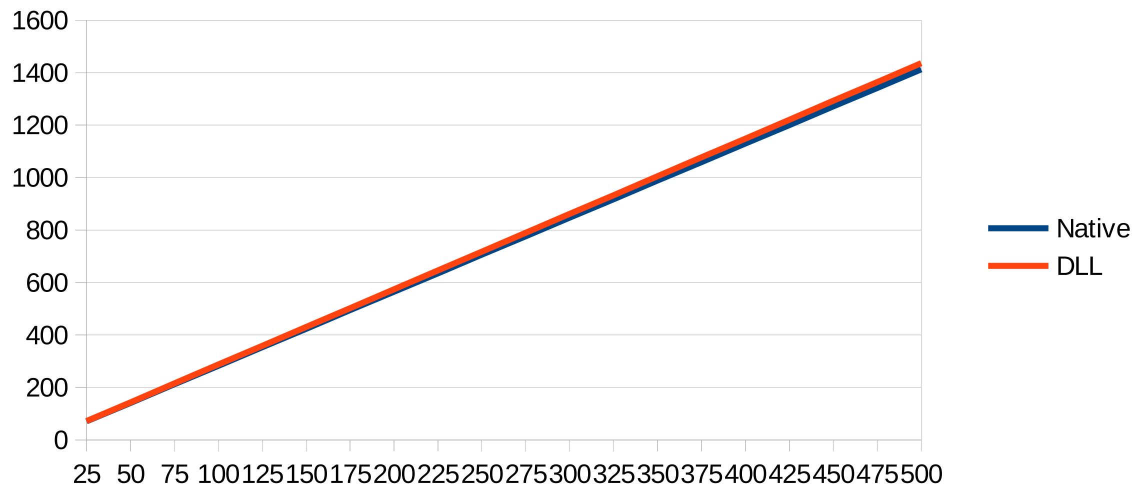


Chart 1: Number of tests loops vs Elaspsed time.

As can be seen in Chart 1, the timing difference is not significant (1,5%), moreover considering that no board swap & setup has been taken into account here. So in case this is taken into account (ex: 8s), the difference could be around 0,42% (with adequate optimizations can be even lower), which is a fair price to pay considering the benefits of using the DLL.

2.2 Final Setup

In order to improve the speed and reduce the test time, and based on the tips described at the end of this document, the following improvements have been suggested.

A) Use C# instead of LabVIEW + 64bit Feasa DLL

B) Open ports at the beginning of the test and close at the end (instead of in each loop).

C) Reduce GAP between LED and Optical head to 3mm (instead of 8mm)

