Hardware simulator for photon correlation spectroscopy

Fabio Ferri and Davide Magatti

Universita’ dell’Insubria a Como and INFM, Dipartimento di Scienze, Chimiche, Fisiche e Matematiche, Via Valleggio 11, I-22100 Como, Italy

(Received 2 April 2003; accepted 16 June 2003)

We present a hardware simulator ideal for testing digital correlators in photon correlation spectroscopy. By using a PCI-6534 National Instrument I/O board, a personal computer (1.5 GHz Pentium 4), and an original algorithm developed in LabVIEW (National Instrument™), we realized an instrument capable of delivering a continuous stream of transistor-transistor logic pulses with the desired statistical properties over one or more channels. The pulse resolution could be set to values multiple of the clock period $\Delta t = 50$ ns available on the board. When a single channel is used, the maximum count rate at $\Delta t = 50$ ns was $\langle I \rangle \sim 350$ kHz. With two channels we obtained $\langle I \rangle \sim 80$ kHz at $\Delta t = 50$ ns and $\langle I \rangle \sim 120$ kHz at $\Delta t = 100$ ns. Pulse streams with Gaussian statistics and in the presence of shot noise were simulated and measured with a commercial hardware correlator. Photodetector defects, such as the presence of afterpulses, were also simulated and their elimination by cross correlation techniques was checked. The simulator works also as a general purpose pulse pattern generator (PPG). Compared with commercial PPGs, our simulator is slower, but permits a continuous output of the pulse stream (not allowed in PPGs). At the same time it offers many other nontrivial advantages related to its flexibility, relatively low cost, and easy adaptability to future technology developments. © 2003 American Institute of Physics. [DOI: 10.1063/1.1602934]

I. INTRODUCTION

Photon correlation spectroscopy (PCS)1–3 is an optical technique based on the measure of the temporal autocorrelation function of the light intensity scattered by a sample illuminated with a laser light. PCS has been used for decades for studying the dynamics of many physical, chemical, and biological systems, which are of interest in both basic and applied science. Examples of applications of PCS are countless, going from the classical fields of laser doppler velocimetry and particle sizing of submicron particles, to the investigation of intriguing new systems (whose dynamics is characterized by a large polydispersity of relaxation times) such as colloidal or polymeric gels, complex fluids, foams, granular materials, etc. A recent review on the historical applications of PCS, as well as on the recent developments and experimental work carried out in this field, can be found in Refs. 4 and 5 and references therein.

In a PCS experiment, the signal correlation function is typically carried out by using a digital correlator, i.e., a device capable of performing the on-line digital signal processing (DSP) of the pulse stream coming from a photodetector, usually a photomultiplier or an avalanche photodiode. With the help of the so-called multi-tau scheme of sampling times,7 digital correlators can work out correlation functions in real time over a huge range of lag times, from $\sim 10$ ns to hours.

Digital correlators are nowadays available as hardware or software correlators. The first ones are powerful electronic devices that carry out the computation of the signal correlation function via hardware. They are easily installable into a personal computer (PC) and are usually provided with non-trivial software libraries for data analysis, such as, for example, the well known CONTIN algorithm.8 Hardware correlators are nowadays commercially available on the market, the most popular ones probably being those manufactured by ALV (Langen, Germany) and by Brookhaven Instrument (Holtsville, NY), and in more recent years by Correlator.com (Bridgewater, NJ). More recently, also fast software correlators were developed and were shown to be competitive with the hardware ones.9,10 In software correlators the signal coming from the photodetector is acquired with a standard fast counter and the correlation function is computed in real time via software. Although software correlators are still somewhat less powerful than hardware correlators, they are prone for fast development and have the further advantage of being much more flexible and less expensive.

In order to check the proper functioning of digital correlators and compare their performances, it is necessary to carry out extensive and accurate testing.11 Usually, these tests are performed by using calibrated samples of latex spheres of different diameters and measuring their correlation functions at different count rates. Although this procedure is well established, it presents several limitations and may lead to wrong or unclear results. For example, uncertainties may derive from problems associated to sample preparation (presence of dust or particle aggregation), or could be introduced by photodetector defects (dead time or afterpulse), or be due to systematic errors introduced by an improper functioning of the optical setup (misalignment, or straight light, or multiple scattering).

