counter: user's manual

A gated photon counter with on-board decisions and coincidence counting, on the Red Pitaya STEMlab 125-14

the instrument, the theory behind it, and its use in quantum-physics labs

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

Red Pitaya `counter` design, version 1.0 (build 0x894f27b0)
November 6, 2026

Abstract

`counter` turns the Red Pitaya into a **gated photon counter**: it counts the pulses of single-photon detectors on ten inputs, in time windows (gates) that follow the trigger of every repetition of an experiment, and it does so with a time resolution of one nanosecond. It adds the counts of every point of a scan (a Rabi or Ramsey sweep, an ODMR spectrum, the pixels of an image) into sums on the board, so that a run can last hours without a computer; it decides within about 105.0 ns after a gate closes whether the count was above or below a threshold, and signals the decision on a pin (state detection of an ion or an NV centre, heralding, preselection); it counts coincidences between up to four detectors with the same definition as a time tagger, and the accidental coincidences with a delayed copy; it writes a record of every shot for count histograms and fidelities computed on a PC. Two of its inputs take analog pulses (NIM, amplified superconducting-nanowire or photomultiplier pulses) through discriminators set in volts. Part I of this manual describes the instrument: what every input, output and setting is for, how it behaves, and how it was verified. Part II is the theory behind it: the statistics of counts, detectors, dead time and discriminators, the readout of quantum states by counting, coincidences and correlations, and the inside of the FPGA. Part III shows how it is used in quantum-physics labs: nitrogen-vacancy centres, trapped ions and neutral atoms, single photons and entangled pairs, with the code that runs the measurements. Terms of physics and of test and measurement are explained where they first appear and collected in the glossary. Figures marked Measured were recorded with the board used here (most with the instrument's built-in test emitter, some with a second Red Pitaya and an oscilloscope); figures marked Computed come from simulations analysed with the design's own functions.

Contents

I The instrument **6**

1 Introduction **6**
1.1 What counter is . . . 6
1.2 Photon counting in a few paragraphs . . . 7
1.3 How it works . . . 7
1.4 Time on the board . . . 8
1.5 counter among the rpfpga instruments . . . 8
1.6 About this manual . . . 9

2 First steps **9**
2.1 What you need . . . 9
2.2 Loading counter . . . 9
2.3 Counts without a detector . . . 10
2.4 A first pulsed measurement . . . 10
2.5 Connecting a detector . . . 11
2.6 Where to go next . . . 11

3 Signals, connectors and levels **11**
3.1 The connectors . . . 11
3.2 Logic levels . . . 12
3.3 Detectors and their pulses . . . 13
3.4 Cables, termination and crosstalk . . . 13
3.5 The analog inputs IN1 and IN2 . . . 14
3.6 The analog outputs OUT1 and OUT2 . . . 14
3.7 The outputs O1–O4 . . . 14
3.8 Pulse generators . . . 14

4 Count channels C1–C8 **15**
4.1 Sampling every nanosecond . . . 15
4.2 Dead time . . . 15
4.3 Delays . . . 17
4.4 Totals and rejected edges . . . 17

5 Analog channels C9 and C10 **17**
5.1 The discriminator . . . 17
5.2 Choosing the threshold . . . 18
5.3 Pulse shape, pair resolution and timing . . . 18
5.4 A detector simulated by awg . . . 19
5.5 Counting a reference frequency . . . 20

6 Rates, RATE records, monitors and the alarm **20**
6.1 Rates on request . . . 20
6.2 RATE records . . . 20
6.3 Rate monitors on OUT1 and OUT2 . . . 21
6.4 The rate alarm . . . 21

7 Shots, points, sweeps and the per-point sums **22**
7.1 Shots and the trigger . . . 22
7.2 Hold-off . . . 22
7.3 Runs . . . 23
7.4 Points and sweeps . . . 23
7.5 Per-point sums . . . 24

8 Gates **24**
8.1 What a gate counts . . . 24
8.2 The GATE input . . . 25
8.3 Signal and reference . . . 25
8.4 The gate-delay scan: time traces without histograms . . . 26

9 Decisions **26**
9.1 What a decision is . . . 27
9.2 The decision's pin . . . 27
9.3 Latency . . . 28
9.4 Feed-forward with a sequencer . . . 29

10 Outputs O1–O4 **30**
10.1 What each source is for . . . 30
10.2 Safe state . . . 31

11 Coincidence units **31**
11.1 Clusters . . . 31
11.2 The delayed unit: accidentals and delay scans . . . 32
11.3 Gating and heralds . . . 32

12 Records and the stream **34**
12.1 The records . . . 34
12.2 From the FPGA to a PC . . . 34

13 The test emitter **35**
13.1 Lanes . . . 35
13.2 Pairs and antibunching . . . 36
13.3 Bright and dark shots . . . 36
13.4 To the outputs and to OUT1, OUT2 . . . 37

14 Timing, calibration and the timebase **37**
14.1 The board calibration . . . 37
14.2 The analog channels' alignment . . . 37
14.3 Latencies . . . 38
14.4 Below a nanosecond . . . 38
14.5 The timebase . . . 39

15 Software **40**
15.1 The pieces . . . 40
15.2 Settings . . . 40
15.3 Presets . . . 41
15.4 counter's methods . . . 41
15.5 Analysis . . . 41
15.6 The notebook . . . 41
15.7 TCP . . . 42

16 Self-tests, verification and troubleshooting **43**
16.1 The board test . . . 43
16.2 How the design was verified . . . 43
16.3 The session with awg and an oscilloscope . . . 43
16.4 The tests on awg's edges . . . 44
16.5 The tests of the next day . . . 45
16.6 Troubleshooting . . . 45

17 Specifications **46**

II Theory **47**

18 Counting statistics **47**
18.1 Poisson statistics . . . 48
18.2 The Fano factor: sub- and super-Poissonian counts . . . 48
18.3 Dead time and the statistics of counts . . . 48
18.4 Signal, background and the signal-to-noise ratio . . . 49
18.5 Ratios, contrasts and their errors . . . 49
18.6 Beyond the counts: drifts . . . 50

19 Detectors, discriminators and dead time **50**
19.1 The figures of merit . . . 50
19.2 Afterpulses . . . 50
19.3 Discriminators . . . 51
19.4 Gating the detector . . . 51

20 Reading out quantum states by counting **51**
20.1 Two states, two Poisson distributions, a threshold . . . 51
20.2 When the states are not Poissonian . . . 52
20.3 The detection time . . . 53
20.4 Adaptive and time-resolved readout . . . 53
20.5 Correcting readout errors . . . 53
20.6 Averaged readout: the NV centre . . . 53

21 Coincidences and correlations **53**
21.1 Accidental coincidences . . . 53
21.2 The second-order correlation $g^{(2)}$ . . . 54
21.3 Heralded single photons . . . 54
21.4 Correlations and Bell's inequality . . . 55
21.5 Errors of coincidence quantities . . . 55

22 Inside the FPGA **55**
22.1 Words, slots and lanes . . . 55
22.2 The samplers . . . 55
22.3 Channels, dead time and delays . . . 55
22.4 Shots and gates . . . 56
22.5 Decisions and the latency . . . 56
22.6 Per-point sums . . . 56
22.7 Coincidence clusters . . . 56
22.8 The analog discriminators . . . 57
22.9 The test emitter . . . 57
22.10Records . . . 57
22.11Resources and verification . . . 57

III Counting in quantum labs **57**

23 NV centres **57**
23.1 The setup . . . 58
23.2 Photoluminescence, saturation and images . . . 58
23.3 The readout window . . . 58
23.4 Rabi oscillations . . . 59
23.5 Pulsed ODMR, Ramsey, echoes and relaxation . . . 60
23.6 Charge-state preselection . . . 60
23.7 How fast . . . 61

24 Trapped ions and neutral atoms **61**
24.1 The setup . . . 61
24.2 State detection: threshold and fidelity . . . 61
24.3 Quantum jumps and the loss of an ion . . . 62
24.4 Micromotion: the photon–RF correlation . . . 63
24.5 Neutral atoms in tweezers . . . 64
24.6 Preselection with awg on a second board . . . 64

25 Photons: sources, correlations and entanglement **64**
25.1 Detectors and their delays . . . 65
25.2 A photon-pair source . . . 65
25.3 Antibunching of a single emitter . . . 66
25.4 Heralded single photons: Grangier's experiment . . . 66
25.5 Hong–Ou–Mandel interference . . . 66
25.6 Polarization entanglement and Bell's inequality . . . 67
25.7 Pulsed sources and gated detection . . . 67
25.8 In the teaching lab . . . 68

A Configuration reference **68**

B TCP commands and Python methods **70**
B.1 Common commands (every rpfpga design) . . . 70
B.2 counter's commands . . . 70

C Record formats **71**

D Registers **72**

E Glossary **77**

Part I
The instrument

1 Introduction

1.1 What counter is

`counter` is an FPGA design for the Red Pitaya STEMlab 125-14 that makes it a ***gated photon counter***: an instrument that counts the pulses of single-photon detectors inside time windows, the ***gates***, which open at a chosen delay after the ***trigger*** of each repetition of an experiment. It is the detection half of a pulsed experiment; the pulse sequence (laser and microwave pulses, the trigger) comes from a pulse generator such as `awg` on a second Red Pitaya, a PulseBlaster, a Pulse Streamer or an ARTIQ system. It has

- **eight count inputs** C1–C8 on the extension connector E1 (DIO0_P–DIO7_P, 3.3 V logic), each sampled every nanosecond, with a dead time and a delay of its own (section 4), and **two analog count inputs** C9 and C10 on the SMA inputs IN1 and IN2: discriminators with a threshold in volts for NIM pulses or the amplified pulses of superconducting-nanowire detectors and photomultipliers (section 5);
- **four control inputs**: TRIG starts a ***shot*** (one repetition of the experiment), GATE is an external gate level, NEXT advances the ***point*** of a scan and SEQ marks the start of a ***sweep*** through the points (section 7);
- **four gates** per shot, each with a delay and a width in 1 ns steps, a set of channels it counts, and a delay step per point that turns a scan of points into a time trace (section 8);
- **per-point sums** on the board: for each of up to 1024 points, the counts of the four gates, the number of shots, the decisions' outcomes and the sum of squared counts, collected by the board's service into 64-bit totals, so a run can last for days without a computer (section 7.5);
- **two decisions** per shot: a count window on each of two gates and any logic of the two ("bright", "both detectors clicked"), signalled on an output pin about 105.0 ns after the gate closes, or as soon as the count reaches its threshold (section 9);
- **four outputs** O1–O4 (DIO4_N–DIO7_N) with 2 ns resolution: the decisions, the gates' windows, an echo of the trigger or of a channel (a clean logic pulse for every counted NIM pulse), the herald of a coincidence, the rate alarm, a pattern (section 10);
- **two coincidence units** of up to four channels each, counting all 15 patterns of clicks with the same definition of a coincidence as the `timetagger` design; the second unit can delay one channel by up to 16 µs to count the accidental coincidences directly (section 11);
- **totals and rates**: 48-bit totals of every channel and every coincidence pattern, RATE records at fixed intervals from 10 µs to 10 s, a rate alarm, and the rate of chosen channels as a voltage on OUT1 or OUT2 for alignment (section 6);
- **records** of every shot, rate interval, sweep and run, streamed to any number of programs on PCs at up to 2 million shots per second (section 12);
- a **test emitter**: pseudo-random detector pulses on any channel, in correlated pairs or anti-bunched, bright or dark per shot from a table of the points, so that every function can be tried, and an experiment rehearsed, without a detector (section 13);
- **software**: Python on the board and on PCs, a Jupyter panel, a TCP command set, presets for the common experiments, and an analysis module with the fits, thresholds, fidelities and coincidence quantities of Part III (section 15).

1.2 Photon counting in a few paragraphs

A ***single-photon detector*** turns each photon it detects into a short electrical pulse, a ***click***. A single-photon avalanche diode (SPAD) module gives a logic pulse of a few to a few tens of nanoseconds; a photomultiplier tube (PMT) gives a short negative current pulse whose height varies from photon to photon; a superconducting-nanowire detector (SNSPD) gives a pulse of a few hundred millivolts after its amplifier. Section 19 describes them. Counting the clicks is the most sensitive way to measure light: at a few photons per second it is the only way.

In a pulsed experiment the same sequence is repeated many times: prepare a quantum system (an NV centre's electron spin, an ion's qubit), let it evolve, read it out by shining light on it and counting the fluorescence photons. Each repetition is a ***shot***. A ***trigger*** pulse from the sequencer marks the start of each shot's readout. The photons that matter arrive in a window of known delay and length after the trigger: counting only there, in a ***gate***, rejects the background (detector dark counts, stray light) that arrives at all other times. One shot gives a handful of counts or none; the answer is in the average over thousands to millions of shots, or, when one shot gives enough photons (an ion's fluorescence), in a decision per shot: bright or dark.

A scan changes a parameter from shot to shot: the length of a microwave pulse in a Rabi measurement, a frequency in a spectrum, the position of a laser focus in an image. Each value is a ***point***; one pass through all points is a ***sweep***. The counter keeps the counts of each point apart and adds them over the sweeps, on the board. The statistics of the counts (Poisson's, section 18) set the error bars.

Some decisions cannot wait for a computer: a measurement result that must select what the sequence does next (***feed-forward***), a check that an atom is in its trap before a long experiment starts (***preselection***), a photon that announces its partner (***heralding***). counter decides on the board and signals the decision on a pin about a tenth of a microsecond after the gate closes.

Two or more detectors that click within a short window, a ***coincidence***, show that photons arrived together: the signature of photon pairs, of entanglement, and, by their absence, of a single-photon source. counter counts coincidences among up to four detectors, and the ***accidental*** coincidences of independent photons, on the board (section 11).

1.3 How it works

Figure 1 shows the paths of the signals. The FPGA runs on a 125 MHz clock: every 8 ns it takes one ***word*** of eight 1 ns samples of every input (the ***slots***). A channel finds the rising (or falling) edges in its words, keeps one edge per dead time and delays the result; the gates count the channels' edges inside their windows; the decisions, the per-point sums and the records take the gates' counts at the end of the shot. Every step works on whole words of eight slots: the instrument resolves 1 ns while its logic runs at 125 MHz.

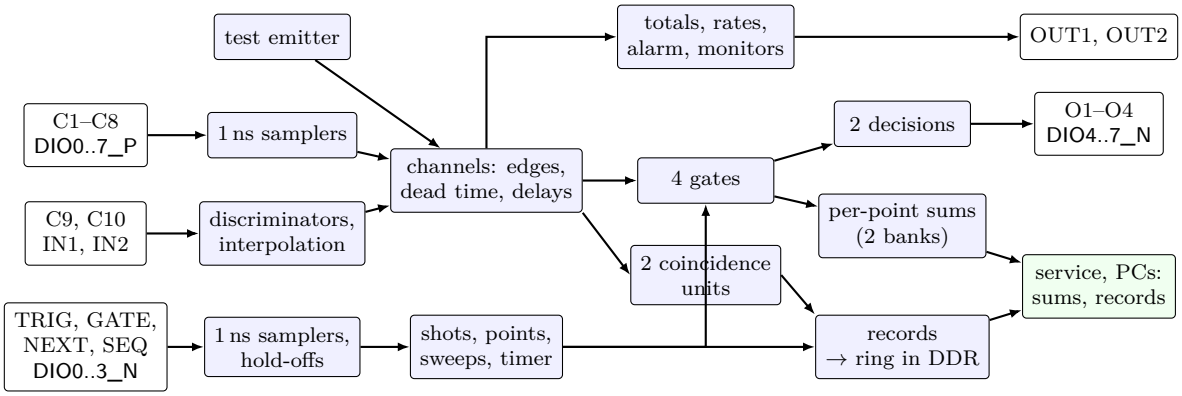


Figure 1: The paths of the signals. Blue: inside the FPGA; green: the board's processor (the service, which collects the sums and serves the records) and PCs.

The board's ARM processor runs Linux and a service that loads the design, applies settings,

collects the per-point sums every second and serves the record stream; programs on PCs talk to it over the network (section 15).

1.4 Time on the board

Three facts about time run through this manual.

- **Times refer to the connectors.** A delay of 100 ns means 100 ns between the edges at the board's connectors. The paths inside the board (the input buffers, the samplers, the converters of IN1 and IN2) are taken out using the board calibration (section 14): a TTL pulse on DIO0_P and a NIM pulse on IN1 that reach the connectors together get the same time.
- **The resolution is 1 ns.** Edges are known to the slot they fell in; delays, widths and windows are set in whole nanoseconds. Finer timing (picoseconds, the shape of a lifetime or of $g^{(2)}(\tau)$ near zero delay) is the job of a time tagger (section 1.5).
- **A channel takes at most one event per 8 ns.** The dead time of every channel is at least 8 ns, one word: 125 million counts per second per channel at most. Real detectors have dead times of 20–100 ns, so this costs nothing in practice, and it keeps every step of the logic simple and fast (section 22).

1.5 counter among the rpfpga instruments

`counter` is one of several designs for the same board [1], built on a common framework (`rpfpga`, on PYNQ [2]) with the same installation, service, Python interface, calibration and SD card image. A board runs one design at a time; the service switches designs in a few seconds (`LOAD awg`). Table 1 places counter among them.

Design	What it is, and how it relates to counter
`timetagger`	A ten-channel time tagger: every edge timed to about 10 ps and sent to a PC, where correlations, lifetimes (TCSPC) and coincidences are computed. Use it when the time of each photon matters to better than a nanosecond ($g^{(2)}(\tau)$ near $\tau = 0$, lifetimes below a few ns, quantum key distribution). counter keeps only counts, decides on the board and needs no PC during a run; its coincidence units use timetagger's definition of a coincidence, so the two agree. The same detector cabling (channels 1–8 on DIO0..7_P) serves both.
`awg`	A two-channel arbitrary waveform generator with eight digital outputs, a sequencer and event inputs: the natural pulse generator for counter's experiments (laser, microwave and trigger pulses), and a sequence that branches on counter's decisions (sections 9, 24.6).
`synth`	Up to 16 tones per output with timed sequences: radio-frequency drives below about 60 MHz (an ion trap's secular-frequency "tickle", an acousto-optic modulator's driver input through an amplifier and mixer).
`lockin`, `feedback`	A lock-in amplifier and a lock box: laser locks and analog detection of modulated light; feedback can also hold an experiment at the maximum of counter's rate monitor.
`atomic_clock`	A 10 MHz source with controlled phase noise; counter measures a 10 MHz reference on IN2 (section 5.5).
`ajku`	An analyser of optical I/Q (homodyne) signals: continuous-variable quantum optics, the counterpart of counter's discrete counting.
`mult_add`, `analog_echo`	Analog computing and signal conditioning on the inputs and outputs.

Table 1: The rpfpga designs and counter. Setups of several boards running different designs together are the subject of a separate document.

1.6 About this manual

Part I takes the instrument a function at a time: what each input, output and setting is for, how it behaves at its limits, and how it was verified (green boxes). Part II explains the physics and the engineering behind the numbers: why counts scatter as they do, what dead time and discriminators do to them, how a threshold separates bright from dark, what a coincidence is, and what happens inside the FPGA. Part III applies all of it to experiments with NV centres, trapped ions and atoms, and photons. The appendices are the reference: every setting with its range and default, every command, every record's layout, the registers, and a glossary.

Examples are Python. On a PC, `dev = CounterClient("192.168.0.102")` connects to the board (the board's address in your network); on the board itself, `dev = DRIVER()` after `import rpfpga`; the methods are the same. Settings are a nested dictionary (`dev.apply({...})` changes the given ones); times are in seconds, voltages in volts at the connectors. The equivalent TCP command is shown below an example where there is one:

TCP: `CONFIG {"channels": {"1": {"enable": true}}}`

Pin names follow the board's silkscreen and schematics: DIO0_P is the positive line of the extension connector's pair 0, E1 pin 3 (figure 3).

2 First steps

2.1 What you need

- A Red Pitaya STEMlab 125-14 with the rpfpga SD card image (the project page has the image and the instructions to write it). The image holds every design; the board's address is the one your network's DHCP server gives it, and also the fixed address 192.168.2.99 on a direct cable.
- A PC with Python 3 on the same network, for scripts, or just a web browser for the board's Jupyter notebooks (`http://<address>:9090`, password `xilinx`).
- For the first measurements nothing else: the ***test emitter*** inside the FPGA makes detector-like pulses (section 13). Later: your detectors, a pulse generator, and cables (section 3).

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.

Caution

The extension-connector pins take 0 to 3.3 V. They are **not 5 V tolerant**: a TTL detector module that drives 5 V into a high impedance must be terminated with 50 Ω at the board (it then gives about 2.5 V), and NIM pulses (negative, −0.8 V) go to IN1 or IN2, not to the pins. Join the grounds of all instruments before connecting signals. Read the equipment-safety section of the repository's `README.md` and section 3 before wiring anything.

2.2 Loading counter

The board's service (`rp-overlay.service`) runs one design at a time. From a PC:

```
from rpfpga.client import Client
Client("192.168.0.102").load("counter")       # a few seconds; the outputs start in high impedance
```

TCP: `LOAD counter`

The repository's Python packages are on the board already; on a PC, add `common/python` and `counter/control` of the repository to Python's path (or install them). Then

```python
from counter import CounterClient
dev = CounterClient("192.168.0.102")
print(dev.idn())         # RedPitaya,rpfpga,counter,1.0,0x894f27b0
print(dev.status())      # the version, the build, the board calibration in use, ...
```

The notebook `counter.ipynb` in the board's Jupyter does the same and opens the panels of section 15.6.

2.3 Counts without a detector

The `selftest` preset turns on the test emitter on C1–C4: about 1 million counts per second on C1, half of them copied 16 ns later to C2, and a source of 2 million per second split between C3 and C4 so that they never click together (an ***antibunched*** source, section 13).

```python
dev.load_preset("selftest")
print(dev.rates(1.0)["channels"])    # counts per second of C1..C8, from two readings 1 s apart
print(dev.counters()["channels"]["1"])   # the 48-bit total and the rejected edges of C1
```

TCP: `PRESET selftest`
TCP: `RATES? 1`

A ***total*** counts from the moment the design was loaded and is never cleared; a rate is the difference of two totals over the time between them (section 6).

2.4 A first pulsed measurement

The `shots-selftest` preset runs the experiment of figure 2 without any wiring: the internal timer triggers a shot every 10 µs; the points run through 16 values, one per shot; gate G1 counts C1 for 2 µs after each trigger; the emitter on C1 is "bright" in a shot with the probability of that shot's point, taken from a table.

```python
dev.load_preset("shots-selftest")
dev.load_emitter_table([i / 15 for i in range(16)])   # point i: bright with probability i/15
dev.run()
time.sleep(2)
dev.stop()
r = dev.results(now=True)
print(r["shots"])          # shots per point
print(r["mean"]["n1"])     # counts of G1 per shot, per point: from about 1 to about 5
```

TCP: `RUN`
TCP: `STOP`
TCP: `RESULTS? NOW`

The counts per shot rise with the point, as the table says: dark shots give the emitter's background, 0.5 million per second over 2 µs (1.0 count), bright shots 5 million per second more at the start, decaying with a time constant of 1 µs. This is how a Rabi oscillation is measured with a real system: replace the table by a microwave pulse of increasing length, the emitter by the fluorescence of an NV centre, and the timer by the sequencer's trigger (section 23.4).

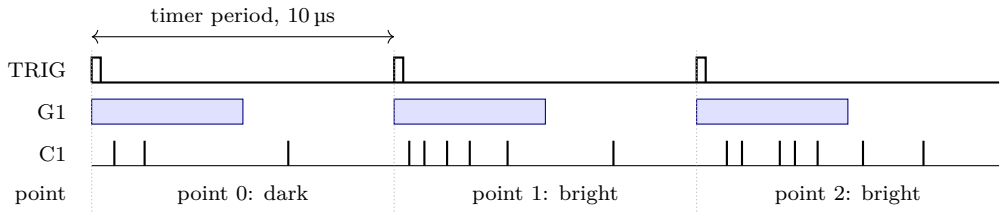


Figure 2: The first measurement: a shot per trigger, gate G1 after each, the counts of C1 in the gate added to the shot's point.

2.5 Connecting a detector

A SPAD module (single-photon counting module) with a TTL output goes to C1 (DIO0_P, E1 pin 3, ground on pin 25 or 26), through 50 Ω coax with a 50 Ω terminator at the board if the module drives 50 Ω loads (section 3.2). Then

```
dev.load_preset("rates")              # C1..C8 counting, RATE records, C1's rate on OUT1
dev.apply({"channels": {"1": {"dead_time": 22e-9}}})   # the module's own dead time, or more
print(dev.rates(1.0)["channels"]["1"])                  # the dark count rate with the light off
```

The dark count rate with the light blocked should match the module's data sheet (tens to thousands per second). With light, OUT1 shows the rate as a voltage, 1 V per million counts per second: on a voltmeter, an oscilloscope or, through an audio amplifier, as a pitch, it is the fastest way to align optics (section 6.3). A NIM detector or a photomultiplier with a discriminator, or an SNSPD's amplified pulses, go to IN1 (C9) instead (section 5).

2.6 Where to go next

- Wiring a pulsed experiment: the pins and levels (section 3), the shots and points (section 7), the gates (section 8).
- A complete experiment with code: Part III (NV centres, section 23; ions and atoms, section 24; photons, section 25).
- Every setting at a glance: appendix A.

3 Signals, connectors and levels

This section is about the electrical side: which pin does what, which voltages they take and give, and how detectors and pulse generators are connected so that every pulse arrives as one clean edge. Section 16.6 lists what goes wrong when they do not.

3.1 The connectors

counter uses the extension connector **E1** (a 2×13 pin header with 2.54 mm pitch, figure 3), the SMA inputs **IN1, IN2** and the SMA outputs **OUT1, OUT2**. The second extension connector E2 (slow analog pins) and the SATA connectors are not used. Table 2 lists every signal.

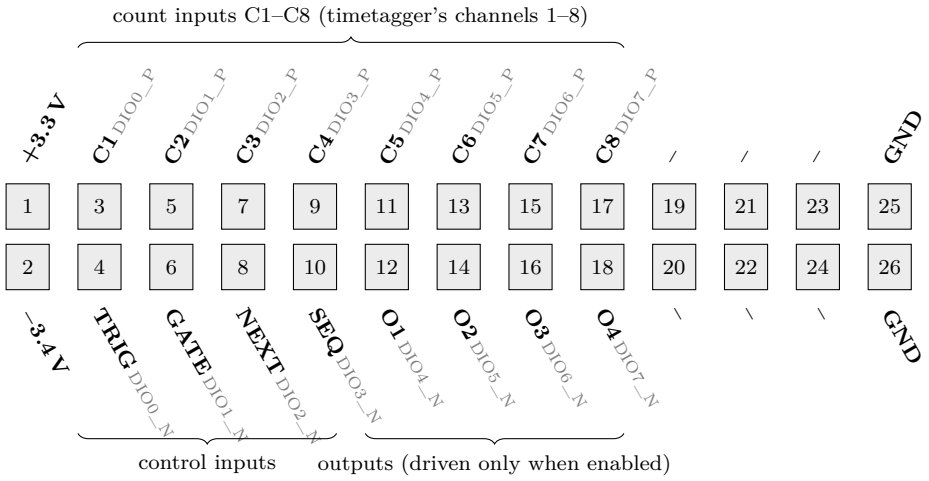


Figure 3: The extension connector E1 as counter uses it: odd pins in the upper row, even pins in the lower; pin 1 is marked on the board. Pins 19–24 are not used by counter; pins 1 and 2 supply power: never connect a signal to them.

Signal	Pin	Direction	What it is for (section)
C1–C8	DIO0_P–DIO7_P, E1 3, 5, . . . , 17	in	detector pulses, counted on rising or falling edges, sampled every 1 ns (4)
C9, C10	IN1, IN2 (SMA)	in	analog detector pulses through a discriminator in volts (5)
TRIG	DIO0_N, E1 4	in	the start of a shot: the gates' time zero (7)
GATE	DIO1_N, E1 6	in	an external gate level: gates in `gate` mode count while it is active (8)
NEXT	DIO2_N, E1 8	in	advances the point (with `point_mode per-next`) (7.4)
SEQ	DIO3_N, E1 10	in	the start of a sweep: point 0, and the sweep's check (7.4)
O1–O4	DIO4_N–DIO7_N, E1 12, 14, 16, 18	out	decisions, gates, echoes, heralds, the alarm, a pattern (10)
OUT1, OUT2	SMA	out	rate monitors in volts, or test pulses for C9, C10 (6.3)
GND	E1 25, 26		the common ground of all signals

Table 2: counter's signals.

All inputs have ***pull-down*** resistors inside the FPGA: an input with nothing connected reads low and counts nothing. An output pin is in ***high impedance*** (not driven, as if disconnected) from the moment the design loads until its source is chosen, and again after `SAFE` (section 10.2).

3.2 Logic levels

The E1 pins are ***LVCMOS33*** pins of the FPGA: 3.3 V CMOS logic. An input reads high above $V_{IH} = 2.0\,V$ and low below $V_{IL} = 0.8\,V$; between the two the threshold lies (typically about 1.5 V) but is not specified. An output swings from 0 to 3.3 V with up to 12 mA of current and fast edges (about 1 ns). The pins must stay between about −0.5 and +3.8 V: **they are not 5 V tolerant**, and a 5 V signal into a high-impedance pin damages the FPGA over time through its protection diodes.

Pulse electronics uses several ***logic families***; table 3 says how each connects to counter.

Family	Levels	To counter
LVCMOS, LVTTL (3.3 V)	0 / 3.3 V	directly to the E1 pins
TTL into 50 Ω	0 / 2–3 V into 50 Ω	to the E1 pins with a 50 Ω terminator at the board (most SPAD modules); check the high level (> 2.0 V) with an oscilloscope
TTL, 5 V CMOS	0 / 5 V into a high impedance	**not directly**: a 50 Ω terminator if the source drives 50 Ω (it halves to about 2.5 V), else a divider or a level translator
NIM (fast)	0 / −0.8 V into 50 Ω (−16 mA)	IN1 or IN2 (C9, C10) with a 50 Ω terminator, negative polarity, threshold about −0.4 V; or a NIM-to-TTL converter to the pins
ECL, PECL, LVDS	differential	a converter to LVCMOS
Analog pulses (PMT, SNSPD, APD)	tens of mV to 1 V	IN1, IN2 with a threshold (section 5); or a discriminator/comparator to the pins

Table 3: Logic families and how they connect.

NIM (Nuclear Instrumentation Module, a standard of nuclear physics electronics, U.S. NIM

Committee [3]) logic is a current: a logic 1 is −16 mA, −0.8 V across a 50 Ω termination, a logic 0 is 0 V. Its pulses are short (a few ns) and fast, made for 50 Ω cables. ***TTL*** (transistor-transistor logic) originally meant 0 and 5 V; "TTL output" on a detector's data sheet usually means a positive pulse of 2–5 V, often specified into 50 Ω. Always look at the actual pulse with an oscilloscope, terminated as it will be in the experiment, before connecting it to the board.

3.3 Detectors and their pulses

SPAD modules (single-photon avalanche diode in a module, "SPCM"). A logic pulse per detection, typically 2.2 V (some models 4.4 V) into 50 Ω, 10–30 ns wide, after a dead time of about 20–50 ns in which the module cannot click again. Connect through 50 Ω coax with a 50 Ω terminator at the board (a BNC T with a 50 Ω cap, or a feed-through terminator, then a short adapter to the header). A 4.4 V module needs an attenuator (a 6 dB 50 Ω attenuator gives 2.2 V). Set the channel's dead time to the module's or a little more (section 4.2). counter has not been tested with a detector: confirm the pulse level and shape on an oscilloscope.

Photomultipliers (PMT) give a negative current pulse per photoelectron, a few ns long, with a height that varies from pulse to pulse. A preamplifier and a discriminator make NIM or TTL pulses of them; with a preamplifier alone, IN1 or IN2 can serve as the discriminator, for pulses that stay a few ns wide after the board's 60 MHz bandwidth (section 5).

Superconducting-nanowire detectors (SNSPD) give, after their cryogenic or room- temperature amplifier, pulses of a few hundred mV with a fast leading edge (below 1 ns) and a slow decay (tens of ns), positive or negative. IN1 and IN2 take them directly (threshold about half the amplitude); for 1 ns timing on the pins a fast comparator is needed.

Other sources Any 3.3 V logic edge counts: a photodiode with a comparator, a laser's sync output (through a level translator), the outputs of another instrument.

Section 19 explains what these detectors do to the statistics of the counts: efficiency, dark counts, afterpulses and dead time.

3.4 Cables, termination and crosstalk

A signal needs about 5 ns per metre of coax (the speed of light times a ***velocity factor*** of about 0.66). On a cable longer than a few tens of centimetres a fast edge behaves as a wave: at an end whose impedance Z differs from the cable's $Z_0 = 50\,\Omega$, a fraction $\Gamma = (Z - Z_0)/(Z + Z_0)$ of it reflects. The E1 pins are a high impedance ($\Gamma \approx 1$), so a pulse arriving on unterminated coax reflects back to the source and, if the source does not absorb it, returns as a second, delayed pulse: a ***double count***. Two rules prevent it:

- a source made to drive 50 Ω (SPAD modules, NIM, most pulse generators' 50 Ω outputs) is ***terminated*** at the receiving end: a 50 Ω resistor to ground at the board;
- a source that cannot drive 50 Ω (counter's own outputs, most logic) is ***source-terminated***: a resistor of about 33–47 Ω in series at the source, a high-impedance receiver, and nothing else on the line.