D) Use Capture in Range 2 instead of Auto-Range

E) Use GETXYI instead of GETRGBI + GETXY

F) Reduce USB port latency to 1ms

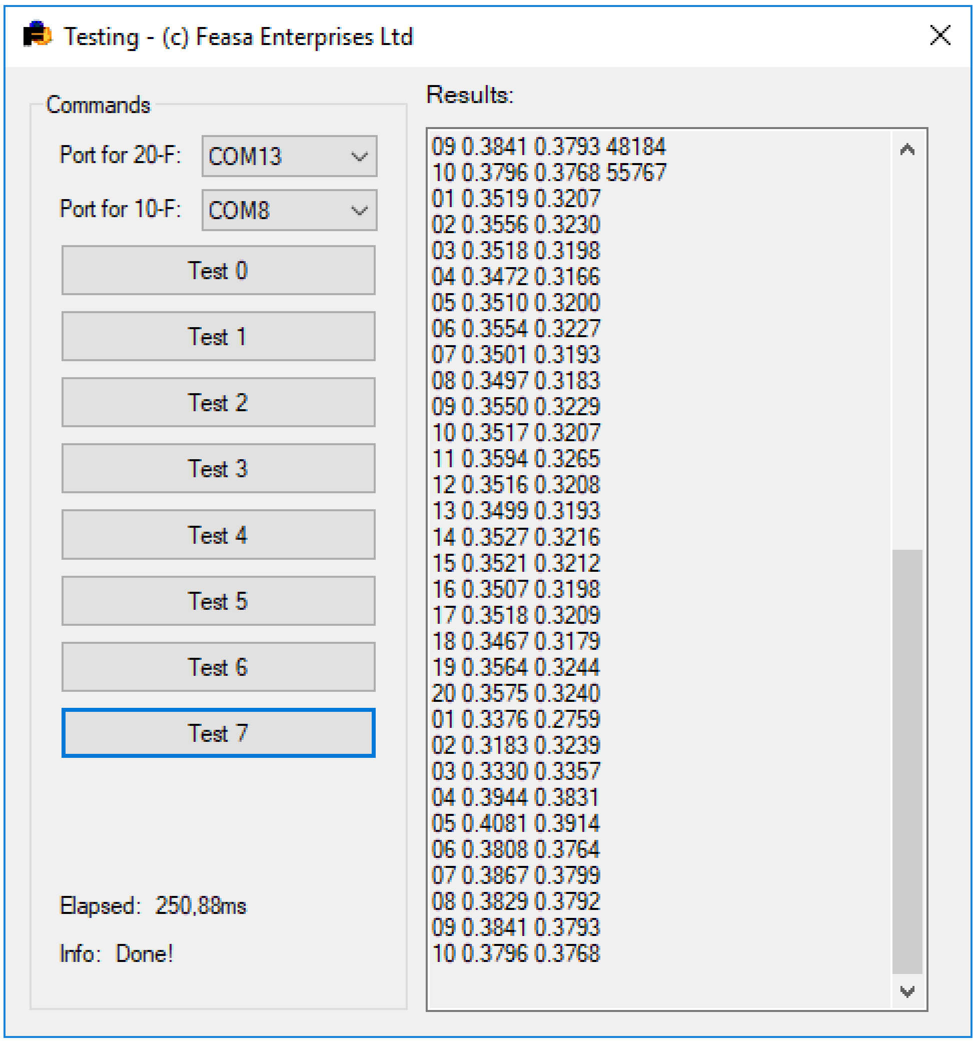
G) Increase baud-rate to 460800

H) Use GETALL commands

I) Use Multi-threading function instead of accessing ports sequentially.

In order to appreciate the effects of timing improvements, we will be progresively applying improvements:

- Test 0: improvement A will be applied → Average loop time: 2686.22ms
- Test 1: improvement B will be applied → Average loop time: 2247.29ms
- Test 2: improvements C & D will be applied → Average loop time: 1750.04ms
- Test 3: improvement E will be applied → Average loop time: 1322.03ms
- Test 4: improvement F will be applied → Average loop time: 805.64ms
- Test 5: improvement G will be applied → Average loop time: 643.6ms
- Test 6: improvement H will be applied → Average loop time: 474.99ms
- Test 7: improvement I will be applied → Average loop time: 250.88ms



```
09 0.3841 0.3793 48184
10 0.3796 0.3768 55767
01 0.3519 0.3207
02 0.3556 0.3230
03 0.3518 0.3198
04 0.3472 0.3166
05 0.3510 0.3200
06 0.3554 0.3227
07 0.3501 0.3193
08 0.3497 0.3183
09 0.3550 0.3229
10 0.3517 0.3207
11 0.3594 0.3265
12 0.3516 0.3208
13 0.3499 0.3193
14 0.3527 0.3216
15 0.3521 0.3212
16 0.3507 0.3198
17 0.3518 0.3209
18 0.3467 0.3179
19 0.3564 0.3244
20 0.3575 0.3240
01 0.3376 0.2759
02 0.3183 0.3239
03 0.3330 0.3357
04 0.3944 0.3831
05 0.4081 0.3914
06 0.3808 0.3764
07 0.3867 0.3799
08 0.3829 0.3792
09 0.3841 0.3793
10 0.3796 0.3768
```

Elapsed: 250,88ms

Info: Done!

Image 5: C# test program

As shown in the picture above, the C# program contains a button to perform each one of the tests described here.

