boxcar: user's manual

A boxcar averager and segmented scope for pump–probe experiments, on the Red Pitaya STEMlab 125-14

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

Red Pitaya `boxcar` design, version 1.0 (build 0xe4e10263)
November 10, 2026

Abstract

`boxcar` turns the Red Pitaya into a boxcar averager for pump–probe experiments. It takes every laser shot from a TTL trigger timed to 1 ns (or an analog level, its own timer or software), sorts it into a category (pump on or off, from a chopper's reference or from a pattern it drives onto a modulator), integrates eight gates on the two detector inputs, normalizes the signal by a reference detector shot by shot, and sums everything per point of a delay scan, exactly, for $\Delta T/T$ with its error. Meanwhile it averages whole traces of both inputs per category, stores raw segments, fires up to eight timed pulses per shot (pump, probe and modulator triggers, with an electronic delay scan), and shows running averages on its analog outputs. This manual is for the lab: the first run, every part of the instrument with the board's own recordings, real setups, and the reference. Figures marked "measured" were recorded on the board used here (build 0xe4e10263, 2026-11-09) through a cable loopback, with the built-in test source playing the laser and the sample, and with a second Red Pitaya running `awg` as an independent laser. The physics behind the measurement, with exercises, is in the theory manual (`theory_manual.pdf`). Behaviour with real lasers and detectors has not been measured: no laser was at hand.

Contents

1 What boxcar is **3**
1.1 What it is meant for . . . 3
1.2 Safety first . . . 3
1.3 What was measured, and what was not . . . 4

2 Getting started **4**
2.1 The board and the service . . . 4
2.2 A first measurement, through two cables . . . 4
2.3 The panel and the live window . . . 5
2.4 Presets . . . 5

3 Shots: triggers, categories, points and pulses **5**
3.1 Triggers . . . 6
3.2 Categories . . . 6
3.3 Points and sweeps . . . 6
3.4 Pulse outputs and markers . . . 7

4 The scope: snapshot, averaged traces and segments **7**
4.1 A snapshot . . . 7
4.2 Averaged traces . . . 7
4.3 Segments . . . 8

5 Gates, normalization and the sums **8**
5.1 Gates . . . 8
5.2 Normalization by the reference . . . 9
5.3 The sums per point and category . . . 9
5.4 Gates and the sampling phase . . . 10

6 Delay scans **11**
6.1 Electronic: the board fires the probe . . . 11
6.2 A PC-driven stage . . . 11
6.3 Sweeps and the service's totals . . . 12

7 The analog outputs **12**
7.1 Boxcar outputs . . . 12
7.2 Limits . . . 13
7.3 The test source . . . 13

8 Setups **14**
8.1 An amplified laser with a chopper (1–10 kHz) . . . 14
8.2 A high-repetition laser with a modulator (50 kHz–2 MHz) . . . 14
8.3 The board as the master: an electronic delay scan . . . 14
8.4 A PC-driven delay stage . . . 15
8.5 Two independent signals . . . 15
8.6 Scope only . . . 15
8.7 An emulated laser on a second board . . . 15

9 Reference **16**
9.1 Methods . . . 16
9.2 TCP commands . . . 16
9.3 Records . . . 17
9.4 Specifications . . . 17
9.5 Acknowledgements . . . 17

1 What boxcar is

In a pump–probe experiment a pump pulse excites a sample and a probe pulse, delayed by τ, measures how its transmission (or reflection, or a photocurrent) changed. The change, $\Delta T/T$, is small, often 10^{-3} to 10^{-6}, so every shot counts: the measurement compares shots with the pump on and off, divides by a reference detector that sees the laser's own fluctuations, and averages many shots at each delay [1, 2]. A *boxcar averager* does the electronic part: it integrates each detector pulse in a *gate* (a time window after the trigger), subtracts a baseline gate, and averages the results [3]. `boxcar` does this for every shot, at up to 2 MHz, and keeps the sums per delay point and category exactly, so that the PC gets $\Delta T/T$ and its error instead of a stream of samples (figure 1).

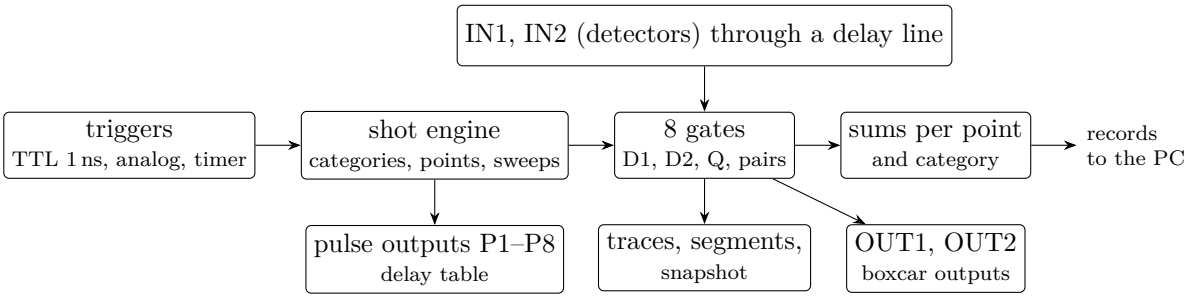


Figure 1: The parts of `boxcar`. Every consumer of the inputs (the gates, the traces, the segments, the snapshot) reads the same delay line, so a gate or a trace may start before its trigger.

1.1 What it is meant for

- amplified lasers at 1–10 kHz: a chopped pump, shot-to-shot $\Delta T/T$, a mechanical delay stage stepped by the PC (the board takes N shots per point and waits);
- lasers at 50 kHz–2 MHz (fibre lasers, cavity-dumped or pulse-picked systems): the pump modulated shot by shot by a modulator that the board drives;
- two lasers fired by the board with an electronic delay (nanoseconds to milliseconds): the board is the master and steps the delay from point to point.

Its inputs are the board's two 14-bit, 125 MS/s converters (IN1, IN2: a signal and a reference detector, or two signals); its digital inputs and outputs are the extension connector's 3.3 V pins (table 1).

Pin	Name	Use
DIO0_P	TRIG	the laser's trigger (its time recorded to 1 ns)
DIO1_P	CHOP	a chopper's reference: the shot's category, with a guard band
DIO2_P	ARM	gates the shots, or NEXT: the next point (a stage's "in position")
DIO3_P	AUX	spare input
DIO0_N–DIO7_N	P1–P8	pulse outputs (2 ns steps), levels, markers, the next category
DIO4_P–DIO7_P	M1–M4	markers (shot, point, sweep, category)
IN1, IN2		the detectors (LV jumpers: ±1 V)
OUT1, OUT2		the boxcar outputs, or the test source

Table 1: The connections. Inputs have pull-downs; the outputs are in high impedance until they are set, and after `SAFE`.

1.2 Safety first

Read the equipment-safety section of the repository's README before wiring. The extension connector's pins are 3.3 V and **not 5 V tolerant**: a laser's or chopper's 5 V TTL output that drives

50 Ω needs a 50 Ω terminator at the board (about 2.5 V then). The inputs IN1 and IN2 take ±1 V on the LV jumpers (±1.13 V on the test board): many photodiode amplifiers give more, so check first and attenuate. A pulse output that fires a laser, a Pockels cell or a shutter needs an external pull-down (e.g. 10 kΩ), so that nothing fires while the board boots or reloads. A boxcar output that feeds a servo or a laser driver gets limits, locked (section 7).

