

## Abstract

This document details the SPAD23 hardware features and connectivity options. It furthermore contains an overview of its software package and user interface.



## Revision history

Version	Notable changes
0.1	Initial release with basic functionalities: <ul style="list-style-type: none"><li>• Live view of the current SPAD photon count rates</li><li>• Photon counting, both single shot/continuous</li><li>• Coarse timestamping with 10 ns time separation</li><li>• Pulse width measurements</li><li>• Voltage control</li><li>• Temperature reading</li></ul> Windows 64-bit only.
1.0	First full release with multithreading support and a remote control option: <ul style="list-style-type: none"><li>• All USB communication is executed in a separate thread, as well as the data processing</li><li>• Timestamping data is saved directly to file</li><li>• Timestamping data is separated per pixel</li><li>• Timestamping module features a dummy count to keep a high datarate and allow to detect coarse counter overruns</li><li>• Timestamping with improved read-out scheme to process events up to 90 Mcps.</li><li>• Voltage control in firmware with range safeguarding and default startup values</li><li>• Remote command interface allows for easy control from external software</li></ul> Features TDC readout options in the timestamping tab in case a TDC system is detected. General bugfixes and GUI improvements. Windows 64-bit + Linux 64-bit.
1.1	Improved GUI handling and DPI scaling. Continuous counting mode image generation. High light intensity shutdown. General TDC system enhancements. Histogram per dwell generation. Restore settings on restart. Daily check for a new version.
1.2	Improvements to the continuous counting mode. TDC laser frequency selection.
1.21	Continuous timestamping mode (if measurement time = 0).

1.22	Real-time streaming of timestamps through TCP/IP. Automatic setting of $V_q$ dependent on operating voltage $V_{OP}$ . Frequency detection of the SMA inputs (firmware versions newer than 20201209 only). General bugfixes and performance tweaks.
1.23	Fixed regression of setting automatic $V_q$ . Check for TDC calibration validity.
1.30	Added support for the new TDC method. Added support for firmwares with new headers. Image per pixel generation in continuous counting mode.
1.31	Fixed regression in timestamping marker generation.
1.32	Changed marker representation as being a negative value. Fixed fine timestamp value printed as absolute instead of relative to laser clock.
1.33	Changed TDC histogram time-axis saved separately to save space. Added TDC timing diagram examples. Added possibility to switch the remote interface port.
1.34	Fixed a bug in the timestamp streaming through TCP/IP. Changed the default $V_{OP}$ operating voltage.
1.40	Altered dummy save behaviour to only record overflows. Tweaked timestamping memory consumption. Small tweaks to the output of continuous counting measurements and image per frame generation. Basic image generation in timestamping mode. Needs dwell/line/frame clocks. Small changes to remote commands and their returns.
1.41	Fixed a bug in the image generation. Fixed a bug in the timestamp streaming.
1.42	Binary streaming of timestamps through TCP/IP.
1.43	Added scrollbars to control panels.
1.44	Added option to disable VOP. Added option to disable the internal clock markers. Added option to reroute clock markers from different SMAs. Added option to align the 23 TDC channels. Added command to check TDC calibration status. Changed TDC coarse counter to be in sync with laser clock. Minor bugfixes, changed continuous intensity measurements to scanning measurements.

## Downloads

Below are the links to the latest software versions for both Windows and Linux. Both software versions should run standalone and feature the same functionalities. The Windows version has been tested on Windows 10. The Linux version is tested on Ubuntu 16.04 and 20.04.

- Windows 64-bit: [https://www.piimaging.com/software/SPAD23\\_standalone\\_win64.zip](https://www.piimaging.com/software/SPAD23_standalone_win64.zip)
- Linux 64-bit: [https://www.piimaging.com/software/SPAD23\\_standalone\\_linux64.zip](https://www.piimaging.com/software/SPAD23_standalone_linux64.zip)

The Windows package contains two folders:

- **Drivers**  
Contains the USB driver to be installed before the first run.
- **SPAD23\_standalone\_win64**  
Contains all required components to run the software, execute it by calling the `SPAD23.exe`.

The Linux package also contains two folders:





- **Install**  
Contains the rules to be set in order to access the USB device. Simply execute `sudo ./install.sh` in a terminal.
- **SPAD23\_standalone\_linux64**  
Contains the main executable to run the software, execute it by calling `SPAD23`, or run `./SPAD23` in a terminal. The application might require execution permissions, set this with `sudo chmod +x SPAD23`.

## Contents

<b>1 Introduction</b>	<b>3</b>
<b>2 Hardware</b>	<b>4</b>
Sensor . . . . .	4
System . . . . .	4
<b>3 Graphical user interface</b>	<b>5</b>
Menu . . . . .	5
Live view . . . . .	6
Photoncounting . . . . .	6
Timestamping . . . . .	7
Coarse system . . . . .	8
TDC system . . . . .	8
Settings . . . . .	10
<b>4 Remote command interface</b>	<b>11</b>
Command guide . . . . .	11
Programming examples . . . . .	12

## 1 Introduction

The SPAD23 system is enclosed in an aluminium box with the properties and connections detailed in Section 2. The systems corresponding software package contains both a graphical user interface (Section 3) and a remote command interface (Section 4). The GUI is divided into four tabs.