In this work, we have realized a fast hardware simulator ideal for testing digital correlators, free of all the limitations mentioned above. By using a commercial I/O board (Na-
II. GENERATION OF SYNTHETIC DATA

The synthetic data describing a typical PCS experiment were generated by following a scheme very similar to the one reported in Refs. 13 and 14 and outlined in Fig. 1. Let us start by recalling (see, for example, Ref. 15) that, when a stochastic variable $y(t)$ characterized by a probability Gaussian distribution with a given average $\langle y \rangle$, standard deviation $\sigma_y$, and correlation function $g_y(\tau)$, is fed through a linear filter $G$, the resulting stochastic variable $z(t)$,

$$
z(t) = \int_{-\infty}^{+\infty} y(t') G(t-t') dt' = y*G,
$$

is still characterized by a Gaussian distribution and by a correlation function $g_z(\tau)$ which is given by

$$
g_z(\tau) = g_y(\tau) * g_G(\tau),
$$

where $g_G(t)$ is the correlation product of $G$, i.e., $g_G(t) = \int_{-\infty}^{+\infty} G(t') G(t+t') dt$. If the data $y(t)$ are $\delta$-correlated and $G$ is the impulsive response function of a low-pass filter characterized by a decay time $t_0$,

$$
G(t) = \frac{e^{-t/t_0}}{t_0} \quad (t \geq 0),
$$

$$
G(t) = 0 \quad (t < 0),
$$

then the variable $z(t)$ exhibits an exponential decay correlation function and is characterized by an average $\langle z \rangle = \langle y \rangle$.

In practice, we can use the random number generator available in a PC as follows. We synthesize the function $y(t)$ by generating a stream of random numbers $y_i$, according to a Gaussian distribution, so that

$$
y(t) = y_i, \quad \Delta t(i-1) < t < \Delta ti, \quad i = 1,2,\ldots,\infty,
$$

where $\Delta t$ is the (arbitrary) time interval between every launch. By inserting Eq. (4) into Eq. (1), using Eq. (3) and sampling the function $z(t)$ at the discrete times $t_i = \Delta ti$, it is straightforward to show that

$$
z_i = [1 - e^{-\gamma}] \sum_{j=0}^{\infty} y_{i-j} e^{-\gamma j},
$$

where $\gamma = \Delta t/t_0$, and the factor $[1 - e^{-\gamma}]$ ensures that

$$
\langle z \rangle = \langle y \rangle.
$$

It is immediate to show that Eq. (5) can rewritten in the recursive way

$$
z_{i+1} = [1 - e^{-\gamma}] y_{i+1} + z_i e^{-\gamma},
$$

which is very useful for iterative generation. From Eq. (5) one can easily show that the standard deviation, $\sigma_z^2 = \langle z^2 \rangle - \langle z \rangle^2$, is given by

$$
\sigma_z^2 = \frac{[1 - e^{-\gamma}]^2}{\left[1 - e^{-2\gamma}\right]} \sigma_y^2,
$$

where the fact that the $y_i$ are uncorrelated and Eq. (6) has been used. Note that, when $\gamma \ll 1$, Eq. (8) becomes $\sigma_z^2 = \sigma_y^2 e^{-2\gamma}$, from which one sees that the convolution filter dampers the squared fluctuations of the variable $y(t)$ by a factor $\Delta t/2t_0$. Finally, the correlation function of the variable $z$ is given by

$$
g_z(\tau_k) = \langle z_i z_{i+k} \rangle = \langle z^2 \rangle + \sigma_z^2 e^{-\gamma k}, \quad k = 1,2,\ldots,
$$

which exhibits the expected exponential decay as a function of the discrete lag times $\tau_k = \Delta tk$.