1.3 What was measured, and what was not

Every figure marked "measured" was recorded on the board described here, with two cables (OUT1 → IN1, OUT2 → IN2) and a jumper from P8 to TRIG: the built-in test source plays the laser (a detector-like pulse on OUT1, its reference on OUT2, a sync on a pin) and a sample with a programmed $\Delta T/T(\tau)$. A second board running `awg` then played an independent, asynchronous laser with a chopper (section 8.7). The board test (`scripts/test_hw.py`) passed 40 and 40 of its 40 checks in two runs on this build. Nothing here was measured with a real laser, detector or chopper: those are the physics of the theory manual, not measurements.

2 Getting started

2.1 The board and the service

The board is set up as described in `common/SETUP.md` (or booted from the project's SD card image) and reachable on the network, here at 192.168.0.102. A control service, `rp-overlay`, runs on it: it loads the last active design at boot, restores its configuration, and answers commands on TCP port 5025. Switch the board to `boxcar` with

```
python3 -m rpfpga.cli --host 192.168.0.102 --load boxcar      # from a PC (common/python on
    ↪ PYTHONPATH)
echo "LOAD boxcar" | nc 192.168.0.102 5025                     # or with netcat
```

Loading reprograms the FPGA with everything off: the outputs P1–P8 and M1–M4 in high impedance, OUT1 and OUT2 at 0 V, the test source off, no run. The board's calibration and input ranges are kept (they belong to the board, not to a design).

From a PC, `BoxcarClient` speaks to the service; on the board, the driver `Boxcar` has the same methods. The examples use `dev` for either:

```
import sys, time; sys.path[:0] = ["common/python", "boxcar/control"]
from boxcar import BoxcarClient, driver, model
dev = BoxcarClient("192.168.0.102")
```

All settings are one nested dictionary in physical units: seconds, volts at the connectors (through the board calibration) and plain numbers. `dev.apply(patch)` merges a patch into the configuration and writes it in one step; `dev.config()` returns the configuration as applied (times rounded to the hardware's steps), `dev.status()["warnings"]` the driver's warnings about it, and `dev.describe()` a readable summary of the pins.

2.2 A first measurement, through two cables

Connect OUT1 to IN1 and OUT2 to IN2 with coax. The preset `loopback-demo` turns the test source into a laser at 50 kHz: the board's timer fires output P1, P1 fires the test source, which plays a detector pulse on OUT1 (the "signal", 10 % larger on the pump-on shots: $\Delta T/T = 0.1$) and the same pulse on OUT2 (the "reference"), both with 20 % of shared shot-to-shot noise, like an unstable laser seen by two detectors. Four gates integrate the signal and its baseline on IN1, the reference and its baseline on IN2; their ratio Q is summed per category into POINT records, which the service adds up:

```
dev.load_preset("loopback-demo")
dev.points_clear(); dev.run(); time.sleep(2); dev.stop()
r = dev.points()[0]
print(r["n"], r["Q"]["mean"], r["dtt"]["ratio"], r["dtt"]["ratio_sem"])
```

On the board this gave $\Delta T/T = 0.09926 \pm 2.5e\text{-}05$ from 20000 shots (section 5). The programmed 0.1 is slightly more than the emulated sample really has (0.09979: the test source works with integer amplitude codes, section 7.3), and the measurement agrees with that within $-0.53\,\%$.

2.3 The panel and the live window

In the board's Jupyter, the notebook `boxcar.ipynb` opens a control panel: run and stop, the run's state, a timing view of one shot (the pulse outputs, the gates and the trace window drawn on the time axis), the averaged traces with the gates as bands, the pump–probe curve from the service's totals, and editors for the gates and the boxcar outputs; presets can be loaded and saved there.

```
from rpfpga.plugins import import_plugin
import_plugin("boxcar").make_gui(dev)
```

On a PC, `tools/live.py` draws the traces and the pump–probe curve in a window and updates them every second while the board runs (it only reads; settings stay with the notebook or a program):

```
PYTHONPATH=common/python:boxcar/control python3 boxcar/tools/live.py --host 192.168.0.102
```

2.4 Presets

A preset sets the whole instrument for one setup (table 2); `dev.presets()` lists them with their descriptions, `dev.load_preset(name)` applies one, and `dev.save_preset(name)` keeps the current configuration as a preset of your own on the board. Section 8 describes the wiring of each.

Preset	Setup
`off`	all outputs in high impedance, the test source off
`loopback-demo`	no laser: the test source through OUT1 → IN1, OUT2 → IN2, $\Delta T/T$ 10 %
`loopback-delay-scan`	no laser: a probe delay scan of 41 points through the cables, an emulated decay
`amplified-chopper`	1–10 kHz, TRIG the laser, CHOP the chopper; four gates, Q, traces, boxcar outputs
`high-rep-modulator`	50 kHz–2 MHz, the pump modulated shot by shot by P1, short gates
`master-delay-scan`	the board as the master: P1 the pump, P2 the probe stepped 0–20 µs
`stage-scan`	a PC-driven stage: 2000 shots per point, then wait for NEXT
`two-signals`	two independent signals, their pump-on minus pump-off on OUT1, OUT2
`scope`	averaged traces of 4096 samples, snapshots and segments

Table 2: The built-in presets (each loads and reads back on the board: the board test).

```
for name, text in dev.presets()["builtin"].items():
    print(name, "-", text)
dev.load_preset("amplified-chopper")
print(dev.status()["warnings"])        # none for a preset
```

3 Shots: triggers, categories, points and pulses

A *run* starts with `dev.run()` and ends at its limit (a number of shots, of sweeps, a duration) or with `dev.stop()`. Within it every trigger is an *event*; an event becomes a *shot* if the previous shot's busy time is over (else it is counted as missed) and, with the ARM gate, if ARM is high. Every shot gets a time, a category, a point and a sweep, and starts whatever is set to follow it: gates, traces, segments, pulses, records.

3.1 Triggers

```
dev.apply({"trigger": {"source": "ttl", "edge": "rising", "min_width": 2e-9, "holdoff": 20e-6}})
dev.apply({"trigger": {"source": "analog", "input": "in1", "level": 0.2, "hysteresis": 0.05}})
dev.apply({"trigger": {"source": "timer", "period": 100e-6}})      # the board as the master
dev.apply({"trigger": {"source": "software"}})                     # dev.trigger() fires one
```

- **TTL** on TRIG (DIO0_P) is sampled every nanosecond: the shot's time is the edge's nanosecond. A glitch filter (`min\_width`, 1–8 ns) ignores shorter pulses, and a hold-off ignores edges for a while after one (ringing on a long cable). Through a jumper from P8 to TRIG the board measured a constant loop of 21 ns from its own output to its input, and moving the output by 2 ns steps moved the recorded time by exactly 2 ns (8 of 1000 shots one nanosecond off, at a sampling boundary).
- **Analog**: a threshold in volts on IN1 or IN2 with a hysteresis (a Schmitt trigger), rising or falling. Its time is the 8 ns sample where the input crossed, referred to the connector through the board calibration: through OUT1 → IN1 the shots came 12 ns after the test pulse's start, where its shape crosses the level.
- **Timer**: shots every `period` (8 ns to 34 s) from the run's start; the board is then the master, and its pulse outputs fire the lasers.
- **Software**: `dev.trigger()`, for a slow experiment driven by the PC.