 Live view
  Photoncounting
  Timestamping
  Settings

- Live view
- Photoncounting
- Timestamping
- Settings

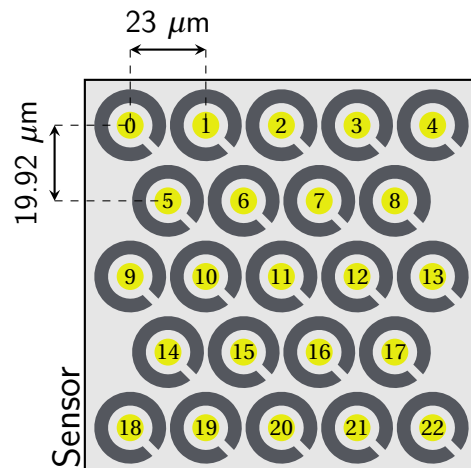
All functionalities accessible through the GUI can be remotely executed as well from any TCP/IP capable software package.



## 2 Hardware

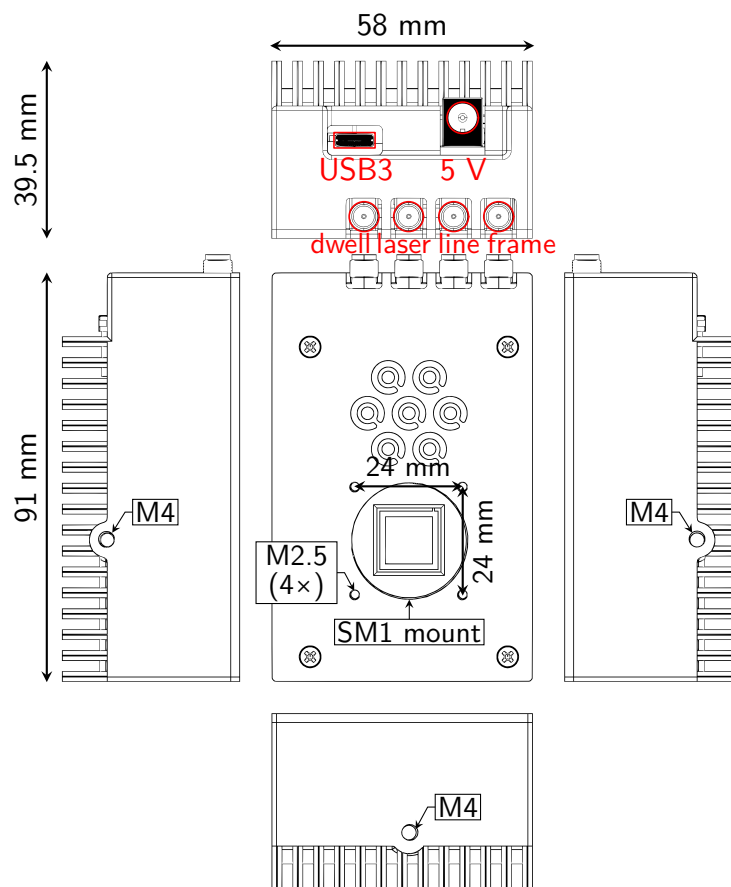
### Sensor

The SPAD23 system contains a sensor with 23 SPAD pixels arranged in a hexagonal configuration. The spacings and dimensions of the pixels are highlighted in the drawing on the right. Pixel 0 is oriented towards the top left corner of the box (towards the USB3 connector).



### System

The system is housed in a small aluminium enclosure measuring 58×91×39.5 mm. The mechanical drawing below shows the different connectors together with relevant mounting information.



It is powered from a single 5 V adapter and data can be read from the USB3 (USB type C) port. The SMA connectors on the top of the package have the following functionality:

- Dwell clock **input**  
High impedance, maximum input voltage: 3.3 V.

- **Laser clock input**  
Frequency: 1–99 MHz, high impedance, maximum input voltage: 3.3 V.
- **Line clock input**  
High impedance, maximum input voltage: 3.3 V.
- **Frame clock input**  
High impedance, maximum input voltage: 3.3 V.

Acceptable input voltage ranges: 0–2.5 V, 0–3.3 V, and 0–5 V (5 V only when terminated with a 50  $\Omega$  load to make it effectively 2.5 V). The minimum detectable pulse width for the frame/line/dwell clocks is 15 ns. These clocks can be reassigned to different SMA connectors from software version 1.44 and firmware version 21 onwards (see the extended description in Section 3).

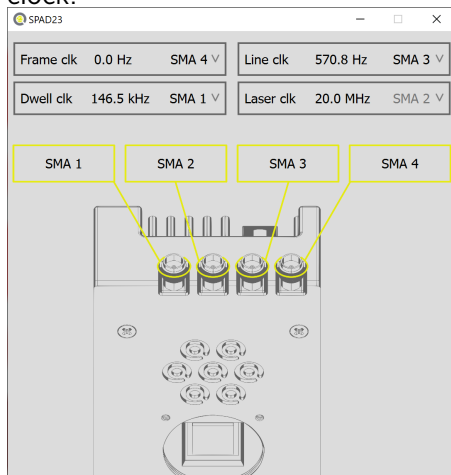
The system, FPGA + sensor, consumes in idle operation mode around 1.8 W. This is mainly static power consumed by the FPGA resources. Under measurement operation, the power consumption increases to maximum 2 W. For optimal thermal performance, the system can be shutdown when unused for an elongated period of time. A proper metal support on the box can help to absorb part of the heat in the large thermal mass of an optical table. Alternatively, a fan can be placed adjacent to the system, in order to actively cool the detector.

### 3 Graphical user interface

#### Menu

The options menu is available in the top left corner of the user interface. There are the following options:

- **Toggle live view to linear/log**  
Switch between the display of the count rate in the live views in a linear or logarithmic fashion.
- **Toggle histogram to linear/log**  
[TDC version] Switch the histogram displayed on the timestamping tab to have either a linear or logarithmic y-axis.
- **Change clock SMA mapping**  
Opens a secondary window in which the user can reassign the SMA ports to a different frame/line/dwell clock.