We want now to use the framework outlined above for generating synthetic intensity data for PCS. As is known,\textsuperscript{16} in a PCS experiment, the electric field $E$ scattered by the sample can be represented as a vector performing a “random walk” in two dimensions, in which the two components of...
the field, \( E_x \) and \( E_y \), are independent stochastic variables characterized by the same Gaussian probability distribution with \( \langle E_x \rangle = \langle E_y \rangle = 0 \) and \( \sigma^2_{E_x} = \sigma^2_{E_y} = (I)/2 \). Here \( I \) represents the average intensity scattered within a coherence area and is related to the electric field by \( I = |\mathbf{E}|^2 \). Thus for obtaining a given \( I \), \( E_x \) and \( E_y \) are generated by using Eq. (7) with \( \langle y \rangle = 0 \) and \( \sigma_y \) chosen according to Eq. (8) so that \( \sigma^2_y = (I)/2 \). The data stream for the intensity \( I \) is therefore

\[
I_i = \langle E_x \rangle^2 + \langle E_y \rangle^2.
\]

(10)

According to the Siegert relation,\(^7\) the new data \( I_i \) are characterized by a correlation function \( g_{I_i} \) given by

\[
g_{I_i}(\tau_i) = \langle I \rangle^2 [1 + e^{-\pi \tau_i}],
\]

(11)

in which the decay time \( \tau_i \) is half of the field decay time, i.e., \( \tau_i = \tau_0/2 \).

If we express \( I \) as the average number of photons scattered per unit time, the number \( R_t \) of photons falling within the \( i \)th interval \( \Delta t \) is simply \( R_t = I_i \Delta t \). Typically \( R_t \) is much smaller than unity and, consequently, it represents the probability of having one photon per time interval. For example, if \( \Delta t = 10 \) ns and \( \langle I \rangle = 10^6 \) Hz, \( R_t \sim 10^{-2} \). In order to obtain from \( R_t \) an integer number \( N_t \) which represents the number of photocounts in the \( i \)th interval \( \Delta t \), it is necessary to pass \( R_t \) through a “Poisson filter” as shown in Fig. 1. This is simply realized by generating a random number according to a Poisson distribution characterized by an average value equal to \( R_t \). Since \( R_t \ll 1 \), the integers \( N_t \) are either zero or one, and “pile-up” effects\(^17\) can be neglected. Indeed, the probability of \( N_t \geq 2 \) is quite small, equal (for an uncorrelated signal) to \( \sim \langle R \rangle^2/2 \sim 5 \times 10^{-5} \) \((\langle R \rangle = 10^{-2})\).

Once the photocounts \( N_t \) have been generated, we can also simulate the defects of a real photodetector by introducing some systematic or stochastic noise in the sequence of \( N_t \). For example, it is straightforward to simulate the dead time of the detector by zeroing all the counts occurring before a given time is elapsed since the last count. Similarly, it is easy to simulate the presence of afterpulses. This can be realized by assigning the conditional probability of having an afterpulse whenever a count occurred, and generating an extra count at a given temporal distance from the previous one. An example of the presence (and of the cure) of afterpulses will be given in Sec. IV.

The last step in the simulation is transforming the sequence of zeros and ones in integrated arrival times and writing them onto the hard disk (HD). Storing the arrival times instead of zeros and ones is mandatory because, otherwise, the size of the zeros/ones file would be much bigger and, most important, it would require a much longer time to be read (see Sec. III). For example, if we simulate a pulse stream with \( \Delta t = 10 \) ns output clock at a count rate \( I \) = \( 10^6 \) Hz for a measuring time of \( T = 100 \) s, the overall number of zeros/ones would be \( T/\Delta t \times 10^4 \), and the corresponding file would be \( \sim 10 \) GB in size. Conversely, the number of arrival times would be smaller by a factor of \( I/\Delta t \times 10^{-2} \) and equal to \( 10^3 \), with a file of the order of \( \sim 1 \) GB. Finally, the option of storing integrated (rather than differential) arrival times is fairly important because it facilitates significantly the procedure of transforming back the arrival times into a sequence of zeros and ones, as it will be described in the next section.