A run's start re-arms the triggers. An edge or a pulse already on its way at the start can be the run's first shot, with a time a little before the start (up to 40 ns for TTL, up to the converters' delay of about 150 ns for the analog trigger); a free-running laser does this.

The busy time and missed shots

`shots.busy` is the time a shot blocks the next ones. Choose it shorter than the laser's period, longer than the slowest thing that follows a shot (a trace, the gates' span): the driver warns when a trace or the gates do not fit (`dev.status()["warnings"]`). Missed and ignored events are counted in `dev.state()`.

3.2 Categories

```
dev.apply({"categories": {"source": "chop", "guard": 1e-6, "invert": False}})    # a chopper on CHOP
dev.apply({"categories": {"source": "pattern", "pattern": "10"}})                 # 1, 0, 1, 0, ...
```

With a chopper, the shot's category is CHOP's level at the trigger (high: 1, the pump on); a shot within `guard` of a chopper edge is *invalid* (a half-blocked pump) and kept apart. With a pattern, the board's own bit sequence (1–16 bits) advances at every trigger, and an output in mode `next` shows the next shot's bit: it drives an AOM, a pulse picker or a shutter so that the pump is on exactly for the category-1 shots. Through a jumper from P7 to CHOP, with the test source's chopper on P7, the categories equalled the test pattern shot by shot (the board test).

3.3 Points and sweeps

A scan has P points (up to 1024) and is repeated in sweeps. The point advances after every shot, after every N shots, at each rising edge of ARM (NEXT), or only when the host sets it. With `hold`, the board takes N shots per point and then waits for NEXT (ARM or `dev.next_point()`): a PC-driven delay stage moves while nothing is measured (section 6).

```
dev.apply({"points": {"mode": "every", "every": 200, "count": 41}, "shots": {"sweeps": 20}})
dev.apply({"points": {"mode": "every", "every": 2000, "count": 100, "hold": True}})
```

3.4 Pulse outputs and markers

Each of P1–P8 (DIO0_N–DIO7_N) in mode `pulse` makes one pulse per shot of the categories it is enabled for, at a delay after the shot's time (1 ns resolution, 2 ns output slots) and of a given width. With `scan`, the delay is $d_0 + T[p]$, T the delay table of the current point: the electronic delay scan. Each output keeps up to 16 pending pulses, so a pulse may come after the next trigger (delays longer than the laser's period, or "negative" delays at high rates). The other modes are levels: low, high, a fixed 2 ns pattern, markers of shots, points, sweeps and triggers, the last shot's category, the next category, the run, the test source's sync and chopper.

```
dev.apply({"outputs": {"P1": {"mode": "pulse", "delay": 2e-6, "width": 100e-9, "categories": [1]},
                       "P2": {"mode": "pulse", "delay": 0.0, "width": 100e-9, "scan": True},
                       "M1": {"mode": "shot"}},
           "points": {"delays": [200e-9 * i for i in range(101)], "count": 101}})
print(dev.describe()["P2"])
```

Outputs are drivers

A pulse output drives its pin from the moment it is set. Never connect an output to another instrument's output, and fit a pull-down on a line that fires a laser (section 1). Measured with an oscilloscope (a TTL edge from awg on a second board): a pulse appears 21.7 ns plus its delay after the edge at TRIG (±0.6 ns rms; every 2 ns step exact); the shortest delay that is on time is 88 ns (the pulse 110 ns after the edge); a shorter one is counted late and starts at about 106 ns. Level outputs (markers, the next category) follow the edge by 74 ns (±2.2 ns rms, whole clocks).

4 The scope: snapshot, averaged traces and segments

The gates are placed on a waveform, so `boxcar` records waveforms too. All three views read the same delay line of the inputs (65 µs): a view may start before its trigger (`pre`), and the driver sizes the line for the view that needs the most.

4.1 A snapshot

One shot's samples of IN1 and IN2 (up to 4096), armed by the host:

```
dev.apply({"snapshot": {"samples": 256, "pre": 400e-9}})
dev.arm_snapshot(); dev.run()
snap = dev.read_snapshot()          # {"t_ns", "dt", "pre", "in1", "in2"} in volts
dev.stop()
```

4.2 Averaged traces

The averager sums whole traces of IN1 and IN2, per category (two traces of up to 2048 points) or one trace (up to 4096 points), each point the sum of 1–64 samples (decimation). It has two halves: one averages while the host reads the other, so no shot is lost. A half closes after N scans (`scans`), optionally only at a point's first shot, or on request; it then *freezes* until the host reads and releases it. `dev.run()` starts with both halves cleared.

```
dev.load_preset("loopback-demo")
dev.apply({"traces": {"enabled": True, "mode": "categories", "points": 160, "decimation": 1,
                      "pre": 480e-9, "scans": 4000}})
dev.run()
blk = dev.trace_block()             # the next frozen half: {"t", "scans", "in1": [cat 0, cat 1],
    ↪ ...}
dev.stop()
```

Figure 2 is what one looks at before gating: the pulse, its baseline, how far its tail reaches. A trace is accepted when the previous one has ended (the driver warns if the traces are longer than the shots' spacing), and the sums are 36-bit: a half that could overflow is flagged as saturated. With

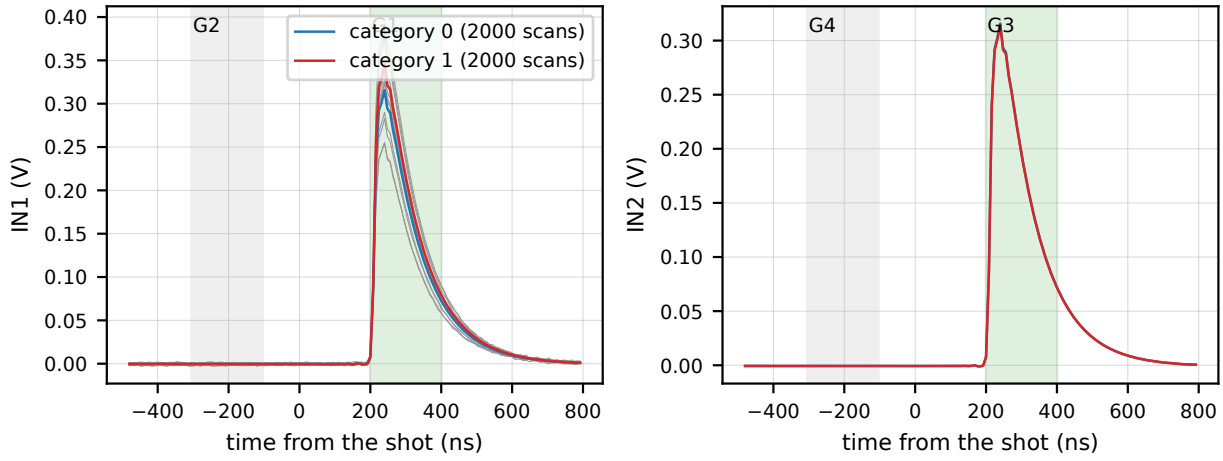


Figure 2: Measured: the averaged traces of the `loopback-demo` preset (2000 scans per category), IN1 the signal and IN2 the reference, with the four gates; grey: twelve raw segments of IN1. The pump-on trace (category 1) peaks at 0.345 V, the pump-off one at 0.316 V, 240 ns after the shot; the shared 20 % laser noise averages out of both.

a real laser, take a pump-on trace with the probe blocked to see the pump's scattered light, and look for saturation of the detector (a flat top) before trusting any $\Delta T/T$.