Now let's show a chart with the time needed to perform a certain number of test with respect to the test time:

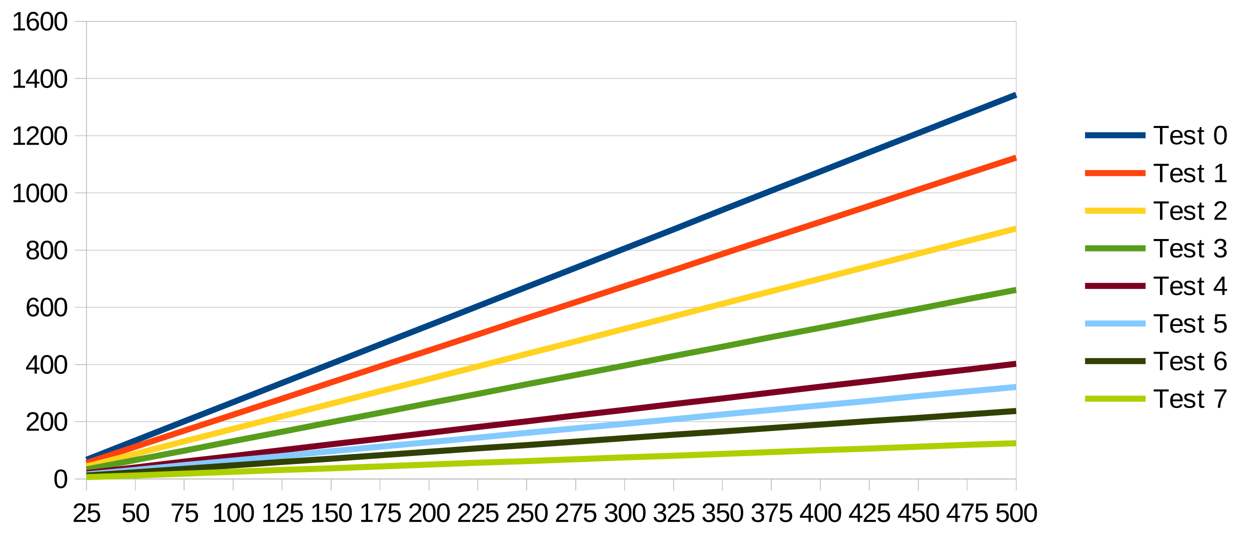


Chart 2: Number of tests loops vs Elaspsed time (for each optimization made).

Now we can represent the same graph in terms of how many tests can be performed along time:

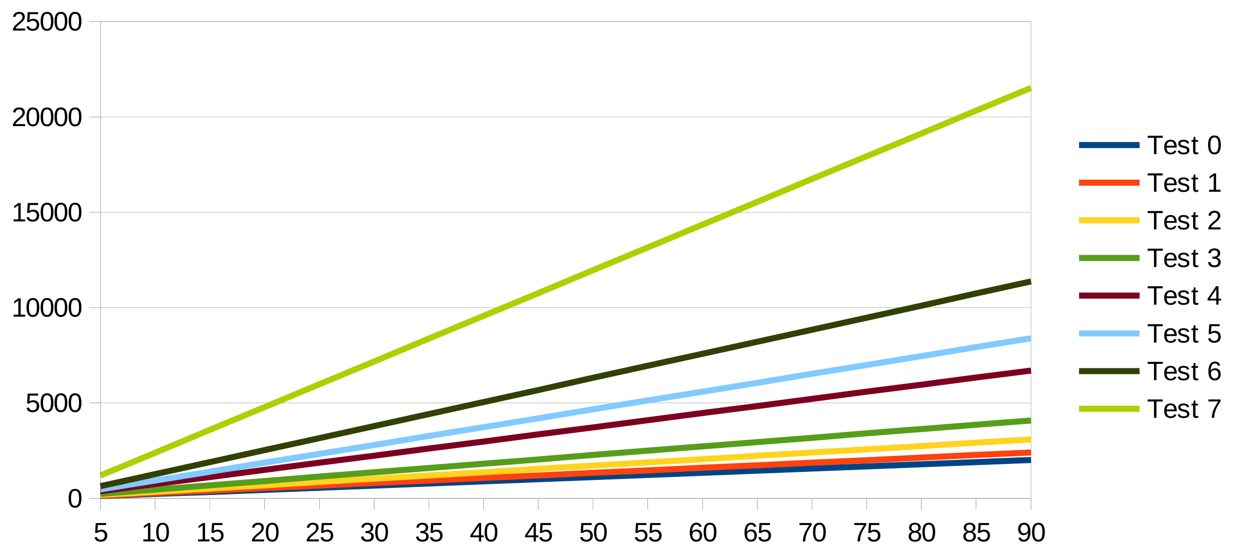


Chart 3: Number of loops vs Time (minutes) (for each optimization made).

2.3 Conclusions

The graphs above are not realistic (~1000% of improvement!), since the board swap & setup time (time between tests) hast not been taken into account, however it does reflect the importace of optimizing the test time in order to to achieve a higher efficiency in production.

Depending on how the production line is implemented and how this affects to the swap time, the effect of the improvements in the production time can change.

In order to evaluate this, the C# program has been modified to introduce a delay at the end of the test, so the same experiment has been repeated for "Test 0" and "Test 7", which throws the following results:

Test	**Swap & setup time**	**Loop/Test time**	**Improvement**
Test 0	6 s	8641.09 ms	-
Test 7	6 s	6334.63 ms	36.4 %

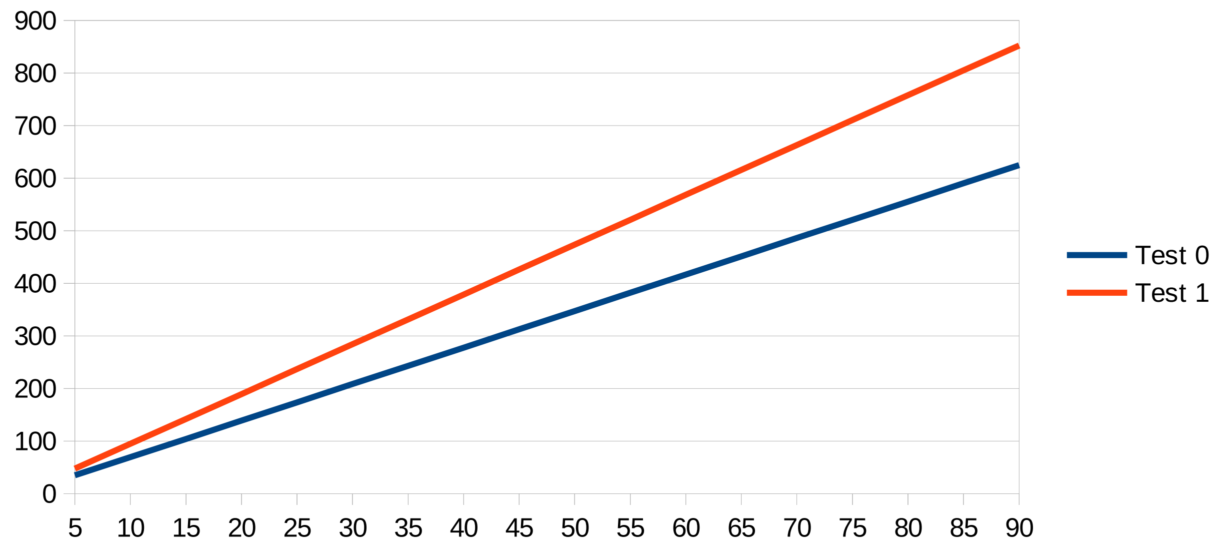


Chart 4: Number of loops vs Time (minutes) (in a more realistic environment).

In the graph above the long-term effects in test time can be clearly seen, since after 90 minutes, 852 test were performed in the improved test, in contrast with the 624 test performed in the initial setup.