Summarizing, the recipe for generating the synthetic data can be outlined in the following points:

1. set the clock period \( \Delta t \), the average count rate \( \langle I \rangle \), and the decay time \( \tau_0 \), thus \( \gamma = \Delta t/\tau_0 \);
2. determine \( \sigma_y \) according to Eq. (8), in which \( \sigma^2_y = (I)/2 \);
3. generate \( (E_x) \) and \( (E_y) \) according to Eq. (7), in which \( y_i \) are random numbers characterized by Gaussian distribution with standard deviation \( \sigma_y \) and average \( \langle y \rangle = 0 \);
4. generate \( I_i \) according to Eq. (10);
5. calculate \( R_i = I_i \Delta t \) and pass \( R_i \) trough a “Poisson filter” with a average count equal to \( R_i \) (for realizing a Poisson filter see, for example, Ref. 18). The output of the Poisson filter is a integer count \( N_i \), \( (0,1,2,...) \);
6. if desired, introduce photodetector defects (after pulse, dead time) as described above;
7. if \( N_i = 0 \), no pulse is present. Return to point 3 for the next step;
8. if \( N_i = 1 \), one pulse is present. Its arrival time is computed by counting the number of steps elapsed since the occurrence of the last pulse. The corresponding integrated arrival time \( T_M = \Sigma_{k=1}^{M} \tau_i \) is recorded on the hard disk;
9. if \( N_i \geq 1 \), two or more pulses are present. Due to the negligible “pile up” associated to our working conditions (see text above), this case is highly unlikely and it is handled exactly as point 8, i.e., by discarding the extra pulse(s);
10. return to point 3 for the next step.

III. SIMULATOR ARCHITECTURE

The simulator uses a I/O board (PCI-6534, National Instrument), a personal computer (1.5 GHz Pentium 4 with 512 MB RAM) running under Windows 2000 professional, and a homemade software developed under LabVIEW 6.1. The board is capable of delivering TTL pulses in parallel on four 8-bit ports (32 lines) at a maximum rate of 20 MHz and is equipped with a very large (64 MB) first-input, first-output (FIFO) buffer. The output pulse stream is obtained by adopting the so-called “I/O pattern” procedure,\(^12\) in which the desired pattern of data (sequence of zeros and ones) are continuously output on the selected lines at the chosen clock frequency. The data handshake between the PC memory and the I/O board takes place by means of a continuous (or double) “data-buffer,” whose size is limited by the RAM memory available on the host PC (\( \sim 6.6 \times 10^7 \) bytes in our case). The data are loaded into the data-buffer in blocks by using a so-called “writing-buffer” and are transferred continuously, via direct memory access (DMA), to the FIFO board. The maximum size of the writing-buffer is \( \sim 10^7 \) bytes. The operations of loading and unloading the data-buffer take places asynchronously: while the unloading occurs regularly at the frequency of the output clock, the loading may occur randomly due to the intrinsic stochastic nature of the signal and to unavoidable latencies occurring between data transfers. Thus the remarkably large FIFO...
buffer available on the board plays a crucial role because it screens out efficiently signal fluctuations and guarantees continuity of the output pulse stream.

The procedure followed for delivering the desired I/O pattern of TTL pulses on a single line of the board is sketched in Fig. 2. At the beginning the board is initialized by selecting the period $\Delta t$ of the output clock and by pre-loading the data-buffer. Then, upon a start signal, the procedure begins: the data are read in blocks from the file as integrated arrival times of the photons, then are processed and transformed in a sequence of zeros and ones, loaded onto the data-buffer, and transferred to the I/O board through its FIFO buffer. In this way the TTL pulses have a width equal to $\Delta t$ and are output on bit 0 of the selected port. Finally, to avoid the situation in which two (or more) pulses arriving on adjacent clock intervals would produce a single longer pulse, the output of the port is sent to an AND gate whose second input is the clock signal. In this way the pulse width becomes $\Delta t/2$ and every pulse is separated by the next one by at least half of the clock period.