4.3 Segments

Every shot's raw samples of IN1 and IN2 (up to 8192 each, with a header of the shot's time, number, point, sweep and category) can be written into the board's memory and read by the PC: for an offline analysis, a new choice of gates, or a check.

```
dev.apply({"segments": {"length": 256, "pre": 400e-9, "per_point": 0}})
dev.run()
seg = dev.segments(1000)            # the next 1000 shots: "in1", "in2" (volts), "category", ...
dev.stop()
```

How it is verified

The board test compares the averaged traces with the sum of the raw segments of the same shots, per category, at 5 rates from 1 kHz to 2.016 MHz: the largest difference was 0 codes, i.e. they are equal. The traces lose no shot while the host reads the other half.

5 Gates, normalization and the sums

5.1 Gates

Each of the eight gates G1–G8 sums the samples of IN1 or IN2 in a window: a delay from the shot (in 8 ns samples, also negative, down to 65 µs before it) and a width (8 ns to 524 µs). The sums are exact 32-bit integers of the converters' codes; `point_results` turns them into the mean level over the gate in volts. A gate may follow the probe in a delay scan (`scan`, its own table `points.gate\_delays`).

```
dev.apply({"gates": {"G1": {"enabled": True, "input": "in1", "delay": 200e-9, "width": 200e-9},
                     "G2": {"enabled": True, "input": "in1", "delay": -300e-9, "width": 200e-9},
                     "G3": {"enabled": True, "input": "in2", "delay": 200e-9, "width": 200e-9},
                     "G4": {"enabled": True, "input": "in2", "delay": -300e-9, "width": 200e-9}},
           "derived": {"d1": {"signal": "G1", "baseline": "G2"}, "d2": {"signal": "G3", "baseline":
    ↪ "G4"}}})
```

D1, D2 and Q. $D_1 = G_a - (w_a/w_b)\,G_b$ subtracts a baseline gate G_b (any width) from a signal gate G_a on the same input: offsets and drifts slower than the gates' distance cancel. D_2 does the same for the reference. $Q = D_1/D_2$ (24 fractional bits) divides the signal by the reference, shot by shot: the laser's pulse energy, seen by both detectors, cancels.

The gates' timing. A shot's gates are taken if they start after the previous shot's have ended (at least 8 clocks) and the previous shot's processing (48 clocks) is done: every shot up to about 2.6 MHz. A gate's edges are on the 8 ns sample grid while the laser's pulse falls anywhere within a sample; put the edges where the pulse is flat (section 5.4).

5.2 Normalization by the reference

In `loopback-demo` both outputs carry the same 20 % of shot-to-shot "laser noise". Figure 3 shows what the ratio does with it: D1 and D2 spread by 11.5 % and 11.5 %, Q by 0.165 %. Without the shared noise Q spreads by 0.162 %: the ratio removed the laser's fluctuation down to what the converters and the test source themselves add (D1 alone spreads 0.255 % then).

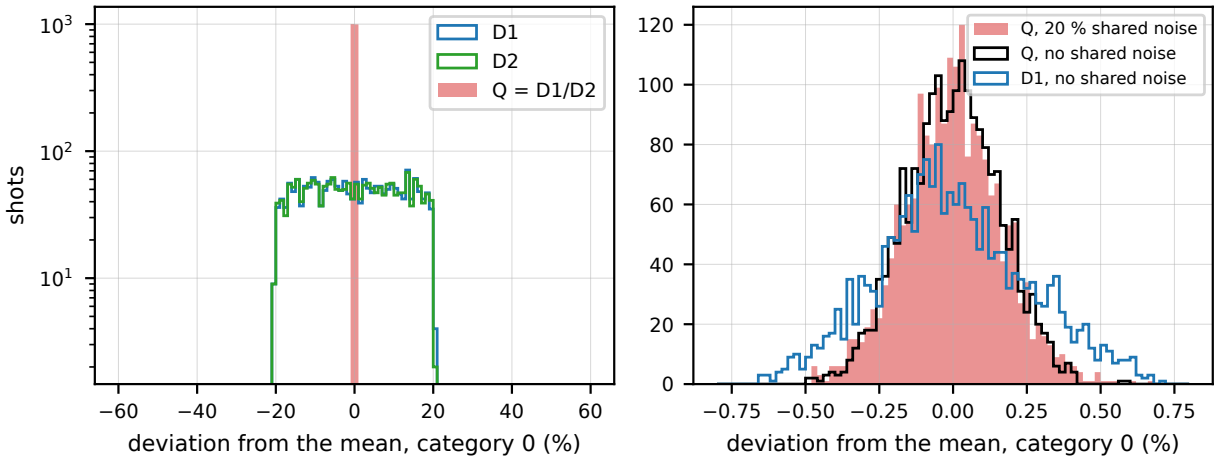


Figure 3: Measured: the shot-to-shot spread of D1, D2 and Q (20000 shots of `loopback-demo`, category 0). Left: with 20 % of shared noise (log scale). Right: Q with and without it, and D1 without it.

5.3 The sums per point and category

For every point and category the board sums, exactly, the number of shots, $\sum x$ and $\sum x^2$ of G1–G8, D1, D2 and Q (48-bit and 80-bit), and $\sum D_1 D_2$; for neighbouring shots of different categories it sums the pair differences $d = Q_{\text{on}} - Q_{\text{off}}$ and d^2. At each point's end (or every 65,536 shots) the sums go out as a POINT record; the service adds them over sweeps. `dev.points()` returns, per point, each value's mean and standard error per category and

$$\Delta T/T = \frac{\langle Q\rangle_1}{\langle Q\rangle_0} - 1 \quad (\text{"ratio"}), \qquad \Delta T/T = \frac{\langle d\rangle}{\langle Q\rangle_0} \quad (\text{"pairs"}), \tag{1}$$

each with its error to first order (theory manual, section 6).

```
dev.load_preset("loopback-demo")
dev.points_clear(); dev.run(); time.sleep(2); dev.stop()
r = dev.points()[0]
print(r["D1"]["mean"], r["D1"]["sem"], r["Q"]["mean"], r["dtt"])
```

The error falls as $1/\sqrt{N}$ (figure 4): ten times more shots, $\sqrt{10}$ times smaller an error. The ratio and the pairs agree here because the test source's noise is white from shot to shot; with a real laser's slow drifts the pairs are better (theory manual, section 5).