Short wires (below about 20 cm) need neither. The header's pins invite jumper wires, which are fine for slow signals but couple to each other. In the session with a second board (section 16.3), 20 cm jumpers carrying 8 ns pulses beside the TRIG wire made about 2 spurious TRIG edges per million TRIGs; each one advanced the point and flagged its sweep. The cures: coax, a ground wire twisted with each signal, pulse wires kept apart from TRIG, and TRIG's hold-off set close to the shortest shot spacing (section 7.2), which removed them all. Lines that idle high are not immune. In the later tests TRIG and SEQ idled high and triggered on falling edges. Wherever the two pulse lines rose together (in the same 8 ns word), a false falling edge could follow about 1 ns later, at two places of a random pattern in 1–4 % of its passes (section 16.4). counter's own records found both: a false

SEQ as bad sweeps, a false TRIG by its time in the SHOT records. Any input that samples every nanosecond can see such spikes (timetagger's 1 ns markers did, in the same bundle); counter's count lines in these tests stayed exact, since they were the lines that switched. A layout with each fast signal on its own coax, or twisted with its own ground wire, avoids it.

Grounds

Join the grounds of the counter, the detectors and the pulse generator before connecting any signal (E1 pins 25, 26, or the SMA shells), and power them from the same mains outlet strip where possible. A detector module on a separate ground can sit volts away from the board.

3.5 The analog inputs IN1 and IN2

IN1 and IN2 are the board's SMA inputs, digitized at 125 million samples per second with 14 bits. Their input impedance is 1 MΩ in parallel with 10 pF; a jumper selects the range, **LV** (±1 V nominal, ±1.13 V on the board used here) or **HV** (±20 V nominal); their bandwidth is about 60 MHz. For NIM or other 50 Ω sources add a 50 Ω terminator at the input (a T and a 50 Ω cap). counter uses them as the count channels C9 and C10: a discriminator in the FPGA at a threshold in volts (section 5); the board calibration converts volts to the converters' codes. The inputs' noise is about 1.5 mV rms on the LV range (measured, idle input), far below any useful threshold. The 60 MHz bandwidth smooths nanosecond pulses: two pulses less than about 16 ns apart are counted once (measured, section 5).

3.6 The analog outputs OUT1 and OUT2

OUT1 and OUT2 (SMA, ±1 V, 50 Ω source impedance) show the rate of chosen channels as a voltage (a ***rate monitor***), or the test emitter's pulses as analog pulses of a set amplitude (the test signal of the analog channels), or 0 V (section 6.3). The voltages are calibrated at the connector. After loading the design, and after `SAFE`, both are at 0 V.

3.7 The outputs O1–O4

O1–O4 drive 3.3 V logic (LVCMOS33, 12 mA, fast edges) with a time resolution of 2 ns (section 10). They are meant for high-impedance inputs: another board's pins, an oscilloscope, a logic gate, an acousto-optic modulator driver's TTL input, a microwave switch's control input. A 50 Ω load pulls the high level far below 2 V: put a line driver between, or source-terminate a coax and keep the far end high-impedance. A NIM input needs a TTL-to-NIM converter. In the session with a second board, O1 drove awg's event input and O4 drove C8 through short jumpers without a miss in millions of pulses (section 16.3).

3.8 Pulse generators

The control inputs TRIG, GATE, NEXT and SEQ take the same levels as the count inputs. A pulse should be at least 2 ns wide (shorter ones can fall between two samples) and at least 8 ns; a shot needs at least 64 ns (section 7). A 5 V TTL pulse generator needs the same care as a 5 V detector. With awg on a second Red Pitaya, its digital outputs (3.3 V) wire directly to counter's control and count inputs; the receipts of that session (section 16.3) show 39075760 shots counted without an error.

Two instruments, two outputs

Connect outputs only to inputs. counter's O1–O4 (DIO4_N–DIO7_N) must never be wired to another output; on another Red Pitaya, which pins are outputs depends on its design (awg drives its DIO0_N–DIO7_N and two of its _P pins). Load both designs before connecting the wires, and check every wire with static levels first: drive one end high and read the other (`dev.pins()`).

4 Count channels C1–C8

A ***channel*** turns the pulses on one input into ***events***: one event per counted edge, with its time to 1 ns. Every function downstream (totals, gates, coincidences, echoes) works on the channels' events. C1–C8 are the pins DIO0_P–DIO7_P; C9 and C10, the analog inputs, are channels too (section 5). Each has four settings:

```
dev.apply({"channels": {"1": {"enable": True,          # off: no events at all
                              "edge": "rising",        # or "falling"
                              "dead_time": 50e-9,      # 8 ns ... 65.535 us, 1 ns steps
                              "delay": 12e-9}}})       # 0 ... 255 ns, 1 ns steps
```

4.1 Sampling every nanosecond

The pins are not watched by a comparator with a time-to-digital converter, as in a time tagger, but ***sampled***: an input deserializer of the FPGA (an ISERDESE2 in ***oversampling*** mode, clocked at 250 MHz on two phases 90° apart) takes the pin's logic level every nanosecond, like a camera taking a frame every nanosecond. Each 8 ns clock period yields a ***word*** of 8 samples, the ***slots*** 0 (earliest) to 7. A rising edge is a 0 in one sample followed by a 1 in the next; the event gets the slot of the 1. So

- an edge's time is known to the nanosecond in which it happened: the timing error is spread evenly over 1 ns (±0.5 ns, 0.29 ns rms), plus the jitter of the edge itself;
- a pulse at least 2 ns wide is always seen; a pulse narrower than 1 ns can fall between two samples and be missed;
- the gap between two pulses must also be about 2 ns or more for them to be two edges.

Choose `edge falling` for detectors whose leading edge falls (inverted logic) or to count the trailing edges of pulses. The current levels of all inputs, and whether an edge came since the last look, are a quick check of the wiring:

```
print(dev.pins(clear=True))   # {"C1": {"pin": "DIO0_P", "level": False, "seen": False}, ...}
```

TCP: `PINS? CLEAR`

4.2 Dead time

The ***dead time*** τ of a channel is the time after a counted event during which it takes no other. counter's dead time is ***non-paralyzable***: an edge inside the dead time is rejected (and counted as ***rejected***) but does not extend the dead time, which always ends τ after the last counted event. (A ***paralyzable*** dead time, as in some detectors, restarts with every edge, so that at very high rates the counter stops counting altogether.) The setting runs from 8 ns to 65.535 µs in 1 ns steps; 8 ns, one word, is the least, so that a channel never has two events in one word (section 22).

Three reasons to set a dead time longer than 8 ns:

- **to match the detector.** Every detector has a dead time of its own (20–50 ns for SPAD modules). If the channel's dead time is at least as long and non-paralyzable, the counted rate follows one known formula and can be corrected exactly (below);
- **to suppress afterpulses**, false clicks that follow a real one within tens to hundreds of nanoseconds (section 19.2);
- **to suppress double counts** from ringing or reflections on the cable (section 3.4).

At a true rate R of independent (Poisson) events, a non-paralyzable dead time τ counts

$$m = \frac{R}{1 + R\tau}, \qquad R = \frac{m}{1 - m\tau}, \tag{1}$$

the second form being the ***dead-time correction*** (`analysis.dead_time_corrected(m, tau)`). At $m\tau = 0.1$, a tenth of the events is lost. Figure 4 shows the counted rate of the test emitter through

four dead times: at 10 Mevents/s C1 counts 0.966, 0.865, 0.687 and 0.511 of the events with 8, 20, 50 and 100 ns. The measured points follow a simulation of the emitter's events (lines) exactly; equation (1) (dotted) describes a Poisson source, and the emitter, which makes at most one event per word, is not quite one at high rates (section 13). Section 18.3 discusses dead time and the statistics of the counts.

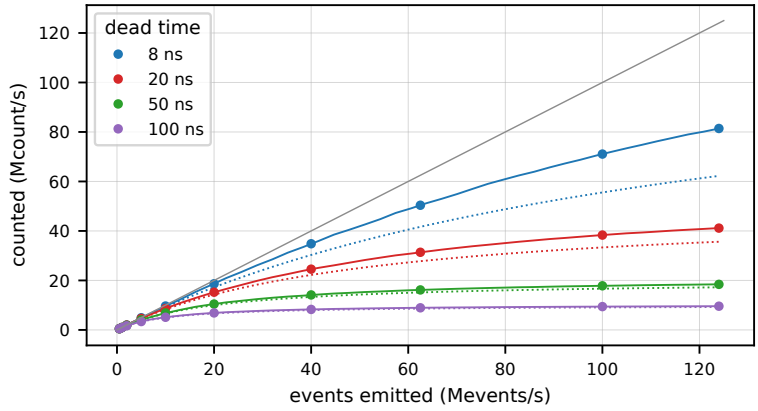


Figure 4: Measured (2026-11-05). The counted rate of C1 against the test emitter's rate, for dead times of 8, 20, 50 and 100 ns (points); lines: a simulation of the emitter's events (one trial per 8 ns word at a random slot, 2 ns pulses) through the same dead time; dotted: equation (1) for a Poisson source; grey: no loss. At 124 Mevents/s and 8 ns the channel counts 81.4 Mcount/s: pulses of the emitter in neighbouring words merge.

The emitter is a source of known statistics; awg's patterns are a source of known events. In the tests of section 16.4, awg played random patterns on C1 (every event known) at 1.27–28 Mevents/s through dead times of 20 ns to 1 µs. In all 25 runs the counted and rejected edges were exactly what the pattern gives. The correction of equation (1) recovers the true rate within 3 % at 1.27 Mevents/s (the rest is the one pattern's own fluctuations) but over-corrects by about 14 % at 11.3 and 48 % at 28 Mevents/s (figure 5). awg's events sit on an 8 ns grid, and neighbouring pulses merge, so they are not a Poisson process, which the correction assumes. Real sources can depart from it too: a pulsed laser's photons come only in its pulses, a detector's afterpulses follow its counts, and light can be bunched or antibunched. Keep $m\tau$ small where the correction matters, or correct with a model of the actual process (the simulations behind figure 4 are one).

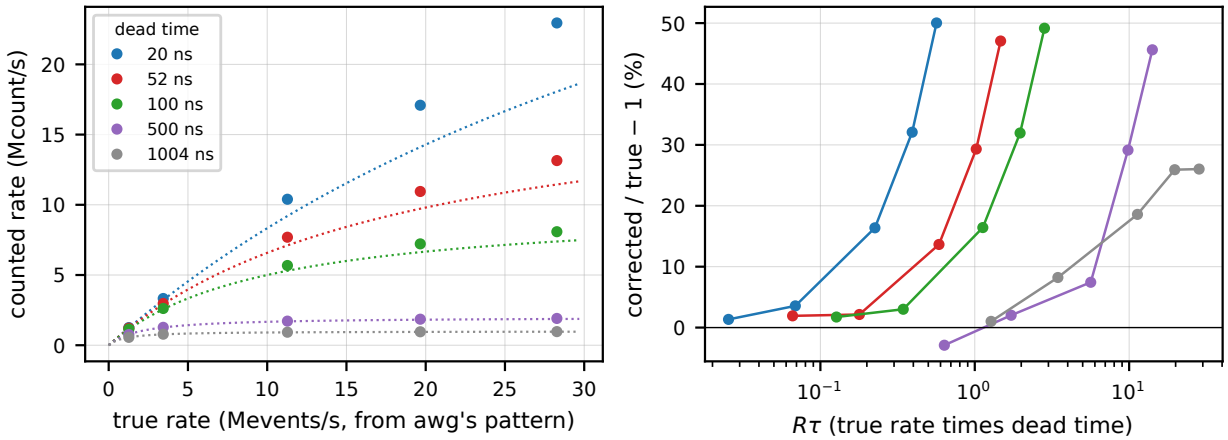


Figure 5: Measured (2026-11-05). Dead time against a known rate: awg's random patterns on C1. Left: counted rate (points) and equation (1) (dotted) against the true rate, for five dead times. Right: the corrected rate's error against $R\tau$.

4.3 Delays

Each channel's events can be delayed by 0 to 255 ns in 1 ns steps. Detectors, amplifiers and cables delay their pulses by different amounts (a metre of coax is 5 ns, an SNSPD's amplifier chain tens of ns); a delay per channel lines them up, so that photons that arrive together at the detectors are counted together: in a coincidence window, or in the same gate. To find the right delay, count the coincidences of two channels against a delay scanned with unit 2 (section 11.2): the peak of correlated photons sits at the relative delay to correct. While C9 or C10 is on, the analog channels' lag (95 ns on the board used here) is added to every pin channel's delay, so that the maximum left for the setting is 255 ns minus that lag (section 14.2).

4.4 Totals and rejected edges

Every channel has a ***total***: a 48-bit count of its events since the design was loaded, never cleared (at 125 Mcount/s it would take 26 days to wrap). Rates, RATE records and every other count are differences of totals, so nothing is ever lost by reading late. The ***rejected*** count (32 bits, wrapping) counts the edges that fell into the dead time: with the detector's own dead time shorter than the channel's, it shows how many edges the channel threw away.

```
c = dev.counters()
print(c["time_s"], c["channels"]["1"]["total"], c["channels"]["1"]["rejected"])
```

TCP: `COUNTERS?`

How it is verified

`make test-hw` compares the counted and rejected events of C1–C4 with a simulation of the emitter's process at 2.5–62.5 Mevents/s and dead times up to 100 ns, within the statistical error. With a jumper from an output to C8, C8 counts a 125 MHz pattern (one edge every 8 ns) exactly, 1 250 000 edges in every 10 ms, on rising and falling edges, and one third of them with a 20 ns dead time. With awg on a second board, C1 and C2 counted 351681840 pulses of known numbers per shot without an error (section 16.3), and the counted and rejected edges of random patterns through dead times of 20–1004 ns were exact in all 25 runs (section 16.4).

5 Analog channels C9 and C10

IN1 and IN2 are count channels too: C9 and C10. Their input is not a logic level but a voltage, sampled every 8 ns by the board's analog-to-digital converters, and a ***discriminator*** in the FPGA decides when a pulse has come: when the voltage crosses a ***threshold***. They take what the logic pins cannot: NIM pulses (−0.8 V), the amplified pulses of SNSPDs and photomultipliers, any analog signal whose crossings are to be counted.

```
dev.apply({"channels": {"9": {"enable": True,
                              "polarity": "negative",   # NIM: negative pulses
                              "threshold": -0.4,        # V at IN1
                              "hysteresis": 0.1,        # V: re-arms only above -0.3 V
                              "dead_time": 20e-9, "delay": 0.0}}})
```

5.1 The discriminator

The discriminator is a ***Schmitt trigger***: a comparator with two levels. For positive pulses it is ***armed*** when a sample is at or below $T - H$ (the threshold T less the ***hysteresis*** H), and fires at the next sample at or above T, which disarms it until a sample again falls to $T - H$. Noise or ringing around T thus cannot fire it twice: the voltage must come back below $T - H$ first. For negative pulses the samples are inverted first: it fires at or below T and re-arms at or above $T + H$. The threshold and hysteresis are in volts at the connector, converted through the board calibration and the input's range (LV or HV jumper); a threshold outside the range is refused, `None` disables firing. The hysteresis is at least one converter step (about 0.14 mV on the LV range).

A pulse fires between two samples 8 ns apart. counter ***interpolates*** the crossing: the time where the straight line between the two samples reaches T, to the nanosecond, so that C9 and C10 events have 1 ns slots like the pins' (section 22 has the arithmetic). The event then passes the same dead time and delay as a pin channel's. Since a sample must re-arm the trigger, two events are at least two words (16 ns) apart on an analog channel.

5.2 Choosing the threshold

Look at the input first: `dev.analog()` reads the input's samples many times and returns their mean, extremes and standard deviation in volts.

```
print(dev.analog(4096)["C9"])    # {"mean": -0.0006, "min": -0.0059, "max": 0.0037, "std": 0.0015}
```

TCP: `ANALOG? 4096`

An idle LV input has about 1.5 mV rms of noise; a threshold of 50 mV is already 30 standard deviations away and never fires on noise (measured). The threshold should sit well above the noise and the baseline's wander, and well below the pulses: in the middle of the ***plateau*** of the counts against the threshold. Figure 6 shows the plateau for the 8 ns, 0.8 V test pulses of OUT1 cabled to IN1: one count per pulse (0.9987 on average) for every threshold from 0.10 to 0.56 V. Below it the pulses' ringing crosses the threshold several times (4.0 counts per pulse at 20 mV, with a hysteresis of 20 mV); above it nothing fires: the converters' 60 MHz bandwidth leaves an 8 ns pulse of 0.8 V with a peak of about 0.58 V in the samples. With a real detector, scan the threshold the same way, with the light on and off: the plateau of the light's counts above the dark counts is where the threshold belongs (section 19.3).

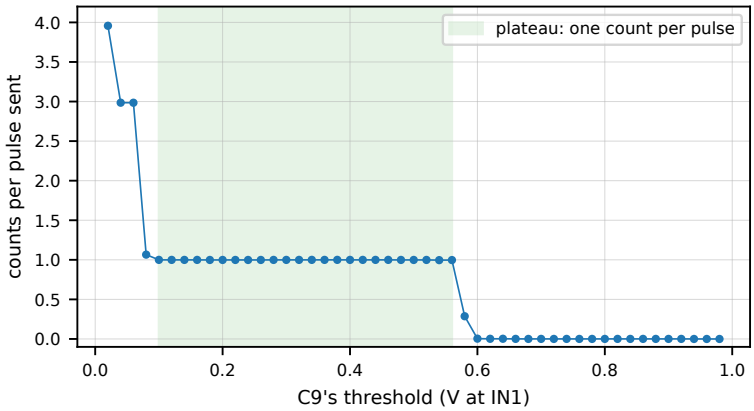


Figure 6: Measured (2026-11-05). Counts of C9 per pulse of OUT1 (8 ns, 0.8 V, 0.1 million per second, OUT1 cabled to IN1) against the threshold, hysteresis 20 mV. Green: the plateau, one count per pulse.

5.3 Pulse shape, pair resolution and timing

The analog front end and the converters' 8 ns sampling set the limits:

- **pair resolution**: two pulses less than about 16 ns apart merge into one. With OUT1's pulses at random times, C9 counted 99.84 % of them at 0.1 million per second and 98.4 % at 1 million per second (measured): the losses are the pairs closer than about 16 ns;
- **timing**: the interpolated crossing is a ***leading-edge*** time: a larger pulse crosses earlier than a smaller one of the same shape (***time walk***). For pulses of steady height (NIM, an amplified SNSPD) it is a constant delay; for photomultiplier pulses of varying height it spreads the times by a fraction of the rise time;
- **measured with awg's pulses** (figure 7): pulses of 0.18–0.81 V with a rise of about 16 ns crossed a 0.08 V threshold over a span of 8.2 ns, the smallest last. At 0.15 and 0.17 V the

span grew to 11.8 and 13.6 ns, the larger as the smallest pulse approaches the threshold. A constant-fraction timing, or a low threshold on fast pulses, keeps the walk small;

- **alignment**: the converters and their filters delay the analog channels by about a tenth of a microsecond against the pins. While C9 or C10 is on, counter adds that delay (Δ, 95 ns on the board used here) to the pin channels, the gates and GATE, so that a pulse on IN1 and an edge on a pin that reach their connectors together are counted together (section 14.2, figure 18).

5.4 A detector simulated by awg

With awg's OUT1 cabled to IN1, pulses of known heights test the discriminator as a detector would (section 16.5). Its digital output marked every pulse on C1, so that counts per pulse are exact ratios:

- **four fixed heights** gave a staircase of exactly 1, $\frac{3}{4}$, $\frac{1}{2}$, $\frac{1}{4}$ and 0 counts per pulse against the threshold (figure 7, left). Each step sits at the height that the samples catch, 1.06 times awg's nominal one, and is a few tens of millivolts wide: the sampling phase decides how close a sample comes to the peak;
- **a photomultiplier-like spectrum** (2048 pulses per pass, 69 % single photoelectrons around 0.45 V, the rest small noise pulses) gave the classic ***plateau curve*** (centre). The counts fall steeply while the threshold cuts into the noise pulses, then slowly across the plateau (0.705 per pulse at 0.16 V for 0.694 single photoelectrons sent), and steeply again through the photoelectron peak. The curve's negative derivative is the ***pulse-height spectrum***: here it shows the peak. Set the threshold at the plateau's start;
- **NIM-like pulses** of −0.75 V at −0.40 V with negative polarity: 1 972 184 counted of 1 972 194 (the 10 missing are the reading's: two channels' totals are read some microseconds apart, section 16.5; SHOT records showed every one of 5.9 million counted). A *positive* threshold of 0.05 V counted 1.00000 per pulse: the overshoot after each negative pulse. Set the polarity to the pulses' sign, and look at a new detector's pulse on an oscilloscope for overshoot and ringing before choosing.

C9 timed awg's analog pulses about 33.5 ns after the digital marks that awg emits with them. counter's alignment of C9 with the pins was checked to 0.5 ns, so the difference is awg's own analog latency against its digital outputs on a board with its nominal calibration, plus the cable.

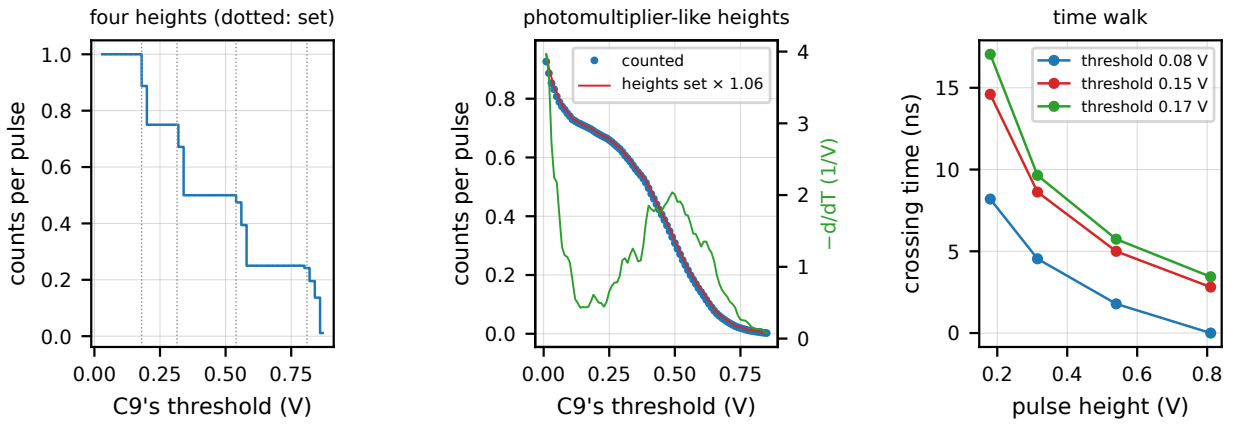


Figure 7: Measured (2026-11-06). awg's OUT1 as a detector on IN1 (C9). Left: four pulse heights, counts per pulse against the threshold (dotted: the heights set). Centre: a photomultiplier-like spectrum: counts per pulse (points), the heights set times the step's gain (line), and the negative derivative (green), the pulse-height spectrum. Right: the time walk, the crossing time against the pulse height, for three thresholds.

5.5 Counting a reference frequency

A sine wave crosses zero once per period upwards: C10 with threshold 0 V and a hysteresis of 0.2 V counts a 10 MHz reference on IN2 once per period. Over 20 s of RATE records every 10 ms (section 12) it gave 10000056.6 Hz: the reference, from a GNSS-disciplined oscillator, reads 5.66 ppm high, so the board's crystal runs that much slow (measured). This is the simplest check of counter's timebase, to which every rate and time refers; Section 14.5 discusses it.

```
dev.apply({"channels": {"10": {"enable": True, "threshold": 0.0, "hysteresis": 0.2}}})
print(dev.rates(1.0)["channels"]["10"])     # about 1e7 (+ the crystal's error)
```

How it is verified

`make test-hw`: C9 and C10 count the emitter's pulses through their dead time as simulated; the idle C9 gives no count with a threshold 50 mV beyond the noise and the expected noise counts at it, for both polarities; OUT1's pulses on IN1 are counted (99 % at least at 0.1 million per second); a 10 MHz sine on IN2 gives one count per period in every 10 ms interval. The bench tests of the RTL compare the discriminators' events, clock by clock, with the bit-exact model, including crafted crossings at every slot. The alignment was measured with an oscilloscope (section 14.2).

6 Rates, RATE records, monitors and the alarm

Counting without shots, gates or a trigger is the everyday mode of a photon counter: the fluorescence of a sample while the optics are aligned, the dark counts of a detector, the singles and coincidences of a photon-pair source. counter offers four views of the same totals: a rate on request, a stream of RATE records at a fixed interval, a voltage on OUT1 or OUT2, and an alarm.

6.1 Rates on request

`dev.rates(seconds)` reads all totals twice, `seconds` apart, with the core's clock: the interval is the clock's, not the PC's, so the rate is exact even if the network delays a reading.

```
r = dev.rates(1.0)
print(r["interval_s"], r["channels"]["1"], r["channels"]["9"])   # counts per second
```

TCP: `RATES? 1.0`

Its precision is that of the counts: $\sqrt{N}$ out of N (section 18).

6.2 RATE records

For time series (blinking emitters, quantum jumps, the drift of a source, the loss of an atom) counter writes a ***RATE record*** every `interval`, from 10 µs to 10 s (in steps of 8 ns), into the record stream (section 12). Each holds the low 32 bits of all ten totals, and the 15 pattern totals of each coincidence unit that is on. The totals are latched at fixed offsets after the interval's end, one per clock, so that **consecutive records differ by exactly one interval in every channel and every pattern**: no event is lost or counted twice between records, and the rates are exact.

```
from counter import RecordStream, analysis, model
dev.apply({"rates": {"enable": True, "interval": 1e-3}})
words = []
with RecordStream("192.168.0.102") as stream:
    for frame in stream.frames(timeout=2.0):
        words += analysis.parse_frame(frame)[1].tolist()
        if len(words) > 13 * 3000:
            break
tab = analysis.rate_table(model.decode(words))
print(tab["end_s"][:3], tab["rates"][:3, 0])     # each interval's end (s) and C1's rate
```

At 10 µs the board makes 100 000 records per second; they reached a PC without a loss (200559 consecutive intervals in each run of figure 21). The records also carry the rate alarm's state.

6.3 Rate monitors on OUT1 and OUT2

A ***rate monitor*** turns the rate of a set of channels into a voltage: in windows of 2^k words ($k = 4 \ldots 24$: 128 ns to 134 ms) it counts their events and, after each window, sets the output to

$$V = \text{offset} + \text{scale} \times \text{rate}, \tag{2}$$

calibrated at the connector and clipped to ±1 V. It is the fastest feedback for a human: a voltmeter, an oscilloscope, or an audio amplifier whose pitch follows the rate while a mirror is turned. It is also an analog signal for other instruments: a chart recorder, an analog input of a laser controller, or a lock box that holds the count rate at its maximum by dithering a piezo (a "peak lock").

```
dev.apply({"monitors": {"1": {"mode": "rate", "channels": [1, 2],   # C1 + C2
                              "scale": 1e-7,                        # 0.1 V per Mcount/s
                              "window": 2**17 / 125e6,              # 1.05 ms
                              "offset": -0.5}}})                    # use the whole range
```

Figure 8 shows OUT1 cabled to IN1, read back by counter itself: the slope is 0.0966 V per Mcount/s for a set 0.1 (−3.4 %: the calibrations of OUT1 and IN1 together), the offset −0.5 mV. A longer window averages more counts (less noise, slower response): the relative noise of each step is $1/\sqrt{N}$ for N counts in the window.

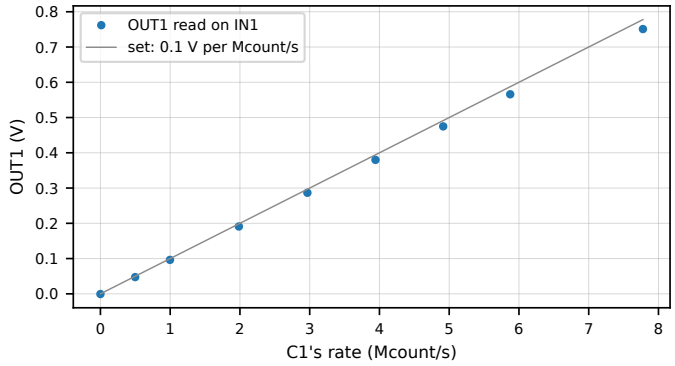


Figure 8: Measured (2026-11-05). OUT1's rate monitor (C1's rate, set to 0.1 V per Mcount/s, 1 ms windows) read on IN1 through a cable, against C1's rate from the test emitter.

In `pulses` mode the output instead gives a pulse of `amplitude` volts for 8 ns at each event of a test-emitter lane: the test signal of the analog channels (section 13). `off` sets the output to `offset` (0 V by default).

6.4 The rate alarm

The ***rate alarm*** watches a set of channels in windows of 1 µs to 2 s and is on while their count in the last window was below `below` × `window`: an ion that has left its trap (its fluorescence gone), an atom lost from a tweezer, a laser that has unlocked, a fibre that has moved. It shows in the status, in every RATE record and on an output pin (section 10), where it can stop a sequencer or light a lamp.

```
dev.apply({"alarm": {"enable": True, "channels": [1], "window": 1e-3, "below": 1e6},
           "outputs": {"3": {"mode": "alarm"}}})          # O3 high while C1 < 1 Mcount/s
print(dev.counters()["alarm"])                             # {"on": False, "last_count": 4012}
```

Figure 9 shows it on a time series of RATE records: the rate drops for a second and the alarm follows within a window.

How it is verified

`make test-hw`: RATE records every 10 ms, none missing, every interval exactly 1 250 000 words, their counts summing to the totals; every 10 µs for 2 s without a lost byte or a dropped record; the alarm on below 3 Mcount/s and off below 1 Mcount/s for a rate of 2.5 Mcount/s, its last window count as expected; OUT1's monitor code as the measured rate gives; OUT2 off at the calibrated 0 V.

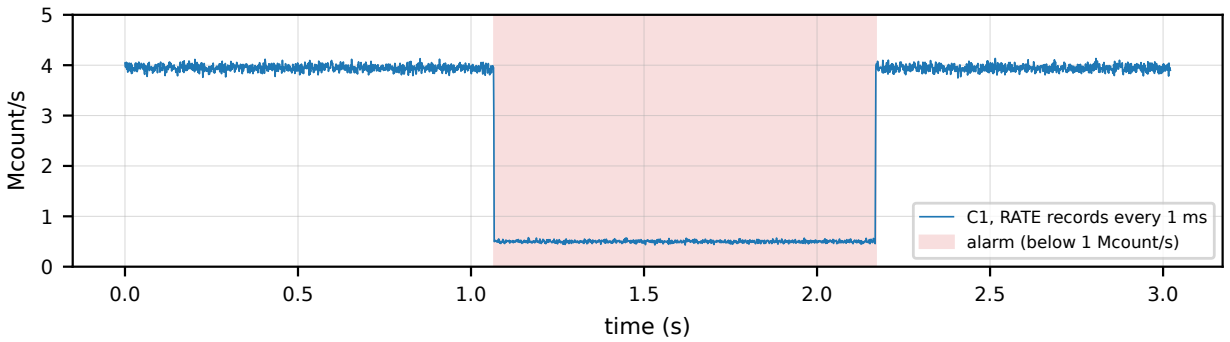


Figure 9: Measured (2026-11-05). C1's rate in RATE records every 1 ms (3022 records) while the test emitter steps from 4 to 0.5 and back to 4 Mcount/s; shaded: the alarm (below 1 Mcount/s in 1 ms windows), as carried by the records.

7 Shots, points, sweeps and the per-point sums

A pulsed experiment repeats a ***shot***: prepare, evolve, read out. counter follows the sequencer through three inputs (TRIG, NEXT, SEQ) and keeps the counts of every ***point*** of a scan apart. This section describes the shots, the points and sweeps that organize them, and the sums in which their counts are collected; the gates that do the counting are the next section's.

7.1 Shots and the trigger

A shot starts at a ***TRIG***: a rising (or falling) edge on DIO0_N, a tick of the internal timer, or a software command (`shots.trigger` `pin`, `timer` or `software`). The TRIG's time, to the nanosecond, is the time zero of the shot's gates. A shot is ***busy*** until its last gate has closed, and at least 64 ns (eight words); a TRIG that comes while a shot is busy is ***missed***: counted, flagged in the busy shot's record, and not started. Missed TRIGs mean that the gates are longer than the shot spacing, or that the TRIG line bounces.

```
dev.apply({"shots": {"trigger": "timer", "timer_period": 10e-6,   # a TRIG every 10 us
                     "timer_seq": True, "points": 16}})            # and a SEQ every 16 TRIGs
dev.apply({"shots": {"trigger": "software"}})
dev.trigger()                                                      # one TRIG now
```

TCP: `TRIG`

The internal timer (64 ns to 34 s in 8 ns steps) makes self-contained experiments: a rate measurement in fixed time bins, the test emitter's rehearsals, or a sequence that a slower device follows. With `timer_seq` it also makes a SEQ with every P-th TRIG, so that its sweeps are complete without any wiring.