The system can work in real time (meaning that the output of the pulse stream can proceed indefinitely) only if the average rate at which the data are written into the data-buffer is equal or higher than the clock frequency. Equivalently, we can state that the real time condition is met when the processing time per pulse, $\delta t_{\text{proc}}$, is equal or smaller than the average arrival time $\tau$ between photons, the latter one being related to pulse count rate $\langle I \rangle$ by $\tau = \langle I \rangle^{-1}$.

The processing time $\delta t_{\text{proc}}$ is the sum of several contributions, the most important ones being those sketched in Fig. 2: the reading of data file (task 1), the handling of arrival times into a sequence of zeros/ones (task 2), and the loading of the writing-buffer (task 3). All these tasks are optimized when carried out over large blocks (or arrays) of data, a requirement which can be easily fulfilled because the LabVIEW compiler can allocate a good fraction of the RAM memory available on the host PC. There are two arrays which govern the data flow between the different tasks: the array of arrival times (equal to the number of pulses) whose size is $M$, and the zeros/ones array (equal to the size to the writing-buffer) whose size is $M$. The two sizes are related to the pulse count rate $\langle I \rangle$ and to the clock period $\Delta t$ by $M = \frac{\langle I \rangle}{\Delta t}$. Clearly, when $M$ is large, $m$ is large as well. For example, if $M = 10^7$, $\Delta t = 50$ ns, $\langle I \rangle = 10^5$ Hz, it turns out that $m = 5 \times 10^4$. Now, for any given $m$ and $M$, let us indicate with $T_1$, $T_2$, and $T_3$ the overall times required for carrying out tasks 1, 2, and 3, and with $\delta t_1 = T_1/M$, $\delta t_2 = T_2/M$, $\delta t_3 = T_3/M$, the corresponding processing times per pulse. A lower bound for the overall processing time/pulse $\delta t_{\text{proc}}$ can be therefore estimated as $\delta t_{\text{proc}} \approx \delta t_1 + \delta t_2 + \delta t_3$.

The task related to the file reading depends only on $m$, with $T_1$ scaling linearly with $m$. Thus $\delta t_1$ is a constant which, after having customized the reading routines available under LabVIEW, was optimized to $\delta t_1 \approx 2.3 \mu s$/pulse. It worth mentioning that such a value was obtained by using a commercial PC with a standard hard disk (ATA-66, 7200 rpm). Faster hard disks would clearly perform better.

The task related to the handling of the arrival times into zeros and ones is less critical, but depends on both $m$ and $M$. Shortly, for any given set of arrival times, this task consists of “building” an array of zeros whose size $M$ is given by the overall time elapsed between the first and last photon, divided by the period $\Delta t$ of the output clock. Then, in correspondence of the $m$ photon arrivals, the zeros are replaced with ones. This replacement can be easily carried out because, being the photon arrivals reported as integrated arrival times, they are integer numbers which coincide with the array components to be replaced with ones. We found that the time $T_2$ required for this task scales as $T_2 (\mu s) \approx 6.4 \times 10^{-3} M^{0.8} m^{0.2}$, implying that $\delta t_2 (\mu s) \approx 6.4 \times 10^{-3} (M/m)^{0.8} \times 6.4 \times 10^{-3} (\langle I \rangle \Delta t)^{-0.8}$. Thus $\delta t_2$ scales almost as the reciprocal of the count rate and to the clock period and, for $\langle I \rangle = 10^5$ Hz and $\Delta t = 50$ ns, this time is $\delta t_2 \approx 0.4 \mu s$/pulse.

Finally, the task related to the loading of the writing buffer depends only on $M$, and the corresponding time $T_3$ scales as $T_3 (\mu s) \approx 3.0 \times 10^{-2} M$. This implies that $\delta t_3 (\mu s) \approx 3.0 \times 10^{-3} M/m \approx 3.0 \times 10^{-3} (\langle I \rangle \Delta t)^{-1}$ and, therefore, it scales as the reciprocal of the count rate and the clock period. For $\langle I \rangle = 10^5$ Hz and $\Delta t = 50$ ns, we found $\delta t_3 \approx 0.6 \mu s$/pulse.