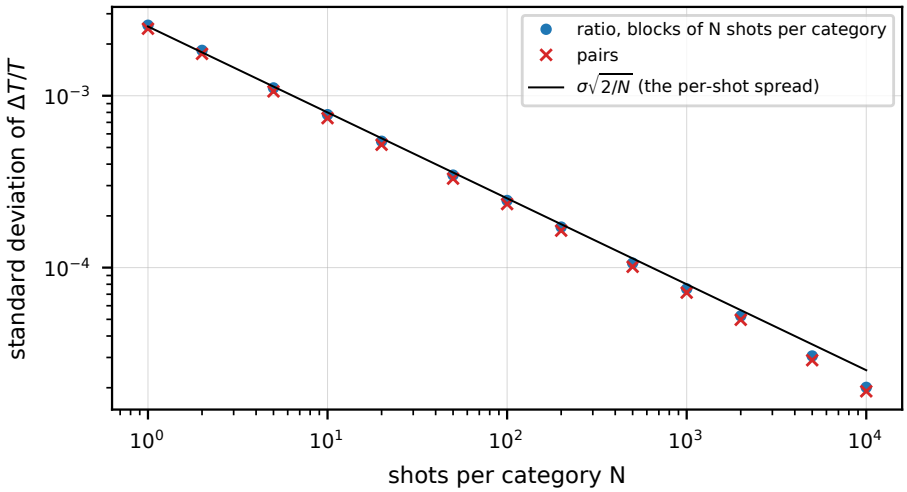


Figure 4: Measured: the standard deviation of $\Delta T/T$ from blocks of N shots per category (400000 shots of `loopback-demo`) against N, and the white-noise expectation $\sigma\sqrt{2/N}$ from the per-shot spread $\sigma = 0.165\,\%$ of Q. The slope is -0.515; at $N = 1000$, 7.5e−05 against 8.0e−05 expected.

How it is verified

The board test recomputes every gate from raw segments and compares them with the SHOT records' values (G1–G4, D1, D2, Q: all 2000 equal), sums the SHOT records per point and category and compares them with the POINT records (all equal), and runs the gates at 2.016 MHz (every shot of 100,000 taken). Through the loopback at 2.016 MHz, 10000000 shots took 5.3 s, every gate was taken (0 missed), and $\Delta T/T$ came out $0.009718 \pm 2.4e\text{-}06$ for an emulated 0.009755.

5.4 Gates and the sampling phase

With a laser that is not synchronous with the board's clock, its pulse falls anywhere within an 8 ns sample. A gate that holds the whole pulse sums almost the same at every position; a gate whose edge cuts the pulse does not. Figure 5 shows it on the board: the test source on its own timer (every 10003 ns, so its pulses run through every nanosecond of the sample grid) with its sync on P8 into TRIG, 10000 shots per shape. For a smooth pulse a tight gate of two samples varies by ±16.9 %, a 10-sample gate that cuts the tail by ±3.4 %, and a gate holding the whole pulse by ±1.5 %. A pulse shorter than the samples (rise 1 ns, decay 6 ns) varies by ±41.2 % even in the widest gate: the test source plays it as 8 ns samples too, so its own area depends on the position (a real detector's pulse, smoothed by the inputs' 60 MHz, does not; theory manual, section 3).

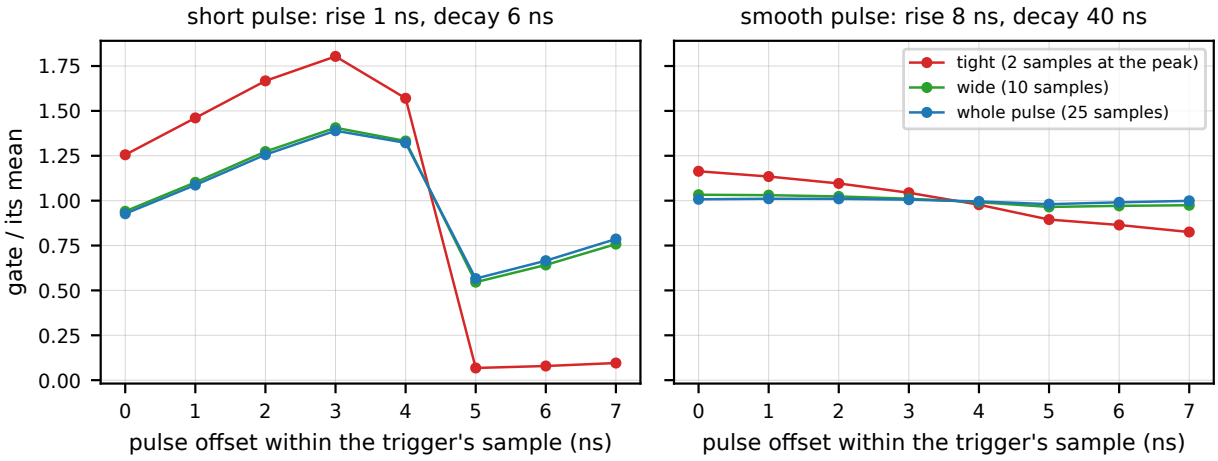


Figure 5: Measured: gate sums (baseline subtracted, relative to their mean) against the pulse's position within the trigger's 8 ns sample, for a short and a smooth pulse of the test source and three gates.

6 Delay scans

6.1 Electronic: the board fires the probe

When the lasers accept triggers, the board steps the delay itself: a pulse output fires the pump on category-1 shots, another fires the probe at $d_0 + T[p]$, T the delay table (up to 1024 entries, 1 ns resolution, 2 ns output slots): linear, logarithmic for a decay over decades, or shuffled so that a slow drift does not follow τ (theory manual, section 7). The gates follow the probe's detector pulse with their own table, which must hold whole samples (multiples of 8 ns).

```
dev.load_preset("master-delay-scan")
grid = [round(10 ** (-8 + 3 * i / 99) / 8e-9) * 8e-9 for i in range(100)]   # 10 ns .. 10 us
dev.apply({"points": {"count": len(grid), "delays": grid, "gate_delays": grid}})
print(dev.status()["warnings"])
```

The preset `loopback-delay-scan` does this through the cables: P1 is the probe and fires the test source, stepped from 0.2 to 8.2 µs in 41 points of 200 shots; P2 is the pump at 1.2 µs (drawn on its pin, not used by the source); the gates follow the probe; and the test source's sample answers with $\Delta T/T = 5\,\% \exp(-\tau/2\,\mu\text{s})$ after the pump. Figure 6 is the result of 20 sweeps (164000 shots, 3.3 s).

At the pump's point the measurement gave 0.04955 for an emulated 0.04978 (programmed 0.05): the residuals stay within 0.00025, a typical error bar is 5.2e−05, and they scale with $\Delta T/T$ (−0.53 % of it: a differential gain of the converters at this signal level, section 7.3). The point just before the pump reads 0.00043 instead of 0: its last shot is played with the next point's $\Delta T/T$ (the test source reads the point after the advance), 0.05/100 for its 100 category-1 shots.

6.2 A PC-driven stage

For an optical delay line the board takes N shots per point and waits: with `hold`, after a point's N-th shot it accepts no shot (the triggers are only counted) until NEXT, a rising edge on ARM (the stage controller's "in position") or `dev.next_point()`.

```
def move_stage(position_mm):
    pass                                    # your stage's driver

def wait_held():
    while not dev.state()["held"] and dev.state()["running"]:
        time.sleep(0.01)

dev.load_preset("stage-scan")               # 2000 shots per point, 100 points
positions = [0.005 * p for p in range(100)]
```

