feedback: user's manual

A lock box with fast PID controllers, lock acquisition, lock-in detection and nanosecond timing, on the Red Pitaya STEMlab 125-14

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

Red Pitaya `feedback` design, version 2.0 (build 0x46632204)
October 27, 2026

Abstract

`feedback` turns the Red Pitaya into a lock box. Two PID controllers run at the full sample rate on an input or a demodulated error signal and act on an output 232 ns later; two lock acquisitions scan, engage and relock them; four slow PIDs act on lock-in readings, the inputs' means, the slow analog inputs or a TTL reference, driving oscillators, outputs or the slow analog outputs. The loops measure their own plants and choose their gains for the margins you ask. A TTL input timed to a nanosecond locks an oscillator to a lab clock, a sync output repeats an oscillator with 2 ns edges, and status pins report the locks. This manual is for the lab: the first loop, every setting and how it behaves, lock acquisition, tuning, the slow and digital I/O, the software, calibration, applications, specifications and the command reference. Figures marked "measured" were recorded on the board used here, in loopback; the others are computed. How it works, with derivations, is in the theory manual shared with `lockin` (`lockin/docs/manual/theory_manual.pdf`).

Contents

1 What feedback is **3**
1.1 What makes it different . . . 3
1.2 The documents . . . 4

2 Before you connect equipment **4**

3 Getting started **5**
3.1 The board and the service . . . 5
3.2 A first loop . . . 5
3.3 The control panel . . . 5
3.4 Presets . . . 6

4 The lock-in core **6**

5 Fast PIDs **6**
5.1 Settings . . . 7
5.2 Latency and bandwidth . . . 7
5.3 The low-pass sections . . . 7
5.4 Demodulated error signals . . . 7
5.5 Two actuators . . . 8

6 Lock acquisition **8**
6.1 Settings . . . 8
6.2 How it behaves . . . 9
6.3 Gating slow PIDs and status pins . . . 10

7 Measuring and tuning loops **10**
7.1 Fast PIDs . . . 10
7.2 Slow PIDs . . . 11
7.3 The static methods . . . 11

8 Slow PIDs and phase-locked loops **11**
8.1 Settings . . . 12
8.2 Typical slow loops . . . 12
8.3 PLLs on an input . . . 12

9 The TTL reference, the sync output and the status pins **13**
9.1 The TTL reference . . . 13
9.2 The sync output . . . 14
9.3 The capture trigger . . . 14
9.4 Status pins . . . 14
9.5 Calibration of the pins . . . 14

10 Slow analog outputs and inputs **16**

11 Software **17**
11.1 Python on the board and on a PC . . . 17
11.2 TCP and the command line . . . 17
11.3 The data stream . . . 17

12 Calibration **17**

13 Applications **18**
13.1 Pound–Drever–Hall . . . 18
13.2 Side of fringe with scan and relock . . . 18
13.3 A laser on a cavity with two actuators . . . 19
13.4 A loop's suppression . . . 19
13.5 A PLL to a lab clock, and a clean copy . . . 19
13.6 A temperature on a heater . . . 19

14 Specifications **19**

A Command reference **20**
A.1 feedback's TCP commands (port 5025) . . . 20
A.2 Python . . . 21

B The data stream's rows **21**

1 What feedback is

`feedback` is an FPGA design for the Red Pitaya STEMlab 125-14 together with the software that controls it. With it the board becomes a *lock box*: the controller that holds a laser on a cavity, a cavity on a laser, an interferometer on a fringe, a resonator on its frequency, a temperature on its setpoint. It combines

Two fast PID controllers
: at the full sample rate (125 MSa/s), each from IN1, IN2, IN1 − IN2 or a demodulator's mixer product (a Pound–Drever–Hall or dither-lock error signal), through two low-pass sections, with P, I and D, added to OUT1 or OUT2 on every sample: 232 ns from input to output connector.

Two lock acquisitions
: one per fast PID: a triangle scan on its output that stops when a signal passes an engage level, hands over to the fast PID, and resumes when the lock is lost.

A lock-in core
: two oscillators, two demodulators with 1 to 8 RC sections, readings of X, Y, R, θ and $d\theta/dt$ in volts and degrees at the connectors (the core of the `lockin` design, cut to two units).

Four slow PID controllers
: at the frame rate (976.5625 kHz) from any reading, an input's mean, a slow analog input or the TTL reference to an oscillator's frequency or phase, an output's amplitude or offset, or a slow analog output; optionally active only while a fast loop is locked.

Slow analog I/O
: four slow outputs AO0–AO3 (PWM with a second-order delta-sigma, 0 to 1.8 V) and four slow inputs AI0–AI3 (the XADC, 0 to 7 V) on the extension connector E2.

Digital timing
: a TTL reference input timed to 1 ns (an oscillator locked to a lab clock, also at N/M of its frequency), a sync output (an oscillator's square wave with 2 ns edges), a capture trigger timed to 1 ns, four status pins (locked, scanning, a scan's direction, . . .).

Measured tuning
: the fast PIDs measure their plant's frequency response and the slow PIDs their step responses, design PI gains for chosen phase and gain margins, and verify the closed loop.

Data
: a continuous stream of X and Y, a four-channel capture (with the fast PIDs' errors and outputs at full rate), a frequency counter.

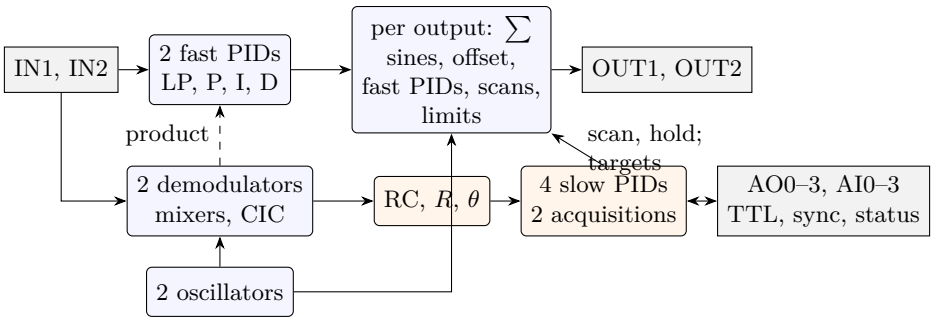


Figure 1: feedback's signal paths. Blue: every sample; orange: every frame (976.5625 kHz).

1.1 What makes it different

- **Fast and slow in one box.** A fast loop on the full-rate signal, a slow loop on a lock-in reading or a slow sensor, and the acquisition logic that coordinates them, without a computer

in any loop.

- **Loops that measure themselves.** `tune_fast` and `tune_pid` measure the actual plant (actuator, cable, experiment) and choose the gains for the margins you ask; the closed loop is measured afterwards to confirm them.
- **Real volts, real phases.** All settings and readings are volts at the connectors through the board calibration, and phases, the TTL reference's and the sync output's included, are referred to the OUT1 connector to about a nanosecond.
- **Exact arithmetic.** The whole signal path is integer arithmetic that a Python model reproduces bit for bit; the FPGA's simulation benches compare every sample against it.

1.2 The documents

document	contents
this manual	operating feedback: settings, fast and slow loops, acquisition, tuning, digital and slow I/O, software, calibration, applications, specifications, the command reference
`lockin/docs/manual/theory_manual.pdf`	how lockin and feedback work, with derivations: the fast PIDs' delay budget, lock acquisition, loop measurement and tuning, the 1 ns pins, the slow outputs' delta-sigma
`lockin/docs/manual/user_manual.pdf`	the lock-in core's measurements in detail (sweeps, spectra, noise, the stream)
`docs/PROTOCOL.md`, `RECIPES.md`	every setting and command in brief; worked examples
`docs/THEORY.md`, `docs/REGISTERS.md`, `docs/VERIFICATION.md`	internals, registers, dated test records

Figures marked *measured* were recorded on the board used throughout, in loopback (OUT1 → IN1, OUT2 → IN2 with 50 Ω terminators, LV jumpers; DIO1_P → DIO0_P and AO0/AO1 → AI0/AI1 jumpers), dated in the captions; the others are computed.

