timetagger: user's manual

A ten-channel picosecond time tagger with 1 ns markers and a coincidence engine, on the Red Pitaya STEMlab 125-14

with an applications part: time tagging in quantum-optics, quantum-communication and clock-synchronization labs

Rok Žitko
Jozef Stefan Institute, Ljubljana, Slovenia
rok.zitko@ijs.si

Red Pitaya `timetagger` design, version 2.0 (build 0x9b2ec69e)
October 31, 2026

Abstract

`timetagger` turns the Red Pitaya into a **time tagger**: every edge on any of its ten inputs gets a timestamp in ticks of 0.98 ps, and the timestamps stream to a PC, where programs turn them into correlation histograms ($g^{(2)}$), lifetime histograms (TCSPC), coincidence counts, rates and clock comparisons. Each channel times its edges with a tapped delay line in the FPGA's carry chain, calibrated by the instrument itself; eight more inputs time slower signals to 1 ns; a coincidence engine in the FPGA counts coincidence patterns at rates far beyond what a network can carry; a conditional filter thins an 80 MHz laser sync to the periods a photon follows; a 10 MHz reference and a PPS relate the timestamps to a lab's time scale. Part I of this manual is about the instrument: the first measurement, connecting signals, every setting and how it behaves, calibration and precision, acquisitions and the stream, the markers, the engine and the filter, references, the self-tests, the software and the specifications. Part II is about using a time tagger in a quantum lab: the measurements of quantum optics, entanglement-based quantum key distribution (QKD) with the synchronization of two stations' clocks from the photons themselves, prepare-and-measure QKD, quantum time transfer, and the comparison and synchronization of clocks, with examples and the code that analyses them. Figures marked Measured were recorded with the board used here and the lab equipment named in their captions; figures marked Computed come from simulations (photon sources, detectors and clocks with this tagger's measured timing), whose timestamps are analysed with the design's own functions exactly as recorded ones would be. The FPGA internals are in `docs/THEORY.md` and `docs/REGISTERS.md`, every setting and the record format in `docs/PROTOCOL.md`, the test records in `docs/VERIFICATION.md`.

Contents

I The instrument **4**

1 Introduction **4**
1.1 What timetagger is . . . 4
1.2 How it works . . . 4
1.3 What sets it apart . . . 5
1.4 The documents . . . 6
1.5 Before you connect equipment . . . 6

2 Getting started **6**
2.1 The board and the service . . . 6
2.2 The first calibration . . . 6

2.3 A first measurement, without wiring . . . 7
2.4 Presets . . . 7
2.5 The panel . . . 7
2.6 TCP and the command line . . . 7

3 Connecting signals **8**
3.1 The extension connector E1 . . . 8
3.2 Levels and termination . . . 8
3.3 The best connection for clean edges . . . 8
3.4 The SATA pairs . . . 9
3.5 IN2: the reference input . . . 9

4 Channels **9**
4.1 Edges, levels and toggles . . . 10
4.2 Dead time, divider and offset . . . 10
4.3 Counters . . . 11

5 Calibration and precision **11**
5.1 The delay lines and their tables . . . 11
5.2 Calibrating . . . 12
5.3 Dither . . . 12
5.4 Live calibration . . . 12
5.5 Temperature . . . 12
5.6 Times at the connectors . . . 13
5.7 Precision . . . 15
5.8 The error budget . . . 17

6 Acquisitions and the time base **18**

7 The stream: records, the ring and the PC **18**
7.1 Rates . . . 18
7.2 On the PC . . . 19

8 Markers **19**

9 The coincidence engine **20**

10 The conditional filter **21**

11 References and time scales **22**
11.1 The IN2 phase meter . . . 22
11.2 Tagged references and PPS . . . 22

12 Test signals and self-tests **23**
12.1 The board test . . . 23

13 Software **24**
13.1 On the board and on a PC . . . 24
13.2 Records and measurements in Python . . . 24
13.3 ttcore . . . 24
13.4 TCP . . . 25
13.5 Stability analyses . . . 25

14 Specifications **25**
14.1 Troubleshooting . . . 26

II Time tagging in quantum labs **26**

15 From time tags to measurements **26**
15.1 Three histograms . . . 26
15.2 Accidental coincidences . . . 27
15.3 The timing budget . . . 27
15.4 Dead time and afterpulsing . . . 28
15.5 Connecting detectors . . . 28

16 Quantum optics with a time tagger **29**
16.1 Photon statistics: Hanbury Brown and Twiss . . . 29
16.2 Fluorescence lifetimes: TCSPC with the conditional filter . . . 30
16.3 Two-photon interference: the Hong–Ou–Mandel dip . . . 31
16.4 Heralded single photons and photon pairs . . . 32

17 Entanglement-based quantum key distribution **33**
17.1 The protocol and what the tags carry . . . 33
17.2 Synchronizing two clocks with the photons . . . 34
17.3 The coincidence window, the QBER and the key rate . . . 36
17.4 Over distance: what the timing buys . . . 37
17.5 Visibility and the Bell parameter . . . 37
17.6 Monitoring a station: the engine, the markers and the filter . . . 38
17.7 Timing side channels . . . 38

18 Prepare-and-measure QKD and quantum time transfer **39**
18.1 Time slots and gates . . . 40
18.2 Clock recovery without a sync . . . 40
18.3 Two-photon protocols: MDI and twin-field QKD . . . 41
18.4 Quantum two-way time transfer . . . 41

19 Clock comparison and synchronization **42**
19.1 Time error, frequency and their statistics . . . 42
19.2 What the tagger measures . . . 42
19.3 Measured: a crystal, a generator and a GNSS-disciplined oscillator . . . 43
19.4 Synchronizing distant clocks . . . 45

A Command reference **47**
A.1 TCP commands . . . 47
A.2 Settings . . . 47
A.3 Records . . . 48

B Glossary **48**

Part I
The instrument

1 Introduction

1.1 What timetagger is

`timetagger` is an FPGA design for the Red Pitaya STEMlab 125-14 that makes it a **time tagger**: an instrument that records the time of every edge on its inputs and hands the list of times to a computer. It has

- **ten channels** timed to picoseconds: eight 3.3 V inputs on the extension connector E1 (DIO0_P–DIO7_P) and two 1.8 V differential pairs on the SATA connector S2. Each channel measures the time of its edges with a **tapped delay line** in the FPGA's carry chain, in ticks of 4 ns/4096 = 0.977 ps, about 10 ps rms per channel (section 5.7);
- **eight markers** (DIO0_N–DIO7_N), slower inputs timed to 1 ns, for gates, frame and line clocks, triggers;
- per channel: the edges to tag (rising, falling, both), a **toggle input** for pulses of any width, a **dead time**, a **divider**, an **offset**, and counters;
- **self-calibration** without any wiring: a ring oscillator in the FPGA calibrates every delay line, the channels' own hits keep the calibration current while the die's temperature changes, and an internal test signal measures the delays from each connector, so that times refer to the connectors;
- a **coincidence engine** in the FPGA that counts coincidence patterns among up to eight channels at up to 125 Mevents/s, far beyond what the network carries, with masks, virtual channels and a gate;
- a **conditional filter** that keeps a laser's 80 MHz sync only where a photon follows;
- **acquisitions** started now, at a given moment or on a channel's edge (a PPS), stopped by a duration;
- a **stream** of the tags to any number of programs on PCs, through a ring in the board's memory: 7 Mevents/s without loss, every loss counted;
- **references**: a 10 MHz sine on IN2, measured by a phase meter, or a tagged clock or PPS, to express the tags in a lab's time scale;
- software: Python on the board and on PCs, a C++ library (ttcore) that measures correlations, start–stop histograms, coincidences and rates as the tags arrive, and the common stability tools for clock analyses.

1.2 How it works

Figure 1 shows the path of an edge. The FPGA's clock ticks every 4 ns; finer times come from the delay line. An edge entering a channel runs up a chain of 80 carry cells, 160 taps of about 30 ps; at every tick of the 250 MHz sample clock, flip-flops beside the taps record how far it has got (figure 2). The number of taps showing the new level, the *code*, measures the time from the edge's arrival to the clock's tick. The taps are not equal, and their delays drift with temperature: a **calibration table** per channel and polarity turns each code into a time, made by *code density*: edges at random phases fall into each bin in proportion to its width, so a histogram of codes is a table of widths. A tag is the sample's number times 4 ns, less the fine time, plus the channel's offsets: 60 bits of ticks, 13 days.

The channels' events wait in FIFOs, from which the **merger** takes them in the order of their 4 ns samples (4.1 µs after the sample), one per clock; the **lookup** applies the calibration, a random

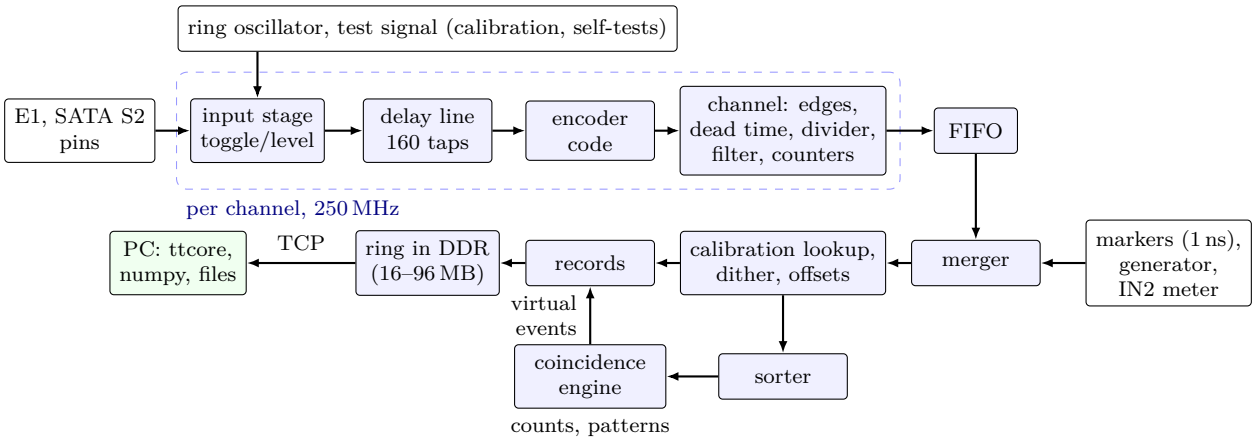


Figure 1: The path of an edge: per channel, the input stage, the delay line and its encoder, the channel's logic and its FIFO; shared, the merger, the calibration lookup and the records; the ring in the board's memory and the stream to PCs. The coincidence engine takes the calibrated events through a sorter of its own.

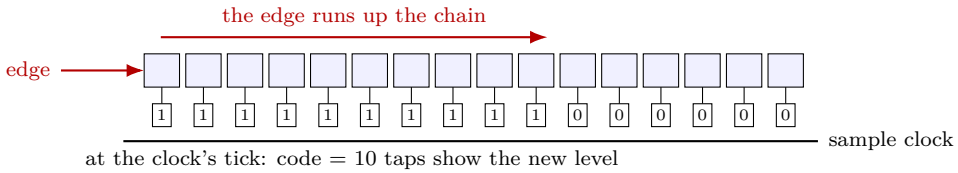


Figure 2: A tapped delay line: the edge has passed ten taps when the sample clock ticks; the code (10) measures the time between the edge and the tick. The real chain has 160 taps of about 30 ps and needs more than one 4 ns period of them.

dither within the bin, and the channel's offsets; the **records** (32-bit words, one per event) go to a **ring** in the board's DDR memory, and a program on the board's processor, the *ring pump*, sends them over TCP to every client. The **coincidence engine** sorts its channels' events into exact time order and counts coincidence patterns as they pass. The details, with the numbers, are in `docs/THEORY.md`.

1.3 What sets it apart

- **It calibrates itself.** The ring oscillator, the live histograms and the test signal need no cables; the tables are kept per board, and `make test-hw` checks the whole instrument in about ten minutes.
- **Times refer to the connectors.** The delays from each pin to its delay line come from the FPGA tool's timing model, scaled by a measurement on the test signal, so that channels agree without a calibration cable.
- **Counting in the FPGA where the network cannot follow.** The coincidence engine and the conditional filter turn rates of tens of millions of events per second into counts or into the few events that matter.
- **Open.** The design, the model it is verified against, the software and the test records are all in the repository; the same board runs the other designs of the framework.

What it is not: an instrument with analog front ends (thresholds, NIM levels; these were planned for a version 3.0 on a larger FPGA), or one whose own clock locks to an external reference (the board's crystal clocks the hardware; references re-express the tags afterwards, section 11).

1.4 The documents

document	contents
this manual	using the instrument (Part I) and time tagging in quantum labs (Part II)
`README.md`	overview, quick start, specifications, hardware setup
`docs/PROTOCOL.md`	every setting, the Python methods, TCP commands, the records stream and format
`docs/THEORY.md`	how it works: delay lines, encoding, merging, coincidences, calibration, the ring, references, the error budget
`docs/REGISTERS.md`	the hardware registers
`docs/VERIFICATION.md`	what was tested and when, with the results
`tools/ttcore/README.md`	the C++ host library

1.5 Before you connect equipment

Before you connect equipment

- **E1 is 3.3 V logic and not 5 V tolerant** (absolute maximum 3.85 V). A 5 V TTL output must be terminated in 50 Ω at the board, which halves it. Nothing negative: NIM pulses need a NIM-to-TTL converter.
- **The SATA connectors are not SATA**: they are 1.8 V FPGA pins without termination on the board. Connect only Red Pitaya daisy-chain cables or 1.8 V differential signals.
- **SATA S1 drives** when the test signal's output is on (`test.sata_out`); never with anything else driving S1.
- **IN2** takes ±1 V with the LV jumper and ±20 V with HV; a +13 dBm reference sine (2.8 V peak to peak into 50 Ω) needs the HV jumper or a 6 dB attenuator and a 50 Ω terminator.
- The board-wide rules are in the repository's root `README.md`.

2 Getting started

2.1 The board and the service

The board runs a control service (`rp-overlay.service`) that loads a design and answers on TCP port 5025; timetagger's records leave on port 5026. On the ready SD card image all designs are installed; from a checkout of the repository, `make install-service` in `timetagger/` (with `BOARD_HOST` set to the board's address) builds what is needed, uploads the design and makes it the active one. The examples here use the address 192.168.0.102. The SD card image is built from the PYNQ image of the `Pynq-Redpitaya-125` project (https://github.com/dspsandbox/Pynq-Redpitaya-125), and the designs' common block design grew out of the tutorials of `FPGA-Notes-for-Scientists` (https://github.com/dspsandbox/FPGA-Notes-for-Scientists), both by Pau Gómez: with thanks.

Programs talk to the service, never to the FPGA directly: on the board through the driver, `pynq.overlays.timetagger.DRIVER()`, in Jupyter or a script; on a PC through the client, `TimeTaggerClient`, which needs only Python's standard library (and numpy to decode the records). Both have the same methods. Below, `tt` is either.

```
import sys; sys.path[:0] = ["common/python", "timetagger/control"]     # a checkout, on a PC
from timetagger import TimeTaggerClient, RecordStream, analysis
tt = TimeTaggerClient("192.168.0.102")
```

2.2 The first calibration

Each delay line needs its table. Once per board (and after an update of the bitstream), with nothing connected:

```
report = tt.calibrate()          # about 12 s: code density, then the delays and polarity offsets
```

The tables are kept on the board, per board and bitstream, and loaded whenever the design is; while measuring, the channels' own hits refine them (section 5).

2.3 A first measurement, without wiring

The **generator** makes random events on two internal sources, channels 31 and 32 in programs, half of them in pairs exactly 10 ns apart. A second's acquisition, streamed to the PC and decoded, shows the pairs in the correlation of the two sources:

```
tt.load_preset("generator")      # 1 Mevent/s, half of them paired 10 ns later
with RecordStream("192.168.0.102") as stream:
    tt.start(duration=1.0)
    decode, chs, ts = None, [], []
    for frame in stream.frames(timeout=2.0):
        head, words = analysis.parse_frame(frame)
        decode = decode or analysis.Decoder(head["time_top"], head["time_hi"])
        c, t = decode(words)
        chs.append(c); ts.append(t)
ch, t = np.concatenate(chs), np.concatenate(ts)
h = analysis.correlation(t[ch == 31], t[ch == 32], 100, 400)   # 100 ps bins, +-20 ns
print(np.argmax(h) * 100 - 20000)                              # 10000 ps
```

The generator's times are exact (they come from the FPGA's clock, not from a delay line), so the pairs fall into a single bin. The **test signal** goes through the delay lines: with the preset `test` every channel tags its edges, and the differences between two channels show the instrument's precision (section 5.7).

2.4 Presets

preset	what it does
`off`	all channels off; generator, test signal and ring oscillator off (power-up, SAFE)
`detectors`	channels 1–8 on the rising edges of DIO0_P–DIO7_P
`all`	all ten channels on rising edges
`test`	all channels on the internal test signal: checks without wiring
`generator`	the test generator: 1 Mevent/s, half of them paired 10 ns later

`tt.save_preset(name)` keeps the current settings under a name of one's own; `tt.presets()` lists them.

2.5 The panel

The notebook `control/timetagger.ipynb`, installed on the board in Jupyter's folder `timetagger`, builds a panel with a row per channel (enable, edge, toggle, offset, dead time, divider, record, filter, rate), the filter, the markers with their live levels, the coincidence engine (lanes, window, gate, the eight masks and the top patterns) and buttons for rates, acquisitions, calibration and the presets.

2.6 TCP and the command line

Every method is also a text command on port 5025 (appendix A):

```
$ nc 192.168.0.102 5025
PRESET generator
OK
ACQ START FOR 1
OK {"acquisition": 3, "k0": 2911873024}
COUNTERS?
OK {"1": {"events": 0, ...}, "generator": {...}, ...}
```

and the common command line takes the same commands:

```
python3 -m rpfpga.cli --host 192.168.0.102 --help
```

3 Connecting signals

3.1 The extension connector E1

The channels 1–8 and the markers are on E1, a 2 × 13 header of 2.54 mm pitch (figure 3). Its only ground pins are **25 and 26**, at the end of the header next to DIO7; pins 19–24 are not connected and pins 1–2 carry 3.3 V.

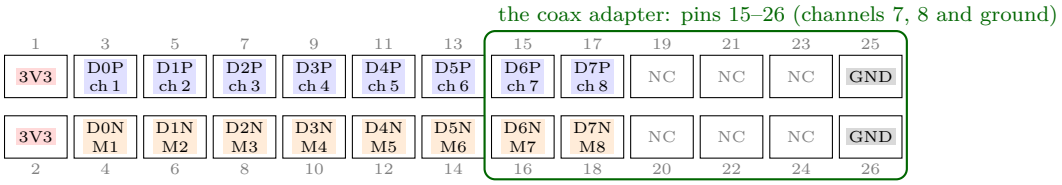


Figure 3: The extension connector E1 seen from above, pin 1 at the left: channels 1–8 on the odd pins 3–17 (DIO0_P–DIO7_P), markers M1–M8 on the even pins 4–18 (DIO0_N–DIO7_N), ground on pins 25 and 26 only.

pins	signals	timetagger
3, 5, . . . , 17	DIO0_P . . . DIO7_P	channels 1–8 (delay lines)
4, 6, . . . , 18	DIO0_N . . . DIO7_N	markers M1–M8 (1 ns)
25, 26	ground	
1, 2	+3.3 V	do not use as signal sources

3.2 Levels and termination

The E1 pins are 3.3 V LVCMOS inputs with pull-downs (an unconnected channel stays quiet). A level above about 2.0 V is high, below 0.8 V low; the absolute limits are −0.4 and +3.85 V.

- **A source with a 50 Ω output and a short cable**, set for a 3.3 V (or 2.5 V) high level into a high impedance: the cable is *series terminated* by the source; the edge doubles at the open input and the reflection is absorbed at the source. The edge at the pin is clean after one round trip of the cable; nothing at the board.
- **A source that drives 50 Ω** (most detector modules, a PPS output into 50 Ω): a 50 Ω terminator at the board. A 5 V TTL output becomes about 2.5 V, within the levels and below the limit; without it, it is not allowed on E1.
- **NIM** levels are negative: a NIM-to-TTL converter first.
- **Short pulses** (a few ns, e.g. from SNSPD electronics): a toggle input (section 4) puts exactly one transition into the delay line per pulse.

3.3 The best connection for clean edges

A wire on a header pin is a poor transmission line: its inductance and the long way to the ground pins ring and slow an edge, and every millivolt of noise on a slow edge becomes time jitter. For precise timing on E1, use the channels nearest the ground pins, 7 and 8 (pins 15 and 17), and a small adapter (figure 4):

- a 2 × 6 female header socket covering pins 15–26, mounted on a piece of copper-clad board that serves as the ground plane, joined to pins 25 and 26 by short, wide copper;

- two edge-mounted SMA jacks whose shells are soldered to the plane, their centre pins to pins 15 and 17 with leads of 10 mm or less;
- a 50 Ω resistor (SMD) from each signal to the plane at the socket, for sources that drive 50 Ω, removable for series-terminated ones.

The same adapter takes a PPS from a GNSS-disciplined oscillator (section 11). To give two channels one source (to split a detector's pulse, or to measure the tagger, as in section 5.7), fan a single jack out to pins 15 and 17 behind one 50 Ω termination: the *fork* adapter. Whatever the wiring, keep the edges at the pins faster than about 0.3 V/ns: slower ones turn the board's own millivolts of noise into tens to hundreds of picoseconds (section 5.7). Lacking an adapter, solder the coax's braid to a two-pin housing on pins 25 and 26 and its centre to a single-pin housing on pin 17, with the shortest leads possible; never a single ground wire run elsewhere on the board. Section 5.7 gives the precision measured this way.

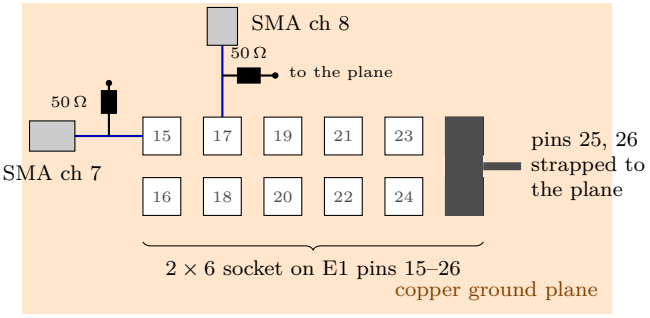


Figure 4: The coax adapter for channels 7 and 8: a socket over pins 15–26 on a ground plane tied to pins 25 and 26, two SMA jacks, short leads to pins 15 and 17, and a 50 Ω termination at the pin (one shown).

3.4 The SATA pairs

Channels 9 and 10 are the two differential pairs of the SATA connector S2 (pairs A and B), in an FPGA bank at 1.8 V; the FPGA terminates each line internally (to half the supply, about 50 Ω each). They take 1.8 V differential signals: another Red Pitaya's daisy-chain output, or the test signal from S1 through a straight SATA cable, which is how they are tested. The connector S1 is an output, driven only when `test.sata_out` is on.

3.5 IN2: the reference input

A 10 MHz sine on IN2 is measured by a phase meter (section 11). With the LV jumper IN2 takes ±1 V; a reference of +13 dBm (2.8 V peak to peak into 50 Ω) needs the HV jumper, or 6 dB of attenuation and a 50 Ω terminator at the input. IN1 is not used.