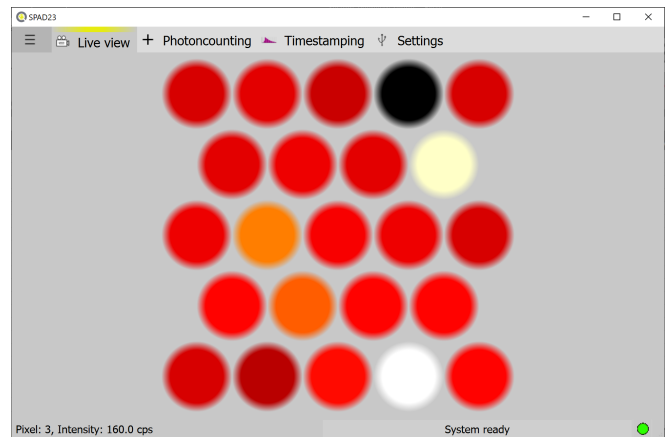
Example mapping for the default configuration.

- **Disable/enable VOP**  
Turn on and off the detectors high-voltage supply.
- **Disable/enable dummy clock markers**  
By default the firmware will generate dummy clock markers for frame/line/dwell clocks as long as no signal is detected on the corresponding SMA port. This additional switch allows to disable the dummy generation in all circumstances.
- **Check for updates.** Forces an online check for a new software version.
- **Help.** Opens this manual.
- **Quit.** Closes the software.

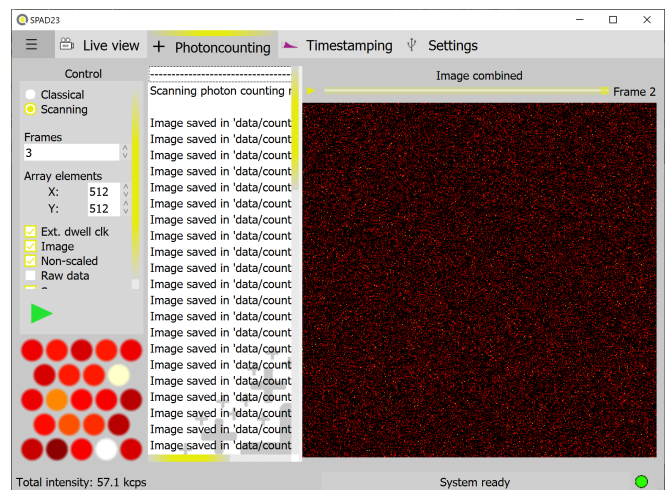
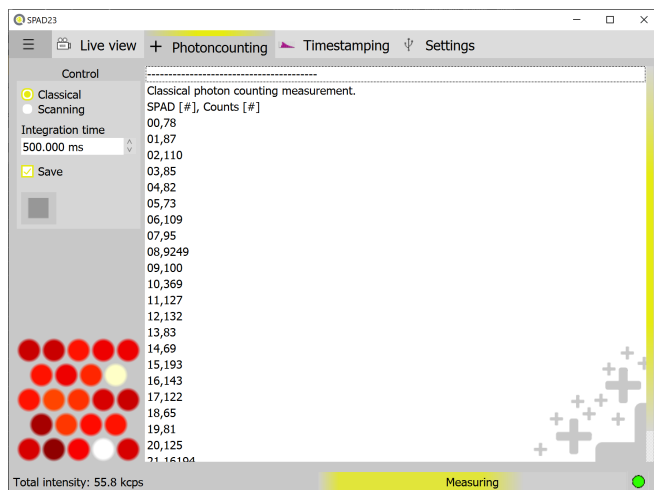
## Live view

The live view tab shows a graphical representation of the current photon counts on the 23 SPAD pixels. The view is adjusted proportionally to the brightest pixel, which is completely white. A black–red–yellow–white colour scale shows the intensity variation over the array. Clicking on one of the pixels in the live view will disable the pixel's visual output. Hovering over any of the pixels will reveal its current count rate in counts per second (cps) in the bottom status bar. Otherwise the integrated photon count rate over the whole array will be displayed.

A small version of the live view is also available on the photoncounting and timestamping tabs in the bottom left corners.



## Photoncounting

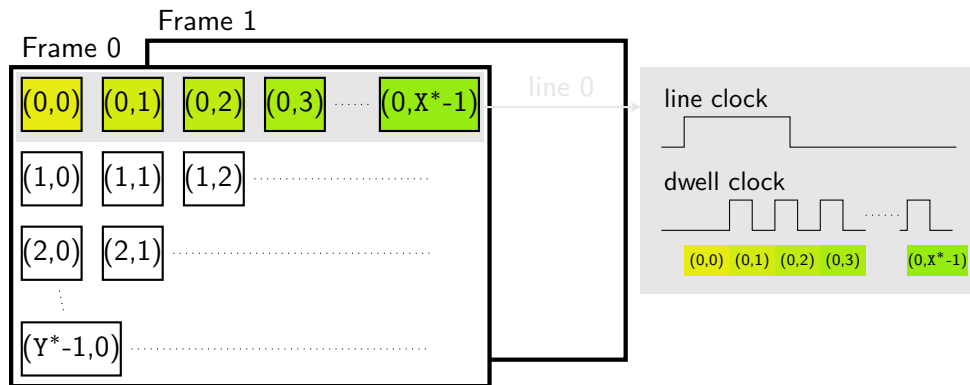


The second tab provides the photoncounting capabilities of the system. On the left is a control panel to setup the parameters for the measurements that are printed in the window on the right. The measurement progression is indicated in the bottom status bar. The following options are available:

- Classical button  
Set the measurement window for classical single-shot intensity measurements.
- Scanning button  
Set to scanning mode, this mode allows to continuously measure with the help of the provided line / dwell clocks on the systems SMA connectors. This mode does not function without a line clock!
- Integration time  
Set the measurement integration time either in milliseconds for classical operation mode, or microseconds for scanning operation mode.
- The following options are only available in scanning operation mode:
  - Frames  
Set the number of frames.
  - X  
Set the number of pixels or dwells in a single line.
  - Y  
Set the number of lines.
  - Ext. dwell clk checkbox  
Enable the use of the external dwell clock provided through the SMA connector and override the internal integration time.