7.2 Hold-off

TRIG, NEXT and SEQ each have a ***hold-off***: an edge within the hold-off after the previous accepted edge of the same input is ignored. It works like a channel's dead time and suppresses bounces, ringing and the crosstalk of neighbouring wires; it can be set from 8 ns to 65.535 µs (default 24 ns). Set it just below the shortest time between two real edges: in the session of section 16.3, 450 ns removed every spurious TRIG that 20 cm of jumper wire beside a pulse line had induced.

```
dev.apply({"controls": {"trig": {"edge": "rising", "holdoff": 450e-9},
                        "next": {"edge": "rising", "holdoff": 24e-9},
                        "seq":  {"edge": "rising", "holdoff": 24e-9}}})
```

A hold-off has a blind side. In the tests of section 16.4, a false TRIG edge from crosstalk 463 ns *before* a real one started the shot early. The real TRIG then fell into the 3 µs hold-off and was ignored: it was not counted as missed, and in 4 % of the sweeps that shot's gates sat 463 ns early. The status showed nothing. The SHOT records did, because each carries its TRIG's time, and a spacing that differs from the sequencer's period stands out. With a 200 ns hold-off the real TRIG came as a missed one instead, but other false edges then came through as well. So the hold-off is no cure for a noisy TRIG line: fix the wiring (section 3.4), keep the hold-off just below the shortest real TRIG spacing against ringing, and check a new setup's SHOT records once for irregular TRIG spacings and its SWEEP records for bad sweeps.

7.3 Runs

Shots are taken only while a ***run*** is on. `dev.run()` clears the per-point results and starts a run at once, or, with `shots.start_at_seq`, at the next SEQ, so that the first sweep is complete; `dev.run(clear=False)` keeps the results. A run stops by `dev.stop()` (a shot in progress completes), after `shot_limit` shots or after `duration` seconds. Each start and stop writes a RUN record (section 12).

```
dev.apply({"shots": {"start_at_seq": True, "shot_limit": 0, "duration": 600.0}})  # 10 minutes
dev.run()
print(dev.run_status())   # {"state": "armed" | "running" | "idle", "point", "shots", "missed",
                          #  "sweeps", "bad_sweeps", "sweep", "swaps", "bank", ...}
```

TCP: `RUN`
TCP: `RUN?`
TCP: `STOP`

7.4 Points and sweeps

Each shot belongs to a ***point*** (0 to $P-1$, $P \leq 1024$), latched at its TRIG. The point follows the sequencer in one of three modes (`point_mode`):

`per-shot` the point advances after every TRIG (a missed one too), and SEQ resets it to 0. One shot per value of the scanned parameter: an NV Rabi sweep plays the pulse lengths $\tau_0, \tau_1, \ldots$ in order, a SEQ at the start of each sweep.

`per-next` NEXT (`DIO2_N`) advances the point, SEQ resets it. N shots per point: an ion experiment repeats each setting 100 times, then moves on; a confocal scanner's pixel clock on NEXT and its frame start on SEQ make the points an image's pixels.

`fixed` the point stays at `fixed_point`: all shots in one point.

After $P-1$ comes 0. Events in the same word take effect in the order of their slots; at the same slot NEXT comes first, then SEQ, then TRIG, so a marker that starts a sweep and its first shot at the same instant gives point 0. "The same instant" means the same nanosecond slot. Edges that a sequencer sends in one 8 ns word reach the inputs a few tens of picoseconds apart, and now and then they straddle a slot boundary. With awg sending SEQ and the first TRIG in one word, 1.1 % of the sweeps (falling edges; 1.5 % with rising edges) saw TRIG one slot before SEQ: that TRIG went to the old sweep, and both sweeps were flagged. NEXT sent in a TRIG's word was always taken first (section 16.5). Send SEQ and NEXT a word (8 ns) before the TRIG that belongs to them.

A ***sweep*** runs from one SEQ to the next. In per-shot mode counter checks every sweep: it must have exactly P TRIGs (missed ones included). A lost TRIG (a glitch on the line, a sequencer that skipped a step) or an extra one would shift every following point of the sweep, and the sweep is flagged ***bad***: in its SWEEP record and in `run_status()["bad_sweeps"]`. With awg's patterns

driving TRIG and SEQ, 1 222 473 sweeps with one TRIG left out on purpose were all flagged, and 4.9 million correct ones none (section 16.3).

```
dev.apply({"shots": {"trigger": "pin", "points": 51, "point_mode": "per-shot"}})
dev.apply({"shots": {"point_mode": "per-next"}})
dev.next_point(); dev.sequence()          # a software NEXT and SEQ (NEXT, SEQ commands)
```

7.5 Per-point sums

For every point the FPGA adds, over its shots,

- the counts $n_1 \dots n_4$ of the four gates,
- the number of shots,
- the number of shots in which D1 and in which D2 was true (section 9),
- $\sum n_1^2$, the sum of G1's squared counts, from which the variance of G1's counts per point follows: $\mathrm{Var}(n_1) = \sum n_1^2/N - (\sum n_1/N)^2$ for N shots.

The sums live in two ***banks*** of block memory. One counts; the other is ***frozen***: the board's service reads it, adds it into 64-bit totals and releases it, and the FPGA zeroes it. The service asks for a ***swap*** every second; a swap happens between two shots, so no shot is split or lost, and automatically before any 32-bit sum could overflow. The totals never overflow: a run can last for days, and the PC can disconnect and come back. `dev.results()` returns the totals since the run's start; `now=True` first swaps, so that the latest shots are included.

```
r = dev.results(now=True)
print(r["shots"][:3], r["n1"][:3], r["d1"][:3], r["n1_sq"][:3])   # 64-bit totals per point
print(r["mean"]["n1"][:3])                                        # counts per shot
c = analysis.contrast(r["n1"], r["n2"])                           # NV: 1 - signal/reference
```

TCP: `RESULTS? NOW`

`swap_at_sweep` makes a swap at every sweep's end too, so that each frozen bank holds whole sweeps; the SWAP record tells at which sweep and point every swap happened. `dev.clear()` discards the results. A shot whose counts would overflow a 32-bit sum when no swap is possible is added saturated and flagged.

How it is verified

`make test-hw`: the means per point follow the emitter's table of bright probabilities (within $1.2\,\sigma$); every shot is in the sums (swaps every second, none missed, no bad sweep); software SEQs at random times flag the sweeps they cut. With awg's patterns (section 16.3): 39 million shots at 2 million per second, the sums of every point exact to the count. The RTL's bench `tb_sums` drives the sums near 2^{32}, through automatic swaps and saturation, clock by clock against the bit-exact model.

8 Gates

A ***gate*** is a time window of a shot in which counts are taken. Gating is what makes a photon counter useful in a pulsed experiment: the photons that carry the information arrive in a known window after the trigger, while dark counts, stray light and the photons of other parts of the sequence arrive at all times. Counting only in the window keeps the signal and rejects most of the background; a second window gives a reference, or the background itself to subtract.

8.1 What a gate counts

Each shot has four gates, G1–G4. Gate g of a shot whose TRIG came at t_T, in point i, covers the half-open interval

$$\left[\, t_T + d_g + i\, s_g,\;\; t_T + d_g + i\, s_g + w_g \,\right) \tag{3}$$

with a delay d_g and a width w_g (each 0 to 4.29 s in 1 ns steps) and a delay step s_g per point (0 to 16.7 ms). It counts the events of its ***channel set***: one or more of C1–C10, summed (two detectors behind a beam splitter, the four outputs of a polarization analyser). The interval is ***half-open***: an event at its start counts, one at its end belongs to the next gate, so **adjacent gates never count an event twice**; overlapping gates both count it. A width of 0 is an empty gate. A gate can start at its TRIG's own nanosecond (delay 0): TRIG reaches the shot logic two clocks before the channels' events of the same instant.

```
dev.apply({"gates": {"1": {"mode": "timed", "channels": [1, 2], "delay": 0.0, "width": 300e-9},
                     "2": {"mode": "timed", "channels": [1, 2], "delay": 2e-6, "width": 1e-6},
                     "3": {"mode": "off"}, "4": {"mode": "off"}}})
```

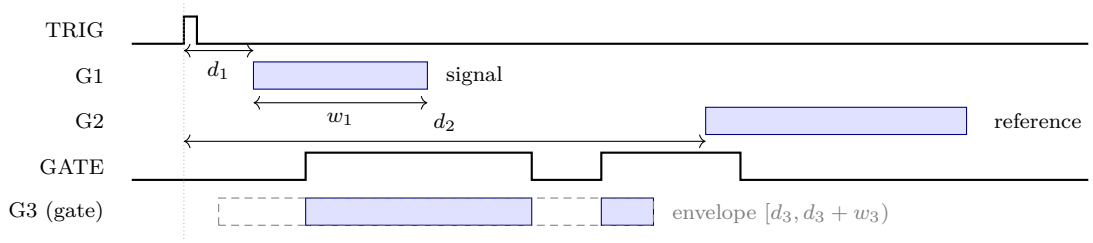


Figure 10: Gates of one shot. G1 and G2 are timed from TRIG: a signal window and a reference window. G3 is in `gate` mode: it counts only where its timed envelope (dashed) and the GATE input are both high.

8.2 The GATE input

A gate in `gate` mode counts only where its timed interval and the ***GATE*** level on DIO1_N are both high: the sequencer's own gate pulse, inside an envelope that limits it to the shot. GATE is active high or low; a `min_width` filter (1 to 8 ns) ignores pulses and gaps shorter than that, and a delay of 0 to 255 ns lines it up with the photons.

```
dev.apply({"controls": {"gate": {"active": "high", "min_width": 3e-9, "delay": 20e-9}},
           "gates": {"3": {"mode": "gate", "channels": [1], "delay": 0.0, "width": 10e-6}}})
```

This is the mode for experiments whose readout windows are not fixed after a trigger: a sequencer that marks each readout with a pulse, several readouts per shot, a lock-in-style chopped measurement in which GATE follows the chopper (section 8.3).

The filter delays the level by $W - 1$ ns (a level counts once it has lasted W ns), so the GATE window follows the pin by $(W - 1)$ ns plus the delay. With awg's GATE pulse of 80 ns and C1 pulses at its edges and 8 ns away, this shift held to the nanosecond for every combination tried (27 of 27 settings exact, section 16.5). With no shift, a C1 edge sent in the same word as GATE's rising edge was inside the window in 0.748 of the shots, one sent with its falling edge in 0.804: GATE's edges reached the board 0.25 ns and 0.80 ns after C1's, so the GATE pulse read 0.55 ns longer than sent. Place gate edges a few nanoseconds away from the photons that matter.

8.3 Signal and reference

Two gates of equal width, one where the signal is and one where only background is, give the signal by subtraction, $n_1 - n_2$; two gates where the signal is in different states give a normalized signal, n_1/n_2. NV-centre readout uses the second: G1 in the first few hundred nanoseconds of the laser pulse, where the fluorescence depends on the spin state, and G2 at the end of the pulse, where the spin has been repolarized and the fluorescence measures only the laser power and the collection efficiency; the ratio cancels their drifts (section 23.3). With chopped or modulated light, G1 in the "on" half and G2 in the "off" half of each period subtract the background exactly, the counting equivalent of a lock-in amplifier, as in the classic gated photon counters [4]. `analysis.contrast()` propagates the Poisson errors of such ratios (section 18.5).

8.4 The gate-delay scan: time traces without histograms

With a delay step s_g, the gate moves by s_g from point to point: in per-shot mode a sweep of P points scans a window of width w_g across $P\,s_g$ of the shot. The per-point sums then hold the ***time trace*** of the signal after the trigger: the fluorescence of an NV centre during its readout pulse, the decay of a quantum dot's emission after a laser pulse, a trapped ion's fluorescence over the phase of its trap's radio-frequency drive (the micromotion measurement of section 24.4). Figure 11 shows a trace of the test emitter: a 20 ns window stepped by 20 ns over 150 points.

```
dev.load_preset("nv-trace")     # G1: 20 ns at 0, 20, 40 ... 2980 ns after TRIG, 150 points
dev.run(); time.sleep(20); dev.stop()
r = dev.results(now=True)
t = np.arange(150) * 20e-9
fit = analysis.fit_exponential(t[5:], np.array(r["mean"]["n1"])[5:])
```

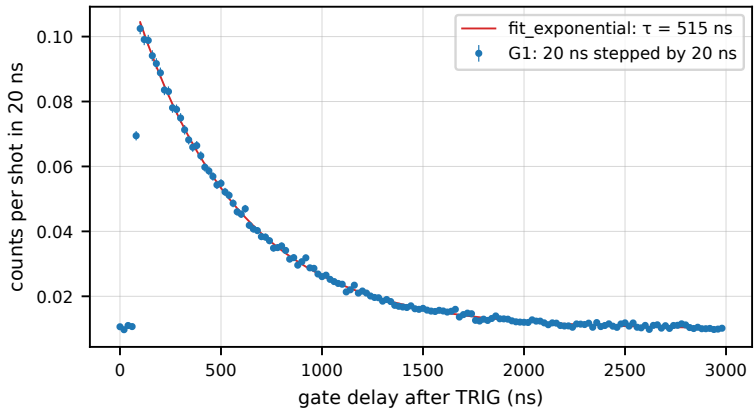


Figure 11: Measured (2026-11-05). A time trace by the gate-delay scan: G1, 20 ns wide, stepped by 20 ns per point, 33344 shots per point, the test emitter bright in every shot with its extra rate decaying with $\tau = 2^6 \times 8$ ns; fit: $\tau = 515 \pm 4$ ns. The emitter's bright state starts 88 ns after TRIG (its own latency): before it, the background of 0.0104 counts per window.

A time trace by the gate-delay scan uses each shot for one window: it collects $1/P$ of the photons a histogram of every photon's time would. That is the price of needing no time tagger and no histogram memory, and it is small when the shots are cheap (an NV readout repeats every few microseconds) or when only a few windows matter. For a detailed lifetime or a fluorescence transient with all photons, use a time tagger and time-correlated single photon counting [5] (`timetagger`, section 1.5); with the gate-delay scan, choose the window and the step to the features: a 300 ns readout window optimized from a coarse trace is all an NV experiment needs (section 23.3).

How it is verified

`make test-hw` runs the gate-delay scan along the test emitter's decay and compares every point's mean with the expected decay (within $1.7\,\sigma$). The bench `tb_shots` checks every gate's count, shot by shot, against the bit-exact model, which follows equation (3) to the nanosecond.

9 Decisions

Some answers are needed before the next shot: is the ion bright or dark, did the herald detector click, is the atom in the trap? A computer that reads the counts over a network answers in milliseconds; counter answers on a pin about a tenth of a microsecond after the gate closes. Each shot has two ***decisions***, D1 and D2.

9.1 What a decision is

A decision looks at two gates A and B of the shot (any of G1–G4, the same one twice if only one matters) and tests each count against a window:

$$a = (\text{low}_a \le n_A \le \text{high}_a), \qquad b = (\text{low}_b \le n_B \le \text{high}_b), \tag{4}$$

then combines a and b by any of the 16 logic functions of two bits. The counts are the ones the SHOT records carry, saturated to 16 bits, so a PC can recompute every decision from the records; `high None` (65535) means no upper limit.

```
dev.apply({"decisions": {
    "1": {"gate_a": 1, "low_a": 3, "logic": "a"},                       # bright: G1 >= 3
    "2": {"gate_a": 1, "low_a": 1, "gate_b": 2, "low_b": 1, "logic": "a and b"}}})  # both clicked
```

The functions are named: `"false"`, `"a"`, `"b"`, `"a and b"`, `"a or b"`, `"a xor b"`, `"not a"`, `"not b"`, `"a and not b"`, `"b and not a"`, `"not (a and b)"`, `"not (a or b)"`, `"a == b"`, `"a or not b"`, `"b or not a"`, `"true"`. Examples:

- **state detection** (an ion, a neutral atom, an NV centre's charge state): D = a with $a = n_1 \ge k$, the threshold k from the count histogram (section 20);
- **a dark shot**: `high_a` 0 (exactly no count), or `"not a"`;
- **heralding**: D = a with $a = n_1 \ge 1$ on the herald detector's gate;
- **a coincidence of two gates**: `"a and b"` with both thresholds 1;
- **exactly one of two detectors**: `"a xor b"` (a which-path detection);
- **a count in a window**: `low_a` 3, `high_a` 5 (neither too few nor too many photons: a single atom rather than none or two).

A decision is taken once per shot, when the results of both its gates are in. It goes into the shot's SHOT record, into the per-point sums (the number of true decisions per point, so that the probability of "bright" per point is d_1/shots), and into the decision's counters:

```
print(dev.decisions())   # {"1": {"taken": 100000, "true": 69914, "last": True}, "2": {...}}
```

TCP: `DECISIONS?`

9.2 The decision's pin

An output O1–O4 set to `d1` or `d2` (section 10) shows the decision in one of four ways (`output`):

`level` the last decision, held until the next shot's decision (the default): for a receiver that reads a level when it needs it (a sequencer's TTL input sampled at its own time). It gives an edge only when the decision changes: awg's sequences jump on edges, so for a jump on every true decision use `pulse-true`.

`pulse-true`, `pulse-false` a pulse of `pulse` (8 ns to 2.048 µs in 8 ns steps) for each true (or false) decision; a new decision during a pulse restarts it. For receivers that act on edges.

`early` high from the moment gate A's running count reaches `low_a` until the next shot's gates start: "bright" is signalled as soon as the threshold-th photon is counted, without waiting for the gate's end. A sequencer can then end the detection early in most bright shots (adaptive readout, section 20.4).

```
dev.apply({"decisions": {"1": {"output": "pulse-true", "pulse": 200e-9}},
           "outputs": {"1": {"mode": "d1"}}})          # O1 (DIO4_N): a 200 ns pulse per bright shot
```

9.3 Latency

The ***latency*** of a decision is the time from the end of its gate (an edge in the gate's last nanosecond is the last that can count) to the edge at the output's connector. It was measured in three ways (figure 12, section 16.3):

- with the internal timer's TRIGs, which fall on the clock's grid, and a jumper from the output to a count input (`measure_latency`, below): 101.6 ns;
- with TRIGs from a second board (asynchronous to counter's clock), timed by counter's own gate-delay scan on the jumpered input: 101 to 109 ns, 105.0 ns on average. The decision is made on the 8 ns clock, so the latency depends on where in the clock period the gate ends: an even spread over 8 ns;
- with an oscilloscope between the TRIG pin and the output pin: 103.5 ns on average, within the jumpers' and probes' uncertainty of counter's own.

In `early` mode the output follows the deciding photon (the `low_a`-th count) by the same 105.5 ns. The latency is inside the FPGA (the 1 ns samplers, the channels, the gates, the decision and the output serializer, section 22.5) and in the pins' buffers; a channel's delay, and the analog lag while C9 or C10 is on, add to it. The simulation of the RTL gave 113.6 ns; the plan of the design aimed at 75–100 ns.

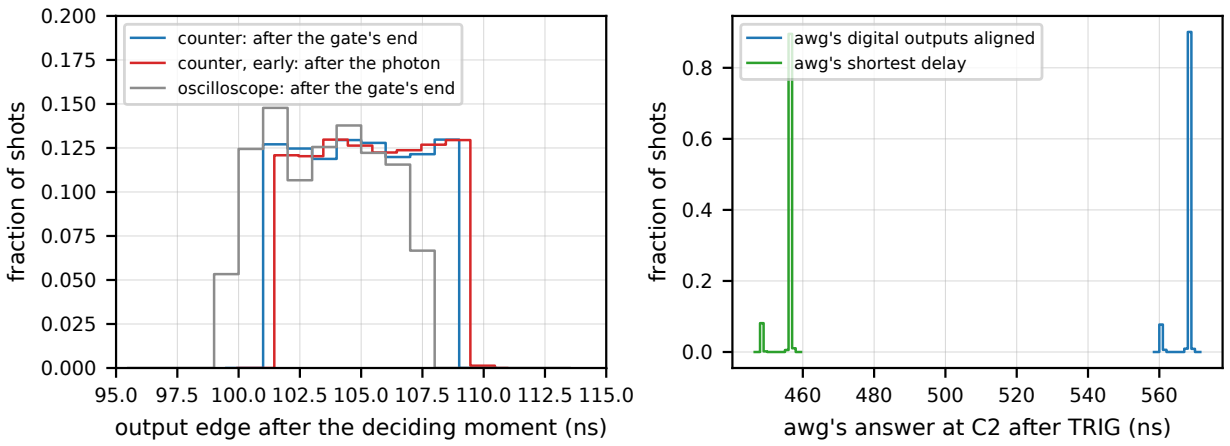


Figure 12: Measured (2026-11-05). Left: the output's edge after the deciding moment (the gate's end, or the photon in `early` mode), timed by counter through a jumper (3826962 shots each) and by an oscilloscope (900 acquisitions). Right: the answer of awg on a second board to the decision, through its event input, at counter's C2 (section 24.6).

`measure_latency` measures it on any board with one jumper from an output to a quiet count input: D1 is always true at G1's end and pulses the output, and a 1 ns gate scanned over the following 200 ns on the jumpered channel finds the edge (about 0.1 s; the configuration is restored afterwards).

```
lat = dev.measure_latency(output=4, channel=8)     # a jumper from O4 (E1 18) to C8 (E1 17)
print(lat["latency_s"], lat["edges_per_shot"])     # 1.015e-07, 1.0
```

TCP: `LATENCY? 4 8`

9.4 Feed-forward with a sequencer

A decision on a pin closes a loop through the sequencer: awg's sequence waits in a step that jumps on an event input, a PulseBlaster or ARTIQ branches on a TTL input. With awg on a second board the whole loop, from the end of counter's gate to awg's output, took 468 ns, or 356 ns with awg's digital outputs at their shortest delay; awg's part, from its event pin to its output, was 363 or 251 ns (section 24.6, figure 12). The uses:

- **preselection**: repeat a check (an atom loaded, an NV centre in its negative charge state) until it succeeds, then start the experiment;
- **heralding**: start or gate a measurement on the photon that announces its partner;
- **adaptive readout**: end a detection as soon as its result is certain;
- **feedback on the quantum state**: a measurement that selects the next operation.

Repeat until success. In the tests of section 16.4 the loop ran a repeat-until-success protocol. awg's sequence step sent a TRIG every 4.096 µs and waited for EVENT0. counter's test emitter played a probabilistic photon source (1.09 Mevents/s on C1, so that a 96 ns gate holds a photon with p = 0.09976). D1 = G1 ≥ 1, pulsed on O1, made awg jump to its step that "uses the state": it answered on C2 and returned to attempting. Of 2 047 844 attempts 204 167 succeeded: p = 0.09970, $-0.3\,\sigma$ from the expected value. Every success was answered exactly once, and no failure was answered. The number of attempts per success follows the ***geometric distribution*** $P(k) = (1-p)^{k-1}p$ (χ^2 = 71.6 for 78 degrees of freedom; mean 10.03 = $1/p$; figure 13), and the chance of success did not depend on the number of failures before it (the process has no memory). A success cut its attempt short: the next attempt began 2608 ns after its TRIG instead of 4096 ns. The same arithmetic sets the rate of heralded entanglement between remote nodes and of loading single atoms into tweezers: the mean time to success is the attempt's duration over p, and the wait is as often long as it is short.

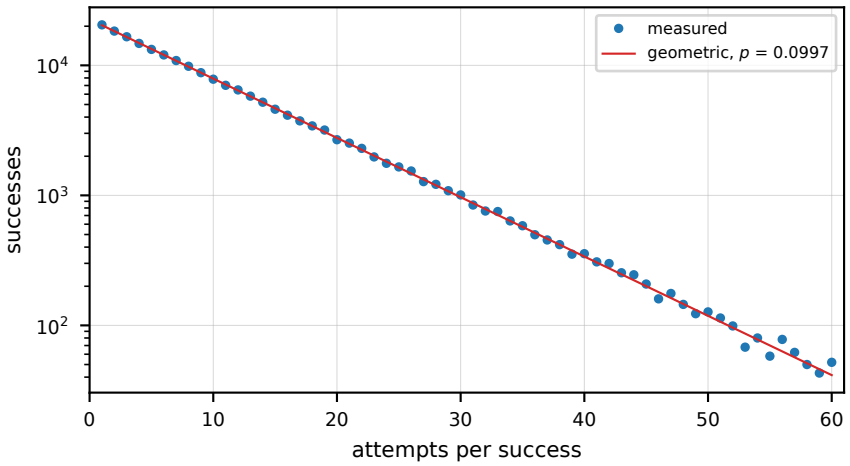


Figure 13: Measured (2026-11-05). Repeat until success through awg: the attempts per success (points) and the geometric distribution with the measured p (line).

How it is verified

`make test-hw`: in 100 000 SHOT records, D1 (G1 in 3..5) and D2 (G1 ≥ 5 xor G2 ≥ 4) are what each record's counts give, every time; the decisions' counters and the per-point D1, D2 sums agree with the

records. With an output jumpered to a count input: one count per shot for 20 000 shots, and the latency. The awg session (section 16.3) measured the latency with asynchronous TRIGs, on an oscilloscope, in the early mode, and the loop through awg; the tests of section 16.4 checked D1 and D2 in 896 000 SHOT records of random patterns and ran the repeat-until-success loop for 2 047 844 attempts.

10 Outputs O1–O4

The outputs O1–O4 (DIO4_N–DIO7_N, E1 pins 12, 14, 16, 18) send counter's results and signals to other instruments: a sequencer that branches, a switch that opens, an oscilloscope that shows where a gate sits, a time tagger that needs a clean logic pulse. Each output is a 3.3 V logic pin with a time resolution of 2 ns (a serializer of the FPGA, an OSERDESE2, shifts out four 2 ns bits per 8 ns word), in high impedance until a source is chosen.

```
dev.apply({"coincidences": {"1": {"enable": True, "channels": [1, 2], "window": 8e-9}},
           "outputs": {"1": {"mode": "d1"},                          # D1's pin
                       "2": {"mode": "gate", "gate": 1},             # G1's window
                       "3": {"mode": "echo", "channel": 9},          # C9's counted pulses as TTL
                       "4": {"mode": "herald", "unit": 1, "herald": [1, 2]}}})   # each C1+C2
    ↪ cluster
```

`mode`	The output shows
`off`	nothing: high impedance (the power-up state)
`low`, `high`	a fixed level
`pattern`	`pattern`: four 2 ns bits (bit 0 first) repeated every 8 ns: `0b0011` is a 125 MHz square wave, `0b0001` 2 ns pulses at 125 MHz
`d1`, `d2`	a decision, as its `output` setting says: a level, pulses, or early (section 9.2)
`gate`	gate `gate`'s window, to 2 ns (on an oscilloscope beside the detector's pulses: where the gate sits)
`trig`	each accepted TRIG as a 4 ns pulse (not the missed ones)
`echo`	channel `channel`'s counted events, after its dead time and delay, as 4 ns pulses
`herald`	a 4 ns pulse when coincidence unit `unit` closes a cluster that holds all channels of `herald`
`alarm`	high while the rate alarm is on (section 6.4)
`emitter`	the test emitter's lane `lane` as 4 ns pulses (section 13)

Table 4: The outputs' sources.

10.1 What each source is for

Decisions close loops through a sequencer (section 9.4).

Gate windows answer the most common question of a new setup: does the gate sit where the photons are? Put the gate's output and the detector's signal on two channels of an oscilloscope (the output appears with the same latency as a decision, about 0.1 µs late: compare the gate with a copy of the TRIG, or use the gate-delay scan, section 8.4).

TRIG echoes show which TRIGs started shots: a missing pulse is a missed TRIG.

Echoes turn a channel's counted events into clean 3.3 V pulses of 4 ns: the counter as a discriminator and pulse shaper. A NIM detector on IN1 (C9) becomes a TTL source for instruments that take only logic levels (a time tagger's channel, another board's trigger); a ringing detector's pulses come out with one clean pulse each after the channel's dead time. Each pulse starts at the even nanosecond of its event's slot, so events 8 ns apart stay 8 ns apart.

Heralds mark coincidences as they happen: a pulse per pair detection to gate a second instrument, to count heralded events elsewhere, or to trigger a measurement of the partner photon (section 11.3).

The alarm stops an experiment or lights a lamp when an ion is lost or a laser unlocks.

Patterns and the emitter are test signals: a 125 MHz pattern checks a cable and an input; the emitter's pulses imitate a detector for another instrument.

10.2 Safe state

`dev.safe()` (TCP `SAFE`) puts every output in high impedance, the rate monitors at 0 V and the test emitter off: the configuration at power-up. Loading the design gives the same state, until the service restores the last saved settings.

How it is verified

`make test-hw` finds any output wired to a count input by driving O1–O4 briefly, then: a 125 MHz pattern counted exactly on both edges, the emitter's 4 ns pulses, D1's pulses one per shot, the latency, a channel's echo (every counted edge looped back, up to 5 Mevents/s), unit 1's heralds (one per cluster) and C9's echo (every counted pulse), all exact. On the board used here the outputs drove another board's event input and a count input through 20 cm and 2 cm of wire without a lost or extra edge in millions of pulses.

11 Coincidence units

A ***coincidence*** is two or more detectors clicking within a short time, the ***coincidence window***. Photon pairs from parametric down-conversion arrive together; the two outputs of a beam splitter almost never click together behind a single-photon source; entangled photons click in correlated pairs. counter has two ***coincidence units*** that count coincidences on the board, among up to four channels each, at any rate the channels take.

11.1 Clusters

A unit takes up to four channels (any of C1–C10, its ***lanes*** L0–L3 in the given order) and groups their events into ***clusters*** with the definition of the `timetagger` design: a cluster opens at an event; every later event, on any lane, within the window W after that opening event joins it; the first event after the window opens the next cluster (figure 14). Each cluster is counted once, by its ***pattern***: the set of lanes that had at least one event in it. A unit keeps a 48-bit total for each of the 15 patterns (never cleared).

```
dev.apply({"channels": {"1": {"enable": True}, "2": {"enable": True}, "3": {"enable": True}},
           "coincidences": {"1": {"enable": True, "channels": [1, 2, 3], "window": 8e-9}}})
co = dev.coincidences()["1"]
print(co["exclusive"]["C1+C2"])     # clusters with C1 and C2 and not C3
print(co["inclusive"]["C1+C2"])     # clusters with at least C1 and C2: C1+C2 and C1+C2+C3
```

TCP: `COINC?`

The ***exclusive*** counts are what the unit counts: clusters of exactly these lanes. The ***inclusive*** counts, clusters with at least these lanes, are sums of exclusive ones; the two-fold and three-fold coincidences of the textbooks are inclusive. The window W runs from 8 to 1023 ns in 1 ns steps.

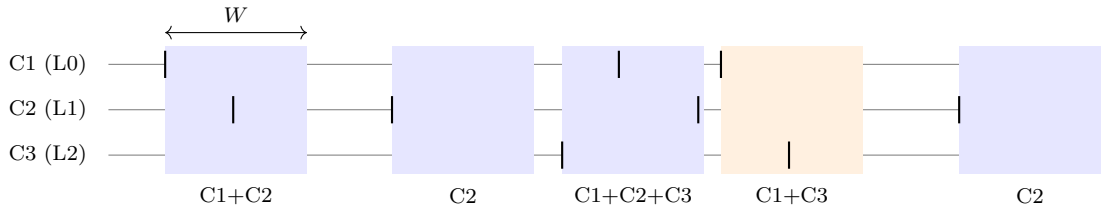


Figure 14: Clusters: each opens at an event and takes every event within W after it, on any lane. The fourth cluster opens at an event just after the third one's window: that event is not in the third cluster, although it is within W of that cluster's last event.

The definition has consequences that matter when comparing numbers:

- an event belongs to one cluster only. A strong peak of correlated pairs takes events away from the accidental clusters next to it: in figure 15 the accidentals just outside the peak dip below their level far away. Definitions that open a window around every event count some pairs twice and give no such dip;
- two lanes coincide if their events are at most W apart in either order: for independent events at rates R_1 and R_2 the accidental two-fold clusters are

$$N_{\text{acc}} \approx R_1 R_2 \,(2W + 1\,\text{ns})\, T \tag{5}$$

  in a time T (the 1 ns: events in the same nanosecond, $|t_2 - t_1| \le W$ in whole ns), as long as $RW \ll 1$. Section 21.1 derives it and its corrections;
- counter's clusters are the same as timetagger's `analysis.coincidences` on the same events, up to the 1 ns resolution of counter's times.

11.2 The delayed unit: accidentals and delay scans

Unit 2 can see one of its channels later by a `delay` of 0 to 16.383 µs (1 ns steps). Delayed by much more than any correlation time of the light, its coincidences are purely accidental: the same detectors, the same rates, the same window, only the correlation removed. Its count is the accidental background of unit 1's measurement, counted rather than computed, with none of the assumptions of equation (5) (stationary rates, uncorrelated dead times). This is the standard way to measure $g^{(2)}(0)$ with two detectors (section 21.2):

```
dev.load_preset("hbt")   # unit 1: C1, C2 within 8 ns; unit 2: the same, C2 seen 1 us later
a = dev.coincidences(); time.sleep(60); b = dev.coincidences()
n12 = b["1"]["inclusive"]["C1+C2"] - a["1"]["inclusive"]["C1+C2"]
acc = b["2"]["inclusive"]["C1+C2"] - a["2"]["inclusive"]["C1+C2"]
print(analysis.g2_zero(n12, accidentals=acc))
```

Scanned in 1 ns steps, the delay maps the coincidences against the time difference of the two channels: the cross-correlation of their events, at the resolution of the window. It finds the delay that aligns two detectors (the peak of correlated pairs), and it shows bunching and antibunching at the scale of nanoseconds. Figure 15 shows both with the test emitter: a peak of correlated pairs, 16 ns after, 30.0 times the accidentals, whose level, 20143 s^{-1}, matches equation (5) (20220 s^{-1} for $R_1 = 1.99$ and $R_2 = 0.60$ Mcount/s); and the dip of an antibunched source, two lanes that never click in the same 8 ns word, down to 0.53 of the accidentals (the cluster simulation: 0.53). For $g^{(2)}(\tau)$ with a resolution of tens of picoseconds, use a time tagger.