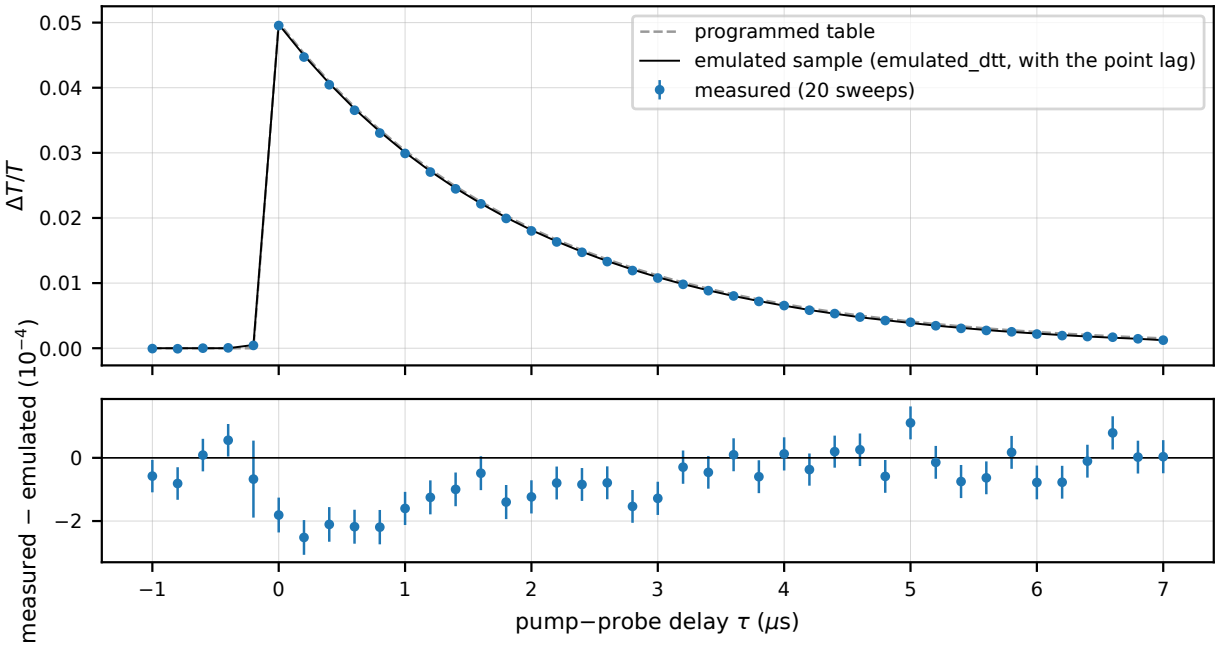


Figure 6: Measured: the `loopback-delay-scan` preset, 20 sweeps of 41 points. Top: $\Delta T/T$ from the service's POINT totals, the programmed table, and the emulated sample's exact values (the table less the test source's truncation, with the shot that ends a point playing the next point's value). Bottom: measured minus emulated.

```
move_stage(positions[0])
dev.points_clear(); dev.run()
for p in range(1, len(positions)):
    wait_held()                              # point p - 1 is done
    move_stage(positions[p])
    dev.next_point()                         # point p starts
wait_held(); dev.stop()
curve = [(positions[p], r["dtt"]["ratio"], r["dtt"]["ratio_sem"]) for p, r in dev.points().items()]
```

6.3 Sweeps and the service's totals

A scan repeated in sweeps, and added, averages slow drifts over the whole curve instead of letting them distort one part of it (theory manual, section 7). The service's worker adds every POINT record into totals per point, also across runs, until `dev.points_clear()`; a point longer than 65,536 shots arrives in parts and is added too, which gives live updates during long points.
TCP: `POINTS?` `POINTS CLEAR`

7 The analog outputs

OUT1 and OUT2 each show a DC level (`dc`), the test source, or a boxcar output.

7.1 Boxcar outputs

A boxcar output keeps a running average per category of one value (a gate, D1, D2 or Q): with every shot of its category, $A \leftarrow A + (s - A)/2^{\tau}$, seeded by the run's first shot of that category, an exponential average over about `average` $= 2^{\tau}$ shots. It shows category 1 (`on`), category 0 (`off`) or their difference, in volts per unit of the value (`scale`: 1 V per volt of a gate's mean level, per unit of Q). This is the classic boxcar output for a chart recorder or a servo. $\Delta T/T$ itself is formed on the PC.

```
dev.load_preset("loopback-demo")
dev.apply({"test": {"out2": {"mode": "off"}},
           "analog": {"out2": {"mode": "boxcar", "source": "D1", "show": "on", "average": 16, "scale
    ↪ ": 1.0}},
           "limits": {"out2": {"low": -0.25, "high": 0.25}}})
print(dev.config()["analog"]["out2"])
```

Figure 7 shows a boxcar output at work: OUT2 shows D1 of the pump-on shots averaged over 16 shots while the test source's sample steps its $\Delta T/T$ between 0 and 20 % every 400 shots; OUT2 is read back through OUT2 → IN2 by a wide gate before each shot. The read-back follows the average computed from the shots' D1 values with a scatter of 0.8 mV and an offset of −7.7 mV (the two converters' calibration).

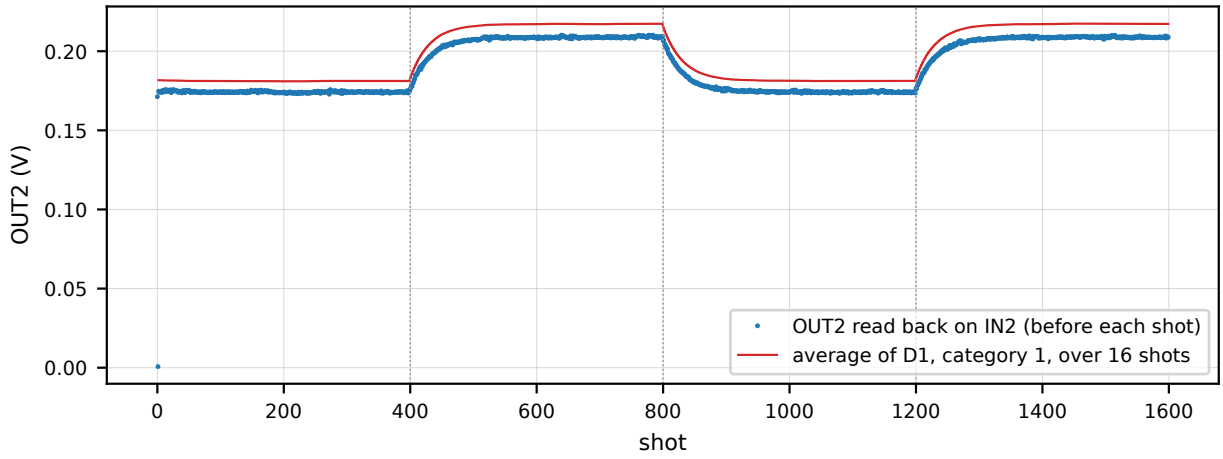


Figure 7: Measured: OUT2 as a boxcar output of D1 (category 1, 16 shots) while the emulated $\Delta T/T$ steps between 0 and 20 % every 400 shots (dotted lines): the level read back on IN2 before each shot, and the average of the shots' D1 values.

A shot's value reaches the outputs, measured with an oscilloscope, 901 ns (OUT1) and 1085 ns (OUT2, 183.9 ns later as designed) after the end of its last gate at the IN connector, ±2.3 ns (whole clocks against an asynchronous laser): the processing's 75 and 98 clocks, the delay line (7 clocks here, plus 7: each clock of a longer pre-trigger anywhere adds 8 ns) and about 190 ns of the two converters.

7.2 Limits

Every word that goes to an output, in any mode, is clamped between `limits.low` and `limits.high` (volts; `None`: the output's range). `dev.lock_limits()` makes the hardware ignore later changes of the limits until the FPGA is loaded again: a servo's input or a laser's modulation input stays protected whatever a program does. The board test reads a limited output back: it stayed at 0.0500 V for a 0.05 V limit (within 3.4 mV).