3. Test optimization tips

- Find the best distance for the combination of your LED + Optical Head, between 2 to 5 mm.

- Ensure that the optical adaptor and fibers are clean and placed at a recommended distance.

- Avoid Opening and closing ports in each test loop, since this can be time consuming.

- Use a known baud-rate to open the ports instead of using *AUTO* for baud-rate detection.

- In case you use some setup commands like *SETEXPOSURE*, place them at the beginning of the test loop, instead of sending it in each loop.

-Use *PUT* commands instead of *SET*, if your firmware provides this feature. Ex: use *PUTEXPOSURE2* instead of *SETEXPOSURE2*

- Avoid the use of AUTO range. Pick the quicker range that provides relative intensity results of 70K to 95K in Logarithmic mode, or higher than 10K in Linear mode. Or use a combination of range + exposure factor. Always starting from less sensitive range (Range 5), to the most sensitive range.

- In case of PWM-modulated LEDs, use the method described above to determine the best capture range and then, test the best number of frames for testing the DUT that provides the better ratio between speed and stability.

- Don't use unnecessary commands or download unnecessary data. Ex: don't use *GETHSI* or *GETRGBI* to download only intensity values.

- In case of testing all the fibers use *GETALL* commands and then parse the information, instead of using commands to retrieve values from individual fibers.

- Avoid the use of laptops if speed is critical

- For RS232 connections, use a physical port, avoiding adapters or purchase a good quality PCIe card with a good chipset and reduced latency.

- In case of using a RS-232 to USB adapter, ensure that it is a good quality product and ensures RS-232 voltage levels.

- In case of communication through USB, reduce the latency to 1ms

- Increase baudrate to 115200 in Serial

- Increase baudrate to 460800 in USB

- For high performance applications, 921600 baud can be used in combination with FTDI DLL libraries and enabling the Event byte (set it to ASCII char 0x04 EOT). Remember to enable the EOT character in the LED Analyser.

- Use USB 1.1 ports or higher. USB2.0/USB3.0 ports are recommended for high performance or intensive test systems.

- Avoid the use of USB hubs or use USB 2.0/3.0/3.1 certified with external power supply.

- In case of using a USB hub with more than 2 devices connected to it, ensure that it has an external power supply able to supply at least 500mA per output, or 750mA for USB3.0 ones.

- Use a device with Firmware Version >=200 (*GETVERSION*) for high performance applications.

- DLL/SO: its usage is always recommended, But for high performance or intensive test systems running on Windows, performance can be affected and a Native communication method should be used instead.

- Use 64bit DLL/SO libraries in 64bit architectures.

- Use Feasa DLL/SO for simplicity or robustness in C/C++ or C# or use Native port programming for speed, through C/C++, C# or Delphi

- In case of multiple analysers, use a multi-port/multi-thread (see DLL multi-thread functions) topology for speed or Daisy-Chain for simplicity.

- The number of cores in your processor is important when working with multi-threading topologies. So in this case, ensure to have at least 4-cores.

- Operative System: use Linux for high performance applications using SO library or embedded systems. Use Windows + Native communication methods for critical or intensive test systems, of Windows + DLL otherwise.

- Ensure your computer has at least 8Gb or RAM

- It's desirable to have a computer with a good motherboard with a good bandwidth (bus speed)

- A modern processor will offer a better performance (5th generation, better than 3rd one, and 7th generation is even better; also Core i3 is better than Core 2 Duo).

- Avoid interpreted languages. The closer the EXE code generated is to the hardware architecture, the better. Enable optimizations for the target platform.

- Try not to install other software or drivers not related to the test system / process, since this can consume resources and create delays.

- Firewalls and anti-virus programs can monitor ports, DLLs and programs, so be sure that, in case of having any of that software installed, ensure that is setup properly and the test system software and hardware have enough privileges to run without delays or interruptions. Despite of having it configured properly, some security software can still induce latencies, so be careful on which security suite to choose.