feedback 2.0: reference documentation

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

Rok Zitko, Jozef Stefan Institute

2026-11-10 (commit 6178370)

Contents

feedback: a lock box with lock-in detection **2**
- Before connecting equipment . . . 3
- Quick start . . . 3
- Presets . . . 4
- Specifications . . . 4
- Hardware setup . . . 6
- Documentation . . . 6
- Changing the design . . . 6
- Source layout . . . 7
- Verification . . . 7
- Author . . . 7

feedback recipes **7**
- Hold a voltage with a fast PID . . . 8
- Pound–Drever–Hall and other demodulated errors . . . 8
- Side of fringe with scan and relock . . . 8
- Slow analog outputs and inputs . . . 9
- A loop’s sensitivity . . . 9
- Gains from a measured plant . . . 9
- A PLL to a TTL clock . . . 9
- A weak signal . . . 10
- Harmonics and intermodulation . . . 10
- Bode plot and resonance fit . . . 10
- Follow the resonance: a loop on the resonator’s phase . . . 10
- A PLL on an external signal . . . 11
- Amplitude and offset loops . . . 11
- The data stream . . . 11
- Captures and spectra . . . 12
- Phases at the connectors . . . 12

How feedback works **12**
- Samples, frames and applies . . . 12
- Oscillators and sines . . . 13
- The reference . . . 13
- Inputs . . . 13
- Mixers and the CIC . . . 13
- Fast PIDs . . . 13
- Lock acquisition . . . 14
- RC filters . . . 14
- R and θ . . . 14
- PIDs . . . 14
- Tuning from a measured loop . . . 15
- Outputs . . . 16
- Slow analog outputs . . . 16
- Slow analog inputs . . . 16
- Status pins . . . 16
- Pins to 1 ns . . . 16

TTL reference . . . 17
Sync output . . . 17
Phase reference and units . . . 17
The frame engine's schedule . . . 18
Readings, FIFO, capture, counter . . . 18
Resources . . . 18
Limits . . . 19

feedback: settings, Python methods, TCP commands, data stream, command line **19**
The configuration . . . 19
Python (driver on the board, `FeedbackClient` on a PC) . . . 21
TCP commands (port 5025) . . . 24
The data stream . . . 25
Command line . . . 25

feedback registers **26**
Control and status (0x000–0x0FF) . . . 26
Configuration words in flip-flops (0x100–0x1C8, 0x200–0x28C) . . . 27
Engine words (0x400–0x7FF) . . . 28
Readings (0x800–0x8FF) . . . 29
FIFO (0x1000–0x1FFF) . . . 29
Capture (0x20000–0x3FFFF) . . . 29

feedback verification **29**
2026-11-05 (evening): the TTL reference locked to awg on a second board . . . 29
2026-10-27 (afternoon): the consolidated software on the board; the user's manual . . . 30
2026-10-27: tuning from the measured loop; an earlier stream's rows . . . 30
2026-10-26 (late night): a level trigger's first sample (simulation, build, board) . . . 31
2026-10-26 (late night): the TTL reference and the sync output referred to OUT1 . . . 31
2026-10-26 (late night): feedback 2.0 on the board . . . 32
2026-10-26 (night): feedback 2.0 (logic freed; offline, simulation, build) . . . 32
2026-10-26 (evening): feedback 1.0 on the board . . . 33
2026-10-26: feedback 1.0 (offline, simulation, build) . . . 33

feedback: a lock box with lock-in detection

feedback turns the Red Pitaya into a **lock box**: fast PID controllers at the full sample rate, slow PIDs and phase-locked loops on lock-in readings, hardware **lock acquisition** (scan, engage, relock), and the board's slow analog outputs and inputs as extra actuators and sensors. It is built from the lockin design's verified core, cut to two demodulators and two oscillators to make room. It has

- **two fast PID controllers** at 125 MSa/s, each from **IN1, IN2, IN1 − IN2** or a **demodulator's mixer product** (the in-phase or quadrature part of a signal at the demodulator's frequency: a Pound–Drever–Hall or dither-lock error signal) through two first-order **low-pass sections**, with P, I (clamped at the limits) and D, **output limits**, hold and reset, added to OUT1 or OUT2 sample by sample;
- **two lock acquisitions**, one per fast PID: a **triangle scan** on its output; in **auto** mode the scan stops when a signal crosses an **engage level** (an input's mean, a slow analog input, or X, Y, R, θ of a demodulator: e.g. a cavity's transmission), the fast PID takes over from there, and when the signal stays beyond a **loss level** the scan resumes (**relock**); counts of engagements and the time locked;
- **two oscillators** from 0 to 62.5 MHz in steps of 0.44 µHz (2^{-48} of the sample clock), switched **phase-continuously** or **coherently** (relative to one zero time t0);
- **two outputs**, each a **sum of the oscillators' sines** with its own amplitudes, plus a DC offset, the fast PIDs and scans that target it, a **monitor** of any reading, optional dither, and **hardware voltage limits** after all of them (lockable until the FPGA is reloaded);
- **two demodulators**, each on IN1, IN2 or IN1 − IN2, mixing at full rate with a **harmonic** m = 0…1023 of an oscillator, optionally a **sideband** m·f_A ± f_B, a sine or square-wave reference and its own phase; **1 to 8 RC sections** (0.5 µs … 10^6 s); readings of **X, Y, R, θ**, the unwrapped phase and **dθ/dt** in rms (or peak) volts at the connectors;
- **four slow PID controllers** at the filter rate (976.5625 kHz), from X, Y, R, θ, dθ/dt, an **input's mean** or a **slow analog input** to an oscillator's frequency or phase, an output's amplitude or offset, or a **slow analog**