7.3 The test source

The test source turns OUT1 and OUT2 into a laser and a sample for a cable loopback. Its shots come from its own timer, or from a pulse output's pulses (master mode: each pulse is a laser shot). Each shot plays a detector-like pulse (a shape at 1 ns resolution: rise and decay times, or a table) on each output: in `reference` mode with the amplitude times a shared noise factor (the laser's fluctuation), in `signal` mode also times $(1 + \Delta)$ on category-1 shots of its own pattern, Δ from a table per point (the sample's $\Delta T/T$). Its sync and its chopper level can go to pins (`test\_sync`, `test\_chop`), so a jumper to TRIG or CHOP makes it an asynchronous laser with a chopper.

```
dev.apply({"test": {"on": True, "with_run": True, "period": 100e-6, "pattern": "01", "noise": 0.2,
                    "dtt": [0.0, 0.02, 0.05],
                    "out1": {"mode": "signal", "amplitude": 0.3},
                    "out2": {"mode": "reference", "amplitude": 0.3},
                    "shape": {"rise": 20e-9, "decay": 100e-9}}})
print(driver.emulated_dtt(dev.config(), dev.calibration()))
```

What the emulated sample really has. The test source works with integer amplitude codes (2167 at 0.3 V on this board), and its product $A\Delta/2^{16}$ is truncated: its sample's $\Delta T/T$ is a little smaller than the table, by up to $1/(A\Delta)$ of it (3 % for $\Delta = 0.01$ at 0.3 V). `driver.emulated_dtt` computes the exact values (bit-exact with the hardware, averaged over its noise); compare a measurement with those, not with the table. Through the cables the board measured 0.09926 for an emulated 0.09979 (0.1 programmed), and 0.009718 for 0.009755 (0.01): −0.53 % and −0.38 %, a differential gain of the converters' chain at this signal level (a 10 % step of the amplitude itself reads 1.0999, and at 0.15 V and 0.6 V the measurement agrees within 0.02 %). The crosstalk between the two channels is 0.17 % (OUT1 into IN2) and 0.17 % (OUT2 into IN1); shots that end a point play the next point's value (section 6).

8 Setups

Each setup below is a preset; the code is in `RECIPES.md`. Times in the presets (the gates' delays in particular) are starting points: place the gates on the averaged traces of your own detectors (section 4).

8.1 An amplified laser with a chopper (1–10 kHz)

From	To	
the laser's sync (TTL)	TRIG, DIO0_P	50 Ω terminator if it drives 5 V into 50 Ω
the chopper's reference	CHOP, DIO1_P	high: the pump passes
the signal detector (after the sample)	IN1	within ±1 V
the reference detector (before the sample)	IN2	within ±1 V
OUT1, OUT2	a recorder, a scope	optional

`amplified-chopper`: TTL shots with a 20 µs hold-off, categories from CHOP with a 1 µs guard, the signal gate and its baseline on IN1, the reference and its baseline on IN2, Q, every SHOT record to the PC, a POINT record every 1000 shots, traces per category, OUT1 the difference of Q (10 V per unit, 256 shots), OUT2 D1 of the pump-on shots, both within ±0.5 V. A chopper at half the laser's rate, locked to the laser, alternates the categories shot by shot (the best rejection of the laser's slow noise, theory manual section 5).

8.2 A high-repetition laser with a modulator (50 kHz–2 MHz)

`high-rep-modulator`: TTL shots, categories 1, 0, 1, 0, ... from the board's pattern, P1 (DIO0_N, mode `next`) driving the modulator (an AOM driver's gate, a pulse picker) with the next shot's category, 48 ns gates with their baselines 64 ns before, POINT records every 65,536 shots (no SHOT records: the sums carry the result), traces of 48 samples. The modulator has the laser's period minus 74 ns (the TTL edge to the pin, measured) to settle: check it on a scope. Through the loopback at 2.016 MHz the board took every shot's gates (10000000 shots).

8.3 The board as the master: an electronic delay scan

`master-delay-scan`: the timer at 10 kHz; P1 fires the pump laser on category-1 shots at 2 µs, P2 the probe at 0–20 µs in 101 points of 200 shots (a pump–probe delay of −2 to 18 µs), the gates following

the probe (its detector pulse assumed 200 ns after P2). Both lasers must accept a trigger with a jitter smaller than the dynamics; the board's own timing is exact to its 2 ns steps.

8.4 A PC-driven delay stage

`stage-scan`: TTL and CHOP as above, 2000 shots per point, then the board holds until NEXT (ARM, `DIO2_P`, from the stage controller, or the host's `next_point()`), 100 points (section 6).

8.5 Two independent signals

`two-signals`: D1 on IN1 and D2 on IN2 are two signals of their own (two probe wavelengths, a transmission and a reflection); OUT1 and OUT2 show their pump-on minus pump-off differences.

8.6 Scope only

`scope`: averaged traces of 4096 samples of IN1 and IN2 from 2 µs before the trigger, snapshots and raw segments of 1024 samples; the busy time (33 µs) makes triggers during a trace missed.

8.7 An emulated laser on a second board

A second Red Pitaya running `awg` plays an independent, asynchronous laser: its digital outputs give the sync (D0 into TRIG) and a chopper's reference (D1 into CHOP), its analog outputs the detectors' pulses (OUT1 into IN1, OUT2 into IN2), with a known $\Delta T/T$ in its own 14-bit codes; boxcar's P1 can trigger it instead (TRIG A) for an electronic delay scan; its D2 can be a stage's "in position" (ARM). The repository's guide `guides/combining` has the wiring; `RECIPES.md` the code. In a session on 2026-11-09 (receipt `docs/data/board-1.0-awg-session-2026-11-09.json`):

- At 5 rates from 1 kHz to 1.95 MHz (up to 4000000 shots) every shot was taken, every trigger was a shot, the categories alternated exactly with awg's chopper, and no shot was missed, ignored or invalid (0 in all); awg's crystal ran −4.3 ppm off boxcar's. $\Delta T/T$ read 0.02024–0.02030 for awg's 0.019984.
- That difference is the converters' linearity: against pulse heights from 0.14 to 0.68 V at IN1 and both signs of $\Delta T/T$ the measurement stays within 4.6×10^{-4} of awg's exact value, an offset rather than a gain (figure 8, right), 8.6×10^{-4} at 0.8 V; with $\Delta T/T$ programmed 0, $4{\times}10^{-5}$.
- A chopper edge close to the gate couples into the inputs: with the chopper's falling edge about 60 ns before the gate on pump-on shots only, $\Delta T/T$ gained 4.3×10^{-4}; with the same pattern but awg's D1 in high impedance, 0.2×10^{-4}. Keep chopper and modulator edges away from the gates (the presets put them half a period away) and their wires away from the analog cables.
- Shared shot-to-shot noise from awg (a common factor spreading 5.78 %): D_1 spread 5.72 %, Q 0.257 %, $\Delta T/T$ 0.01990.
- The sampling phase with a truly asynchronous laser (figure 8, left): a tight gate varied by ±35 % with the trigger's nanosecond, a gate holding the whole pulse by ±0.05 % (awg's pulses are band-limited; compare the test source's in figure 5).
- The chopper guard counts to the clock where CHOP is read, two clocks after the trigger's period: with an edge 200 ns before each pump-on trigger no shot was invalid up to a guard of 216 ns and every pump-on shot from 232 ns. Choose the guard about 16 ns longer than the edge's distance you want to reject.
- The hold with NEXT from awg's D2: exactly 30 shots in each of 10 points, each point starting at its NEXT.

