lockin 1.0: reference documentation

overview, recipes, theory, control reference, registers, test records

Rok Zitko, Jozef Stefan Institute

2026-11-10 (commit 6178370)

Contents

lockin: a four-demodulator digital lock-in amplifier **2**
- Before connecting equipment . . . 3
- Quick start . . . 3
- Presets . . . 4
- Specifications . . . 5
- Hardware setup . . . 6
- Documentation . . . 6
- Changing the design . . . 7
- Source layout . . . 7
- Verification . . . 8
- Author . . . 8

lockin recipes **8**
- A weak signal . . . 8
- Harmonics and intermodulation . . . 8
- Bode plot and resonance fit . . . 9
- Follow the resonance: a loop on the resonator’s phase . . . 9
- A PLL on an external signal . . . 9
- Amplitude and offset loops . . . 9
- The data stream . . . 10
- Captures and spectra . . . 10
- Noise around the reference, a time constant, stability . . . 10
- Phases at the connectors . . . 10

How lockin works **11**
- Samples, frames and applies . . . 11
- Oscillators and sines . . . 11
- The reference . . . 11
- Inputs . . . 11
- Mixers and the CIC . . . 12
- RC filters . . . 12
- R and θ . . . 12
- PIDs . . . 12
- Outputs . . . 13
- Phase reference and units . . . 13
- The frame engine’s schedule . . . 13
- Readings, FIFO, capture, counter . . . 14
- Spectra and statistics of X and Y . . . 14
- Resources . . . 15
- The design’s history and limits . . . 16

lockin: settings, Python methods, TCP commands, data stream, command line **16**
- The configuration . . . 16
- Python (driver on the board, `LockinClient` on a PC) . . . 17
- TCP commands (port 5025) . . . 19
- SR86x port (port 23) . . . 20
- The data stream . . . 21

Command line . . . 21

lockin registers **22**
Control and status (0x000–0x0FF) . . . 22
Configuration words in flip-flops (0x100–0x1C8) . . . 23
Engine words (0x400–0x7FF) . . . 23
Readings (0x800–0x8FF) . . . 24
FIFO (0x1000–0x1FFF) . . . 24
Capture (0x20000–0x3FFFF) . . . 24

lockin verification **24**
2026-10-27 (afternoon): the consolidated software on the board; the manuals . . . 24
2026-10-27: an SR86x-compatible port; spectra and statistics of X and Y . . . 25
2026-10-26 (late night): the engine's memories in block RAM (simulation, build, board) . . . 26
2026-10-25 (night): faster transactions, PLL frequency acquisition . . . 27
2026-10-25 (evening): the PLL's phase error, characterization . . . 27
2026-10-25: lockin 1.0 . . . 30

lockin: a four-demodulator digital lock-in amplifier

lockin turns the Red Pitaya into a **dual-input, four-demodulator lock-in amplifier with built-in oscillators, PID controllers and phase-locked loops**, in the style of a benchtop digital lock-in. It has

- **four oscillators** from 0 to 62.5 MHz in steps of 0.44 µHz (2^{-48} of the sample clock), with phases, switched **phase-continuously** or **coherently** (every phase relative to one zero time t0, so that returning to a frequency restores its phase);
- **two outputs**, each a **sum of the four oscillators' sines** with its own amplitudes, plus a DC offset, a **monitor** of any reading, optional dither, and **hardware voltage limits** (lockable until the FPGA is reloaded);
- **four demodulators**, each on **IN1, IN2 or IN1 − IN2**, mixing at full rate (125 MSa/s) with a **harmonic** m = 0…1023 of an oscillator, optionally a **sideband** m·f_A ± f_B with a second oscillator (intermodulation products), with a **sine or square-wave reference** and its own phase;
- **low-pass filters** of **1 to 8 RC sections** (6 to 48 dB/octave) with any time constant from 0.5 µs to 10^6 s, applied to X and Y;
- readings of **X, Y, R, θ**, the **unwrapped phase** and the **frequency deviation** dθ/dt, in **rms volts** (or peak) at the connectors, phases referred to the OUT1 connector;
- **four PID controllers** working at the filter rate (976.5625 kHz), from X, Y, R, θ or dθ/dt of any demodulator to an **oscillator's frequency or phase**, an **output's amplitude** of an oscillator, or an **output's offset**: phase-locked loops (to an external signal on an input), amplitude and offset loops, with limits, anti-windup, a filtered D term, **hold** (manual or when R drops below a level) and **lock detection**;
- **continuous data**: rows of X and Y at up to 976 562.5 rows/s (the hardware) through a FIFO, streamed by the board's service (`DATA STREAM`: 7.6 k rows/s of two demodulators, without loss in 6 of 7 runs, measured);
- a **capture** of 8192 words of four sources (inputs, outputs, the demodulators' inputs, X, Y, R, θ, dθ/dt, PID outputs and errors) at 125 MSa/s divided by 2^k, triggered immediately, by the DIO3_P pin, by an input level or by software, with pretrigger;
- a **frequency counter** on IN1 or IN2;
- software: averaged readings, **auto-phase**, **PLL set-up** (seeded by the counter), **sweeps** of frequency, amplitude, phase or offset (nested), a **resonance fit** (f0, Q), **spectra** of the inputs, and a measurement of the inputs' **loop delay** that refers phases to the connectors;
- the demodulators' output analysed: a **zoom spectrum** around the reference (X + iY's FFT, the filter divided out), the input's **noise density** at the reference, a **time constant chosen from the signal** for a given precision, the **Allan deviation** and running statistics of long recordings;
- an **SR86x-compatible port** (TCP 23): programs written for a Stanford Research SR860/SR865 (pymeasure's `SR860`, LabVIEW over a socket) drive demodulator 1, oscillator 1 and OUT1 with a subset of its commands (`docs/PROTOCOL.md`, "SR86x port").

Volts are **real volts at the SMA connectors** through the **board calibration** shared by every design, which lockin can also measure (../common/CALIBRATION.md).

lockin 1.0 is the lock-in amplifier with slow PIDs. The fast (125 MSa/s) PIDs, lock acquisition and the slow analog outputs and inputs (AO/AI) are in a second bitstream, feedback (lockin's core with two demodulators), whose 2.0 also has the TTL reference input and a sync output (`docs/THEORY.md`, "The design's history and limits").

Before connecting equipment

Read the repository's Connecting to lab equipment: read first section. For lockin in particular:

- **Loading lockin leaves the outputs quiet**: both outputs off at 0 in converter units, a few mV from 0 V; PIDs and the stream off. After a reboot, the service restores the last configuration.
- **PIDs act on the outputs.** A PID with an amplitude or offset target moves an output by up to its limits; set them for your load (`limits`, in the target's units), and set the output's **voltage limits** (`outputs.<o>.limits`, V): the FPGA clamps every sample to them, after all sums and PID terms. `protect(o)` locks them until the FPGA is reloaded; then no configuration, preset, SAFE or calibration can widen them.
- **Load.** Amplitudes are set **into the load the board was calibrated with** (the calibration's loopback: 50 Ω terminators at the inputs); into 1 MΩ you get about twice.
- **Extension-connector pin DIO3_P** (the capture trigger) is a 3.3 V LVCMOS input with a pull-down, not 5 V tolerant.
- **Safe state:** `SAFE` (Python `lockin.safe()`) switches both outputs off (0 V or the nearest protected limit) and stops every PID and the stream.

Quick start

The board is set up (../common/SETUP.md) and reachable at `192.168.0.149`. For a first measurement, cable OUT1 → IN1 with a 50 Ω terminator at IN1.

- **Jupyter** on the board: open `http://192.168.0.149:9090` (password `xilinx`), then `lockin/lockin.ipynb`, and run the cells. The panel has four tabs: **Settings** (oscillators, demodulators, outputs, PIDs, the stream; one Apply), **Meters** (the readings, live), **Presets** and **Calibration**.
- **Python on the board:**

```
import rpfpga                                   # must come first
from pynq.overlays.lockin import DRIVER, analysis
lockin = DRIVER()                               # loads lockin if the board runs something else
lockin.load_preset("loopback")                  # OUT1: 10 kHz, 0.2 V; demodulator 1 on IN1
print(lockin.readings()["demodulators"]["1"])  # x, y, r (V rms), phase (degrees), ...
lockin.apply({"demodulators": {"1": {"time_constant": 0.1, "order": 4}}})
print(analysis.auto_phase(lockin, 1))           # phase so that Y = 0
```

- **Python on a PC**, through the board's control service (standard library only), from the repository root:

```
import sys; sys.path[:0] = ["common/python", "lockin/control"]
from lockin import LockinClient, DataStream
with LockinClient("192.168.0.149") as li:        # the board must run lockin
    li.set_frequency(1, 32768.0)
    li.set_output(1, enabled=True, amplitudes={"1": 0.1})
    li.set_demodulator(1, input="in1", oscillator=1, time_constant=1e-3, order=4)
    print(li.average((1,), count=16)["1"])
    bode = li.sweep("frequency", {"oscillator": 1}, start=30e3, stop=35e3, points=101,
↪   fit=True)
    print(bode["fit"]["1"])                       # f0, Q, amplitude, phase of a resonance
    li.set_stream(1000, demodulators=(1,))
with DataStream("192.168.0.149") as s:            # rows of X, Y as the board measures them
    rows, lost = s.read()
```

  If the board runs another design, switch with `rpfpga.client.connect("192.168.0.149").load("lockin")`. Reloading invalidates existing handles; reconnect explicitly.

- **TCP** from anything: `nc 192.168.0.149 5025`, then for example `FREQ 1 10000`, `AMP 1 1 0.2`, `OUT 1 ON`, `TC 1 0.01`, `R? 1`, `READ?`, `AUTOPHASE 1`, `PLL 1 {"bandwidth": 100, "oscillator": 2, "demodulator": 2, "seed": "in2"}`, `SWEEP {"parameter": "frequency", "target": {"oscillator": 1}, "start": 1e3, "stop": 1e6, "points": 31, "scale": "log"}`, `PRESET harmonics`, `SAFE`.
- **Command line:** `python3 -m rpfpga.cli --host 192.168.0.149 --osc 1=10e3 --amp 1=1,0.2 --demod 1=in1,0.01,4` (with `common/python` and `lockin/control` on `PYTHONPATH`).

docs/PROTOCOL.md lists every setting, method, TCP command and command-line option. RECIPES.md has worked examples.

Presets

Name	Settings
`off`	everything off, 0 V (the default)
`loopback`	OUT1: 10 kHz, 0.2 V; demodulator 1 on IN1 at 10 kHz, τ = 10 ms, 24 dB/oct (OUT1 → IN1: R = 0.141 V rms)
`harmonics`	OUT1: 10 kHz, 0.2 V; demodulators 1–4 on IN1 at harmonics 1–4
`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

Specifications

Computed figures come from the bit-exact model; measured ones from the board used here (loopback OUT1 → IN1, OUT2 → IN2 with 50 Ω terminators, LV jumpers; 2026-10-25). Board-wide figures are in the root README.

Quantity	Value
Oscillators	4; 0 … 62.5 MHz in steps of 0.44 µHz; phase in steps of 2^{-32} cycle; continuous or coherent (relative to t0) switching on a frame boundary
Outputs	2: Σ of the 4 oscillators' sines (24-bit amplitude words, about 0.07 µV steps; corrected for the DAC's sinc droop and the calibration's output response) + offset (2^{-16} DAC step) + PID terms + a monitor; TPDF dither; limits clamp every sample (measured: 0.150009 V for 0.15 V); clipped samples counted
Demodulators	4; input IN1, IN2 or IN1 − IN2 (offsets and optional digital AC coupling, 1st order, any corner); reference m·f_A + n·f_B, m = 0 … 1023, n ∈ {−1, 0, +1}, exact to the 48-bit phase; sine (22-bit interpolated table, error ≤ 0.57 LSB of 2^{-17} before the 18-bit rounding, computed) or square reference (reports the fundamental)
Mixing and decimation	125 MSa/s products; CIC of order 2, decimation 128: frame rate 976.5625 kHz; the 2f product falls on the CIC's nulls when f is a multiple of 488.28125 kHz
Filters	1 … 8 cascaded RC sections (6 … 48 dB/oct) on X and Y, τ = 0.5 µs … 10^6 s (realized within $2{\cdot}10^{-5}$, computed); ENBW and settling time reported (`status()`); measured: τ = 0.99977 of the set 0.1 s from a step
Readings	X, Y, R (V rms or V peak), θ (2^{-32} cycle), unwrapped θ (48 bits: ±32768 cycles), dθ/dt (2^{-32} cycle per frame: 0.23 mHz steps, ±488 kHz); one frame read consistently (SNAP: about 3.4 ms per reading through Python on the board, 1.3 ms of it the framework's check that the FPGA is loaded, measured)
Accuracy	R against the amplitude of the same captured input samples (a least-squares fit): 0.99984 … 1.00005 from 1 kHz to 10 MHz; harmonic 3: 1.00000, sideband: 0.49999 for 0.5, square reference 0.99944, IN1 − IN2 1.50089 for 1.5 (measured). Absolute amplitude: the board calibration (DC) times the board's analog response (below)
Analog loopback response	OUT1 → IN1 and OUT2 → IN2 through 50 cm RG316 and a T with 50 Ω, relative to the set amplitude (calibration of 2026-10-04): 0.960 (IN2 0.962) from DC to 1 kHz (the calibration does not match this cabling by 4 %), a step of +6.8 % at a pole-zero near 21 kHz (zero 20.6 kHz, pole 22.0 kHz; +1.75° at 20 kHz), 1.022 at 1 MHz, 1.056 at 10 MHz, 1.08 near 19 MHz, 1.00 at 50 MHz, 0.74 at 55 MHz, 0.34 at 60 MHz (measured); correct it with response tables (`CAL RESPONSE IN1 …`, `CAL RESPONSE OUT1 …`)
Linearity	the converter loop (OUT1 → IN1, dither, 0.1 V offset): within 0.34 % from 0.8 V to 10 mV, 2.5 % at 1 mV (measured, against 0.4 V)
Noise	IN1 and IN2 (LV, 50 Ω terminated through the loop cables, outputs off): 5.2 µV/√Hz at 37 Hz, 2.2 at 113 Hz, 1.3 at 317 Hz, 1.1–1.3 from 1 to 10 kHz, 1.6 at 31 kHz, 0.65 at 100 kHz, 84 nV/√Hz at 1 MHz, 90 nV/√Hz at 10 MHz; mains at 50 Hz about 25 µV (measured, from X and Y of captured readings; the idle outputs included)
Stability	2 h of readings at 1 MHz, 1.3 MHz, 10 kHz and 10.7 MHz in loopback (die 71–73 °C): R 56–59 ppm rms on all four together, Allan deviation 10 ppm at 1 and 10 s, 20 ppm at 100 s (the converters' common gain; the IN1/IN2 ratio 3.8 ppm at 1 s); θ 0.0014° rms at 1 MHz, 0.014° at 10.7 MHz, following the temperature as a 15 ps/K delay (measured)
Crosstalk	OUT2 → IN1: −54.4 dB at 100 kHz, −55.1 dB at 1 MHz, −55.2 dB at 10 MHz (measured; on IN1 while IN2 carries OUT2)
Phase	referred to the OUT1 connector (`docs/THEORY.md`, "Phase reference"): the driver removes the core's 14 samples and the inputs' stored converter loop delays (this board: 119.91 ns IN1, 119.99 ns IN2, measured; a pure delay from 1 to 10 MHz within 0.3°, the analog filters' group delay rising to 129 ns at 36 MHz: −39° at 39 MHz beyond it)