- Image checkbox  
Enable the conversion of the raw data to a set of images.
- Non-scaled checkbox  
By default the images will be scaled with respect to the brightest pixel. This box disables scaling. The non-scaled value represents the photon counts with 8-bit counter depth.
- Raw data checkbox  
Process and print/save the raw data in text format.
- Save checkbox  
Saves the measurements to a file or an image. A folder data/counters is created in the run directory and files are saved with the name Run\_classicalXXX.txt, Run\_scanningXXX.txt or IMGXXX-YY.png. XXX denotes the run iteration, YY is the pixel number 0–22 and 23 for the combined result.
- Start button  
Launches the chosen measurement.

The standard measurement data is formatted with the first column being the pixel number and the second being the measured counts in the given measurement time. The scanning measurement principle is shown below. The rising edge of the line clock starts the integration of the first dwell of that line, each consecutive rising edge of the dwell clock increments the dwell counter by one. At the end of the line, the dwell counter is reset to zero and the system waits until the next rising edge of the line clock. Once the system has integrated the requested number of lines, the frame counter is incremented and the procedure starts from line zero again. The frame clock marker is unused in the current implementation.



The scanning data is formatted as (Frame number Z, Line number Y, Dwell number X), 23 pixel count values. See below for an example with  $Z^*=2$ ,  $Y^*=64$ ,  $X^*=128$ .

```
(0,0,0),1,2,2,3,4,2,3,3,0,0,3,0,1,3,1,2,3,3,0,1,4,3,2
(0,0,1),6,1,1,1,3,3,7,1,1,0,1,1,3,2,2,2,2,3,4,1,3,1,5
(0,0,2),2,2,2,2,3,0,3,5,4,0,1,3,1,2,2,0,2,3,0,0,6,1,4
...
(1,63,126),0,2,0,4,2,3,2,4,2,0,2,0,2,0,2,1,1,2,1,4,0,2,1
(1,63,127),0,2,1,2,1,3,1,1,1,0,1,3,2,0,1,5,2,1,1,3,6,1,1
```

The generated image with a size of  $X^* \times Y^*$  is displayed on the right hand side of the screen. One can scroll through the frames with the slider on top of the image or play the sequence with the play button. By default, a scaled image with the summation of the pixel counts from all 23 detectors is shown. The generated image for each individual SPAD detector can be displayed by clicking on the pixel in the mini-liveview in the left side of the window. When saving, all 23+1 images will be generated in the data/counters folder for each frame.

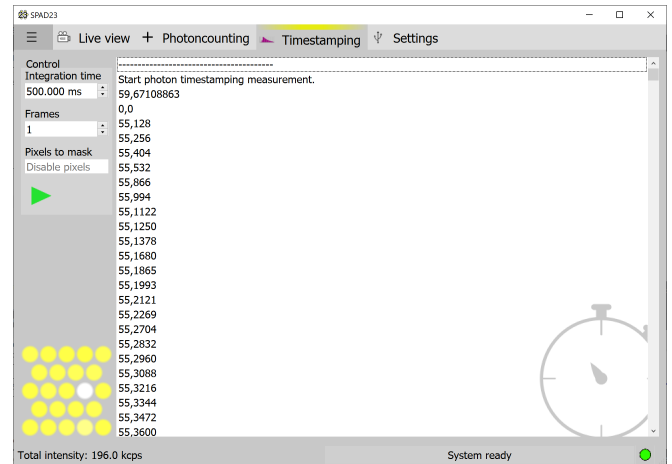
## Timestamping

The third tab gives access to the timestamping capabilities. This interface is adapted according to the detected firmware version. The first system version features coarse (10 ns) timestamping. The second has time-to-digital converters (TDCs) with a fine resolution ( $\sim 10$  ps).

## Coarse system

The control panel on the left has the following options:

- **Integration time**  
Set the measurement integration time in milliseconds. Setting a time equal to zero will start an unlimited time measurement. The measurement can be stopped with a click on the stop button.
- **Pixels to mask**  
Set the pixels that are not required, separated with a comma or semicolon: 0,1,2,...,22.
- **Image checkbox**  
Generate an image for each of the 23 pixels + the combined results (currently the images contain just the intensity information). The 24 images for each frame are written into a new folder `data/tdc/RunXXX`. Note that this mode requires a dwell, line and frame trigger to operate successfully.
- **Raw data checkbox**  
Save the raw data directly in a new folder `data/timers/RunXXX`.
- **Start button**  
Launches the chosen measurement.



The progress of the measurement and the processing of the data is shown in the bottom status bar. The measurements are saved to separate files per pixel. A folder `data/timers/RunXXX` is created in the run directory and files are saved with the name `pixXX.txt`. Special markers and additional information is saved in `pix23.txt`, this information is also printed in the right side of the window.

The pixel data is formatted as a single column with the value of the coarse clock counter at the photon arrival time. The real-time streaming data is formatted slightly different, see the Command guide.