2 Before you connect equipment

Read the repository's section *Connecting to lab equipment: read first* (the root `README.md`). For feedback in particular:

- **It drives actuators.** A fast PID or a scan moves its output by up to its own `limits`; slow PIDs by theirs. Set the output's voltage limits (`outputs.<o>.limits`) for the device it drives (a laser's current input, a piezo driver): the FPGA clamps every sample to them after every sum, PID term and scan. `protect(o)` locks them until the FPGA is reloaded.
- **A loop with the wrong sign runs to its limit.** Start with small gains and narrow limits; the scans and acquisitions are off until you choose a mode.
- **Loading feedback leaves everything quiet**: outputs off at 0 in converter units, all loops, scans, slow outputs and the stream off, status pins low. After a reboot the service restores the last configuration, *including enabled loops*: send `SAFE` first if the setup changed.
- **Load.** Amplitudes and offsets are set into the load the board was calibrated with (50 Ω at the inputs).
- **Extension connectors.** DIO0_P (TTL in), DIO3_P (trigger in), DIO1_P (sync out) and DIO0_N–DIO3_N (status) are 3.3 V LVCMOS, not 5 V tolerant. AO0–AO3 are 1.8 V PWM at 488 kHz behind first-order RC filters that leave a large ripple: **put an external low-pass between an AO pin and any actuator** (section 10). AI0–AI3 take 0 to 7 V, never more.

- **Safe state.** `SAFE` switches both outputs off (0 V or the nearest protected limit), stops every PID, fast PID, scan and the stream, sets the slow outputs to 0 V, the sync output low and the TTL reference off.

3 Getting started

3.1 The board and the service

The board is set up as described in `common/SETUP.md` and reachable on the network (here at 192.168.0.149). Its control service loads the last active design at boot and answers commands on TCP port 5025. Switch it to feedback with

```
python3 -m rpfpga.cli --host 192.168.0.149 --load feedback    # from a PC (common/python on
    ↪ PYTHONPATH)
echo "LOAD feedback" | nc 192.168.0.149 5025                  # or with netcat
```

3.2 A first loop

Cable OUT1 to IN1 with a 50 Ω terminator at IN1, LV jumpers. The plant is then a cable: a gain of about 1 and the converters' delay. Hold IN1 at 0.1 V with fast PID 1 acting on OUT1:

```
feedback.apply({"outputs": {"1": {"enabled": True, "limits": [-0.5, 0.5]}},
                "fast": {"1": {"enabled": True, "input": "in1", "setpoint": 0.1, "i": 1e6,
                               "output": 1, "limits": [-0.5, 0.5]}}})
print(feedback.readings()["fast"]["1"])            # error and output (V, means over a frame)
print(feedback.status()["fast"]["1"]["latency"])   # input to output connector (s)
```

With $i = 10^6\,\mathrm{s}^{-1}$ the loop crosses over near $i/2\pi = 160\,\mathrm{kHz}$, safely below the limit the 232 ns delay sets (section 5.2). Then let the loop choose its own gains from its measured plant:

```
t = analysis.tune_fast(feedback, 1)
print(t["p"], t["i"], t["crossover"], t["phase_margin"])          # the design
print(t["measured"]["crossover"], t["measured"]["phase_margin"])  # the closed loop, measured
```

On the board this gives a crossover of 385 kHz with 63.0° of phase margin, and the closed loop measures 389 kHz and 62.8° (figure 3, section 7).

The same from a PC:

```
import sys; sys.path[:0] = ["common/python", "feedback/control"]   # from the repository's root
from feedback import FeedbackClient
with FeedbackClient("192.168.0.149") as fb:
    fb.set_output(1, enabled=True, limits=[-0.5, 0.5])
    fb.set_fast(1, enabled=True, input="in1", setpoint=0.1, i=1e6, output=1)
    print(fb.readings()["fast"]["1"])
```

and over TCP:

```
$ nc 192.168.0.149 5025
OUT 1 {"enabled": true, "limits": [-0.5, 0.5]}
FAST 1 {"enabled": true, "input": "in1", "setpoint": 0.1, "i": 1e6, "output": 1}
READ?
SAFE
```

3.3 The control panel

On the board, the notebook `feedback/feedback.ipynb` (at `http://192.168.0.149:9090`, password `xilinx`) opens a control panel with five tabs: **Settings** (oscillators, demodulators, outputs, PIDs, the stream), **Feedback** (fast PIDs, acquisitions and scans, slow outputs, status pins), **Meters** (the readings, live), **Presets** and **Calibration**; edits are applied together with **Apply**.

3.4 Presets

name	settings
`off`	everything off, 0 V (the power-up state)
`loopback`	OUT1: 10 kHz, 0.2 V; demodulator 1 on IN1 at 10 kHz, $\tau = 10$ ms, 24 dB/oct
`harmonics`	OUT1: 10 kHz, 0.2 V; demodulators 1 and 2 on IN1 at harmonics 1 and 2
`pll`	PID 1 locks oscillator 2 to a signal near 10 kHz on IN2 (100 Hz bandwidth, ±500 Hz, held below 10 mV); OUT2 repeats oscillator 2 at 0.1 V

4 The lock-in core

feedback's oscillators, outputs, inputs and demodulators are those of the `lockin` design, cut to two oscillators and two demodulators; the lockin user's manual describes them and their measurements in detail. In brief:

section	settings
`oscillators.1`, `.2`	`frequency` (0 to 62.5 MHz, 0.44 μHz steps), `phase` (degrees); `phase_mode` `continuous` or `coherent` (relative to `t0`)
`outputs.1`, `.2`	`enabled`, `amplitudes` (V peak per oscillator), `offset`, `dither`, `limits` (V, every sample), `monitor` (a reading as a voltage); the read-only `protect`
`inputs.in1`, `.in2`	`offset` (V), `ac` (digital AC coupling) with `ac_corner` (Hz)
`demodulators.1`, `.2`	`input` (`in1`, `in2`, `diff`), `oscillator`, `harmonic` (0–1023), `sideband`, `phase`, `reference` (`sine`, `square`), `time_constant` (0.5 μs to 10^6 s), `order` (0–8)
`stream`	`rate` (rows/s), `demodulators`
`units`	`rms` or `peak`

Each output is the sum of the oscillators' sines, its offset, its monitor, the fast PIDs and scans aimed at it, and the slow PIDs' amplitude and offset targets, clamped to the output's limits on every sample. A sine $V\sin(2\pi ft + \theta)$ at a demodulator's input reads $X = V\cos\theta$, $Y = V\sin\theta$ (V rms by default), θ referred to the OUT1 connector. Readings (`readings()`) are of one frame; `status()` reports each demodulator's frequency, ENBW and settling time. Averages, auto-phase, sweeps with resonance fits, spectra of the inputs, the capture and the counter work as in lockin; the zoom spectrum, noise density, auto time constant and SR86x port are lockin's only.

feedback's oscillators run 11 samples ahead of the signal path internally, so that a demodulator's product of an input sample is ready for a fast PID four samples after the raw sample; nothing of this is visible in the settings or readings, except that an apply needs a lead of 13 samples.

5 Fast PIDs

Two PID controllers run on every sample. Each takes one input, filters it with up to two low-pass sections, compares it with its setpoint and adds its output to one output's sum on the same sample index, 14 samples after the input sample.

5.1 Settings

key	default	meaning
`fast.f.enabled`	false	off: output 0, state cleared
`fast.f.input`	`in1`	`in1`, `in2`, `diff` (IN1 − IN2), or `product`: a demodulator's mixer product
`fast.f.demodulator`, `quadrature`	1, `x`	the product's demodulator and part: `x` (in phase) or `y` (quadrature)
`fast.f.filter`	[0, 0]	Hz: the two low-pass sections' corners (0: none; realized in octaves from 13.8 MHz to 607 Hz)
`fast.f.setpoint`	0	V at the input (raw inputs: absolute, with the input's offset setting; a product: the in-phase amplitude, in `units`); error = setpoint − input
`fast.f.p`, `i`, `d`	0	V/V; s^{-1}; s (output volts per input volt)
`fast.f.output`	1	the output it adds to
`fast.f.limits`	$[-1, 1]$	V: its output's range (the integrator is clamped to it); the output's own limits apply after the sum
`fast.f.hold`	false	freeze the integrator, zero P and D

`reset_fast(f)` (`FASTRESET`) zeroes the integrator. `readings()["fast"]` has each fast PID's error and output averaged over the last frame; the capture records them at full rate (`fast_error<f>`, `fast_out<f>`).

Gains. p is volts at the output per volt of error; i is in s^{-1}: on a plant of gain G an integral-only loop crosses over near $f_c = Gi/2\pi$ ($i = 10^6\,s^{-1}$: 160 kHz on a unity plant); d in seconds. The hardware's gains are 18-bit mantissas with four exponent steps: p reaches about 1000, i about $5 \times 10^8\,s^{-1}$ and d about 8 μs for an input and an output of equal scales, and the applied configuration reports the realized values. A gain beyond the range is refused.