Quantity	Value
PIDs	4, at the frame rate; inputs X, Y, R, θ (the error wrapped to ±180°), dθ/dt, the unwrapped phase; targets: an oscillator's frequency (0.44 µHz steps) or phase, an output's amplitude of an oscillator, an output's offset; gains m·2^(e − 48) (|gain| < 2^{17} register units: a type-2 PLL up to about 19 kHz bandwidth, computed); D low-pass 2^k frames; limits with anti-windup; hold, hold on R below a level; lock detection (|e| ≤ tolerance for 2^k frames); a frame's targets act 3 frames after its end (3.07 µs) plus the output stage's 14 samples (computed)
Measured loops	PLL (200 Hz bandwidth, seeded by the counter, after its demodulator gathered 181 cycles unlocked): oscillator within 1 mHz of the input (mean), phase error 0.0004°, 0.04 Hz rms frequency jitter per reading; relocking from an offset of one bandwidth in 14.5 ms (100 Hz), 1.46 ms (1 kHz), 0.145 ms (10 kHz), pull-in range 7 bandwidths (100 Hz, 1 kHz) or 3 (10 kHz), which `lock_pll`'s counter seed starts well inside; with `acquire` (the frequency locked first) from 100 bandwidths (100 Hz, 1 kHz) and 30 (10 kHz: 300 kHz) in about 0.3 s; amplitude loop: R held at 50.015 mV for 50 mV, decay time constants 4.60, 1.35, 0.42 ms for I = 300, 1000, 3000 V/V/s (single-pole predictions 4.62, 1.39, 0.46 ms)
Stream	rows of X, Y of chosen demodulators every 2^j frames (976 562.5 / 2^j rows/s), timestamped by frame, flagged after a gap; FIFO of 8192 words; through the service: 7629 rows/s of two demodulators (no loss in 6 of 7 runs), 7.7–9.5 k rows/s at most (measured)
Capture	4 × 8192 words; sources: IN1, IN2, OUT1, OUT2, the demodulators' inputs (means over 2^k samples, k = 0 … 24) and X, Y, R, θ, dθ/dt, PID outputs and errors (per frame); triggers: immediate, DIO3_P rising/falling, IN1/IN2 Schmitt levels, software; pretrigger 0 … 8191
Frequency counter	IN1 or IN2, Schmitt levels, gate 2^g samples; within 0.1 ppm of the board's clock (measured, 0.1 s gate)
Settings after loading	everything off, outputs at 0 in converter units

Hardware setup

- **Outputs.** Into 50 Ω loads for the set amplitudes (the calibration's loopback); add a low-pass filter for the band you use to remove the DAC images near 125 MHz ± f.
- **Inputs.** LV jumpers (±1 V) for small signals; the inputs are DC coupled (use the digital AC coupling, `inputs.in1.ac`, to remove a DC level).
- **Loopback self-test.** Cable OUT1 → IN1 and OUT2 → IN2 (LV jumpers), 50 Ω terminators at the inputs; `make test-hw` uses it (it is awg's wiring; the DIO jumpers it may also have are not used by lockin 1.0).
- **Calibration.** Without one, nominal converter constants are used (about 14 % off in amplitude on this board). lockin can measure it in loopback (`CAL LOOPBACK`), and the inputs' loop delays (`analysis.measure_input_delay`, then `CAL DELAY IN1 …`).

Documentation

Document	Contents
docs/manual/user_manual.pdf	user's manual (26 pages, measured and computed figures): what lockin is, safety, the first measurement, every setting, readings, measurements (auto-phase, auto τ, sweeps and fits, captures, the counter, spectra, the zoom spectrum and noise density, harmonics), the data stream and long-term stability, control loops and PLLs, software, the SR86x port, calibration, applications, specifications, command reference
docs/manual/theory_manual.pdf	the theory of lockin and feedback (70 pages, computed figures): converters and calibration, oscillators, references and mixing, the CIC, RC filters, the CORDIC, noise and resolution, the frame engine, PIDs, PLLs and pull-in, the fast PIDs' delay budget, lock acquisition, loop measurement and tuning, nanosecond pins, the slow outputs' delta-sigma, acquisition and measurement
docs/PROTOCOL.md	every setting, Python methods, TCP commands, the data stream, command line
RECIPES.md	worked examples: a weak signal, harmonics, Bode plot and resonance fit, a PLL on an external signal, amplitude and offset loops, the stream, captures and spectra, phases at the connectors

Document	Contents
docs/THEORY.md	how it works: datapath, reference, CIC and filters, CORDIC, PIDs, outputs, the frame engine's schedule, apply, readings, FIFO, capture, resources, the design's history and limits
docs/REGISTERS.md	the hardware registers and word formats
docs/VERIFICATION.md	what was tested, when; resource use and timing
../common/CALIBRATION.md	the board calibration, shared by every design (with the inputs' response and loop delay)
../common/README.md	the shared framework: TCP service, common API, build workflow
NOTICE.md	sources
AGENTS.md	notes for maintainers and coding agents

Changing the design

Rebuilding needs Vivado 2025.1 on a build machine (../common/SETUP.md). From this directory:

```
make                        # create the project (first time), build, export the overlay
make test                   # offline tests: model, driver, protocol, client, stream, command line,
 ↪  GUI
make sim                    # simulation: system ID, reference, datapath, frame engine, core system
 ↪  against the model
make test-remote            # upload, program the board, check identity and power-up state
make install-service        # the control service, with lockin active
python3 ../common/tests/verify_build.py --design lockin
```

`make test-hw` runs `scripts/test_hw.py` on the board (loopback OUT1 → IN1, OUT2 → IN2).

Source layout

Path	Contents
`design.tcl`	design description for the common recipe: ID "LKIN", RTL, one `lk_core` cell, the trigger pin
`src/lk_core.v`	top: reset, sample counter, frame strobes, registers, datapath, engine, FIFO, capture, counter, clip counts
`src/lk_datapath.v`	the full-rate part: inputs, oscillators, sines, four demodulators, two outputs, dither
`src/lk_nco.v`, `src/lk_sine.v`, `src/lk_sine_rom.v`	oscillators (48-bit, phase preloads), sines (quarter-wave table with interpolation; the ROM is generated)
`src/lk_ref.v`, `src/lk_demod.v`, `src/lk_dsp_acc.v`	reference phase m·φ_A ± φ_B and its sine/cosine or square; mixers and the CIC's integrators in DSP48s
`src/lk_input.v`, `src/lk_out.v`, `src/lk_xoro.v`	input conditioning; the output's sum (DSP48 cascade), rounding, dither, limits; the random generator
`src/lk_engine.v`, `src/lk_iir.v`, `src/lk_cordic.v`, `src/lk_scale.v`, `src/lk_f32.v`	the frame engine: combs and RC filters, CORDIC, PIDs, monitors, readings, FIFO rows, the next targets
`src/lk_regs.v`, `src/lk_fifo.v`, `src/lk_capture.v`, `src/lk_counter.v`	AXI4-Lite registers and the apply, the FIFO, the capture, the frequency counter
`constraints/dio.xdc`	the trigger pin
`control/lockin/`	Python package: bit-exact model (`model.py`), driver, analysis (sweeps, fits, spectra, PLL, delay), TCP commands, client, data stream and the service's worker, command-line options, Jupyter GUI
`control/lockin.ipynb`	the notebook
`scripts/test_hw.py`	the loopback hardware test (`make test-hw`)
`docs/manual/`	the LaTeX manuals (`user_manual.tex`, `um_*.tex`; `theory_manual.tex`, `th_*.tex`, `references.bib`), their figures, values and board recordings (`data/`)
`tools/`	host-only: `theory_figures.py` (the theory manual's figures and numbers), `manual_figures.py` and `record_manual_data.py` (the user's manual's figures and its board recordings, through the service); numpy, matplotlib

Path	Contents
tests/	tb_ref.sv, tb_datapath.sv, tb_engine.sv, tb_system.sv (simulation), vectors.py → lk_vectors.sv, lk_*.mem (the model's vectors), Python tests, fake_core.py (the model as a register block, in a loopback)

Verification

Version 1.0 passes the offline tests, the four simulations (bit-exact against the model, the system bench through real AXI transactions) and all 31 loopback checks on the board: readings against the captured input to 0.01 %, harmonic, sideband, square reference and IN1 − IN2, the filter's time constant, a PLL, PID loops, limits, coherent phases and the stream (2026-10-25). After the engine's block-RAM rework (2026-10-26: about 990 LUTs freed) and with the SR86x port and the output analyses (2026-10-27), all 38 board checks pass, and again with the software shared with feedback in common/ and the manuals' recordings (2026-10-27). Build, timing and board results: docs/VERIFICATION.md.

Author

Rok Zitko, Jozef Stefan Institute (Ljubljana, Slovenia), rok.zitko@ijs.si.

lockin recipes

Worked examples. lockin is the driver on the board (from pynq.overlays.lockin import DRIVER, analysis; lockin = DRIVER()); with LockinClient on a PC the same calls work (lockin.sweep(...), lockin.auto_phase(1), ... run on the board). Volts at the connectors; X, Y, R in V rms.

A weak signal

Drive the experiment from OUT1 at 1.234 kHz and read the response on IN1 with 100 ms of 24 dB/oct filtering (ENBW 0.78 Hz: with the input's noise floor of about 1.3 µV/√Hz near 10 kHz on this board, about 1.2 µV rms of noise in X):

```
lockin.apply({"oscillators": {"1": {"frequency": 1234.0}},
              "outputs": {"1": {"enabled": True, "amplitudes": {"1": 0.5}}},
             "demodulators": {"1": {"input": "in1", "oscillator": 1, "time_constant": 0.1, "order":
              ↪  4}}})
print(lockin.status()["demodulators"]["1"])          # frequency, ENBW, settling time
analysis.auto_phase(lockin, 1)                       # the signal on +X
r = analysis.average(lockin, ("1",), count=100)["1"]
print(r["x"], r["x_std"])
```

A DC offset at the input does no harm (it is far from the reference); to remove a large one before the demodulators, {"inputs": {"in1": {"ac": True, "ac_corner": 1.0}}}.

Harmonics and intermodulation

A nonlinear device between OUT1 and IN1: demodulators 1–4 on the first four harmonics.

```
lockin.load_preset("harmonics")
r = lockin.readings()["demodulators"]
print([r[d]["r"] for d in "1234"])
```

Two tones f_A and f_B into a mixer, the sum product f_A + f_B on demodulator 3:

```
lockin.apply({"oscillators": {"1": {"frequency": 100e3}, "2": {"frequency": 37e3}},
              "outputs": {"1": {"enabled": True, "amplitudes": {"1": 0.2}},
                          "2": {"enabled": True, "amplitudes": {"2": 0.2}}},
              "demodulators": {"3": {"input": "in1", "oscillator": 1, "harmonic": 1,
                                     "sideband": {"oscillator": 2, "sign": 1}}}})
```

Bode plot and resonance fit

A resonator (a quartz tuning fork, a cantilever's piezo) between OUT1 and IN1:

```
values = analysis.sweep_values(32.6e3, 32.9e3, 151)
lockin.apply({"demodulators": {"1": {"time_constant": 10e-3, "order": 4}}})
result = analysis.sweep(lockin, "frequency", {"oscillator": 1}, values, count=4)
d = result["demodulators"]["1"]
fit = analysis.fit_resonance(result["values"], d["x"], d["y"])
print("f0 %.3f Hz, Q %.0f" % (fit["f0"], fit["q"]))
```

Over TCP: `SWEEP {"parameter": "frequency", "target": {"oscillator": 1}, "start": 32.6e3, "stop": 32.9e3, "points": 151, "fit": true}`. Nested sweeps (e.g. frequency for several drive amplitudes): `outer={"parameter": "amplitude", "target": {"output": 1, "oscillator": 1}, "values": [0.05, 0.1, 0.2]}`.

Follow the resonance: a loop on the resonator's phase

After the fit, keep oscillator 1 on the resonance with PID 1: the phase at f0 is the setpoint, and near f0 the phase falls by 360·Q/(π·f0) degrees per Hz, so an integral gain i = −2π·bw / slope gives a loop of bandwidth bw (keep it well below the resonance's half width f0 / (2Q) and the filter's bandwidth):

```
import math
slope = 360 * fit["q"] / (math.pi * fit["f0"])          # degrees per Hz at f0
bw = 1.0                                                # Hz
lockin.apply({"oscillators": {"1": {"frequency": fit["f0"]}},
              "pids": {"1": {"enabled": True, "input": {"demodulator": 1, "quantity": "phase"},
                             "setpoint": fit["phase"], "i": -2 * math.pi * bw / slope,
                             "target": {"kind": "frequency", "oscillator": 1, "output": 1},
                             "limits": [-50.0, 50.0]}}})
print(lockin.readings()["oscillators"]["1"])            # the live frequency follows f0
```

A PLL on an external signal

An external reference near 10 kHz on IN2; oscillator 2 follows it (counter first, then the loop), and OUT2 repeats it:

```
lockin.apply({"outputs": {"2": {"enabled": True, "amplitudes": {"2": 0.1}}},
              "demodulators": {"2": {"input": "in2"}}})
analysis.lock_pll(lockin, pid=1, bandwidth=100, oscillator=2, demodulator=2, hold_below=0.01,
 ↪  seed="in2")
```

Demodulators referenced to oscillator 2 (or its harmonics) now measure IN1 against the external reference. `hold_below` keeps the frequency when the reference disappears. The loop pulls in from about 7 bandwidths off (3 at a 10 kHz bandwidth; measured, see `docs/VERIFICATION.md`): without the seed, set the oscillator that close to the input first; farther off it does not lock and drifts to its frequency limit. The default limits, ±5 bandwidths around the oscillator's set frequency (`frequency_range`), keep a loop that lost lock inside that range, so it pulls back in once the input returns near the set frequency.

Where the counter cannot seed (a harmonic or sideband, IN1 − IN2, a noisy or weak signal), let the PID lock the frequency first, here anywhere within ±5 kHz of 10 kHz:

```
analysis.lock_pll(lockin, pid=1, bandwidth=100, oscillator=2, demodulator=2, acquire=5e3)
```

It locks dθ/dt, then switches to the phase with the limits centred on the frequency found (measured: from 100 bandwidths at 100 Hz and 1 kHz, 30 at 10 kHz, in about 0.3 s). The filter attenuates the beat at the start by about (offset / 8 bw)²: the signal must stay above the noise there.