4 Channels

```
tt.apply({"channels": {"1": {"enable": True, "edge": "rising", "dead_time": 50e-9},
                       "2": {"enable": True, "edge": "falling", "offset": 1.5e-9},
                       "3": {"enable": True, "divider": 100}}})
```

Each of the ten channels has the settings below (`channels.n.*`, $n = 1 \ldots 10$). A channel that is not enabled produces nothing and costs nothing.

key	default	meaning
`enable`	false	the channel produces events
`edge`	`rising`	`rising`, `falling` or `both`; falling edges are the negative channel number in programs ($-n$)
`toggle`	true	a flip-flop toggled by the chosen edge drives the delay line: one transition per edge, whatever the pulse's width. False: the pin itself (needed for `both`)
`input`	`pin`	`pin`; or an internal source: `ring` (the ring oscillator), `test` (the test signal), `test_toggle` (the test signal through the channel's toggle)
`dead_time`	0	seconds, up to 262 µs: events closer than this after an accepted one are ignored
`divider`	1	keep every N-th event, N up to 65536
`offset`	0	seconds, ±2.05 µs: added to the channel's times (cable delays)
`record`	true	the events go into the records; false: to the coincidence engine only
`calibrate`	true	the channel's hits refine its calibration (live calibration)
`filter`	false	channels 1 and 2: the conditional filter (section 10)
`hit_cell`	0	the delay-line cell where a transition counts (expert; THEORY.md)

4.1 Edges, levels and toggles

A delay line times *transitions*. With a **level input** (`toggle` false) the pin drives the line itself: a pulse puts two transitions into it, its leading and its trailing edge, and both are timed, each with its polarity. If the pulse is shorter than the line (about 5 ns) the two transitions are in the line together and the second is flagged and dropped. With a **toggle input** (the default) a flip-flop flips at every chosen edge, and its output drives the line: exactly one transition per edge, however short the pulse (figure 5). The toggle's transitions alternate in polarity, and the two polarities run through the line slightly differently; the instrument measures this *polarity offset* and corrects it (section 5).

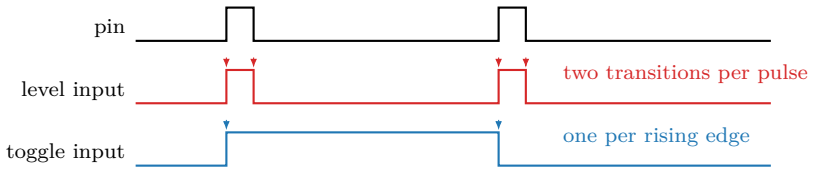


Figure 5: A short pulse at a level input puts two transitions into the delay line; at a toggle input (rising edges) one.

For detectors that deliver a short pulse per photon, use toggle inputs (the default) and the edge the detector specifies as its timing edge. For `both` edges (e.g. the width of a pulse) use a level input and pulses longer than the line.

4.2 Dead time, divider and offset

The **dead time** ignores a channel's events for a time after each accepted one, in whole 4 ns samples (an event exactly at its end is accepted). The delay line has a dead time of its own of about 5 ns. A programmable dead time longer than a detector's own makes its behaviour known and the same for all detectors (section 17.7).

The **divider** keeps every N-th event (after the dead time): a laser's 80 MHz sync divided by 100 becomes 800 kevents/s, and the PC interpolates between the kept edges if it needs all of them. For lifetime measurements the conditional filter is usually the better choice (section 10).

The **offset** is added to the channel's times: a cable's delay, a detector's latency. Up to ±2.05 µs are applied in the FPGA (which matters for the coincidence engine and the filter, which see the

shifted times); larger offsets belong on the PC.

4.3 Counters

`tt.counters()` returns per channel the accepted `events` (edges after the edge selection, before the divider), the `flagged` hits (two transitions closer than the delay line can separate, dropped), and the `drops` (the channel's FIFO was full); and for the whole instrument the events `released` into the records, `discarded` (outside an acquisition), `stale`, the coincidence sorter's `late` events, the ring's bytes and stalls, and the markers' and the generator's counts. `tt.clear_counters()` zeroes them. A channel's rate is its events per second:

```
before = tt.counters()["1"]["events"]
time.sleep(1.0)
rate = tt.counters()["1"]["events"] - before          # events per second on channel 1
```

5 Calibration and precision

5.1 The delay lines and their tables

A delay line's taps are not equal: inside a carry cell the outputs are not in delay order, the cells, the boundary between two clock regions of the chip and the sample clock's routing add steps of their own, and everything slows as the die warms. The calibration measures each bin by **code density**: edges that arrive at random phases of the 4 ns sample period land in each bin in proportion to its width, so a histogram of N codes with h_c hits in code c gives the widths

$$w_c = 4\,\mathrm{ns} \cdot \frac{h_c}{N}, \qquad t_c = \sum_{c'<c} w_{c'} + \tfrac{1}{2} w_c, \tag{1}$$

and the table of each channel and polarity holds every code's left edge and width. The random edges come from a **ring oscillator** in the FPGA, a loop of inverters running freely at 97.7 MHz, unrelated to the sample clock; two million hits per polarity take four seconds and leave about 3 ps of statistical error.

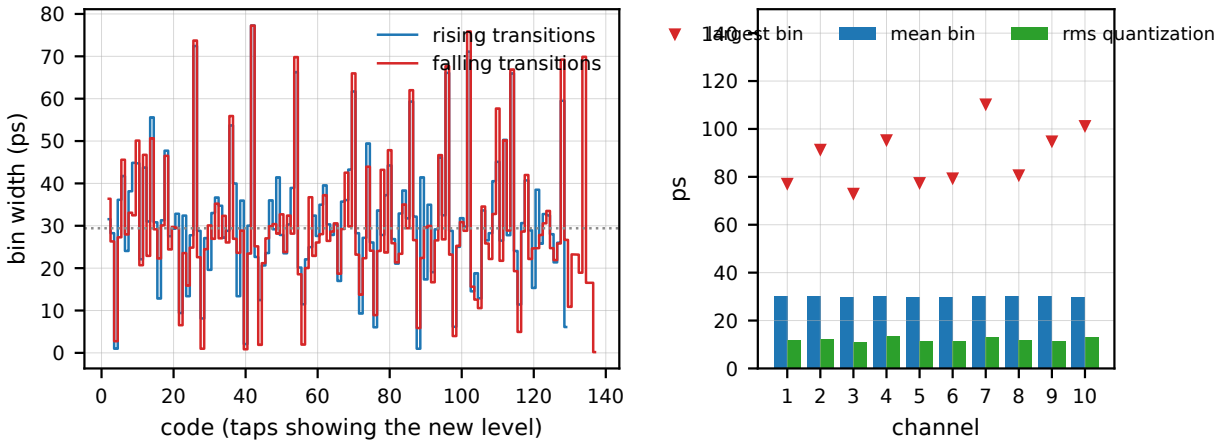


Figure 6: Measured (2026-10-30, die at 49 °C). Left: the bins of channel 1, per code, for its rising and falling transitions (dotted: the mean). Right: every channel's mean bin, largest bin and rms quantization.

On this board the bins average 28.8–31.2 ps, the largest are 68–110 ps, and one 4 ns period takes 128–139 codes (figure 6). The *rms quantization*, the timing error of a random arrival from the bins' widths alone, $\sqrt{\sum w_c^3/(12 \cdot 4\,\mathrm{ns})}$, is 10.9–13.6 ps. Uncorrected, the line would be far from straight: its integral non-linearity spans 72 ps (figure 7); the table removes it.

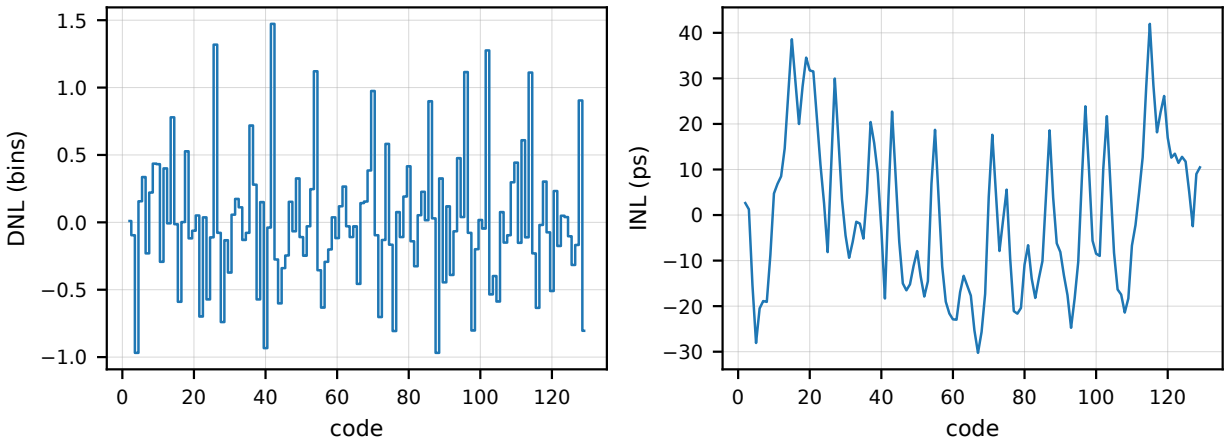


Figure 7: Measured (2026-10-30). Channel 1's rising transitions: the differential non-linearity (each bin against the mean) and the integral non-linearity (the bins' centres against a straight line), which the calibration table corrects.

5.2 Calibrating

```
report = tt.calibrate()          # code density (4 s), then the delays and polarity offsets (7 s)
print(report["1"]["1"])           # channel 1, rising: hits, codes, mean_bin_ps, max_bin_ps
```

`calibrate()` switches the channels to the ring oscillator, histograms their codes, turns the histograms into tables, loads them and keeps them on the board (per board and bitstream: the tables depend on the chip and the placement), and then calibrates the delays to the connectors (below). The service loads the kept tables whenever it loads the design. Calibrate once per board and after an update of the bitstream; the live calibration (below) follows the temperature. The report includes the ring's frequency, and warns if it is within 3 MHz of 125, 83.3 or 62.5 MHz, where the sample clock could pull it and bias the tables.

5.3 Dither

A table turns each code into one time, so without further steps the tags of a channel take only the 128–139 values per period, and a histogram of time differences with bins finer than the delay line shows a comb. With `dither` on (the default) each time is placed at random within its bin; the times are then spread smoothly, at the cost of the bins' own spread (the rms quantization), which they carry in any case (figure 8). Dither off is useful only to look at the bins themselves.

5.4 Live calibration

The delays follow the die's temperature (section 5.5). While measuring, each channel's hits feed a histogram of its codes (`calibrate`, on by default), and every 5 s the service turns the histogram of every channel and polarity with at least a million hits into a new table. Only random arrivals give a code density: a signal locked to the board's clock (the internal test signal, or a clock derived from the board's crystal) fills only a few codes. The service therefore keeps a histogram only if no more than 10 % of the codes that the previous table says are wider than 5 ps are empty, and counts the others as rejected (`tt.worker()`). Photon detections are random enough; a laser's sync at a rate related to the board's clock is not, but it is rejected automatically.

5.5 Temperature

The delay lines, the routes to them and the board's crystal all depend on the die's temperature. To see how much, the board's fan was switched off for 20 minutes and then on again (2026-10-31, `tools/record_manual_data.py --session temp`): the die went from 48 to 71 °C and back. About

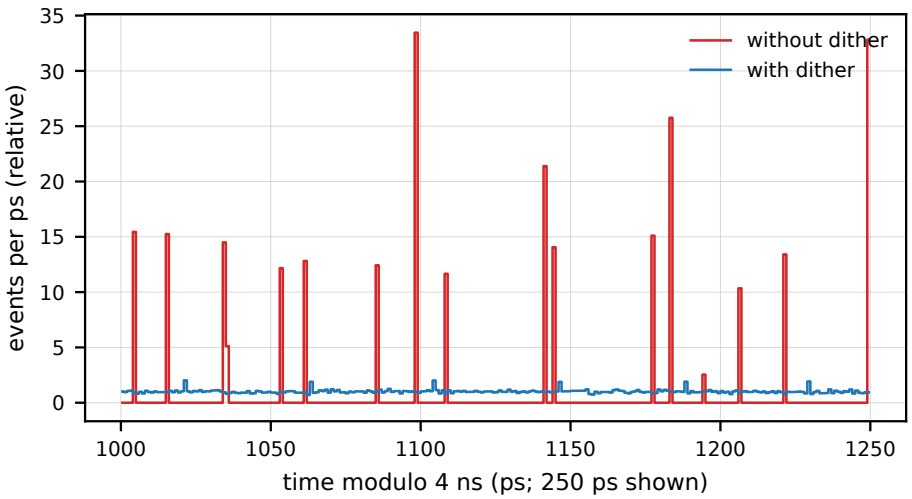


Figure 8: Measured (2026-10-30). The times of 500 k asynchronous edges on channel 1 (a function generator's 1 MHz square), modulo the 4 ns period, 250 ps of it shown: without dither every time is one of the bins' centres; with dither they spread uniformly.

twice a minute the program calibrated (2 s), measured channels 2, 5 and 9 against channel 1 on the test signal, and looked at the crystal for 10 s against a GPSDO on IN2 and a function generator on channel 1: 92 steps (figure 9).

- **The delay lines.** The mean bin changes by −0.021 % per kelvin (0.5 % over the run): the carry chains get slightly faster as the die warms, while the ring oscillator's lookup tables and routes get slower (its frequency −0.018 % per kelvin). A table made cold is then off by up to 0.5 % of the fine time, about 19 ps at the end of the 4 ns period. Live calibration follows the change (on by default, with random hits); otherwise call `calibrate()` after the temperature has changed.
- **The routes to the lines.** The fitted scale λ of the tool's delays (section 5.6) grows by 0.64×10^{-3} per kelvin: the routes slow down too. Recalibrated at each step, the channels' offsets against channel 1 changed by 0.02–0.15 ps per kelvin (the most on channel 9, a SATA pair): a few picoseconds over the whole run, about the scatter of one calibration.
- **The crystal.** Its frequency changed by −128 ppb per kelvin of the die (2783 ppb over the run), against the GPSDO and against the generator alike (−128 ppb per kelvin), while the generator against the GPSDO stayed put (0.3 ppb per kelvin). Section 19.3 shows what this does over hours in a room, and how it fooled a three-cornered hat.

Keep the fan on

Without a fan this board's die passed 70 °C within 20 minutes; with one it stays near 48 °C and follows only the room. After switching on, or a change of airflow, wait for the die to settle (about five minutes here; `tt.hardware()["temperature_c"]`) and calibrate. For frequency and long intervals, use a reference on IN2 (section 11): the crystal follows the room.

5.6 Times at the connectors

A channel's raw time is the moment its transition entered its delay line, against the sample clock's edge there. The paths from the ten connectors to their delay lines differ by nanoseconds, and the clock's arrival by tens of picoseconds. The FPGA tool's timing model knows every path at two extremes of the chip's speed (the slow and the fast corner); the build exports them with the bitstream. The real board lies between the corners at about the same fraction λ for every path, which `calibrate_delays()` (part of `calibrate()`) measures with the test signal, which reaches

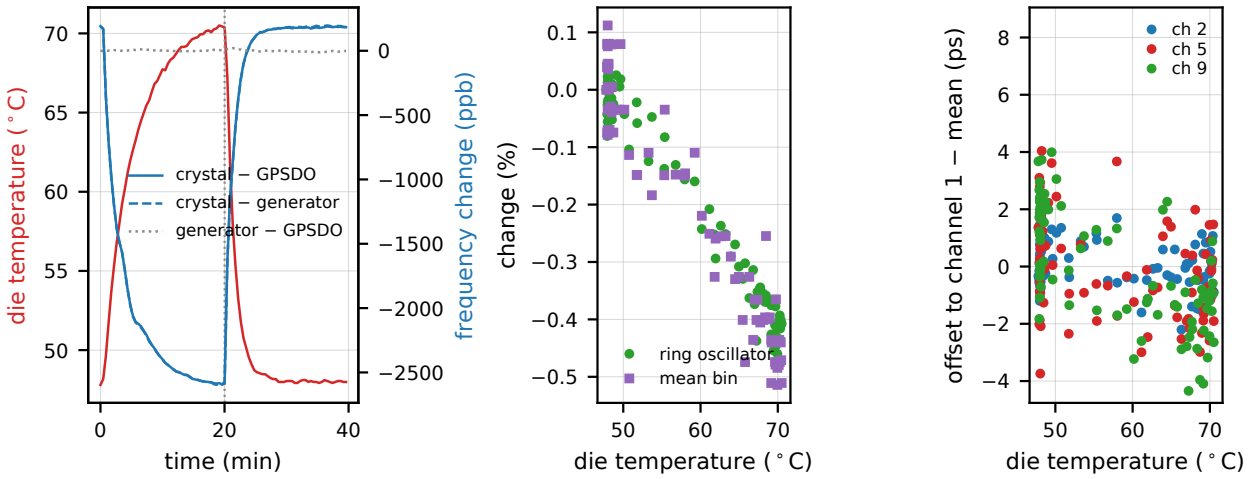


Figure 9: Measured (2026-10-31). The board's fan off for 20 minutes, then on. Left: the die's temperature, and the frequency changes of the crystal against the GPSDO and against the function generator (on top of each other) and of the generator against the GPSDO. Middle: the ring oscillator's frequency and the mean bin width of all channels' tables. Right: channels 2, 5 and 9 against channel 1 on the test signal, recalibrated at every step.

every channel's input stage: $\lambda = 0.062$ here, with 39 ps rms of residuals on the test paths. With `connectors` on (the default) each channel's delay from its connector is subtracted (3.25–3.89 ns, figure 10), so that the channels' times refer to their connectors, up to the board's traces and the model's error.

The same measurement gives each line's **polarity offsets**: the test signal clocks each channel's own toggle, whose transitions then alternate in polarity at one phase; the difference of the two polarities' times is the offset to add to the falling transitions. On this board the toggle paths' offsets are −57 to 21 ps and the test paths' −358 to −211 ps; all are corrected in the FPGA.

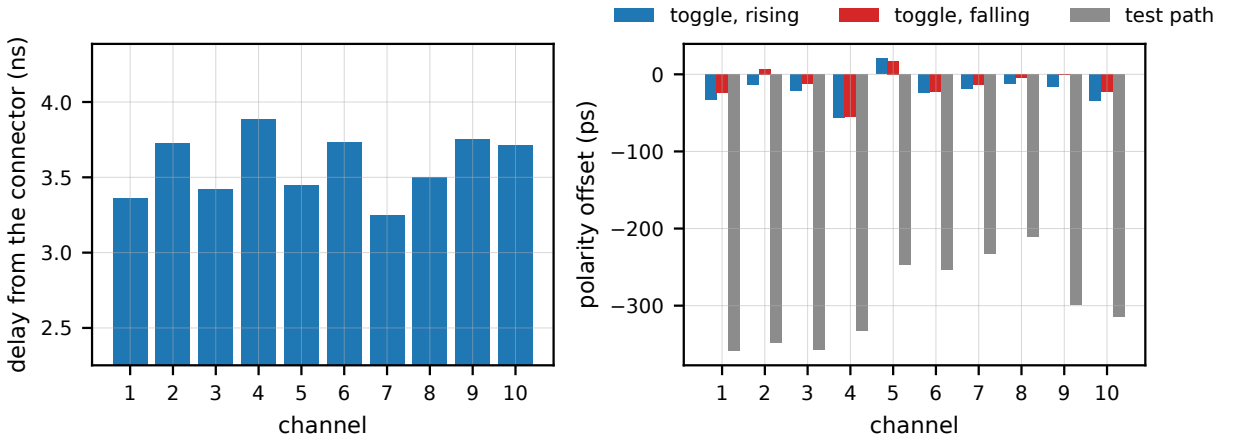


Figure 10: Measured (2026-10-30). Left: the delay from each channel's connector to its delay line, subtracted from its times. Right: the polarity offsets measured by `calibrate_delays()`, corrected in the FPGA.

What the correction does not include is the board's traces from the connectors to the FPGA's pins, the cables, and the model's error on each path. For the best agreement between two channels, measure it: one signal split into both with equal cables, then the cables swapped (the source's and the cables' skew cancel), and set the difference as a channel offset.

5.7 Precision

The internal test signal reaches all ten channels at once, from one flip-flop, so the difference of two channels' times does not depend on the test signal's own jitter. Its rms at one phase of the period is the two channels' precision together; with all 45 pairs of the ten channels, the variances $\sigma_{ij}^2 = \sigma_i^2 + \sigma_j^2$ give each channel's own (figure 11). The pairs are 5.6–29.5 ps rms (median 15.2), one channel 9.1–12.1 ps (median **10 ps rms**), over eight phases of the period, both edges, without dither.

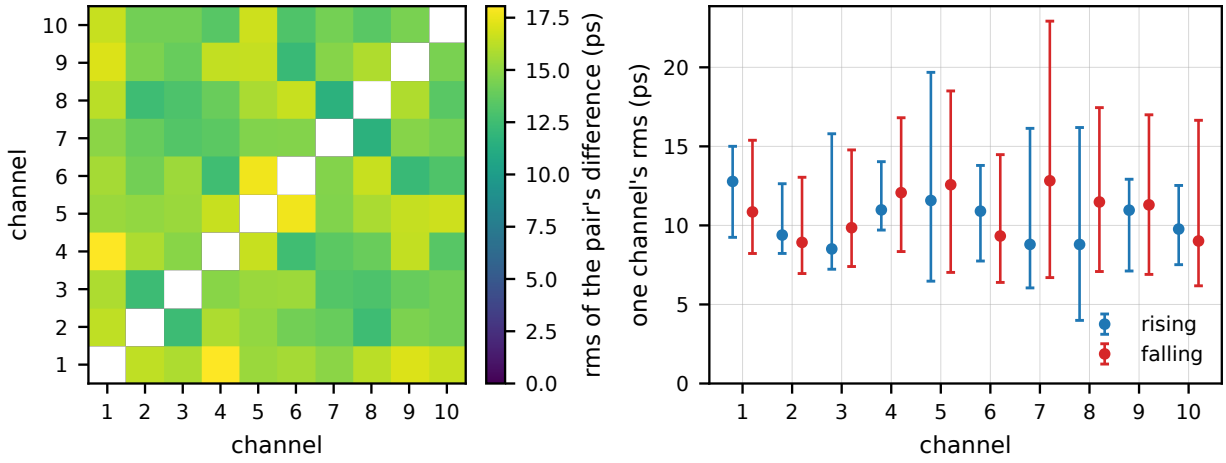


Figure 11: Measured (2026-10-30). The internal test signal on all ten channels. Left: the rms of each pair's time difference (rising edges, the median over 8 phases). Right: each channel's own rms from all pairs (median over the phases; bars: the range).

Moving the test signal over the period tests the tables: the difference between two channels, whose bins differ, would change with the phase if a table were wrong. Over eight phases it moves by 5.5–13.8 ps rms (figure 12): that is how well two channels' calibrations agree. Asynchronous edges from outside test the same: a function generator's 1 MHz on channel 1 (1.0 million edges), binned by their phase in the period, has a time error whose mean varies by only 8.2 ps rms (29 ps peak to peak) over the period (figure 13). The intervals of those edges spread by 171 ps rms, which is the generator's and the wire's jitter, not the tagger's.

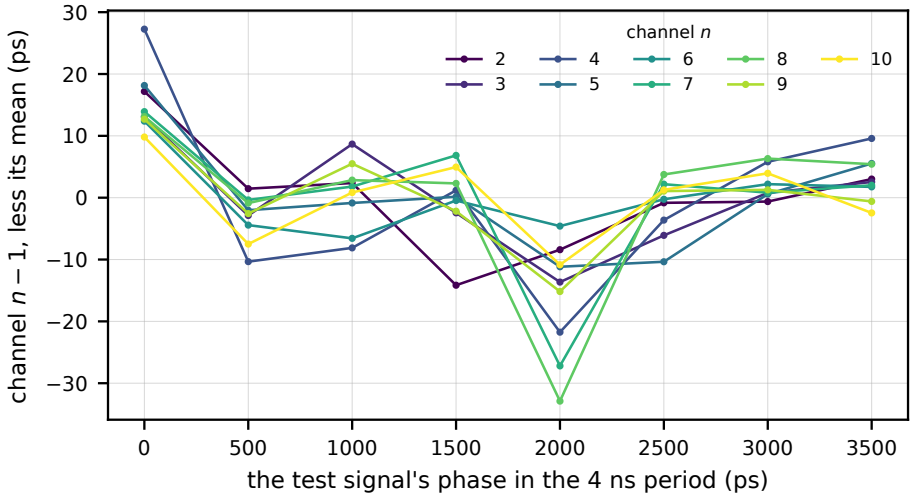


Figure 12: Measured (2026-10-30). Channel n less channel 1 on the test signal at eight phases of the 4 ns period (each less its mean): the calibrations agree to 5.5–13.8 ps rms.