Limits and anti-windup. The PID's output is clamped to its `limits`, and so is its integrator before each addition: a loop that saturates does not wind up and recovers as soon as the error changes sign. The output's own limits act after everything is summed.

5.2 Latency and bandwidth

A raw input sample reaches the output 14 samples (112 ns) later inside the FPGA; with the converters, about 232 ns from the input connector to the output connector (a product: 264 ns). `status()["fast"][f]["latency"]` reports it from the calibration. The delay limits the loop: on a plant that is otherwise flat, the largest crossover with 60° of phase margin and 10 dB of gain margin is about 340 kHz with I alone and 390 kHz with PI. A plant with its own lag (an actuator's bandwidth, a resonance, a long cable at 5 ns per metre) lowers it; `tune_fast` measures and designs for the actual plant (section 7). Below its crossover the loop divides disturbances by $|1 + L|$; just above it, it amplifies them by up to a few decibels (figure 3, right).

5.3 The low-pass sections

Two first-order sections on the input limit the error's noise bandwidth and remove a product's $2f$ term. Their corners are $-\ln(1 - 2^{-k}) f_s/2\pi$, $k = 1 \ldots 15$: 13.8 MHz, 5.72 MHz, …, 607 Hz, about an octave apart (table 1). Each costs $\arctan(f_c/f_{\text{corner}})$ of phase at the crossover f_c: keep the corners several times above the crossover.

5.4 Demodulated error signals

With `input product` the error is a demodulator's mixer product at the full rate, the input times the reference, before the demodulator's own filters: the in-phase (`x`) or quadrature (`y`) part. This is the error signal of the Pound–Drever–Hall technique and of dither locks: phase-modulate the light

k	1	2	3	4	5	6	7	8
corner	13.8 MHz	5.72 MHz	2.66 MHz	1.28 MHz	632 kHz	313 kHz	156 kHz	77.9 kHz
k	9	10	11	12	13	14	15	
corner	38.9 kHz	19.4 kHz	9.72 kHz	4.86 kHz	2.43 kHz	1.21 kHz	607 Hz	

Table 1: The fast PIDs' low-pass sections: the corner of $\alpha = 2^{-k}$ (computed). A requested corner is rounded to the nearest.

(or dither the actuator) from an output at the oscillator's frequency, detect at that frequency, and lock the in-phase part to zero. The product's DC part is the in-phase amplitude, scaled like the demodulator's X; its $2f$ part must be removed by the sections. With a carrier of 10 MHz and both sections near 600 kHz the $2f$ ripple is about 10^{-3} of the signal, at a cost of about 18° at a 100 kHz crossover; lower carriers cost more (the theory manual has the trade-off). Set the demodulator's `phase` for the steepest error slope: scan, capture `fast_error1`, adjust.

5.5 Two actuators

A fast PID has the range of its output, about a volt; slow drifts are larger. The slow PIDs cannot read a fast PID's output, so split the controller: let the fast PID give the proportional action on the fast actuator (OUT1), with little or no integral gain, and let a slow PID integrate the same error onto a slow actuator with a large range (an output's offset, or a slow analog output to a piezo amplifier or a temperature controller). The slow PID's input is the error's slow version: the input's frame mean (`in1`) for a raw input, demodulator X for a product. At low frequencies the slow integrator dominates and takes the large excursions; above, the fast actuator acts alone around the middle of its range. Gate the slow PID by the acquisition (`gate`) so that it acts only while the fast loop is locked, keep the hand-over frequency well below the fast loop's crossover, and check that both actuators push the error the same way. (A fast PID with a large integral gain would hold the error at zero by itself and leave the slow loop nothing to see.)

6 Lock acquisition

An error signal is linear only near its lock point: a loop closed far from the resonance runs its integrator to a limit. Each fast PID has an acquisition unit that finds the lock point first: it scans the PID's output with a triangle until a *source* signal shows the resonance, then freezes the scan there and lets the fast PID take over, and scans again if the lock is lost.

6.1 Settings

key	default	meaning
`acquire.a.mode`	`off`	`off`; `scan` (scan, fast PID a held at 0); `lock` (scan frozen, fast PID a running); `auto` (scan until the source engages, lock, scan again after a loss)
`acquire.a.source`	R of 1	as a slow PID's input: X, Y, R, θ, $d\theta/dt$ of a demodulator, `in1`/`in2` (frame means), `ai0`–`ai3`, `ttl`
`acquire.a.engage`	0, above	`level` and `direction` (`above` or `below`)
`acquire.a.loss`	0, 0	`level` and `time` (s): lost when the source stays on the far side of the level that long (2^w frames)
`acquire.a.scan`	0, 0, 10	`low`, `high` (V added to fast PID a's output), `frequency` (triangles per second)

`readings()["acquire"][a]` gives the `state` (`off`, `scan`, `lock`), the engagements counted since the FPGA was loaded (`relocks`, modulo 2^{16}) and the time spent in the present lock (`locked_s`). `set_mode(a, mode)` (`MODE <a> AUTO`) switches the mode alone.

6.2 How it behaves

- **Scanning**, the fast PID is held with its integrator reset: the output is the scan.
- **Engaging.** The source is compared with the levels once per frame. In the first frame where it passes the engage level, the unit locks: the scan freezes where it is and the fast PID starts from zero around that point (no jump). The decision acts about 4 μs after the frame.
- **Locked**, the fast PID holds the error; the unit watches the source.
- **Losing.** When the source has stayed beyond the loss level for the loss time, the unit scans again from where the scan froze, in its direction, after resetting the PID's integrator. The engage and loss levels make a hysteresis; the loss time ignores brief dips that the loop rides through.

Choosing the levels and the scan. Engage on a signal that shows the resonance unambiguously (a transmission, a demodulated amplitude R), not on the error signal itself, which also crosses zero far from the resonance. The scan must not run away from the resonance before the lock acts: in the up to 5.4 μs between the frame where the source passes the level and the lock, the scan moves $2(\text{high} - \text{low}) f_{\text{scan}} \times 5.4\,\mu\text{s}$, which should stay within the error signal's linear range. For a resonance whose error is linear over ±4 mV of scan voltage and a 1 V scan span, keep f_{scan} below about 370 Hz. A side-of-fringe loop holds only one side of a resonance: engaged on the other, it pushes away, loses the lock and engages on the right side within one more scan period (counted in `relocks`).

Figure 2 shows the unit cycling on the board: OUT1 scans, the source is IN2, which follows a slow sine from OUT2 through the engage and loss levels, and while locked fast PID 1 holds IN1 at 0.05 V.