Amplitude and offset loops

Keep R of demodulator 1 at 50 mV by the drive amplitude (P = 0, I = 50 V/V/s: about 6 Hz of bandwidth in a unity loop), and steer OUT2's offset to hold X at 0 through a DC actuator (the signs of p and i depend on how your system responds):

```
lockin.apply({"pids": {
    "2": {"enabled": True, "input": {"demodulator": 1, "quantity": "r"}, "setpoint": 0.05,
          "target": {"kind": "amplitude", "oscillator": 1, "output": 1}, "i": 50.0, "limits":
           ↪  [-0.19, 0.3]},
```

```
    "3": {"enabled": True, "input": {"demodulator": 1, "quantity": "x"}, "setpoint": 0.0,
          "target": {"kind": "offset", "output": 2}, "p": -1.0, "i": -1e3, "limits": [-0.5, 0.5]}},
    "outputs": {"2": {"enabled": True, "limits": [-0.5, 0.5]}}})
```

A monitor puts a reading on an output as an analog voltage (1 V per mV of R):

```
lockin.apply({"outputs": {"2": {"enabled": True, "monitor": {"source": "r", "index": 1, "scale":
 ↪  1000.0}}}})
```

The data stream

Rows of X, Y of demodulators 1 and 2 at 976.5625 rows/s, on a PC:

```
from lockin import DataStream                          # (on the board: from pynq.overlays.lockin)
lockin.apply({"stream": {"rate": 1000, "demodulators": [1, 2]}})
with DataStream("192.168.0.149", interval=0.1) as s:
    for _ in range(50):
        rows, lost = s.read()
        for frame, t, x1, y1, x2, y2, *rest in rows:
            ...
```

A row with `flags` 1 follows a gap. The service keeps up with about 7.6 k rows/s of two demodulators on this board; above, rows are dropped at the FIFO (counted in `HW?`).

Captures and spectra

IN1 and OUT1 at 1.95 MSa/s (means of 64 samples), with X of demodulator 1:

```
cap = lockin.capture(("in1", "out1", "x1"), rate=125e6 / 64)
spec = analysis.spectrum(lockin, "in1", averages=8, rate=125e6 / 16)   # dBV and V/sqrt(Hz)
```

Triggered by IN1 crossing 0.1 V upwards, 1000 words before the trigger: `lockin.capture(("in1",), trigger="in1_up", level=(0.1, -0.1), pretrigger=1000)`.

Noise around the reference, a time constant, stability

The spectrum around demodulator 1's reference is a zoom FFT of X + iY, here ±50 kHz in 954 Hz bins. Its density is the input's own around the reference, the filter's response divided out:

```
z = analysis.zoom_spectrum(lockin, 1, span=100e3, points=1024, averages=8)
n = analysis.noise_density(lockin, 1)              # V/sqrt(Hz) at the reference, signal removed
print(n["density"], n["density_xy"])               # spectral median; from X, Y's variance
t = analysis.auto_tau(lockin, 1, precision=1e-4)   # the shortest tau giving R to 1e-4
```

For drift over minutes, record the stream and take its Allan deviation (on a PC, with `LockinClient`):

```
rec = client.record(600, rate=976.5625, demodulators=(1,))       # 10 min at about 1 kHz
adev = analysis.allan_deviation(rec["x1"], rec["rate"])          # {"tau", "adev", "n"}
stats = analysis.RunningStats().add(rec["x1"]).summary()
```

The Allan deviation's minimum is the averaging time beyond which drift dominates.

Phases at the connectors

Until the inputs' loop delays are measured, θ includes the converters' delay (about 120 ns, i.e. 43° at 1 MHz). With OUT1 → IN1 and OUT2 → IN2 cabled (cables of known delay, e.g. 0.5 ns each):

```
analysis.measure_input_delay(lockin, "in1", output=1, cable_delay_s=0.5e-9, store=True)
analysis.measure_input_delay(lockin, "in2", output=2, cable_delay_s=0.5e-9, store=True)
```

Afterwards a sine at an input in phase with OUT1's sine reads θ = 0, and phases are referred to the OUT1 connector (docs/THEORY.md). To correct the inputs' amplitude response as well, measure R against a calibrated source and store `[[Hz, read/actual], ...]`, e.g. `lockin.set_input_response("in1", [[1e3, 0.96], [1e5, 1.02], [1e7, 1.056]])` (these are this board's loopback figures: output and input together).

How lockin works

lockin is one module, `lk_core`, between the converters' AXI streams, with a 256 KiB AXI4-Lite register window at 0x43C00000 (docs/REGISTERS.md). Everything runs on the 125 MHz sample clock. The Python model `control/lockin/model.py` is the bit-exact reference of the RTL: the benches (`tests/tb_*.sv`) compare the RTL with it sample by sample, frame by frame and through real AXI transactions.

```
IN1, IN2 ─► inputs: 2c − background (offset, AC: the engine's DC estimate) ─► x1, x2, x1 − x2
oscillators A..D: 48-bit phase φ = acc + offset + PID phase; f = F + PID frequency
  ├─► 4 sines ─► per output: Σ (A_k + dA_k) sin φ_k  (DSP48 cascade) + DC + slow word ─► dither,
  ↪  round, clamp ─► OUT1, OUT2
  └─► demodulator d: ψ = m φ_A + n φ_B + POW ─► cos ψ, −sin ψ (or squares)
         x · ref ─► I1 (mixer DSP) ─► I2 (DSP) ──────────────── per sample (125 MSa/s)
═══════ frame = 128 samples, 976.5625 kHz ═══════════════════════════════════════════
combs (CIC, order 2) ─► RC sections 1..8 on X and Y ─► normalize ─► CORDIC: R, θ ─► unwrap, dθ
X, Y, R, θ, dθ ─► 4 PIDs ─► next targets (frequency, phase, amplitude, DC), monitors
readings ─► snapshot, ring of 4 frames (capture), FIFO rows (stream)
```

Samples, frames and applies

A **sample** n is one clock; the sample counter `now` reads n in the clock where the inputs' stream carries sample n. A **frame** is 128 samples: frame k ends at sample 128 (k + 1) − 1. In the **period** P (samples 128 P … 128 P + 127) the engine filters frame P − 1, finds R and θ of frame P − 2 and computes the PIDs of frame P − 3; the targets of frame k act from sample 128 (k + 4) (L_FRAMES = 4).

Configuration is double-buffered. Writes go to pending words: flip-flops for the full-rate settings (oscillators, demodulators' references, outputs, inputs) and the pending half of a block RAM (the engine's coefficient bank: filters, PIDs, monitors, stream). An **apply** (CTRL APPLY_NOW: the next frame boundary at least two samples ahead; APPLY_AT: APPLY_TIME rounded up to a frame boundary, late if less than two samples ahead) makes all of them live together: the datapath's stages load their pending words at the strobes apl[j], j clocks after the last clock of the frame, so that every stage switches on the same sample; the bank flips at the same boundary, and its new pending half is refilled from the active one (512 clocks). AXI writes wait (no AWREADY) from three clocks before the boundary until the stages are loaded and the refill is done. The apply count (modulo 256) is in every FIFO row.

Coherent phases. In `phase_mode` "coherent" the driver applies at a sample n_a about 2 ms ahead and preloads every oscillator's phase accumulator with F (n_a − t0) (REG_NCO_P, LOADS): each oscillator runs as if it had run at its present frequency since t0. Two oscillators at one frequency keep their phase difference, and returning to a frequency restores its phase (measured: within 0.001°).

Oscillators and sines

Each oscillator adds its 48-bit frequency word F (+ the PID's dF) to its accumulator every sample; its phase is acc + POFF·2^{16} + dP (the PID's phase target). Frequencies are 0 … 62.5 MHz in steps of 125 MHz / 2^{48} = 0.44 µHz.

Sines come from a quarter-wave table of 1024 entries (`lk_sine_rom.v`, generated by the model) with a first-order Taylor correction in a DSP48: $s(p) = S \sin(2\pi\,(p + ½)/2^{24})$ for the top 24 bits of the phase, $S = 2^{17} - 1$. The interpolated 22-bit value is within 0.57 LSB of 2^{-17}; the rounded 18-bit sine within 1.07 LSB (computed over all phases in `tests/test_model.py`).

The reference

Demodulator d's reference phase is the top 24 bits of m·φ_A + n·φ_B (48-bit, exact: two DSP48s form m·φ_A's low and high halves) plus its offset POW (2^{-24} cycle), with m = 0 … 1023 the **harmonic** of oscillator A and n ∈ {−1, 0, +1} a **sideband** with oscillator B. The references are r_I = S cos ψ and r_Q = −S sin ψ, or for a **square reference** ±S from the quadrants (the driver scales its readings by π/4, the fundamental's share).

Inputs

The 14-bit converter codes c become x = sat16(2c − bg) in half converter steps (IN1, IN2) or sat16(2(c1 − c2) − (bg1 − bg2)) (IN1 − IN2). The background bg is the input's offset word (the driver puts the converter's own offset A there, so that x is zero at 0 V) plus, with **AC coupling**, the engine's estimate of the input's DC level: one first-order

RC section per input on the decimated input (its own channel of the filter engine, any corner frequency), updated every frame. The datapath subtracts it from every sample, so the high-pass costs no full-rate filter.

Mixers and the CIC

Each demodulator has two DSP48 mixers, x·r_I and x·r_Q (16 × 18 bits), whose P registers accumulate the products: the CIC's first integrator I1. A cascaded DSP48 (P + PCIN) is the second integrator I2. Both wrap modulo 2^{48}. Once per frame the engine latches I2 and runs the two combs: c2 = (v − v') − (v' − v''), the sum of the products over the last two frames weighted as a triangle: a CIC of order 2 with decimation 128, DC gain $128^2 = 2^{14}$. The filters' input is u = round(c2 / 2^{15}), so that a sine of amplitude a (x units) reads X = (S / 4) a cos θ.

The CIC's response is sinc²(f / 976.5625 kHz): the mixers' 2f products are rejected by (B / (2f mod f_frame))² for a filter bandwidth B, and vanish when 2f is a multiple of the frame rate (f a multiple of 488.28125 kHz). The RC filters after it remove the rest.

RC filters

Each of the 10 channels (X and Y of 4 demodulators, the 2 inputs' DC estimates) has 8 first-order sections, evaluated by one multiplier in a fixed schedule (`lk_iir.v`):

```
Y += round(floor((T − Y) / 2^32) · m · 2^(15 − s)),   α = m · 2^−(17 + s)
```

with the state Y in 72 bits (40 fraction bits), T the input (u·2^{40}) or the previous section's state, m a 17-bit mantissa ($2^{16} \le m < 2^{17}$) and s the exponent: α = 1 − e^(−1/(τ f_frame)), realized within $2 \cdot 10^{-5}$ for τ from 0.5 µs to 10^6 s. The configured **order** (1 … 8) picks the section whose state is the output (order 0: the CIC output itself). The driver reports the equivalent noise bandwidth ENBW = Γ(n − ½) / (4 √π τ Γ(n)) (1/(4τ) for order 1) and the settling time to 10^{-4}.

R and θ

One CORDIC core (26 iterations) serves the four demodulators in turn. Before it, X and Y are shifted together by sh = L − 29, L the bit length of their one's-complement magnitudes, so that the core always works with 29 significant bits: θ is accurate for signals of a few nV as for volts. R is shifted back (and divided by the CORDIC gain K in the driver). θ is a 32-bit fraction of a cycle; the unwrapped phase adds each frame's change dθ (a signed 32-bit wrap) to a 48-bit sum (±32768 cycles); dθ per frame is the **frequency deviation** (2^{-32} cycle per frame = 0.23 mHz).

PIDs

Each PID reads one quantity v of one demodulator's results (frame k): X, Y, R, θ, dθ or the unwrapped phase, all in register units, and computes (`model.Pid`):

```
e = sat48(setpoint − v)          (θ: e = signed32(setpoint − θ), within ±½ cycle)
P = e·kp;   I ← clamp(I + e·ki, lo, hi);   D = (e − e_prev)·kd, low-passed by 2^−kdf
out = clamp(P + I + D, lo, hi)
```

with gains g = m·2^(e − 48) (18-bit mantissa, e ≤ 48: right shifts only, |g| < 2^{17} register units; the integrator keeps 16 more fraction bits, so ki carries 2^{16}). The driver converts p (target units per input unit), i (per second) and d (seconds) to these words with the input's and target's scales; a gain beyond the range is refused with its limit in physical units. **Hold** (manually, or when R of the demodulator is below `hold_below`) freezes I and zeroes P and D. For θ the error wraps (a 32-bit difference): the classic phase detector, whose loops lock from any starting phase; the unwrapped phase (quantity 5) keeps every cycle since the FPGA's reset (±32768 cycles), so a loop on it first winds back the cycles accumulated before it started. **Lock**: |e| ≤ tolerance for 2^w frames sets the locked flag (in the readings, the FIFO rows and the status). A disabled PID outputs 0 and clears its state; `reset_pid` clears I and D at the next frame.

The outputs of the PIDs with the same target are **summed**: the frequency and phase targets of each oscillator, the amplitude target of each oscillator on each output (saturated to 25 bits), the DC target of each output (added to the slow word). The engine computes the PIDs in a 27-step micro-sequence per PID (its 15 coefficient words read in steps 0 … 14, the gain stage for P, I and D in steps 6 … 8, results in 13 … 15, the arithmetic in steps 13 … 20 with one adder per step, the state update in step 22, its target sum in step 23, its readings in 23 … 26); PID p + 1 starts 22 steps after PID p.

Loop bandwidth. The targets of frame k act from sample 128 (k + 4): 3.07 µs after the frame's end, plus the output stage's 14 samples and the converters. With the demodulator's filter, PID loops of several kHz bandwidth are practical. For a type-2 PLL (`analysis. pll_settings`: P = −2π·bw/360 Hz per degree, I = P·2π·bw/4 per second) the integral gain reaches the hardware's limit at about 19 kHz.

PLL acquisition. The PLL's phase detector is θ, wrapped to ±180°: a sawtooth that, together with the demodulator's filter (order 2, corner 8·bw), pulls the loop in from about 7 bandwidths (3 at 10 kHz, where the loop delay adds to the filter's lag; measured). From farther, `lock_pll` either sets the oscillator from the frequency counter first (`seed`), or first locks the frequency (`acquire`): the same PID then takes dθ/dt as its input, a type-1 loop (I only, −π·bw per second, at most 3000 /s: a frequency-to-frequency gain reaches the hardware's limit near 3800 /s) whose detector reads the beat's frequency however far it is attenuated by the filter (within ±488 kHz, half the frame rate), as long as the signal stays above the noise. Once the beat stays below bw/2, one apply turns the PID into the PLL with its limits centred on the correction found; the integrator, in the target's units, keeps that correction, so the switch needs no reset and leaves no transient beyond the new P term.

Outputs

Each output sums its four oscillators' sines in a DSP48 cascade: term k computes (A_k + dA_k)·s_k in the DSP48's pre-adder and multiplier and adds it to the previous term's P. A_k is the configured amplitude (25-bit word, full scale 2^{24}) and dA_k the PIDs' amplitude target; the pre-adder sums them **modulo 2^{25}** (the driver bounds the PID's limits so that A + dA stays within $\pm(2^{24} - 1)$; Vivado's DRC warnings REQP-1723/1724 about the pre-adder's range refer to this). The first term's C input adds the DC word (configured offset + the PIDs' DC targets + the monitor, 2^{-16} DAC steps), the rounding constant and, with **dither**, a triangular-PDF value of ±1 DAC step from a xoroshiro128+ generator (two 16-bit uniforms per output per sample). The sum, rounded to 14 bits, is clamped to the output's **limits** (CLAMP; clipped samples are counted and flagged in the frame's FIFO row). PROTECT locks an output's CLAMP word until the FPGA is reloaded. Amplitudes are corrected for the DAC's zero-order-hold droop sinc(f / 125 MHz) and the board calibration's output response.

A **monitor** puts any reading (X, Y, R, θ, dθ of a demodulator, a PID's output or error) on an output's DC word, scaled (volts per unit) and saturated: an analog copy of a reading, updated every frame.