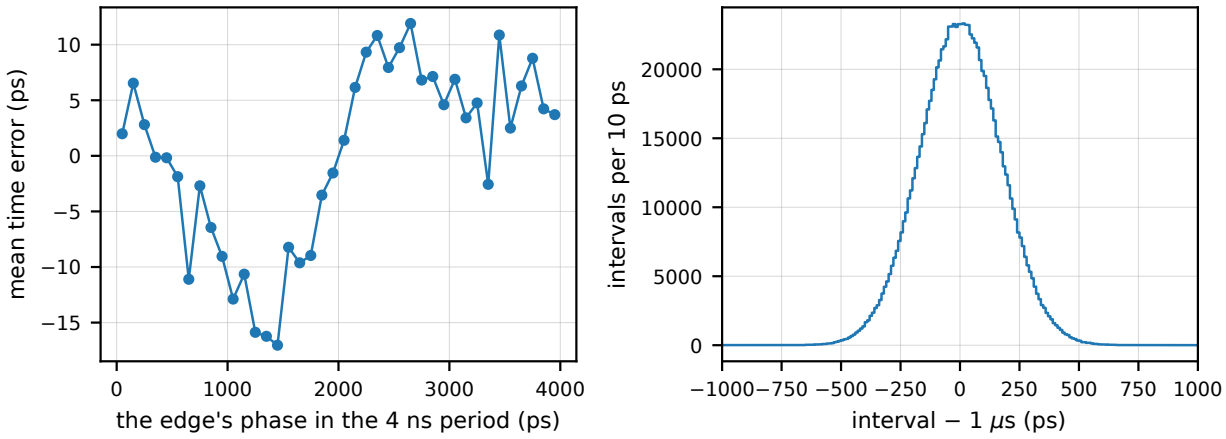


Figure 13: Measured (2026-10-30). A function generator's 1 MHz square on channel 1 (header wires). Left: the mean time error of its edges against their phase in the 4 ns period (100 ps bins): flat to 8.2 ps rms. Right: its intervals, whose spread is the generator's and the wiring's.

Clean edges from outside. One source on two channels cancels the source's own jitter, as the test signal does inside. A function generator (SRS DS345) drove the fork adapter of section 3.3: one 50 Ω coax, terminated at the socket and fanned out to pins 15 and 17 (channels 7 and 8), 0 to 3 V at the pins (2026-10-31; `tools/record_manual_data.py --session edges`, about a million edges per setting). On a 1 MHz square the two channels differ by 21.5 ps rms on rising edges: **one channel 15.2 ps rms** (figure 14), with the asynchronous edges' quantization included, without dither. Falling edges give 22.2 ps, the level input instead of the toggle 16.2 and 18.3 ps, and dither adds the quantization again (20.0 ps). At 10 and 30 MHz the rising edges give 14.9–15.2 ps and the falling ones 20.5–22.2 ps: busy inputs (60 million edges a second on the two) cost nothing; no event was flagged or dropped. The difference's mean is flat over the 4 ns period to 6.7 ps rms. The generator's own edges, meanwhile, jitter by 1027 ps rms from period to period at 1 MHz (108 ps at 10 MHz): the difference removes all of it.

Slow edges. Sine waves of the same amplitude cross the threshold at a known slew rate, $\pi f V_{pp}$: 9.5 mV/ns at 1 MHz, 283 mV/ns at 30 MHz. At 30 MHz the precision is the square wave's (15.2 ps); at 1 MHz it is 155 ps (rising) and 85 ps (falling). The extra jitter of the slower sines is a voltage noise of 0.8–5 mV rms at the inputs, divided by the slew rate, and much of it is the board's own: the difference's mean swings by 163 ps rms once per 4 ns period (figure 14, right), about 1.5 mV of the 250 MHz sample clock reaching the inputs. The rest changes in steps (the rms within one acquisition moved by up to a factor 1.5 between steady stretches), as the board's activity does. The two inputs' thresholds also differ: on the 1 MHz sine the difference's mean moved by 560 ps against the square wave's, about 5 mV of threshold. Hence the rule: give the inputs edges faster than about 0.3 V/ns (1 V in 3 ns), and none of this matters.

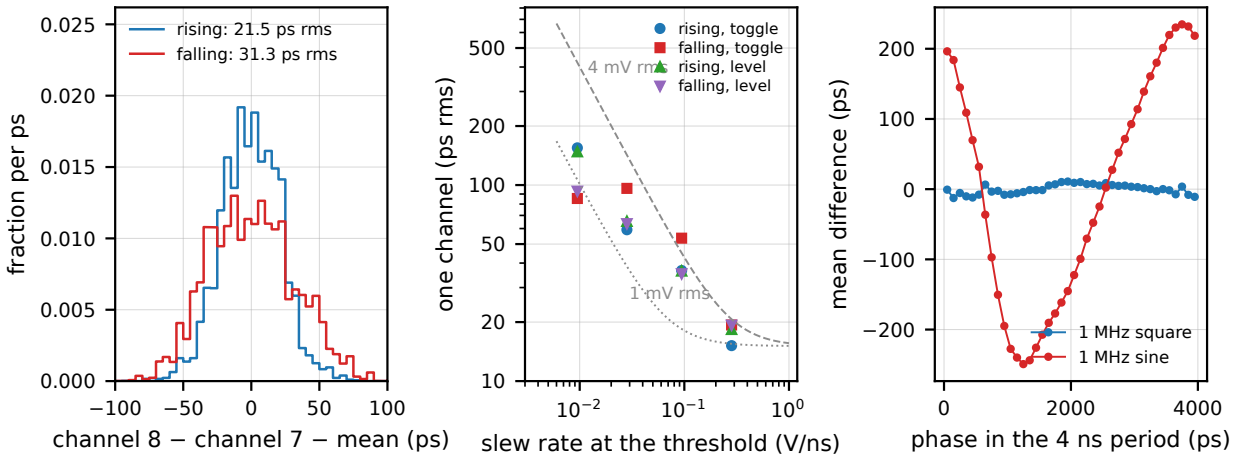


Figure 14: Measured (2026-10-31). One source on channels 7 and 8 through the fork adapter. Left: their difference on a 1 MHz square (1.0 million edges, toggle input, no dither). Middle: one channel's rms (the difference's over $\sqrt{2}$) on sine waves of 1–30 MHz against their slew rate at the threshold, with the square waves' precision combined with a voltage noise of 1 and 4 mV rms (lines). Right: the difference's mean against the edges' phase in the sample clock's 4 ns period: flat on the square, a 250 MHz swing on the slow sine.

5.8 The error budget

term	size on this board
quantization of the bins	10.9–13.6 ps rms; dither adds about as much again
one channel's precision, internal test signal	9.1–12.1 ps rms (median 10), at one phase
one channel on clean external edges	15.2 ps rms rising, 22.2 ps falling (all phases, quantization included)
slow edges	0.8–5 mV rms at the inputs divided by the slew rate; negligible above 0.3 V/ns
two channels' calibrations, over the period	agree to 5.5–13.8 ps rms
the calibration on asynchronous edges	8.2 ps rms over the period
delays to the connectors	the model at the measured scale: 39 ps rms on the test paths; the board's traces not included
the sample clock's jitter	common to all channels of a board; between two boards, part of their clocks' comparison (section 19)

6 Acquisitions and the time base

```
tt.start()                          # now (t = 0 about 1 us ahead), until tt.stop()
tt.start(duration=10.0)             # now, stopped in hardware after 10 s
tt.start(arm_channel=7)             # t = 0 at the next event of channel 7 (a PPS)
```

The FPGA counts its 250 MHz samples from the moment the design is loaded; an **acquisition** picks a sample k_0 as its zero. Tags are times since k_0, in ticks of 4 ns/4096 (60 bits: 13 days), which programs see as integer picoseconds. An acquisition starts

- **now**: 256 samples (about 1 µs) after the command;
- **at a sample** (`start(at=k)`): to start several things together, or at a computed time;
- **armed** on a channel (`start(arm_channel=n)`): at the next event of channel n, which is then at $t = 0$ (up to the channel's connector delay, which is subtracted from its time but not from the sample). With a PPS this aligns the acquisition with a second.

It ends at `stop()`, at a given sample, or after a **duration** set at the start, in hardware: the last tags are those of events before the stop's sample. Each start puts an **EPOCH** record into the stream, with the acquisition's number and k_0, so a program always knows which acquisition a tag belongs to.

Events are inside or outside an acquisition by their 4 ns sample. An event's time can lie a little before 0 or past the stop by its channel's offset (the connector delay, a user offset): that is the event's time, correctly; only the decision used the sample. Starts and stops are exact: no event from before the start and none from after the stop enters, and none inside is lost, also while events arrive at every clock (tested with the generator at 10 Mevents/s, section 12).

`tt.acquisition()` returns whether one is `running` or `armed`, its number and k_0, and the stop's sample. Starting a new acquisition while one runs ends it.

```
ACQ START FOR 10
ACQ ARM 7
ACQ STOP
ACQ?
```

7 The stream: records, the ring and the PC

The tags leave the FPGA as 32-bit **records**: an event is one word (its source, its edge and the low 26 bits of its time), and control records carry the higher bits of the time (only when they change), the EPOCH of each acquisition, losses (DROP), and the IN2 meter's windows (METER). The FPGA writes them into a **ring** in the board's memory (16 to 96 MB of contiguous memory, as much as the service can allocate: 32 MB on the board used here), and a small compiled program on the board's processor, the **ring pump**, sends them to every client of TCP port 5026 as frames of up to 256 kB. Port 5025's `RECORDS STREAM` relays the same frames. `docs/PROTOCOL.md` defines every word and the frames.

7.1 Rates

The merger takes one event per clock, 125 Mevents/s; the ring and the network take less (figure 15). Over 20 seconds the stream carried **7 Mevents/s** to a PC without loss; at 8 Mevents/s 0.4 % were dropped. Shorter bursts are absorbed by the ring: 10 Mevents/s for 2 s arrived complete. Nothing is ever lost silently. When the ring is full, the FPGA's writer waits, the channels' FIFOs fill and their events are dropped and counted there (`drops` in the counters, DROP records in the stream); a client that falls behind loses whole frames, which the next frame's header reports in bytes. The coincidence engine and the conditional filter work before all this, at the full rate.

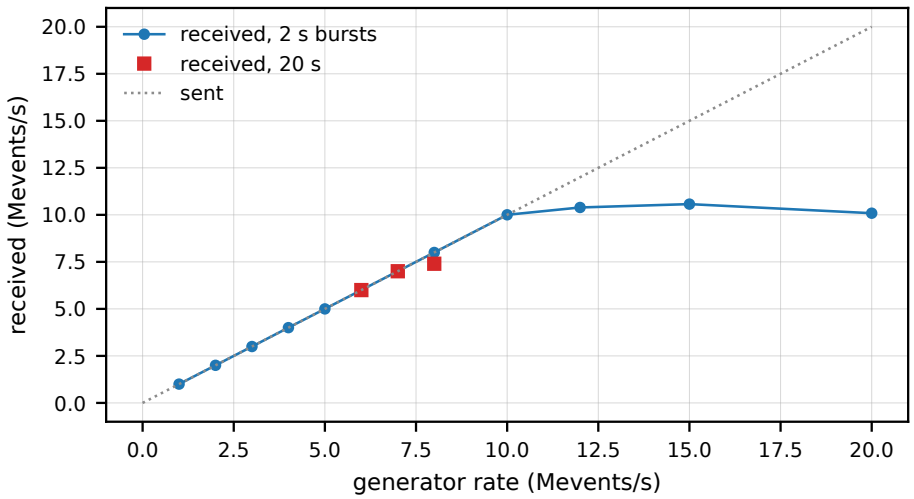


Figure 15: Measured (2026-10-30). The generator's events received on a PC against its rate: complete in bursts of 2 s up to 10 Mevents/s (the ring holds the excess), and over 20 s up to 7 Mevents/s.

7.2 On the PC

The client `RecordStream` delivers the frames; `analysis.parse_frame` and `analysis.Decoder` turn them into channel numbers and times in ps (numpy, a few Mevents/s); the tags of different channels are in order only to within a few samples, so sort by time when it matters. **ttcore**, the C++ library with a Python module, receives, decodes, sorts and measures at the stream's full rate and can record the frames into a `.ttbin` file (`tools/ttcore/README.md`):

```
import ttcore
stream = ttcore.StreamClient("192.168.0.102")
rates, g2 = ttcore.CountRate([1, 2]), ttcore.Correlation(1, 2, 100, 400)
stream.attach(rates); stream.attach(g2)
stream.record("run.ttbin")                        # optional: the frames as received
stream.start()
tt.start(duration=10.0)
...                                               # rates.rates(), g2.g2() at any time
stream.stop()
ch, t, controls = ttcore.FileReader("run.ttbin").read()      # later: the sorted events
```

`analysis.read_recording(path)` reads the same files with numpy. The channel numbers in programs: 1–10 the channels (falling edges −1 to −10), 11–18 the markers, 21–28 the coincidence engine's virtual channels, 31 and 32 the generator.

8 Markers

```
tt.apply({"markers": {"1": {"enable": True, "edge": "both"}, "2": {"enable": True}}})
```

The eight pins DIO0_N–DIO7_N (E1 pins 4–18, even) are **markers**: slower signals timed to 1 ns beside the channels, for gates, frame and line clocks of a scanning microscope, triggers, the state of a modulator. Each pin is sampled four times per 4 ns sample, 1 ns apart; an edge is placed in the middle of the nanosecond in which it happened, so its time has the quantization of a 1 ns bin, 289 ps rms. A pulse shorter than 4 ns that leaves the level as it was is counted as a **glitch** and not timed. The markers share one offset (`marker_offset`) and have no dead time or divider; their events are channels 11–18 in programs (falling edges −11 to −18). `tt.hardware()["marker_levels"]` reads the eight pins' levels at any time.

The markers' samples reach the timing logic about three samples after they are taken; the driver subtracts this delay (`MARKER_DELAY_S`, 12.705 ns) with the connector corrections, so that markers

and channels on the same edge agree. Measured with a wire joining DIO0_P and DIO0_N (channel 1 and marker 1 on the same edges), the difference spreads as a 1 ns box (291 ps rms) around 265 ps (figure 16).

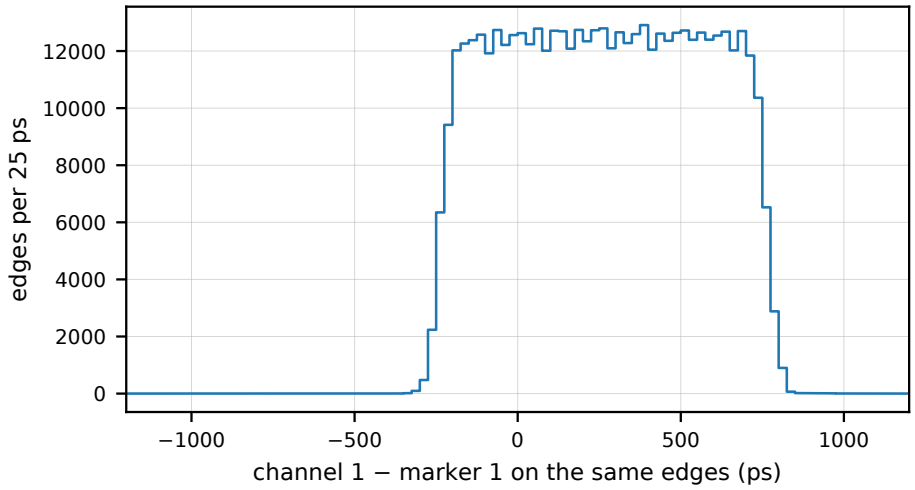


Figure 16: Measured (2026-10-30). Channel 1 less marker 1 on the same edges (500 k edges, a wire from DIO0_P to DIO0_N): the marker's 1 ns bins.

A marker can gate the coincidence engine (section 9): it then counts only while the marker's level is high (or low).

9 The coincidence engine

```
tt.apply({"coincidence": {"enable": True, "channels": [1, 2, 3], "window": 2e-9,
                          "masks": {"1": {"channels": [1, 2]},
                                    "2": {"channels": [1, 2, 3], "exclusive": True, "virtual": True
    ↪ }}}})
tt.clear_coincidences()
```

and `tt.coincidences()` returns `{"patterns": {"1,2": n, ...}, "masks": {"1": n, ...}, "clusters": n, ...}`.

The engine counts coincidences in the FPGA, as the events pass, with the same definition as `analysis.coincidences` and ttcore on a PC (section 15.1): an event that does not belong to the open cluster starts one, the following events within the **window** of its start join it, and the cluster's **pattern** is the set of channels that took part (a channel's falling edges count as that channel).

- **Lanes**: up to eight of channels 1–10 and the generator's 31 and 32 (`coincidence.channels`); seven with a gate. The engine sorts its lanes' events into exact time order first, in a sorter that holds eight events, so it needs no more than eight lanes firing together.
- **Window**: up to 16.4 µs.
- **Patterns**: every cluster increments the count of its pattern (4096 counters of 48 bits), and the cluster count.
- **Masks**: eight masks of lanes, each *inclusive* (the cluster contains the mask's channels) or *exclusive* (exactly them), with a counter each.
- **Virtual channels**: a mask marked `virtual` puts an event into the records at the start of each matching cluster, channel $20 + m$ in programs (21–28): coincidences as a channel of their own.
- **Engine-only channels**: a channel with `record` false (the generator: `generator.record`) feeds the engine but not the stream, so the engine counts at rates the stream cannot carry;

it took the generator's pairs at 60 Mevents/s, with counts that agree with a simulation of its draws.

- **Gate**: `coincidence.gate.marker` (1–8) counts lane events only while that marker's gate is open, from its rising edge to its falling edge (`invert`: the other way); closed at every start until its first opening edge.

Clearing takes 33 µs; clusters that close meanwhile are not counted.

How it is verified

The board test compares the FPGA's patterns with `analysis.coincidences` on the same recorded events: equal for the generator's pairs (320814 clusters), for the test signal on eight channels, and for the gated engine; the masks' counts agree with the patterns; the virtual events come exactly at the clusters' starts. Engine-only, the generator's pairs at 15, 30 and 60 Mevents/s give 8.98, 16.18 and 26.67 million clusters per second, as a simulation of the generator's draws predicts.

Late events. The sorter emits an event when no earlier one can still come (from how long the lanes' events have waited). When more events come together than it holds, or a lane's FIFO backs up (a single source near the 62.5 Mevents/s that the merger takes from one channel), an event can come out of order: it is counted as `late` and skipped by the engine. Below about 65 Mevents/s in all, with no more than eight lanes, there are none.

10 The conditional filter

```
tt.apply({"channels": {"1": {"enable": True, "filter": True}, "2": {"enable": True}},
          "filter": {"triggers": [2], "window": 20e-9}})
```

A pulsed laser's sync at 80 MHz is more than one channel can pass on (62.5 Mevents/s) and far more than the stream carries; for a lifetime only the syncs that a photon follows matter (section 16.2). On channels 1 and 2 the **conditional filter** keeps an event only if a **trigger** channel (`filter.triggers`, any of 1–10) has an event within `filter.window` after it (up to 120 ns: the period of an 8.3 MHz laser; slower syncs stream without a filter). The filtered events wait in a shift register for the window, and the decision is made per 4 ns sample: a trigger up to 4 ns before an event in its sample keeps it, and so does one up to the window plus 4 ns after it. The tags themselves are exact; the PC computes the histogram. The channel's `events` counter counts all its events, kept or not.

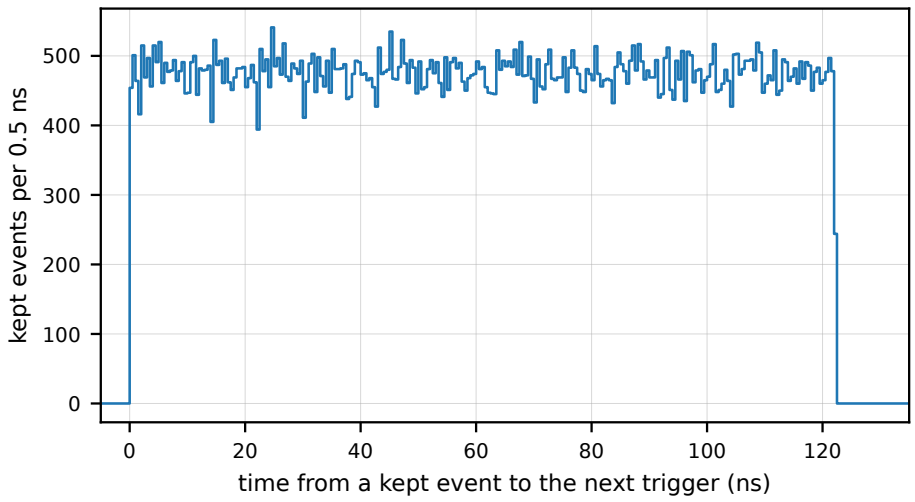


Figure 17: Measured (2026-10-30). The ring oscillator (97.6 MHz, divided by two) on channel 1, filtered by the test signal's edges on channel 2 with a 120 ns window: the time from each kept event to the next trigger is uniform from 0 to the window's end (6.1 kept per trigger), nothing dropped.

11 References and time scales

The board's own crystal oscillator clocks the hardware, and the tags count its ticks. A crystal is off by some parts per million and wanders with temperature; timetagger does not lock it to an external reference. Instead, a reference is measured along with the tags, and the tags are re-expressed in the reference's time scale afterwards, on the PC. Three references are possible:

- **a 10 MHz sine on IN2**, measured by a phase meter in the FPGA;
- **a tagged clock** (a 10 MHz square through a divider) or a **PPS** on any channel;
- **an armed start** on a PPS, which puts $t = 0$ on a second's edge.

11.1 The IN2 phase meter

```
tt.apply({"reference": {"meter_window": 0.01}})      # a METER record every 10 ms
```

The meter correlates IN2 with a 10 MHz sine and cosine made from the board's own clock (exactly 2/25 of the 125 MHz sample rate) over windows of a whole number of periods, and puts one **METER** record per window into the stream: the in-phase and quadrature sums, whose angle is the reference's phase against the board's clock. A crystal that runs fast by y makes that phase drift by $2\pi \cdot 10\,\text{MHz} \cdot y$ per second; the phase, unwrapped, is the offset between the board's time and the reference's. On the PC:

```
hw = tt.hardware()                                    # (after setting the window)
...                                                   # an acquisition: ch, t, controls
tm, offset, amplitude = analysis.meter_offsets(controls, window_clocks=1250000,
                                               start_clock=hw["meter_start_clock"], k0=k0)
t_ref = analysis.to_reference(t, tm, offset)          # every tag in the reference's time
```

`window_clocks` is the window in 8 ns clocks (here 10 ms), `k0` the acquisition's start sample from its EPOCH record (or `tt.acquisition()["k0"]`). The offsets are defined up to a constant (the reference's phase origin); `to_reference` interpolates them between windows.