The same scan on real wires: in the tests of section 16.4, awg played a pattern with 348 pairs per pass (C2 40 ns after C1) on independent backgrounds, 4.8 Mevents/s per channel in all. Unit 2 (W = 12 ns) scanned C1's delay from 0 to 80 ns in 1 ns steps (figure 16). The pairs stand on a plateau $2W + 1$ = 25 ns wide around 40 ns: 419 clusters per pass on 139 accidentals per pass. The accidentals were counted with C1 delayed by 1 µs; equation (5) with awg's 8 ns grid gives 144. That is a coincidence-to-accidental ratio (***CAR***) of 2.01, the figure of merit of a photon-pair source (sections 21.1, 25.2). Every setting at least 2 ns from a window's edge counted exactly the clusters of the pattern (61 of 61; both units, 82 settings each); at the edges the result depends on where the two edges fall within the nanosecond (section 14.4). The steps every 8 ns come from awg's grid; real photons arrive at any time, and their accidental floor is flat.

11.3 Gating and heralds

With `gate` true a unit takes only the events inside gate G1's windows: coincidences only during the excitation pulse, or only in the shots that a sequence selects. And a unit's clusters can ***herald*** on an output: a 4 ns pulse in the word in which a cluster closes whose pattern holds all channels of the output's `herald` list (`[1]`: every cluster with C1; `[1, 2]`: every cluster with C1 and C2), about a

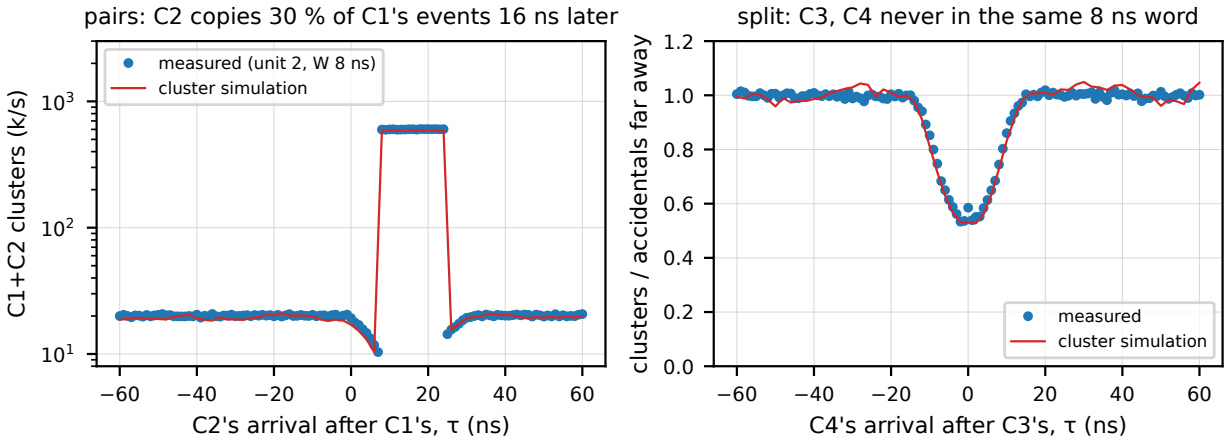


Figure 15: Measured (2026-11-05). Coincidences against a delay scanned with unit 2 ($W = 8$ ns, 0.2 s per point), with the test emitter. Left: C2 copies 30 % of C1's events 16 ns later; the plateau of the peak is $2W + 1 = 17$ ns wide; just outside it the accidentals dip (an event is in one cluster only). Right: C3 and C4 share one source but never in the same word: an antibunched pair of channels, normalized to the accidentals far from zero delay. Lines: the clusters of simulated events of the same processes.

tenth of a microsecond after the cluster's window ends. A heralded single-photon source uses it to announce each photon to another instrument.

Both units' 15 pattern totals also go into the RATE records (section 6.2): time series of coincidence rates with exact intervals.

How it is verified

`make test-hw`: unit 1's seven exclusive patterns of C1, C2, C3 (the emitter's pairs and an independent source, $W = 20$ ns, 13.6 million clusters) within $1.8\,\sigma$ of a cluster simulation of the same processes; unit 2 with C2 delayed by 2 µs within $0.25\,\sigma$; the RATE records with both units' patterns, exact intervals; the herald one pulse per cluster, exactly. The model's clusters are checked against the reference definition above on random events, and the RTL's against the model clock by clock (`tb_coinc`).

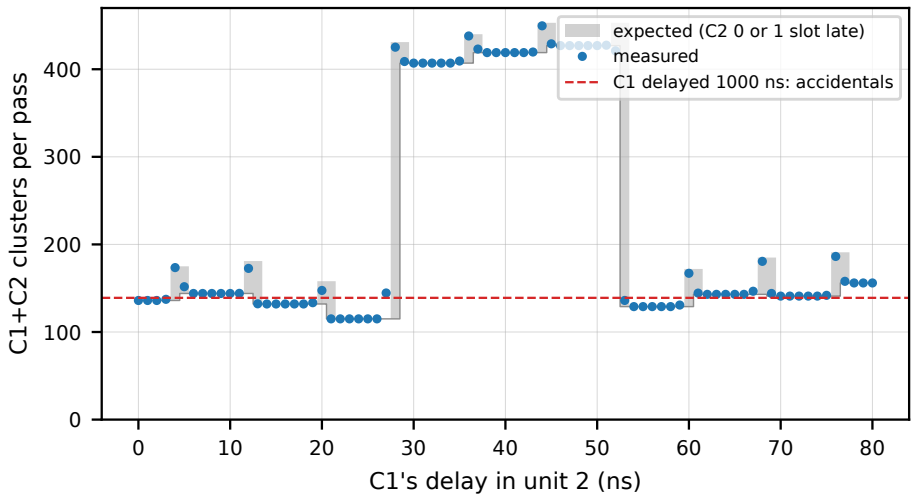


Figure 16: Measured (2026-11-05). A delay scan on wires: awg's pairs (C2 40 ns after C1) on random backgrounds, unit 2 ($W = 12$ ns) with C1 delayed by 0–80 ns, 400 passes of 262 µs per point. Grey: the clusters of the pattern's events for C2's edge 0 or 1 slot after C1's (its sub-ns offset); dashed: the accidentals, C1 delayed by 1 µs.

12 Records and the stream

The per-point sums answer "how many, on average"; some questions need every shot: the distribution of counts per shot (to choose a threshold and know its fidelity), correlations between consecutive shots (quantum jumps, the charge state of an NV centre), the decisions shot by shot. counter writes ***records*** into a ring buffer in the board's memory, and the board serves them to any number of programs on PCs.

12.1 The records

SHOT one per shot when `shots.records` is on: the point, the TRIG's time in nanoseconds, the sweep, the four gates' counts (16 bits each, saturated at 65535), D1, D2, and flags for a missed TRIG during the shot and a saturated count. 5 words.

RATE one per `rates.interval` when `rates.enable` is on: the interval's end time and the low 32 bits of all channels' totals (and the coincidence units' patterns), with exact intervals between consecutive records (section 6.2). 13 words, 15 more per coincidence unit that is on.

SWEEP at the end of each sweep: its index, its TRIGs (missed ones included), the missed, whether it was checked and found bad.

SWAP at each swap of the per-point banks: the bank, why (the service, a sweep's end, automatic), the sweep and point at that moment.

RUN at each start and stop of a run: the run's index, the time, the reason for a stop, the shots and the missed.

Every record carries its time (in the core's clock, from the design's loading). The layouts are in appendix C.

12.2 From the FPGA to a PC

The records go through queues of whole records into a ***ring buffer*** of several megabytes in the board's memory (written by the FPGA directly, through the processor's high-performance memory port). The board's service runs a ***ring pump*** that reads the ring and serves it, as frames, on TCP

port 5026 (and as `RECORDS STREAM` on the command port 5025) to every connected client. Nothing is lost silently:

- a client that falls behind loses whole frames; each frame's header says how many bytes were lost before it;
- if the ring is full, the FPGA drops whole records and counts them (`hardware()["records_dropped"]`, `run_status()["records_dropped"]`);
- RATE records carry a sequence number, SHOT records the TRIG's time, so a gap shows.

A client on the board itself received 2 000 000 SHOT records per second (40 MB/s) without a loss; a PC on the lab's gigabit network received 100 000 RATE records per second and 423491 SHOT records in 2 s without a loss (measured).

```
from counter import RecordStream, analysis, model
dev.apply({"shots": {"records": True}})
words = []
with RecordStream("192.168.0.102") as stream:          # port 5026 (the ring pump)
    for frame in stream.frames(timeout=2.0):
        header, w = analysis.parse_frame(frame)         # header["lost"]: bytes lost before it
        words.append(w)
        if sum(len(x) for x in words) > 5 * 100000:
            break
records = model.decode(np.concatenate(words).tolist())  # dicts: {"type": "SHOT", "point", "counts",
    ↪ ...}
h = analysis.count_histogram(records, gate=1)           # h[n]: the shots with n counts in G1
```

Records can span frames: decode the concatenated words, not each frame alone. To keep a run, write the frames' payloads to a file (`np.save`); `model.decode` reads them back.

How it is verified

`make test-hw`: SHOT records at 500 000 shots per second, one per shot, none lost, their counts equal to the per-point sums'; at 1 and 2 million shots per second, none lost; RATE records every 10 µs, none lost; no record dropped in the FPGA over the whole test. The benches compare every word of the ring with the bit-exact model.

13 The test emitter

The ***test emitter*** is a source of detector-like pulses inside the FPGA. It exercises every function of the counter without a detector, a laser or a sample: to learn the instrument, to check a configuration before the optics are ready, to rehearse an experiment and its analysis with known answers, and to test the instrument itself (most of `make test-hw` runs on it). Its pulses enter the channels where a detector's pulses would, before the edge and dead-time logic, so they test that logic too.

13.1 Lanes

The emitter has four ***lanes***. Each makes one random trial per 8 ns word with a probability set by its `rate` (up to 125 million per second), at a random slot of the word, and sends a 2 ns pulse into the channel `channel` (1–10; 0: none), ORed with whatever the input carries. The random numbers come from xoroshiro generators, one random decision per word and lane.

```
dev.apply({"channels": {"1": {"enable": True}},
           "emitter": {"lanes": {"1": {"rate": 2e6, "channel": 1}}}})
print(dev.counters()["emitter"])      # each lane's events (32 bits)
```

Because a lane makes at most one event per word, its counts are ***binomial***, not Poisson: in N words with probability p per word, the mean is Np and the variance $Np(1-p)$, a ***Fano factor*** (variance over mean) of $1-p$. At 1 Mevents/s ($p = 0.008$) that is 0.992, as good as Poisson; at 50 Mevents/s it is 0.6, and the dead time lowers it further (figure 21, section 18).

13.2 Pairs and antibunching

Lanes 1–2 and 3–4 can be coupled (`emitter.pairs`):

`independent` each lane on its own;

`pairs` lane 2 copies each event of lane 1 with `probability`, `delay` later (0–248 ns in 8 ns steps): correlated pairs, a photon-pair source with a heralding efficiency and a relative delay;

`split` each event of lane 1 goes to lane 2 with `probability`, otherwise stays on lane 1, never both: a source behind a beam splitter whose two outputs never click together within a word, an ***antibunched*** source.

In both coupled modes lane 2's own rate is unused. Figure 15 shows the coincidences of both against a delay.

13.3 Bright and dark shots

With shots running, the emitter imitates a quantum system read out by fluorescence: at each TRIG it draws a ***state***, bright with the probability `table[point]` of the shot's point (a table of up to 1024 probabilities loaded by the PC), dark otherwise. In a bright shot each lane's rate is raised by `bright` (events/s, may be negative), and this extra rate decays with `decay` (0: constant; the decay is by a power of two per word, $\tau \approx 2^k \times 8\,\text{ns}$, the nearest to the setting). The bright state takes effect 88 ns after the TRIG (the emitter's own latency), which the gates' delays should allow for.

```
taus = [20e-9 * k for k in range(51)]
dev.apply({"channels": {"1": {"enable": True}},
           "emitter": {"lanes": {"1": {"rate": 0.23e6, "channel": 1, "bright": 0.2e6}}, "decay": 400
    ↪ e-9},
           "shots": {"trigger": "timer", "timer_period": 4e-6, "timer_seq": True, "points": 51},
           "gates": {"1": {"mode": "timed", "channels": [1], "width": 300e-9},
                     "2": {"mode": "timed", "channels": [1], "delay": 2.5e-6, "width": 300e-9}}})
dev.load_emitter_table([0.5 * (1 + math.cos(2 * math.pi * t / 200e-9) * math.exp(-t / 800e-9)) for t
    ↪  in taus])
```

TCP: `TABLE 1.0, 0.95, ...`

Figure 17 is this rehearsal of an NV-centre Rabi measurement: the table sets a Rabi oscillation of 200 ns period, the rates give NV-like signals, 0.071 to 0.103 counts per shot in G1 and 0.0690 in the reference window, and `analysis.fit_rabi` recovers the period, 199.9 ± 0.4 ns. The same emitter with a long gate and the table at 0.5 rehearses ion detection (figure 22).

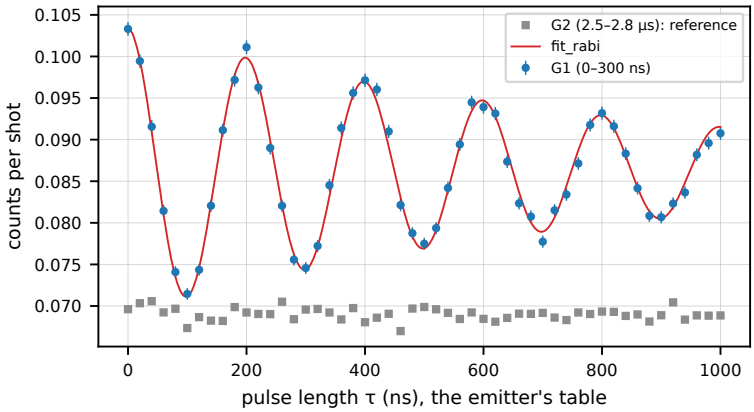


Figure 17: Measured (2026-11-05). A Rabi measurement rehearsed with the test emitter: 147091 shots per point (7501691 in 30 s), G1 the first 300 ns of each shot, G2 a reference window after the bright state's decay; the table's oscillation has a period of 200 ns and a decay time of 800 ns; `fit_rabi`: 199.9 ± 0.4 ns and 836 ± 38 ns, χ^2 per degree of freedom 1.11.

13.4 To the outputs and to OUT1, OUT2

A lane can also go to an output pin (`outputs` mode `emitter`: 4 ns logic pulses, a test source for another instrument or, through a jumper, for a count input) and to OUT1 or OUT2 as analog pulses (`monitors` mode `pulses`: 8 ns pulses of `amplitude` volts): with a cable from OUT1 to IN1 they test the analog channels' discriminators (figure 6).

```
dev.apply({"emitter": {"lanes": {"1": {"rate": 0.1e6, "channel": 0}}},     # not into a channel
           "monitors": {"1": {"mode": "pulses", "lane": 1, "amplitude": 0.8}},
           "channels": {"9": {"enable": True, "threshold": 0.25, "hysteresis": 0.05}}})
```

How it is verified

`make test-hw`: each lane's events per word are binomial (the count of 125 million words within 1.5 σ of the set probability); the pairs copy the set fraction; the split source's lanes share its events in the set proportion; the shots' means per point follow the table; the gate-delay scan follows the decay.

14 Timing, calibration and the timebase

Every time in counter refers to the connectors, and every rate to the board's clock. This section explains how, what the board calibration contributes, and how well it was checked.

14.1 The board calibration

All rpfpga designs on a board share one ***board calibration*** (`common/CALIBRATION.md`): the gains and offsets that convert the converters' codes to volts at the SMA connectors, the delays of the converters, and two delays of the extension-connector pins. counter uses

- the input calibration of IN1 and IN2 for C9's and C10's thresholds in volts, and the output calibration of OUT1 and OUT2 for the monitors and the test pulses;
- `adc_delay`, the delay from the DAC's port in the FPGA through OUT1, a cable and IN1 to the ADC's port (119.9 ns on the board used here), `pins_delay_s`, the time an output pin changes after OUT1 shows the sample it is aligned with (−28.6 ns), and `pins_loop_s`, the delay from an output pin through a short jumper to a 1 ns input (4.5 ns): together they give the analog channels' lag against the pins.

A board without a calibration uses nominal values: volts are then good to a few per cent and the analog alignment to a few nanoseconds. `dev.status()["calibration"]` names what is in use.

14.2 The analog channels' alignment

An analog pulse takes longer to become a count than a logic edge: the input amplifier and filter, the converter's pipeline and the interpolation add about a tenth of a microsecond. While C9 or C10 is on, counter adds that lag Δ to every pin channel's delay, to every gate's delay and to GATE's delay, so that all settings and times stay referred to the connectors:

$$\Delta = \text{adc_delay} - \text{pins_loop_s} + \text{pins_delay_s} + A, \tag{6}$$

where A, the analog lanes' lag inside the FPGA against the pins', is a constant of the design (`driver.A_LAG`). The simulation of the RTL gave $A = 23.5$ ns; the measurement of figure 18 showed that this timed C9 15.5 ns early, and the driver now uses the measured $A = 7.95$ ns: $\Delta = 95$ ns on the board used here (`describe()["alignment"]`). The difference, two clock periods within the measurement's error, is not explained; the pins' input pipeline counted against the definition of `pins_loop_s` is its likely place.

The measurement: one test-emitter event as an 8 ns pulse on OUT1 (cabled to IN1, C9, threshold 0.25 V) and on O4 (jumpered to C8) and O2. Unit 2's delay scan gave C9's time minus C8's: 6.6 ns with $\Delta = 110$ ns. An oscilloscope (OUT1's cable on a coax input through a T at 1 MΩ, like IN1; O2

on a probe whose delay against the coax input was measured, 7.04 ns, by putting both on the same pulse) gave the truth: OUT1's 0.25 V crossing at the cable's end 22.15 ns after C8's edge. With $\Delta = 95$ ns counter gives 21.70 ns, within the measurement's uncertainty of about 1 ns.

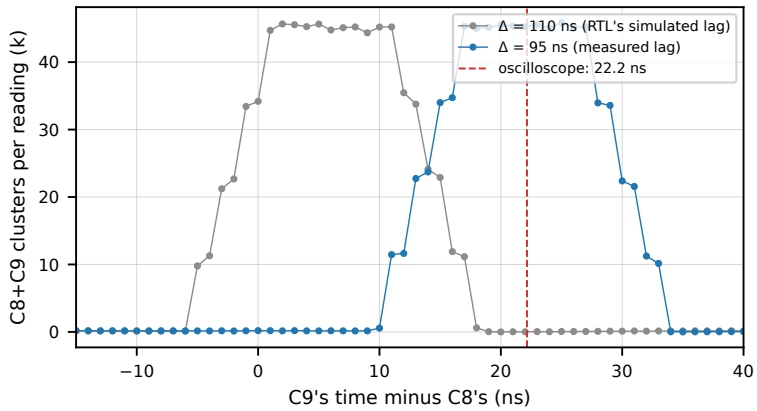


Figure 18: Measured (2026-11-05). C8+C9 coincidences (unit 2, $W = 8$ ns) against C9's time minus C8's, for one emitter event on OUT1→IN1 and on O4→C8, with the alignment of the RTL's simulated lag (grey) and of the measured lag (blue); dashed: the oscilloscope's value. The width of each box is the window (±8 ns) and OUT1's 8 ns grid.

Pulses of another shape cross the threshold at another point between the samples: a fast NIM edge, an SNSPD's sub-nanosecond rise. For coincidences between an analog and a pin channel at the 10 ns level, measure the relative delay with the coincidence scan of unit 2 (section 11.2) on the real signals and correct it with the channels' delays.

14.3 Latencies

Path	Latency (connector to connector)
gate's end → decision pin	101–109 ns, 105.0 ns on average (by where TRIG falls in the 8 ns clock); 101.6 ns with the internal timer; oscilloscope 103.5 ns
deciding count → early pin	105.5 ns
gate's end → decision pin, TRIG from a pin (awg)	104.5 ns on average (O4 into C8)
a channel's event → echo pin	92.5 ns (C1's edge to O4's, read on C8 through a 2 cm jumper); 155–157 ns after TRIG by the 2 ns output steps
a cluster's first event → herald pin	121.5 ns $+W$ on average, for W = 8–128 ns; spread evenly over 8 ns (the word in which $t_0 + W$ falls)
analog channels	Δ more than the pins (95 ns here)
gate's end → awg's output (preselection loop)	468 ns, 356 ns with awg's shortest digital delay

Table 5: Measured latencies (sections 9.3, 16.3, 16.4).

14.4 Below a nanosecond

counter samples its inputs every nanosecond, yet it can measure a fixed offset between two edges far more finely when the source runs on another clock. Two boards' crystals differ by a few ppm, so the edges of awg's patterns drift slowly through counter's sampling instants (by 1 ns in about 200 µs here). Over many shots each edge lands at a uniformly random point within the nanosecond. A gate edge placed at an edge's nominal time then counts that edge in a fraction of the shots, and the fraction is the offset. In figure 19 (left), C1's edge, nominally 64 ns after TRIG, fell into the window $[64, 72)$ ns in 48.0 % of 2 million shots: it arrives −0.52 ns early against TRIG's edge (C2's: −0.47 ns). The gate that ends at 64 ns counted it in exactly the other shots, so adjacent gates never

count an event twice or lose it. The same principle, ***dithering***, makes the average of a converter's readings finer than its step. It needs the two clocks to be unrelated: with a sequencer locked to counter's clock every edge lands at the same point, and the fraction is 0 or 1.

Measured this way, an edge on a 20 cm jumper came 95 ps early or 94 ps late depending on the pulse 8–16 ns before it (figure 19, right): the wire's ringing, invisible at 1 ns, yet enough to bring the second pulse of a pair 16 ns apart inside a 16 ns dead time in 10.6 % of the pairs. It also brought a C1+C2 pair 16 ns apart inside a 16 ns window in 0.946 of the cases (0.949 predicted from the two edges' offsets). A boundary placed exactly on a sequencer's grid is decided by such tenths of a nanosecond; place gate edges, dead times and windows a few nanoseconds away from the times that matter.

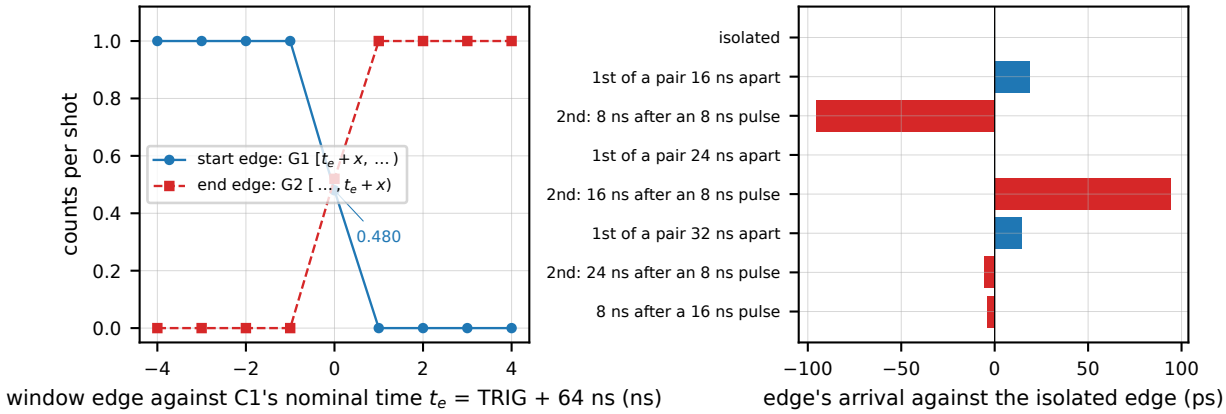


Figure 19: Measured (2026-11-05). Left: counts per shot of C1's edge (nominally TRIG + 64 ns) in a window whose start (G1, blue) or end (G2, red) sits x ns from that time, 8 ns wide, the gate-delay scan stepping x by 1 ns per point; awg and counter on unlocked crystals. Right: the arrival of single 8 ns pulses of one awg pattern on one 20 cm jumper, against an isolated pulse, from the same fractions.

14.5 The timebase

counter's clock is the board's 125 MHz crystal oscillator. Every rate, interval and time refers to it: a crystal that runs slow by ϵ makes each of its seconds $1 + \epsilon$ real seconds long, and every rate read high by ϵ. Counting a 10 MHz reference from a GNSS-disciplined oscillator on IN2 (section 5.5) gave a reading 5.66 ppm high on the board used here: its crystal runs 5.66 ppm slow. The crystal's frequency also changes with the die's temperature, by about −0.13 ppm per kelvin (measured with timetagger on the same board). For counting experiments this is far below the counts' statistics; when it matters (a rate compared with another instrument's, an absolute frequency) measure ϵ with a reference and divide the rates by $1 + \epsilon$.

Overnight on 2026-11-05, C10 counted the GNSS oscillator's 10 MHz in RATE records of exactly one second of the board's clock for 12 hours, with the die's temperature read every 30 s (figure 20). The crystal ran 5.70 ppm slow on average, between 5.66 and 5.77 ppm in minute averages, while the die stayed between 54.8 and 56.2 °C. The two do not follow a simple coefficient: when the die warmed by 0.8 K after eight hours, the crystal slowed by 0.08 ppm at once and recovered within about 1.5 hours while the die stayed warm. The die's temperature is not the crystal's. The Allan deviation of the board's time against the reference, 4.4×10^{-8} at 1 s (one count of 10^7 limits a single second), falls to 2.1×10^{-9} at 54 s and rises again to 1.5×10^{-8} at 4320 s as the temperature wanders. So a rate measured over a minute is good to a few parts in 10^9 against the crystal's own mean, and over hours to some 10^{-8}; an absolute rate needs the reference.

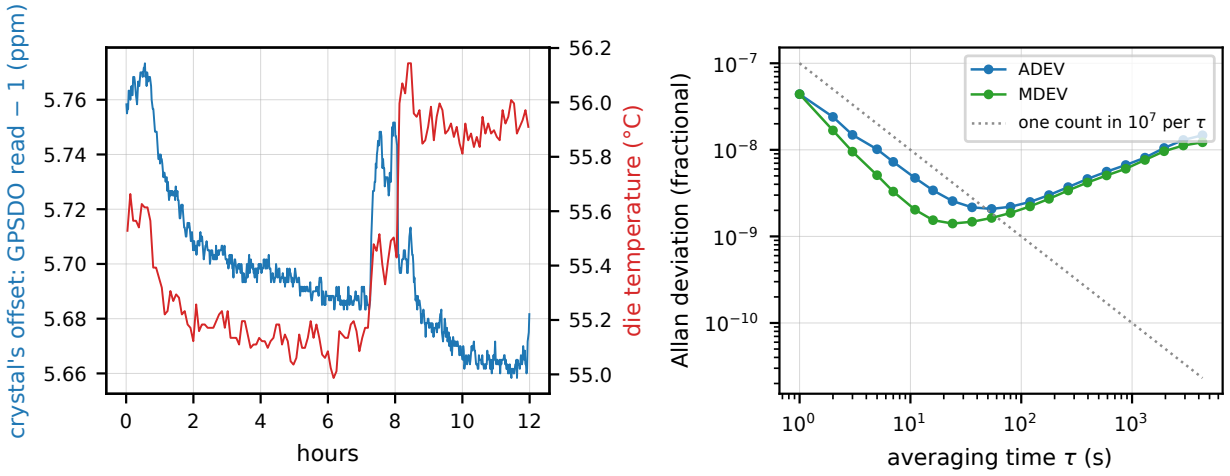


Figure 20: Measured (2026-11-05). The board's crystal against a GNSS-disciplined 10 MHz on IN2 (C10), 12 hours. Left: its offset in minute averages (blue) and the die's temperature (red, 5-minute averages). Right: the Allan and modified Allan deviations of the board's time against the reference; dotted: one count of the 10 MHz per averaging time.

15 Software

counter is controlled by Python, from a PC or on the board, through the board's service. The same methods work in both places; the service also speaks a line-based TCP protocol for any other language, and the board's Jupyter server has a notebook with panels.

15.1 The pieces

The service (`rp-overlay.service` on the board, TCP port 5025) loads the design, applies and saves settings (restored after a reboot), serializes the requests of several clients, runs the worker that collects the per-point sums every second, and starts the ring pump (port 5026) that serves the records.

`rpfpga` the common Python package: the overlay base class, the client, the board calibration, the record stream, the GUI panels.

`counter` the design's package (`counter/control/counter`): `driver` (on the board), `client` (`CounterClient`, on a PC), `protocol`, `model` (the bit-exact model of the FPGA and the records' decoder), `analysis` (fits, statistics, coincidences; numpy), `gui`.

```
from counter import CounterClient          # on a PC (the repository's common/python and
dev = CounterClient("192.168.0.102")        #  counter/control on the path)
# on the board: import rpfpga; from counter import DRIVER; dev = DRIVER()
```

15.2 Settings

The settings are one nested dictionary (appendix A lists every key with its range and default). `apply` changes the keys it is given and leaves the rest; with `replace=True` it sets the whole configuration (missing keys take their defaults). Every apply is checked first: an out-of-range value raises an error and changes nothing. The applied values are returned, rounded as the hardware takes them (times to whole nanoseconds, rates to the emitter's steps).

```
applied = dev.apply({"gates": {"1": {"width": 300.4e-9}}})   # -> 300e-9
cfg = dev.config()                  # the whole configuration
print(dev.describe())               # in words: each channel, output, monitor and the alignment
dev.safe()                          # outputs in high impedance, monitors 0 V, the emitter off
```

TCP: `CONFIG {...}`

TCP: `CONFIG?`

TCP: `SAFE`

Several clients can be connected at once; each apply bumps a revision number, and `apply(..., expected_revision=r)` refuses a change made on top of settings that another client has changed meanwhile.

15.3 Presets

A ***preset*** is a named configuration. counter's presets set up the experiments of Part III; `dev.save_preset("mine")` saves the current settings as a preset of your own.

Preset	What it sets up
`off`	everything off, the outputs in high impedance (the power-up state)
`rates`	C1–C8 counting, RATE records every 0.1 s, C1's rate on OUT1 at 1 V per Mcount/s
`selftest`	the emitter without wiring: 1 Mcount/s on C1, half of them copied 16 ns later to C2, a split (antibunched) source of 2 Mcount/s on C3 and C4
`shots-selftest`	shots from the timer every 10 µs over 16 points, the emitter bright or dark per shot on C1, G1 counting C1 for 2 µs
`nv-readout`	TRIG at each laser readout, SEQ at each sweep, one shot per point (100 points); G1 the signal (0–300 ns), G2 the reference (2–3 µs) on C1
`nv-trace`	the readout's time trace: G1 a 20 ns window stepped by 20 ns over 150 points
`ion-detection`	G1 counts C1 for 500 µs after TRIG; D1 = G1 $\geq$ 3 held on O1; SHOT records
`preselection`	a 100 µs gate; D1 = G1 $\geq$ 2 as a 200 ns pulse on O1 for a sequencer's event input; D2 = G1 = 0 on O2
`hbt`	C1, C2: unit 1 their clusters within 8 ns, unit 2 with C2 1 µs later; RATE records every second
`heralded-g2`	the herald on C1, a beam splitter's outputs on C2, C3: both units, unit 2 delayed
`bell`	Alice on C1 (+), C2 (−), Bob on C3 (+), C4 (−): the clusters of the four, unit 2 with C3 delayed

Table 6: counter's presets.

```
print(dev.presets())
dev.load_preset("nv-readout")
```

TCP: `PRESET nv-readout`

15.4 counter's methods

15.5 Analysis

`counter.analysis` has the calculations of Parts II and III; it needs numpy and runs on a PC or on the board.

15.6 The notebook

`counter.ipynb` on the board's Jupyter (and `counter.gui.make_gui(dev)` in any notebook) shows: the channels with their rates, enable, edge or threshold, dead time and delay; the analog channels; the coincidence units with a live table of patterns and $g^{(2)}$; the shots, points, gates (with a timing diagram of the gates against TRIG) and decisions; the per-point results; the count distribution of a gate from SHOT records, with the two-Poisson fit, the threshold and the fidelity; the emitter; RATE

Method	Does
`rates(seconds=0.1)`	every channel's rate from two readings of the totals (section 6)
`counters()`	the totals, rejected edges, the emitter's events, the alarm
`pins(clear=False)`	the inputs' levels and edges seen
`run(clear=True)`, `stop()`, `clear()`	runs and the per-point results (section 7.3)
`trigger()`, `next_point()`, `sequence()`	a software TRIG, NEXT, SEQ
`run_status()`	the run's state and counters
`results(now=False)`	the per-point 64-bit totals and means (section 7.5)
`load_emitter_table(p)`	the emitter's bright probability per point (section 13.3)
`decisions()`	the decisions' counters (section 9)
`measure_latency(output, channel, jumper)`	the decision latency through a jumper (section 9.3)
`coincidences()`	both units' pattern totals, exclusive and inclusive (section 11)
`analog(samples=256)`	IN1's and IN2's levels and noise in volts (section 5.2)
`hardware()`	the core's layout version, the samplers' PLL, the ring writer
`worker()`	the service worker's statistics

Table 7: counter's methods, besides the common ones (`apply`, `config`, `describe`, `status`, `safe`, `presets`, `load_preset`, `save_preset`, `calibration`, `ranges`, `set_input_range`). Appendix B has their TCP commands.

records, the alarm, the monitors and the outputs; the presets and the board calibration. The panels poll the board in the background and never block one another.