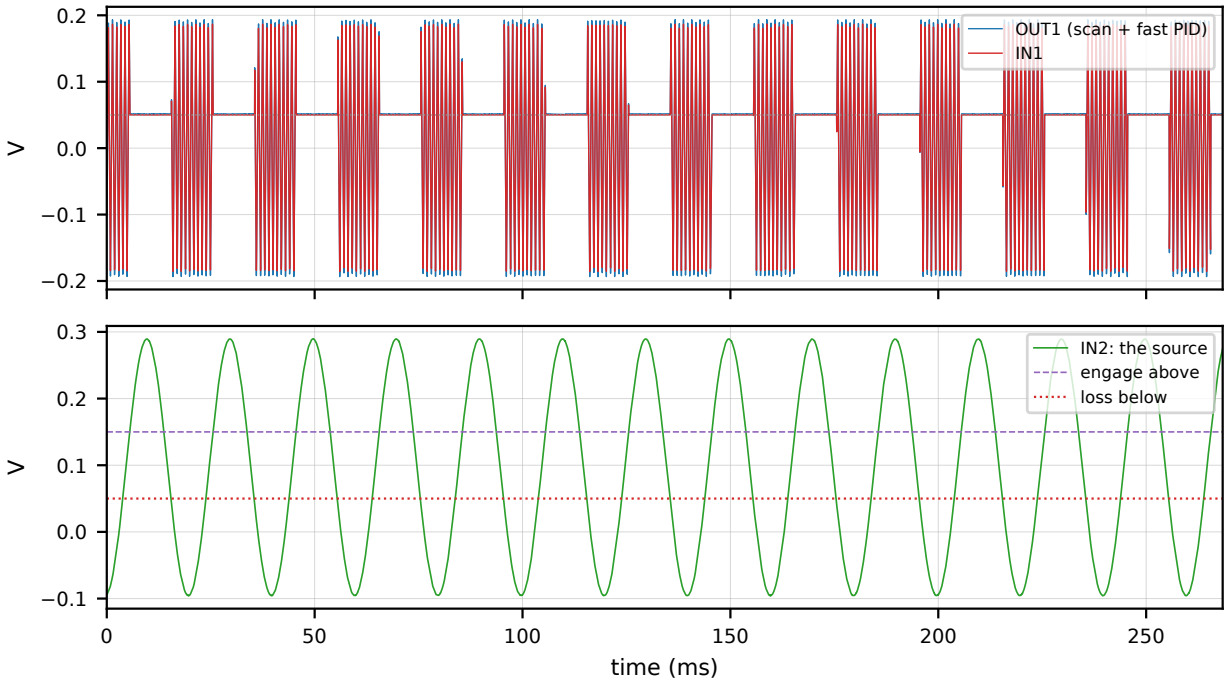


Figure 2: Lock acquisition in mode `auto` on the board (measured, 2026-10-27). OUT1 scans ±0.2 V at 1 kHz; the source IN2 follows a 50 Hz sine of 0.2 V around 0.1 V from OUT2: above 0.15 V the unit engages and fast PID 1 holds IN1 at 0.05 V; below 0.05 V for 2^7 frames it scans again. 14 engagements in the 268 ms shown; while locked IN1 stays within 110 μV rms of its setpoint.

6.3 Gating slow PIDs and status pins

A slow PID with `gate` 1 or 2 is held unless that acquisition is locked: a slow loop that only makes sense while the fast one holds (the two-actuator split of section 5.5, a slow loop on the locked cavity's transmission). The status pins can show an acquisition's state to other equipment: `{"dio": {"0": "locked1", "1": "rising1"}}` puts "acquisition 1 locked" on DIO0_N and the scan's direction (a trigger at its turning points for an oscilloscope) on DIO1_N.

7 Measuring and tuning loops

Every loop's gains are only as good as what is known about its plant. feedback measures its own plants and designs the gains for the margins you ask; the theory manual's chapter on tuning derives the method.

7.1 Fast PIDs

`tune_fast(device, f)` (`TUNE {"kind": "fast", "unit": f}`):

1. **measures the plant** with the PID held: a free oscillator puts a 10 mV sine on the PID's output at 24 frequencies from 20 kHz to 8 MHz; captures of the output, the error and the PID's output give the plant $P = -E/\mathrm{OUT}$ at each (the converters, the pipeline, the outside plant and the PID's input sections);
2. **designs** the PI gains with the largest integral gain whose loop keeps every phase margin $\geq$ `margin` (60°), every gain margin $\geq$ `gain_margin` (10 dB) and the crossover inside the measured range (and below `bandwidth` if given; `kind i`: integral only);
3. **applies** them bumplessly (the integrator kept);
4. **verifies** (`verify`): measures the running loop's gain $L = -U/\mathrm{OUT}$ at the same frequencies, and its margins.

Pass `frequencies` for another range (a plant with a resonance above 8 MHz), `amplitude` for a larger or smaller probe. `fast_response(device, f)` returns the measured plant alone, and with `closed=True` the loop gain of a running loop: $|1/(1+L)|$ is then the loop's suppression of disturbances at each frequency.

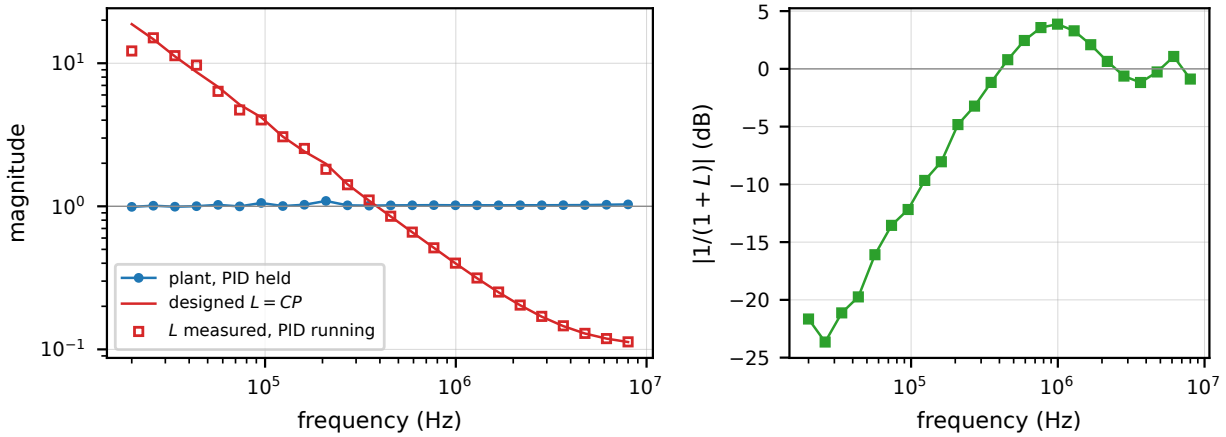


Figure 3: `tune_fast` on the board, OUT1 → IN1 (measured, 2026-10-27). Left: the plant measured with the PID held, the designed loop $L = CP$ (line) and the loop measured with the PID running (squares): crossover 385 kHz designed, 389 kHz measured. Right: the measured suppression $|1/(1 + L)|$: −22 dB at 20 kHz.

7.2 Slow PIDs

`tune_pid(device, p)` (`TUNE {"kind": "pid", "unit": p}`) steps the PID's target setting (an oscillator's frequency by 10 Hz or phase by 1°, an amplitude or offset by 10 mV, a slow output by 20 mV) four times with the PID held, captures its input per frame, and takes the plant from the averaged step response. The PI design is the fast one's with the slow PID's form (its output acting four frames after its error). With `verify`, four setpoint steps of the closed loop are captured and compared with the error predicted from the measured plant (figure 4).

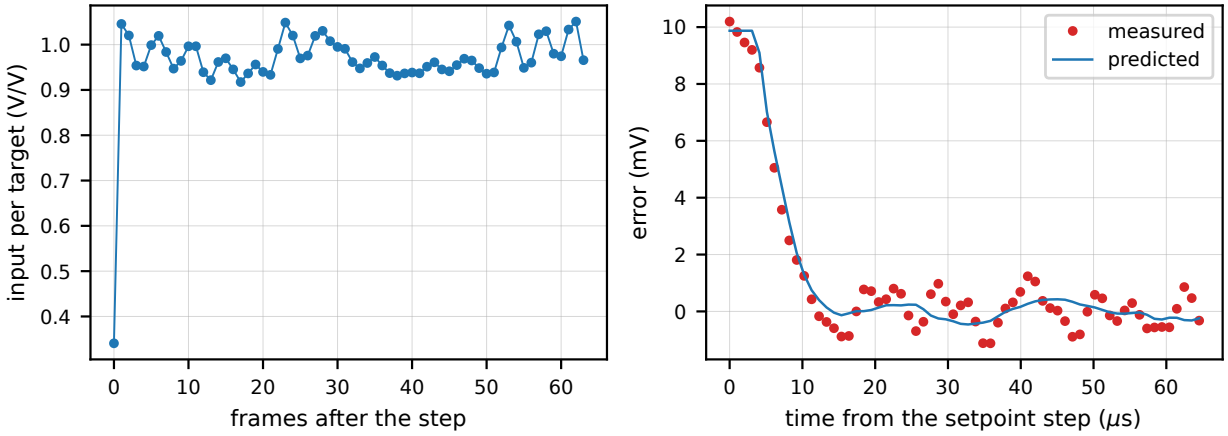


Figure 4: `tune_pid` on the board: PID 1 holds IN1's frame mean by OUT1's offset (measured, 2026-10-27). Left: the plant's unit step response (settled gain 0.987). Right: the closed loop's error after setpoint steps, measured (four averaged) and predicted from the step response: crossover 21.5 kHz, phase margin 60.8°; they differ by 5.8 % rms of the step.

7.3 The static methods

`method static` needs no frequency response: two steps of the output (fast) or target (slow) with the PID held give the plant's static gain G, and an integral-only loop $i = 2\pi\,\mathrm{bw}/G$ is set. For a fast PID the bandwidth is what the computed delay allows with the phase margin $((90° - \mathrm{margin})/(360°\,T)$, at most a third of an input section's corner): 359 kHz on the board. For a slow PID it is a quarter of the demodulator's bandwidth, 10 kHz on an input's mean, 1 kHz on a slow input or the TTL reference. Static tuning assumes the plant is flat up to the crossover; a plant with its own lag needs the measured methods.