The marker data is formatted with the first column being the marker or special event number, which has to be read as follows:

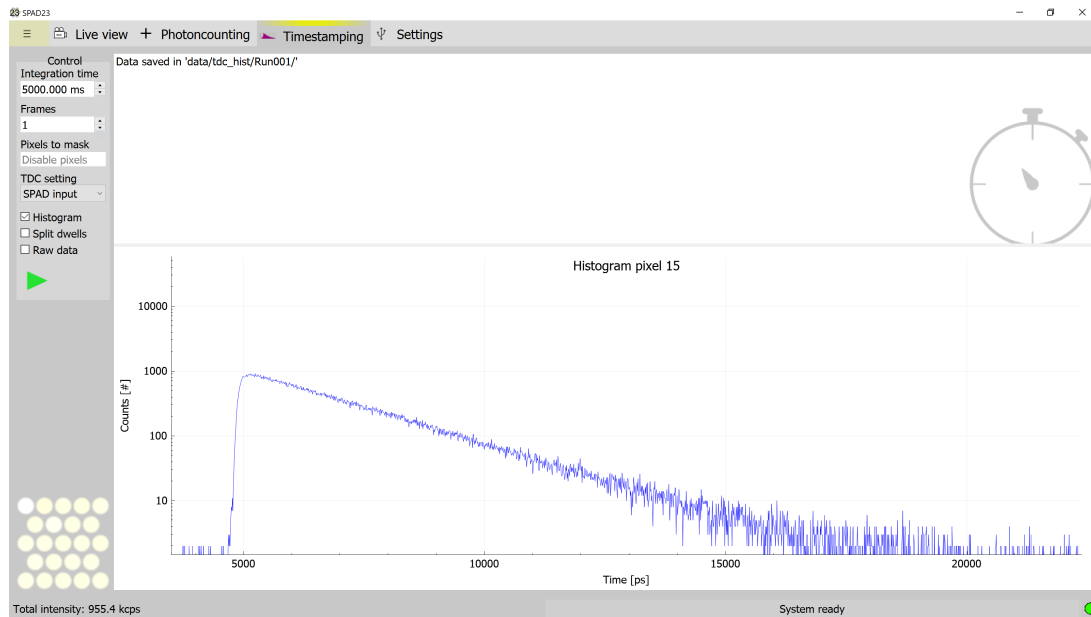
- -59 – File header
- -55 or -56 – Dummy pixel, with  $\sim 1$  Mcps count rate in low light conditions, used to track coarse clock counter wraparounds, only the wraparound data is printed
- -17 – USB FIFO almost full, send only markers
- -16 – USB FIFO reasonably empty, resume normal operation
- -12 – Frame marker
- -10 – Line marker
- -9 – Dwell marker
- -0 – End of measurement

The second column is the value of the coarse clock counter running at 100.8 MHz, i.e. this allows to determine the arrival time down to 10 ns.

## TDC system

The TDC version features additional options compared to the coarse timing version. Main differences are highlighted in this section.





Additional options in the control panel:

- TDC setting dropdown box  
The operation of the TDC can be selected here, there are three modes.
  - Calibrate, runs the calibration of the TDCs. Measures the length of the TDC channels and stores them for further measurements.
  - SPAD input, normal operation.
  - Align pixels, requires a synchronous light source, measures the histograms of all pixels and calculates the rising edge offsets in each TDC channel for alignment of further measurements. The table is stored and loaded on startup.
- Clock setting dropdown box (if the system has a laser clock output, see the log in the settings tab for additional information)  
Select the clock frequency for the laser in  $2^n$  divisions from 40 MHz (default). Currently available frequencies: 40/20/10/5/2.5/1.25/0.625 MHz.
- Histogram checkbox  
Generate and save a histogram from the measurement data for each pixel (the histogram will update at real-time). The data is saved directly in a new folder `data/tdc_hist/RunXXX` and the histogram is plotted in the right window. Other pixels can be selected by clicking in the mini-liveview on the left. The data is formatted with the first row being the time in picoseconds, all further rows contain the number of counts in that time window and per dwell (if enabled). For newer systems, the first row is saved separately in `data/tdc_hist/RunXXX/pixtimeaxis.txt` and is no longer saved in each pixel file to save space.
- Split dwells checkbox  
Split the histograms for each new dwell. A slider will be shown to cycle through the dwells. In the output file, the histogram for each dwell will be printed on a new line. Note that this might cause some lag for a dwell amount  $>2000$ .
- Image checkbox  
Generate an image for each of the 23 pixels + the combined results (currently the images contain just the intensity information). The 24 images for each frame are written into a new folder `data/tdc/RunXXX`. Note that this mode requires a dwell, line and frame trigger to operate successfully.
- Raw data checkbox  
Save the raw data directly in a new folder `data/tdc/RunXXX`.