The phase is known only modulo a period of 10 MHz, 100 ns, so consecutive windows must not differ by half of it: with 10 ms windows the crystal must be within 5 ppm of the reference. This board's crystal is −4.50 ppm off (section 19.3), close to that limit. Give `meter_offsets` the rough offset as `y0` (reference − board time per board time; from a PPS's intervals in board time, $y_0 = 1 - \overline{\Delta t}/1\,\text{s}$), or use shorter windows (1 ms: 50 ppm):

```
pps = t[ch == 7]
y0 = 1 - np.median(np.diff(pps)) / 1e12
tm, offset, amplitude = analysis.meter_offsets(controls, 1250000, hw["meter_start_clock"], k0, y0=y0)
    ↪
```

11.2 Tagged references and PPS

A reference square wave or a PPS on a channel gives the same from its own tags: `analysis.reference_edges(t_ref_channel, period_ps, smooth)` numbers the edges by the elapsed time (missing edges are allowed), averages the offsets over `smooth` neighbouring edges, and returns the offset reference − board along the acquisition for `to_reference`. A 10 MHz square needs a divider on its channel (every 10th edge: 1 MHz), a PPS none.

An acquisition armed on the PPS (`tt.start(arm_channel=7)`) starts at a second's edge (the first PPS is tagged a few nanoseconds before 0: $t = 0$ is the start of its 4 ns sample, and its tag lies earlier by its place in the sample and the connector delay; −6.25 ns in section 19.3): with a GNSS-disciplined PPS, $t = 0$ is a UTC second (the PC's clock names which), and two taggers armed on their own GNSS-disciplined PPS share their zero to the receivers' agreement, tens of nanoseconds. Section 19 shows all of this measured against a GNSS-disciplined oscillator.

12 Test signals and self-tests

Three internal sources let the instrument check itself without any wiring:

- the **ring oscillator** (`ring`): the random edges of the calibration;
- the **test signal** (`test`): 25 MHz from a second clock generator in the FPGA, divided to $12.5\,\text{MHz}/2^n$, whose phase moves in steps of 17.86 ps on average (`tt.set_phase(steps)`, 224 steps per 4 ns). It reaches every channel's input stage (`input test`, or `test_toggle` through the channel's toggle) and, with `test.sata_out`, leaves on the SATA connector S1, so that a straight SATA cable from S1 to S2 loops it into channels 9 and 10 (figure 18);
- the **generator** (`generator`): random events at a chosen rate (channel 31), a fraction of them copied to channel 32 a fixed delay later. Its times are exact, so it tests the stream, the coincidence engine and the analysis with known answers.

```
tt.apply({"generator": {"rate": 1e6, "pair_fraction": 0.5, "pair_offset": 10e-9}})
```

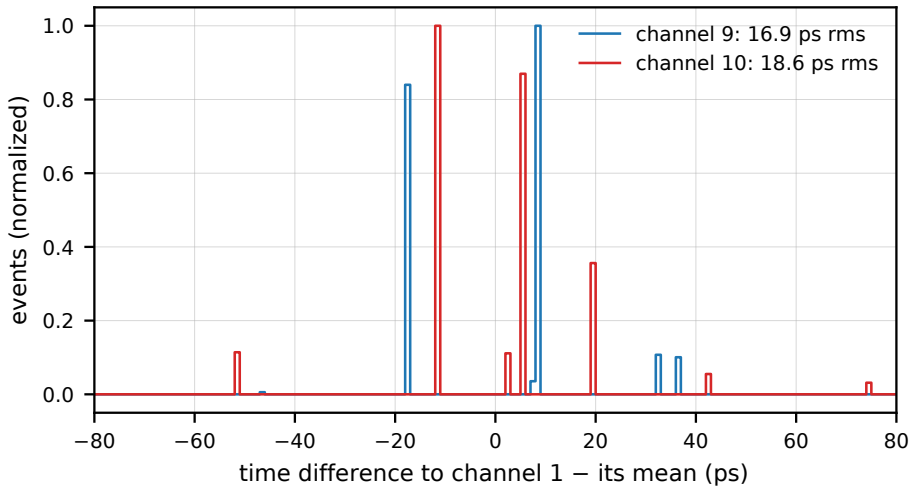


Figure 18: Measured (2026-10-30). The test signal out on S1 and back through a SATA cable into channels 9 and 10, against channel 1 on its internal path (level inputs, no dither, one phase): 16.9–18.6 ps rms, the two channels together with the cable and the outputs.

12.1 The board test

`make test-hw` in `timetagger/` (or `scripts/test_hw.py` with the options `--external`, `--sata`, `--markers` for the optional wiring) runs about 90 checks in ten minutes, printing each and saving the numbers as JSON on the board (`docs/VERIFICATION.md` keeps the dated records):

- the ring oscillator's frequency and the calibration of all ten channels (bins, rms quantization, no flagged hits);
- the delays' scale and the polarity offsets; the channels' agreement and precision on the test signal at eight phases; the toggles' polarities;
- the generator through the stream (counts, exact pairs, nothing lost) and the lossless rate;
- dead time and divider; starts at a sample, durations, armed starts; exact starts and stops with the generator at 10 Mevents/s;
- the coincidence engine against the PC's clusters (recorded, engine-only, on the test signal, gated by a marker); the conditional filter;
- with the wiring: the SATA loop, an external source's edges, the markers against a channel.

How it is verified

Version 2.0 passed all 89 checks on the board used for this manual (build 0x9b2ec69e, 2026-10-30), with the SATA cable, a function generator on DIO0_P and a wire from DIO0_P to DIO0_N.

13 Software

13.1 On the board and on a PC

On the board (Jupyter, scripts): `from pynq.overlays.timetagger import DRIVER; tt = DRIVER()` (after `import rpfpga`). On a PC: `TimeTaggerClient(host)` from `timetagger/control` with `common/python` on the path; it needs only Python's standard library. Both have the configuration methods of every design of the framework (`config`, `apply`, `status`, `presets`, `load_preset`, `save_preset`, `safe`) and timetagger's own:

method	what it does
`start(at, duration, arm_channel)`, `stop()`, `acquisition()`	acquisitions (section 6)
`counters()`, `clear_counters()`	events, flagged, drops per channel; the merger, the ring, the markers, the generator
`calibrate(channels, seconds)`	the code-density calibration and the delays (section 5)
`calibrate_delays(channels)`	only the delays' scale and the polarity offsets
`histogram()`, `tables()`	the live calibration histogram; the kept tables
`coincidences()`, `clear_coincidences()`	the coincidence engine's counts
`set_phase(steps)`	the test signal's phase
`hardware()`	status bits, the ring, the meter, the markers' levels, the die's temperature
`worker()`	the service's worker: bytes streamed, live calibrations, rejected histograms

13.2 Records and measurements in Python

`RecordStream(host)` delivers the stream's frames; `analysis` decodes them and holds the numpy reference of every measurement:

function	
`parse_frame`, `Decoder`, `decode`	frames to (channels, times in ps), the control records
`histogram(t_start, t_stop, bin, bins)`	start–stop (TCSPC)
`correlation(t_a, t_b, bin, bins)`, `correlation_norm`	cross-correlation and its $g^{(2)}$ normalization
`coincidences(ch, t, channels, window)`	coincidence patterns (the engine's definition)
`find_offset`, `track_offset`	the offset (and its drift) between two stations' tags (section 17.2)
`meter_offsets`, `reference_edges`, `to_reference`	tags in a reference's time scale (section 11)
`time_error(t, period, reference)`	a periodic signal's time error, for stability analyses
`read_recording(path)`	a `.ttbin` recording

13.3 ttcore

`tools/ttcore` is a C++17 library with a Python module (pybind11) that receives the stream, decodes and sorts it and runs measurements as the events arrive, at the stream's full rate: `CountRate`, `Counter` (counts per time bin), `Correlation` (with $g^{(2)}$), `StartStop`, `Coincidences`, and `StreamClient.record` for `.ttbin` files, read back by `FileReader`. Every measurement is tested against the numpy reference. Build it with CMake (`tools/ttcore/README.md`).

13.4 TCP

Every method is a text command on port 5025 (appendix A), so any language or instrument-control program can drive the tagger; the records stream is binary, on a connection of its own (`RECORDS STREAM`, or port 5026 directly).

13.5 Stability analyses

`tools/stability.py` analyses a tagged clock or PPS from a `.ttbin` recording with the common stability tools: time error, ADEV, MDEV, TDEV, MTIE with ITU-T masks, noise types and a PDF report:

```
python3 timetagger/tools/stability.py report run.ttbin --channel 7 --period 1e12 -o pps.pdf
```

Section 19 uses it.

14 Specifications

Measured on the board used for this manual (build 0x9b2ec69e, die at about 49 °C with a fan) unless marked.

quantity	value
channels	10: 8 × 3.3 V LVCMOS (E1 DIO0_P–DIO7_P), 2 × 1.8 V differential (SATA S2)
tick	4 ns/4096 = 0.976 562 5 ps; 60-bit times (13 days)
delay line	80 carry cells, 160 taps; 128–139 codes per 4 ns; mean bins 28.8–31.2 ps, largest 68–110 ps
rms quantization	10.9–13.6 ps
precision, one channel	9.1–12.1 ps rms (median 10 ps), internal test signal
precision on clean external edges	15.2 ps rms rising, 22.2 ps falling, one channel, all phases (coax to channels 7, 8; 1–30 MHz; edges faster than 0.3 V/ns)
precision, two channels	5.6–29.5 ps rms (median 15.2 ps), internal test signal
calibrations' agreement	5.5–13.8 ps rms over the 4 ns period (two channels)
calibration on asynchronous edges	8.2 ps rms over the period
SATA pairs through a cable	16.9–18.6 ps rms against an internal channel
dead time	about 5 ns (the delay line), plus a programmable 0–262 µs
divider, offset	1–65536; ±2.05 µs in the FPGA
delays to the connectors	3.25–3.89 ns, subtracted; the model's scale $\lambda = 0.062$, 39 ps rms on the test paths
polarity offsets	toggle paths −57 to 21 ps, corrected
markers	8 × 3.3 V LVCMOS (E1 DIO0_N–DIO7_N), 1 ns bins (291 ps rms against a channel)
rates	125 Mevents/s through the merger and to the coincidence engine (design); to a PC 7 Mevents/s sustained (20 s), 10 Mevents/s in 2 s bursts (32 MB ring)
coincidence engine	8 of channels 1–10 and the generator; window up to 16.4 µs; 4096 patterns, 8 masks, virtual channels, a marker gate
conditional filter	channels 1, 2; window up to 120 ns
acquisitions	now, at a sample, armed on a channel; duration in hardware
references	IN2 10 MHz (phase meter), a tagged clock or PPS on any channel; a GPSDO's PPS re-expressed through the meter: seconds to 35 ps rms, ADEV 4.3×10^{-11} at 1 s (section 19.3)
the board's crystal	−4.50 ppm against a GPSDO, −128 ppb per kelvin of the die; ADEV 1.8×10^{-9} at 1 s, 1.9×10^{-8} at 1000 s (the room's temperature, fan on); timestamps re-expressed in a reference's time scale on the PC
temperature (die 48–71 °C)	bins −0.021 %/K (live calibration follows); channels' offsets 0.02–0.15 ps/K after calibration (section 5.5)

14.1 Troubleshooting

symptom	what to check
no events	the channel enabled; the edge; the level reaches 2 V; an acquisition running; `counters()` shows `events`
`flagged` counts	two transitions within about 5 ns: a short pulse on a level input (use a toggle), or ringing on a wire (section 3.3)
`drops`, DROP records	more events than the stream takes: reduce the rate, use dividers, the filter or engine-only channels
lost bytes in frame headers	the PC does not keep up: ttcore instead of Python, fewer clients
times comb-like in histograms	dither off (turn it on)
two channels disagree by a constant	cables and detector latencies: measure and set offsets (section 5.6)
`late` coincidences	more than eight lanes firing together, or a single lane near 62.5 Mevents/s
`calibrate()` warns about the ring	the ring oscillator near a clock subharmonic: tell the maintainers (the tables may be biased)

Part II
Time tagging in quantum labs

15 From time tags to measurements

In a photon-counting experiment the primary data are detection events: a detector clicked, at some time. A time tagger keeps exactly that, a list of (channel, time) pairs, and nothing is decided while the experiment runs. Every measurement of Part II is computed afterwards from the list, and can be computed again with another bin width, window, gate or channel combination from the same data. A start–stop histogram for a lifetime, a correlation for $g^{(2)}$, coincidence patterns, rates, a time series of a clock's error all come from the same recording. This is the main difference from instruments that make one measurement in hardware: a time-to-amplitude converter and a multichannel analyser make one start–stop histogram; a coincidence unit counts one pattern in one window. The price is the data rate: this tagger streams about 7 Mevents/s to a PC without loss (section 7). The coincidence engine (section 9) and the conditional filter (section 10) reduce what has to be streamed when the raw rates are higher.

This section introduces the three histograms used throughout, the accidental coincidences that every coincidence measurement carries, and how the detectors and the tagger together set the timing.

15.1 Three histograms

Figure 19 shows two channels' tags and the three ways of pairing them.

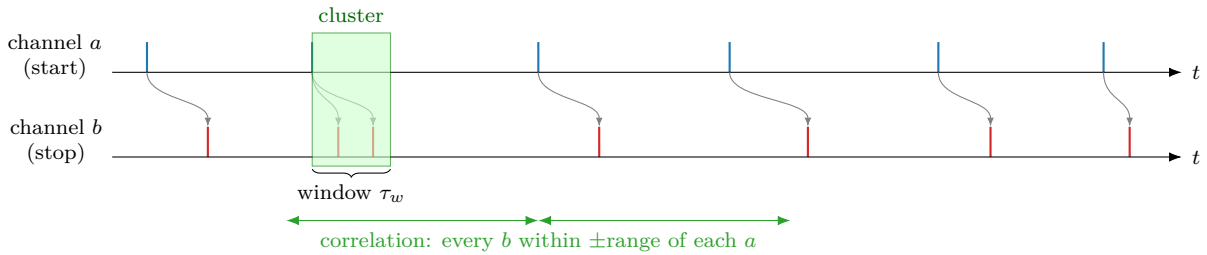


Figure 19: Two channels' time tags and the three ways of pairing them: the **start–stop** histogram takes each stop with the last start before it (grey), the **correlation** takes every pair within a range (green arrows: the pairs of one start), and **coincidence counting** groups events into clusters within a window.

Start–stop (TCSPC). For every event of the stop channel, its time since the last event of the start channel. With a pulsed laser's sync as start and a detector as stop this is the classic time-correlated single photon counting histogram of a fluorescence lifetime [1, 2]. With a sync rate far above the photon rate the roles are usually reversed in hardware TCSPC (the photon starts, the next sync stops), because a converter is busy after each start; a time tagger has no such restriction, only the stream's rate, which the conditional filter takes care of (section 16.2). `analysis.histogram(t_start, t_stop, bin_ps, bins)`; ttcore's `StartStop`.

Correlation. Every pair of an event a and an event b with $-R \le t_b - t_a < R$, binned into bins of width $\Delta\tau$: the cross-correlation $G(\tau)$. For two independent (Poisson) channels with N_a and N_b events in a recording of duration T, each bin expects

$$\bar{G} = \frac{N_a N_b \, \Delta\tau}{T}, \tag{2}$$

and the normalized second-order correlation is $g^{(2)}(\tau) = G(\tau)/\bar{G}$: 1 for uncorrelated light, above 1 for bunched light (thermal light, photon pairs), below 1 for antibunched light (a single emitter). `analysis.correlation(t_a, t_b, bin_ps, bins)` and `analysis.correlation_norm(n_a, n_b, duration_ps, bin_ps)`; ttcore's `Correlation`, which also returns $g^{(2)}$. A correlation of a channel with itself counts each event with itself at $\tau = 0$; subtract N_a there.

Coincidences. Events are grouped into *clusters*: an event that does not belong to the open cluster starts one, and the following events within the window τ_w of its start join it. Each cluster has a *pattern*, the set of channels that took part. `analysis.coincidences(ch, t, channels, window_ps)` returns the count of every pattern; the FPGA's engine (section 9) counts the same patterns with the same definition at the full rate, and ttcore's `Coincidences` does so on a PC.

15.2 Accidental coincidences

Two independent channels with rates R_a and R_b produce coincidences by chance. An event of a finds an event of b within $\pm\tau_w/2$ with probability $R_b\tau_w$ (for $R_b\tau_w \ll 1$), so the accidental rate is

$$R_{\text{acc}} = R_a R_b \, \tau_w. \tag{3}$$

Correlated events (photon pairs, a herald and its photon) add their own rate R_c, and the **coincidence-to-accidental ratio** $\text{CAR} = R_c/R_{\text{acc}}$ measures how clean the correlation is. In a correlation histogram the accidentals are the flat background away from the peak, which can be measured directly: count in a window shifted far from the peak (a *delayed coincidence*), or take the median of the histogram's bins.

Example: how many accidentals?

Two detectors at $R_a = R_b = 200\,\text{kcounts/s}$ and a window of 1 ns: $R_{\text{acc}} = (2 \times 10^5)^2 \times 10^{-9}\,\text{s} = 40$ per second. A pair source giving 4000 true coincidences per second has a CAR of 100. With a 10 ns window (needed with slow detectors) the CAR is 10, and with 100 ns it is 1. A tagger's precision matters because it lets the window shrink to what the detectors allow.

15.3 The timing budget

A coincidence of two detections spreads by the timing jitter of both detectors and both tagger channels, and of the clock that times them (common to channels on one board, so it cancels in a difference; not between two boards, section 17). For Gaussian contributions the variances add:

$$\sigma_{\Delta t}^2 = \sigma_{\text{det},a}^2 + \sigma_{\text{det},b}^2 + \sigma_{\text{tag},a}^2 + \sigma_{\text{tag},b}^2, \qquad \text{FWHM} = 2.355\,\sigma. \tag{4}$$

This tagger's channels contribute about 10 ps rms each (measured, section 5.7). Single-photon detectors range from a few picoseconds to hundreds [3, 4], and silicon avalanche diodes add a slow

tail from carriers that diffuse into the junction. Part II uses three illustrative detector models throughout, *not* the data of any product: SNSPD-like (12 ps rms), thin SPAD-like (25 ps rms, 15 % of the detections with an exponential tail of 120 ps) and thick Si SPAD-like (150 ps rms, 25 % with a 350 ps tail). Figure 20 shows what a pair of each, with two tagger channels, makes of a pair of simultaneous photons. Measure your own detectors: a pair source or a split laser pulse into two channels gives this histogram directly.

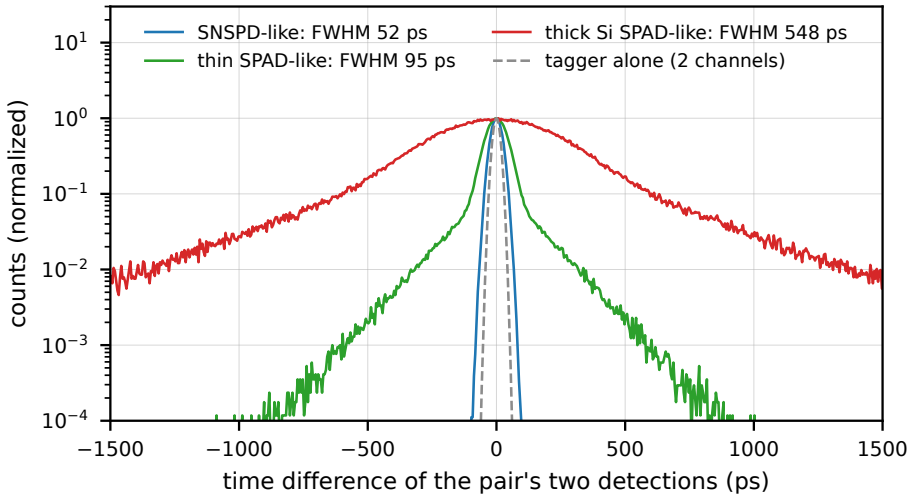


Figure 20: Computed. The time difference of a pair's two photons, as two detectors and two tagger channels see it, for the three illustrative detector models (FWHM 52, 95 and 548 ps), and for the tagger alone (dashed: two channels of 10 ps rms). The silicon detectors' diffusion tails make the histograms asymmetric about zero only in their sum; each detector's own tail is one-sided.

15.4 Dead time and afterpulsing

After a detection, a detector is blind for its *dead time* (tens of nanoseconds for avalanche diodes; the recovery of a superconducting nanowire is similar), and an avalanche diode fires again with some probability shortly after, an *afterpulse*, from carriers trapped during the avalanche [5]. Both distort a detector's correlation with itself: no pairs closer than the dead time, and an excess just after it (figure 21). This is why $g^{(2)}$ is measured with two detectors behind a beam splitter, the arrangement of Hanbury Brown and Twiss [6]: each detector's dead time and afterpulses are uncorrelated with the other's clicks.

The tagger has its own dead time per channel, about 5 ns (the delay line), well below any detector's, and a programmable one (`dead_time`, up to 262 µs) that can extend it. Setting all detectors of an experiment to the same, known dead time in the tagger, longer than each detector's own, is a simple way to make them behave alike, which matters in QKD (section 17.7).

15.5 Connecting detectors

Most single-photon detectors deliver a logic pulse per detection: silicon avalanche-diode modules typically a TTL or LVTTL pulse into 50 Ω, superconducting-nanowire systems a pulse from their own amplifier and comparator, sometimes in NIM levels. The E1 inputs are 3.3 V logic and are **not 5 V tolerant**; a 5 V TTL output must be terminated in 50 Ω at the board, which halves it. NIM pulses are negative and must not reach E1 at all: a NIM-to-TTL converter belongs between. Section 3 has the connections and terminations; for short pulses, a channel's *toggle input* (section 4) puts exactly one transition into the delay line per pulse, however short.

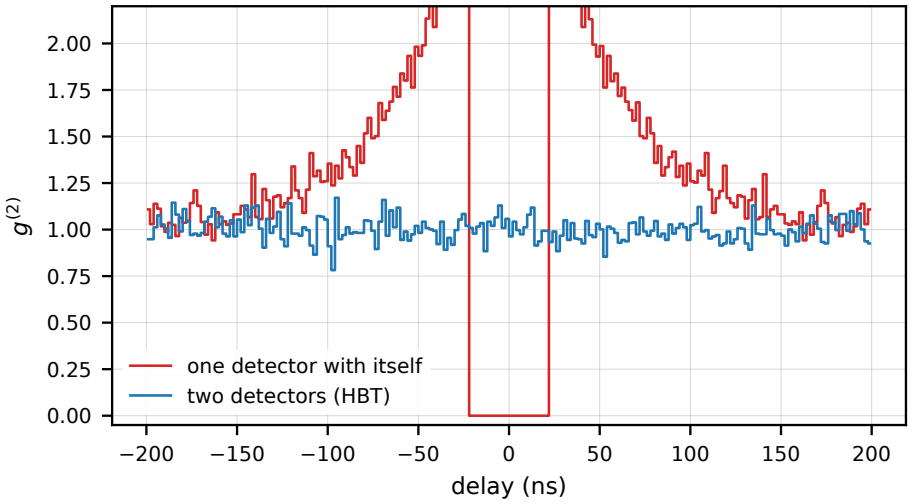


Figure 21: Computed. Coherent light (Poisson photons) on a beam splitter and two detectors with 22 ns of dead time and 4 % afterpulsing (exponential, 40 ns after the dead time): one detector's correlation with itself shows the dead time as a hole and the afterpulses as a bump; the two detectors' correlation is flat at $g^{(2)} = 1$, as it should be for coherent light.

16 Quantum optics with a time tagger

The measurements in this section characterize light at the level of single photons: whether a source emits one photon at a time, how long an emitter takes to emit, whether two photons are indistinguishable, how clean a source of photon pairs is. Each is a histogram of section 15.1 on the tags of two or three detectors. The figures are computed (there are no photon detectors on the bench used for this manual), with this tagger's measured timing and the illustrative detector models of section 15.3; the code is the code one runs on recorded tags.