Before tuning

Set the output's limits and the PID's limits for the actuator first: the measurements move the output by the probe's amplitude (fast: 10 mV) or the step (slow) around its present value. Tune with the loop at its operating point (the cavity near resonance, the laser at its current): the plant's gain there is what matters.

8 Slow PIDs and phase-locked loops

Four PID controllers run once per frame (976.5625 kHz) and act four frames later. They read a lock-in reading, an input's mean, a slow input or the TTL reference, and drive an oscillator's frequency or phase, an output's amplitude or offset, or a slow output.

8.1 Settings

key	default	meaning
`pids.p.enabled`	false	off: output 0, state cleared
`pids.p.input`	X of 1	`demodulator` with `quantity` `x`, `y`, `r` (V), `phase` (degrees, wrapped error), `frequency` ($d\theta/dt$, Hz), `phase_unwrapped`; or `in1`, `in2` (V: the input's mean over a frame), `ai0`–`ai3` (V: a slow input), `ttl` (degrees: the TTL reference; wrapped error; held while the reference is lost)
`pids.p.setpoint`	0	the input's units; error = setpoint − input
`pids.p.target`	none	`kind` `frequency` or `phase` of `oscillator`; `amplitude` of `oscillator` on `output`; `offset` of `output`; `ao`: slow output `channel` (V, added to its offset)
`pids.p.p`, `i`, `d`, `d_filter`	0	target units per input unit; per second; seconds; the D term's low-pass (s)
`pids.p.limits`	$[-1, 1]$	the output's range in target units (anti-windup)
`pids.p.hold`, `hold_below`	false, none	freeze; hold while the demodulator's R is below a level
`pids.p.gate`	none	1 or 2: held unless that acquisition is locked
`pids.p.lock_tolerance`, `lock_time`	0, 1 ms	locked while $\lvert e\rvert \leq$ tolerance for the time

The applied configuration reports the realized gains; a gain beyond the hardware's range is refused with the limit in your units. `reset_pid(p)` clears the integrator. The PIDs with the same target add. `tune_pid` chooses the gains from the measured plant (section 7).

8.2 Typical slow loops

Amplitude stabilization input R, target an output's amplitude.

Offset following input X of a dispersive signal (zero at the resonance), target an output's offset: a side-of-fringe or Pound–Drever–Hall lock at the frame rate, when the fast PIDs are busy or not needed.

Slow actuators input a slow input (a temperature sensor's voltage on AI0) or an input's mean, target a slow output (a heater driver on AO0): section 10.

The slow half of a two-actuator lock input the error's frame mean, target a slow output, gated by the acquisition (section 5.5).

8.3 PLLs on an input

`lock_pll(device, pid, bandwidth, oscillator, demodulator)` makes a PID lock an oscillator to the signal on its demodulator's input, as in lockin: a type-2 loop crossing over at the bandwidth (up to 19 kHz), the demodulator's filter set to order 2 at eight bandwidths, limits of ±5 bandwidths around the set frequency (inside the pull-in range of 7 bandwidths, 3 at 10 kHz); `seed` sets the oscillator from the frequency counter first, `acquire` locks the frequency first (for signals the counter cannot measure), `hold_below` holds the frequency while the signal is missing. The lockin user's manual and the theory manual's PLL chapter have the details.

```
feedback.apply({"oscillators": {"1": {"frequency": 976562.5}, "2": {"frequency": 976600.0}},
                "outputs": {"2": {"enabled": True, "amplitudes": {"1": 0.2}}},
                "demodulators": {"2": {"input": "in2", "oscillator": 2}}})
analysis.lock_pll(feedback, pid=1, bandwidth=1000.0, oscillator=2, demodulator=2)
```

9 The TTL reference, the sync output and the status pins

The extension connector E1 connects feedback to other equipment's digital signals: a TTL reference input timed to 1 ns, a sync output with 2 ns edges, a capture trigger timed to 1 ns, and four status outputs. All are 3.3 V LVCMOS; the inputs are not 5 V tolerant.

pin (E1)	role	
DIO0_P (pin 3)	TTL reference in	a lab clock's or an experiment's edges, timed to 1 ns
DIO1_P (pin 5)	sync out	an oscillator's square wave, 2 ns edges
DIO3_P (pin 9)	capture trigger in	timed to 1 ns
DIO0_N–DIO3_N (pins 4, 6, 8, 10)	status out	locked, scanning, a scan's direction, a PID locked, frame starts, clipping
ground (pin 26)		

9.1 The TTL reference

key	default	meaning
`ttl.enabled`	false	off: its state cleared, lost
`ttl.oscillator`	1	the oscillator whose phase it measures at the taken edges
`ttl.divide`	1	M: every M-th rising edge is taken (1 to 65536)
`ttl.average`	1	edges averaged per reading (a power of 2 up to 2^{15})
`ttl.loss_time`	1 ms	lost after this long without a taken edge (up to 1.07 s)

At every taken edge the reference computes the oscillator's phase at that instant, to 1 ns, and averages 2^q of them. `readings()["ttl"]` has the last frame's value (`phase`: degrees, referred to OUT1: the oscillator's phase at the OUT1 connector when the edge reaches DIO0_P), whether the reference is `lost`, and the `edges` taken. A PID or an acquisition reads it as the source `ttl`.

The sign of the TTL phase

The TTL reference reports the *oscillator's* phase at the edges. An edge that comes when the oscillator's sine at OUT1 crosses zero upward reads 0°; an edge a quarter period earlier (leading the oscillator by 90°) reads −90°. This is the opposite sign to a demodulator's θ, which is the input's phase against the reference. The sync output's `phase` follows the demodulator's convention (a positive phase advances the square wave), so the TTL reference reads minus the sync's phase (figure 6).

Locking to a lab clock. A PID from `ttl` to the oscillator's frequency is a PLL to the external clock; `lock_ttl` sets it up, seeding the oscillator from the frequency counter on DIO0_P (its edges also timed to 1 ns) at N/M times the clock:

```
analysis.lock_ttl(feedback, pid=1, oscillator=2, bandwidth=1e3, multiply=3, divide=2, average=16,
    ↪ seed=False)
feedback.apply({"outputs": {"2": {"enabled": True, "amplitudes": {"2": 0.2}}}})
```

Oscillator 2 then runs at 3/2 of the clock, phase-locked to it, and OUT2 repeats it as a sine; demodulators referenced to oscillator 2 measure against the external clock. While the clock is missing the PID holds the frequency. Figure 5 shows a 1 kHz loop locked to the sync output's 5 MHz looped back.

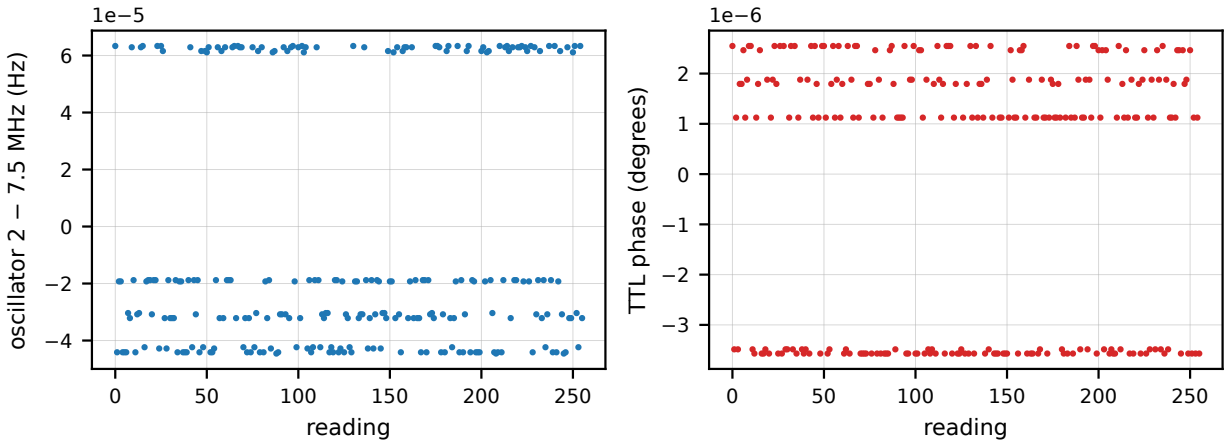


Figure 5: Oscillator 2 locked through the TTL reference to the sync of oscillator 1 (5 MHz, looped DIO1_P → DIO0_P) at 3/2 of its frequency (1 kHz bandwidth, averages of 16 edges; measured, 2026-10-27): 256 readings of oscillator 2's frequency against 7.5 MHz (mean −4 μHz, rms 4.39e-05 Hz) and of the TTL phase (rms 2.62e-06°). The sync and the TTL input share the board's clock, so the loop sees no jitter: what remains is the readings' resolution.