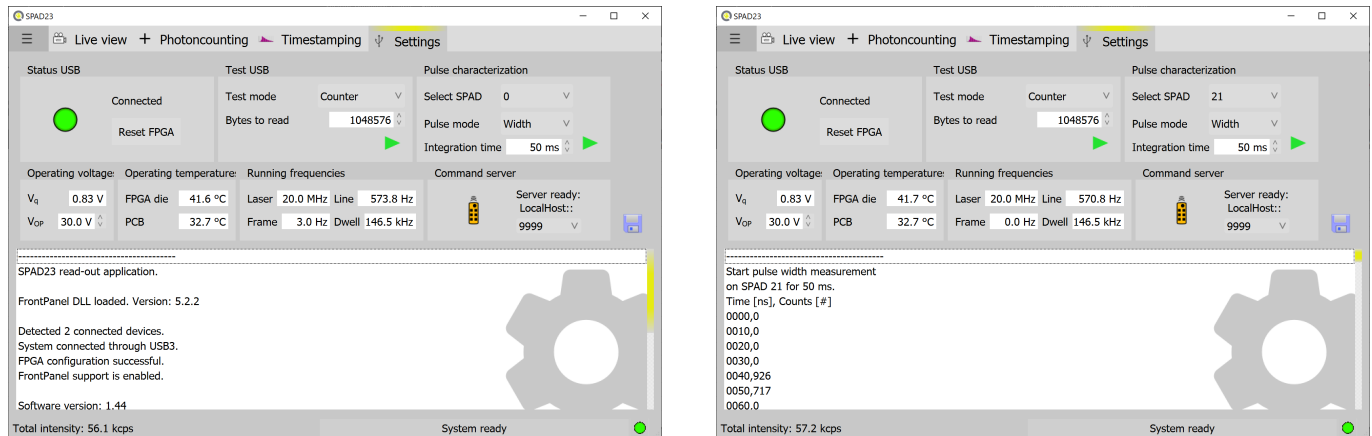
Before the first measurement, the TDC needs to be calibrated. The software is already setup in calibration mode, which can be run by simply clicking the play button.

The pixel data is formatted in 2 columns. From left to right they represent:

- The converted TDC result (in picoseconds) with respect to the laser clock.
- The coarse counter value, synchronous with the supplied laser clock.

The format of the marker data is unchanged with respect to the coarse counting system, with the exception of the coarse counter being shorter. Note that the data format for the TCP/IP stream option is slightly different, see the remote command guide section.

## Settings



The last tab shows the system settings. The top half is divided in 7 boxes:

- Top left  
Status indication of the USB connection. This is indicated green when the system is connected by USB3. The system can be reinitialized by clicking the reset button. The software gives a warning when the system is not connected with USB3.
- Top middle  
Speed test for the USB connection. The transfer speed is calculated by transferring the set number of bytes and dividing by the required time.
- Top right  
Pulse width (i.e. the SPAD dead-time) and pulse distance measurements. On the FPGA, a histogram is generated during the set integration time for the selected SPAD pixel. The data is formatted with the first column being the time in nanoseconds, the second being the number of counts in that time window. The data can be saved with the save button into a file RunXXX.txt in a new data/debug folder. With the default operating voltages, the average pulse width should be in the order of 50 ns.
- Bottom left  
Current operating voltages. The operating voltages can be changed on the fly.  $V_q$  will automatically be calculated upon changes in  $V_{OP}$  to tune the pixel dead-time to a value in the order of 50 ns.
- Bottom middle (I)  
Current operating temperatures of the FPGA and the PCB on which the sensor is located. Therefore, the PCB temperature should be a good indication of the sensors current operating temperature. Ideally, the sensor temperature should be kept as low as possible. Without air flow, the PCB temperature should be below 40°C.
- Bottom middle (II)  
Current SMA input frequencies. Shows the current running frequency for the SMA inputs: laser, frame, line, dwell clock.
- Bottom right  
Status of the remote communication server. The software can be programmed from any capable TCP/IP software package, see the command guide for instructions. The listening port on the localhost can be changed through the dropdown menu.

## 4 Remote command interface

The software can be operated over TCP/IP with a simple command protocol. The software is listening for new remote users on the PCs localhost port 9999 by default. This port can be changed on the settings tab. The following sections list the available commands and show some programming examples.

### Command guide

All commands start with capital characters, followed by its parameters. Parameters indicated in triangle brackets `< >` should be replaced by a numerical value, parameters in square brackets `[ ]` should be replaced by a string.

- Classical intensity measurement

```
I,<measurement time in ms>
```

Command returns the intensity data from the 23 pixels. It is a list of 23 lines with on each line `<pixel nr.>,<pixel counts>`.

- Scanning intensity measurement

```
C,<measurement time in us>,<frames>,<X-elements>,<Y-elements>
```

```
CI,<measurement time in us>,<frames>,<X-elements>,<Y-elements>
```

If `<measurement time> = 0`, the external dwell clock is used.

Command `C` returns the path to the stored raw data file.

Command `CI` returns the paths to the stored images. The software will store 23 non-scaled images for each pixel and the combined result for each measured frame.

- Timestamping measurement

```
T,<measurement time in ms>
```

Command returns the path to the stored data files.

- Additional timestamping and TDC commands:

- Set pixels to mask

```
T,m,[pixels]
```

`[pixels]` is a semicolon separated string with the pixel numbers to mask, e.g. `2;10;18`.

- Enable or disable the raw data saving functionality

```
T,r,<enable>
```

`<enable>` 0 for disable, 1 for enable.

- Enable or disable the image generation functionality

```
T,i,<enable>
```

`<enable>` 0 for disable, 1 for enable.

- [TDC version] Enable or disable the histogram functionality

```
T,h,<enable>
```

`<enable>` 0 for disable, 1 for enable.

- [TDC version] Perform a new calibration

```
T,c,1
```

Command returns when calibration completed.

- [TDC version] Check if the TDC is calibrated

```
T,v,1
```

Command returns the state of the system's calibration.