16.1 Photon statistics: Hanbury Brown and Twiss

Light that reaches a 50:50 beam splitter is detected at its two outputs; the correlation of the two detectors' tags is $g^{(2)}(\tau)$, the probability of detecting a photon at τ after another, relative to independent photons (figure 22). Thermal light is bunched ($g^{(2)}(0) = 2$), a laser is not ($g^{(2)} = 1$), and a single quantum emitter, which has to be re-excited after each photon, cannot emit two at once: $g^{(2)}(0) = 0$, *antibunching*, first seen in resonance fluorescence [7]. The measurement goes back to Hanbury Brown and Twiss [6].

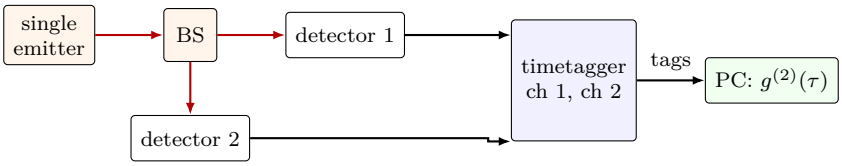


Figure 22: The Hanbury Brown–Twiss arrangement: two detectors behind a beam splitter, their tags correlated on the PC. Two detectors are needed because each one's dead time and afterpulses are uncorrelated with the other's clicks (figure 21).

For a two-level emitter excited incoherently at rate R that decays at rate Γ,

$$g^{(2)}(\tau) = 1 - \mathrm{e}^{-|\tau|/\tau_c}, \qquad \tau_c = \frac{1}{R+\Gamma}. \tag{5}$$

The measured curve is this convolved with the pair's timing (equation (4)): when $\sigma_{\Delta t}$ approaches τ_c the dip fills in. Figure 23 shows an emitter with $\tau_c = 0.8$ ns: with SNSPD-like detectors the measured

$g^{(2)}(0)$ is 0.03, with the slow silicon detectors 0.21, both well below the usual single-emitter criterion $g^{(2)}(0) < 0.5$, but the second one only because τ_c is long. For a fast emitter (a short lifetime, or strong pumping) the detectors' timing decides whether antibunching can be seen at all; this tagger's channels add only 10 ps rms each.

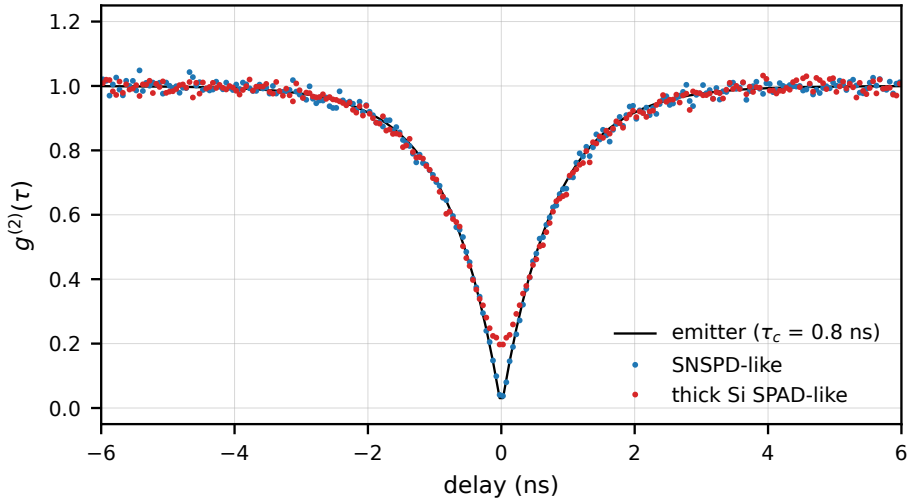


Figure 23: Computed. Antibunching of a single emitter ($\tau_c = 0.8$ ns, black) as measured with SNSPD-like and thick-silicon-SPAD-like detectors and two tagger channels: 30 minutes at 200 kcounts/s per detector, bins of 50 ps, Poisson noise. The ideal curve convolved with each pair's timing distribution.

On a PC the correlation is computed as the tags arrive. With ttcore (the C++ library, at the stream's full rate; section 13) and the client `tt` of the board, a live $g^{(2)}$ is a few lines:

```
import time, ttcore
g2 = ttcore.Correlation(1, 2, 50, 400)          # 50 ps bins, +-10 ns
stream = ttcore.StreamClient("192.168.0.102")   # receives, sorts and measures in a thread
stream.attach(g2)
stream.start()
tt.start(duration=60.0)                          # a 60 s acquisition
time.sleep(62.0)                                 # (g2.g2() can be read at any time)
stream.stop()
print(g2.g2()[200])                              # g2 at zero delay
```

and offline from recorded tags (channels `ch`, times `t` in ps) with the numpy reference functions:

```
a, b = t[ch == 1], t[ch == 2]
counts = analysis.correlation(a, b, 50, 400)
g2 = counts / analysis.correlation_norm(len(a), len(b), t[-1] - t[0], 50)
```

16.2 Fluorescence lifetimes: TCSPC with the conditional filter

A pulsed laser excites a sample at a repetition rate of tens of MHz; each detected fluorescence photon's time since the last laser pulse, histogrammed, is the decay of the excited state convolved with the instrument response (the laser pulse, the detector and the tagger) [1, 2]. The tagger times the laser's sync output on one channel and the detector on another. The difficulty is the sync's rate: 80 million per second, more than the tagger takes from one channel (62.5 Mevents/s) and far more than the stream's 7 Mevents/s. A divider would keep every N-th sync and leave the others to be interpolated on the PC, which works for a stable laser. The **conditional filter** (section 10) does better: on channels 1 and 2 it keeps only the syncs that a detection follows within a window of up to 120 ns, decided in the FPGA, so the syncs streamed are about as many as the photons (figure 24).

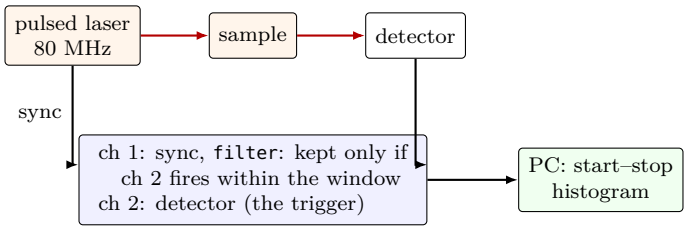


Figure 24: TCSPC with the conditional filter: the laser's sync on channel 1, filtered by the detector on channel 2.

```
tt.apply({"channels": {"1": {"enable": True, "filter": True},      # the laser's sync
                       "2": {"enable": True}},                       # the detector
          "filter": {"triggers": [2], "window": 16e-9}})
```

Figure 25 shows the result for a lifetime of 2.5 ns: the start–stop histogram from the kept syncs is the same as from all of them (the filter keeps the sync before each photon), the fitted lifetime is 2.53 ns, and the stream carries 352 k syncs per second instead of 80 M for 201 k photons per second. The window must be longer than the laser's period, so that the sync before every photon is kept; a little longer keeps a second, earlier sync as well, which helps when the decay is longer than the period. As in any TCSPC measurement, keep the detection rate to a few per cent of the laser's rate, or the detector's dead time distorts the decay (pile-up) [2].

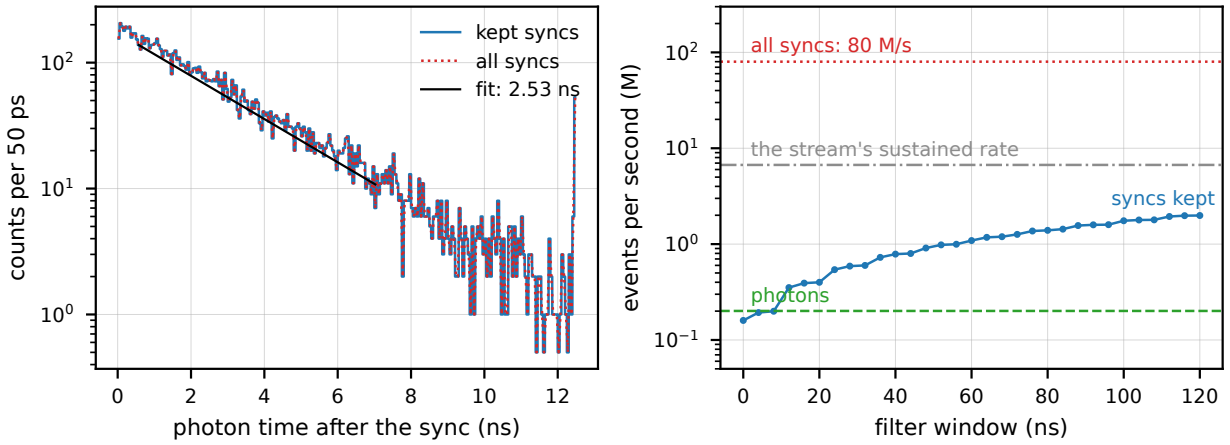


Figure 25: Computed. Left: the start–stop histogram of a 2.5 ns decay excited at 80 MHz, from the syncs the filter keeps (16 ns window) and from all syncs: they coincide; the fit gives 2.53 ns. Right: the syncs streamed against the filter's window.

16.3 Two-photon interference: the Hong–Ou–Mandel dip

Two photons that arrive at a 50:50 beam splitter at the same time, one in each input, leave together through the same output if they are indistinguishable: the coincidences of two detectors at the outputs vanish [8]. Scanning the delay between the photons traces a dip whose width is the photons' coherence time,

$$C(\delta) \propto 1 - V\,\mathrm{e}^{-(\delta/\tau_c)^2} \tag{6}$$

for Gaussian spectra, with the visibility V measuring how indistinguishable the photons are (figure 26). The coherence time of photons with a bandwidth of nanometres is picoseconds (2.2 ps for 1 nm at 810 nm): far below any detector's timing. The dip is scanned with a translation stage, and the tagger only counts coincidences in a window comfortably wider than the detectors' jitter, at each position. The FPGA's coincidence engine counts them live, with nothing to stream:

```
tt.apply({"coincidence": {"enable": True, "channels": [1, 2], "window": 2e-9,
                          "masks": {"1": {"channels": [1, 2]}}}})
tt.clear_coincidences()
tt.start(duration=1.0)                    # one stage position
```

and `tt.coincidences()["masks"]["1"]` reads the coincidences after the second.

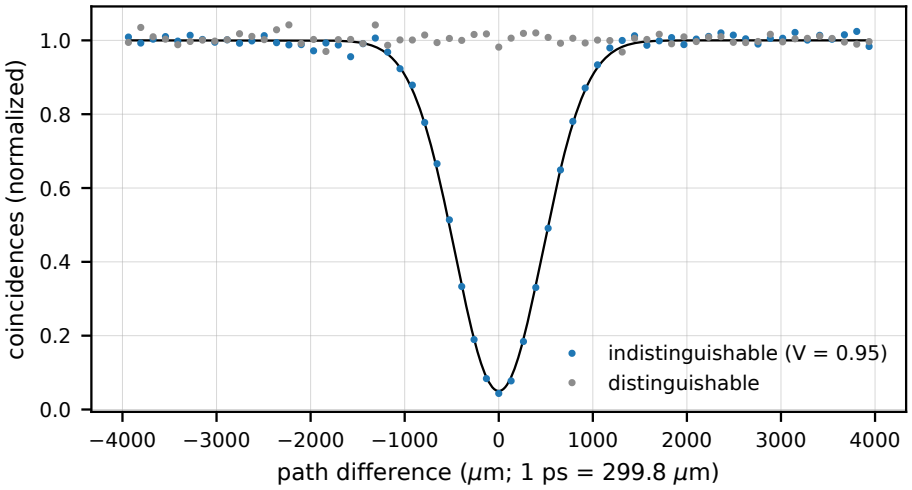


Figure 26: Computed. The Hong–Ou–Mandel dip of two photons of 1 nm bandwidth at 810 nm (coherence time 2.2 ps, visibility 0.95) against the path difference a translation stage sets, with Poisson noise for 4000 coincidences per point; distinguishable photons give no dip.

Two photons from *independent* sources interfere only if they also arrive within their coherence time; then the tagger's precision matters in another way: it measures and steers the arrival times. This is the situation of measurement-device-independent QKD (section 18).

16.4 Heralded single photons and photon pairs

Spontaneous parametric down-conversion makes photon pairs at random times; detecting one photon of a pair (the herald) announces its partner [9]. With the herald on channel 1 and the heralded photon on channel 2, the pairs appear as a peak in the correlation on the flat background of accidentals (section 15.2). For a pair rate P and detection efficiencies η_1, η_2, the coincidences are $\eta_1\eta_2 P$ and the accidentals $(\eta_1 P)(\eta_2 P)\,\tau_w$, so that at high rates

$$\mathrm{CAR} \approx \frac{1}{P\,\tau_w}\,: \tag{7}$$

a brighter source is a noisier one, and a narrower window, made possible by better timing, buys back the difference (figure 27). At low rates the dark counts set the CAR. The heralding efficiency, $\eta_2 = C/N_1$ (coincidences per herald), is the efficiency of the heralded arm.

The engine makes several of these measurements without streaming the photons:

- **Heralded photons as a channel of their own.** A mask on channels 1 and 2 marked `virtual` puts a record (channel 21) into the stream at each coincidence; with the two detectors set to `record`: False, only the heralded events are streamed, at the rate of the coincidences.
- **Heralded $g^{(2)}$.** With the herald on channel 1 and the heralded photon split onto detectors 2 and 3, the heralded second-order correlation at zero delay is

  $$g_h^{(2)}(0) = \frac{N_1 N_{123}}{N_{12} N_{13}}, \tag{8}$$

  where the N count heralds, double and triple coincidences. The engine counts every pattern of channels 1–3 in one window, at full rate; the inclusive masks {1,2}, {1,3} and {1,2,3} give the three coincidence counts directly.

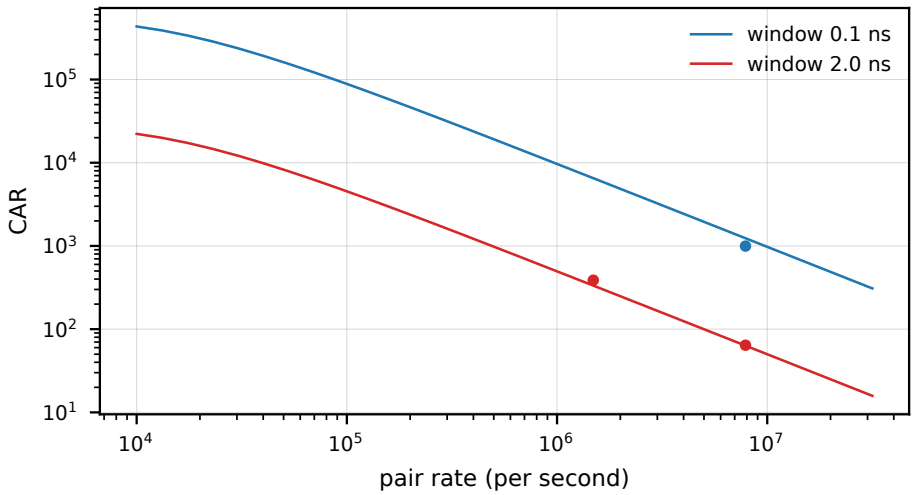


Figure 27: Computed. The coincidence-to-accidental ratio of a pair source against its pair rate, for detection efficiencies of 10 % and 500 dark counts per second, with coincidence windows of 0.1 and 2.0 ns: lines from equations (3) and (7), points from simulated tags.

```
tt.apply({"coincidence": {"enable": True, "channels": [1, 2, 3], "window": 1e-9,
                          "masks": {"1": {"channels": [1, 2]}, "2": {"channels": [1, 3]},
                                    "3": {"channels": [1, 2, 3]}}}})
```

Polarization-entangled pairs [10] measured with polarizers before the two detectors give the coincidences of a Bell test; the CHSH combination of four settings is bounded by 2 for local hidden-variable models and reaches $2\sqrt{2}$ for a maximally entangled state [11]. The same coincidences, between two distant stations, are the raw data of entanglement-based QKD, the subject of the next section.

17 Entanglement-based quantum key distribution

Quantum key distribution lets two parties, Alice and Bob, grow a shared secret key whose secrecy rests on quantum physics rather than on computational assumptions [12–15]. In the entanglement-based protocols [16, 17] a source between them, or at one of them, sends the two photons of entangled pairs to the two stations; each station measures its photon in a basis it chooses at random, and when the two bases agree the outcomes are correlated: they are the raw key. Such systems have run over fibre networks [18, 19], over free-space links [20] and from a satellite [21, 22].

Time tagging is at the centre of these systems. The two stations' detections belong together only pair by pair, and the only thing that tells which detection at Alice belongs to which at Bob is *when* they happened. The tags decide which coincidences are found at all (clock synchronization), how many accidental coincidences are taken with the true ones (the window, set by the timing), and they are the place where some attacks on the detectors show up. This section goes through each, with simulated data analysed by the design's functions.

17.1 The protocol and what the tags carry

In BBM92 [17] the source emits polarization-entangled pairs in the state $(|HH\rangle + |VV\rangle)/\sqrt{2}$. At each station a 50:50 beam splitter chooses the basis passively: in one arm a polarizing beam splitter separates H and V, in the other a half-wave plate and a polarizing beam splitter separate the diagonal polarizations D and A. Four detectors per station; each detection is a tag on the channel of its detector, and the channel says the basis and the bit (figure 28).

A run then goes as follows.

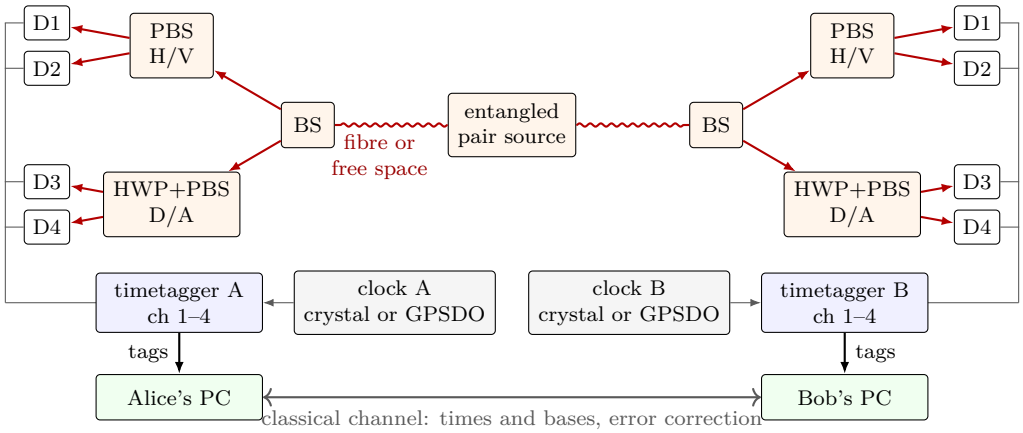


Figure 28: Entanglement-based QKD (BBM92) with a time tagger at each station. A beam splitter chooses the basis; four detectors give the basis and the bit; each station's tagger times its detections with its own clock. The tags of the two stations are matched over a classical channel.

1. Both stations record their tags: time, and the detector (basis and bit).
2. Bob sends Alice the times and bases of his detections, not the bits. Alice finds the coincidences: her detections within a window τ_w of one of Bob's, once Bob's times are in her time scale (section 17.2). She tells Bob which coincidences had the same basis; both keep those bits, the *sifted key*.
3. A random sample of the sifted bits estimates the quantum bit error rate (QBER); error correction and privacy amplification turn the rest into the secret key, shorter by the information an eavesdropper could have, which the QBER bounds [14].

A detection takes a few bytes on the classical channel; at the rates of section 17.3 (hundreds of kilocounts per second per station) that is a megabit per second or so. The tagger delivers the tags to the station's PC through its stream (section 7) or records them (`.ttbin`).

17.2 Synchronizing two clocks with the photons

Each station's tags are in its own clock's time scale. A detection at true time t has the tags

$$t_A = t + \epsilon_A, \qquad t_B = t + \delta + y\,t + x(t) + \epsilon_B, \tag{9}$$

where δ is the offset between the two acquisitions' zeros and the path difference from the source, y the clocks' relative frequency offset, $x(t)$ their relative wander, and ϵ the detections' timing (detector and tagger). To find coincidences, δ, y and x must be known to better than the window. There are two ways, and they combine well.

A common reference. Each tagger is referred to a GNSS-disciplined oscillator: its 10 MHz on IN2 relates the tags to the reference's time scale (the phase meter, section 11), and an acquisition armed on the reference's PPS starts both stations at the same second. Two GNSS-disciplined clocks agree to tens of nanoseconds in time and to 10^{-12} or better in frequency over seconds (measured with this tagger in section 19); what remains of δ is the path delay and the clocks' residual offset, found as below with a small search span.

The photons themselves. Without a reference, the pairs alone carry the information [20, 23, 24]: in the cross-correlation of the two stations' tags, the pairs make a peak at δ on the flat background of accidentals. Searching a span of milliseconds directly would take many passes; instead both tag streams are binned into bins of width Δ (say 1 ns), folded modulo a power of two of bins L, and

correlated by FFT:

$$C(k) = \sum_n h_A(n)\, h_B(n+k) = \mathcal{F}^{-1}\Big[\,\overline{\mathcal{F}[h_A]} \cdot \mathcal{F}[h_B]\,\Big](k). \tag{10}$$

Folding keeps a pair at the same lag (modulo L) and only adds background. The peak holds $N_c = R_c T$ pairs against $B = R_A R_B T \Delta$ per bin (or $N_A N_B / L$ when folded), and stands out when $N_c/\sqrt{B} \gtrsim 10$. A fine histogram of the differences around the peak, in picosecond bins, then places it to a fraction of the pairs' timing. This is `analysis.find_offset`:

```
d, pairs, background = analysis.find_offset(t_a, t_b, span_ps=2e9, bin_ps=1000)
```

Figure 29 shows it on half a second of simulated tags, 70 k detections per second at each station: the offset of 0.73 ms is found to 0.9 ps from 1246 pairs, with the coarse peak 62 times the noise above the background.

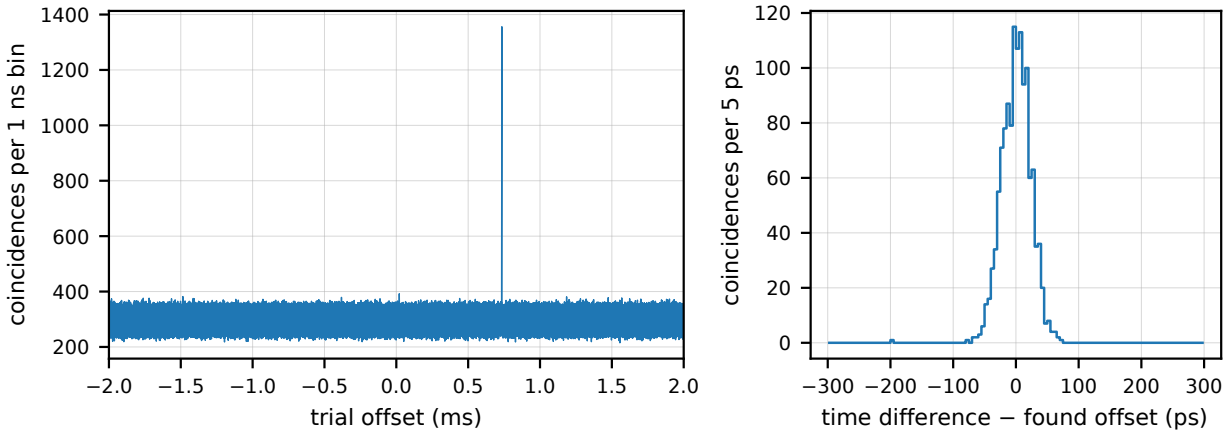


Figure 29: Computed. Left: the FFT cross-correlation of two stations' tags over ±2 ms in 1 ns bins (0.5 s of data, folded): the pairs' peak at the unknown offset. Right: the fine histogram around it, 5 ps bins: FWHM 55 ps, the pairs' timing ($\sigma = 22$ ps: SNSPD-like detectors and the tagger at both ends).