Resolution. One edge is timed to within a nanosecond (289 ps rms). A clock whose period is a whole number of nanoseconds (10 MHz) puts every edge at the same place within the nanosecond: averaging does not reduce that error (up to ±0.5 ns, constant). Other frequencies, or jitter, spread the edges and averaging helps (the theory manual's figure on the pins). Edges near a sampling instant toggle by 1 ns between readings.

9.2 The sync output

`{"sync": {"enabled": True, "oscillator": 2, "phase": 0.0}}` puts on DIO1_P a square wave that is high while the oscillator's phase at OUT1 plus `phase` is in its first half cycle: with phase 0 it rises at OUT1's upward zero crossing (to about a nanosecond, through the board calibration's pin delays). Its edges are placed in 2 ns slots: a frequency whose period is a whole number of 2 ns slots has steady edges; others walk across a slot (0.58 ns rms). Use it to trigger an oscilloscope or another instrument on the modulation, or as the clock of another instrument's PLL. Figure 6 checks the sync's phase setting against the TTL reference.

9.3 The capture trigger

A capture with trigger `rising` or `falling` starts at an edge on DIO3_P; the edge's position within its sample is reported (`trigger_subsample`, eighths of a sample) and the capture's times `t` start from it, so that captures of a repetitive experiment align to a nanosecond.

9.4 Status pins

`{"dio": {"0": "locked1"}}` (`DIO 0 locked1`) shows one state per pin: `low`, `high`, `locked1`, `locked2`, `scanning1`, `scanning2`, `rising1`, `rising2` (a scan's direction), `pid1`–`pid4` (a PID locked), `frame` (a frame's first sample), `out_clip`, `in_clip`. They are levels for other equipment, an LED or an oscilloscope, not timed to the sample.

9.5 Calibration of the pins

Two board-calibration fields refer the pins to OUT1: `pins_delay_s`, an output pin against OUT1 (`CAL PINS`), and `pins_loop_s`, an output pin into a 1 ns input (measured with the jumper DIO0_N

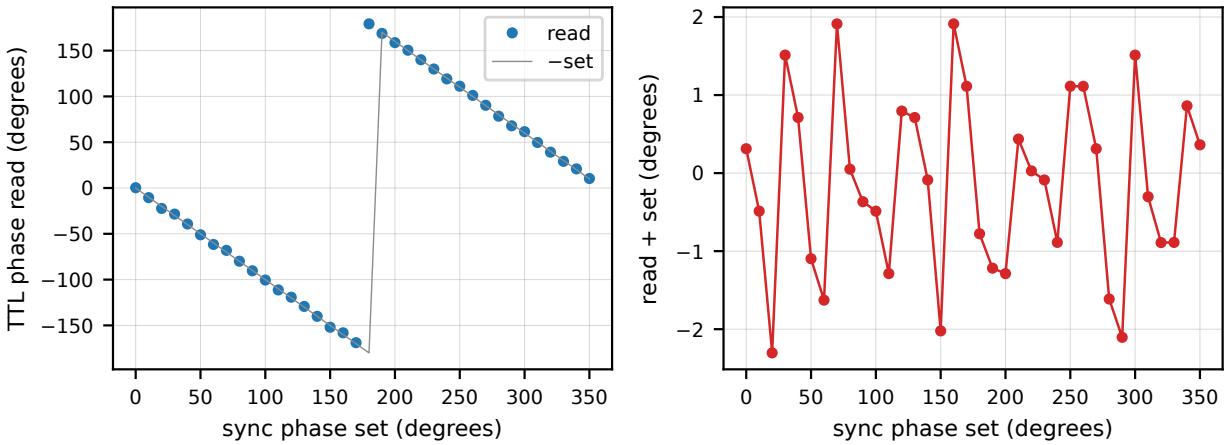


Figure 6: The sync output of oscillator 1 at 5 MHz looped into the TTL reference (averages of 1024 edges, 16 readings each), both referred to OUT1 (measured, 2026-10-27). Left: the TTL phase read against the sync's phase set: the reference reads the oscillator's phase at the edges, which is minus the sync's phase. Right: read + set: −0.14° on average, 4.22° peak to peak over the settings: the 1 ns and 2 ns grids of the two pins (1.8° per nanosecond at 5 MHz).

→ DIO3_P: `measure_pins_loop(store=True)`, `PINSLOOP? STORE`). Without them the TTL and sync phases are off by the pins' delays (a few tens of nanoseconds).

10 Slow analog outputs and inputs

The extension connector E2 carries four slow analog outputs AO0–AO3 and four slow analog inputs AI0–AI3. They suit slow actuators and sensors: a piezo amplifier's or a current controller's input, a heater driver, a temperature sensor, a photodiode's DC level.

outputs AO0–AO3 (E2 pins 17–20)	PWM at 488.28125 kHz behind the board's RC filters; the duty from a second-order delta-sigma of a 16-bit value, so that the mean is exact; 0 to about 1.8 V (nominal until calibrated); 0 switches an output off
inputs AI0–AI3 (E2 pins 13–16)	the XADC's auxiliary inputs behind the board's dividers: 12 bits, 0 to about 7.0 V (nominal), a new value about every 14 μs

`{"ao": {"0": {"offset": 0.9}}}` (`AO 0 0.9`) sets AO0; a slow PID with target `ao` adds to it. `readings()["ao"]` and `["ai"]` (`AI?`) report the levels in volts.

Filter the slow outputs

The PWM's ripple at 488 kHz is large behind the board's own filters: the slow inputs read 0.14 to 0.17 V peak to peak at half scale through AO → AI, and at the pin it may be larger (a 160 kHz first-order filter would leave about 0.85 V). Put an external low-pass between an AO pin and any actuator: a first-order RC of corner f_c divides the ripple by about 488 kHz/f_c (10 kΩ and 160 nF, 100 Hz: well under 1 mV), followed by a buffer unless the load has a high impedance. Keep a slow PID's bandwidth on that output well below f_c: the filter's delay is in the loop.

Figure 7 shows AO0 and AO1 read back by AI0 and AI1 through jumpers, with the board's filters only.

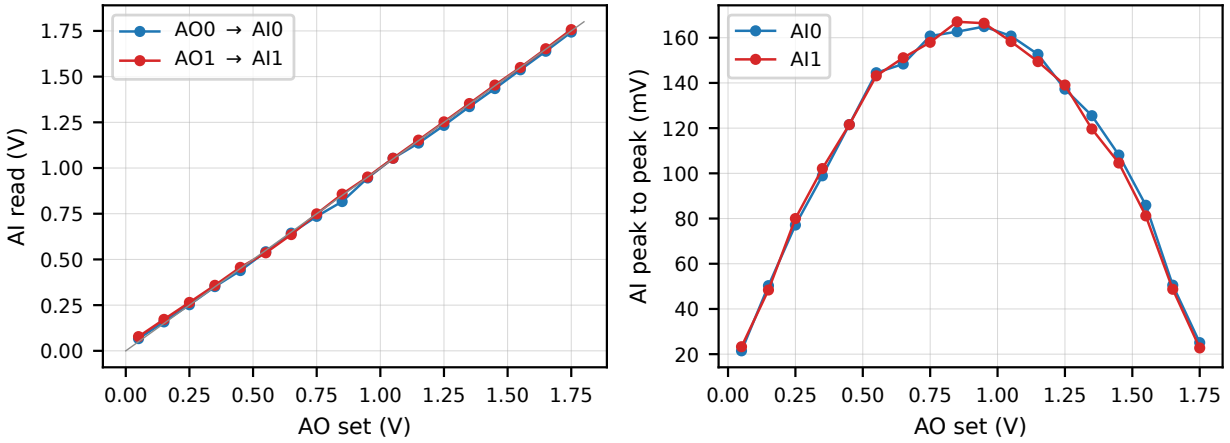


Figure 7: AO0 → AI0 and AO1 → AI1 at nominal calibrations (measured, 2026-10-27): the mean of 32 readings against the setting (left; slopes 0.989 and 0.993) and their peak-to-peak spread (right), dominated by the PWM ripple that the XADC samples: largest at half scale.