15.7 TCP

Every method is a text command on port 5025 (appendix B): one line in, one line out, `OK` and JSON or `ERR` and a message. For example, with nothing but Python's standard library:

```
import socket
f = socket.create_connection(("192.168.0.102", 5025)).makefile("rw")
f.write("RATES? 1\n"); f.flush()
print(f.readline())      # OK {"interval_s": 1.0000..., "channels": {"1": 1012.0, ...}}
```

Function	Computes
`parse_frame`, `rate_table`	a stream frame's words; RATE records into counts and rates per interval, with the patterns
`dead_time_corrected(rate, tau)`	the true rate behind a non-paralyzable dead time, eq. (1)
`contrast(signal, reference)`	$1 - S/R$ with its Poisson error, per point (NV readout)
`fit`, `fit_rabi`, `fit_ramsey`, `fit_exponential`, `fit_lorentzian`	weighted Levenberg–Marquardt fits with errors
`count_histogram(records, gate)`	the distribution of a gate's counts from SHOT records
`two_poisson(histogram)`	a mixture of two Poisson distributions by expectation–maximization
`threshold(dark, bright)`	the threshold of the best mean fidelity and its errors
`g2_zero`, `heralded_alpha`, `visibility`, `correlation`, `chsh`	coincidence quantities (section 21)

Table 8: The analysis module.

16 Self-tests, verification and troubleshooting

16.1 The board test

`make test-hw BOARD_HOST=<address>`, run in the repository's `counter` directory (no Vivado needed), copies `scripts/test_hw.py` to the board and runs it there against the loaded design. Without any wiring it tests, on quiet inputs and with the test emitter: the binomial statistics of the emitter, the channels' dead time against a simulation, the pairs and the split source, RATE records (exact intervals at 10 ms, none lost at 10 µs), the alarm, the monitors, the shots with the emitter's table, the sums and swaps, the sweep checks, the gate-delay scan, SHOT records up to 2 million per second, the decisions against every SHOT record, the analog channels (through the emitter, and their noise), and both coincidence units against a cluster simulation. It looks for wiring and tests what it finds: an output jumpered to a count input (the patterns, echoes, heralds, decision pulses and the latency), a cable from OUT1 to IN1 (the analog channels' pulses), a 10 MHz sine on IN2. It prints a line per check and saves its results as JSON on the board.

16.2 How the design was verified

- A **bit-exact model** (`control/counter/model.py`) follows the FPGA's registers clock by clock; it is checked against simple event-level definitions (the dead time, the clusters, the discriminator) on random events.
- The **simulation benches** (`make sim`) drive the RTL with vectors from the model and compare every register read, every record word and the analog channels' events clock by clock.
- **Builds** are checked for timing and resources; the design builds reproducibly from its sources.
- The **board tests** above, after every step of the design's development, twice.
- A **session with a second board and an oscilloscope** (below).

The dated records are in `docs/VERIFICATION.md`.

16.3 The session with awg and an oscilloscope

On 2026-11-05 counter (192.168.0.102) was wired to a second Red Pitaya running `awg` (192.168.0.104) with 20 cm jumper wires: awg's digital outputs D0–D3 to C1, C2, TRIG and SEQ, counter's O1 to awg's event input EVENT0, counter's O4 to C8 (neighbouring pins), the grounds joined; OUT1

cabled to IN1 (0.5 m) and a 10 MHz sine on IN2 as before. An oscilloscope (Siglent SDS2504X Plus, 1 GSa/s per channel) was read over the network; its probes sat on free pins carrying copies of the wired signals. The results:

- **exact counts**: awg's pattern played 8 points per sweep, 512 ns apart, with $k + 1$ pulses on C1 and $8 - k$ on C2 at point k: 39075760 shots in 20 s, 4884470 at each point, every per-point sum exact (351681840 counts), no bad sweep; with one TRIG left out per sweep, all 1222473 sweeps flagged;
- **crosstalk**: with the default 24 ns TRIG hold-off, C2's pulses on the jumper beside TRIG's made 2 spurious TRIGs per million (a few ppm off in the sums, some sweeps flagged); a 450 ns hold-off removed them (section 3.4);
- **latency**: 101–109 ns, 105.0 ns on average, from the gate's end; early mode 105.5 ns after the photon; the oscilloscope 103.5 ns (section 9.3);
- **the preselection loop** through awg: 468 ns from the gate's end to awg's output, 356 ns at awg's shortest digital delay (section 24.6);
- **the wired checks** of the board test, run for the first time: exact (after three faults in the test script itself were fixed);
- **the analog alignment**: corrected by 15.5 ns and verified within 0.5 ns (section 14.2).

Its receipt is `docs/data/board-1.0-awg-scope-session.json`.

16.4 The tests on awg's edges

On the evening of 2026-11-05, with the same wiring, awg's patterns served as sources whose every event is known, so that every count has an exact expected value:

- **exactness on random patterns**: 7 random patterns of 64 shots (four gates with a delay scan, two decisions, a plain and a gated coincidence unit), 2000 passes each: 896 000 shots and 129 million edges. Five were exact in every sum, record and cluster. In two, a false TRIG or SEQ edge from crosstalk (section 3.4) moved some shots. Each of the 256 000 SHOT records of those two equalled the counts computed at its own TRIG's time, 78 records at false TRIG times included, and so did the points renumbered by a false SEQ. So counter computed exactly what its inputs gave it;
- **the 1 ns boundaries**: gates are half-open (adjacent gates share no event and lose none), dead times, windows and unit 2's delay act to the nanosecond, and the fraction at a boundary measures the edges' offsets to tens of picoseconds (section 14.4);
- **latencies** of the echo, the herald and the decision through O4 → C8 (table 5);
- **accidentals and a delay scan** on independent backgrounds with correlated pairs: exact away from the window's edges, CAR 2.01 (figure 16);
- **dead time against a known rate**: the counts exact in 25 runs, and the textbook correction's limit for a non-Poisson source (figure 5);
- **repeat until success** through awg: the geometric distribution (figure 13);
- **records to a PC**: SHOT records without loss up to 1.5 million per second (30 MB/s).

The same evening, timetagger and feedback were loaded on the same board for the guide's tests (`guides/combining`). Overnight, awg's sweep of 64 shots ran for 12 hours at 244 000 shots per second: all 10 546 874 944 SHOT records (214 GB over the network) were checked on the PC as they arrived, and none was wrong or lost; the per-point sums were exact at every minute, and the coincidence totals, channel totals and shots exact at the end. C10 counted a GNSS reference meanwhile (figure 20; `docs/VERIFICATION.md`). The receipt is `docs/data/board-1.0-awg-tests-2026-11-05.json`.

16.5 The tests of the next day

On 2026-11-06 two more jumpers carried awg's D5 to NEXT and D7 to GATE, and a cable carried awg's OUT1 to IN1 (counter's own OUT1 unplugged); every wire was first checked with static levels:

- **NEXT**: points advanced by NEXT (4 points, 5 shots each): exact in 3 of 3 runs, 1.2 million shots. Two NEXT pulses 16 ns apart counted once with the default 24 ns hold-off, on rising and on falling edges, and twice with 8 ns (the points skipped exactly as predicted);
- **GATE**: exact in 27 settings; the filter's $(W - 1)$ ns and the delay add to the nanosecond; active low counts the complement (section 8.2);
- **one word**: NEXT, SEQ and TRIG in one word follow their slots; at a slot boundary the edges' sub-ns skew decides (section 7);
- **analog pulses from awg**: the staircase, the plateau curve, the time walk and NIM-like pulses (section 5.4);
- **crosstalk**: C2 and C8 carried no signal and counted nothing in any run;
- **totals of two channels**: `counters()` reads the channels one after another, tens of microseconds apart; at a megahertz their totals differ by a few counts between two readings. RATE records hold all channels at one instant.

Receipt: `docs/data/board-1.0-awg-tests-2026-11-06.json`.

16.6 Troubleshooting

Symptom	Causes and remedies
No counts	The channel is not enabled; the wrong pin (`_P` and `_N` are neighbours); the pulse stays below 2.0 V (a 50 Ω load on a source that cannot drive it, a long thin wire); the pulse is narrower than 1–2 ns; no ground connection. `dev.pins(clear=True)` shows each input's level and whether an edge came.
Too many counts, double counts	Reflections on an unterminated cable (terminate 50 Ω sources at the board); ringing; afterpulses of the detector. Look at the pulse with an oscilloscope; set the channel's dead time to cover the ringing or the afterpulses (section 4.2); compare the `rejected` count.
Fewer counts than expected at high rates	Dead time (the channel's and the detector's): correct with equation (1); the detector's saturation; a level that sags at high rates (AC coupling, a weak source).
Missed TRIGs	The gates (the longest delay plus width) last longer than the shot spacing; TRIG bounces or rings: raise TRIG's hold-off; crosstalk from neighbouring wires.
Bad sweeps	A TRIG lost or added: glitches on TRIG (hold-off, coax), a sequencer that skips; SEQ in the wrong place (it must come before the sweep's first TRIG, or in the same word).
A few points' sums slightly off, no missed TRIG	A false TRIG edge shortly before a real one, whose TRIG then fell into the hold-off (section 7.2): look for irregular TRIG spacings in the SHOT records; a false SEQ shows as bad sweeps. Crosstalk in jumper bundles hits lines idling high as well as low (section 3.4): separate the wires, use coax.
The gate misses the signal	Put the gate's window on an output beside the detector's pulse on an oscilloscope; scan the gate's delay (section 8.4). With C9 or C10 on, every gate gets the analog lag added.

Symptom	Causes and remedies
A decision never true	The gate's counts against `low`, `high` (16-bit saturated counts); `gate_a` the wrong gate; the logic function; look at `dev.decisions()` and the SHOT records.
Few coincidences	The channels' relative delay: scan it with unit 2 and correct the channels' delays (section 11.2); a window shorter than the detectors' jitter.
Many accidentals	A window wider than needed; high singles rates (eq. (5)).
Lost records	The PC or network cannot keep up: a longer RATE interval, fewer SHOT records, a client on the board; the frames' `lost` field and `records_dropped` say how many.
Every negative pulse also counted with positive polarity	The pulse's overshoot after it: set the polarity to the pulse's sign, the threshold above the overshoot (section 5.4).
Two channels' totals disagree by a few counts	`counters()` reads the channels microseconds apart: compare RATE records (one instant), or read with the source stopped.
An analog channel counts noise	The threshold too close to the baseline; no hysteresis; the HV jumper (20 V range: a coarse threshold); a missing 50 Ω termination; ground loops. `dev.analog()` shows the input's level and noise.
Rates drift	The source, the laser or the alignment; a reference window or a second detector measures it; the crystal's temperature is a fraction of a ppm per kelvin (section 14.5).
The board does not answer	Its network address (DHCP); the service: restart it, or load the design again (`LOAD counter`).

17 Specifications

Design values unless marked *measured* (on the board used for this manual, with the test emitter, a second Red Pitaya running awg, or an oscilloscope); values marked *not measured* are expectations only.

Quantity	Value
Count inputs	C1–C8 on E1 (DIO0_P–DIO7_P), 3.3 V LVCMOS, pull-down, not 5 V tolerant; sampled every 1 ns; pulses ≥ 2 ns always seen; rising or falling edges
Analog count inputs	C9, C10 on IN1, IN2 (SMA, 1 MΩ ∥ 10 pF, ±1 V LV or ±20 V HV, 60 MHz): a Schmitt trigger at a threshold in volts with a hysteresis (≥ 1 code, 0.14 mV on LV), either polarity, the crossing interpolated to 1 ns; pair resolution about 16 ns *(measured)*; noise 1.5 mV rms on LV *(measured)*
Dead time	non-paralyzable, 8 ns–65.535 µs in 1 ns steps per channel (at most 125 Mcount/s); rejected edges counted (32 bits)
Delay	0–255 ns in 1 ns steps per channel (less the analog lag while C9 or C10 is on)
Totals	48 bits per channel and coincidence pattern, never cleared (26 days at 125 Mcount/s)
Rates	from the totals and the core's clock, any interval
RATE records	every 10 µs–10 s (8 ns steps); exact intervals in every channel and pattern; 100 000/s to a PC without loss *(measured)*
Rate alarm	a set of channels below a rate in windows of 1 µs–2 s; status, records, an output

Quantity	Value
Rate monitors	OUT1, OUT2: offset + scale × rate, windows of 128 ns–134 ms; 0.0966 V per Mcount/s for a set 0.1 read back on IN1 *(measured)*; or 8 ns test pulses of a set amplitude
Control inputs	TRIG, GATE, NEXT, SEQ on DIO0_N–DIO3_N, 1 ns; hold-offs 8 ns–65.535 µs; GATE's minimum width 1–8 ns and delay 0–255 ns
Shots	TRIG from the pin, the internal timer (64 ns–34 s, 8 ns steps) or software; a shot lasts its gates and at least 64 ns (at most 15.6 million shots per second); TRIGs meanwhile counted as missed; 2 million shots per second with exact sums *(measured)*
Points and sweeps	1–1024 points: per shot, per NEXT or fixed; sweeps checked (TRIGs = points) in per-shot mode
Gates	4 per shot: delay and width 0–4.29 s, delay step per point 0–16.7 ms (1 ns); a channel set each; timed, or timed and GATE
Per-point sums	per point: 4 counts, shots, D1, D2, $\sum n_1^2$; two banks, swaps between shots; 64-bit totals in the service
Decisions	2 per shot: a count window on each of two gates (16-bit counts), any of 16 logic functions; a level, pulses (8 ns–2.048 µs) or early; latency from the gate's end to the pin 101–109 ns, 105.0 ns on average *(measured)*
Outputs	O1–O4 on DIO4_N–DIO7_N, 3.3 V LVCMOS, 12 mA, 2 ns resolution, high impedance until enabled; a channel's echo 92.5 ns after its edge, a herald 121.5 ns + W after the cluster's first event *(measured)*
Coincidences	2 units of up to 4 channels (any of C1–C10); timetagger's clusters; window 8–1023 ns (1 ns); 15 exclusive patterns' 48-bit totals; optionally inside G1; unit 2 delays one channel by 0–16.383 µs; heralds on an output
Records	SHOT (5 words), RATE (13 + 15 per unit), SWEEP, SWAP, RUN into a ring in DDR, served on TCP port 5026; 2 million SHOT records per second to a client on the board without loss, 1.5 million (30 MB/s) to a PC over Gigabit Ethernet (at 2 million 2.4 % dropped) *(measured)*
Test emitter	4 lanes, 0–125 Mevents/s each (one trial per 8 ns word), into any channel, an output or OUT1/OUT2; pairs (0–248 ns), split; bright/dark shots from a 1024-entry table with a decay
Timebase	the board's 125 MHz crystal: 5.66 ppm slow against a GNSS-disciplined 10 MHz *(measured)*; about −0.13 ppm/K
Analog alignment	95 ns on the board used here, within 0.5 ns for OUT1's 8 ns pulses *(measured)*
Detectors	SPAD modules, PMTs, SNSPDs: *not measured* (no detector was used in any test)
FPGA	Zynq 7010: 11 335 LUTs (64 %), 3 849 slices (87.5 %), 23 block RAMs (36 kb), 40 DSP48; timing met with +0.235 ns of slack at 125 MHz (build 0x894f27b0)

Part II
Theory

18 Counting statistics

Counts scatter. Even with a perfectly steady light source the number of photons detected in a gate changes from shot to shot, and the error bars of every result in this manual come from that scatter.

This section collects what is needed to predict it, to recognize when the counts do not behave as predicted, and to propagate it into ratios, contrasts and fits.

18.1 Poisson statistics

If photons are detected independently of one another at a steady rate R, the number n detected in a window of length T follows the ***Poisson distribution***

$$P(n) = \frac{\mu^n}{n!}\,\mathrm{e}^{-\mu}, \qquad \mu = RT, \qquad \langle n\rangle = \mu, \qquad \mathrm{Var}(n) = \mu. \tag{7}$$

The standard deviation is $\sqrt{\mu}$: the ***shot noise***. A measurement of N counts has a relative uncertainty of $1/\sqrt{N}$, whatever the gate, the rate or the number of shots that gave them: 100 counts are known to 10 %, 10^4 counts to 1 %, 10^6 to 0.1 %. In a pulsed experiment with μ counts per shot (an NV centre gives $\mu \approx 0.03$–0.1 in a 300 ns window), N_s shots give $N_s\mu$ counts and the mean per shot is known to

$$\sigma_{\bar n} = \sqrt{\mu/N_s}. \tag{8}$$

Poisson statistics hold for light whose intensity is steady (a laser well above threshold, a lamp averaged over a gate much longer than its coherence time) detected with an efficiency below one, and for dark counts. They are the reference against which everything else is measured [6, 7].

18.2 The Fano factor: sub- and super-Poissonian counts

The ***Fano factor*** $\mathcal{F} = \mathrm{Var}(n)/\langle n\rangle$ compares the scatter with Poisson's:

- $\mathcal{F} = 1$: Poisson;
- $\mathcal{F} > 1$, ***super-Poissonian***: the rate itself fluctuates from shot to shot. Laser intensity noise, a blinking emitter, thermal light (bunched photons), and above all a **mixture of states**: a quantum system that is bright in some shots and dark in others gives a count distribution much broader than Poisson's (section 20);
- $\mathcal{F} < 1$, ***sub-Poissonian***: the events are more regular than random. Antibunched light (a single emitter cannot emit two photons at once, Mandel [8]), and, in any counter, **dead time**: after each count the next one is forbidden for τ, which regularizes the stream.

counter measures $\mathcal{F}$ per point from the per-point sums: with N_s shots, $\mathcal{F} = \left(\sum n_1^2/N_s - \bar n_1^2\right)/\bar n_1$. Over the shots of a point it shows at once whether the counts are Poissonian or contain a mixture.

The test emitter is sub-Poissonian by construction: one trial per 8 ns word with probability p gives binomial counts, $\mathcal{F} = 1 - p$. Figure 21 shows its counts in 10 µs intervals of RATE records: 0.985 at 1 Mevents/s, 0.855 at 10 Mevents/s and 0.436 at 50 Mevents/s with 8 ns of dead time, and 0.438 at 10 Mevents/s with 50 ns of dead time: the dead time alone makes a Poisson-like stream strongly sub-Poissonian.

18.3 Dead time and the statistics of counts

A ***non-paralyzable*** dead time τ (counter's, and most detectors' to a good approximation) turns a Poisson stream of rate R into a stream of rate $m = R/(1 + R\tau)$, equation (1): each count is followed by τ of blindness and then, on average, $1/R$ of waiting. The intervals between counts are τ plus an exponential; their relative variance gives, for windows much longer than τ, [9, 10]

$$\mathcal{F} = \frac{1}{(1 + R\tau)^2} = (1 - m\tau)^2. \tag{9}$$

A ***paralyzable*** dead time restarts with every event, counted or not: $m = R\,\mathrm{e}^{-R\tau}$, which has a maximum at $R = 1/\tau$ and then falls: two values of R give the same m, and a very bright source can look dark. Detectors with an active quench are close to non-paralyzable; passively quenched ones and photomultipliers with slow electronics behave partly paralyzably. For counting:

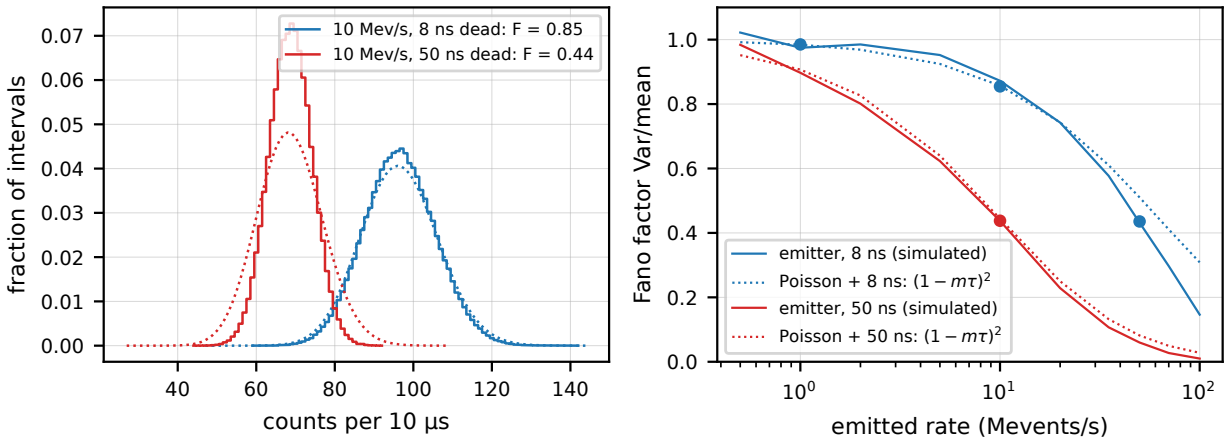


Figure 21: Measured (2026-11-05). Left: the distribution of C1's counts in 10 µs intervals (RATE records, 200559 intervals or more per run) for the test emitter at 10 Mevents/s with 8 and 50 ns of dead time; dotted: Poisson distributions of the same means. Right: the Fano factor against the emitted rate (points: measured; lines: the emitter's events simulated through the dead time; dotted: a Poisson source through a non-paralyzable dead time, $(1-m\tau)^2$).

- keep $m\tau$ small (below about 0.1, a 10 % correction) where accuracy matters;
- when two dead times are in series (the detector's and the channel's), set the channel's longer than the detector's: the stream then has one non-paralyzable dead time, the channel's, and equation (1) corrects it exactly;
- in pulsed experiments a detector clicks at most once per dead time: if a light pulse shorter than τ brings μ detectable photons, the click probability is $1-\mathrm{e}^{-\mu}$, not μ (***pile-up***). Keep $\mu \ll 1$ per pulse and detector, or split the light onto several detectors.

18.4 Signal, background and the signal-to-noise ratio

A gate of length T collects signal at rate S and background (stray light, dark counts) at rate B. If the background is measured in a second gate of the same length and subtracted, the signal $\hat{S}T = n_1 - n_2$ has variance $(S+B)T + BT$, and

$$\mathrm{SNR} = \frac{S\sqrt{T}}{\sqrt{S+2B}}. \tag{10}$$

With background known well from a long measurement the 2 becomes 1. The gate's length is a compromise: a gate wider than the signal collects background without signal; a narrower one loses signal. For a signal that decays (an NV centre's spin-dependent fluorescence, a lifetime) the optimum is where the signal's rate at the gate's edge has fallen to about the background's noise; section 23.3 works it out for an NV centre.

18.5 Ratios, contrasts and their errors

Many results are ratios: a contrast $C = 1 - S/R$ between a signal and a reference window, a normalized fluorescence n_1/n_2, a probability d_1/shots. For Poisson counts S and R (totals over all shots),

$$\frac{S}{R} \pm \frac{S}{R}\sqrt{\frac{1}{S}+\frac{1}{R}}, \qquad \sigma_C = \frac{S}{R}\sqrt{\frac{1}{S}+\frac{1}{R}}, \tag{11}$$

which `analysis.contrast(signal, reference)` returns per point. A probability estimated from k successes in N shots has the binomial error $\sqrt{p(1-p)/N}$. Fits (`analysis.fit_rabi` and the others) take these errors as weights and return parameter errors from the fit's covariance; a χ^2 per degree of freedom near 1 shows that the error bars describe the scatter.

18.6 Beyond the counts: drifts

Shot noise falls as $1/\sqrt{N}$ only as long as nothing else changes. Over minutes and hours the laser power, the alignment, the temperature and the sample drift; a scan whose points are measured one after another then turns the drift into a false signal. Two habits keep it out: interleave the points (a sweep through all points, repeated, rather than one point after another, which is how counter's per-shot points work), and normalize by a reference window measured in the same shot (section 8.3).

19 Detectors, discriminators and dead time

The counts are only as good as the clicks. This section describes what single-photon detectors add to the photons' own statistics, and what a discriminator does to an analog detector pulse. The reviews Hadfield [11], Eisaman et al. [12] and Natarajan et al. [13] cover the detectors in depth; Hamamatsu Photonics K.K. [14] the photomultiplier.

19.1 The figures of merit

Detection efficiency the probability that a photon arriving at the detector gives a click: 20–70 % for silicon SPADs in the visible (a strong function of wavelength), 10–40 % for photomultipliers (the photocathode's ***quantum efficiency*** times the collection efficiency), above 90 % for the best SNSPDs. It multiplies every rate but leaves Poisson light Poissonian.

Dark counts clicks without light: thermal generation in a SPAD (tens to thousands per second, falling with cooling), thermionic emission from a photocathode (reduced by cooling; the rest from radioactivity and cosmic rays), below 100 per second for SNSPDs. They are a Poisson background, B in equation (10): gates reduce them in proportion to the time the gates are open.

Afterpulses false clicks that follow a real one (section 19.2).

Timing jitter the spread of the delay from a photon's arrival to the click's edge: tens of picoseconds (SNSPD) to a few hundred (SPAD modules). At counter's 1 ns resolution it matters only for coincidence windows at their 8 ns minimum and for gate edges.

Dead time the time after a click during which the detector is blind: 20–50 ns for actively quenched SPAD modules, a few to tens of ns for photomultiplier electronics, tens of ns (the reset time) for SNSPDs (section 18.3).

Saturation at high rates the dead time loses counts, the module may oscillate, and some detectors latch or overheat. Module makers give a ***correction factor*** $1/(1-\tau m)$ for the counted rate m, equation (1); at 10% of the maximum rate it is already about 1.1.

19.2 Afterpulses

In a SPAD some of the avalanche's charge is trapped in defects and released later, triggering a new avalanche; in a photomultiplier, ions from the residual gas hit the photocathode. Either way a click is followed, with a probability of a fraction of a percent to a few percent, by a false click tens of nanoseconds to microseconds later. Afterpulses

- add counts in proportion to the signal (they look like extra efficiency, but a nonlinear one: the fraction depends on the dead time);
- make the counts super-Poissonian and fake bunching in the correlation of a detector with itself; this is why intensity correlations ($g^{(2)}$) are measured with two detectors behind a beam splitter, whose afterpulses are uncorrelated (section 21.2);
- are cut by a dead time longer than most of their delays: the channel's dead time can be set far longer than the detector's own (up to 65.5 µs).

counter measures a detector's afterpulsing without any other instrument: at a steady, moderate light level, step the channel's dead time from the detector's own upwards and record the counted rate. Equation (1) predicts a small, known loss; any loss beyond it is the afterpulses that fell within the added dead time, and its slope against the dead time is their distribution in delay.

19.3 Discriminators

A photomultiplier's or an SNSPD's pulses are analog: their heights vary (a photomultiplier's gain fluctuates from photoelectron to photoelectron), and noise rides on them. A ***discriminator*** turns them into clicks: it fires when the voltage crosses a threshold. Its setting decides what is counted:

- the ***pulse-height distribution*** of a photomultiplier has a peak of single photoelectrons and, below it, a rising tail of noise and small pulses; the threshold belongs in the valley between them. Counting against the threshold gives the integral of the distribution: a ***plateau*** where the counts hardly change with the threshold (figure 6). A threshold on the plateau makes the count rate insensitive to the gain's drift; one on the slope makes it follow every change of high voltage or temperature;
- raising the threshold above the valley does not remove dark counts usefully: a photomultiplier's dark pulses are mostly single photoelectrons from the cathode, as tall as the signal's. It removes electronic noise and the ringing of the pulses;
- Gaussian noise of standard deviation σ crosses a level T upwards at a rate that falls as $\mathrm{e}^{-T^2/2\sigma^2}$ (Rice's formula): at 5σ a 60 MHz bandwidth gives a fraction of a crossing per second, at 10σ none in the age of the universe. counter's idle inputs have about 1.5 mV of noise: a threshold of 20 mV is far beyond it;
- the ***hysteresis*** keeps a slow or ringing crossing from firing twice: the trigger re-arms only below $T - H$ (figure 6: at 20 mV with 20 mV of hysteresis each test pulse fired about four times on its ringing, from 0.1 V on once).

A ***leading-edge*** discriminator, like counter's, gives a crossing time that depends on the pulse's height (***time walk***, a fraction of the rise time); a ***constant-fraction*** discriminator fires at a fixed fraction of each pulse's height and removes it. For counting, walk does not matter; for coincidences at the 10 ns level it does not either; for time tagging at picoseconds it does.

19.4 Gating the detector

Some detectors can be switched on and off (a SPAD module's gate input, a gated InGaAs SPAD). Switching a detector on is not instantaneous: the first tens of nanoseconds after enabling can give spurious clicks or lose timing, and module makers advise discarding the pulses within one dead time of the enable. Gating the counts (counter's gates) leaves the detector running and has no such transient: prefer it, and gate the detector only to protect it from bright pulses (a laser pulse that would blind or damage it), with counter's gate opening a dead time later.

20 Reading out quantum states by counting

Most quantum systems in optics labs are read out by fluorescence: a laser drives a cycling transition that scatters many photons if the system is in one state (***bright***) and few or none if it is in another (***dark***). Trapped ions, neutral atoms and some solid-state defects at low temperature scatter enough photons to tell the state in a single shot; an NV centre at room temperature does not, and its spin is read out on average over many shots. Both are counting.

20.1 Two states, two Poisson distributions, a threshold

In a detection window the bright state gives a Poisson distribution of counts with mean μ_B, the dark state one with mean μ_D (background and dark counts). A ***threshold*** n_c calls a shot bright if

$n \geq n_c$. The two errors are

$$\epsilon_D = P(n \geq n_c \mid \text{dark}), \qquad \epsilon_B = P(n < n_c \mid \text{bright}), \tag{12}$$

and the (mean) ***readout fidelity*** is $F = 1 - (\epsilon_D + \epsilon_B)/2$. `analysis.threshold(mu_D, mu_B)` finds the n_c that maximizes it. The means come from calibration runs (the system prepared bright, then dark) or, when the states cannot be prepared separately, from a fit of a mixture of two Poisson distributions to the histogram of a mixed preparation, `analysis.two_poisson(h)` (expectation–maximization). The histogram comes from SHOT records:

```
dev.load_preset("ion-detection")                          # G1 500 us, SHOT records, D1 = G1 >= 3
# ... collect SHOT records (section on records) into `records`
h = analysis.count_histogram(records, gate=1)
mix = analysis.two_poisson(h)                             # {"dark", "bright", "bright_fraction"}
th = analysis.threshold(mix["dark"], mix["bright"])       # {"threshold", "fidelity", "dark_error", ...}
    ↪
dev.apply({"decisions": {"1": {"low_a": th["threshold"]}}})
```

Figure 22 (left) shows this with the test emitter: half of the shots bright, a 2 µs window, 423491 shots. The fit gives $\mu_D = 1.18$ and $\mu_B = 9.60$, the threshold 5 and a fidelity of 97.75 %; D1 on the board, set to the same threshold, agreed with every shot's count (0 wrong).

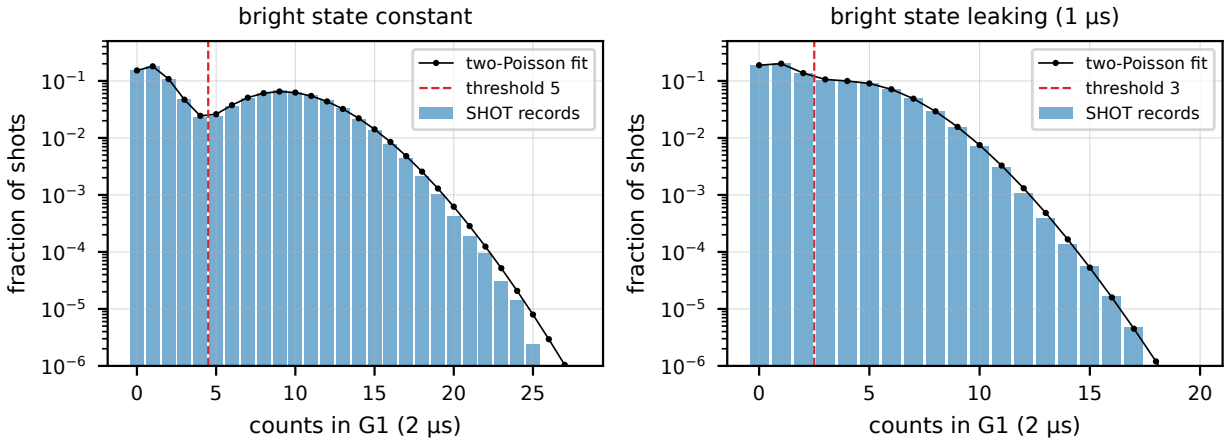


Figure 22: Measured (2026-11-05). Count histograms of G1 (2 µs) from SHOT records, the test emitter bright in half of the shots, with the two-Poisson fit and its best threshold. Left: the bright state's extra rate constant over the window; right: decaying with 1 µs, as an ion that is pumped into a dark state during detection: the bright distribution is no longer Poisson, it leaks into the dark one, and the fidelity falls from 97.75 to 89.02 %.

20.2 When the states are not Poissonian

Real bright and dark states are rarely pure Poisson distributions:

- **leakage**: off-resonant scattering pumps a bright ion into a dark state during detection, or a dark one into the bright cycle; an NV centre's charge state can flip under the readout laser. A leaking bright state gives a long tail towards zero (figure 22, right), and its mixture fit and threshold are then only approximations;
- **non-Poissonian background**: cosmic rays and radioactivity give photomultipliers bursts of dark counts; laser intensity noise broadens the bright distribution;
- **detector effects**: dead time and pile-up compress the bright distribution at high rates; afterpulses add a tail.

Look at the histogram on a logarithmic scale: the tails show these effects long before the fidelity does. Myerson et al. [15] discuss them for trapped ions.