The frequency offset. Over the integration time T the offset moves by yT. Free crystals differ by parts per million: $y = 2 \times 10^{-6}$ moves it by 2 µs in a second, two thousand 1 ns bins, and the peak is smeared away. The search is then over y as well: Bob's tags are rescaled by $1/(1+y_k)$ for a grid of trial values, with a step of about $\Delta/2T$, from coarse bins to fine ones (100, 10 and 1 ns), and the trial with the highest peak wins:

```
def peak(t_a, t_b, y, bin_ps):
    return analysis.find_offset(t_a, np.round(t_b / (1 + y)).astype(np.int64), 2e9, bin_ps)[1]
ys = np.arange(-2e-5, 2e-5, 2.5e-7)                  # +-20 ppm in 100 ns bins (T = 0.2 s)
y = ys[np.argmax([peak(t_a, t_b, yk, 100000) for yk in ys])]
# ... then +-5e-7 around it in 10 ns bins, +-5e-8 in 1 ns bins
```

The left of figure 30 shows the three passes on 0.2 s of tags; they find $y = 2.3675$ ppm (true: 2.370).

Tracking. With y known approximately, the offset is followed in consecutive windows short enough that the remaining drift within one stays below a coarse bin, each searched around the value predicted from the last two. `analysis.track_offset` does this; the offsets' slope refines y (here to 0.01 ppb), and what remains after a line is the clocks' relative wander, which the coincidence window has to accommodate or the tracking has to follow:

```
c, offsets, pairs = analysis.track_offset(t_a, t_b, window_ps=1e11, span_ps=2e9, bin_ps=1000)
ok = np.isfinite(offsets)
t_b_in_a = t_b - np.round(np.interp(t_b, c[ok] + offsets[ok], offsets[ok])).astype(np.int64)
```

The right of figure 30 compares two free crystals (white frequency noise of 3×10^{-10} at 1 s: 181 ps rms of wander over 10 s) with two taggers on GNSS-disciplined oscillators (4.3 ps, the tracking's own noise). With free crystals the offset must be tracked continuously; with a common reference a single offset holds for long stretches.

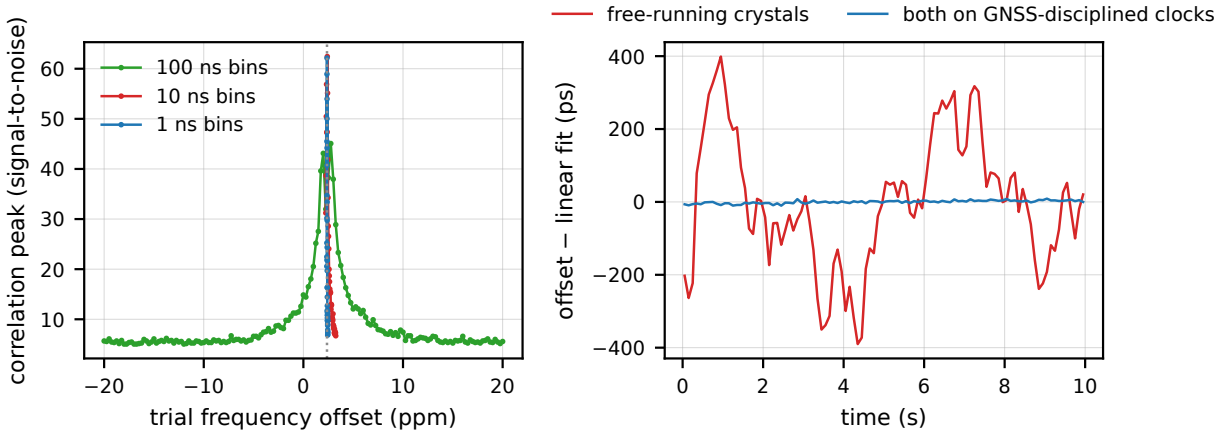


Figure 30: Computed. Left: the search for the clocks' frequency offset on 0.2 s of tags, in three passes (100, 10 and 1 ns bins; dotted: the true value). Right: the offset tracked in 0.1 s windows over 10 s, less a straight line, for two free-running crystals and for two taggers referred to GNSS-disciplined oscillators.

17.3 The coincidence window, the QBER and the key rate

A wider window takes more of the true coincidences and more accidentals. With the pairs' timing $\sigma_{\Delta t}$ (equation (4)), a fraction $\operatorname{erf}(\tau_w/(2\sqrt{2}\,\sigma_{\Delta t}))$ of the C true coincidences falls within $\pm\tau_w/2$, and the $A = R_A R_B \tau_w$ accidentals carry random bits, wrong half the time. The source and the analysers make an intrinsic error e of their own, so

$$\mathrm{QBER}(\tau_w) = \frac{e\,C(\tau_w) + \frac{1}{2}A(\tau_w)}{C(\tau_w) + A(\tau_w)}. \tag{11}$$

Half of the coincidences have matching bases (a passive 50:50 choice), and in the asymptotic limit a fraction $1 - 2h(\mathrm{QBER})$ of the sifted bits survives error correction and privacy amplification, h being the binary entropy (zero above QBER $\approx 11\,\%$) [14]:

$$R_{\mathrm{key}}(\tau_w) = \tfrac{1}{2}\left(C(\tau_w) + A(\tau_w)\right)\left(1 - 2h(\mathrm{QBER})\right). \tag{12}$$

Figure 31 shows both against the window for a bright source (20 M pairs per second, 2 % detection at each station after the links, $e = 1\,\%$). The best window is about twice the pairs' rms timing: 137 ps with SNSPD-like detectors, 228 ps with thin SPADs and 992 ps with thick silicon SPADs, where the QBER is 1.1, 1.2 and 2.0 %. The points are the QBER of simulated tags of two stations sifted with that window (`qsim.sift`): the formula holds.

Three practical points follow. The window is centred on each pair of detectors' own delay, which differs by cable lengths and detector latencies: measure it from the cross-correlation of each detector pair and correct it in the tagger's channel offsets (`channels.n.offset`) or on the PC. The timing that matters is that of the *pair* of stations, including their clocks' residual wander after tracking. And finite-key effects, which a real system must include, reduce the rate further, most for short blocks [14].

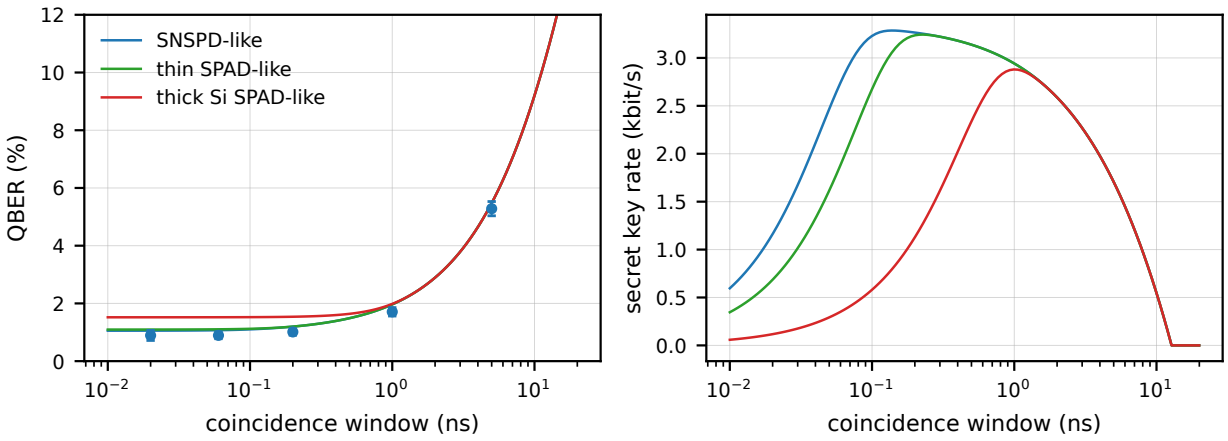


Figure 31: Computed. The QBER (left) and the asymptotic secret key rate (right) against the coincidence window, for the three detector models with the tagger at both stations. Points: 2 s of simulated tags of two stations, sifted (SNSPD-like detectors; error bars: counting statistics).

17.4 Over distance: what the timing buys

With the source in the middle of a fibre link, each photon crosses half of it (0.2 dB/km at 1550 nm). The true coincidences fall with the product of the two transmissions, the accidentals from multiple pairs with it too, but those from dark counts do not: at long distance the dark counts in the window decide where the key ends, and the window is set by the timing. Figure 32 optimizes the pair rate (up to 10^9 pairs per second) and the window at each length of fibre in the model of equations (11) and (12), with each station's four detectors, their dead times, and the stream's limit: a station's tags may not exceed 7 Mevents/s, which caps the key at short distance. Three cases:

- SNSPD-like detectors (85 %, 12 ps rms, 100 dark counts per second each, 50 ns dead time) with this tagger: 208 kbit/s at 100 km (window 119 ps), 1 bit/s up to 410 km;
- the same detectors timed in 1 ns bins (289 ps rms, as a counter clocked at 1 GHz would give): 82.1 kbit/s at 100 km (window 1221 ps), 1 bit/s up to 348 km;
- InGaAs-SPAD-like detectors (20 %, 80 ps rms, 1000 dark counts per second each, 10 µs dead time against afterpulses) with this tagger: 0.769 kbit/s at 100 km, 1 bit/s up to 304 km. Their dead time, not the timing, limits them at short distance.

These are upper bounds in the spirit of the analysis of Ma, Fung and Lo [25], and they leave out what a real link adds: finite-key corrections, error correction above the Shannon limit, Raman noise from classical channels in the same fibre and, above all, **chromatic dispersion**. A photon of spectral width $\Delta\lambda$ spreads by $D\,\Delta\lambda\,L$ in a fibre of dispersion $D \approx 17\,\mathrm{ps/(nm\,km)}$: a 1 nm wide photon by 17 ps per kilometre, more than the tagger's and an SNSPD's timing together after one kilometre. The timing advantage survives only with narrow-band pairs, dispersion-compensating fibre or modules, or the nonlocal cancellation of dispersion between energy-time entangled photons (opposite dispersion in the two arms) [26].

17.5 Visibility and the Bell parameter

The coincidence counts of all detector combinations measure the entanglement directly. For analysers at angles a and b, the correlation is $E(a,b) = (C_{++} + C_{--} - C_{+-} - C_{-+})/(C_{++} + C_{--} + C_{+-} + C_{-+})$ from the four coincidence counts of the two outputs at each side, and the CHSH parameter

$$S = \left|E(a,b) - E(a,b') + E(a',b) + E(a',b')\right| \tag{13}$$

is at most 2 for local hidden-variable models and $2\sqrt{2}$ for a maximally entangled state [11]. For a state with visibility V (entangled light mixed with white noise) $S = 2\sqrt{2}\,V$ and the QBER in each

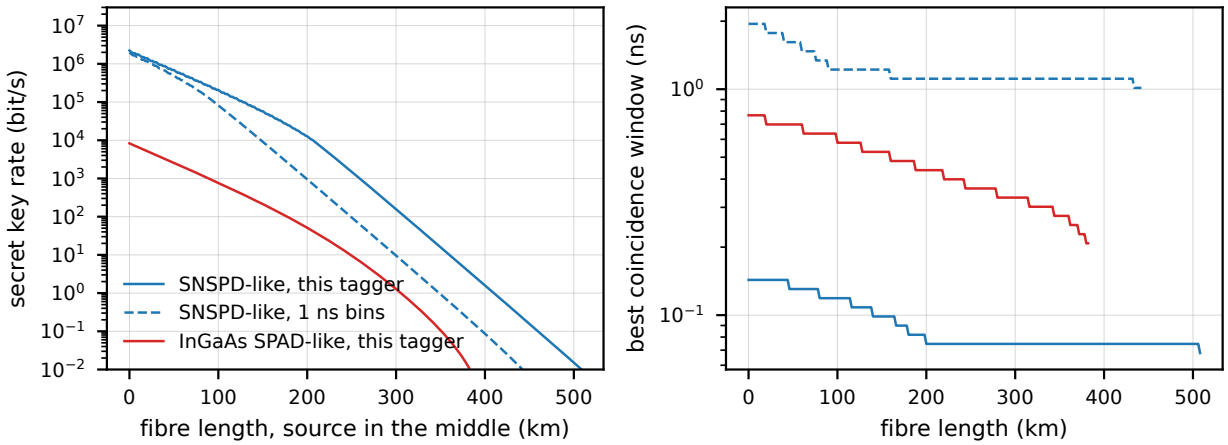


Figure 32: Computed. BBM92 over fibre with the source in the middle: the asymptotic secret key rate (left) and the coincidence window that gives it (right), the pair rate and the window optimized at each length (at most 10^9 pairs/s, each station's tags within the stream's 7 Mevents/s, $e = 1\,\%$, dispersion compensated, no noise from the fibre). Illustrative detector models, not a product's data.

basis is $(1 - V)/2$. Ekert's protocol uses S as the check of the source and the measurements [16]; BBM92 with a monitored visibility is equivalent in practice. With a time tagger these numbers come from the same tags as the key, so they can be followed throughout a run.

17.6 Monitoring a station: the engine, the markers and the filter

Some of what a station must watch involves only its own detectors and can be counted in its FPGA, at any rate, while the key's tags stream:

- **Double clicks.** Two of a station's detectors firing within a window come from multi-photon events, dark counts or an attack with bright light. Security proofs assign them a random bit [14]; their rate is a sensitive monitor. The coincidence engine counts every pattern of the four detectors' channels; masks on the pairs {1,2} (two outcomes of one basis) give live counters.
- **Markers** time slow signals to 1 ns beside the detections: the state of an active basis switch, a polarization-compensation cycle, a frame or block boundary. A marker can also *gate* the engine so that it counts only during the frames that carry key.
- **Engine-only channels** let the engine count at rates the stream cannot carry, for instance while the source is turned up to align the links.

```
tt.apply({"coincidence": {"enable": True, "channels": [1, 2, 3, 4], "window": 1e-9,
                          "masks": {"1": {"channels": [1, 2]}, "2": {"channels": [3, 4]}},
                          "gate": {"marker": 1}}})              # count during key frames only
```

17.7 Timing side channels

The security proofs assume detectors that cannot be told apart. Real detectors differ in timing, and timing differences can leak the key. If two detectors' efficiencies depend on a photon's arrival time differently, an eavesdropper who can shift the arrival times biases which of them clicks (figure 33): the efficiency mismatch [27] and the time-shift attack [28, 29]. Dead time is another channel: after a click a detector is blind, and which detector was blind is information.

A time tagger is the instrument that finds and limits these effects:

- characterize each detector's timing response (delay, jitter, tails) with a pair source or a split pulse, and its efficiency against arrival time by scanning a delay;

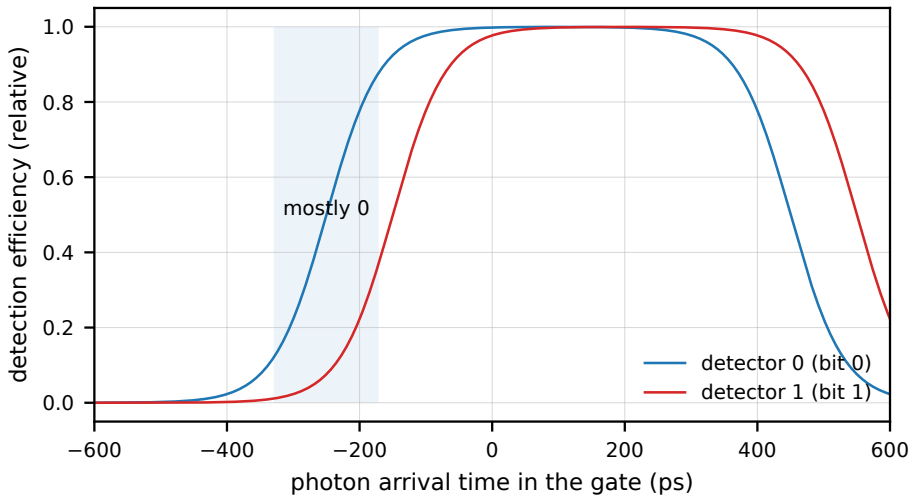


Figure 33: Computed. Illustration: two detectors whose efficiencies rise at arrival times 100 ps apart. Photons shifted into the shaded interval are detected mostly by detector 0: the eavesdropper who shifts them learns the bit.

- equalize the detectors' delays in the tagger (channel offsets) and use windows centred per detector, so that no detector sees a different part of the arrival-time distribution;
- give all detectors of a station the same dead time, longer than each one's own (`dead_time` in the tagger), and discard on the PC the detections of any detector that fall within the common dead time of another;
- watch the arrival-time histogram of each detector during operation: a shift of one detector's distribution, or of all photons against the window, is the signature of a time-shift attack.

A two-station recipe

1. At each station: the four detectors on channels 1–4 (rising edges, toggle inputs), `tt.calibrate()` once, the delays between detectors measured and set as channel offsets.
2. Optionally, each tagger's IN2 on a GNSS-disciplined 10 MHz and the PPS on channel 7 (section 3); start both acquisitions armed on the PPS (`tt.start(arm_channel=7)`), so they share a second.
3. Record the tags (`ttcore.StreamClient.record`), or process them as they come.
4. Find the offset and frequency once (`find_offset`, with the frequency search if the clocks are free), then track it (`track_offset`); map Bob's tags into Alice's time scale.
5. Sift with a window of about twice the pairs' rms timing; estimate the QBER; correct the errors and amplify privacy.

18 Prepare-and-measure QKD and quantum time transfer

In prepare-and-measure QKD Alice prepares single-photon-level pulses herself, at a high and regular rate, and Bob measures them. Timing enters differently than with pairs: Bob must know which of Alice's pulses each detection belongs to, which is a question of Alice's clock rather than of a coincidence. This section treats the time slots and gates of BB84-type systems, the recovery of Alice's clock from the detections themselves, the timing needs of two-photon protocols (MDI and twin-field QKD), and closes with the use of photon pairs to transfer time itself.

18.1 Time slots and gates

In BB84 [12] with decoy states [30–32] Alice sends pulses at a rate f of 100 MHz to a few GHz, each in one of four states (polarizations, or time bins with a phase) and one of several intensities. Bob's detection belongs to slot

$$n = \mathrm{round}((t - t_0)/T_{\mathrm{slot}}), \tag{14}$$

and only the part of the slot where Alice's photon can arrive carries signal: dark counts and afterpulses fall anywhere. Bob therefore needs Alice's clock, from a sync signal (a clock on another wavelength or fibre, divided before the tagger by the channel's `divider` and interpolated on the PC), from a common reference (GNSS-disciplined oscillators at both ends, section 19), or from the detections themselves (below).

Folding Bob's detection times into the slot shows the signal on a background (figure 34); a *gate*, accepting only detections near the expected arrival, cuts the background in proportion to its width. With 2 ns slots and thin-SPAD-like detectors, a 200 ps gate gives a QBER of 1.2 %, the whole slot 2.5 %: the dark counts and the afterpulses of the earlier detections are spread over the slot. The detector's timing tail decides how narrow the gate can be before it costs signal.

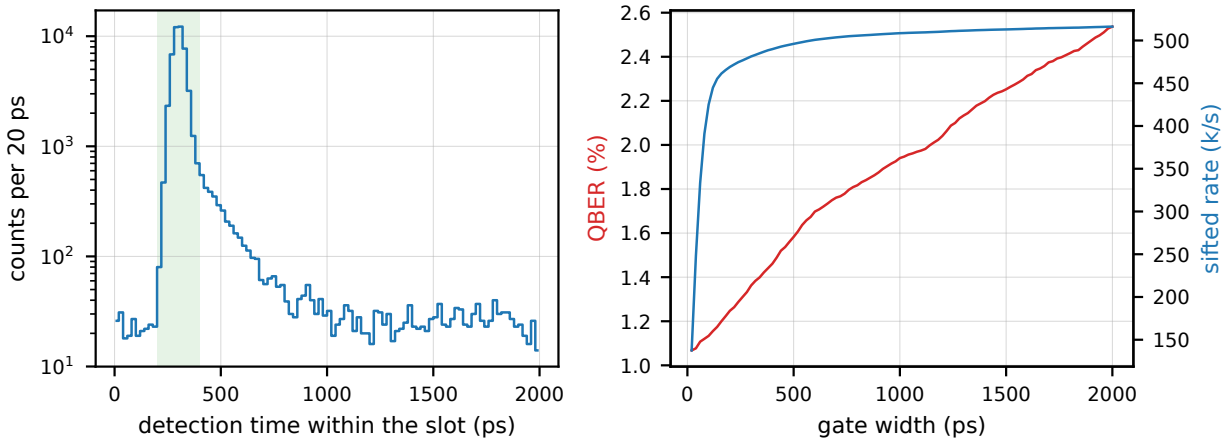


Figure 34: Computed. Left: Bob's detections folded into Alice's 2 ns slots: the signal (thin-SPAD-like detector and the tagger), dark counts and afterpulses; shaded, a 200 ps gate. Right: the QBER (1 % optical error, random bits for the background) and the sifted rate against the gate's width.

Time-bin encodings, and the coherent-one-way protocol [33], put the bit into an early or a late bin within the slot, typically a nanosecond or less apart: the tagger must separate the two bins cleanly, which needs a timing of a small fraction of their spacing.

18.2 Clock recovery without a sync

Alice's pulses are periodic, so Bob's detections carry her clock even though only a small fraction of the pulses is detected. The periodogram of the detection times t_k,

$$P(T) = \frac{1}{N}\left|\sum_k \mathrm{e}^{2\pi \mathrm{i} t_k/T}\right|^2, \tag{15}$$

peaks at Alice's period as Bob's clock measures it, with a width of about T^2/t_{span}: a tenth of a second of detections fixes the period to 10^{-7} and better. Folding the detections at that period shows the slot (figure 35); the slot numbering (which pulse is which) comes from a known pattern that Alice sends at the start, after which the period is followed as above. This is, in essence, the qubit-based synchronization of Calderaro et al. [34], which needs no sync channel at all.

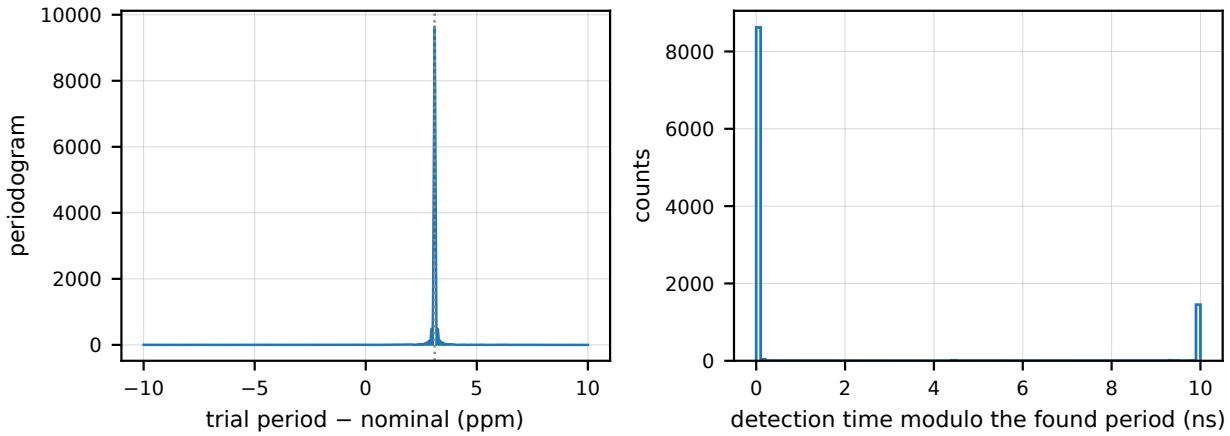


Figure 35: Computed. Bob's detections of Alice's 100 MHz pulses (106 k per second for 0.1 s), his clock 3.1 ppm fast. Left: the periodogram over trial periods finds the offset (3.100 ppm). Right: the detections folded at the found period.