In conclusion, the overall time $\delta t_{\text{proc}}$ is determined for a good fraction by a fixed offset represented by the reading task. The other two tasks require substantial less time and their contributions decrease almost linearly with the count rate and the clock period. As an example, for $\langle I \rangle = 10^5$ Hz and $\Delta t = 50$ ns, it turns out $\delta t_{\text{proc}} \approx 3.3 \mu s$/pulse and consequently, since at this count rate the average arrival time is $\tau \approx 10 \mu s$, the real time condition is easily met. By imposing $\delta t_{\text{proc}} = \tau$, we estimated that the maximum count rates at which the real time condition can be achieved are $3.8 \times 10^5$ Hz at $\Delta t = 50$ ns and $4.1 \times 10^5$ Hz at $\Delta t = 100$ ns. It should be pointed out that such limits represent only esti-
mates of the upper bounds for the maximum attainable count rates. In reality there are other reasons which contribute to reduce these limits, such as the time necessary for running the overall LabView program, the unavoidable latencies occurring between data transfers, the time reserved for the Windows operating system, but most of all, the intrinsic fluctuations of the signal. In practice, we found that we could drive reliably the simulator in real time when the effective pulse count rate was about 80%–90% of the maximum values estimated above. Experimentally, we achieved the maximum count rates of \(\sim 3.5 \times 10^5\) Hz at \(\Delta t = 50\) ns and \(\sim 3.7 \times 10^5\) Hz at \(\Delta t = 100\) ns.

When more than one output line is used, the situation is a little bit more delicate. Let us suppose that we want to drive two lines (corresponding to bit 0 and 1 of a given port) and deliver two streams of TTL pulses generated at the same count rate. Suppose also that the data relative to the arrival times of the two lines are read from two different files. For both lines the procedure is identical to the one outlined above, with the only difference that, for the second line, we have to replace the “ones” with “twos.” Then we have to sum up the components of the arrays corresponding to the two lines, so to have a single 8-bit array to be passed to the writing-buffer. The components of this array will be either 0 (no pulse), 1 (pulse on first line only), 2 (pulse on second line only), or 3 (pulses on both lines). The sum of the two arrays requires that they have the same length, i.e., the same number of components. But this is not the case because the arrays are built from a sequence of arrival times (different for each line) and therefore they have a different length. We coped with this problem by buffering the second line so to trim the second array to the same length of the first array. The time \(T_d\) required for this operation (together with the time necessary for carrying out the sum), depends only on \(M\) and scales as \(T_d (\mu s) \sim 1.9 \times 10^{-2} M\). Thus the corresponding time/pulse is \(\delta t_d (\mu s) \sim 1.9 \times 10^{-2} M / m \sim 1.9 \times 10^{-2} (\langle I \rangle \Delta t)^{-1}\). Numerically, for \(\langle I \rangle = 10^5\) Hz and \(\Delta t = 50\) ns, we have \(\delta t_d \sim 3.8\) \(\mu s/pulse\).

Summarizing, the overall processing time required for reading two data files, handling and transforming the arrival times into two sequences of zeros/ones and zeros/twos, summing up the corresponding arrays, and loading the writing-buffer, amounts to \(\delta t_{proc} \sim 2 \delta t_1 + 2 \delta t_2 + \delta t_3 + \delta t_4\), which for \(\langle I \rangle = 10^5\) Hz and \(\Delta t = 50\) ns, corresponds to \(\delta t_{proc} \sim 9.8\) \(\mu s/pulse\). By rising the clock period to \(\Delta t = 100\) ns this time reduces to \(\delta t_{proc} \sim 7.3\) \(\mu s/pulse\). Thus we expect that when \(\langle I \rangle = 10^5\) Hz we can easily drive two lines at \(\Delta t = 100\) ns, but not reliably at \(\Delta t = 50\) ns. Experimentally, we found that the maximum count rates at which we could drive reliably two lines were \(\sim 8.0 \times 10^4\) Hz (\(\Delta t = 50\) ns) and \(\sim 1.2 \times 10^5\) Hz (\(\Delta t = 100\) ns).