Phase reference and units

X is the in-phase part of a sine. The hardware's references are cos ψ and −sin ψ, so a sine sin φ in phase with the oscillator would read θ = −90°. The driver writes the demodulator's POW as the set phase − 90° − 360°·f·T: an input V sin(ωt + θ) at the connector reads X = V cos θ, Y = V sin θ (rms by default, `units` "peak" for amplitudes).

Phases are referred to the OUT1 connector. T is the loop delay from a sample's oscillator phase at OUT1's connector to the input sample that sees it there: the core's 14 samples (output latency) plus the input's calibrated **converter loop delay** (`rpfpga.calibration.adc_delay`: from OUT1's DAC port to the input's ADC port, cables excluded; 0 until measured). `analysis.measure_input_delay` measures it from the phase slope of a loopback (122 ns OUT1 → IN1 with the test cable on this board, measured) and stores it; then a sine at an input connector in phase with OUT1's sine reads θ = 0, and IN1 − IN2 comparisons are skew-free. f is the demodulator's configured frequency m·f_A + n·f_B: a PID that steers the frequency by Δf leaves a phase error of 360°·Δf·T (0.04° per kHz for 120 ns).

Volts. X, Y and R are scaled by the input's calibrated gain (half converter steps per volt), the reference (sine: S/4 per unit amplitude; square: × π/4), and the input's calibrated amplitude **response** at the demodulator's frequency (1 without a table). IN1 − IN2 uses the mean of both inputs' gains, responses and delays.

The frame engine's schedule

`lk_engine.v` runs a fixed 128-clock schedule per period P (slot = the sample counter's low 7 bits; coefficient words are read three clocks after their address is set):

Slots	Work
1 … 6	the filter words (4 demodulators, 2 inputs) are read
16	the I2 words of the 10 channels are latched
17 … 27	the combs, two stages per channel
19 + 10 r + c	section r of channel c issued (results 10 clocks later, forwarded); the last tap leaves in slot 106
14 + 22 p	PID p of frame P − 3: steps 0 … 26

Slots	Work
99, 101, 107	the FIFO's word; the monitors' source and gain words (issued in 105 and 111)
103 …	the FIFO row of frame P − 3 (every 2^j frames)
105 … 122	the next targets: the PIDs' sums read, saturated and written
106	the taps' frame is latched; the snapshot's frame is taken
107 … 114	the taps X, Y of frame P − 1 read: converted to floats, and demodulator 0's normalized
109 + 27 d	demodulator d of frame P − 1: normalization (3 clocks; its taps read again in the two clocks before, d = 1 … 3), the CORDIC core from 112 + 27 d, done D = 11 + 27 d of period P + 1; X, Y, θ, R, dθ and the unwrapped phase written in D + 1 … D + 6
14, 41, 68, 95 (+ 0 … 4)	R, θ, dθ and the unwrapped phase of frame P − 2 converted (D + 3, from the write-back's registers)
115 … 126	the snapshot's oscillator words, DC estimates and frame number
125	the monitors' slow words

A conversion's command is set in clock t, its value selected in t + 1, and its float written three clocks later (into the readings ring and, when requested, the snapshot).

The engine's memories are block RAMs with one read port each, shared by the schedule (no two readers in a clock): the taps (by frame parity: read in 107 … 114 and in the two clocks before each later normalization), the results (X, Y, R, θ, dθ, the unwrapped phase by frame parity: read for the previous θ and unwrapped phase in D − 1 and D, by the PIDs in steps 3 and 4, by the monitors in 104 and 110) and the PIDs' state (the integrator, the D filter, the previous error, the lock count, the output: read in steps 0 … 3 and by the monitors, written in steps 22 … 26 with the error and the output first). The schedule's registered decodes and multi-clock arithmetic are what closed timing on the 7010 (docs/VERIFICATION.md); the model is unaffected because every result meets its deadline in the same frame.

Readings, FIFO, capture, counter

Snapshot. CTRL SNAP takes the next latched frame (a request before slot 106 of period P takes frame P − 1): its X, Y, R, θ, dθ, unwrapped phase, the PIDs' outputs, errors, integrators and flags, the oscillators' live frequencies, the inputs' DC estimates and the frame number, all of one frame, readable when SNAP_PENDING clears.

Readings ring. Every reading of frame k is also written as a float into a ring of four frames (`{k mod 4, source}`); the capture keeps copies of it and the FIFO reads its rows from it.

FIFO. Every 2^j frames (rate 976 562.5 / 2^j rows/s) the engine pushes a row: the frame number (48 bits), the apply count, the PIDs' locked flags and the frame's clip flags, and X, Y (floats) of the chosen demodulators. 8192 words; a row that does not fit is dropped and counted. An apply that starts a stream clears the FIFO first: an earlier stream's rows left in it would arrive first, with their old frame numbers. Reading the FIFO window pops one word per read. The board's service drains it every 50 ms and publishes the rows to `DATA STREAM` clients (numpy decoding).

Capture. Four channels of 8192 32-bit words in block RAM. Full-rate sources (IN1, IN2, OUT1, OUT2 codes, the demodulators' inputs) store the sum over 2^k samples (k ≤ 16; above, the sum / 2^(k − 16)); frame-rate sources (X, Y, R, θ, dθ, PID outputs and errors) store the reading of the frame (4 frames back: `floor(n / 128) − 4`). The capture stage runs 15 clocks after the sample counter with every source of the same sample. Triggers: immediate, the DIO3_P pin's edges (two synchronizer stages), IN1/IN2 Schmitt levels, software; the trigger is accepted once the pretrigger words are recorded.

Frequency counter. Rising edges of IN1 or IN2 through Schmitt levels in a gate of 2^g samples: the count and the first and last edge's samples give the frequency.

Spectra and statistics of X and Y

Software (`analysis.zoom_spectrum`, `noise_density`, `auto_tau`, `allan_deviation`, `RunningStats`; the capture and the stream are the hardware).

The demodulator's output as a complex signal. With rms outputs, X and Y are √2 times the low-passed products of the input v with cos and sin of the reference phase. Then z = X + iY = √2·LPF(v·e^(−iφ(t))) is the input's spectrum shifted down by the reference frequency f_ref and filtered:

- a tone of rms amplitude A at f_ref + δ gives z = A e^(i(2πδt + θ)), a line at +δ;

- for input noise of one-sided density S_v(f) (V²/Hz), the product with e^(−iφ) has the two-sided density S_v(f_ref + δ)/2, and the factor √2 doubles it.

So **the two-sided density of z at δ equals the input's one-sided density at f_ref + δ**, times |H(δ)|². This is a zoom FFT: the resolution and span are set by z's sampling, not by f_ref. The filter H is the CIC's sinc²(δ/f_frame) times the n RC sections at the frame rate, $[\alpha/(1-(1-\alpha)\,z^{-1})]^n$ with z = e^(i2πδ/f_frame) and α = 1 − e^(−1/(τ f_frame)) (the model's realized α).

Sampling. A capture of x_d, y_d at 976.5625 kHz/2^j holds one frame's reading per word, every 2^j frames, with no averaging. Whatever lies above half that rate folds back unless the RC filter removed it. `zoom_spectrum` therefore picks the rate from the span (default 8 ENBW) and reports |H|² at half the rate (`aliasing`). Each record of N words gets a Blackman-Harris window w, so the bin width is rate/N and a bin's noise bandwidth is rate·Σw²/(Σw)². The averaged power gives:

- the density, √(|Z|²/(rate Σw²)) per bin;
- the amplitude of a tone in a bin, |Z|/Σw;
- with `compensate`, both divided by |H|, which is reliable where |H|² is not tiny.

In the model a tone 8 bins above the reference appears at +15.26 kHz with its rms amplitude within 1 %.

Noise density (`noise_density`). Each record's mean (the signal at the reference) is removed, so that the window's −92 dB sidelobes cannot hide the noise. The density is the median of the compensated power density over |δ| ≤ ENBW/2, excluding ±4 bins around 0, divided by `median_factor(K)`, the median of a mean of K exponential variables (a K-times averaged bin is χ^2 with 2K degrees of freedom; ln 2 for K = 1). The median ignores spurs. For white noise, X and Y each have the variance S_v·ENBW (ENBW one-sided: $\Gamma(n-\tfrac{1}{2})/(4\sqrt{\pi}\,\tau\,\Gamma(n))$). So √((σ_X² + σ_Y²)/(2·ENBW)) is a second estimate, `density_xy`. On synthetic white noise through the exact filter, both come within 10 % (`tests/test_control.py`).

Time constant from the signal (`auto_tau`). The output noise is e_n√ENBW. For a precision p of R, ENBW ≤ (pR/e_n)², which gives $\tau = \Gamma(n-\tfrac{1}{2})/(4\sqrt{\pi}\,\Gamma(n)\,\mathrm{ENBW})$ at the chosen order, within 0.5 µs … `max_tau`. The settling time follows from τ and n (`settle_time`).

Allan deviation (`allan_deviation`). It applies to samples y_i taken at a rate f_s, each standing for an interval 1/f_s (captures, or the stream through `LockinClient.record`). It is the overlapping estimator

$$\sigma^2(m/f_s) = \tfrac{1}{2}\,\langle(\bar{y}_{j+m} - \bar{y}_j)^2\rangle_j, \quad \bar{y}_j = \text{mean of } y_j \ldots y_{j+m-1}$$

computed from cumulative sums. White noise gives σ/√m, and a drift D gives Dτ/√2. Its minimum marks the averaging time beyond which drift wins: a useful longest time constant. `RunningStats` keeps a running mean, standard deviation (Welford's update), minimum and maximum.

Resources

Build 0x2ae6d985 of 2026-10-26 (Vivado 2025.1, xc7z010-1; the whole design with the PS7, interconnect and converters' IP):

Resource	Used	Of	
LUTs	11913	17600	67.7 % (LUT as memory 365)
Flip-flops	14713	35200	41.8 %
Slices	4211	4400	95.7 %
Block RAM tiles	54	60	90 % (capture 32, FIFO 8, readings and snapshot, bank, the engine's taps, results and PID state)
DSP48E1	56	80	70 %

Timing: WNS +0.088 ns, WHS +0.019 ns at 125 MHz; the worst paths are the IIR's combs. 1.0 as first built (0x05dbcb82, 2026-10-25) used 12 902 LUTs (1 258 as memory), 96.7 % of the slices and 51 block RAM tiles: the engine's LUT memories (the taps, the results, the PIDs' state) moved to block RAM with the schedule above and the capture's intervals of 2^17 samples and more share one shifter (feedback 2.0's rework), freeing about 990 LUTs. In 1.0 the engine was about half of the logic, the datapath (4 references, 4 × 2 mixers with their CIC integrators, 4 oscillators, 2 outputs) about a quarter.

The design's history and limits

An area study (2026-10-26) placed the planned 2.0 hardware (fast PIDs, a TTL reference with 1 ns timestamps and a sync output, AO/AI) next to lockin 1.0: 86 % of the LUTs, 4 399 of 4 400 slices, WNS +0.010 ns, and a leaner trial failed timing. That hardware therefore went to a second bitstream, feedback: lockin's core cut to two demodulators and two oscillators, with fast PIDs, lock acquisition, AO/AI and status pins in 1.0 and the TTL reference, sync output and 1 ns trigger in its 2.0.

lockin kept four demodulators and grew in software: the SR86x-compatible port and the analyses of X and Y (2026-10-27). feedback 2.0's block-RAM rework, back-ported on 2026-10-26, brought lockin to 68 % of the LUTs and 96 % of the slices, with a thin timing margin (+0.088 ns).

Complete for this device. On 2026-10-27 both designs were declared complete for the xc7z010. The hardware ideas that remained, a synchronous filter (whole reference periods) and a mains canceller (50/60 Hz harmonics subtracted at the inputs), would need a larger FPGA or the removal of other features.

lockin: settings, Python methods, TCP commands, data stream, command line

All values are physical: **volts at the SMA connectors**, **Hz**, **degrees**, **seconds**. X, Y and R are **rms volts** (`units` "rms") or **peak volts** (`units` "peak") of a sine at the demodulator's frequency, at its input (IN1, IN2 or IN1 − IN2); output amplitudes are peak volts. Units are numbered from 1 (keys and references). The common commands and methods (CONFIG, SNAPSHOT?, PRESET, SAFE, CAL, RANGE, ...) are in ../../common/README.md.

The configuration

`lockin.config()` returns, and `lockin.apply(patch)` merges, a JSON-compatible dict. The returned values are the realized ones (frequencies on the 0.44 µHz grid, quantized time constants and gains).

```
{"oscillators": {"1": OSCILLATOR, ..., "4": ...},
 "demodulators": {"1": DEMODULATOR, ..., "4": ...},
 "inputs": {"in1": {"offset": 0.0, "ac": False, "ac_corner": 10.0}, "in2": {...}},
 "outputs": {"1": OUTPUT, "2": OUTPUT},
 "pids": {"1": PID, ..., "4": ...},
 "stream": {"rate": 0.0, "demodulators": [1]},
 "units": "rms",                  # or "peak"
 "phase_mode": "continuous",      # or "coherent": oscillator phases relative to t0
 "t0": 0}                         # zero time (a sample index) of the coherent phases
```

Key	Default	Meaning
`oscillators.k.frequency`	1e6	Hz, 0 ... 62.5 MHz (steps of 0.44 µHz)
`oscillators.k.phase`	0.0	degrees (an offset of the oscillator's phase, its sines and every reference it drives)
`demodulators.d.input`	`"in1"`	`in1`, `in2` or `diff` (IN1 − IN2)
`demodulators.d.oscillator`	d	the reference's oscillator A (1–4)
`demodulators.d.harmonic`	1	m = 0 ... 1023: the reference is m·f_A (+ the sideband)
`demodulators.d.sideband`	`{"oscillator": 1, "sign": 0}`	n·f_B with oscillator B: sign −1, 0, +1
`demodulators.d.phase`	0.0	degrees subtracted from θ (the reference's phase)
`demodulators.d.reference`	`"sine"`	or `"square"` (±1; the readings are scaled to the fundamental)
`demodulators.d.time_constant`	1e-3	τ of each RC section, 0.5 µs ... 10^6 s
`demodulators.d.order`	4	sections 0 ... 8 (6 dB/oct each; 0: the CIC output only)
`inputs.in1.offset`	0.0	V subtracted from the input (before the demodulators)

Key	Default	Meaning
`inputs.in1.ac`	False	digital AC coupling: the input's DC estimate subtracted
`inputs.in1.ac_corner`	10.0	its corner frequency (Hz)
`outputs.o.enabled`	False	off: 0 V (amplitudes and offset 0)
`outputs.o.amplitudes`	`{"1": 0.0, ...}`	V peak of each oscillator's sine on this output (may be negative)
`outputs.o.offset`	0.0	V
`outputs.o.dither`	False	triangular dither of ±1 DAC step
`outputs.o.limits`	[−1.2, 1.2]	V: the FPGA clamps every sample (clipped samples counted)
`outputs.o.protect`	False	read only: the limits are locked (`protect(o)`) until the FPGA is reloaded
`outputs.o.monitor`	`{"source": "off", "index": 1, "scale": 1.0}`	a reading on the output's DC: `x`, `y`, `r`, `phase`, `frequency` of demodulator `index`, `pid` (output) or `error` of PID `index`; `scale` V per unit (per V, degree, Hz, or the PID target's unit)
`pids.p.enabled`	False	off: output 0, state cleared
`pids.p.input`	`{"demodulator": 1, "quantity": "x"}`	`x`, `y`, `r` (V), `phase` (degrees; the error wraps to ±180°, the setpoint is taken modulo 360°), `frequency` (Hz: dθ/dt), `phase_unwrapped` (degrees, every cycle since the FPGA's reset: a loop on it winds back the cycles accumulated before it starts)
`pids.p.setpoint`	0.0	the input's units; error = setpoint − input
`pids.p.target`	`{"kind": "none", "oscillator": 1, "output": 1}`	`frequency` or `phase` of `oscillator`; `amplitude` of `oscillator` on `output`; `offset` of `output`
`pids.p.p`	0.0	target units per input unit
`pids.p.i`	0.0	target units per input unit per second
`pids.p.d`	0.0	target units per (input unit per second)
`pids.p.d_filter`	0.0	s: the D term's low-pass (2^k frames; 0 none)
`pids.p.limits`	[−1.0, 1.0]	the output's range in target units (anti-windup); amplitude targets are also kept within the DAC word
`pids.p.hold`	False	freeze I, zero P and D
`pids.p.hold_below`	None	V: hold while R of the input's demodulator is below it
`pids.p.lock_tolerance`	0.0	input units: locked while |error| ≤ it for `lock_time`
`pids.p.lock_time`	1e-3	s (2^k frames)
`stream.rate`	0.0	rows/s, rounded to 976 562.5 / 2^j; 0 off
`stream.demodulators`	[1]	the demodulators in each row

`status()` adds per demodulator its `frequency` (Hz), `enbw` (Hz) and `settle` (s, to 10^{-4}), per output the `peak` and `offset`, per PID the realized `d_filter` and `lock_time`.