18.3 Two-photon protocols: MDI and twin-field QKD

In measurement-device-independent QKD [35] Alice and Bob both send their pulses to a third station, which interferes them on a beam splitter and announces which detectors clicked; the scheme removes all attacks on the detectors. The two photons interfere only if they are indistinguishable, which includes arriving within their coherence time: two independent lasers must be aligned in time to a fraction of their pulse length, picoseconds to tens of picoseconds (section 16.3). A tagger at the measuring station times both senders' pulses against its clock, and the measured offset steers their delays. Twin-field QKD [36] interferes single photons from the two senders and needs the same timing alignment, besides a stabilized optical phase.

18.4 Quantum two-way time transfer

The correlations that synchronize the clocks of a QKD link can be the purpose of the link: transferring time between two stations with photon pairs. Proposals of quantum-enhanced clock synchronization [37–39] and early demonstrations with pairs [40] led to a practical scheme, the symmetric two-way transfer [41, 42]: each station has a pair source, keeps one photon of each pair (tagged locally) and sends the other to the far station (tagged there) (figure 36).

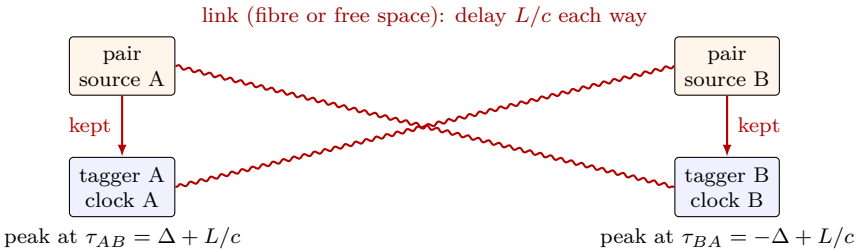


Figure 36: Two-way time transfer with photon pairs: each station tags its own photons and the photons from the other station; the two cross-correlation peaks give the clocks' offset Δ and the link's delay.

The cross-correlation of A's kept photons with their partners tagged at B peaks at $\tau_{AB} = \Delta + L/c$, the clocks' offset plus the link's delay, and the other direction at $\tau_{BA} = -\Delta + L/c$. For a reciprocal link the delay cancels:

$$\Delta = \tfrac{1}{2}\,(\tau_{AB} - \tau_{BA}), \qquad L/c = \tfrac{1}{2}\,(\tau_{AB} + \tau_{BA}). \tag{16}$$

Each peak is found as in section 17.2 (`analysis.find_offset`), and each measurement of Δ is precise to about $\sigma_{\Delta t}/\sqrt{2N_c}$ for N_c pairs per direction. Demonstrations reached picosecond stability

over a 103 km urban fibre with 38 dB of loss [43] and over a 250 km link [44]; a single pair source can also serve both directions [45], and correlated photons alone can synchronize free crystals [24]. The comparison of such a quantum time transfer with a classical one (a GNSS-disciplined PPS at each end) uses the clock analysis of the next section.

19 Clock comparison and synchronization

Every time tagger is also a clock comparator: it times the edges of one clock with another, its own. With a reference, it compares any clock with that reference to picoseconds, and the comparison of clocks, from a crystal to a GNSS-disciplined oscillator, is how one knows what a time scale is worth. This section introduces the quantities of time and frequency metrology [46–49], shows how the tagger measures them, and applies them to a GNSS-disciplined oscillator, the tagger's own crystal and a function generator, all measured on the bench used for this manual.

19.1 Time error, frequency and their statistics

A clock's **time error** $x(t)$ is the difference between its time and a reference's; its fractional **frequency** offset is $y = \mathrm{d}x/\mathrm{d}t$. A real clock is described by

$$x(t) = x_0 + y_0\, t + \tfrac{1}{2} D\, t^2 + \varepsilon(t), \tag{17}$$

an offset, a frequency offset, a drift and noise. The noise is characterized by how the frequency fluctuates when averaged over a time τ. The **Allan deviation** [46]

$$\sigma_y^2(\tau) = \tfrac{1}{2}\left\langle (\bar{y}_{k+1} - \bar{y}_k)^2 \right\rangle = \frac{1}{2\tau^2}\left\langle (x_{k+2} - 2x_{k+1} + x_k)^2 \right\rangle, \tag{18}$$

with x_k sampled every τ, falls as τ^{-1} for white phase noise (a timing jitter), as $\tau^{-1/2}$ for white frequency noise, is flat for flicker frequency noise and rises for random walk and drift. The **modified Allan deviation** (MDEV) averages the phase over each τ before differencing and separates white from flicker phase noise [50]; the **time deviation** $\sigma_x(\tau) = \tau\,\mathrm{MDEV}(\tau)/\sqrt{3}$ (TDEV) is the time stability of a clock in seconds, the quantity telecommunication standards specify; and the **maximum time interval error** (MTIE) is the largest peak-to-peak time error within any window τ. ITU-T Recommendations G.810, G.811 and G.8262 [51–53] define them and the masks a clock must meet. The common stability tools (`tools/stability.py`) compute all of these, with confidence intervals and noise-type fits.

19.2 What the tagger measures

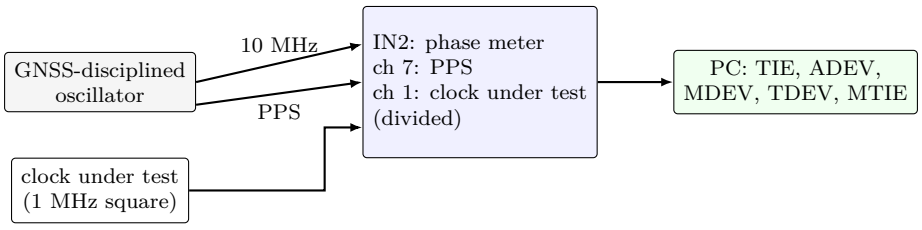


Figure 37: Comparing clocks with the tagger: a GNSS-disciplined oscillator's 10 MHz on IN2 and its PPS on a channel; a clock under test on another channel. All tags are expressed in the reference's time scale on the PC.

Three comparisons come out of one acquisition (figure 37):

- **the tagger's crystal against the reference**, from the IN2 meter's phase: the board's frequency offset, its drift and its dependence on temperature;

- **the reference's PPS against its 10 MHz**, from the PPS tags re-expressed in the 10 MHz's time scale: how well the PPS output follows its oscillator;
- **a clock under test against the reference**, from its tags in the reference's time scale: its frequency offset and stability.

The tagger itself limits such measurements with its timing noise, white phase noise of about 10 ps per tag: an Allan deviation floor of about $\sqrt{3}\,\sigma_x/\tau$, 2×10^{-11} at 1 s for one tag per second, and lower for a clock tagged often, since the tags average.

19.3 Measured: a crystal, a generator and a GNSS-disciplined oscillator

The setup of figure 37 on this manual's bench: a GNSS-disciplined oscillator's 10 MHz on IN2 (`meter_window` 10 ms) and its PPS on channel 7, a function generator's 1 MHz square on channel 1 (divided by 1000 in the FPGA), one acquisition armed on the PPS for 2.0 h (2026-10-31, the GPSDO switched on five hours before and reporting lock; die 47.4–48.8 °C, the board's fan on). The program is `tools/record_manual_data.py --session gpsdo`; the reduction (`reduce_gpsdo`) is a page of the analysis functions above.

The board's crystal against the reference. The IN2 meter's phase is the crystal's time error against the GPSDO (figure 38). The crystal runs −4.50 ppm off its nominal 125 MHz. Its frequency moves by 158 ppb (10 s averages) during the recording, and once the mean frequency is removed its time error still wanders by 81 µs. The die's temperature, drawn inverted, follows it: with the fan on the die follows the room, which changed by 0.6 K (2 min averages), and the crystal's 2 min frequencies follow at −139 ppb per kelvin (correlation −0.71). Heating the board alone (section 5.5) gives −128 ppb per kelvin of the die, against the GPSDO and against the generator alike: an ordinary quartz oscillator, without temperature compensation. Nothing timed in the board's own time scale is better than about 10^{-7} per kelvin of the room, which is why the tags are re-expressed in a reference's.

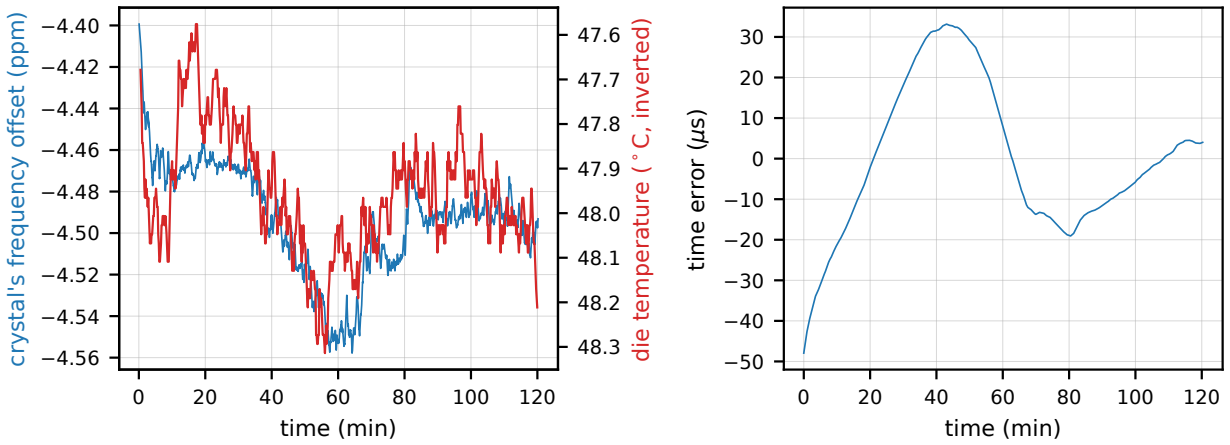


Figure 38: Measured (2026-10-31). The board's crystal against the GNSS-disciplined oscillator, from the IN2 meter. Left: its fractional frequency offset (10 s averages) and the die temperature. Right: its time error once the mean frequency is removed.

The PPS in two time scales. In the board's time, the GPSDO's seconds differ from 1 s by −4491 ns and vary by 26.4 ns rms: the crystal's offset and its wander with the room. Re-expressed through the meter (`to_reference`) the same tags are seconds of the GPSDO's 10 MHz to 35 ps rms (figure 39): the PPS output's own jitter against its oscillator, the tagger's and the meter's interpolation together. The mapping removes a crystal error of parts per million, and its wander, to picoseconds. The first PPS of the armed acquisition is tagged at −6.25 ns (section 11).

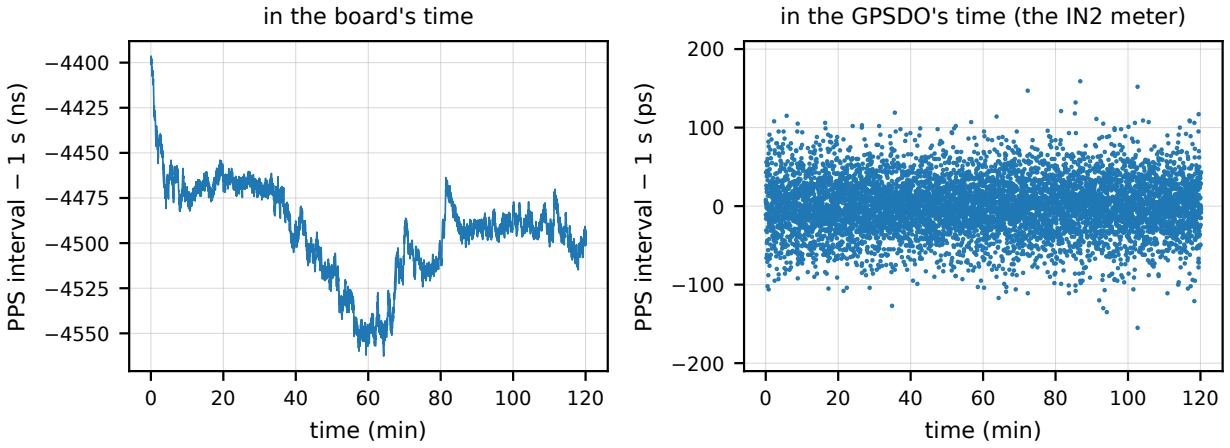


Figure 39: Measured (2026-10-31). The GPSDO's PPS intervals minus one second: left in the board's time scale (the crystal's offset and wander), right in the GPSDO's own time scale through the IN2 meter.

A GNSS-disciplined oscillator's PPS while it warms up

The same GPSDO was first recorded minutes after it was switched on (2026-10-31, morning). In a check just before, its PPS jumped by 208 ns against its 10 MHz, and its rising edge then moved by 2.6 ns per second while the falling edge stayed on the 10 MHz grid: the oscillator moving its second to the satellites'. During that recording it stepped once more, by +200 ns at minute 40, and moved by a few nanoseconds back and forth over the next quarter of a minute (15 seconds more than 1 ns off). Five hours after switch-on the PPS kept to its 10 MHz for the whole recording (0 such seconds; the analyses would take them out). A tagger with the 10 MHz on IN2 shows such events at once. Wait for the oscillator's lock indication and a few hours of warm-up before using it as a reference, and keep the PPS's falling edge (`edge "falling"`, channel −7) or both edges in view if in doubt.

Which clock moves: the three-cornered hat, and its trap. Three clocks compared in pairs separate each one's noise [47, 54]. With the crystal X, the generator F and the GPSDO G, the recording holds all three pairs: X−G from the meter, F−G from the generator's tags in the GPSDO's time, and F−X from the same tags in the board's time. If the three clocks are independent, the pairs' Allan variances are sums of the clocks' own, and with $\sigma_{XG}(\tau)$ the Allan deviation of the pair X−G and $\sigma_X(\tau)$ the crystal's own,

$$\sigma_X^2 = \tfrac{1}{2}(\sigma_{XG}^2 + \sigma_{FX}^2 - \sigma_{FG}^2) \tag{19}$$

and its permutations give each one. Figure 40 applies it. Against the GPSDO the crystal and the generator trace the same curve (their 60 s frequencies correlated by 0.90, swinging by 143 ppb), while against each other they move by 63 ppb; the hat therefore gives the wander to the GPSDO: 1.7×10^{-8} at 1000 s, against the crystal's 8.0×10^{-9}.

That answer is wrong. The GPSDO reported lock to the satellites, and a locked oscillator keeps its time within tens of nanoseconds of theirs, which bounds its Allan deviation at 1000 s to about 10^{-10}. Heating one clock alone decides it: with the board's fan off (section 5.5) the crystal's frequency changed by −128 ppb per kelvin of the die against the GPSDO and against the generator alike, while the generator against the GPSDO moved by 0.3 ppb per kelvin (6.4 ppb rms over the run). That coefficient accounts for the crystal's wander in the room, so the GPSDO did not wander; the generator, which follows the same curve against the GPSDO, follows the room as well, with nearly the same coefficient, and the two cancel in their difference. The hat's one assumption, independent clocks, failed: two oscillators in one room are correlated through its temperature, and the hat gives their common wander to the third clock. Negative estimates are the usual sign of such correlation, and here the generator's estimate is negative near 1000 s; but the absence of negative ones proves nothing. The remedies: perturb one clock at a time, as here; keep the clocks in different rooms

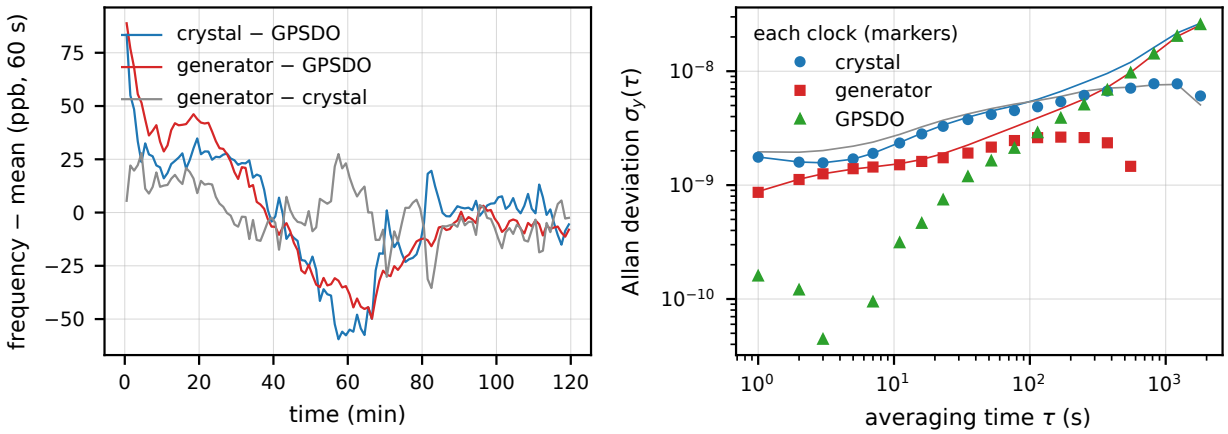


Figure 40: Measured (2026-10-31). Three clocks in pairs. Left: the frequencies of the three pairs over 60 s. Right: their Allan deviations (lines) and each clock's own from equation (19) (markers; a negative estimate is left out). The GPSDO's markers are wrong: they are the wander that the crystal and the generator share (see the text).

or enclosures; or estimate the correlations as well, which needs more than three pairs or more assumptions [55]. Nor can the hat tell how far any of the three is from UTC: that needs a reference known to be locked, or several, compared in the same way.

Stability. Figure 41 compares the clocks against the GPSDO with the Allan deviation; with a locked GPSDO each curve is the clock's own:

- the **board's crystal**: 1–2 × 10^{-9} from 10 ms to 10 s (1.8×10^{-9} at 1 s), then rising with the room's temperature to 1.9×10^{-8} at 1000 s;
- the **function generator** (3.19 ppm against the GPSDO): its edges' jitter, about 141 ps rms (the generator's and the wire's), dominates below 0.1 s as the $1/\tau$ slope of white phase noise; above, its oscillator (8.8×10^{-10} at 1 s) and the room again (1.7×10^{-8} at 1000 s);
- the **PPS against the GPSDO's 10 MHz**: 4.3×10^{-11} at 1 s, falling as $1/\tau$: white phase noise of tens of picoseconds per second. This is the floor of the whole measurement (the PPS, the tagger, the meter): one to two orders of magnitude below the clocks measured at 1 s, and close to the floor of a single tag (dotted).

Against the masks. Telecommunication clocks are specified by TDEV and MTIE masks. Figure 42 shows both for the three series, with the masks of a primary reference clock (G.811) and of a synchronous-Ethernet equipment clock (G.8262 option 1). The free-running crystal and generator leave the masks within seconds (TDEV at 10 s: 12 and 7.5 ns; MTIE over 100 s: 8000 and 9300 ns, with their mean frequency removed), as they must: the masks are for clocks locked to a reference, and beyond a minute both curves follow the room's temperature. The PPS against its 10 MHz stays two to three orders of magnitude below the masks (TDEV 0.0075 ns at 10 s): the tagger, referred to a good reference, can test a clock against these masks with a wide margin.

19.4 Synchronizing distant clocks

Comparing clocks in one room needs only cables; comparing distant ones needs a link whose delay is known or cancels. GNSS time transfer compares each clock with the satellites' time, in common view or through the GNSS time scale [56]; two-way satellite time and frequency transfer sends signals both ways through a geostationary satellite, so that the path delay cancels [57]; in networks, the Precision Time Protocol (IEEE 1588 [58]) exchanges time stamps both ways, and White Rabbit extends it to

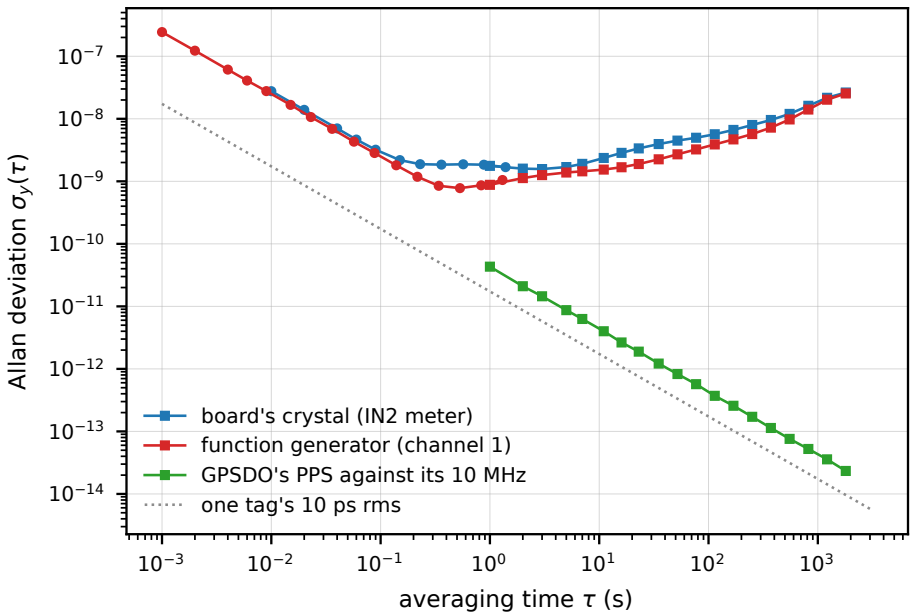


Figure 41: Measured (2026-10-31). Allan deviations against the GPSDO: the board's crystal (from the IN2 meter, circles from 10 ms windows over 200 s, squares from one point per second over the recording), the function generator (circles from its 1 kHz edges over 20 s, squares from one edge per second), and the GPSDO's PPS against its own 10 MHz. Dotted: the white phase noise of one tag of 10 ps rms per sample.

sub-nanosecond accuracy over fibre with synchronous Ethernet and phase measurements [59]. The quantum methods of section 18.4 follow the same two-way principle with photon pairs.

A tagger evaluates all of them the same way: the PPS (or a clock) of the two ends of a link, cabled to two channels in one room or each referred to a common reference, gives the link's time error $x(t)$, and the analyses above its quality. Cable delays enter as offsets: measure each cable once (a split signal, the cables swapped) and set it as the channel's offset.