20.3 The detection time

A longer window separates the two distributions better ($\mu_B - \mu_D$ grows as T, their widths as $\sqrt{T}$) until leakage takes over: then a longer window gives more time to leak. The best window balances the two; figure 27 (section 24.2) computes it for a trapped ion. With counter, measure it rather than compute it: G1–G4 as four consecutive windows of the detection (or one gate with a scanned width over points) give, from the same shots, the histograms of four detection times.

20.4 Adaptive and time-resolved readout

Most bright shots reach the threshold long before the end of the window, and a shot with many counts early is almost certainly bright. Two refinements use this:

- **adaptive readout**: stop the detection as soon as the decision is certain. counter's `early` decision mode raises a pin at the threshold-th count; a sequencer that branches on it ends the detection early in most bright shots, which saves time and reduces the leakage of the remaining ones [16, 17];
- **time-resolved discrimination**: the times of the photons carry information that the total count does not (a dark state pumped into bright gives its photons late, a bright state that leaks gives them early). With the four gates as consecutive bins of the window, each SHOT record holds a coarse arrival-time histogram of the shot, and a maximum-likelihood decision on the PC uses it [15].

20.5 Correcting readout errors

With known error rates, a measured probability of "bright" p_m (D1 true over shots, per point) corrects to the probability of the bright state

$$p = \frac{p_m - \epsilon_D}{1 - \epsilon_D - \epsilon_B}, \tag{13}$$

with errors from p_m's binomial error and the calibration's. This is how a Rabi oscillation of an ion recovers its full contrast.

20.6 Averaged readout: the NV centre

At room temperature an NV centre's electron spin is read out by the difference of its fluorescence in the first few hundred nanoseconds of a green laser pulse: the $m_s = \pm 1$ states fluoresce about 30 % less until the laser has repolarized them into $m_s = 0$ [18]. With $\bar{n} \approx 0.03$–0.1 photons per shot in that window, a single shot says almost nothing; the spin's population is the average over N_s shots. The ***readout contrast*** C and the counts set the spin's uncertainty per shot,

$$\sigma_P \approx \frac{1}{C}\sqrt{\frac{2}{\bar{n}}}\,\frac{1}{\sqrt{N_s}} \tag{14}$$

(for $C \ll 1$): with $C = 0.3$ and $\bar{n} = 0.05$, $\sigma_P \approx 21/\sqrt{N_s}$, so 10^5 shots give a population to about 7 %. Improving $\bar{n}$ (collection optics) and C (the window, section 23.3) is worth more than anything else [19–21]. Single-shot readout of NV spins needs resonant excitation at low temperature or a nuclear-spin memory [22]; its charge state, NV^- against NV^0, can be read in single shots at room temperature with weak orange light, the basis of charge-state preselection (section 23.6).

21 Coincidences and correlations

21.1 Accidental coincidences

Two detectors that click independently, at rates R_1 and R_2, still click within a window of each other now and then: the ***accidental*** coincidences. With counter's clusters, a C1 event and a C2 event

coincide if they are at most W apart in either order; times are whole nanoseconds, so $2W/\mathrm{ns} + 1$ values of the difference count. For $R_i W \ll 1$ the accidental rate is

$$r_{\mathrm{acc}} = R_1 R_2 \,(2W + 1\,\mathrm{ns}), \tag{15}$$

equation (5). Measured with the test emitter (figure 15): 20143 per second against 20220 from the singles. Corrections enter when the rates are high: an event belongs to one cluster only, so a cluster that has just closed cannot take a partner (the dip beside a strong peak), and dead times remove partners at short delays. Three-fold accidentals of independent channels scale as $R_1 R_2 R_3$ times the square of the window.

The ***coincidence-to-accidental ratio*** (CAR) is the true coincidences over the accidentals in the same window, the figure of merit of a photon-pair source: it falls as the pump power rises (true pairs grow as the power, accidentals as its square) and as the window widens. Measure the accidentals with unit 2's delayed channel rather than from equation (15): the delayed count needs no assumption about steady rates and dead times.

21.2 The second-order correlation $g^{(2)}$

The normalized intensity correlation of light,

$$g^{(2)}(\tau) = \frac{\langle I(t)\, I(t+\tau)\rangle}{\langle I\rangle^2}, \tag{16}$$

is measured with two detectors behind a 50:50 beam splitter (Hanbury Brown and Twiss, Hanbury Brown and Twiss [23]): the coincidences at delay τ divided by those of independent light of the same rates. Coherent light has $g^{(2)} = 1$ at all delays, thermal light $g^{(2)}(0) = 2$ (bunching), a single emitter $g^{(2)}(0) = 0$ (***antibunching***: it cannot emit two photons at once). With counter,

$$g^{(2)}(0) \approx \frac{N_{12}}{N_{\mathrm{acc}}}, \tag{17}$$

unit 1's coincidences over unit 2's (delayed far beyond the correlation time), in the same time: `analysis.g2_zero(n12, accidentals=acc)`. What counter measures is $g^{(2)}$ averaged over the window: for a single emitter with $g^{(2)}(\tau) = 1 - \mathrm{e}^{-|\tau|/t_e}$ (lifetime t_e),

$$\bar{g}^{(2)}_W = 1 - \frac{t_e}{W}(1 - \mathrm{e}^{-W/t_e}), \tag{18}$$

so an NV centre ($t_e \approx 12\,\mathrm{ns}$) gives 0.27 with $W = 8\,\mathrm{ns}$, a quantum dot ($t_e \approx 1\,\mathrm{ns}$) 0.88: the window must be short against the lifetime to see antibunching, and counter's 8 ns minimum reaches only long-lived emitters. Background light of fraction $1-\rho$ of the singles raises it further, $g_{\mathrm{meas}} = 1 - \rho^2(1-g)$ [24]. For pulsed excitation the photons come in pulses: the coincidences of one pulse against those of the next (unit 2 delayed by one period) give $g^{(2)}(0)$ directly, independently of the lifetime, as long as the period exceeds the window's span. For the full shape of $g^{(2)}(\tau)$ with sub-nanosecond resolution, use a time tagger.

21.3 Heralded single photons

A photon-pair source sends one photon to a herald detector (H) and the other to a beam splitter with two detectors (T, R). Grangier, Roger and Aspect [25] defined the anticorrelation parameter

$$\alpha = \frac{N_H\, N_{HTR}}{N_{HT}\, N_{HR}}, \tag{19}$$

the heralded $g^{(2)}(0)$: 1 for classical light, 0 for an ideal single photon; teaching labs reach 0.01–0.1 [26]. counter's unit 1 on H, T, R counts all of them at once (the `heralded-g2` preset), `analysis.heralded_alpha`. The ***heralding efficiency*** (Klyshko efficiency) $\eta_T = N_{HT}/N_H$ is the probability that a heralded photon reaches detector T: an absolute calibration of T's detection efficiency, without a calibrated source [27].

21.4 Correlations and Bell's inequality

With two polarization analysers, each with two detectors (Alice: ±, Bob: ±), the coincidences $N_{++}, N_{--}, N_{+-}, N_{-+}$ at analyser angles a, b give the correlation

$$E(a,b) = \frac{N_{++} + N_{--} - N_{+-} - N_{-+}}{N_{++} + N_{--} + N_{+-} + N_{-+}} \tag{20}$$

(`analysis.correlation`), and four settings give the CHSH parameter

$$S = E(a,b) - E(a,b') + E(a',b) + E(a',b'), \tag{21}$$

bounded by 2 for local hidden variables and reaching $2\sqrt{2}$ for a maximally entangled state at suitable angles [28, 29] (`analysis.chsh`). One unit with the four channels counts all four coincidences at once (the `bell` preset); unit 2, delayed, counts their accidentals, which should be subtracted before computing E when the singles rates are high. The ***visibility*** of a correlation fringe, $V = (N_{\max} - N_{\min})/(N_{\max} + N_{\min})$, must exceed $1/\sqrt{2}$ for a violation [30].

21.5 Errors of coincidence quantities

Coincidence counts are Poisson: $\sqrt{N}$. A ratio such as $g^{(2)}(0) = N_{12}/N_{\mathrm{acc}}$ has the relative error $\sqrt{1/N_{12} + 1/N_{\mathrm{acc}}}$; since unit 2 counts its accidentals in the same time and window, the measurement time sets both. For S the four correlations' errors add in quadrature; each follows from its four counts. A violation is quoted in standard deviations: $(S-2)/\sigma_S$.

22 Inside the FPGA

This section describes how the FPGA does what Part I describes: enough to understand every timing and limit, and to read the sources (`counter/src/ct_*.v`) and the bit-exact model (`counter/control/counter/model.py`), which follow it register by register. `docs/THEORY.md` section 2 is the formal contract of the same; `docs/REGISTERS.md` the register map (appendix D).

22.1 Words, slots and lanes

Everything runs on the board's 125 MHz sample clock. A ***word*** is one clock period, 8 ns, holding eight ***slots*** of 1 ns (bit 0 the earliest); a time is $t = 8n + j$ ns for slot j of word n, counted from the core's reset. A ***lane*** carries the events of one input after its conditioning (C1–C10, TRIG, NEXT, SEQ). Because every lane's dead time or hold-off is at least 8 ns, **a lane holds at most one event per word**: an event is a valid bit and a 3-bit slot, a delay by whole words keeps one event per word, and every block below handles at most one event per lane per clock. This is the design's central simplification; it costs a minimum dead time of 8 ns, shorter than any detector's.

22.2 The samplers

Each of the twelve input pins goes to an ISERDESE2, the FPGA's input deserializer, in ***oversampling*** mode [31]: clocked by 250 MHz in two phases 90° apart from a PLL, it samples the pin at four instants per 4 ns, every nanosecond. Two such 4-bit words make the 8-slot word of a 125 MHz clock; pin word n reaches the logic in clock $n + 4$. The sampling instants are fixed against the clock, so an edge's slot is exact to the nanosecond, with the edge's own jitter and the pins' buffer delays (taken out by the calibration, section 14).

22.3 Channels, dead time and delays

A channel (`ct_chan`) finds its edges by comparing each slot with the one before, across the word boundary (the previous word's last slot), after inverting the word for falling edges and ORing in the

test emitter's pulses. The dead time is a down-counter of the remaining blind time: in each word, the edges in slots still covered by it are masked by a ***thermometer*** (all slots below the remaining time), the first edge left is the event, and the counter restarts at the dead time minus the slots left in the word. Masked edges count as rejected. A delay of D ns is a slot shift by $D \bmod 8$ (with a carry into the next word) followed by $\lfloor D/8 \rfloor$ words in a shift register (SRLC32E). The totals are 48-bit counters in DSP48 blocks, one per lane, fed from the delayed lanes.

22.4 Shots and gates

The controls (`ct_ctl`) run TRIG, NEXT and SEQ through the same edge and hold-off logic and reach the shot logic two clocks before the channels' lanes of the same pin word, so a gate can open at its TRIG's own slot. The shot logic (`ct_shots`) accepts a TRIG when no shot is busy, latches the point, runs the timer and the sweep checks. Each gate (`ct_gate`) loads, at the TRIG, its start time ($t_T + d + i\,s$: the delay step times the point comes from a DSP48 kept with the point) and its width as word counters; in the start and end words a thermometer mask selects the slots inside $[t_{\text{start}}, t_{\text{end}})$; the masked events of the gate's channel set are counted (0–10 per word) into a 32-bit count. When the gate's last word has passed, its result (`fin`) goes to the decisions, the sums and the SHOT record.

22.5 Decisions and the latency

A decision (`ct_decide`) compares the two gates' counts (saturated to 16 bits) with its windows, looks up its table, and drives its pin. For an edge in pin word w that is the last a gate counts: the samplers deliver word w in clock $w + 4$; the channel and the delay take it to clock $w + 8$; the gate counts it in $w + 9$ and its result is complete two clocks later; the decision's registers change in clock $w + 12$; the output serializer (an OSERDESE2 on the 250 MHz clock) puts it on the pin within the next word, and the pins' buffers add a few nanoseconds. Twelve to thirteen clocks and the I/O give the measured 101–109 ns; the spread is the position of the gate's end within its word, since the decision is taken once per clock. The early mode compares the gate's running count with `low_a` every clock on the same path.

22.6 Per-point sums

The sums (`ct_sums`) are nine columns of 1024 32-bit words (G1–G4, shots, D1, D2, and the two halves of $\sum n_1^2$) in two banks of block RAM. At each shot's end the counting bank's words of the shot's point are read, added and written back (five clocks, all columns at once). The frozen bank is read by the processor through the register window, released, and zeroed (8 µs) before it can count again; a swap waits for a zeroed bank and happens between two shots' updates, so no shot is split. Before a 32-bit word could overflow, a swap is forced; if none is possible, the shot is added with saturation and flagged.

22.7 Coincidence clusters

A unit (`ct_coinc`) keeps the open cluster's state: open or not, its pattern, and the time left to its end ($t_0 + W$ minus the current word's start). In each word an event at slot s joins if s is within the time left; the events that do not join open the next cluster at their earliest slot (all of them in it), the old one closes. With $W \geq 8$ ns a word closes at most one cluster and opens at most one, so one clock handles any number of events. The closed cluster's pattern indexes a distributed RAM of 15 48-bit totals, read, incremented and written in one clock; the RATE records and the bus read it through two more ports. Unit 2's delayed lane goes through a slot shift and a ring of 4096 words in block RAM (up to 16.4 µs); the other lanes wait four clocks to keep their alignment.

22.8 The analog discriminators

`ct_analog` takes one 14-bit sample per clock from each ADC stream (as a 16-bit code), inverts it for negative polarity, and runs the Schmitt trigger (section 5.1). When it fires at sample $y_n \geq T$ after $y_{n-1} < T$, the crossing lies a fraction a/d of the way from sample $n-1$ to sample n, with $a = T - y_{n-1}$ and $d = y_n - y_{n-1}$ ($1 \leq a \leq d$); the event's slot is

$$j = \left\lceil \frac{8a}{d} \right\rceil - 1 = \left\lfloor \frac{8a - 1}{d} \right\rfloor , \tag{22}$$

computed without a divider by three compare-and-subtract steps ($8a - 1$ against $4d$, $2d$, d), one per clock. The event word appears five clocks after the sample, and the event's time lies 7–8 ns after the straight line's crossing; the analog lag A of section 14.2 contains this.

22.9 The test emitter

Each lane draws, per word, a 24-bit uniform random number from a xoroshiro generator and fires if it is below the lane's probability q; three more random bits give the slot. In a bright shot $q = p + \Delta p\, d$ with d starting at 1 at the TRIG and decreasing as $d \leftarrow d - d\,2^{-k}$ per word (the decay, $\tau \approx 2^k \times 8$ ns); the state is drawn at the TRIG from the table's 17-bit probability. Pairs copy events through a delay line of words; the split mode sends each event to one lane of the pair by another random draw.

22.10 Records

Records are assembled in queues of whole records (`ct_shotrec` for SHOT, SWEEP, SWAP and RUN, `ct_rates` for RATE), merged (`ct_merge`) and written by the ring writer (`rp_ring_writer`, shared with timetagger) in AXI3 bursts over the processor's HP0 port into a ring in DDR memory. The service's ring worker publishes the write pointer, keeps a heartbeat (the writer stops if the processor stops reading) and runs the ring pump that streams the ring to the clients.

22.11 Resources and verification

The design fills most of the Zynq 7010: 11 335 LUTs (64 %), 3 849 slices (87.5 %), 23 block RAMs and 40 DSP48 slices, with +0.235 ns of timing slack at 125 MHz. Fourteen counters were moved from logic into DSP48 blocks to make room, and the cluster logic was retimed (the decisions on 16-bit counts, the cluster's "time left" flag precomputed per slot). Every function was written first in the bit-exact Python model, checked there against simple event-level definitions, then in Verilog, checked against the model clock by clock in simulation, then built and tested on the board (section 16).

Part III
Counting in quantum labs

23 NV centres

The nitrogen-vacancy (NV) centre in diamond is a spin $S = 1$ in its negatively charged ground state, with a zero-field splitting of 2.87 GHz between $m_s = 0$ and $m_s = \pm 1$, optically initialized and read out at room temperature: under green light (532 nm) it fluoresces in the red (637–800 nm), about 30 % less brightly from $m_s = \pm 1$ during the first few hundred nanoseconds, because those states decay partly through a dark intersystem crossing that also pumps them into $m_s = 0$ [18, 32]. A single NV centre behind a good objective gives 50–300 kcount/s on a SPAD module. Counting its fluorescence is its readout: the pulse sequence (laser, microwave) comes from a pulse generator, counter gates the photons, sums them per point, and decides when a decision is needed. Hopper et al. [20] review the readout, Barry et al. [21] the sensitivity of NV magnetometers.

23.1 The setup

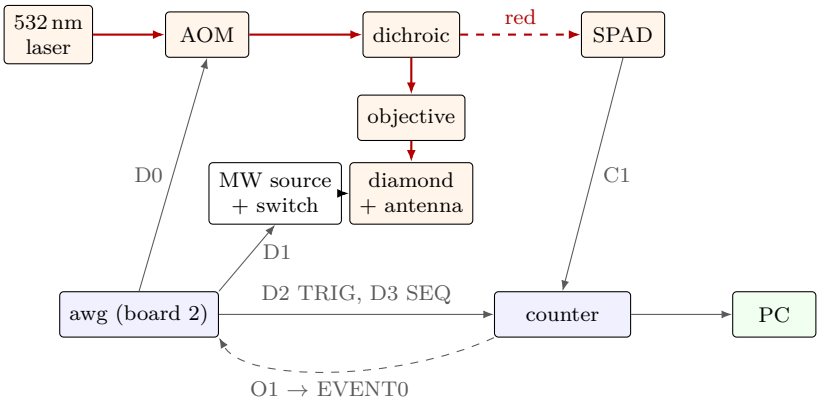


Figure 23: A pulsed NV experiment: awg on a second board sequences the laser (through an acousto-optic modulator, AOM), the microwave switch and counter's TRIG and SEQ; counter counts the SPAD's clicks in its gates and, for preselection, sends its decision back to awg.

Figure 23 shows the usual setup. A pulse generator switches the laser through an ***acousto-optic modulator*** (AOM, a crystal driven by radio frequency that deflects the beam into the experiment while the drive is on: rise times of tens of nanoseconds) and the microwave through a fast switch, and marks each readout with TRIG and each sweep with SEQ. counter's `nv-readout` preset expects exactly this:

```
dev.load_preset("nv-readout")    # TRIG on DIO0_N at each readout, SEQ on DIO3_N, 100 points per
    ↪ sweep
dev.apply({"channels": {"1": {"dead_time": 25e-9}},         # the SPAD module's dead time or more
           "shots": {"points": 51}})
```

The AOM's delay (hundreds of nanoseconds from its TTL to the light) and the fluorescence path's delay put the photons later than the TRIG: find them with the readout trace (section 23.3) and set G1's delay to the start of the light.

23.2 Photoluminescence, saturation and images

Before any pulse sequence: the count rate. The `rates` preset puts C1's rate on OUT1 (1 V per Mcount/s): the fastest feedback while focusing the objective onto an NV centre. A ***saturation curve***, the rate against the laser power P, follows $R = R_\infty P/(P + P_{\text{sat}}) + bP$ (the NV's saturation and a linear background); measure each point with `dev.rates(1.0)` and correct the dead time first (equation (1)), which already matters at 1 Mcount/s.

A ***confocal image*** is the count rate at each position of the focus; single NV centres were first found this way [33]. With a scanner that gives a pulse per pixel and one per line, counter takes one shot per pixel: the pixel clock on TRIG, G1 as wide as the pixel's dwell, SHOT records, and the line start on SEQ so that each line is a sweep of per-shot points. Each SHOT record then names its line (the sweep) and pixel (the point), and the PC assembles the image, of any size (figure 24):

```
dev.apply({"channels": {"1": {"enable": True}},
           "shots": {"trigger": "pin", "points": 64, "point_mode": "per-shot", "records": True},
           "gates": {"1": {"mode": "timed", "channels": [1], "delay": 0.0, "width": 4.9e-3}}})
# on the PC: image[r["sweep"] % 64, r["point"]] += r["counts"][0] for every SHOT record r
```

The per-point sums meanwhile give the average line, a useful live view of a line scan.

23.3 The readout window

The spin information is in the first few hundred nanoseconds of the readout pulse: $m_s = \pm 1$ fluoresces less until the laser has pumped it into $m_s = 0$ (about 250 ns). The `nv-trace` preset measures the

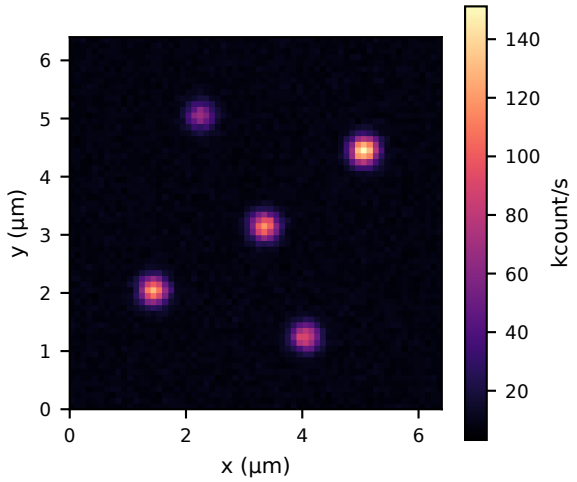


Figure 24: Computed. A confocal image from SHOT records: 4096 pixels of 5 ms, one shot per pixel (the pixel clock on TRIG, the line start on SEQ), five NV centres on a background of 8 kcount/s, with their Poisson noise.

readout's time trace with the gate-delay scan (section 8.4): with the microwave off (all $m_s = 0$) and with a π pulse (all $m_s = \pm 1$) in alternate sweeps, the difference of the two traces is the spin signal. Its window follows from equation (14): a longer window collects more photons but averages in the repolarized, spin-independent tail. Figure 25 computes it for a typical NV centre (200 kcount/s, a contrast of 30 % at the start decaying with 250 ns, 5 kcount/s of background): the best window is 313 ns, with 0.064 photons per shot and a contrast of 0.17 over the window; one shot then gives the population to ±33, and $1.1e+07$ shots give it to 1 %. The reference window G2 sits at the end of the readout pulse (2–3 µs in `nv-readout`), where the spin is repolarized.

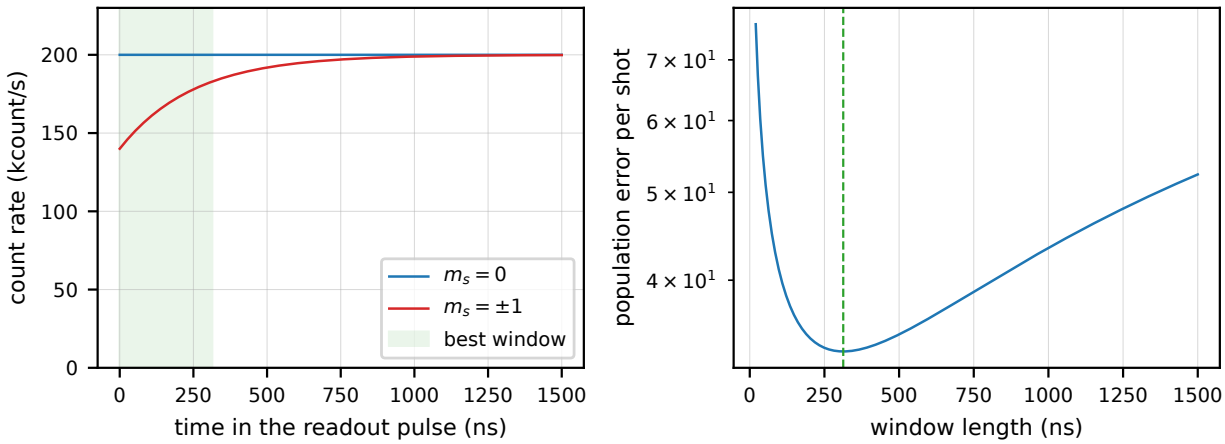


Figure 25: Computed. Left: the count rate of $m_s = 0$ and $m_s = \pm 1$ during a readout pulse (a simple model: a 30 % dip recovering with 250 ns). Right: the population's uncertainty from one shot against the readout window's length, with 5 kcount/s of background; the best window is shaded.

23.4 Rabi oscillations

A Rabi measurement steps the microwave pulse's length τ, one value per shot. With awg on a second board (its digital outputs: D0 laser, D1 microwave, D2 TRIG, D3 SEQ), the whole sweep is one pattern, played in a loop:

```
from awg import AwgClient
awg = AwgClient("192.168.0.104")
LASER, MW, TRIG, SEQ = 0x01, 0x02, 0x04, 0x08
taus = [20e-9 * k for k in range(25)]
words = []
def hold(word, seconds):
    words.extend([word] * max(1, round(seconds / 8e-9)))
for k, tau in enumerate(taus):
    if tau:
        hold(MW, tau)                                  # the microwave pulse
    hold(0, 100e-9)
    hold(LASER | TRIG | (SEQ if k == 0 else 0), 16e-9)   # readout: TRIG at the laser's start
    hold(LASER, 3e-6 - 16e-9)                          # 3 us: readout, then repolarization
    hold(0, 1e-6)
awg.save_pattern("rabi", words)
awg.set_digital(run="continuous", program="rabi", enable=0x0F,
                programs={"rabi": {"mode": "memory", "pattern": "rabi", "rate": 125e6}})
dev.load_preset("nv-readout")
dev.apply({"shots": {"points": len(taus)}})
dev.run(); time.sleep(60); dev.stop()
r = dev.results(now=True)
c = analysis.contrast(r["n1"], r["n2"])
fit = analysis.fit_rabi(taus, c["ratio"], sigma=c["error"])
print(fit["pi_time"])
```

Each sweep's 25 TRIGs are checked against the 25 points (section 7.4); a lost TRIG flags the sweep instead of shifting every later point. Figure 17 is the same measurement rehearsed with the test emitter. To cancel slow drifts, the sequence can play each τ twice, the second time with an extra π pulse before the readout: points $2k$ and $2k+1$ then hold the two, and their difference is free of the laser's drift.

23.5 Pulsed ODMR, Ramsey, echoes and relaxation

Pulsed ODMR a π pulse at a stepped microwave frequency: the synthesizer's list mode advances on a trigger from the sequence (one per point), counter's points follow the same sequence. The dip's centre measures a magnetic field to a fraction of its width (figure 26). Low-power pulsed ODMR avoids the power broadening of continuous-wave ODMR [34].

Ramsey $\pi/2$ – free evolution τ – $\pi/2$, τ stepped: the fringes decay with T_2^* (`analysis.fit_ramsey`).

Hahn echo and dynamical decoupling $\pi/2 - \tau - \pi - \tau - \pi/2$: T_2; with a final $3\pi/2$ instead of $\pi/2$ in alternate points, the difference of the two cancels laser-power drifts.

T_1 initialize, wait τ in the dark, read out: an exponential (`fit_exponential`).

For each, the sequence changes; counter's configuration stays `nv-readout` with the number of points.

23.6 Charge-state preselection

An NV centre spends part of its time in the neutral charge state NV^0, which has no usable spin and fluoresces at shorter wavelengths. Weak orange light (about 594 nm) excites NV^- but hardly ionizes it, so a few milliseconds of it, through a long-pass filter, gives many counts for NV^- and almost none for NV^0: a single-shot readout of the charge state [35]. A sequence that repeats "green pulse (randomizes the charge), orange readout" until the readout says NV^-, then runs the experiment, removes NV^0's contribution and raises the contrast. counter's `preselection` preset decides on the board: D1 = G1 $\geq$ threshold, as a pulse on O1 into awg's event input; awg's sequence waits in its readout step and jumps to the experiment on the event (section 24.6 has the sequence). The loop from the gate's end to awg's next pulse took 356–468 ns (measured), negligible against the milliseconds of the charge readout.

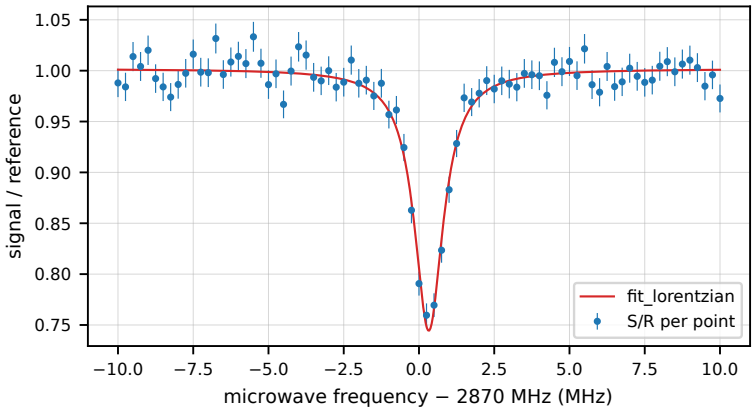


Figure 26: Computed. Pulsed ODMR: 81 frequencies, 200000 shots each, the signal window over the reference window with Poisson errors (`analysis.contrast`), and `fit_lorentzian`: the centre to ±20.5 kHz, the width 1.16 MHz.

23.7 How fast

A typical pulsed NV shot (polarization, microwave, readout) lasts 3–5 µs: 200 000–300 000 shots per second, each summed on the board. With $\sigma_P \approx 33$ per shot (figure 25), each point of a 25-point sweep gets about 10 000 shots per second and reaches 1 % in about 20 minutes. The sums do not need the PC, so a measurement can run overnight and be read in the morning.

24 Trapped ions and neutral atoms

A trapped ion or a single atom in an optical tweezer is read out by ***state-dependent fluorescence***: a laser drives a cycling transition from one qubit state (bright), while the other state is far off resonance (dark). With a few per cent of the light collected, a bright ion gives tens of counts in a few hundred microseconds, a dark one only the background: one shot tells the state [15, 36]. counter counts these photons, decides bright or dark on the board, records every shot for the histograms, and watches the fluorescence for the loss of the ion.

24.1 The setup

The experiment's sequencer (ARTIQ, a PulseBlaster, awg) switches the detection laser on and marks the detection's start with a TRIG; the photomultiplier or SPAD goes to C1 (a PMT through a discriminator, or through a preamplifier to IN1 as C9). The `ion-detection` preset counts C1 for 500 µs after each TRIG in G1, writes a SHOT record per shot and holds D1 = (G1 ≥ 3) on O1 until the next shot, a level the sequencer can read at any time afterwards:

```
dev.load_preset("ion-detection")
dev.apply({"channels": {"1": {"dead_time": 50e-9}},
           "gates": {"1": {"delay": 2e-6, "width": 300e-6}}})   # after the AOM's delay, 300 us
```

24.2 State detection: threshold and fidelity

Prepare the ion alternately bright and dark (or in a superposition), take SHOT records, and fit (section 20):

```
from counter import RecordStream
words = []
with RecordStream("192.168.0.102") as stream:
    for frame in stream.frames(timeout=2.0):
        words += analysis.parse_frame(frame)[1].tolist()
        if len(words) > 5 * 20000:
            break
h = analysis.count_histogram(model.decode(words), gate=1)
mix = analysis.two_poisson(h)
th = analysis.threshold(mix["dark"], mix["bright"])
dev.apply({"decisions": {"1": {"low_a": th["threshold"]}}})
print(th["threshold"], th["fidelity"], th["dark_error"], th["bright_error"])
```

The detection time is a trade-off (section 20.3): longer separates the two distributions, but off-resonant scattering pumps the bright state dark and the dark state bright during the detection. Figure 27 computes the error for 40 kcount/s of detected bright fluorescence and 0.5 kcount/s of background: without pumping it falls without limit; with pumping in both directions at a time constant of 20 ms the best time is 260 µs (error 0.75 %), at 5 ms it is 180 µs (2.24 %). Measure it rather than trust a model: G2–G4 as later windows of the same detection give the histograms of longer detection times from the same shots.

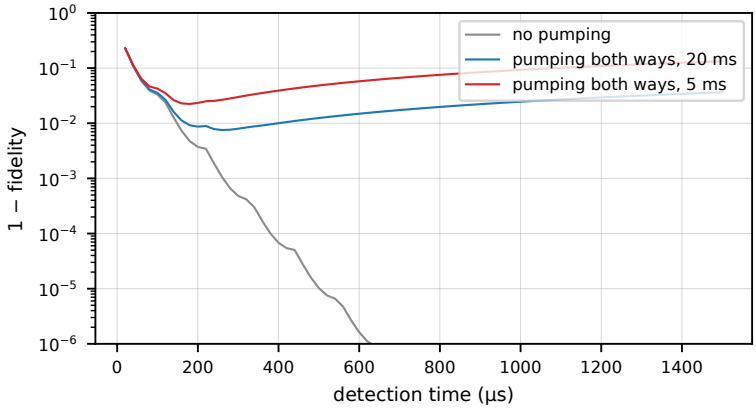


Figure 27: Computed. The error at the best threshold, $1 - F$, against the detection time, for a bright rate of 40 kcount/s and a background of 0.5 kcount/s, without pumping and with pumping between the states at 20 ms and 5 ms.