### IV. EXPERIMENTAL RESULTS

The simulator was tested by processing its pulse stream with a commercial hardware correlator, namely the model Flex2k-12×2 from Correlator.com. Such a correlator adopts a multi-tau scheme with a minimum gate time of 12.5 ns and is capable of performing both auto- and cross-correlation functions.

The first test was carried out by generating a pulse stream characterized by a single exponential decay correlation function [see Eq. (11)] with a correlation time \(t_1 = 100\) \(\mu s\). The pulses were delivered on a single line with a \(\Delta t = 50\) ns clock, at a count rate \(\langle I \rangle = 3 \times 10^5\) Hz, for a time \(T = 200\) s. Panel (a): count rate trace; panel (b): measured autocorrelation function (open circles) and fitted function (solid line); panel (c): original data out from the Flex correlator (closed symbols) and smoothed data (open symbols) recovered with the procedure described in the text; and panel (d): relative deviations between data and fitting of panel (b).

![Graph](image-url)
integration time smaller than the clock period \( \Delta t = 50 \) ns, and exhibit strong correlation peaks in correspondence of lag time multiples of 50 ns. For the first set [squares, panel (c)] the peaks height is 8 because there are four channels every clock time, while for the second set (triangles) it is 4 because the number of channels per clock time is only two. Starting from the third stage the integration time is equal or larger than the clock period, and no anticorrelation is observed. In conclusion, the smoothing procedure consists of grouping the different channels of the original correlation function so as to obtain a reduced number of new channels whose integration time is equal to the period of the output clock. Thus the 16 channels of the first set become four, and the eight channels of the second set become four as well. These 4 + 4 new channels [circles, panel (c)] are the ones reported in the short lag time data \((\tau < 400 \) ns\) of panel (b).

The overall data of panel (b) of Fig. 3 were fitted to the single exponential function

\[
g(\tau) = B + \beta \exp(-\tau/\tau_1),
\]

in which the baseline B, the amplitude \( \beta \), and the decay time \( \tau_1 \) were the fitting parameters. The fitted function is reported in panel (b) as a solid line. The matching between the data and the fit is remarkably good, as evidenced by the residual plot reported in panel (c), in which the relative deviations are shown to be nonsystematic, less than \( 10^{-3} \) rms. The recovered decay time was \( \tau_1 = 100.1 \pm 0.06 \) \( \mu s \) while the amplitude was \( \beta = 0.9720 \pm 2 \times 10^{-4} \). As for the count rate, the slight difference between \( \beta \) and the theoretical value of 1 is attributable to pile-up effects.\(^{17} \)

The second test involved the generation of two different pulse streams and the measurement of their cross correlation function. The aim of the test was to simulate a typical situation encountered in PCS, in which, to eliminate the distortions introduced into the autocorrelation function by the presence of the photodetector afterpulses, the optical signal is split into two parts and detected with two different detectors. Thus, since the afterpulses of the two detectors are not correlated, the cross correlation of their outputs reproduces accurately the autocorrelation function of the optical signal. With this scheme in mind, we generated an optical signal characterized by a single exponential decay correlation function with \( \tau_1 = 1 \) ms and by a count rate (\( \bar{\lambda} \)) = \( 10^3 \) Hz. Then, such a signal was passed through two independent Poisson filters and each sequence of photocounts was biased by stochastically introducing an extra count due to the event of an afterpulse. Practically, we assigned a conditional probability \( p_c \) of having an afterpulse whenever a genuine photon pulse occurred. Then the afterpulse delay was chosen according to a Gaussian distribution centered around the average delay \( \tau_{ap} \) and spread over the standard deviation \( \sigma_{ap} \). The two afterpulses generated in this test had the following characteristics. Signal A: \((p_c)_A = 2\%\), \((\tau_{ap})_A = 0.5 \mu s\), \((\sigma_{ap}/\tau_{ap})_A = 15\%\); signal B: \((p_c)_B = 1\%\), \((\tau_{ap})_B = 1 \mu s\), \((\sigma_{ap}/\tau_{ap})_B = 15\%\). The two pulse streams were output with a 100 ns clock and analyzed with the Flex correlator for an overall measuring time \( T = 200 \) s. The results are reported in Fig. 4, in which the autocorrelation functions A-A (squares) and B-B (triangles) clearly exhibit their afterpulse peaks, while the cross correlation function A-B (circles) is immune to them. For the sake of clarity, the A-A and B-B correlation functions have been shifted vertically by a factor 1 and 0.5, respectively. Panel (b): relative deviations between the data A-B of panel (b) and the corresponding fitting.