A PPS stability measurement

1. The reference's 10 MHz to IN2 (HV jumper or 6 dB and 50 Ω), its PPS to channel 7 through the adapter with a 50 Ω termination (section 3.3); the clock under test on another channel (with a divider if it is fast).
2. `tt.apply({"channels": {"7": {"enable": True}}, "reference": {"meter_window": 0.01}})` and an acquisition armed on the PPS, recorded with ttcore (`StreamClient.record`).
3. the report, or the analysis functions directly as above:

```
python3 timetagger/tools/stability.py report run.ttbin --channel 7 \
        --period 1e12 --mask G.8262-1 -o pps.pdf
```

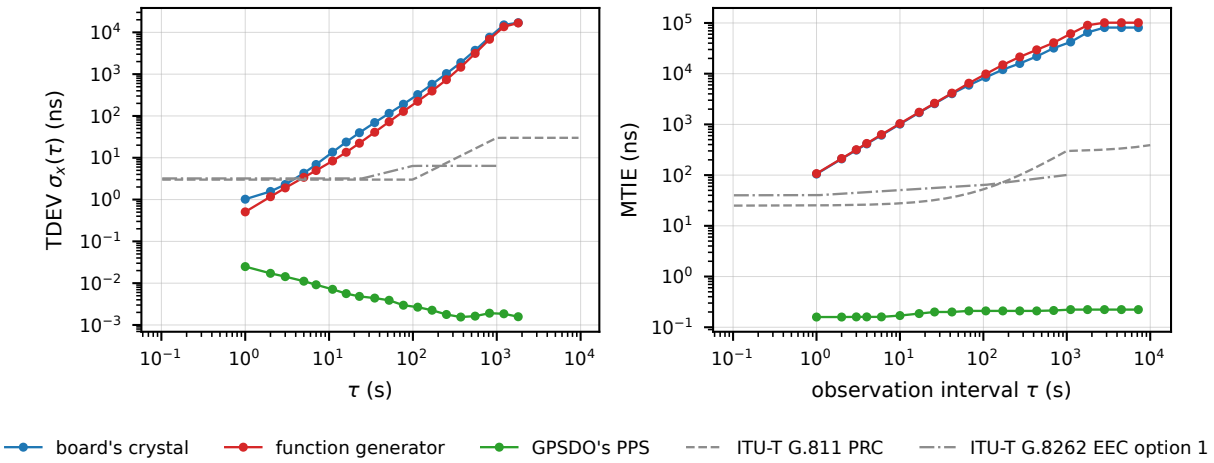


Figure 42: Measured (2026-10-31). TDEV (left) and MTIE (right, mean frequency removed) of the board's crystal, the function generator and the GPSDO's PPS against the GPSDO's 10 MHz, with the ITU-T masks of G.811 and G.8262 (option 1).

A Command reference

A.1 TCP commands

Port 5025, one line per command, `OK` or `ERR` replies (JSON where noted). The common commands of the framework (`CONFIG`, `CONFIG?`, `PRESET`, `SAFE`, `STATUS?`, `*IDN?`, ...) are in `common/README.md`.

command	reply
`ACQ START [AT <sample>] [FOR <s>]`	JSON `{"acquisition", "k0"}`
`ACQ ARM <channel>`	JSON (`k0` null until the event)
`ACQ STOP`	`OK`
`ACQ?`	JSON: running, armed, acquisition, k0, k_stop
`COUNTERS?` / `COUNTERS CLEAR`	JSON as `counters()` / `OK`
`HARDWARE?`	JSON as `hardware()`
`CALIBRATE [<ch> ...]  [SECONDS <s>]`	JSON as `calibrate()` (a long command)
`CALIBRATE DELAYS [<ch> ...]`	JSON as `calibrate_delays()`
`HISTOGRAM?`	the live calibration histogram (JSON)
`TABLES?`	the kept calibration tables (JSON)
`COINCIDENCES?` / `COINCIDENCES CLEAR`	JSON as `coincidences()` / `OK`
`PHASE <steps>`	`OK <steps reached>`
`WORKER?`	the service worker's statistics
`RECORDS STREAM`	the records, on a connection of its own

A.2 Settings

The configuration (`tt.config()`, `tt.apply(patch)`) in seconds; the realized values are returned (dead times in whole samples, offsets in ticks).

key	default	
`channels.n.*`		section 4
`markers.n.enable`, `.edge`	false, rising	section 8
`marker_offset`, `marker_record`	0, true	
`coincidence.enable`, `.channels`, `.window`	false, [], 1 ns	section 9
`coincidence.masks.m.channels`, `.exclusive`, `.virtual`	[], false, false	
`coincidence.gate.marker`, `.invert`	0, false	
`filter.triggers`, `filter.window`	[], 0	section 10
`generator.rate`, `.pair_fraction`	0, 0	events/s; the fraction copied to 32
`generator.offset`, `.pair_offset`, `.record`	0, 0, true	
`test.enable`, `.divider`, `.sata_out`, `.phase_steps`	false, 0, false, 0	section 12
`ring.enable`, `ring.divider`	false, 3	the ring oscillator, divided by 2^{d+1}
`dither`	true	
`connectors`	true	times at the connectors (section 5.6)
`reference.meter_window`	0	seconds, 0 (off) or at least 100 µs

A.3 Records

An event record is one 32-bit word: bits 31–27 the source (0–9 the channels, 10–17 the markers, 20–27 the virtual channels, 28 and 29 the generator), bit 26 the edge (1 rising), bits 25–0 the time's low bits in ticks. Source 31 marks control records: `TIME_HI` and `TIME_TOP` (the time's higher bits, when they change), `EPOCH` (an acquisition's start and k_0), `DROP` (losses per source), `METER` (an IN2 meter window), `NOP`. A frame is a 24-byte header (magic `TTR1`, sequence, bytes, bytes lost before it, and the time context) followed by records. `docs/PROTOCOL.md` has every field.

B Glossary

Accidental coincidence Two uncorrelated events that happen to fall within the coincidence window; on average $R_1 R_2 \tau_w$ per second (section 15.2).

Afterpulse A false detection that follows a real one in the same detector, from charge trapped in a SPAD; it appears in a $g^{(2)}$ or a slot histogram as a tail after the peak.

Allan deviation (ADEV) $\sigma_y(\tau)$: the frequency stability of a clock over an averaging time τ, from the second differences of its time error (equation (18)). MDEV, the modified Allan deviation, averages the phase first.

Armed start An acquisition that starts at the next event of a chosen channel, e.g. a PPS: that event is at $t = 0$ (section 6).

BB84, BBM92 The prepare-and-measure QKD protocol of Bennett and Brassard, and its entanglement-based form of Bennett, Brassard and Mermin, in which a pair source sends one photon to each party (section 17).

CAR Coincidence-to-accidental ratio: true coincidences divided by accidentals in the same window, the figure of merit of a heralded or pair source.

Carry chain The FPGA's fast adder logic (CARRY4 cells), used as a tapped delay line: an edge travels along it, and a snapshot of its taps at the clock edge tells how far it got.

CHSH parameter S from four correlation measurements; above 2 it excludes local hidden variables, $2\sqrt{2}$ for a maximally entangled state (section 17.5).

Code density calibration Many edges at random times fill each tap in proportion to its delay; the histogram of codes gives every bin's width (section 5).

Coincidence window τ_w: the largest time difference counted as a coincidence (full width in this manual's QKD sections; the engine's `window` is measured from a cluster's first event).

Connector delay The delay from a channel's pin to its delay line, subtracted from its times so that they refer to the E1 connector.

Dead time The time after an event during which a channel (or a detector) takes no other; the tagger's is about 5 ns plus any programmed extra.

Decoy states Pulses of other intensities sent at random in BB84, which reveal a photon-number-splitting attack.

DNL, INL Differential and integral nonlinearity: each bin's width against the mean, and the accumulated deviation of the bin edges from a linear scale.

Dither A random fraction of each event's bin added to its time, which spreads the tags evenly instead of on the bins' centres.

Engine-only A channel whose events feed the coincidence engine but are not recorded: coincidences at rates far beyond the stream's.

EPOCH The record at the start of an acquisition, with its number and starting sample.

$g^{(2)}(\tau)$ The normalized second-order correlation of a light field: 1 for coherent light, 2 at $\tau = 0$ for thermal light, below 1 (antibunching) for a single emitter.

GPSDO A GNSS-disciplined oscillator: a quartz or rubidium oscillator steered to the satellites' time, with a 10 MHz and a PPS output.

Heralding Using the detection of one photon of a pair to announce the presence of the other.

HOM dip The Hong–Ou–Mandel effect: two indistinguishable photons meeting at a beam splitter leave together, so coincidences behind it vanish at zero delay.

IRF Instrument response function: the histogram of the delay of a delta-like signal, i.e. the whole system's timing (laser, detector, tagger).

Jitter The random part of a timing error, quoted as rms (FWHM = 2.355×rms for a Gaussian).

Marker One of the eight inputs DIO0_N–DIO7_N timed to 1 ns, for gates, line and frame clocks (section 8).

MTIE Maximum time interval error: the largest peak-to-peak time error of a clock within any window τ.

PPS One pulse per second, its edge marking the second of a time scale.

QBER Quantum bit error rate: the fraction of sifted key bits in which the parties disagree.

Quantization (rms) The timing error from a bin's width alone: $w/\sqrt{12}$ for a bin of width w, averaged over all bins.

Ring, ring pump The buffer in the board's memory that the FPGA writes the records into, and the program on the board that sends them to the PC (section 7).

Sifting Keeping only the events where both parties measured in the same basis.

SNSPD, SPAD Superconducting nanowire single-photon detector (high efficiency, a few to tens of picoseconds of jitter, cryogenic); single-photon avalanche diode (silicon or InGaAs, tens to hundreds of picoseconds, afterpulsing).

TCSPC Time-correlated single-photon counting: the histogram of photon arrival times after the excitation pulses, e.g. a fluorescence decay.

TDEV Time deviation $\sigma_x(\tau) = \tau\,\mathrm{MDEV}(\tau)/\sqrt{3}$: a clock's time stability, the quantity telecommunication masks specify.

Three-cornered hat Three clocks compared in pairs: if they are independent, each one's Allan variance follows from the pairs' (equation (19), section 19.3); two clocks that follow the same room are not.

Tick The tagger's unit of time, 4 ns/4096 = 0.9765625 ps; programs see integer picoseconds.

TIE Time interval error: a clock's time minus the reference's, $x(t)$.

Toggle input An input stage that changes its level at every rising (or falling) edge of the pin, so that a pulse shorter than the delay line still makes one clean transition.

Virtual channel Channels 21–28: an event in the stream at each coincidence of one of the engine's masks.

White PM, white FM Phase noise whose time error is uncorrelated from sample to sample ($\sigma_y \propto \tau^{-1}$), and frequency noise whose frequency is ($\sigma_y \propto \tau^{-1/2}$).

References

[1] D. V. O'Connor and D. Phillips. *Time-Correlated Single Photon Counting*. Academic Press, London, 1984. doi: 10.1016/B978-0-12-524140-3.X5001-1.

[2] W. Becker. *Advanced Time-Correlated Single Photon Counting Techniques*. Springer Series in Chemical Physics. Springer, Berlin, Heidelberg, 2005. doi: 10.1007/3-540-28882-1.

[3] R. H. Hadfield. Single-photon detectors for optical quantum information applications. *Nature Photon.*, 3: 696–705, 2009. doi: 10.1038/nphoton.2009.230.

[4] M. D. Eisaman, J. Fan, A. Migdall, and S. V. Polyakov. Invited review article: Single-photon sources and detectors. *Rev. Sci. Instrum.*, 82:071101, 2011. doi: 10.1063/1.3610677.

[5] A. W. Ziarkash, S. K. Joshi, M. Stipčević, and R. Ursin. Comparative study of afterpulsing behavior and models in single photon counting avalanche photo diode detectors. *Sci. Rep.*, 8:5076, 2018. doi: 10.1038/s41598-018-23398-z.

[6] R. Hanbury Brown and R. Q. Twiss. Correlation between photons in two coherent beams of light. *Nature*, 177:27–29, 1956. doi: 10.1038/177027a0.

[7] H. J. Kimble, M. Dagenais, and L. Mandel. Photon antibunching in resonance fluorescence. *Phys. Rev. Lett.*, 39:691–695, 1977. doi: 10.1103/PhysRevLett.39.691.

[8] C. K. Hong, Z. Y. Ou, and L. Mandel. Measurement of subpicosecond time intervals between two photons by interference. *Phys. Rev. Lett.*, 59:2044–2046, 1987. doi: 10.1103/PhysRevLett.59.2044.

[9] C. K. Hong and L. Mandel. Experimental realization of a localized one-photon state. *Phys. Rev. Lett.*, 56:58–60, 1986. doi: 10.1103/PhysRevLett.56.58.

[10] P. G. Kwiat, K. Mattle, H. Weinfurter, A. Zeilinger, A. V. Sergienko, and Y. Shih. New high-intensity source of polarization-entangled photon pairs. *Phys. Rev. Lett.*, 75:4337–4341, 1995. doi: 10.1103/PhysRevLett.75.4337.

[11] J. F. Clauser, M. A. Horne, A. Shimony, and R. A. Holt. Proposed experiment to test local hidden-variable theories. *Phys. Rev. Lett.*, 23:880–884, 1969. doi: 10.1103/PhysRevLett.23.880.

[12] C. H. Bennett and G. Brassard. Quantum cryptography: Public key distribution and coin tossing. *Theor. Comput. Sci.*, 560:7–11, 2014. doi: 10.1016/j.tcs.2014.05.025. Reprint of the original BB84 paper in Proceedings of the International Conference on Computers, Systems and Signal Processing, Bangalore, India, December 1984, pp. 175–179.

[13] N. Gisin, G. Ribordy, W. Tittel, and H. Zbinden. Quantum cryptography. *Rev. Mod. Phys.*, 74:145–195, 2002. doi: 10.1103/RevModPhys.74.145.

[14] V. Scarani, H. Bechmann-Pasquinucci, N. J. Cerf, M. Dušek, N. Lütkenhaus, and M. Peev. The security of practical quantum key distribution. *Rev. Mod. Phys.*, 81:1301–1350, 2009. doi: 10.1103/RevModPhys.81.1301.

[15] S. Pirandola, U. L. Andersen, L. Banchi, M. Berta, D. Bunandar, R. Colbeck, et al. Advances in quantum cryptography. *Adv. Opt. Photon.*, 12:1012, 2020. doi: 10.1364/AOP.361502.

[16] A. K. Ekert. Quantum cryptography based on Bell's theorem. *Phys. Rev. Lett.*, 67:661–663, 1991. doi: 10.1103/PhysRevLett.67.661.

[17] C. H. Bennett, G. Brassard, and N. D. Mermin. Quantum cryptography without Bell's theorem. *Phys. Rev. Lett.*, 68:557–559, 1992. doi: 10.1103/PhysRevLett.68.557.

[18] S. Wengerowsky, S. K. Joshi, F. Steinlechner, H. Hübel, and R. Ursin. An entanglement-based wavelength-multiplexed quantum communication network. *Nature*, 564:225–228, 2018. doi: 10.1038/s41586-018-0766-y.

[19] S. K. Joshi, D. Aktas, S. Wengerowsky, M. Lončarić, S. P. Neumann, B. Liu, et al. A trusted node–free eight-user metropolitan quantum communication network. *Sci. Adv.*, 6:eaba0959, 2020. doi: 10.1126/sciadv.aba0959.

[20] I. Marcikic, A. Lamas-Linares, and C. Kurtsiefer. Free-space quantum key distribution with entangled photons. *Appl. Phys. Lett.*, 89:101122, 2006. doi: 10.1063/1.2348775.

[21] J. Yin, Y. Cao, Y.-H. Li, S.-K. Liao, L. Zhang, J.-G. Ren, et al. Satellite-based entanglement distribution over 1200 kilometers. *Science*, 356:1140–1144, 2017. doi: 10.1126/science.aan3211.

[22] J. Yin, Y.-H. Li, S.-K. Liao, M. Yang, Y. Cao, L. Zhang, et al. Entanglement-based secure quantum cryptography over 1,120 kilometres. *Nature*, 582:501–505, 2020. doi: 10.1038/s41586-020-2401-y.

[23] C. Ho, A. Lamas-Linares, and C. Kurtsiefer. Clock synchronization by remote detection of correlated photon pairs. *New J. Phys.*, 11:045011, 2009. doi: 10.1088/1367-2630/11/4/045011.

[24] C. Spiess, S. Töpfer, S. Sharma, A. Kržič, M. Cabrejo-Ponce, U. Chandrashekara, et al. Clock synchronization with correlated photons. *Phys. Rev. Applied*, 19:054082, 2023. doi: 10.1103/PhysRevApplied.19.054082.

[25] X. Ma, C.-H. F. Fung, and H.-K. Lo. Quantum key distribution with entangled photon sources. *Phys. Rev. A*, 76:012307, 2007. doi: 10.1103/PhysRevA.76.012307.

[26] J. D. Franson. Nonlocal cancellation of dispersion. *Phys. Rev. A*, 45:3126–3132, 1992. doi: 10.1103/PhysRevA.45.3126.

[27] V. Makarov, A. Anisimov, and J. Skaar. Effects of detector efficiency mismatch on security of quantum cryptosystems. *Phys. Rev. A*, 74:022313, 2006. doi: 10.1103/PhysRevA.74.022313.

[28] B. Qi, C.-H. F. Fung, H.-K. Lo, and X. Ma. Time-shift attack in practical quantum cryptosystems. *Quantum Inf. Comput.*, 7:73–82, 2007. doi: 10.26421/QIC7.1-2-3.

[29] Y. Zhao, C.-H. F. Fung, B. Qi, C. Chen, and H.-K. Lo. Quantum hacking: Experimental demonstration of time-shift attack against practical quantum-key-distribution systems. *Phys. Rev. A*, 78:042333, 2008. doi: 10.1103/PhysRevA.78.042333.

[30] W.-Y. Hwang. Quantum key distribution with high loss: Toward global secure communication. *Phys. Rev. Lett.*, 91:057901, 2003. doi: 10.1103/PhysRevLett.91.057901.

[31] H.-K. Lo, X. Ma, and K. Chen. Decoy state quantum key distribution. *Phys. Rev. Lett.*, 94:230504, 2005. doi: 10.1103/PhysRevLett.94.230504.

[32] X.-B. Wang. Beating the photon-number-splitting attack in practical quantum cryptography. *Phys. Rev. Lett.*, 94:230503, 2005. doi: 10.1103/PhysRevLett.94.230503.

[33] D. Stucki, N. Brunner, N. Gisin, V. Scarani, and H. Zbinden. Fast and simple one-way quantum key distribution. *Appl. Phys. Lett.*, 87:194108, 2005. doi: 10.1063/1.2126792.

[34] L. Calderaro, A. Stanco, C. Agnesi, M. Avesani, D. Dequal, P. Villoresi, et al. Fast and simple qubit-based synchronization for quantum key distribution. *Phys. Rev. Applied*, 13:054041, 2020. doi: 10.1103/PhysRevApplied.13.054041.

[35] H.-K. Lo, M. Curty, and B. Qi. Measurement-device-independent quantum key distribution. *Phys. Rev. Lett.*, 108:130503, 2012. doi: 10.1103/PhysRevLett.108.130503.

[36] M. Lucamarini, Z. L. Yuan, J. F. Dynes, and A. J. Shields. Overcoming the rate–distance limit of quantum key distribution without quantum repeaters. *Nature*, 557:400–403, 2018. doi: 10.1038/s41586-018-0066-6.

[37] R. Jozsa, D. S. Abrams, J. P. Dowling, and C. P. Williams. Quantum clock synchronization based on shared prior entanglement. *Phys. Rev. Lett.*, 85:2010–2013, 2000. doi: 10.1103/PhysRevLett.85.2010.

[38] I. L. Chuang. Quantum algorithm for distributed clock synchronization. *Phys. Rev. Lett.*, 85:2006–2009, 2000. doi: 10.1103/PhysRevLett.85.2006.

[39] V. Giovannetti, S. Lloyd, and L. Maccone. Quantum-enhanced positioning and clock synchronization. *Nature*, 412:417–419, 2001. doi: 10.1038/35086525.

[40] A. Valencia, G. Scarcelli, and Y. Shih. Distant clock synchronization using entangled photon pairs. *Appl. Phys. Lett.*, 85:2655–2657, 2004. doi: 10.1063/1.1797561.

[41] J. Lee, L. Shen, A. Cerè, J. Troupe, A. Lamas-Linares, and C. Kurtsiefer. Symmetrical clock synchronization with time-correlated photon pairs. *Appl. Phys. Lett.*, 114:101102, 2019. doi: 10.1063/1.5086493.

[42] F. Hou, R. Quan, R. Dong, X. Xiang, B. Li, T. Liu, et al. Fiber-optic two-way quantum time transfer with frequency-entangled pulses. *Phys. Rev. A*, 100:023849, 2019. doi: 10.1103/PhysRevA.100.023849.

[43] H. Hong, R. Quan, X. Xiang, Y. Liu, T. Liu, M. Cao, et al. Quantum two-way time transfer over a 103 km urban fiber. *J. Lightwave Technol.*, 42:1479–1486, 2024. doi: 10.1109/JLT.2023.3323434.

[44] B. Shi, H. Hong, X. Xiang, R. Quan, Y. Liu, T. Liu, et al. Quantum two-way time transfer over a 250 km direct fiber-optic link. *Opt. Express*, 32:43805, 2024. doi: 10.1364/OE.536691.

[45] J. Lee, L. Shen, A. N. Utama, and C. Kurtsiefer. Absolute clock synchronization with a single time-correlated photon pair source over a 10 km optical fibre. *Opt. Express*, 30:18530–18538, 2022. doi: 10.1364/OE.455542.

[46] D. W. Allan. Statistics of atomic frequency standards. *Proc. IEEE*, 54:221–230, 1966. doi: 10.1109/PROC.1966.4634.

[47] W. J. Riley. Handbook of Frequency Stability Analysis. NIST Special Publication 1065, National Institute of Standards and Technology, Gaithersburg, MD, 2008. URL https://doi.org/10.6028/NIST.SP.1065.

[48] E. Rubiola. *Phase Noise and Frequency Stability in Oscillators*. Cambridge University Press, Cambridge, 2008. doi: 10.1017/CBO9780511812798.

[49] IEEE. IEEE Standard Definitions of Physical Quantities for Fundamental Frequency and Time Metrology—Random Instabilities. IEEE Std 1139-2008, 2008. URL https://doi.org/10.1109/IEEESTD.2008.4797525. Approved 26 September 2008, published 27 February 2009; superseded by IEEE Std 1139-2022.

[50] D. W. Allan and J. A. Barnes. A modified "Allan variance" with increased oscillator characterization ability. In *Proceedings of the 35th Annual Frequency Control Symposium*, pages 470–475, 1981. doi: 10.1109/FREQ.1981.200514.

[51] ITU-T. Definitions and terminology for synchronization networks. Recommendation ITU-T G.810 (08/96), https://www.itu.int/rec/T-REC-G.810-199608-I/en, 1996. With Corrigendum 1 (11/01).

[52] ITU-T. Timing characteristics of primary reference clocks. Recommendation ITU-T G.811 (09/97), https://www.itu.int/rec/T-REC-G.811-199709-I/en, 1997. With Amendment 1 (04/16).

[53] ITU-T. Timing characteristics of synchronous equipment clocks. Recommendation ITU-T G.8262 (10/24), https://www.itu.int/rec/T-REC-G.8262-202410-I/en, 2024.

[54] J. E. Gray and D. W. Allan. A method for estimating the frequency stability of an individual oscillator. In *Proceedings of the 28th Annual Symposium on Frequency Control*, pages 243–246, May 1974. URL https://tf.nist.gov/general/pdf/57.pdf.

[55] A. Premoli and P. Tavella. A revisited three-cornered hat method for estimating frequency standard instability. *IEEE Trans. Instrum. Meas.*, 42(1):7–13, 1993. doi: 10.1109/19.206671.

[56] W. Lewandowski, J. Azoubib, and W. J. Klepczynski. GPS: Primary tool for time transfer. *Proc. IEEE*, 87:163–172, 1999. doi: 10.1109/5.736348.

[57] D. Kirchner. Two-way time transfer via communication satellites. *Proc. IEEE*, 79:983–990, 1991. doi: 10.1109/5.84975.

[58] IEEE. IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems. IEEE Std 1588-2019, 2019. URL https://doi.org/10.1109/IEEESTD.2020.9120376. Approved 7 November 2019, published 16 June 2020.

[59] M. Lipiński, T. Włostowski, J. Serrano, and P. Alvarez. White Rabbit: A PTP application for robust sub-nanosecond synchronization. In *2011 IEEE International Symposium on Precision Clock Synchronization for Measurement, Control and Communication (ISPCS)*, pages 25–30, 2011. doi: 10.1109/ISPCS.2011.6070148.