A slow loop. A heater driven from AO0 holding a temperature sensor's voltage on AI0 at 1.2 V with PID 4:

```
feedback.apply({"ao": {"0": {"offset": 0.9}},
                "pids": {"4": {"enabled": True, "input": {"quantity": "ai0"}, "setpoint": 1.2,
                               "target": {"kind": "ao", "channel": 0}, "p": 0.5, "i": 0.1,
                               "limits": [-0.9, 0.9]}}})
print(feedback.readings()["ai"], feedback.readings()["ao"])
```

Calibration. Until measured the nominal full scales apply (1.8 V per 65536 units out, 7.0 V per 65536 in). With a voltmeter, store each channel's volts per unit and offset: `set_slow_calibration("ao", 0, G, B)` or `CAL SLOW AO0 G B`.

11 Software

feedback is controlled like every design of the repository: by the Python driver on the board, the Python client on a PC, TCP commands, or the command line; the notebook's panel uses the driver or the client. Appendix A lists every command.

11.1 Python on the board and on a PC

```
import rpfpga                                    # first: it prepares the environment PYNQ needs
from pynq.overlays.feedback import DRIVER, analysis
feedback = DRIVER()                              # attaches to feedback, or loads it
```

`FeedbackClient(host)` talks to the service from a PC with the standard library (put `common/python` and `feedback/control` on the path); it has the driver's methods plus `set_fast`, `reset_fast`, `set_acquisition`, `set_mode`, `set_ao`, `ao`, `ai`, `set_dio`, `set_ttl`, `lock_ttl`, `set_sync`, `tune_fast`, `tune_pid`, `fast_response`, `pid_step_response`, and the lock-in core's (`set_frequency`, `set_output`, `set_demodulator`, `set_pid`, `lock_pll`, `average`, `auto_phase`, `sweep`, `spectrum`, `record`, ...); `DataStream(host)` reads the data stream. One request at a time per connection; a long measurement (a tuning, a sweep, a capture) is cancelled by another client's edit; reloading the FPGA makes old handles stale.

11.2 TCP and the command line

```
FAST 1 {"enabled": true, "input": "in1", "setpoint": 0.1, "i": 1e6, "output": 1}
ACQ 1 {"mode": "auto", "source": {"quantity": "in2"}, "engage": {"level": 0.15}}
MODE 1 SCAN
TUNE {"kind": "fast", "unit": 1}
TTLLOCK {"pid": 1, "oscillator": 2, "bandwidth": 1000}
SYNC {"enabled": true, "oscillator": 1}
AO 0 0.9
AI?
READ?
SAFE
```

```
python3 -m rpfpga.cli --host 192.168.0.149 --fast 1 '{"enabled": true, "input": "in1", "i": 1e6}' --
    ↪ ao 0=0.9
```

The command line's options: `-osc`, `-demod`, `-amp`, `-offset1`, `-offset2`, `-units`, `-pid P JSON`, `-fast F JSON`, `-acquire A JSON`, `-ao K=V`, `-ttl JSON`, `-sync JSON`, `-autophase D`, and the common ones (`-load`, `-preset`, `-safe`, `-range`, `-calibration`, `-json`). It prints the readings, the active loops, the acquisitions, the slow inputs and the counters.

11.3 The data stream

As lockin's, with two demodulators: `DATA STREAM [interval]` sends rows of 11 float64 fields, `frame t_s x1 y1 x2 y2 locked clipped apply flags units_peak` (appendix B).

12 Calibration

feedback uses the board calibration shared by every design (`common/CALIBRATION.md`): the converters' gains and offsets, which make every setting and reading volts at the connectors; the inputs' loop delays, which refer phases and the fast PIDs' latencies to the connectors; the pins' delays, which refer the TTL reference and the sync output to OUT1; and the slow channels' constants.

what	how
converters (DC)	`CAL OFFSETS`, `CAL HOLD` with a DMM, `CAL LOOPBACK` (OUT1 → IN1, OUT2 → IN2), with a design that measures its inputs (lockin, awg, synth, mult_add); feedback uses the result
inputs' loop delays	lockin's `measure_input_delay` (or feedback's `analysis.measure_input_delay`), stored per input and range
pins	`CAL PINS <s>` (an output pin against OUT1); `measure_pins_loop(store=True)` with the jumper DIO0_N → DIO3_P
slow outputs and inputs	`set_slow_calibration(kind, k, G, B)`, `CAL SLOW AO0 G B`, from a voltmeter
input ranges	`RANGE IN1 LV|HV`: the jumpers, which no software can read

Without a calibration the nominal constants apply: about 14 % off in amplitude on the board used here; loops on the board's own signals (a fast PID holding IN1 through OUT1) stay right, since they compare the signal with itself. Recalibrate after warm-up and after changing jumpers or cables.

13 Applications

Worked examples. `feedback` is the driver on the board; with `FeedbackClient` the same calls work. The signs of the gains depend on your plant: with the wrong sign a loop runs to a limit. Set the output's limits (and `protect` them) for the actuator before enabling a loop. These recipes follow the board's tested paths (`docs/VERIFICATION.md`); with real optics, tune with `tune_fast` at the operating point.

13.1 Pound–Drever–Hall

A phase modulator driven from OUT2 at 20 MHz, the reflected light's photodiode on IN1, the laser's fast actuator on OUT1. Demodulator 1 detects IN1 at 20 MHz; fast PID 1 locks its in-phase product to zero; acquisition 1 scans until the cavity's transmission (on IN2) shows the resonance:

```
feedback.apply({"oscillators": {"1": {"frequency": 20e6}},
                "outputs": {"2": {"enabled": True, "amplitudes": {"1": 0.5}},
                            "1": {"enabled": True, "limits": [-0.2, 0.2]}},
                "demodulators": {"1": {"input": "in1", "oscillator": 1, "phase": 0.0}},
                "fast": {"1": {"enabled": True, "input": "product", "demodulator": 1, "quadrature":
    ↪ "x",
                               "filter": [1.3e6, 1.3e6], "setpoint": 0.0, "p": 0.5, "i": 1e5,
                               "output": 1, "limits": [-0.2, 0.2]}},
                "acquire": {"1": {"mode": "auto", "source": {"quantity": "in2"},
                                  "engage": {"level": 0.3, "direction": "above"},
                                  "loss": {"level": 0.1, "time": 1e-3},
                                  "scan": {"low": -0.15, "high": 0.15, "frequency": 20.0}}}})
```

The sections at 1.3 MHz remove the product's 40 MHz term. Set demodulator 1's `phase` for the steepest error slope: mode `scan`, capture `fast_error1` with `in2`, adjust, then `auto`. The product path adds 32 ns to the loop delay.

13.2 Side of fringe with scan and relock

A cavity's transmission on IN1 (0 to 1 V on resonance), its piezo driven from OUT1. Acquisition 1 scans until the transmission passes 0.6 V upwards, then holds IN1 at 0.5 V; below 0.2 V for 1 ms it scans again:

```
feedback.apply({"outputs": {"1": {"enabled": True, "limits": [-1.0, 1.0]}},
                "fast": {"1": {"enabled": True, "input": "in1", "setpoint": 0.5, "p": 0.2, "i": 2e4,
                               "output": 1, "limits": [-0.3, 0.3]}},
                "acquire": {"1": {"mode": "auto", "source": {"quantity": "in1"},
                                  "engage": {"level": 0.6, "direction": "above"},
                                  "loss": {"level": 0.2, "time": 1e-3},
                                  "scan": {"low": -0.8, "high": 0.8, "frequency": 20.0}}},
                "dio": {"0": "locked1", "1": "rising1"}})
r = feedback.readings()["acquire"]["1"]
print(r["state"], r["relocks"], r["locked_s"])
```

A side-of-fringe loop holds one side of the resonance; engaged on the other it pushes away, loses the lock and engages on the right side within one more scan period. DIO0_N is high while locked, DIO1_N follows the scan's direction.

13.3 A laser on a cavity with two actuators

Add a slow actuator for the drifts (a piezo amplifier on AO0 behind an external RC): fast PID 1 keeps proportional action only, PID 1 integrates the error's frame mean onto AO0 while acquisition 1 is locked:

```
feedback.apply({"fast": {"1": {"p": 0.5, "i": 0.0}},
                "ao": {"0": {"offset": 0.9}},
                "pids": {"1": {"enabled": True, "input": {"quantity": "in1"}, "setpoint": 0.5,
                               "target": {"kind": "ao", "channel": 0}, "i": 20.0,
                               "limits": [-0.85, 0.85], "gate": 1}}})
```

The slow loop's integral gain sets the hand-over frequency; keep it a decade or more below the fast loop's crossover and the external RC's corner, and check the two actuators' signs agree.