With the threshold set, D1 per shot is the measurement, and the per-point sums count the bright shots: a Rabi flop, a Ramsey fringe, a spectroscopy scan as the probability of bright per point, d_1/shots, corrected for the readout errors (section 20). For N shots per point (a sequencer that repeats each setting), use `per-next` points with a NEXT after each setting's last shot, or per-shot points with the sequence playing the settings in turn.

```
dev.apply({"shots": {"points": 41, "point_mode": "per-next"}})   # NEXT after each setting's shots
r = dev.results(now=True)
p = [d / s if s else 0.0 for d, s in zip(r["d1"], r["shots"])]   # P(bright) per point
```

24.3 Quantum jumps and the loss of an ion

Driven continuously, an ion with a long-lived shelving level blinks: its fluorescence switches off when it jumps into the metastable state and back on when it returns, the ***quantum jumps*** [37, 38]. RATE records every millisecond record the telegraph signal (figure 28); the histogram of the dark periods' lengths gives the metastable level's lifetime. The same records, with the rate alarm on the cooling fluorescence, catch the loss of the ion (section 6.4, figure 9): an output in `alarm` mode can stop the sequence or start the reloading.

```
dev.apply({"rates": {"enable": True, "interval": 1e-3},
           "alarm": {"enable": True, "channels": [1], "window": 10e-3, "below": 2e3},
           "outputs": {"3": {"mode": "alarm"}}})
```

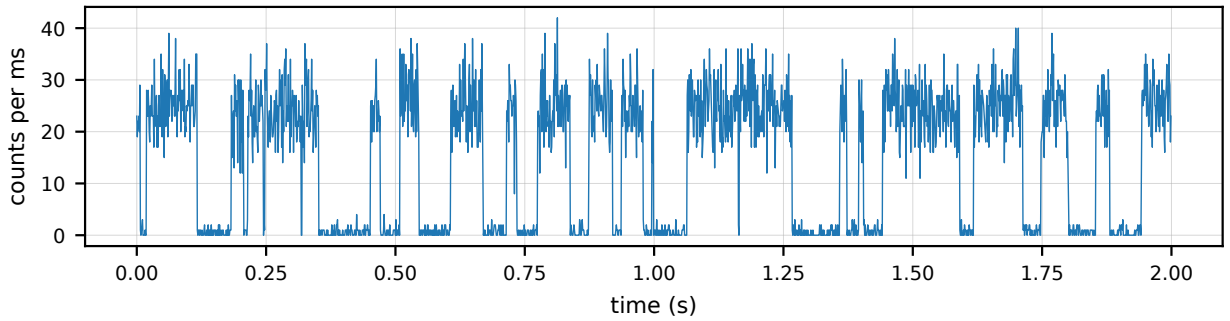


Figure 28: Computed. Quantum jumps in RATE records every 1 ms: an ion fluorescing at 25 kcount/s that jumps into a dark metastable state and back.

24.4 Micromotion: the photon–RF correlation

An ion displaced from the null of its Paul trap's radio-frequency (RF) field oscillates at the RF frequency (***excess micromotion***), which Doppler-modulates its fluorescence at that frequency. The ***photon–RF correlation*** measures it [39]: the fluorescence against the phase of the RF. counter does it with the gate-delay scan: a TRIG from the RF (through a comparator, divided to below 15 MHz, or with TRIG's hold-off longer than a few periods), G1 a few nanoseconds wide stepped by 1 ns over one RF period. The modulation's amplitude, minimized by the electrodes' compensation voltages, measures the micromotion (figure 29).

```
dev.apply({"channels": {"1": {"enable": True}},
           "controls": {"trig": {"holdoff": 300e-9}},                # every 8th RF period at 24
    ↪ MHz
           "shots": {"trigger": "pin", "points": 42, "point_mode": "per-shot"},
           "gates": {"1": {"mode": "timed", "channels": [1], "delay": 0.0, "width": 2e-9, "step": 1e
    ↪ -9}}})
```

With per-shot points and no SEQ the point simply cycles; every point gets the same number of shots, and their phases are spread evenly over the period.

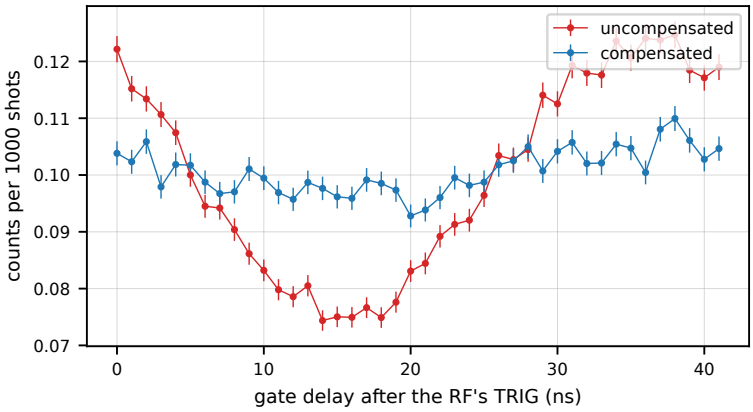


Figure 29: Computed. The photon–RF correlation for a 24 MHz trap: the fluorescence in a 2 ns window against its delay after the RF's trigger, for an uncompensated and a compensated ion.

24.5 Neutral atoms in tweezers

A tightly focused tweezer loads at most one atom (***collisional blockade***), with a probability of about one half [40]. A short fluorescence detection on an SPCM tells whether the atom is there: the same threshold and fidelity as for an ion, with the complication that the detection light can heat the atom out of the trap; Fuhrmanek et al. [41] show how to detect without loss. counter's uses:

- **loading**: D1 = an atom is present, as a pulse into the sequencer, which starts the experiment only then (preselection, below), or triggers a rearrangement of a small array [42];
- **release and recapture**: switch the tweezer off for a time t (the point), back on, and detect: the recapture probability against t (d_1/shots per point) gives the atom's temperature;
- **the magneto-optical trap's fluorescence**: RATE records follow its loading curve; the alarm notices a lost trap.

24.6 Preselection with awg on a second board

The decision pulse closes a loop: awg's sequence waits in a detection step and jumps to the experiment when counter's O1 pulses its event input (DIO4_P). The loop was measured on the board: 468 ns from the end of counter's gate to awg's output, 356 ns with awg's digital outputs at their shortest delay; counter's part is 105.0 ns (figure 12). The sequence (`RECIPES.md` has it with comments): one step starts the detection pattern (the laser and TRIG) and waits for the event; without an event it repeats the detection.

```
from awg import AwgClient
awg = AwgClient("192.168.0.104")
LASER, MW, TRIG = 0x01, 0x02, 0x04
awg.save_segment("z64", [0.0] * 64)                                  # 512 ns of silence
awg.save_pattern("detect", [LASER | TRIG] * 2 + [LASER] * 12498) # 100 us of light, TRIG first
awg.save_pattern("go", [MW] * 25)                                    # the experiment's first pulse
awg.save_sequence("presel", [
    {"segment": "z64", "repeat": 200, "digital": "detect", "on_event": "now", "event_target": 1, "
    ↪ next": 0},
    {"segment": "z64", "repeat": 4, "digital": "go", "next": 0}])
prog = lambda p: {"mode": "memory", "pattern": p, "rate": 125e6, "repeats": 1, "end": "idle"}
awg.apply({"arb": {"sequence": "presel", "rate": 125e6, "run": "continuous",
                   "jump": {"source": "event0", "edge": "rising"}},
           "outputs": {"1": {"arb": {"on": True, "amplitude": 0.0}}},
           "digital": {"run": "sequencer", "enable": 0x07, "delay": -112e-9,
                       "programs": {"detect": prog("detect"), "go": prog("go")}}})
dev.load_preset("preselection")      # D1 = G1 >= 2 as a 200 ns pulse on O1
dev.run()
```

awg takes an event only after its waiting step has started, so the waiting step must begin before counter's gate ends (here the detection step is itself the waiting step). awg jumps on a rising edge: use D1's `pulse-true` mode, so that every success gives a new edge; the `level` mode suits sequencers that read a level when they need it.

25 Photons: sources, correlations and entanglement

Counting coincidences between detectors is the basic measurement of quantum optics: it shows photon pairs, single photons and entanglement. It is also the core of the classic teaching experiments [26, 30]. counter counts them on the board, among four detectors per unit, with the accidentals counted next to them. This section goes through the experiments with the presets and analysis functions that serve them; section 21 has the theory.

25.1 Detectors and their delays

SPAD modules on C1–C4 (terminated, section 3.3), each channel's dead time set to the module's. Detectors and cables differ in delay by nanoseconds: find the relative delay of each pair with unit 2's delay scan on correlated light (a pair source, or one source's light split onto both detectors through a pulsed laser), and set the channels' delays so that the peak sits at zero (figure 15 shows such a scan with the test emitter's pairs, peaked at 16 ns).

```
dev.apply({"channels": {str(n): {"enable": True, "dead_time": 30e-9} for n in (1, 2)},
           "coincidences": {"2": {"enable": True, "channels": [1, 2], "window": 8e-9, "delayed":
    ↪ 2}}})
scan = []
for d in range(0, 40):
    dev.apply({"coincidences": {"2": {"delay": d * 1e-9}}})
    a = dev.coincidences()["2"]["counts"][3]; time.sleep(0.5)
    scan.append(dev.coincidences()["2"]["counts"][3] - a)
```

To scan the other sign, delay channel 1 instead (`"delayed": 1`).

25.2 A photon-pair source

Spontaneous parametric down-conversion (SPDC) in a nonlinear crystal splits a pump photon into two, emitted together [43]. With the two photons on C1 and C2:

- the **singles** R_1, R_2 and the **coincidences** N_{12} (unit 1);
- the **accidentals** (unit 2, C2 delayed by a microsecond) and the ***coincidence-to-accidental ratio*** CAR. The true pairs grow with the pump power, the accidentals with its square: the CAR falls as the power rises (figure 30: 874 at 50 000 pairs per second with 25 % heralding efficiency in each arm and an 8 ns window, 272 at 200 000);
- the **heralding efficiencies** $\eta_1 = N_{12}/R_2$ and $\eta_2 = N_{12}/R_1$ (after subtracting the accidentals): each detector's efficiency including the optics, measured without a calibrated source [27].

```
dev.load_preset("hbt")       # unit 1: C1, C2 within 8 ns; unit 2: C2 1 us later
a = dev.coincidences(); r = dev.rates(10.0); b = dev.coincidences()
n12 = b["1"]["inclusive"]["C1+C2"] - a["1"]["inclusive"]["C1+C2"]
acc = b["2"]["inclusive"]["C1+C2"] - a["2"]["inclusive"]["C1+C2"]
eta2 = (n12 - acc) / (r["channels"]["1"] * r["interval_s"])     # C2's heralding efficiency
```

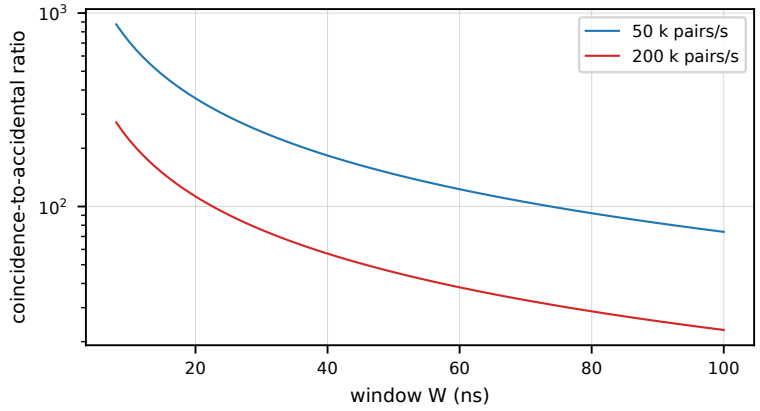


Figure 30: Computed. The coincidence-to-accidental ratio of a pair source (25 % heralding efficiency per arm, 2 kcount/s of dark counts) against the window, at two pair rates.

25.3 Antibunching of a single emitter

Behind a 50:50 beam splitter, a single emitter (an NV centre, a molecule, a quantum dot) never gives two photons at once: $g^{(2)}(0) < 1$, below 0.5 for a single emitter [23, 44]. The `hbt` preset counts C1+C2 coincidences in unit 1 and their accidentals in unit 2; `analysis.g2_zero` divides them. counter's window averages $g^{(2)}(\tau)$ over $\pm W$ (equation (18)): an NV centre (12 ns lifetime) shows 0.27 with $W = 8$ ns, more with background (figure 31); a quantum dot's sub-nanosecond lifetime is invisible in an 8 ns window under continuous excitation. With pulsed excitation the photons come in pulses: unit 1 counts the coincidences within a pulse and unit 2, delayed by one laser period, those of consecutive pulses; their ratio is $g^{(2)}(0)$ for any lifetime, provided the period is longer than the window's span ($2W + 1$ ns, 17 ns at least: lasers of up to about 50 MHz, or a pulse picker). The shape of $g^{(2)}(\tau)$ at sub-nanosecond delays needs a time tagger.

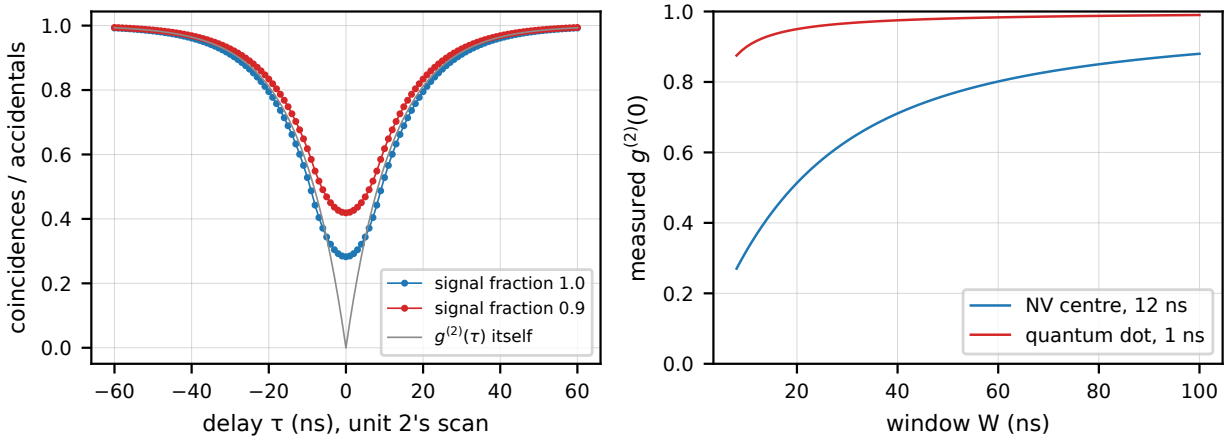


Figure 31: Computed. Left: the coincidences of an NV centre's two detectors over the accidentals, against unit 2's delay ($W = 8$ ns), without and with 10 % background light; grey: $g^{(2)}(\tau)$ itself. Right: the measured $g^{(2)}(0)$ against the window for lifetimes of 12 ns and 1 ns.

25.4 Heralded single photons: Grangier's experiment

With the herald on C1 and the two outputs of a beam splitter on C2 and C3, one unit counts all the patterns at once (the `heralded-g2` preset); $\alpha = N_1 N_{123}/(N_{12} N_{13})$ is the heralded $g^{(2)}(0)$, 0 for an ideal single photon and 1 for classical light [25, 26]:

```
dev.load_preset("heralded-g2")
a = dev.coincidences(); time.sleep(300); b = dev.coincidences()
inc = {k: b["1"]["inclusive"][k] - a["1"]["inclusive"][k] for k in b["1"]["inclusive"]}
print(analysis.heralded_alpha(inc["C1"], inc["C1+C2"], inc["C1+C3"], inc["C1+C2+C3"]))
```

The three-fold coincidences are rare: an hour of counting at typical teaching-lab rates gives α to a few hundredths. Unit 2 with C2 delayed counts the accidental part of the two-fold coincidences.

25.5 Hong–Ou–Mandel interference

Two indistinguishable photons meeting on a 50:50 beam splitter leave together: the coincidences between its outputs vanish when the two paths are equal [45]. The measurement is coincidences against the position of a translation stage in one path. counter's part is to count C1+C2 per position: the PC moves the stage and reads `coincidences()` before and after each dwell, or, for unattended scans, RATE records with the units' patterns (exact intervals) are split by the times of the stage's moves. The dip's width is the photons' coherence time (picoseconds: a stage scan, not an electronic delay); its depth, the ***visibility***, measures their indistinguishability (figure 32).

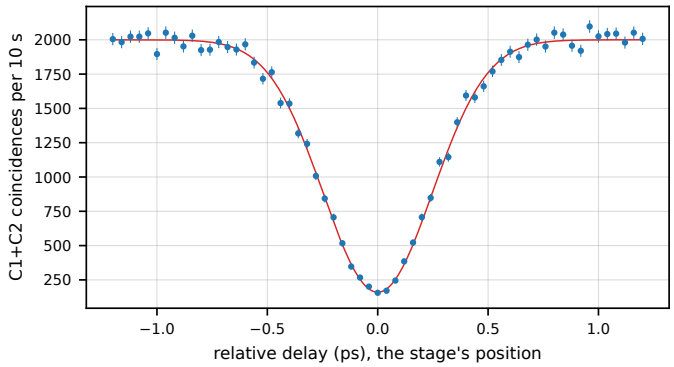


Figure 32: Computed. A Hong–Ou–Mandel dip: C1+C2 coincidences per 10 s against the stage's delay, visibility 0.92.

25.6 Polarization entanglement and Bell's inequality

A source of polarization-entangled pairs [43], Alice's polarizing beam splitter on C1 (+) and C2 (−), Bob's on C3 (+) and C4 (−): unit 1 with all four channels counts N_{++} (C1+C3), N_{--} (C2+C4), N_{+-} (C1+C4) and N_{-+} (C2+C3) at once (the `bell` preset). Turning Bob's analyser draws the correlation fringes (figure 33); four settings give CHSH's S [28, 29]:

```
dev.load_preset("bell")
E = []
for setting in ("a b", "a b'", "a' b", "a' b'"):
    input("set the analysers to %s, then press Enter" % setting)
    x = dev.coincidences(); time.sleep(60); y = dev.coincidences()
    d = {k: y["1"]["inclusive"][k] - x["1"]["inclusive"][k] for k in ("C1+C3", "C2+C4", "C1+C4", "C2+
    ↪ C3")}
    E.append(analysis.correlation(d["C1+C3"], d["C2+C4"], d["C1+C4"], d["C2+C3"]))
print(analysis.chsh(*E))       # {"S", "error", "violation_sigma"}
```

With a visibility of 0.94 and 4×10^4 coincidences per setting per minute, the computed example gives $S = 2.656 \pm 0.007$ (the ideal for this visibility is 2.659), a violation by 88 standard deviations. At high singles rates subtract unit 2's accidentals from each count before computing E.

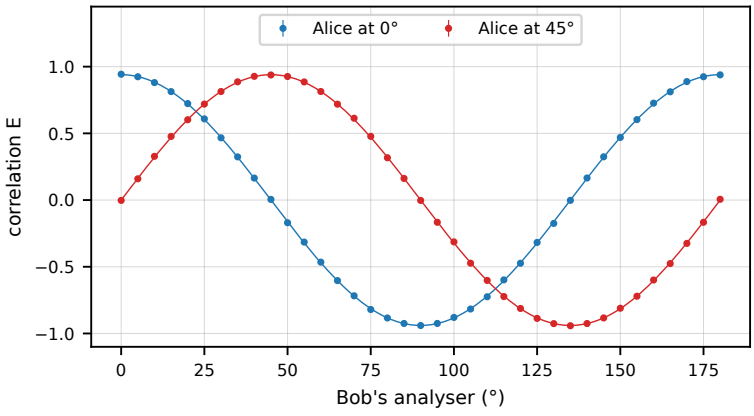


Figure 33: Computed. Correlation fringes of a polarization-entangled source of visibility 0.94: E against Bob's analyser for Alice at 0° and 45°, one minute per point.

25.7 Pulsed sources and gated detection

A pulsed laser's sync output (through a level translator to 3.3 V, and divided to below about 15 MHz) as TRIG lets the gates select photons by their time after the pulse: G1 after the laser pulse's

scattered light has passed, so that a quantum dot's or a molecule's emission is counted without the laser's reflection (***temporal filtering***); G2 at the pulse itself for the laser's own light. The gate-delay scan gives the emission's decay for lifetimes of a few nanoseconds upwards; shorter lifetimes need a time tagger [46]. Single-photon sources based on quantum dots [47, 48] are typically characterized this way: the brightness from gated counts, $g^{(2)}(0)$ from pulsed coincidences, the indistinguishability from Hong–Ou–Mandel.

25.8 In the teaching lab

The test emitter's pairs and its antibunched split source (section 13) let students learn the coincidence counter, the accidentals and the meaning of $g^{(2)}(0)$ before the optics work, and compare the same analysis on simulated and real photons. The rate monitor on OUT1, through a small loudspeaker amplifier, makes the alignment of a fibre coupler audible; the notebook's live coincidence table shows singles, coincidences and $g^{(2)}$ as the analysers turn.

A Configuration reference

Every setting, by section of the configuration dictionary (`dev.config()`, `dev.apply({...})`, TCP `CONFIG?`, `CONFIG {json}`). Times are in seconds, rates in events per second, voltages in volts at the connectors; values are rounded to the hardware's steps and the applied values returned. The defaults are the power-up state (`off`).

Key	Range (default)	Meaning
channels `"1"`–`"8"` (C1–C8) and `"9"`, `"10"` (C9, C10 on IN1, IN2)		
`enable`	bool (false)	counts its events
`edge`	rising, falling (rising)	C1–C8: the edges counted
`dead_time`	8 ns–65.535 µs (8 ns)	non-paralyzable, 1 ns steps (section 4.2)
`delay`	0–255 ns (0)	1 ns steps; less the analog lag while C9 or C10 is on
`polarity`	positive, negative (positive)	C9, C10: the pulses' sign
`threshold`	within the input's range, or null (null: never)	C9, C10: V at the connector
`hysteresis`	≥ 0 V (0: one code)	C9, C10: re-arms below $T - H$ ($T + H$ negative)
controls		
`trig`, `next`, `seq`: `edge`	rising, falling (rising)	TRIG (DIO0_N), NEXT (DIO2_N), SEQ (DIO3_N)
`holdoff`	8 ns–65.535 µs (24 ns)	edges within it after an accepted one are ignored
`gate`: `active`	high, low (high)	GATE (DIO1_N)
`min_width`	1–8 ns (1 ns)	shorter pulses and gaps are ignored
`delay`	0–255 ns (0)	
shots		
`trigger`	pin, timer, software (pin)	the source of TRIG
`timer_period`	64 ns–34 s (1 µs)	8 ns steps
`timer_seq`	bool (false)	a SEQ with every `points`-th timer TRIG
`points`	1–1024 (1)	
`point_mode`	per-shot, per-next, fixed (per-shot)	section 7.4
`fixed_point`	0–points−1 (0)	the point in fixed mode
`records`	bool (false)	SHOT records
`swap_at_sweep`	bool (false)	swap the per-point banks at each sweep's end
`start_at_seq`	bool (false)	a run starts at the next SEQ
`shot_limit`	0–$2^{32} - 1$ (0: none)	a run stops after so many shots

Key	Range (default)	Meaning
`duration`	0–about 2.4 h (0: none)	a run stops after this time (2.048 µs steps)
gates `"1"`–`"4"`		
`mode`	off, timed, gate (off)	gate: timed and where GATE is active
`channels`	list of 1–10 ([1])	counted, summed
`delay`, `width`	0–4.29 s (0)	1 ns steps, from TRIG
`step`	0–16.7 ms (0)	added to the delay per point
decisions `"1"`, `"2"`		
`gate_a`, `gate_b`	1–4 (1)	the gates tested
`low_a`, `low_b`	0–65535 (0)	
`high_a`, `high_b`	0–65534 or null (null: none)	on the 16-bit saturated counts
`logic`	a name or 0–15 (false)	D = table[$2b + a$] (section 9)
`output`	level, pulse-true, pulse-false, early (level)	the pin's behaviour
`pulse`	8 ns–2.048 µs (8 ns)	8 ns steps
outputs `"1"`–`"4"` (O1–O4, DIO4_N–DIO7_N)		
`mode`	off, low, high, pattern, emitter, alarm, gate, trig, d1, d2, echo, herald (off)	table 4
`pattern`	0–15 (0)	four 2 ns bits, bit 0 first
`lane`	1–4 (1)	emitter mode
`gate`	1–4 (1)	gate mode
`channel`	1–10 (1)	echo mode
`unit`, `herald`	1, 2 (1); a list of the unit's channels ([]: any cluster)	herald mode
coincidences `"1"`, `"2"`		
`enable`	bool (false)	
`channels`	up to 4 of 1–10, distinct ([])	the lanes L0–L3 in this order
`window`	8–1023 ns (8 ns)	1 ns steps
`gate`	bool (false)	only events inside G1
`delayed`, `delay`	unit 2: one of its channels or null; 0–16.383 µs	seen that much later
rates, **alarm**		
`rates.enable`, `rates.interval`	bool (false); 10 µs–10 s (0.1 s)	RATE records, 8 ns steps
`alarm.enable`, `alarm.channels`	bool (false); list ([1])	
`alarm.window`, `alarm.below`	1 µs–2 s (0.1 s); counts/s (1000)	on while the count is below `below` × `window`
monitors `"1"`, `"2"` (OUT1, OUT2)		
`mode`	off, rate, pulses (off)	section 6.3
`channels`	list ([1])	rate mode: summed
`window`	$2^k \times 8$ ns, $k = 4 \ldots 24$ (1.05 ms)	rate mode
`scale`, `offset`	V per count/s (1e-6); V (0)	
`lane`, `amplitude`	1–4 (1); V (0)	pulses mode
emitter		
`lanes.{1..4}.rate`	0–125 Mevents/s (0)	one trial per 8 ns word, steps of 7.45/s

Key	Range (default)	Meaning
`lanes.{1..4}.channel`	0–10 (0: none)	ORed into that channel
`lanes.{1..4}.bright`	events/s, signed (0)	added in a bright shot
`pairs.{1-2, 3-4}.mode`	independent, pairs, split (independent)	section 13
`pairs.*.probability`	0–1 (0)	
`pairs.*.delay`	0–248 ns in 8 ns steps (0)	pairs mode
`decay`	s (0: none)	the bright rate's decay, $\approx 2^k \times 8$ ns

The emitter's table of bright probabilities per point is not part of the configuration: load it with `dev.load_emitter_table([...])` (TCP `TABLE p0, p1, ...`); it is not kept across a load.

B TCP commands and Python methods

The service listens on TCP port 5025: one command per line, one reply line, `OK` followed by a value (JSON for structured ones) or `ERR` followed by a message. The Python client wraps each command in a session-guarded request; plain commands work from any language (section 15). `RECORDS STREAM` turns its connection into a binary stream; the ring pump serves the same on port 5026.

B.1 Common commands (every rpfpga design)

Command	Python	Reply, effect
`*IDN?`	`idn()`	`RedPitaya,rpfpga,counter,1.0,<build>`
`HELP`		the commands
`DESIGNS?`, `DESIGN?`	`designs()`, `design()`	installed designs, the active one
`LOAD [design]`	`load(design)`	switch designs or reprogram
`CONFIG?`	`config()`	the configuration (JSON)
`CONFIG <json>`	`apply(patch)`	patch the configuration; returns it as applied
`CONFIG REPLACE <json>`	`apply(c, replace=True)`	replace it (missing keys: defaults)
`DESCRIBE?`, `STATUS?`	`describe()`, `status()`	the settings in words; the status
`PRESETS?`, `PRESET <name>`	`presets()`, `load_preset(n)`	presets
`PRESET SAVE <name>`, `PRESET DELETE <name>`	`save_preset(n)`, `delete_preset(n)`	your own presets
`SAFE`	`safe()`	the safe state
`RANGE?`, `RANGE IN1 LV`	`ranges()`, `set_input_range("in1", "LV")`	the inputs' jumper settings (they must match the jumpers)
`CAL?`	`calibration()`	the board calibration

B.2 counter's commands

Command	Python	Reply, effect
`PINS?`, `PINS? CLEAR`	`pins(clear)`	for C1–C8, TRIG, GATE, NEXT, SEQ: `pin`, `level`, `seen`
`HARDWARE?`	`hardware()`	`abi`, `locked` (the samplers' PLL), `alarm`, the ring writer's state and counters, `records_dropped`

Command	Python	Reply, effect
`COUNTERS?`	`counters()`	`time_s`, per channel `total` (48 bits) and `rejected` (32), the emitter's events, the alarm
`RATES? [s]`	`rates(s)`	`interval_s`, counts/s of C1–C10 (0.001–10 s, default 0.1)
`WORKER?`	`worker()`	the service worker's statistics
`RUN [KEEP]`	`run(clear)`	start a run (the results cleared unless `KEEP`)
`STOP`	`stop()`	stop it (a shot in progress completes)
`TRIG`, `NEXT`, `SEQ`	`trigger()`, `next_point()`, `sequence()`	a software event at slot 0 of the current word
`CLEAR`	`clear()`	discard the per-point results
`RUN?`	`run_status()`	`state`, `point`, `shots`, `missed`, `sweeps`, `bad_sweeps`, `sweep`, `swaps`, `bank`, ...
`RESULTS? [NOW]`	`results(now)`	per point: `shots`, `n1`–`n4`, `d1`, `d2`, `n1_sq` (64 bits), `mean`; `NOW` swaps first
`TABLE p0, p1, ...`	`load_emitter_table(p)`	the emitter's bright probabilities per point
`DECISIONS?`	`decisions()`	per decision: `taken`, `true`, `last`
`LATENCY? [o c [jumper]]`	`measure_latency(o, c, j)`	with Oo wired to Cc: `latency_s` and the details
`COINC?`	`coincidences()`	per unit: `channels`, `window`, `counts` (16), `exclusive`, `inclusive`
`ANALOG? [n]`	`analog(n)`	IN1's and IN2's `mean`, `min`, `max`, `std` in volts
`RECORDS STREAM`	`RecordStream(host)`	the record stream (appendix C)

C Record formats

The stream's frames: a 24-byte header of little-endian 32-bit words (the magic `"CNR1"`, the frame's sequence number, the payload's bytes, the bytes lost before this frame, 0, 0), then the payload: records of 32-bit little-endian words. A record is a header word and the words that follow it; padding words `0xF8000000` may come between records. `model.decode(words)` reads them into dictionaries; `analysis.parse_frame(frame)` splits a frame.

Header bits	Meaning
31..27	type: 1 SHOT, 2 RATE, 3 SWEEP, 4 SWAP, 5 RUN (31 reserved: padding)
26..20	the number of words that follow
19..0	type-specific (below)

Table 14: The header word of every record.

RATE (type 2; 12 words, plus 15 for each coincidence unit that is on). Header [19:0]: the record's sequence number (mod 2^{20}, from 0 when `rates` is switched on; every firing counts, so a gap is a lost record). Words: E[31:0]; {[18:17] the units in the record, [16] the alarm, [15:0] E[47:32]}; the low 32 bits of the totals of C1 ... C10; then those of patterns 1 ... 15 of unit 1 and of unit 2 (each unit that is on). E is the record's time in words of 8 ns; channel k's value counts its events (after its delay) of the words before $E+k-1$, so that consecutive records differ by exactly one interval in every channel and pattern: `(b - a) mod 2**32`. A record in progress when `rates` is switched off completes.

SHOT (type 1; 4 words; with `shots.records`). Header [9:0] the point, [10] a TRIG was missed during the shot, [11] a count saturated (above 65535), [12] D1, [13] D2. Words: the TRIG's time in ns [31:0]; {the sweep's index [31:16], the time [47:32]}; {n_2, n_1}; {n_4, n_3} (16 bits each, saturated). D1 and D2 are the shot's decisions.

SWEEP (type 3; 4 words). Header [15:0] the closed sweep's index, [16] checked (per-shot points), [17] bad (its TRIGs differ from the points). Words: the SEQ's time in words [31:0], [47:32]; the sweep's TRIGs (with the missed); the missed.

SWAP (type 4; 3 words). Header [0] the bank frozen, [1] requested by software, [2] at a sweep's end, [3] automatic (a word could overflow), [4] the frozen bank had a saturated word. Words: the time in words [31:0], [47:32]; {the sweep [31:16], the point [9:0]}.

RUN (type 5; 4 words). Header [0] a stop (0: a start), [3:1] the stop's reason (1 software, 2 the shots' limit, 3 the duration), [19:4] the run's index. Words: the time in words [31:0], [47:32]; the run's shots; the missed (at a stop; 0 at a start).

Times in words of 8 ns wrap after $2^{48} \times 8$ ns, 26 days; the SHOT records' times in ns after 2^{48} ns, 78 hours. Records of different queues (shots' records, RATE records) can be out of time order by a few records; each carries its time.

D Registers

For programmers of the FPGA side and for reading the bit-exact model: the register map of the `ct_core` block, generated from `docs/REGISTERS.md`. Users never need it: the driver writes every register, checks it, and restores the configuration after a reload; a direct access to the FPGA's addresses while it is not configured hangs the board.

`ct_core` is a 128 KiB AXI4-Lite window at **0x43C00000** (32-bit words, byte offsets). Access it only through the driver (`rpfpga.Overlay`): a read or write while the FPGA is not configured hangs the board (`../../common/README.md`, "Safe loading"). The common sysid block is at 0x40000000 (design ID `CNTR`, version 1.0).

Layout version (ABI) **5**, counter 1.0 (as 0.4): the counting of the plan's step 1 (0x000 .. 0x18C), the shots of step 2 (0x200 .. 0x2D4, the emitter's table at 0x4000, the frozen per-point bank at 0x10000), the decisions of step 3 (0x300 .. 0x33C, OUT_SEL's sources 5 and 6) and the analog channels and coincidences of step 4 (0x340 .. 0x4FC, OUT_SEL's source 7, MON_CFG's pulses).

Timing. A write whose address and data the bus presents in clock c takes effect in clock c + 2; a read presented in clock c returns the value of clock c + 1 (the bit-exact model, `control/counter/model.py`, and `tests/tb_front.sv` use exactly this). Clock c is the value of TIME (the core's `now`).