Python (driver on the board, `LockinClient` on a PC)

Method	
`apply(patch)`, `config()`, `safe()`, `describe()`, `status()`, presets, calibration	common (`rpfpga.Overlay`)
`readings()`	one frame's readings: `{"frame", "units", "demodulators": {d: {"x", "y", "r", "phase", "frequency_offset", "phase_unwrapped"}}, "pids": {p: {"output", "error", "integral", "locked", "saturated", "held"}}, "oscillators": {k: {"frequency"}}, "inputs_dc": {"in1", "in2"}}`

Method	
`hardware()`	applies, clipped samples, FIFO words and dropped rows, the PIDs' flags, protected outputs, the DIO3_P level, the sample counter
`clear_counts()`, `reset_pid(p)`, `protect(o)`, `zero_time()`	clip and drop counts to 0; clear a PID's state; lock an output's limits; t0 := now
`capture(sources, rate, trigger, pretrigger, level, timeout_s, words)`	8192 words of up to 4 sources (`in1`, `in2`, `out1`, `out2`, `xin<d>`, `x<d>`, `y<d>`, `r<d>`, `phase<d>`, `frequency<d>`, `pid<p>`, `error<p>`) at 125 MHz / 2^k (rounded); `trigger`: `immediate`, `rising`, `falling` (DIO3_P), `in1_up`, `in1_down`, `in2_up`, `in2_down` (Schmitt `level` (high, low) V), `software`. Returns `{"t", "rate", "trigger_sample", source: values}`; `words`: an immediate capture stopped after its first `words` words (a short record)
`count_frequency(channel, gate_s, hysteresis, level)`	the frequency (Hz) of IN1 or IN2, None without two edges
`drain_fifo()`, `decode_rows(rows, mask, config)`, `stream_rows(...)`	the FIFO's rows (the board's service drains them: use `DATA STREAM`)
`loop_delay(input)`, `phase_offset(config, d)`, `demod_scale(config, d)`	the phase reference and volt scales (docs/THEORY.md)
`set_input_delay(ch, s)`, `set_input_response(ch, points)`	common: the input's loop delay and amplitude response in the board calibration

The client adds `set_frequency(k, Hz)`, `oscillator(k)`, `set_oscillator(k, ...)`, `demodulator(d)`, `set_demodulator(d, ...)`, `set_inputs(...)`, `output(o)`, `set_output(o, ...)`, `enable(o, on)`, `set_amplitude(o, k, V)`, `set_offset(o, V)`, `pid(p)`, `set_pid(p, ...)`, `lock_pll(...)`, `set_stream(rate, demodulators)`, `stream_settings()`, `units()`, `set_units(u)`, `value(quantity, d)`, `average(...)`, `auto_phase(d)`, `sweep(...)`, `spectrum(...)`, `zoom_spectrum(d, ...)`, `noise_density(d, ...)`, `auto_tau(d, ...)`, `record(seconds, rate, demodulators)` (the stream for that long: `{"t", "x<d>", "y<d>", "rate", "lost", "gaps"}`); `DataStream(host)` reads the stream.

`control/lockin/analysis.py` (on the board, with the driver or the client):

Function	
`average(device, demodulators, count)`	mean X, Y, R, phase and standard deviations over `count` readings
`auto_phase(device, d)`	set demodulator d's phase so that Y = 0; returns it
`pll_settings(bandwidth, oscillator, demodulator, frequency_range, hold_below)`	a PID patch: a type-2 PLL (P = −2π·bw/360 Hz/°, I = P·2π·bw/4 /s), limits ±`frequency_range` (default 5·bw, inside the pull-in range)
`lock_pll(device, pid, bandwidth, oscillator, demodulator, frequency_range, hold_below, seed, gate_s, filter, acquire, acquire_timeout_s)`	the PLL, the demodulator's filter (order 2, τ = 1/(2π·8·bw)), optionally the oscillator set first from the counter on `seed` (`in1`, `in2`), or the frequency locked first within ±`acquire` Hz (`fll_settings`; then the PLL's limits are ±`frequency_range` around the correction found; RuntimeError if the beat does not stay below bw/2, and R above `hold_below` if given, within `acquire_timeout_s`, default 0.2 s + 100/bw)
`fll_settings(bandwidth, oscillator, demodulator, acquire_range)`	a PID patch: a type-1 frequency loop on dθ/dt (I = −π·bw /s, at most 3000 /s), limits ±`acquire_range`
`sweep(device, parameter, target, values, demodulators, settle, settle_factor, count, outer, restore)`	step `frequency`, `phase` (target `{"oscillator": k}`), `amplitude` (`{"output": o, "oscillator": k}`) or `offset` (`{"output": o}`); wait the slowest demodulator's settling time (or `settle` s); average `count` readings; `outer`: a nested sweep; returns `{"values", "demodulators": {d: {"x", "y", "r", "phase"}}}`. On the board it is an acquisition: another edit cancels it
`sweep_values(start, stop, points, scale)`	linear or `log` values
`fit_resonance(frequencies, x, y)`	a complex Lorentzian a/(1 + 2iQ(f − f0)/f0) + c: `{"f0", "q", "fwhm", "amplitude", "phase", "background", "residual"}` (numpy)

Function	
`spectrum(device, channel, averages, rate)`	an input's spectrum from captures (Blackman-Harris): `{"frequency", "dbv", "density", "bin", "enbw"}` (numpy)
`zoom_spectrum(device, d, span, points, averages, compensate, remove_mean)`	the spectrum around demodulator d's reference from captures of z = X + iY at 976.5625 kHz / 2^j ≥ `span` (default 8 ENBW): `{"offset", "frequency", "density" (the input's V/√Hz), "amplitude" (V rms per bin), "bin", "enbw", "rate", "reference", "filter", "aliasing", "x_rms", "y_rms"}` (docs/THEORY.md, "Spectra and statistics of X and Y"; numpy)
`noise_density(device, d, averages, points, exclude)`	the input's noise density around the reference: the zoom spectrum's median over ±ENBW/2 less ±`exclude` (default 4 bins), bias-corrected; and from X, Y's variance: `{"density", "density_xy", "x_rms", "y_rms", "enbw", "bins", "reference"}` (numpy)
`auto_tau(device, d, precision, order, max_tau, averages, apply)`	the shortest τ whose noise e_n√ENBW is at most `precision`·R: `{"time_constant", "order", "enbw", "settle", "noise_density", "r", "noise"}`
`filter_response(tau, order, offsets)`, `demod_z(device, d, rate, points)`, `tau_for_enbw(bw, order)`, `median_factor(k)`	the demodulator's response at offsets from its reference; a capture of z; τ for a noise bandwidth; a χ^2 median (numpy)
`allan_deviation(values, rate, taus)`	the overlapping Allan deviation of samples at `rate`: `{"tau", "adev", "n"}` (numpy)
`RunningStats()`	`.add(values)`, `.count`, `.mean`, `.std`, `.min`, `.max`, `.summary()`
`measure_input_delay(device, channel, output, cable_delay_s, frequencies, store)`	the input's converter loop delay from a loopback's phase slope (docs/THEORY.md, "Phase reference")
`enbw(tau, order)`, `settle_time(tau, order, fraction)`	the filters' noise bandwidth and settling time

TCP commands (port 5025)

Command	Reply
`OSC <k> <json>`, `OSC <k>?`	patch / query oscillator k
`FREQ <k> <Hz>`, `FREQ <k>?`	oscillator k's frequency (the realized one)
`DEMOD <d> <json>`, `DEMOD <d>?`	patch / query demodulator d
`TC`, `ORDER`, `HARM`, `INPUT`, `REF`, `DPHASE <d> <value>`	one demodulator setting: time constant, order, harmonic, input (IN1, IN2, DIFF), reference (SINE, SQUARE), phase
`INPUTS <json>`, `INPUTS?`	the inputs' offsets and AC coupling
`OUT <o> <json>`, `OUT <o>?`, `OUT <o> ON|OFF`	output o
`AMP <o> <k> <V>`, `OFFS <o> <V>`	oscillator k's amplitude on output o; output o's offset
`PROTECT <o>`	lock output o's limits until the FPGA is reloaded
`PID <p> <json>`, `PID <p>?`, `PIDRESET <p>`	PID p
`PLL <p> <json>`	`{"bandwidth", "oscillator", "demodulator"[, "range", "hold_below", "seed", "gate", "filter", "acquire", "acquire_timeout"]}`: `lock_pll`
`STREAM <json>`, `STREAM?`	the FIFO rows' rate and demodulators
`UNITS RMS|PEAK`, `UNITS?`	
`ZERO`	t0 := now; replies t0
`READ?`	`readings()` (JSON)
`X? <d>`, `Y? <d>`, `R? <d>`, `THETA? <d>`	one reading
`AVG? <count> [<d> ...]`	`average`
`AUTOPHASE <d> [<count>]`	the new phase
`HW?`, `CLEAR`	`hardware()`; `clear_counts()`
`CAPTURE? <json>`	`{"sources", "rate", "trigger", "pretrigger", "level", "timeout"}`: the capture without `t` (rebuild it from `rate` and `pretrigger`)
`COUNT? [IN1|IN2] [<gate s>]`	the frequency, or NaN

Command	Reply
`SWEEP <json>`	`{"parameter", "target", "values"` or `"start", "stop", "points", "scale", "demodulators", "settle", "settle_factor", "count", "outer", "restore", "fit"}`; with `fit` (a frequency sweep) also `fit: {d: fit_resonance}`. At most 40000 readings per reply
`SPECTRUM? [IN1|IN2] [<averages>] [<rate>]`	`spectrum`
`ZOOM? <json>`	`{"demodulator"[, "span", "points", "averages", "compensate", "remove_mean"]}: zoom_spectrum`
`NOISE? <json>`	`{"demodulator"[, "averages", "points", "exclude"]}: noise_density`
`AUTOTAU <json>`	`{"demodulator"[, "precision", "order", "max_tau", "averages", "apply"]}: auto_tau`
`DATA STREAM [<interval s>]`	the stream (below), on its own connection

Replies are `OK <value>` or `ERR <message>`; a refused setting changes nothing.

SR86x port (port 23)

A program written for a Stanford Research SR860/SR865, for example pymeasure's `SR860` class or a LabVIEW driver over a raw TCP socket, can drive lockin on the service's second port, 23 (an SR860's telnet port; `rpfpga.server --compat-port`). The port speaks a subset of the SR860's remote commands (its manual, chapter 4, "Programming"), mapped onto **demodulator 1** referenced to **oscillator 1**, with that oscillator's sine on **OUT1** (`control/lockin/compat.py`).

Replies. Replies are bare lines, as an SR860's.

- Commands answer nothing; a query answers its value in base units: V rms, Hz, degrees.
- Commands on one line are separated by `;`, and the queries among them answer on one line, separated by `;`.
- An unknown command sets bit 5 (CMD) of the standard event status byte. A bad value or a failed command sets bit 4 (EXE). Neither answers anything; `*ESR?` reads and clears the bits.

Arguments. Commands are case-insensitive. Numbers may carry a unit: NV UV MV V, HZ KHZ MHZ, UDEG MDEG DEG URAD MRAD RAD. Enumerations take their index or their name, either its capitals or in full, e.g. `ISRC A-B`, `OUTP? THETA`, `OUTP? TH`.

Per connection. The sensitivity and the status enables belong to the connection. The native protocol on port 5025 works alongside. Settings made on either port show on both, and the SR86x commands are ordinary configuration edits: they cancel another client's running acquisition.