- boxcar as the master (P1 into awg's TRIG A, awg answering with a pulse): awg's pulse rose at IN1 392 ns after P1; over a delay scan of 21 points (0.5–10.5 µs) with the gates following, D_1 stayed the same within ±0.05 %.

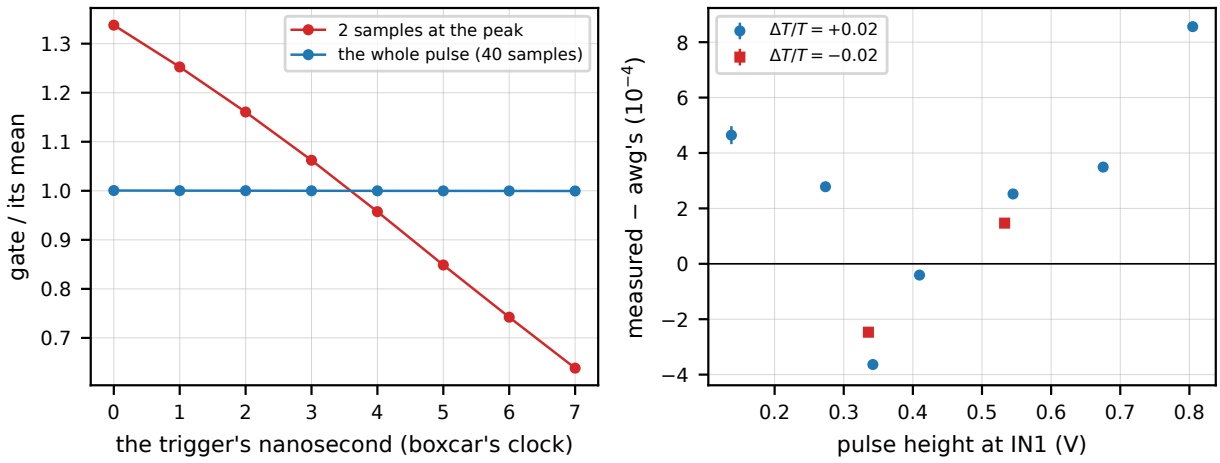


Figure 8: Measured with awg on a second board as the laser. Left: two gates against the trigger's nanosecond (20,000 shots at 10 kHz, awg 4.3 ppm off boxcar's clock). Right: measured $\Delta T/T$ minus awg's exact value against the pulse height at IN1, for $\Delta T/T = \pm 0.02$ (400,000 shots each at 100 kHz).

9 Reference

The complete definitions (every register, every clock) are in `docs/THEORY.md` section 2 and `docs/REGISTERS.md`; the TCP commands in `docs/PROTOCOL.md`; the dated test records in `docs/VERIFICATION.md`. A bit-exact Python model of the whole core (`control/boxcar/model.py`) is the reference that the hardware is tested against, clock by clock.

9.1 Methods

Method	Does
`apply(patch)`, `config()`	change and read the configuration (physical units)
`load_preset(name)`, `presets()`	the presets
`run()`, `stop()`, `state()`	a run and its counts (shots, missed, point, sweep, . . .)
`trigger()`, `next_point()`	a software trigger; NEXT (releases a hold)
`arm_snapshot()`, `read_snapshot()`	one shot's samples
`trace_block(swap)`, `trace_status()`	the next frozen half of the averaged traces
`segments(n)`	the next n shots' raw samples
`points()`, `points_clear()`	the service's POINT totals: means, errors, $\Delta T/T$ per point
`records()`	the record stream to a PC (`collect()`: its words)
`lock_limits()`	the outputs' limits fixed until the FPGA is reloaded
`safe()`	every output released, the run stopped

9.2 TCP commands

The service answers text commands on port 5025 (one per line, `OK` or `ERR` and a reason): the core's (`CONFIG`, `CONFIG?`, `PRESET`, `SAFE`, . . .) and `boxcar`'s:

```
RUN  STOP  TRIGGER  NEXT  STATE?  SNAPSHOT ARM  SNAPSHOT DATA?  PINS? [CLEAR]  HARDWARE?
TRACE SWAP  TRACE RELEASE  TRACE STATUS?  TRACE? [SWAP]  SEGMENTS? n [s]
POINTS? [CLEAR]  POINTS CLEAR  WORKER?  RECORDS STREAM  LIMITS LOCK
```

9.3 Records

RUN (start and end), SWEEP, SHOT (the trigger's time to 1 ns, the point, the category, the sweep, the shot's number and the chosen values: G1–G8, D1, D2, Q), POINT (106 words: the sums per category, the pairs), BLOCK (a frozen half of the traces), SEGMENT and ERROR records go into a ring in the board's memory; the service's worker reads them (the POINT totals) and streams them to a PC (`RECORDS STREAM`, `BoxcarClient.records()`); `model.parse_records` decodes them. SHOT records reach a PC losslessly up to about 0.5 M shots/s (every M-th above).

```
dev.load_preset("amplified-chopper")
dev.apply({"shots": {"limit": 10000}})
mask = sum(1 << model.SUM_NAMES.index(v) for v in dev.config()["records"]["values"])
with dev.records() as stream:
    dev.run()
    words = stream.collect(timeout=1.0)
shots = [model.shot_fields(info, ws, mask) for typ, info, ws in model.parse_records(words)
         if typ == model.REC_SHOT]
```

9.4 Specifications

Inputs	IN1, IN2: 14 bits at 125 MS/s, ±1 V (LV), about 60 MHz; TRIG, CHOP, ARM, AUX: 3.3 V, 1 ns
Triggers	TTL (1 ns, glitch filter 1–8 ns, hold-off), analog (8 ns), timer (8 ns–34 s), software
Shots	up to about 2.6 MHz with every gate (2.016 MHz measured); categories, up to 1024 points
Gates	8 per shot, 8 ns–524 µs, from 65 µs before the shot; exact 32-bit sums
Sums	per point and category: n, $\sum x$, $\sum x^2$ (48/80 bits), $\sum D_1 D_2$, pairs
Traces	2 × 2048 or 4096 points, 1–64 samples each, 36-bit sums, no dead time
Segments	up to 8192 samples per shot into the board's memory
Pulse outputs	P1–P8: delay (1 ns), width, per category, a delay table; 16 pending each
Analog outputs	boxcar outputs (an average over 1–2^{20} shots) or the test source; lockable limits

9.5 Acknowledgements

The framework builds on the Red Pitaya FPGA notes of FPGA-Notes-for-Scientists and Pynq-Redpitaya-125 (dspsandbox).

References

[1] Jagdeep Shah. *Ultrafast Spectroscopy of Semiconductors and Semiconductor Nanostructures*. Springer, Berlin, 2nd edition, 1999.

[2] Christopher A. Werley, Stephanie M. Teo, and Keith A. Nelson. Pulsed laser noise analysis and pump-probe signal detection with a data acquisition card. *Review of Scientific Instruments*, 82: 123108, 2011.

[3] *Model SR250 Gated Integrator and Boxcar Averager: Operation and Service Manual*. Stanford Research Systems, Sunnyvale, CA.