13.4 A loop's suppression

`analysis.fast_response(feedback, 1, closed=True)` measures the running loop's gain L at a set of frequencies; $|1/(1+L)|$ is the factor by which the loop divides disturbances there (the `tune_fast` figure shows it for the loopback).

13.5 A PLL to a lab clock, and a clean copy

A 10 MHz reference into DIO0_P: oscillator 2 locks to it at 3/2 (15 MHz), OUT2 repeats it as a sine, the sync output repeats it as a square wave:

```
analysis.lock_ttl(feedback, pid=1, oscillator=2, bandwidth=1e3, multiply=3, divide=2, seed=False)
feedback.apply({"outputs": {"2": {"enabled": True, "amplitudes": {"2": 0.2}}},
                "sync": {"enabled": True, "oscillator": 2, "phase": 0.0}})
```

The loop's bandwidth decides what the oscillator follows: the clock's slow wander within it, while the oscillator's own (exact) phase sets what is outside it. A narrow loop cleans up a noisy clock.

13.6 A temperature on a heater

Section 10: a sensor's voltage on AI0, a heater driver on AO0, PID 4 between them; a loop of a few hertz at most, its gains from `tune_pid` (steps of 20 mV on AO0).

14 Specifications

Computed values come from the bit-exact model; measured ones from the board used here (dates in `docs/VERIFICATION.md` and the captions). The lock-in core's measured figures (accuracy, noise, stability, the analog response) are lockin's: its user's manual.

quantity	value
fast PIDs	2 at 125 MSa/s; input IN1, IN2, IN1 − IN2 or a demodulator's in-phase or quadrature mixer product; two first-order low-pass sections, corners 13.8 MHz to 607 Hz about an octave apart (or none); error 25 bits; P to 1000 V/V, I to about $5 \times 10^8\,\mathrm{s}^{-1}$, D to 8 μs (unity scales, computed); integrator clamped at the limits; hold, reset
fast PID latency	14 samples (112 ns) from a raw input, 18 (144 ns) from a product, plus the converters: about 232 ns and 264 ns connector to connector (computed); measured 235.6 ns around OUT1 → IN1 with 50 cm of cable
fast loop, measured	`tune_fast` on OUT1 → IN1: crossover 385 kHz, phase margin 63.0° designed; 389 kHz, 62.8° measured in the closed loop
lock acquisition	2; triangle scan of a fast PID's output (steps of 2^{-32} DAC step per sample: 1.8 μHz to 7.6 kHz for a 1 V span); engage at a level (above or below), loss after 2^w frames beyond a level; sources X, Y, R, θ, $d\theta/dt$, an input's mean, a slow input, the TTL reference; decided per frame, acting about 4 μs later
slow PIDs	4 at 976.5625 kHz; inputs X, Y, R, θ, $d\theta/dt$, the unwrapped phase, IN1/IN2 frame means, AI0–AI3, the TTL reference; targets an oscillator's frequency or phase, an output's amplitude or offset, AO0–AO3; gated by an acquisition (optional); PLLs up to 19 kHz bandwidth
slow outputs	AO0–AO3: PWM at 488.28125 kHz, second-order delta-sigma of a 16-bit value (mean exact); nominal 0 to 1.8 V; ripple behind the board's filters 0.14–0.17 V p-p at half scale as AI reads it (measured)
slow inputs	AI0–AI3: XADC, 12 bits, about 14 μs per update; nominal 0 to 7.0 V; AO → AI slopes 0.98–1.01 at nominal scales (measured)
TTL reference	DIO0_P: rising edges timed to 1 ns; every M-th taken (M = 1 to 65536), averages of 2^q (q = 0 to 15); the oscillator's phase at OUT1, 2^{-32} turn; lost after up to 1 s without edges (PIDs hold); measured: an oscillator locked to a 5 MHz clock (and 3/2 of it) to 10^{-5} Hz on average
sync output	DIO1_P: an oscillator's square wave at OUT1's phase plus an offset (phase 0 rises at OUT1's upward zero crossing, ±1 ns), 2 ns edges
1 ns trigger, counter	the capture's DIO3_P trigger and the counter's TTL source timed to 1 ns; counter on IN1/IN2 (Schmitt) or DIO0_P
status pins	DIO0_N–DIO3_N: low, high, locked, scanning, a scan's direction, a PID locked, frame starts, clipping
lock-in core	2 oscillators, 2 outputs, 2 demodulators, 1–8 RC sections, readings in V and degrees at the connectors: as lockin 1.0
capture	4 × 8192 words: inputs, outputs, the demodulators' inputs, the fast PIDs' errors and outputs (full rate), the frame-rate readings
after loading	everything off, outputs at 0 in converter units, status pins low

A Command reference

A.1 feedback's TCP commands (port 5025)

Replies are `OK <value>` or `ERR <message>`; a refused setting changes nothing. The lock-in core's commands are lockin's (`OSC`, `FREQ`, `DEMOD`, `TC`, `ORDER`, `HARM`, `INPUT`, `REF`, `DPHASE`, `INPUTS`, `OUT`, `AMP`, `OFFS`, `PROTECT`, `PID`, `PIDRESET`, `PLL`, `STREAM`, `UNITS`, `ZERO`, `READ?`, `X?`, `Y?`, `R?`, `THETA?`, `AVG?`, `AUTOPHASE`, `HW?`, `CLEAR`, `CAPTURE?`, `SWEEP`, `SPECTRUM?`, `DATA STREAM`; the lockin user's manual, appendix A). feedback adds:

command	reply, action
`FAST <f> <json>`, `FAST <f>?`, `FASTRESET <f>`	fast PID f (1, 2); zero its integrator
`ACQ <a> <json>`, `ACQ <a>?`	acquisition a (1, 2)
`MODE <a> OFF|SCAN|LOCK|AUTO`	acquisition a's mode
`AO <k> <V>`, `AO?`	slow output k's offset (0–3); all four
`AI?`	the slow inputs (V)
`DIO <k> <source>`, `DIO?`	status pin DIOk_N's source
`TTL <json>`, `TTL?`	the TTL reference
`TTLLOCK <json>`	`{"pid", "oscillator", "bandwidth"[, "multiply", "divide", "range", "average", "seed", "gate"]}`: `lock_ttl`
`PINSLOOP? [STORE]`	`measure_pins_loop` (jumper DIO0_N → DIO3_P); `STORE`: into the calibration
`SYNC <json>`, `SYNC?`	the sync output
`TUNE <json>`	`{"kind": "fast"|"pid", "unit"[, "method", "bandwidth", "step", "count", "margin", "gain_margin", "kind", "apply", "verify"; fast: "frequencies", "amplitude", "oscillator", "settle"; pid: "frames", "steps"]}`: `tune_fast`, `tune_pid`
`RESPONSE? <json>`	`{"kind": "fast", "unit"[, "frequencies", "amplitude", "oscillator", "closed", "settle", "periods", "count"]}`: `fast_response`; `{"kind": "pid", "unit"[, "step", "frames", "count", "rate"]}`: `pid_step_response`
`COUNT? [IN1|IN2|TTL] [<gates>]`	the frequency (`TTL`: DIO0_P's edges, timed to 1 ns), or NaN

A.2 Python

`analysis` (with the driver or the client) adds to the lock-in core's functions (`average`, `auto_phase`, `lock_pll`, `sweep`, `fit_resonance`, `spectrum`, `measure_input_delay`, `enbw`, `settle_time`): `tune_fast`, `tune_pid`, `fast_response`, `pid_step_response`, `pid_setpoint_step`, `predict_setpoint_step`, `design_pi`, `loop_margins`, `fast_controller`, `pid_controller`, `ttl_pll_settings`, `lock_ttl`, `measure_pins_loop`. The driver adds `reset_fast(f)`, `capture_step(sources, patch, rate, after)` (a capture armed inside the apply that makes a change), `ttl_delay()`, `sync_delay()`; the client the setters of section 11.

B The data stream's rows

`DATA STREAM [interval]` replies with a header line and then, every interval, `DATA {"rows": n, "lost": ..., "data": base64}`: rows of 11 little-endian float64 fields,

```
frame t_s x1 y1 x2 y2 locked clipped apply flags units_peak
```

as lockin's (with two demodulators): the frame number and its end in seconds, X and Y (NaN if not streamed), the PIDs' locked flags (bits 0–3), the frame's clip flags (1 IN1, 2 IN2, 4 an output), the apply count (modulo 256), 1 in `flags` after a gap, 1 in `units_peak` for peak units.