Command	lockin
`FREQ(?) f`, `FREQINT(?) f`	oscillator 1's frequency (0 … 62.5 MHz; the realized value)
`FREQDET?`	demodulator 1's detection frequency (harmonic × f)
`FREQEXT?`	0 (no external reference)
`PHAS(?) p`	demodulator 1's reference phase, wrapped to ±180°
`HARM(?) i`	its harmonic, 1 … 1023 (an SR860: 1 … 99)
`RSRC(?) i`	0 (internal) only
`SLVL(?) v`	oscillator 1's amplitude on OUT1, V rms (OUT1 enabled by a set)
`SOFF(?) v`	OUT1's offset, V
`APHS`	auto-phase of demodulator 1 (`analysis.auto_phase`)
`IVMD(?) i`	0 (voltage) only
`ISRC(?) i`	0 = A (IN1), 1 = A−B (IN1 − IN2); beyond an SR860, 2 = B (IN2)
`ICPL(?) i`	0 = AC, 1 = DC (the input, or both for A−B)
`IRNG(?) i`	0 (the range is the board's LV/HV jumper)
`IGND(?) i`	1 (ground: the inputs are single-ended)
`ILVL?`	the input's peak over 8192 samples against its full scale: 0 (< 1 %), 1 (< 10 %), 2 (< 50 %), 3 (< 100 %), 4 (clipped)
`SCAL(?) i`	the sensitivity, 0 (1 V) … 27 (1 nV): used by `ASCL` and `LIAS?`; readings are in volts whatever it is
`ASCL`	the smallest sensitivity at least 1.1 R

Command	lockin
`OFLT(?) i`	demodulator 1's time constant from the SR860's table, 0 (1 µs) … 21 (30 ks); the query gives the nearest index
`OFSL(?) i`	its order: 6, 12, 18, 24 dB/oct = orders 1 … 4 (the query reports orders above 4 as 3)
`SYNC(?) i`	0 (no synchronous filter)
`ENBW?`	its equivalent noise bandwidth (Hz)
`OUTP? j`	one parameter: 0 X, 1 Y, 2 R (V rms), 3 θ (deg), 12 the reference phase, 13 the sine amplitude, 14 the DC level, 15 the internal frequency; 4–11 and 16 (aux inputs and outputs, noise, external frequency) read 0
`SNAP? j, k[, l]`	two or three parameters of one frame, comma-separated
`SNAPD?`	X, Y, R, θ of one frame
`*IDN?`	`RedPitaya,lockin SR86x subset,<build ID>, v<version>`
`*RST`	lockin's default configuration (outputs off); sensitivity 1 V
`*CLS`, `*OPC(?)`, `*TST?`, `*ESR? [j]`, `*ESE(?)`, `*STB? [j]`	IEEE 488.2 status: `*STB?` bit 3 (LIA) and 5 (ESB) from the enables
`LIAS? [j]`, `LIAE(?)`	the LIA status word: bit 4 (input clipped since the last read), bits 8–10 (X, Y, R beyond the sensitivity); read and cleared
`ERRS?`	0

Everything else (data capture, CH1/CH2 outputs, aux I/O, display, dual reference, chop, timebase) is not provided: such a command sets CMD and answers nothing. Example, with pymeasure:

```
from pymeasure.instruments.srs import SR860
lia = SR860("TCPIP::192.168.0.149::23::SOCKET")    # terminators: "\n"
lia.frequency = 10e3
lia.sine_voltage = 0.1                              # V rms on OUT1
lia.time_constant = 0.01
print(lia.x, lia.y, lia.magnitude, lia.theta)
```

The data stream

`DATA STREAM [interval]` (0.05 … 5 s, default 0.2) replies with one header line, `OK {"fields": [...], "format": "base64 of little-endian float64 rows", "interval_s", "rate", "demodulators", "queue_rows", "units", "session", "utc"}`, then one line every interval (also when empty): `DATA {"rows": n, "lost": rows dropped for this client so far, "data": base64}`. Each row has 15 float64 fields:

```
frame  t_s  x1 y1  x2 y2  x3 y3  x4 y4  locked  clipped  apply  flags  units_peak
```

`frame` counts frames since the FPGA's reset, `t_s` its end in seconds; X and Y are NaN for demodulators not in the stream; `locked` the PIDs' flags (bits 0 … 3); `clipped` 1 IN1, 2 IN2, 4 an output, during the frame; `apply` the apply count (modulo 256); `flags` 1: rows were lost before this one (the FIFO overflowed, the client fell behind, or the configuration changed), so the series is continuous only between flags. `STOP` ends it. The rows flow only while `stream.rate` is set. `DataStream(host).read()` returns `(rows, lost)`.

Command line

`python3 -m rpfpga.cli --host 192.168.0.149` (or on the board without `--host`), plus:

Option	
`--osc K=HZ[,DEG]`	oscillator K's frequency and phase (repeatable)
`--demod D=INPUT[,TAU[,ORDER]]`	demodulator D's input, time constant, order (repeatable)
`--amp O=K,V`	output O carries oscillator K with V peak, and is switched on (repeatable)
`--offset1 V`, `--offset2 V`	output offsets
`--units rms|peak`	
`--pid P JSON`	patch PID P (repeatable)
`--autophase D`	auto-phase demodulator D

It prints the description, the readings of every demodulator and the active PIDs, and the clip and drop counts

(`--json` for everything).

lockin registers

One AXI4-Lite block at **0x43C00000**, 256 KiB (18-bit addresses), 32-bit words (`model.py` has the same map). Use them only through `rpfpga.Overlay` (the driver): an access to an unconfigured FPGA hangs the board.

Range	Contents
0x00000–0x000FF	control, status, counters, apply time, capture and counter set-up, scratch, phase preloads
0x00100–0x001FF	configuration words in flip-flops (pending; live at an apply)
0x00400–0x007FF	engine words: the pending half of the double-banked coefficient RAM
0x00800–0x008FF	readings (the snapshot)
0x01000–0x01FFF	the FIFO (every read pops a word)
0x20000–0x3FFFF	the capture: channel c at 0x20000 + c·0x8000, word i at + 4i

Control and status (0x000–0x0FF)

Offset	Name	Access	Contents
0x000	ID	R	0x4C4B494E (”LKIN”)
0x004	VERSION	R	0x00010000 (1.0)
0x008	CTRL	W	strobes: [0] APPLY_NOW, [1] APPLY_AT, [2] CANCEL (a pending apply), [3] SNAP, [4] CLEAR (clip counts, dropped rows, late flag, counter done), [5] FIFO_CLEAR, [6] CAP_ARM, [7] CAP_TRIG (software trigger), [8] CAP_STOP, [9] COUNT_START, [15:12] PID_RESET (PID p at the next frame)
0x00C	STATUS	R	[0] apply pending, [1] late, [2] snapshot pending, [3] FIFO overflow, [5:4] PROTECT, [6] counter busy, [7] counter done, [8] DIO3_P level, [15:12] PIDs locked, [19:16] PIDs saturated, [23:20] PIDs held
0x010, 0x014	COUNT_LO, COUNT_HI	R	the sample counter (reading LO latches HI)
0x018	LATENCY	R	14: the output’s samples from the counter to the core’s output stream
0x01C	CONSTANTS	R	[7:0] L_FRAMES (4), [15:8] log2 frame (7)
0x020, 0x024	APPLY_TIME_LO/HI	RW	APPLY_AT’s sample (rounded up to a frame boundary)
0x028, 0x02C	LAST_APPLY_LO/HI	R	the sample of the last apply
0x030	APPLY_COUNT	R	applies since reset (the FIFO rows carry its low 8 bits)
0x034	LATE_COUNT	R	APPLY_AT requests that were late (applied at the next boundary)
0x040–0x04C	CLIPS_IN1, IN2, OUT1, OUT2	R	samples at the converter’s limits (inputs) or clamped/saturated (outputs)
0x050	FIFO_LEVEL	R	words in the FIFO (0 … 8192)
0x054	FIFO_DROPPED	R	rows dropped (no room)
0x058	PROTECT	RW1S	[1:0]: OUT1/OUT2’s CLAMP locked until the FPGA is reloaded (set only)
0x05C	LOADS	RW1S	[3:0]: preload oscillator k’s accumulator from NCO_P at the next apply
0x060	CAP_CFG0	RW	sources: [5:0] channel 0 … [23:18] channel 3 (codes below), [28:24] k (rate 125 MHz / 2^k, ≤ 24)
0x064	CAP_CFG1	RW	[12:0] pretrigger words, [18:16] trigger: 0 immediate, 1 DIO3_P rising, 2 falling, 3 IN1 up, 4 IN1 down, 5 IN2 up, 6 IN2 down (software: CAP_TRIG)
0x068	CAP_LEVEL	RW	Schmitt levels for IN1/IN2 triggers: [15:0] high, [31:16] low (signed 14-bit codes)
0x06C	CAP_STATUS	R	[1:0] 0 idle, 1 armed (before the trigger), 2 triggered (recording), 3 done; [4] done; [28:16] the trigger’s word
0x070, 0x074	CAP_TRIG_LO/HI	R	the trigger’s sample

Offset	Name	Access	Contents
0x078, 0x07C	CAP_FIRST_LO/HI	R	the first interval's start sample
0x080	CNT_CFG	RW	frequency counter: [0] input (0 IN1, 1 IN2), [12:8] gate 2^g samples
0x084	CNT_LEVEL	RW	its Schmitt levels (as CAP_LEVEL)
0x088	CNT_COUNT	R	rising edges in the gate
0x08C–0x098	CNT_FIRST, CNT_LAST	R	the first and last edge's samples (64-bit pairs)
0x09C, 0x0A0	CNT_START	R	the gate's start sample
0x0B0–0x0BC	SCRATCH0–3	RW	kept for the driver: [0] flags (1 units peak, 2 coherent), [1], [2] t0
0x0C0 + 8k	NCO_P[k] LO/HI	RW	oscillator k's phase preload (48 bits)

Configuration words in flip-flops (0x100–0x1C8)

Pending words, live at an apply; read back from a mirror.

Offset	Name	Contents
0x100 + 16k	F_LO, F_HI (+4), POFF (+8)	oscillator k: frequency word (48 bits: 125 MHz / 2^{48} steps), phase offset (32 bits, $\times 2^{16}$ in the 48-bit phase)
0x140 + 8d	DCFG	demodulator d: [1:0] input (0 IN1, 1 IN2, 2 IN1 − IN2), [3:2] oscillator A, [5:4] oscillator B, [7:6] sideband n (0: 0, 1: +1, 3: −1), [8] square reference, [25:16] harmonic m
0x144 + 8d	POW	demodulator d's reference phase offset (24 bits, 2^{-24} cycle)
0x180 + 32o	A0 … A3 (+0 … +12)	output o: amplitude words of oscillators 0 … 3 (signed 25-bit, full scale 2^{24})
0x190 + 32o	DC	output o's DC word (signed, 2^{-16} DAC step)
0x194 + 32o	OCFG	[0] output enabled, [1] dither
0x198 + 32o	CLAMP	[13:0] low, [29:16] high (signed 14-bit DAC codes); write-protected by PROTECT
0x1C0, 0x1C4	OFFSET1, OFFSET2	the inputs' offsets (signed 16-bit, half converter steps)
0x1C8	ICFG	[1:0] AC coupling of IN1, IN2

Engine words (0x400–0x7FF)

Writes go to the pending half of a double-banked block RAM, which an apply makes active (and then refills from it). Reads return the pending half.

Offset	Name	Contents
0x400 + 4d	FILT[d]	demodulator d's filter: [16:0] m, [22:17] s (α = m·2^−(17+s)), [27:24] order 0 … 8
0x410 + 4i	AC[i]	input i's AC-coupling section (α as FILT; order ignored)
0x440 + 64p	PID p, 16 words	see below
0x540 + 8o	MSRC	output o's monitor: [3:0] 0 off, 1 X, 2 Y, 3 R, 4 θ, 5 dθ, 6 PID output, 7 PID error; [5:4] demodulator or PID
0x544 + 8o	MGAIN	its gain word (DC words per register unit)
0x550	FIFO	[4:0] j (a row every 2^j frames), [19:16] the demodulators in each row, [20] enable

Gain words: [17:0] signed mantissa m, [29:24] e: gain = m·2^(min(e, 48) − 48).

PID words (offsets from 0x440 + 64p, ×4):

Word	Name	Contents
0	PSRC	[2:0] quantity (0 X, 1 Y, 2 R, 3 θ: the error is signed32(SP − θ), within ±½ cycle; 4 dθ, 5 the unwrapped phase), [5:4] demodulator, [8] enable, [9] hold, [10] hold when R < RMIN
1, 2	SP_LO, SP_HI	setpoint (signed 48-bit, the quantity's register units)

Word	Name	Contents
3, 4, 5	KP, KI, KD	gain words (KI includes 2^{16}: the integrator's fraction bits)
6	DCFG	[4:0] kdf (D low-pass 2^−kdf per frame; 0 none), [12:8] lock window 2^w frames
7, 8, 9, 10	LO, HI	output limits (signed 48-bit, the target's register units)
11	TARGET	[2:0] 0 none, 1 frequency, 2 phase, 3 amplitude, 4 DC; [5:4] oscillator (frequency, phase) or output ([4]: amplitude, DC); [7:6] the amplitude's oscillator
12, 13	RMIN_LO/HI	R below which the PID holds (with PSRC[10])
14	TOL	lock tolerance (|e| ≤ TOL, unsigned)

Register units: X, Y: filter output / 2^{16} (X = (S/4)·a·cos θ for a sine of a half converter steps, S = $2^{17} - 1$); R: the same times the CORDIC gain K = 1.6468; θ: 2^{-32} cycle; dθ: 2^{-32} cycle per frame. Targets: frequency 125 MHz / 2^{48}; phase 2^{-48} cycle; amplitude the A words; DC the DC words.

Readings (0x800–0x8FF)

The snapshot of one frame (CTRL SNAP; valid when STATUS[2] clears).

Offset	Contents
0x800 + 32d	demodulator d: X, Y, R (float32 of the register units: X, Y with 16 fraction bits), θ (u32), dθ (s32), unwrapped θ lo (u32), hi (s16)
0x880 + 16p	PID p: output (float32, the target's units), error (float32), integrator (float32, 16 fraction bits), flags [0] locked, [1] saturated, [2] held
0x8C0 + 8k	oscillator k's live frequency word (lo, hi: 48 bits, with the PIDs' dF)
0x8E0, 0x8E4	the inputs' DC estimates (float32, half converter steps)
0x8F0, 0x8F4	the frame number (lo, hi)

FIFO (0x1000–0x1FFF)

Every read pops one word (any address in the window). A row: word 0 the frame number's low 32 bits; word 1: [15:0] its high bits, [23:16] the apply count, [27:24] the PIDs' locked flags, [30:28] the frame's clip flags (IN1, IN2, an output); then X, Y (float32) of each demodulator in FIFO[19:16], in order.

Capture (0x20000–0x3FFFF)

Channel c's word i at 0x20000 + 0x8000·c + 4i, in the order recorded (the trigger's word is CAP_STATUS[28:16]). Source codes (6 bits): 0 off, 1 IN1, 2 IN2, 3 OUT1, 4 OUT2 (codes), 5 + d demodulator d's input x; frame-rate: 9 + d X, 13 + d Y, 17 + d R, 21 + d θ, 25 + d dθ, 29 + p PID output, 33 + p PID error (floats, except θ and dθ). Full-rate words are the sum of 2^k samples for k ≤ 16, else the sum / 2^(k − 16).

lockin verification

Dated records, newest first. Board: the Red Pitaya STEMlab 125-14 used for every design here (PYNQ, 192.168.0.149), board calibration "loopback + dmm" of 2026-10-04 (no response tables; the inputs' loop delays added on 2026-10-25, evening), LV jumpers, OUT1 → IN1 and OUT2 → IN2 through 50 cm RG316 cables with a T and a 50 Ω terminator at each input.

2026-10-27 (afternoon): the consolidated software on the board; the manuals

Software only (bitstream 0x2ae6d985), after the consolidation of lockin's and feedback's shared software into `rpfpga.demod` and `rpfpga.rows`.

Board (`make test-hw`): **38 of 38**. The loopback response 0.9604 (1 kHz) … 1.0564 (10 MHz) of the set amplitude; the phase slope shows 2.29 ns beyond the stored loop delay (the cables); input noise 1201, 1058, 83, 82 nV/√Hz at 10 kHz, 100 kHz, 1 MHz, 10 MHz; the noise density at 1 MHz 79.5 (zoom spectrum) and 81.3 nV/√Hz (X, Y); `auto_tau` 2.69 µs for 10^{-4} of R (R's scatter $1.8 \cdot 10^{-4}$); stream through the service without loss up to 7629 rows/s.

Found while recording the user's manual's data, and fixed:

- **A sweep's deadline left out the applies.** Every step of a sweep applies a configuration (about 97 ms on the board), but `sweep_duration` counted only the settling and the readings: an 81-point sweep at τ = 1 ms took about 11 s against a deadline of 12.5 s and was cancelled ("Acquisition deadline expired"). `sweep_duration` now adds 0.15 s per step (`rpfpga.demod.APPLY_S`), and the clients' reply timeouts use it; the same sweep then took 16 s. Test: `test_sweep_duration_counts_the_applies`.
- **A slow capture timed out before it could finish.** The capture's deadline was its `timeout_s` (10 s) from arming, whatever its own length: an immediate capture of 8192 words at a low rate (the zoom spectrum and the noise density at references below about 1 kHz) could never complete ("No trigger within 10 s"). The deadline (and the acquisition's) now adds the record's duration (lockin's and feedback's `capture`, feedback's `capture_step`).