- Timestamping measurement in real-time streaming mode

```
S,<measurement time in ms>
```

Command returns a list of the timestamps in real-time. Format is string-based:

[TDC version] `<pixel nr.+32 / marker nr.>,<coarse counter value>,<TDC value>`

[Base version] `<pixel nr.+32 / marker nr.>,<coarse counter value>`

```
SB,<measurement time in ms>
```

Command returns a list of the timestamps in real-time. Format is binary-based:

[TDC version] 1-byte for pixel nr.+32 / marker nr. | 2-bytes (forming a 16-bit integer) for the coarse counter value | 4-bytes (forming a 32-bit integer) for the TDC value

[Base version] 1-byte for pixel nr.+32 / marker nr. | 4-bytes (forming a 32-bit integer)

for the coarse counter value

- Pulse width or distance measurement

P,<measurement time in ms>,<pixel>,<pulse mode>

<pixel> is the SPAD pixel number from 0 to 22.

<pulse mode> is either 0 for pulse width, or 1 for pulse distance.

Command returns the path to the stored data file.

- Get current operating temperatures

R

Command returns the temperatures of the FPGA and PCB in °C. Format is <FPGA temperature>,<PCB temperature>

- Set voltages V,<Vq>,<VOP>

<Vq> range is 0.4 to 0.9 V. This value will be ignored in case the automatic tuning of  $V_q$  is active.

<VOP> range is 26 to 31.5 V.

Command returns a confirmation if changes are successful.

## Programming examples

Below follow some programming examples for Matlab/Octave/Python. The first one measures the count rates at different voltages. The second one measures the count rates versus the integration time. The third one performs a timestamping measurement when the TDC is enabled. The last one shows the capability of real-time streaming of photon timestamps.

### Code example 1: Matlab – Count rate versus operating voltage

```
1 % open the device on the localhost, port 9999
2 t = tcpip('localhost', 9999);
3 fopen(t);
4
5 % read the server response
6 bytesav = get(t, 'bytesavailable');
7 msg = fread(t, bytesav);
8 msgstr = convertCharsToStrings(msg)
9
10 % set the voltage sweep for VOP that we want to execute from 26 to 32 V
11 voltages = 26:0.5:31.5;
12 counts = zeros(size(voltages));
13 for i = 1:length(voltages)
14
15     % set the voltages and wait for the acknowledgment
16     fwrite(t, ['V,' num2str(0.8) ',' num2str(voltages(i))]);
17     bytesav = get(t, 'bytesavailable');
18     while bytesav == 0
19         pause(0.1)
20         bytesav = get(t, 'bytesavailable');
21     end
22     msg = fread(t, bytesav);
23
24     % start an intensity measurement for 250 ms and read the data
25     fwrite(t, ['I,' num2str(250.0)]);
26     bytesav = get(t, 'bytesavailable');
27     while bytesav == 0
28         pause(0.1)
29         bytesav = get(t, 'bytesavailable');
30     end
31     msg = fread(t, bytesav);
32
33     % convert the data to numerical values
34     msgstr = convertCharsToStrings(msg);
35     msgstrspl = textscan(msgstr, '%s', 'delimiter', '\n');
36     msgstrspl = cellfun(@(s) textscan(s, '%f', 'delimiter', ','), {msgstrspl{1:end}}, 'UniformOutput', false);
```

```

37     msgvals      = cell2mat(cell2mat(msgstrspl));
38
39     % take the mean of the 23 counts
40     counts(i)    = mean(msgvals(2,:));
41
42 end
43
44 % close communication channel
45 fclose(t)
46
47 % plot the data and convert the count rate to counts per second [cps]
48 figure
49 plot(voltages, counts*4)
50 xlabel("Voltage [v]")
51 ylabel("Counts [cps]")

```

### Code example 2: Python – Count rate versus integration time

```

1  #!/usr/bin/env python
2
3  import socket
4  import matplotlib.pyplot as plt
5  import numpy as np
6
7  # open the device on the localhost, port 9999
8  t = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
9  t.connect(('127.0.0.1', 9999))
10
11 # read the server response
12 data = t.recv(8192)
13 print(data.decode('utf8'))
14
15 # set the measurement times sweep from 50 to 500 ms
16 integration_times = np.arange(50, 550, 50)
17 counts = np.zeros(len(integration_times))
18 iteration = 0
19 for i in integration_times:
20     # start an intensity measurement for i ms and read the data
21     command = bytes("I," + str(i), "utf8")
22     t.send(command)
23
24     data = t.recv(8192) # get the data
25     datastr = data.decode('utf8') # decode into a string
26     datasplit = datastr.split("\n") # split the lines
27     for j in datasplit:
28         # get the counts for each pixel and add them together
29         readdata = j.split(",")
30         counts[iteration] += int(readdata[1])
31
32     iteration += 1
33
34 # make a plot of the integration time versus the count rate
35 plt.plot(integration_times, counts) # this should be a pretty linear line
36 plt.title('Count rate vs. integration time')
37 plt.xlabel('Integration time [ms]')
38 plt.ylabel('Counts [#]')
39 plt.grid(True)
40 plt.show()
41
42 # close communication channel
43 t.close()