V. DISCUSSION

By using a commercial I/O board (National Instrument, model PCI-6534) and a Personal Computer (1.5 GHz Pentium 4) we have developed a hardware simulator ideal for testing and bench marking digital correlators commonly used for photon correlation spectroscopy. The simulator is capable of delivering a continuous stream of TTL pulses with the desired pattern over one or more channels, the different channels being synchronized with the same output clock. The pulse resolution (minimum distance between adjacent pulses) was selectable via software to values multiple of the
clock period available on the I/O board, namely $\Delta t = 50,100, \ldots$ ns. When a single channel is used, the maximum count rate is only slightly dependent on the pulse resolution and, at $\Delta t = 50$ ns, was $\langle I \rangle \sim 350$ kHz. With two channels we obtained $\langle I \rangle \sim 80$ kHz at $\Delta t = 50$ ns and $\langle I \rangle \sim 120$ kHz at $\Delta t = 100$ ns.

A simple algorithm (based on the idea reported in Refs. 13 and 14) was implemented in order to generate synthetic PCS data with the desired Gaussian statistics, at the desired count rate, and in the presence of shot noise. In particular, data characterized by a single exponential decay correlation function were simulated, but the procedure can be easily generalized to multidecay times or to other kinds of statistics. For example, it would be straightforward to simulate the signal deriving from a polydisperse sample and compare the size distribution recovered by the inversion routines available on commercial correlators with the expected one (see, for example, Ref. 8).

The simulator is fairly flexible and permits one to simulate common problems encountered in PCS. We reported the example of photodetector afterpulses and showed how to eliminate them by cross correlation. Simulations of other problems, such as photodetector dead time, presence of dust particles, straight light, and so on, can be easily taken into account.

The performances of the simulator, such as the time resolution and the maximum count rate of the output pulses, are determined and limited by the performances of the hardware components used by the system. While the time resolution is determined by the PCI-6534 I/O board, the maximum count rate is limited by the PC, in particular by the speed at which the data are read from the hard disk. In this work we have used a commercial PC equipped with a fairly standard hard disk (ATA-66, 7200 rpm), which allows downloading the data at a rate of $\sim 2.3 \mu$s/point. Currently, this is the narrowest bottleneck in our system, and for example, by reducing this rate by a factor of $\sim 5$, it would correspond to an increase of the maximum count rate by (almost) the same factor.

The hardware simulator can also be utilized as a general purpose pulse pattern generator (PPG). Commercial PPGs are sophisticated (and very expensive) instruments capable of delivering any desired pattern of output pulses at very high frequency (up to GHz). However, the length of the pulse pattern is limited by the memory available on the instrument (typically $\sim 10^6$ bit or less) and, consequently, the duration of output signal is very short (depending on the clock frequency), unless the pattern is repeated. Our simulator is definitely slower than a PPG, but has the remarkable advantage of allowing a continuous stream of the output pulses, which is of fundamental importance in PCS and many other applications. Moreover, it offers several other advantages related to the fact that it has been developed by using a general purpose I/O board and a PC, namely it is very flexible, at relatively low cost, and easily adaptable to future technology developments.

ACKNOWLEDGMENTS

The authors thank D. S. Cannell and A. Smart for having drawn to their attention the possibility of simulating PCS data by the simple algorithm used in this work [Eq. (7)] and originally reported in Refs. 13 and 14. The authors also thank G. David for the technical support received during the early phases of this project. This work was partially supported by funds from the Italian Space Agency (ASI).

17 See, for example, B. Saleh, Photon-electronics Statistics (Springer, Berlin, 1987).