The manuals (`docs/manual/`): the theory manual of lockin and feedback (`theory_manual.pdf`, its figures and numbers from `tools/theory_figures.py`, regenerated from scratch with identical values) and lockin's user's manual (`user_manual.pdf`, its measured figures recorded on this board with `tools/record_manual_data.py`). Working out the theory checked the designs' loops against the exact sampled models: `lock_pll`'s pull-in range simulated sample by sample (8, 7, 3.25 bandwidths at 100 Hz, 1 kHz, 10 kHz) agrees with the board's records (7–8, 7–8, 3–5) and with a 90° lag criterion; the margins of `lock_pll`'s loop are 60°, 48° and 36° at 1, 10 and 19 kHz (the last the integral gain's limit); a pure 232 ns delay predicts the board's measured fast loop (388 kHz, 63°). `tests/test_manual.py` runs the user's manual's examples on the model, checks every source excerpt, regenerates the user figures from the recordings and builds both PDFs.

The user's manual's recordings (this board, 2026-10-27, OUT1 → IN1, OUT2 → IN2):

- the `loopback` preset: X 137.559 mV rms after auto-phase, Y −6 µV, 4.4 µV rms scatter at 7.8 Hz ENBW;
- the loopback response swept from 1 kHz to 60 MHz (81 points, 16 s): +10.0 % at 10 MHz against 1 kHz;
- input noise (outputs off, `noise_density`): 0.89 µV/√Hz at 1 kHz, 0.85 at 100 kHz, 84 nV/√Hz at 1 MHz, rising to 1.9 µV/√Hz at 100 Hz, a bump near 30 kHz (1.4 µV/√Hz);
- a 1 mV tone 300 Hz off a 10 kHz reference: 0.7257 mV rms in the zoom spectrum, 0.7472 mV rms from a fit to the captured input (the window's scalloping);
- harmonics of a 0.8 V, 10 kHz sine through the loopback: at most −64 dBc;
- a 100 Hz PLL relocking after its set frequency jumped by one and three bandwidths: 14.4 and 19.0 ms until $|\theta| < 5°$;
- the amplitude loop (i = 1000 /s): 1.52 ms against 1.39 ms from the plant's measured gain;
- 5 minutes of a 1 MHz loopback: the Allan deviation of R at least $5.7 \cdot 10^{-6}$ (at 0.27 s);
- `auto_tau` for 10^{-3} of a 2 mV signal at 10 kHz: τ 25.1 ms (order 4), R's scatter $0.88 \cdot 10^{-3}$ over 20 s;
- the counter at 1.234568 MHz: the scatter of repeated counts at the 8 ns/√6 timing limit from 1 ms to 1 s gates.

2026-10-27: an SR86x-compatible port; spectra and statistics of X and Y

Software only (bitstream 0x2ae6d985).

The SR86x port (`compat.py`, the service's port 23; docs/PROTOCOL.md "SR86x port")

- **Model tests:** settings onto demodulator 1, oscillator 1 and OUT1; units and enumerations; several commands per line; readings and SNAP? of one frame; APHS; ILVL?; the status registers.
- **Network:** a real connection to the server's second port.
- **From a PC on the board's network:** `*IDN?` → `RedPitaya,lockin SR86x subset,0x2ae6d985,v1.0`, `FREQ 12.345 KHZ;FREQ?` → 12345, `OFLT 6;OFSL 3;OFLT?;OFSL?;ENBW?` → 6;3;78.12 Hz, `SLVL 50 MV;SLVL?` → 0.05000000091, `BOGUS` → no reply and `*ESR?` → 32, `SNAP? X, Y, R` → three values of one frame.
- **pymeasure 0.16's `SR860` class** (pyvisa-py, `TCPIP::…::SOCKET`, on the model's port) sets and reads the frequency, sine amplitude, time constant, slope, input, X, Y, R, θ, snap, harmonic, phase and noise bandwidth. pymeasure itself caps the frequency at 500 kHz, and its `id` is not implemented for this class.

Output analyses (docs/THEORY.md "Spectra and statistics of X and Y").

- **Model:** a tone 8 bins above the reference is at +15.26 kHz in the zoom spectrum, with its amplitude within 1 %.
- **Synthetic white noise through the exact filter:** the spectral and the X, Y noise densities are within 10 %, and `auto_tau`'s τ follows the formula.
- **Allan deviation:** 1/√m for white noise, Dτ/√2 for a drift.
- `RunningStats` agrees with Python's statistics.
- `LockinClient.record` collects the stream.

Also: the capture takes a word count (a short record), and an apply that starts a stream clears the FIFO, so an earlier stream's rows cannot arrive first (found by feedback's board test; `FifoTests`). Tests: lockin 56 (1 skipped), common 227 (5 skipped).

Board (`make test-hw`): **38 of 38** (31 before, plus 7 new checks):

Check	Result
noise density at 1 MHz (IN1 terminated, τ 10 µs, order 4): zoom spectrum's median (122 bins) / X and Y's	82.8 / 81.6 nV/√Hz = 1.014
zoom spectrum: OUT1's tone 50 bins (2.98 kHz) above the reference	at its bin; amplitude 0.99998 of the captured input's
`auto_tau` for 10^{-4} of R (0.14 V rms at 1 MHz)	τ 3.1 µs, order 4; R's scatter over 64 readings $1.25 \cdot 10^{-4}$
SR86x port on the board: `*IDN?`, `FREQ?`, `SNAP? X, Y` against the driver	lockin's subset; equal; R 0.99993

2026-10-26 (late night): the engine's memories in block RAM (simulation, build, board)

feedback 2.0's area rework back-ported; the model is unchanged.

- **Engine**: the taps (16 × 48 bits), the results (X, Y, R, θ, dθ and the unwrapped phase by {quantity, frame parity, demodulator}) and the PIDs' state (integrator, D filter, previous error, lock count, output) moved from LUT memories and registers to block RAM, one read port each, shared by the schedule (docs/THEORY.md). The CORDIC runs two clocks later (done in slot 11 + 27 d); demodulators 1–3 read their taps again in the two clocks before their normalization; the R, θ, dθ and unwrapped-phase floats come from the write-back's registers (slots 14, 41, 68, 95). The PIDs' error reading was a copy of the previous error (`pid_e`): removed.
- **Capture**: one shifter for the words of intervals of 2^17 samples and more (a channel per clock, written four clocks later). The benches cannot reach such intervals; a throwaway bench against the previous capture (k = 14, 17, 20, four channels) agrees word for word.
- **Timing**: the first build with these failed (WNS −0.473 ns, 15 endpoints, all on the capture's trigger path: mode → level comparison → accept → post-trigger count), so feedback's precomputed trigger decodes were ported too. That found a bug in feedback's version (since its 1.0): the level comparison, registered in the clock before, used the previous capture's levels and source in an ARM's clock, so a level trigger could fire on its first sample. Fixed here and in feedback. `tb_system` (both designs) now arms a level capture whose first sample would trigger with the previous levels: the model triggers 69 samples later; feedback's committed RTL triggered on the ARM's sample.
- **Vectors**: the engine bench's monitors now also read R and a PID's error, and a PID runs on Y.

All five benches pass (`tb_system`: 38 400 samples, 989 reads); lockin tests: 48 passed, 1 skipped.

Build ID	0x2ae6d985 (2026-10-26)
LUTs	11913 / 17600 (67.7 %), 365 as memory (1.0: 12902, 1258)
Flip-flops	14713 / 35200 (41.8 %)
Slices	4211 / 4400 (95.7 %)
Block RAM tiles	54 / 60 (51 RAMB36 + 6 RAMB18)
DSP48E1	56 / 80
Timing	WNS +0.088 ns, TNS 0, WHS +0.019 ns (the worst paths: the IIR's combs)
DRC	warnings only, as for 0x05dbcb82

`common/tests/verify_build.py --design lockin`: all 8 checks pass; the recipe's build reproduced build ID 0x2ae6d985.

Board: `make test-remote` (LKIN 1.0, build 0x2ae6d985, loaded through the driver, power-up state verified), the service reinstalled with lockin, `make test-hw`: **31 of 31 checks pass**. As before: R against the captured IN1 within 0.01 % from 0.8 V down to 0.01 V, harmonic 3 and the 137 kHz sideband, the square reference, IN1 − IN2 (1.501), the RC time constant (0.99994 τ), captured R against the readings, coherent phase after a frequency excursion, the counter (−0.02 ppm), the PLL (−0.4 mHz, −0.0002°), the amplitude PID, the limits, the clip count, the stream lossless at 7629 rows/s, SAFE.

2026-10-25 (night): faster transactions, PLL frequency acquisition

Software only (the bitstream is build 0xa3ee4010). Measured on the board with the new `rpfpga` and `lockin` packages run from scratch copies beside the installed ones.

Transactions (`common/`): every outermost transaction re-verifies that the FPGA holds the design. The checks are unchanged, but the record, the boot ID, PYNQ's state file and the board's coordination metadata are parsed again only when their files change (`files.read_cached`: a stat stamp, and git's "racily clean" rule against coarse file times), the level shifters are read through a mapping kept open, and the lock file stays open between transactions (re-opened if it was replaced). Common tests: 213, including new ones for the cache (changes found, recent files not trusted) and the lock (another process excluded, a removed lock file re-created).

Per call, on the board	Before	After
Load check (`record.verified`)	2.93 ms	0.72 ms
Lock enter and exit	0.60 ms	0.17 ms
Empty transaction	4.49 ms	1.25 ms
`readings()`	6.66 ms	3.39 ms

PLL acquisition (`analysis.lock_pll(..., acquire=range)`): the PID first locks dθ/dt (a type-1 frequency loop, `fll_settings`), then becomes the PLL with its limits centred on the frequency found (no reset: its integrator keeps the correction). Model test: 15 bandwidths off (150 kHz at 10 kHz) locks; a search range too small and a missing signal (with `hold_below`) raise. lockin tests: 48. On the board (OUT2: 1 MHz, 0.2 V into IN2; oscillator 3 started df above; `acquire` = 1.5 df; 2 trials each, all locked):

bw	df	Frequency stage (the call)	Lock flag after the switch	Oscillator − input (one reading)
100 Hz	10, 30, 100 bw	0.30–0.31 s	0.11–0.13 s (its lock time 10/bw = 0.1 s)	≤ 2 mHz
1 kHz	10, 30, 100 bw	0.28–0.35 s	0.031 s	≤ 0.09 Hz
10 kHz	10, 30 bw (to 300 kHz)	0.34–0.35 s	0.031 s	≤ 3.9 Hz (the loop's jitter per reading)

Phase errors ≤ 0.02° after locking. Without `acquire`, the same loops pull in from 7 bandwidths (100 Hz, 1 kHz) and 3 (10 kHz) (the evening record).

Installed on 2026-10-26 (`make test-remote`, `make install-service`): `make test-hw` 31/31, twice. The stream through the service at 15 259 rows/s gave 8052 and 7733 rows/s (9009–9118 in the last runs with the old software, 7869 in one), and in the second run it lost 787 rows at 7629 rows/s (none in the other six runs since 1.0). The drain loop itself is not slower: the worker's `step` run alone at 15 259 rows/s with the service stopped (10 s each, alternating) drained 9713 and 9715 rows/s with the new framework (90 ms per step) and 9444 and 9502 with the committed one (94 ms per step). The scatter is in the service's path to the client.

2026-10-25 (evening): the PLL's phase error, characterization

Fix. The PID input "phase" used the unwrapped phase, which is never reset: a PLL enabled after its demodulator had seen an unlocked signal for a while started that many whole cycles from its setpoint and ran into its limit (found when the notebook's PLL cell, run headless on the board, railed at +1 kHz). "phase" is now θ with the error wrapped to ±180° (a signed 32-bit difference); the new quantity "phase_unwrapped" (code 5) keeps the old behavior, and the driver wraps a "phase" setpoint to ±180°. Changed: the model, the engine (`lk_engine.v`), the driver, the vectors, the tests (a PLL locking after its demodulator unwound many cycles at a 50 kHz beat; the wrapped error) and the docs. The previous engine fails the regenerated `tb_engine` vectors (140 mismatches); the new one passes them.

Offline and simulation: `make test`: 47 tests; `make sim`: all five benches PASS.

Build (`make`):

Build ID	0xa3ee4010 (2026-10-25 19:42)
LUTs	12948 / 17600 (73.6 %), 1280 as memory
Flip-flops	14283 / 35200 (40.6 %)
Slices	4278 / 4400 (97.2 %)
Block RAM tiles	51 / 60 (48 RAMB36 + 6 RAMB18)

DSP48E1	56 / 80
Timing	WNS +0.100 ns, TNS 0, WHS +0.021 ns (after post-route phys_opt; −0.060 ns before it)
DRC	warnings only, as for 0x05dbcb82

Board (user-approved upload: `make test-remote`, `make install-service`; `make test-hw`: **PASS, 31 checks**). The PLL check now first lets its demodulator run on the unlocked input (it gathered 181 cycles of unwrapped phase), then locks: the oscillator within 0.8 mHz of the input (mean), phase error 0.0004°. τ = 1.00005 of the set 0.1 s. The loop-delay check adds back the stored input delays (it reads only the cable's delay once they are stored).