```

### Code example 3: Octave – TDC calibration and measurement

```

1 % load the instrument control package

```

```

2 pkg load instrument-control
3
4 % open the device on the localhost, port 9999
5 t = tcpclient('localhost', 9999);
6 fopen(t);
7
8 % read the server response
9 bytesav = get(t, 'bytesavailable');
10 msg = fread(t, bytesav);
11 msgstr = native2unicode(msg)
12
13 % check if the TDC calibration is done, useful for consecutive measurements, otherwise
    calibrate
14 fwrite(t, ['T,v,1']);
15 pause(0.1)
16
17 bytesav = get(t, 'bytesavailable');
18 while bytesav < 20
19     pause(0.1)
20     bytesav = get(t, 'bytesavailable');
21 end
22 msg = fread(t, bytesav);
23 msgstr = native2unicode(msg);
24
25 if strcmp(msgstr, "TDC calibration is invalid")
26     % do the TDC calibration and wait till its finished
27     fwrite(t, ['T,c,1']);
28     pause(1)
29
30     bytesav = get(t, 'bytesavailable');
31     while bytesav < 30
32         pause(0.1)
33         bytesav = get(t, 'bytesavailable');
34     end
35     msg = fread(t, bytesav);
36     msgstr = native2unicode(msg);
37 end
38
39 % execute a TDC measurement with pixels 5 and 8 masked in histogram mode for 2 s
40 fwrite(t, ['T,m,5;8']);
41 pause(0.1)
42 fwrite(t, ['T,h,1']);
43 pause(0.1)
44 fwrite(t, ['T,i,0']);
45 pause(0.1)
46 fwrite(t, ['T,r,0']);
47 pause(0.1)
48 fwrite(t, ['T,2000.0']);
49
50 % wait for the measurement to be finished
51 bytesav = get(t, 'bytesavailable');
52 while bytesav == 0
53     pause(0.1)
54     bytesav = get(t, 'bytesavailable');
55 end
56 msg = fread(t, bytesav);
57 msgstr = native2unicode(msg);
58
59 % close communication channel
60 fclose(t)
61
62 % get the path from the server response
63 msgstr = textscan(msgstr, '%s', 'delimiter', "'");
64 path = msgstr{1}{2}
65

```

```

66 % load the data from the path and process the data
67 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

```

#### Code example 4: Python – Real-time streaming of binary timestamps

```

1  #!/usr/bin/env python
2
3  import socket
4  import matplotlib.pyplot as plt
5  import numpy as np
6  import struct
7  import time
8
9  # open the device on the localhost, port 9999
10 t = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
11 t.connect(('127.0.0.1', 9999))
12
13 # read the server response
14 data = t.recv(8192)
15 print(data.decode('utf8'))
16
17 # set all pixels to be active by masking a non-existing pixel
18 command = bytes("T,m,23", "utf8")
19 t.send(command)
20 time.sleep(0.1)
21
22 # check if the calibration of the TDCs has completed
23 command = bytes("T,v,1", "utf8")
24 t.send(command)
25 data = t.recv(8192)
26
27 # if the TDC is not calibrated, execute the calibration command
28 if data.decode('utf8') == "TDC calibration is invalid":
29     command = bytes("T,c,1", "utf8")
30     t.send(command)
31     data = t.recv(8192)
32     print(data.decode('utf8'))
33
34
35 start = time.time()
36 # do a stream of the data
37 command = bytes("SB," + str(10000), "utf8")
38 t.send(command)
39
40 counts = [0]*23 # create a list of int's to get the count rates
41 timestamps = [[] for _ in range(23)] # create a list of lists for each pixels timestamps
42 # look for the response
43 while 1:
44     data = t.recv(57344) # try different buffer sizes with multiples of 7
45
46     for i in range(0, len(data)-7, 7):
47         readdata_pixelnr = data[i+0]
48         if readdata_pixelnr > 31 and readdata_pixelnr < 55:
49             # readdata_coarsecount = int.from_bytes([data[i+1],data[i+2]], 'big') # coarse
50             counter timestamp with respect to the system clock (100 MHz or 40 MHz)
51             readdata_finecount = int.from_bytes([data[i+3],data[i+4],data[i+5],data[i+6]],
52             'big') # fine timestamp in picoseconds with respect to the laser clock
53
54             # sum to get the countrate
55             counts[readdata_pixelnr-32] += 1
56
57             # store the fine timestamps for pixel 0
58             if readdata_pixelnr-32 == 0:
59                 timestamps[readdata_pixelnr-32].append(readdata_finecount)

```

```

59     # elif readdata_pixelnr == 55 or readdata_pixelnr == 56:
60         # this is dummy pixel data for low-intensity events
61         # one can use this information to keep track of absolute time
62
63     # elif readdata_pixelnr == 9:
64         # this is a dwell marker event
65
66     # elif readdata_pixelnr == 10:
67         # this is a line marker event
68
69     # elif readdata_pixelnr == 12:
70         # this is a frame marker event
71
72     elif readdata_pixelnr == 17:
73         # this is a fifo overflow event, one should take care if this happens
74         print("FIFO overflow")
75
76     if data[-4:] == bytearray("DONE", 'utf8'):
77
78         print("Process complete")
79         break
80
81 # close communication channel
82 t.close()
83 read = time.time()
84 print("Read time: ", "{:.2f}".format(read - start), " s")
85
86 # print the count rates of all pixels
87 pix = 0
88 total_counts = 0
89 for i in counts:
90     print(pix,',',i)
91     pix += 1
92     total_counts += i
93
94 print("Total counts:",total_counts)
95
96 # make a histogram from timestamps for pixel 0
97 time_max = max(timestamps[0])
98 time_axis = np.arange(0, time_max+10, 10)
99 histogram = np.zeros(len(time_axis))
100
101 for i in timestamps[0]:
102     histogram[round(i/10)] += 1
103
104 # merge first and last bins
105 histogram[0] += histogram[-1]
106
107 end = time.time()
108 print("Elapsed time: ", "{:.2f}".format(end - start), " s")
109
110 # create a plot to show the histogram
111 plt.plot(time_axis[:-1], histogram[:-1]) # ignore the last bin
112 plt.title('Histogram of pixel 0')
113 plt.xlabel('Time [ps]')
114 plt.ylabel('Counts [#]')
115 plt.xlim([0, time_max])
116 plt.grid(True)
117 plt.show()

```