General

Offset	Name		Meaning
0x000	ABI	R	the register layout's version (5)
0x004	STATUS	R	[0] the 1 ns samplers' PLL locked; [3] the ring writer runs; [4] its watchdog tripped; [5] the rate alarm
0x008	LEVELS	R	each input's latest 1 ns sample: [7:0] DIO0..7_P (C1..C8), [11:8] DIO0..3_N (TRIG, GATE, NEXT, SEQ)
0x00C	SEEN	R / W1C	a rising edge since the last clear, same bits; writing 1 clears a bit
0x018	TIME_LO	R	clocks (8 ns) since the core's reset, bits 31..0 (reading it latches TIME_HI)
0x01C	TIME_HI	R	bits 63..32, latched by the last TIME_LO read

Channels (C1..C10; C9, C10 are the analog discriminators of step 4 and idle in 0.1)

Offset	Name		Meaning
0x024	CH_CTRL	RW	[9:0] channel enabled; [25:16] falling edges (the pin inverted)
0x080 + 4 i	CH_CFG i	RW	channel i + 1: [15:0] dead time − 8 (ns; the dead time 8 ... 65,535 ns), [23:16] delay (0 ... 255 ns)

Offset	Name		Meaning
0x0C0 + 8 i	TOTAL_LO i	R	lane i's 48-bit total, bits 31..0 (reading it latches its bits 47..32)
0x0C4 + 8 i	TOTAL_HI i	R	[15:0] bits 47..32, latched by the last TOTAL_LO read
0x110 + 4 i	REJECTED i	R	channel i + 1's edges not counted (within its dead time), 32 bits, wrapping

The totals count each lane's events after its delay and are never cleared (2^48 counts: 26 days at 125 Mcount/s); in clock c they hold the events of (delayed) pin words < c − 9.

RATE records and the alarm

Offset	Name		Meaning
0x028	RATE_PERIOD	RW	T − 1: a RATE record every T clocks (10 µs ... 10 s from the driver; values below 63 act as 63, so at most one record per 64 clocks; power-up 12,499,999: 0.1 s). The driver keeps it while records are off: written before RATE_CTRL, a short period would otherwise fire in the clocks between the two writes
0x02C	RATE_CTRL	RW	[0] on; off resets the sequence count (the first record T clocks after on)
0x030	ALARM_CTRL	RW	[9:0] the lanes counted; [16] on
0x034	ALARM_WINDOW	RW	W − 1: windows of W clocks
0x038	ALARM_LEVEL	RW	the alarm is on after a window with fewer counts than this
0x03C	ALARM_LAST	R	the last window's count (kept when the alarm is turned off)
0x060	REC_DROPS	R	RATE records not made for want of room in the record FIFO (the mailbox counts these and the shots' records' drops)
0x064	RATE_SEQ	R	the RATE firings since on (also those without a record while the writer is off)

The ring writer (owned by the service's worker, `rpfpga.ringworker`)

Offset	Name		Meaning
0x040	W_CTRL	RW	[0] the ring writer runs (a rising edge restarts it: written position 0, base and size taken)
0x044	W_BASE	RW	the ring's physical address (4 KiB mailbox, then the data area)
0x048	W_SIZE	RW	the ring's size in bytes
0x04C	W_RD	RW	the consumer's position (bytes, mod 2^32); every write is the watchdog's heartbeat
0x050	W_WD	RW	the watchdog in clocks (0: off)
0x054	W_WR	R	the written position (bytes, mod 2^32), after the bursts' responses
0x058	W_BURSTS	R	data bursts written
0x05C	W_STALLS	R	clocks with a full burst waiting for room

Records are made only while the writer runs. The mailbox (the ring's first 4 KiB) carries the magic 0x434E5452 ("CNTR"), its sequence, the written position, the core's time and REC_DROPS.

The test emitter

Offset	Name		Meaning
0x140 + 4 l	EM_P l	RW	lane l + 1: [23:0] p, the probability of an event per word × 2^24 (a rate of p / 2^24 × 125 MHz); in the modes pairs and split, lane 2's (4's) p is the copy probability or the fraction
0x150	EM_CFG	RW	[4l+3:4l] lane l + 1's channel (0 none, 1 … 10); [17:16] lanes 1, 2 and [19:18] lanes 3, 4: 0 independent, 1 pairs, 2 split
0x154	EM_DELAY	RW	in pairs mode: [4:0] lane 2's delay, [12:8] lane 4's (words of 8 ns, 0 … 31)
0x160 + 4 l	EM_COUNT l	R	lane l + 1's events (32 bits, wrapping)

Monitors (OUT1, OUT2) and outputs (O1..O4)

Offset	Name		Meaning
0x170 + 4 o	MON_CFG o	RW	OUT(o + 1): [9:0] the lanes counted; [20:16] k (windows of 2^k clocks, 4 … 24); [24] on; [25] pulses (instead of the rate); [27:26] the pulses' emitter lane − 1
0x178 + 4 o	MON_GAIN o	RW	the rate: [17:0] the gain g (signed); [28:24] the shift s (in steps of 4: bits 25:24 ignored). Pulses: [15:0] the amplitude (signed code)
0x180 + 4 o	MON_OFFSET o	RW	[15:0] the offset code (signed; read sign-extended)
0x188 + 4 o	MON_CODE o	R	the output's code (signed)
0x010	OUT_EN	RW	[3:0] O1..O4 (DIO4..7_N) driven; 0 = high impedance (power-up, SAFE)
0x014	OUT_PAT	RW	[4k+3:4k] output k + 1's bits of each sample period, bit 0 first, 2 ns each
0x020	OUT_SEL	RW	[8k+2:8k] output k + 1's source: 0 the pattern, 1 an emitter lane ([8k+5:8k+4]: the lane − 1; 4 ns pulses), 2 the rate alarm (high while on), 3 a gate's window ([8k+5:8k+4]: the gate − 1; 2 ns bits, a bit high when either of its two slots is in the window), 4 the accepted TRIGs (4 ns pulses), 5 a decision's pin ([8k+4]: 0 D1, 1 D2), 6 a channel's echo ([8k+7:8k+4]: the channel − 1, 0 … 9; its counted edges as 4 ns pulses), 7 a herald ([8k+7:8k+4]: the lanes that a cluster must hold; the unit in OUT_HERALD; a 4 ns pulse from slot 0 of the word after the cluster closed)

A monitor's code is `sat16(offset + ((last · g) » s))` four clocks after each window's end (`last`: the window's count, saturated to 2^24 − 1); with pulses it is `sat16(offset + amplitude)` in the clock after each event of the emitter lane, else the offset; while off it is the offset (the driver's calibrated 0 V by default).

Control inputs, shots and gates (plan step 2)

Offset	Name		Meaning
0x200	CTL_CFG	RW	[0] TRIG on falling edges, [1] GATE active low, [2] NEXT falling, [3] SEQ falling; [10:8] GATE's minimum width − 1 (ns); [23:16] GATE's delay (0 … 255 ns)
0x204	HOLD_TN	RW	[15:0] TRIG's hold-off − 8, [31:16] NEXT's (ns; power-up 16: 24 ns)
0x208	HOLD_S	RW	[15:0] SEQ's hold-off − 8 (ns; power-up 16)

Offset	Name		Meaning
0x20C	SHOT_CFG	RW	[1:0] TRIG: 0 the pin, 1 the timer, 2 software only; [3:2] the point: 0 per shot, 1 per NEXT, 2 fixed; [4] SHOT records; [5] a swap at each sweep's end; [6] runs start at the next SEQ; [7] the timer gives a SEQ with every P-th TRIG
0x210	POINTS	RW	[10:0] P (1 . . . 1024; power-up 1); [25:16] the fixed point
0x214	TIMER	RW	the timer's period − 1 (clocks; power-up 124: 1 µs)
0x218	RUN_CTRL	W	pulses: [0] start, [1] stop, [2] TRIG, [3] NEXT, [4] SEQ (software; slot 0), [5] swap, [6] release the frozen bank, [7] clear (between runs: zero both banks)
0x21C	SHOT_LIMIT	RW	a run stops after this many shots (0: none)
0x220	DUR_LIMIT	RW	a run stops after this many × 256 clocks (0: none)
0x224	RUN_STATUS	R	[1:0] 0 idle, 1 armed (waits for a SEQ), 2 running; [2] a shot busy; [3] the counting bank; [4] the frozen bank released; [5] zeroing a bank; [6] a word saturated; [25:16] the point register
0x228	SHOTS	R	accepted TRIGs of the run (32 bits, wrapping: $4.29{\cdot}10^9$ shots, 4.9 h at 244,000 shots/s; the per-point sums' shots are 64-bit)
0x22C	MISSED	R	TRIGs while a shot was busy (32 bits, wrapping)
0x230	SWEEPS	R	sweeps closed (32 bits, wrapping)
0x234	BAD_SWEEPS	R	sweeps closed with TRIGs ≠ P (per-shot points)
0x238	SWAPS	R	the banks' swaps
0x23C	SREC_DROPS	R	the shots' records not made (their slot full or no room), since the bitstream was loaded (a run does not clear it)
0x240 + 0x20 g	G_CFG g	RW	gate g + 1: [1:0] 0 off, 1 timed, 2 timed and the GATE level; [25:16] the channels counted
0x244 + 0x20 g	G_DELAY g	RW	the delay d (ns)
0x248 + 0x20 g	G_WIDTH g	RW	the width w (ns)
0x24C + 0x20 g	G_STEP g	RW	[23:0] the delay's step s per point (ns)
0x250 + 0x20 g	G_LAST g	R	the gate's count of the last shot (32 bits, saturating)
0x2C0 + 4 l	EM_DP l	RW	[24:0] lane l + 1's step in the bright state (signed, per word × 2^24)
0x2D0	EM_DECAY	RW	[4:0] k: the bright state's decay d −= d » k per word (0: none)
0x2D4	SWEEP_IDX	R	[15:0] the open sweep's index
0x4000 + 4 i	EM_TABLE i	W	[16:0] point i's bright probability × 65536 (65536: always)
0x10000 + 0x1000 c + 4 i	BANK c, i	R	the frozen bank: column c (0 n1, 1 n2, 2 n3, 3 n4, 4 shots, 5 D1 true, 6 D2 true, 7 $n1^2$ low, 8 $n1^2$ high) of point i

A gate of a shot with TRIG at t_T (in ns of the pin time) and point i covers [t_T + d + i·s, t_T + d + i·s + w); a shot is busy for its TRIG's word and the next max(7, the gates' last words) words, and its gates' results are final four clocks after its last word (docs/THEORY.md 2.5, 2.6). Writes of a gate's registers, of POINTS or SHOT_CFG make the gate compute d + p·s again (from 4 clocks after the last such write).

Decisions (plan step 3)

Offset	Name		Meaning
0x300 + 0x20 k	D_CFG k	RW	D(k + 1): [1:0] gate A − 1, [3:2] gate B − 1, [7:4] the table (D = bit 2b + a), [9:8] the pin: 0 level, 1 pulses on true, 2 pulses on false, 3 early; [23:16] the pulse's words − 1
0x304 + 0x20 k	D_LOW_A k	RW	[15:0]: a = LOW_A ≤ n_A ≤ HIGH_A (n_A saturated to 16 bits)

Offset	Name		Meaning
0x308 + 0x20 k	D_HIGH_A k	RW	[15:0] (power-up 0xFFFF: no limit)
0x30C + 0x20 k	D_LOW_B k	RW	[15:0]: b = LOW_B ≤ n_B ≤ HIGH_B
0x310 + 0x20 k	D_HIGH_B k	RW	[15:0] (power-up 0xFFFF)
0x314 + 0x20 k	D_COUNT k	R	decisions taken since loading (wrapping)
0x318 + 0x20 k	D_TRUE k	R	of them true
0x31C + 0x20 k	D_LAST k	R	[0] the last decision

A decision is taken in the clock after the later of its two gates' results (each gate's result is in at its end); D enters the per-point sums and the SHOT record one clock after the shot's done. The early pin is gate A's running count ≥ LOW_A while that gate counts and until the next shot's gates are loaded (docs/THEORY.md 2.7).

Analog channels and coincidences (plan step 4)

Offset	Name		Meaning
0x340 + 0x20 u	CO_CFG u	RW	unit u + 1: [0] on, [1] only inside G1's window; [25:16] W (ns, 8 … 1023; below 8 acts as 8; power-up 8; a new W applies to the clusters that open from the clock after its write takes effect)
0x344 + 0x20 u	CO_LANES u	RW	[4i+3:4i] lane i's channel − 1 (0 … 9; 10 … 15: none; power-up 0xFFFF)
0x368	CO_DELAY	RW	unit 2: [13:0] D (ns, 0 … 16,383), [17:16] the delayed lane
0x380	OUT_HERALD	RW	[k] output k + 1's herald: 0 unit 1, 1 unit 2
0x3A0 + 0x10 k	AN_CFG k	RW	C(9 + k) on IN(1 + k): [0] negative pulses (the samples inverted: y = ~x)
0x3A4 + 0x10 k	AN_THRESH k	RW	[16:0] T (signed, in y; fires at y ≥ T; power-up 0x08000: never)
0x3A8 + 0x10 k	AN_HYST k	RW	[15:0] H (armed by y ≤ T − H; 0 acts as 1; power-up 1)
0x3AC + 0x10 k	AN_SAMPLE k	R	[15:0] the latest sample x (16-bit stream code)
0x400 + 0x80 u + 8 p	PAT u, p	R	unit u + 1's total of the exclusive pattern p (1 … 15: the lanes' bits), bits 31..0 (reading it latches bits 47..32)
0x404 + 0x80 u + 8 p	PAT_HI u, p	R	bits 47..32, latched by the last PAT read

The analog channel's event word enters its count channel (CH_CFG 8, 9: the dead time and the delay) five clocks after its sample was on the converter's port; the lanes of a pulse at the connector come Δ later than the pins' (docs/THEORY.md 2.3): the driver adds Δ to the pin channels', the gates' and GATE's delays while an analog channel is enabled. The units take the gates' lanes one clock later (pin word c − 10); unit 2's lanes, G1's window and its delayed lane 4 clocks after that (its delayed lane D ns more: a slot shift, then a ring of 4096 words in a block RAM). A closed cluster's pattern is counted one clock after its close (distributed RAM); the RATE records read the totals one per clock after the channels'.

Configuration registers

Restored and checked by `rpfpga.Overlay` (the default configuration compiles to the power-up values with the nominal calibration: 0 but RATE_PERIOD, HOLD_TN, HOLD_S, POINTS, TIMER, the decisions' HIGH_A, HIGH_B, CO_CFG, CO_LANES, AN_THRESH and AN_HYST as above), written in this order: CH_CTRL, CH_CFG 0..9, EM_P 0..3, EM_DP 0..3, EM_DECAY, EM_DELAY, EM_CFG, RATE_PERIOD, RATE_CTRL, ALARM_WINDOW, ALARM_LEVEL, ALARM_CTRL, MON_GAIN, MON_OFFSET, MON_CFG (both monitors), CTL_CFG, HOLD_TN, HOLD_S, TIMER, SHOT_LIMIT, DUR_LIMIT, POINTS, each gate's DELAY, WIDTH, STEP, CFG, each decision's LOW_A, HIGH_A, LOW_B, HIGH_B, CFG, each

analog channel's THRESH, HYST, CFG, unit 1's LANES, CFG, unit 2's LANES, DELAY, CFG, OUT_HERALD, SHOT_CFG, OUT_PAT, OUT_SEL, OUT_EN. The ring registers belong to the service's worker; RUN_CTRL's pulses and the emitter's table are operations, not configuration.

E Glossary

Accidental coincidence Two independent events that happen to fall within the coincidence window: $R_1 R_2(2W + 1\,\text{ns})$ per second for counter's clusters (section 21.1).

Acousto-optic modulator (AOM) A crystal driven by radio frequency that diffracts a laser beam while the drive is on: the usual fast optical switch of pulse sequences (tens of ns).

Afterpulse A false click that follows a real one in the same detector, from trapped charge (SPAD) or ions (PMT), tens of ns to µs later (section 19.2).

Antibunching Fewer photon pairs at zero delay than for random light, $g^{(2)}(0) < 1$: the signature of a single emitter.

Armed A run waiting for its first SEQ; a discriminator ready to fire.

Bank One of the two copies of the per-point sums: one counts, the other is read by the service.

Binomial counts Counts from a fixed number of trials, each with a probability p: variance $Np(1-p)$, below Poisson's (the test emitter).

Board calibration The constants, shared by all rpfpga designs on a board, that convert the converters' codes to volts and give the delays of the analog and digital paths.

Bright, dark The two states of a fluorescence readout: many photons or few.

Busy A shot from its TRIG to its last gate's end (at least 64 ns); TRIGs meanwhile are missed.

CAR Coincidence-to-accidental ratio: true coincidences over accidentals in the same window.

Channel C1–C10: an input's events after edge detection, dead time and delay.

Channel set The channels a gate counts, summed.

CHSH parameter S from four correlations; above 2 it excludes local hidden variables, up to $2\sqrt{2}$ in quantum mechanics (section 21.4).

Click One detector pulse: one detected photon (or a dark count or an afterpulse).

Cluster A group of events within the window W after the first one; counted once by its pattern (section 11.1).

Coincidence Clicks of two or more detectors within the coincidence window.

Collisional blockade The light-assisted collisions that leave at most one atom in a small optical tweezer.

Confocal image A map of the fluorescence counted at each position of a focused spot.

Constant-fraction discriminator A discriminator that fires at a fixed fraction of each pulse's height, free of time walk.

Correction factor The ratio of true to counted rate of a detector with dead time, $1/(1 - m\tau)$.

Dark count A click without light: thermal, from radioactivity, cosmic rays.

Dead time The time after a counted event during which a channel or detector takes no other (section 4.2).

Decision D1, D2: a count window on each of two gates, any logic of the two, once per shot (section 9).

Discriminator A comparator that turns an analog pulse into a logic event at a threshold.

Double count Two counts from one pulse, from ringing or a reflection.

Early A decision's pin mode that signals "bright" at the threshold-th count, before the gate's end.

Echo An output that repeats a channel's counted events as 4 ns logic pulses.

Event A counted edge of a channel, with its time to the nanosecond.

Exclusive, inclusive patterns Clusters of exactly a set of channels; of at least that set.

Fano factor Variance over mean of counts: 1 for Poisson, below for sub-Poissonian, above for super-Poissonian counts.

Feed-forward Using a measurement's result to choose the following operations of the same experimental run.

Fidelity (readout) $1 - (\epsilon_D + \epsilon_B)/2$, the mean probability of calling a state right.

Frozen bank The bank of per-point sums that the service reads while the other counts.

$g^{(2)}(\tau)$ The normalized intensity correlation at delay τ (section 21.2).

Gate A time window of a shot in which counts are taken (section 8).

GATE The external gate-level input (DIO1_N).

Gated photon counter An instrument that counts detector pulses in time windows after a trigger.

Half-open interval $[a, b)$: includes its start, excludes its end; adjacent gates never share an event.

Herald A click that announces the presence of a partner photon; an output pulse per such cluster.

Heralding efficiency The probability that a heralded photon is detected (Klyshko efficiency).

High impedance An output that does not drive its pin, as if disconnected.

Hold-off TRIG's, NEXT's and SEQ's dead time against bounces and glitches.

Hysteresis The gap between a discriminator's firing and re-arming levels.

Interpolation Locating a threshold crossing between two 8 ns samples to the nanosecond.

Lane A stream of events of one input inside the FPGA, at most one per word.

Latency The time from a cause (a gate's end) to its effect at an output connector.

Leading-edge timing The time at which a pulse crosses a fixed threshold.

Logic family A convention of voltage levels for logic signals: LVCMOS, TTL, NIM, ECL.

LVCMOS33 3.3 V CMOS logic: high above 2.0 V, low below 0.8 V; the E1 pins.

Micromotion An ion's oscillation at its trap's radio frequency when displaced from the RF null.

Missed TRIG A TRIG that came while a shot was busy: counted, not started.

NIM Fast negative logic of nuclear electronics: -16 mA (-0.8 V into 50 Ω) for a 1.

Non-paralyzable, paralyzable A dead time that does not, or does, restart with every event inside it.

Oversampling Sampling a pin's level every nanosecond with the FPGA's input deserializers.

Pattern The set of channels in a cluster; also an output's fixed four 2 ns bits.

Photon–RF correlation Fluorescence against the phase of an ion trap's RF: micromotion.

Pile-up Two photons within one detector's dead time counted as one.

Plateau The range of thresholds over which a discriminator's count rate hardly changes.

Point The index of a scan's parameter value; the per-point sums keep points apart.

Poisson distribution The counts of independent events: variance equal to the mean.

Preselection Repeating a check until it succeeds before an experiment starts.

Preset A named configuration.

Pull-down A resistor that holds an unconnected input low.

Pulse-height distribution The histogram of a detector's pulse heights.

Quantum efficiency The probability that a photon frees an electron in a photocathode or detector.

Quantum jumps The sudden on/off switching of a driven ion's fluorescence when it is shelved in a metastable level.

Rate alarm An alarm on while the count of chosen channels in a window is below a level.

Rate monitor A voltage on OUT1/OUT2 proportional to a count rate.

RATE record The totals at the end of each interval, with exact intervals between records.

Readout contrast The relative difference of two states' fluorescence in the readout window.

Record A block of words in the stream: SHOT, RATE, SWEEP, SWAP, RUN.

Rejected edge An edge inside a channel's dead time, not counted.

Ring buffer, ring pump The records' buffer in the board's memory, and the service that streams it.

Run The time during which shots are taken.

Saturation curve A fluorescence rate against excitation power, flattening at high power.

Schmitt trigger A comparator with hysteresis.

Shot One repetition of an experiment, started by a TRIG.

Shot noise The $\sqrt{N}$ scatter of N Poisson counts.

Single-photon detector A detector that gives a pulse per detected photon: SPAD, PMT, SNSPD.

Slot A nanosecond within an 8 ns word.

Source termination A series resistor at a line's source that absorbs its reflections.

SPAD, SPCM Single-photon avalanche diode; a module with one, its quenching and a logic output.

SPDC Spontaneous parametric down-conversion: a pump photon split into a photon pair.

Sub-, super-Poissonian Counts with a Fano factor below or above 1.

Swap Exchanging the counting and the frozen bank of per-point sums, between two shots.

Sweep One pass through all points, from SEQ to SEQ.

Temporal filtering Selecting photons by their arrival time after a laser pulse.

Termination A resistor equal to the cable's impedance at its end: no reflection.

Test emitter The FPGA's source of detector-like pulses (section 13).

Thermometer mask All bits below a position set: selects the slots inside a window.

Threshold The level at which a discriminator fires; the count at which a decision says "bright".

Time trace Counts against the time after the trigger, from the gate-delay scan.

Time walk The dependence of a leading-edge time on the pulse's height.

Total A channel's or pattern's 48-bit count since loading, never cleared.

TRIG The input that starts a shot (DIO0_N); the internal timer and software can also trigger.

TTL Transistor-transistor logic: positive pulses of 2–5 V; on detectors often "into 50 Ω".

Velocity factor A cable's signal speed relative to light's (about 0.66 for coax).

Visibility $(N_{\max} - N_{\min})/(N_{\max} + N_{\min})$ of an interference or correlation fringe.

Window W: a coincidence unit's cluster length (8–1023 ns).

Word One 8 ns clock period of eight 1 ns slots.

References

[1] Red Pitaya. Red Pitaya documentation. Online documentation. URL https://redpitaya.readthedocs.io/. Undated; accessed 5 November 2026. The STEMlab 125-14 hardware: https://redpitaya.readthedocs.io/en/latest/developerGuide/hardware/ORIG_GEN/125-14/top.html.

[2] AMD. PYNQ: Python for adaptive computing. Project web site. URL https://www.pynq.io/. Undated; accessed 5 November 2026. Documentation: https://pynq.readthedocs.io/.

[3] U.S. NIM Committee. Standard NIM instrumentation system. Report DOE/ER-0457T, U.S. Department of Energy, Office of Energy Research, Washington, DC, May 1990. URL https://www.osti.gov/biblio/7120327.

[4] *Model SR400 Gated Photon Counter*. Stanford Research Systems, Sunnyvale, CA, November 2018. URL https://www.thinksrs.com/downloads/pdfs/manuals/SR400m.pdf. Manual, revision 2.7.

[5] 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.

[6] R. Loudon. *The Quantum Theory of Light*. Oxford University Press, Oxford, third edition, 2000. doi: 10.1093/oso/9780198501770.001.0001.

[7] L. Mandel and E. Wolf. *Optical Coherence and Quantum Optics*. Cambridge University Press, Cambridge, 1995. doi: 10.1017/CBO9781139644105.

[8] L. Mandel. Sub-Poissonian photon statistics in resonance fluorescence. *Opt. Lett.*, 4:205–207, 1979. doi: 10.1364/OL.4.000205.

[9] J. W. Müller. Dead-time problems. *Nucl. Instrum. Methods*, 112:47–57, 1973. doi: 10.1016/0029-554X(73)90773-8.

[10] G. F. Knoll. *Radiation Detection and Measurement*. Wiley, fourth edition, 2010. ISBN 978-0-470-13148-0.

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

[12] 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.

[13] C. M. Natarajan, M. G. Tanner, and R. H. Hadfield. Superconducting nanowire single-photon detectors: Physics and applications. *Supercond. Sci. Technol.*, 25:063001, 2012. doi: 10.1088/0953-2048/25/6/063001.

[14] Hamamatsu Photonics K.K. *Photomultiplier Tubes: Basics and Applications*. Hamamatsu Photonics K.K., Electron Tube Division, fourth edition, April 2017. URL https://www.hamamatsu.com/content/dam/hamamatsu-photonics/sites/documents/99_SALES_LIBRARY/etd/PMT_handbook_v4E.pdf. Document TOTH9001E04.

[15] A. H. Myerson, D. J. Szwer, S. C. Webster, D. T. C. Allcock, M. J. Curtis, G. Imreh, et al. High-fidelity readout of trapped-ion qubits. *Phys. Rev. Lett.*, 100:200502, 2008. doi: 10.1103/PhysRevLett.100.200502.

[16] D. B. Hume, T. Rosenband, and D. J. Wineland. High-fidelity adaptive qubit detection through repetitive quantum nondemolition measurements. *Phys. Rev. Lett.*, 99:120502, 2007. doi: 10.1103/PhysRevLett.99.120502.

[17] B. D'Anjou, L. Kuret, L. Childress, and W. A. Coish. Maximal adaptive-decision speedups in quantum-state readout. *Phys. Rev. X*, 6:011017, 2016. doi: 10.1103/PhysRevX.6.011017.

[18] M. W. Doherty, N. B. Manson, P. Delaney, F. Jelezko, J. Wrachtrup, and L. C. L. Hollenberg. The nitrogen-vacancy colour centre in diamond. *Phys. Rep.*, 528:1–45, 2013. doi: 10.1016/j.physrep.2013.02.001.

[19] J. M. Taylor, P. Cappellaro, L. Childress, L. Jiang, D. Budker, P. R. Hemmer, et al. High-sensitivity diamond magnetometer with nanoscale resolution. *Nature Phys.*, 4:810–816, 2008. doi: 10.1038/nphys1075.

[20] D. A. Hopper, H. J. Shulevitz, and L. C. Bassett. Spin readout techniques of the nitrogen-vacancy center in diamond. *Micromachines*, 9:437, 2018. doi: 10.3390/mi9090437.

[21] J. F. Barry, J. M. Schloss, E. Bauch, M. J. Turner, C. A. Hart, L. M. Pham, et al. Sensitivity optimization for NV-diamond magnetometry. *Rev. Mod. Phys.*, 92:015004, 2020. doi: 10.1103/RevModPhys.92.015004.

[22] L. Robledo, L. Childress, H. Bernien, B. Hensen, P. F. A. Alkemade, and R. Hanson. High-fidelity projective read-out of a solid-state spin quantum register. *Nature*, 477:574–578, 2011. doi: 10.1038/nature10401.

[23] 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.

[24] M. Beck. Comparing measurements of $g^{(2)}(0)$ performed with different coincidence detection techniques. *J. Opt. Soc. Am. B*, 24:2972–2978, 2007. doi: 10.1364/JOSAB.24.002972.

[25] P. Grangier, G. Roger, and A. Aspect. Experimental evidence for a photon anticorrelation effect on a beam splitter: A new light on single-photon interferences. *Europhys. Lett.*, 1:173–179, 1986. doi: 10.1209/0295-5075/1/4/004.

[26] J. J. Thorn, M. S. Neel, V. W. Donato, G. S. Bergreen, R. E. Davies, and M. Beck. Observing the quantum behavior of light in an undergraduate laboratory. *Am. J. Phys.*, 72:1210–1219, 2004. doi: 10.1119/1.1737397.

[27] D. N. Klyshko. Use of two-photon light for absolute calibration of photoelectric detectors. *Sov. J. Quantum Electron.*, 10:1112–1117, 1980. doi: 10.1070/QE1980v010n09ABEH010660.

[28] 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.

[29] A. Aspect, P. Grangier, and G. Roger. Experimental realization of Einstein–Podolsky–Rosen–Bohm Gedankenexperiment: A new violation of Bell's inequalities. *Phys. Rev. Lett.*, 49:91–94, 1982. doi: 10.1103/PhysRevLett.49.91.

[30] D. Dehlinger and M. W. Mitchell. Entangled photons, nonlocality, and Bell inequalities in the undergraduate laboratory. *Am. J. Phys.*, 70:903–910, 2002. doi: 10.1119/1.1498860.

[31] *7 Series FPGAs SelectIO Resources User Guide*. Xilinx, Inc., May 2018. URL https://docs.amd.com/v/u/en-US/ug471_7Series_SelectIO. UG471 (v1.10).

[32] F. Jelezko, T. Gaebel, I. Popa, A. Gruber, and J. Wrachtrup. Observation of coherent oscillations in a single electron spin. *Phys. Rev. Lett.*, 92:076401, 2004. doi: 10.1103/PhysRevLett.92.076401.

[33] A. Gruber, A. Dräbenstedt, C. Tietz, L. Fleury, J. Wrachtrup, and C. von Borczyskowski. Scanning confocal optical microscopy and magnetic resonance on single defect centers. *Science*, 276:2012–2014, 1997. doi: 10.1126/science.276.5321.2012.

[34] A. Dréau, M. Lesik, L. Rondin, P. Spinicelli, O. Arcizet, J.-F. Roch, et al. Avoiding power broadening in optically detected magnetic resonance of single NV defects for enhanced dc magnetic field sensitivity. *Phys. Rev. B*, 84:195204, 2011. doi: 10.1103/PhysRevB.84.195204.

[35] N. Aslam, G. Waldherr, P. Neumann, F. Jelezko, and J. Wrachtrup. Photo-induced ionization dynamics of the nitrogen vacancy defect in diamond investigated by single-shot charge state detection. *New J. Phys.*, 15:013064, 2013. doi: 10.1088/1367-2630/15/1/013064.

[36] D. Leibfried, R. Blatt, C. Monroe, and D. Wineland. Quantum dynamics of single trapped ions. *Rev. Mod. Phys.*, 75:281–324, 2003. doi: 10.1103/RevModPhys.75.281.

[37] W. Nagourney, J. Sandberg, and H. Dehmelt. Shelved optical electron amplifier: Observation of quantum jumps. *Phys. Rev. Lett.*, 56:2797–2799, 1986. doi: 10.1103/PhysRevLett.56.2797.

[38] J. C. Bergquist, R. G. Hulet, W. M. Itano, and D. J. Wineland. Observation of quantum jumps in a single atom. *Phys. Rev. Lett.*, 57:1699–1702, 1986. doi: 10.1103/PhysRevLett.57.1699.

[39] D. J. Berkeland, J. D. Miller, J. C. Bergquist, W. M. Itano, and D. J. Wineland. Minimization of ion micromotion in a Paul trap. *J. Appl. Phys.*, 83:5025–5033, 1998. doi: 10.1063/1.367318.

[40] N. Schlosser, G. Reymond, I. Protsenko, and P. Grangier. Sub-poissonian loading of single atoms in a microscopic dipole trap. *Nature*, 411:1024–1027, 2001. doi: 10.1038/35082512.

[41] A. Fuhrmanek, R. Bourgain, Y. R. P. Sortais, and A. Browaeys. Free-space lossless state detection of a single trapped atom. *Phys. Rev. Lett.*, 106:133003, 2011. doi: 10.1103/PhysRevLett.106.133003.

[42] M. Endres, H. Bernien, A. Keesling, H. Levine, E. R. Anschuetz, A. Krajenbrink, et al. Atom-by-atom assembly of defect-free one-dimensional cold atom arrays. *Science*, 354:1024–1027, 2016. doi: 10.1126/science.aah3752.

[43] 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.

[44] C. Kurtsiefer, S. Mayer, P. Zarda, and H. Weinfurter. Stable solid-state source of single photons. *Phys. Rev. Lett.*, 85:290–293, 2000. doi: 10.1103/PhysRevLett.85.290.

[45] 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.

[46] 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.

[47] P. Michler, A. Kiraz, C. Becher, W. V. Schoenfeld, P. M. Petroff, L. Zhang, et al. A quantum dot single-photon turnstile device. *Science*, 290:2282–2285, 2000. doi: 10.1126/science.290.5500.2282.

[48] P. Senellart, G. Solomon, and A. White. High-performance semiconductor quantum-dot single-photon sources. *Nature Nanotechnol.*, 12:1026–1039, 2017. doi: 10.1038/nnano.2017.218.