Characterization (with lockin's own demodulators and captures, through the driver on the board; host scripts not kept in the repository):

- **Converter loop delays**, from the phase of OUT1 → IN1 and OUT2 → IN2 at the 40 multiples of the frame rate from 0.98 to 39 MHz (τ = 2 µs, order 4): a pure delay from 1 to 10 MHz, 122.413 ns (IN1) and 122.326 ns (IN2), intercept +0.3°, repeatable to 0.01 ns; above 10 MHz the group delay grows from 122.1 ns (at 1 MHz) to about 128.8 ns at 36 MHz and the phase falls 39° below the pure delay at 39 MHz (the analog anti-aliasing and reconstruction filters); the amplitude peaks at +8 % near 19 MHz. Stored in the board calibration (`set_input_delay`, LV) after removing the cable and T (2.5 ns) and, for IN2, OUT2's calibrated skew (−0.165 ns): **119.913 ns (IN1), 119.991 ns (IN2)**. Afterwards the loopbacks read −0.82° at 1 MHz (IN1) and −0.94° at 1.3 MHz (IN2), the cable's delay within 0.3°. The inputs' skew (IN2 − IN1) is 0.08 ns; the 0.11° at 1 MHz in the morning record was crosstalk (see there).
- **Analog loopback response** (R against the set 0.2 V, 74 frequencies from 10 Hz to 62 MHz): flat within 10^{-4} below 1 kHz at 0.96031 (IN1) and 0.96223 (IN2): the DC calibration of 2026-10-04 does not match this cabling by 4 %. A pole-zero pair (zero 20.6 kHz, pole 22.0 kHz, to 1.026) gives the step of the morning record and +1.75° of phase at 20 kHz; then 1.022 at 1 MHz, 1.056 at 10 MHz, 1.08 near 19 MHz, 0.9995 at 50 MHz, 0.74 at 55 MHz, 0.34 at 60 MHz. A loopback does not separate the output's and the input's share, so no response tables were stored.
- **Input noise below 10 kHz** (IN1 and IN2, LV, terminated through the loop cables, outputs off; X and Y of captured readings):

Frequency	IN1	IN2
37 Hz	5.24 µV/√Hz	5.11 µV/√Hz
113 Hz	2.17	2.09
317 Hz	1.33	1.32
1.03 kHz	1.19	1.12
3.11 kHz	1.10	1.07
10.0 kHz	1.28	1.30
31.0 kHz	1.61	1.58
100 kHz	0.65	0.63

  Below 300 Hz the excess falls about as f^−0.9 in density and crosses the floor of 1.1 µV/√Hz near 200 Hz; the density rises again from 3 to 31 kHz before falling to 0.65 µV/√Hz at 100 kHz. Mains at 50 Hz is about 25 µV (−92 dBV).
- **Amplitude loop** (PID 2 holds R of 50 kHz on IN1 at 50 mV through OUT1's amplitude, I only; demodulator τ = 20 µs, order 2; a PID reset during a capture): exponential relocking with time constants 4.597, 1.346, 0.417 ms for I = 300, 1000, 3000 V/V/s (single-pole predictions from the loop gain 4.618, 1.386, 0.462 ms); R settles at 50.00 mV.
- **Notebook** (`control/lockin.ipynb`, every cell executed headless on the board with nbconvert): before the fix the PLL cell railed (above); afterwards two runs locked. The `harmonics` preset reads H2 −72 dBc, H3 −56 dBc, H4 −79 dBc of the 10 kHz loopback tone.
- **Frequency counter** (10 kHz on IN2, 40 counts per gate): standard deviation 0.18 Hz with a 10 ms gate, 0.01 Hz with 0.1 s; the means agree with the set frequency.
- **Readings' cost** (Python on the board, after the change): 6.9 ms per reading: 4.6 ms the framework's check that the FPGA is loaded (once per transaction), 0.7 ms the burst read of the snapshot, about 1 ms the decoding (2.2 ms before it used `struct`).
- **Drift** (2 h at 1 reading/s, 4 demodulators at τ = 10 ms, order 4: 1 MHz on IN1 and 1.3 MHz on IN2 at 0.2

V, 10 kHz on IN1 and 10.7 MHz on IN2 at 0.1 V; the Zynq's die temperature 71.4–73.3 °C, 10-minute means 72.0–72.7 °C):

Channel	R: sd, p-p	R: Allan deviation 1 s, 10 s, 100 s, 1000 s	θ: sd, p-p	θ against temperature
1 MHz, IN1	56 ppm, 296 ppm	10.5, 10.7, 21, 15 ppm	0.0014°, 0.008°	−0.0055 °/K (r = −0.83)
1.3 MHz, IN2	58 ppm, 305 ppm	10.5, 10.8, 20, 18 ppm	0.0020°, 0.010°	−0.0077 °/K (r = −0.85)
10 kHz, IN1	59 ppm, 418 ppm	34, 15, 19, 11 ppm	0.0020°, 0.020°	none
10.7 MHz, IN2	57 ppm, 279 ppm	10.9, 10.7, 21, 18 ppm	0.014°, 0.080°	−0.059 °/K (r = −0.89)

The amplitude wanders together on all four channels (both outputs, both inputs): the ratio of the 1 MHz IN1 and 1.3 MHz IN2 readings has a standard deviation of 18.5 ppm and an Allan deviation of 3.8 ppm at 1 s, so most of the 10 ppm at 1 s and the slow wander are common to the dual converters (their references or supplies; not separable in a loopback). R falls with temperature by about 100–170 ppm/K (correlation −0.4 … −0.6 with the die temperature, only a proxy for the analog parts'). The phases' temperature coefficients are one delay of about 15 ps/K at every frequency (invisible at 10 kHz). The 1 Hz series had 17 gaps longer than 1.5 s (the longest 6 s).

- **PLL lock time and pull-in range** (`analysis.pll_settings` and `pll_filter`: a type-2 loop, demodulator filter order 2 at $\tau = 1/(2\pi \cdot 8 \cdot \mathrm{bw})$; limits ±50 bw so that they do not bound the pull-in). OUT2 sends 1 MHz at 0.2 V to IN2; PID 1 locks oscillator 3, set df above the input. Once locked (the PID output at −df), the PID is reset (integrator and output cleared): the oscillator jumps back to df off and relocks. OUT1 monitors the PID output into IN1, so the capture of θ and the PID output triggers on the reset's step (`in1_up`, 1024 samples of pretrigger); cycle slips from the readings' unwrapped phase before and after. Time from the reset until |θ| stays below 5°, 3 trials each (identical to the capture's resolution: 131, 16, 2 µs):

df	bw 100 Hz	bw 1 kHz	bw 10 kHz
0.3 bw	9.4 ms	0.93 ms	0.090 ms
1 bw	14.5 ms	1.46 ms	0.145 ms
2 bw	17.4 ms	1.75 ms	0.176 ms
3 bw	19.1 ms	1.92 ms	0.195 ms
5 bw	40 ms, 8 slips	4.5 ms, 10 slips	no lock
7 bw	137 ms, 66 slips	24 ms, 133 slips	no lock
10, 15, 20, 30 bw	no lock	no lock	no lock

Lock times scale as 1/bw (about 1.5/bw at an offset of one bandwidth); to $|\theta| < 1°$ they are 0.7/bw longer. The pull-in range is set by the demodulator filter's phase lag at the beat frequency: 7–8 bw at 100 Hz and 1 kHz, 3–5 bw at 10 kHz, where the loop's delay adds to the lag. A float model of the same loop locks up to 8 bw (100 Hz, 1 kHz) and 3 bw (10 kHz) and fails beyond. Outside the range the loop does not pull in: on the board it drifted away from the input to its frequency limit within 3 s (bandwidths of 100 Hz and 1 kHz) or wandered (10 kHz). `lock_pll`'s counter seed starts the oscillator within a fraction of a hertz of the input, far inside the range.

- **Relocking from a limit** (after the commit above, `pll_settings`' default limits went from ±10 to ±5 bw and the `pll` preset's from ±1 kHz to ±500 Hz). Locked with the counter's seed, the input (OUT2) moved 20 bw away for 3 s, then back; 2 trials each:

bw	limits ±5 bw	limits ±10 bw
100 Hz	parked at −5 bw; relocked (lock flag in 0.17 s)	parked at −10 bw; no relock in 3 s
1 kHz	parked at −5 bw; relocked (0.03 s)	parked at −10 bw; no relock in 3 s
10 kHz	wandered within ±3 bw; relocked (0.03 s)	wandered within ±3 bw; relocked (0.03 s)

At 10 kHz ±3 bw limits relocked as well.

Decisions left to the user: recalibrating the DC loop gain for this cabling (4 %; needs the DMM); response tables (need the output's and the input's share separately); whether the framework's per-transaction check (4.6 ms per

reading) may be relaxed.

2026-10-25: lockin 1.0

Simulation (`make sim`, Vivado 2025.1 XSim, against the bit-exact model):

Bench	Result
`tb_sysid` (common)	PASS
`tb_ref`	PASS: 6000 reference phases and sines (harmonics, sidebands, offsets, square)
`tb_datapath`	PASS: 6142 samples of the output codes and 47 frames of the CIC's I2 words
`tb_engine`	PASS: 400 frame periods (targets, DC words and PID flags at slot 126), 888 FIFO words, 2 snapshots of 64 words; a coefficient bank changing 4 times, PID resets, clip flags
`tb_system`	PASS: `lk_core` through real AXI transactions: 38 400 samples of both output codes and 986 register, snapshot, FIFO and capture reads; three timed applies (APPLY_AT) with phase preloads and PID resets, two snapshots, two captures (a DIO3_P edge with pretrigger, immediate), the frequency counter (APPLY_NOW, used by every apply in continuous mode, is exercised by the board test)

Offline tests (`make test`): 45 tests (model against float theory: the sine table and ROM, exact harmonic/sideband phases, demodulation $X = (S/4)\,a \cos\theta$, the CIC's gain, τ round trips within $2{\cdot}10^{-5}$ from 1 µs to 10^4 s, step responses, CORDIC accuracy, floats, gains, the PID, outputs and dither statistics; the driver on the model in a loopback: readings, harmonic/sideband/square/difference, the phase convention and the connector plane, the input delay measurement, derive round trip, a PID, a PLL locking, FIFO rows, captures, the counter, the calibration hooks, protected limits, coherent preloads, presets; protocol, client and server, the stream hub and worker, numpy against plain decoding, the command line, analysis and the GUI), plus the common tests with the new calibration fields (inputs' response and loop delay).

Build (`make`): the first build met the LUT budget (75.4 %) but failed timing with WNS −2.676 ns (6724 failing endpoints, 95 % of the slices: wide single-clock arithmetic in the engine, slot decodes on the shared counter, a 64-bit apply compare with a large fanout, reset and AXI write fanout). After pipelining (AGENTS.md, "Timing closure recipe"; the model unchanged, every bench passing):

Build ID	0x05dbcb82 (2026-10-25 16:10)
LUTs	12902 / 17600 (73.3 %), 1258 as memory
Flip-flops	14284 / 35200 (40.6 %)
Slices	4256 / 4400 (96.7 %)
Block RAM tiles	51 / 60 (48 RAMB36 + 6 RAMB18)
DSP48E1	56 / 80
Timing	WNS +0.084 ns, TNS 0, WHS +0.014 ns (post-route phys_opt)
DRC	warnings only: DSP pipelining advice, REQP-1723/1724 (the outputs' modulo-2^{25} pre-adder, intended), one unrouted-load net

`common/tests/verify_build.py --design lockin`: all 8 checks pass (recipe build, unchanged-build reuse, missing-export recovery, vendor HDL edit, design RTL edit, failed build blocks the upload, clean, regeneration from the saved project); the recipe's build reproduced build ID 0x05dbcb82.

Board (`make test-remote`: loaded through the driver, sysid LKIN 1.0, build 0x05dbcb82, power-up state verified, preset `off`; `make install-service`; `make test-hw`: **PASS, 31 loopback checks**):

Check	Measured	Expected
R·√2 against the amplitude of the same captured IN1 samples (least-squares fit), 1 kHz, 10 kHz, 100 kHz, 1 MHz, 3 MHz, 10 MHz	0.99984, 1.00003, 1.00005, 0.99989, 1.00002, 1.00004	1 ± 0.005
Converter loop delay OUT1 → IN1 with the cable (phase slope 1–3 MHz)	122.17 ns (15.27 samples)	50 … 250 ns
Phase at 10 MHz from that delay	−1.04°	0 ± 5°
IN1/IN2 balance (OUT1 → IN1 against OUT2 → IN2), 100 kHz	0.99907	1 ± 0.03
Lock-in against the captured input at 1 MHz: 0.8 V, 0.1 V, 10 mV, 1 mV	1.00003, 0.99997, 0.99965, 0.99696	1 ± 0.003 (1 mV: ± 0.03)
Harmonic 3 of 100 kHz against the 300 kHz tone	1.00000	1 ± 0.002

Check	Measured	Expected
Sideband 100 + 37 kHz (0.1 V) against 300 kHz (0.2 V)	0.49999	0.5 ± 0.005
Square reference against the sine reference	0.99944	1 ± 0.003
IN1 − IN2 with OUT2 = −OUT1/2	1.50089	1.5 ± 0.02
RC time constant 0.1 s (order 1) from a step's readings	0.99977 τ	1 ± 0.01
Captured R (frame rate) against the readings	1.00006	1 ± 0.002
Coherent phases: back after a frequency excursion	+0.0009°	0 ± 0.2°
Frequency counter on IN2, 123456.789 Hz, 0.1 s gate	−0.021 ppm	0 ± 1 ppm
PLL (200 Hz, seeded by the counter): oscillator 3 against IN2, mean of 64 readings	−3.0 mHz	0 ± 10 mHz
PLL phase error (mean)	−0.0011°	0 ± 0.2°
PID: R held by the amplitude, setpoint 50 mV	50.015 mV	± 0.5 mV
PID: OUT2's offset follows R (P = −1), read on IN2	0.966 of R	1 ± 0.05
Limits: OUT1 maximum and minimum for [−0.2, 0.15] V	0.150009 V, −0.199995 V	± 2 mV
Clipped OUT1 samples counted	yes	yes
Stream at 953.7 rows/s (row time stamps)	1.000 of the rate	1 ± 0.1
Stream at 953.7 rows/s: rows lost (client and FIFO)	0	0
SAFE: R on IN1 with the outputs off	3.1 µV	0 ± 100 µV

Information from the same run (no pass/fail):

- **Analog loopback response** (R against the set 0.2 V, DC-calibrated board): 0.9602 at 1 kHz, 0.9725 at 10 kHz, 1.0207 at 100 kHz, 1.0221 at 1 MHz, 1.0243 at 3 MHz, 1.0560 at 10 MHz, 1.1017 at 20 MHz, 1.0472 at 40 MHz. The captured input shows the same (the lock-in agrees with it to 10^{-4}): a gain step of about 6 % between 1 kHz and 100 kHz in the board's analog path, uncorrected by the DC calibration.
- **Loop linearity** (converters, dither on), R/A against 0.4 V at 1 MHz: with a 0.1 V offset 1.00342 (0.8 V), 1.00115 (0.1 V), 0.99864 (10 mV), 0.97547 (1 mV); without offset 1.00102, 0.99974, 1.00027, 1.05044.
- **Input noise** (IN1, LV, terminated through the loop cable, outputs off; X and Y of 8192 captured readings): 1314 nV/√Hz at 10 kHz, 807 nV/√Hz at 100 kHz, 84 nV/√Hz at 1 MHz, 90 nV/√Hz at 10 MHz.
- **Crosstalk** OUT2 → IN1 (0.14 V rms on IN2): −54.4 dB at 100 kHz, −55.1 dB at 1 MHz, −55.2 dB at 10 MHz.
- **Skew**: IN2 − IN1 with OUT2 90° ahead of OUT1 at 1 MHz: 89.891°. (Not the channels' skew, as first read: with both outputs at one frequency the −55 dB crosstalk pulls each phase by about 0.1°; measured separately the skew is 0.08 ns, see the evening record.)
- **PLL**: 0.051 Hz rms frequency jitter in single readings, locked flag set.
- **Stream** (two demodulators, through the service): 3815 and 7629 rows/s without loss; at 15 259, 30 518 and 61 035 rows/s the service received 9118, 9497 and 9168 rows/s and the FIFO dropped the rest (counted). Readings through Python on the board: about 12 ms each, 7.5 ms after `readings()` became one transaction with the scales cached per revision (the final run below).

After the final software changes (one-transaction readings, cached scales, numpy stream decoding) the board test passed again: 31 checks, τ = 0.99994 of the set 0.1 s from 106 readings of the step.

Earlier runs on the same day found mistakes in the test itself (fixed in `scripts/test_hw.py`): a DFT over a non-integer number of cycles and a continuous-time sinc for the capture's averaging (now a least-squares fit and the discrete response), a step response started after the step (now timed from the apply's sample), an IN1/IN2 balance step that left OUT2 on (its −55 dB crosstalk added coherently to 1 mV signals), and a 10 kHz noise reading that let the input's DC offset through a 16 kHz filter.

Not measured in 1.0: the absolute phase at the connectors (the inputs' loop delays are not stored yet: `analysis.measure_input_delay`), the analog response's split between output and input, noise below 10 kHz and its 1/f corner, the capture's DIO3_P trigger (no driving pin in 1.0's wiring), PID loops faster than 200 Hz on the board, long-term drift.