output: PLLs, amplitude loops, the slow half of a two-actuator lock; optionally **held unless an acquisition is locked**;
- **slow analog outputs AO0–AO3** (extension connector E2: PWM at 488 kHz with a second-order delta-sigma behind the board's RC filters, 0 to about 1.8 V) and **slow analog inputs AI0–AI3** (the XADC, 0 to 7.0 V), both through the board calibration;
- **status pins** DIO0_N–DIO3_N (locked, scanning, the scan's direction for a scope trigger, a PID locked, frame starts, clipping);
- a **TTL reference input** (DIO0_P, edges timed to **1 ns**): an oscillator's phase at every M-th rising edge, averaged, as a slow-PID source — a **PLL to an external clock**, also at N/M of its frequency, held while the clock is missing; the frequency counter times its edges to 1 ns to seed the oscillator;
- a **sync output** (DIO1_P): an oscillator's square wave with **2 ns edges** and its own phase, e.g. to trigger a scope or another instrument on the modulation;
- a **1 ns capture trigger** on DIO3_P (the edge's position within the sample is reported, and the capture's time axis starts from it);
- **continuous data** (rows of X, Y through a FIFO, `DATA STREAM`), a **capture** of four sources (inputs, outputs, the fast PIDs' errors and outputs at full rate, the readings per frame), a **frequency counter**;
- software: averaged readings, auto-phase, PLL set-up (to an input or the TTL reference), **gains tuned from the measured loop** (fast PIDs: the plant's frequency response from captures, PI gains for given phase and gain margins, the closed loop measured; slow PIDs: from step responses, a setpoint step checked against its prediction), sweeps with a resonance fit, spectra, the inputs' loop delay, a Jupyter panel.

Volts are **real volts at the SMA connectors** (and at E2) through the **board calibration** shared by every design (../common/CALIBRATION.md).

feedback 1.0 brought the fast PIDs, the acquisitions, the slow I/O and the status pins; 2.0 adds the TTL reference, the sync output, the 1 ns trigger and the tuning (after freeing logic in the frame engine: `docs/THEORY.md`, "Resources").

Before connecting equipment

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

- **It drives actuators.** A fast PID or scan moves its output by up to its own **limits** (`fast.<f>.limits`, `acquire.<a>.scan`, V); slow PIDs by theirs. Set the output's **voltage limits** (`outputs.<o>.limits`) for the device it drives (a laser's current input, a piezo driver): the FPGA clamps every sample to them after every sum, PID term and scan. `protect(o)` locks them until the FPGA is reloaded.
- **Loading feedback leaves everything quiet**: outputs off at 0 in converter units, fast and slow PIDs, scans, slow outputs and the stream off, status pins low. After a reboot, the service restores the last configuration — including enabled loops: SAFE first if the setup changed.
- **A loop with the wrong sign runs to its limit.** Start with small gains and narrow limits; the scan and acquisition are off until you choose a mode.
- **Load.** Amplitudes and offsets are set **into the load the board was calibrated with** (50 Ω at the inputs in the calibration's loopback).
- **Extension connectors.** DIO0_P (TTL reference in), DIO3_P (capture trigger), DIO1_P (sync out: a square wave while enabled, low otherwise and after SAFE) and DIO0_N–DIO3_N (status) are 3.3 V LVCMOS, not 5 V tolerant: drive the inputs with 3.3 V logic only. AO0–AO3 are 1.8 V PWM at 488 kHz behind the board's first-order RC filters, which leave a large ripple (0.14–0.17 V p-p seen by AI at half scale): **put an external low-pass between AO and any actuator** (Hardware setup) and load it with a high impedance. AI0–AI3 take 0 to 7.0 V (the board's dividers) — never more.
- **Safe state:** `SAFE` (Python `feedback.safe()`) switches both outputs off (0 V or the nearest protected limit), stops every PID, fast PID, scan and the stream, sets the slow outputs to 0 V and the sync output low, and switches the TTL reference off.

Quick start

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

- **Jupyter** on the board: open `http://192.168.0.149:9090` (password `xilinx`), then `feedback/feedback.ipynb`. The panel has five tabs: **Settings** (oscillators, demodulators, outputs, PIDs, the stream), **Feedback** (fast PIDs, acquisitions and scans, slow outputs, status pins), **Meters** (the readings, live), **Presets** and **Calibration**.

- **Python on the board:**

```
import rpfpga                                    # must come first
from pynq.overlays.feedback import DRIVER, analysis
fb = DRIVER()                                  # loads feedback if the board runs something else
fb.apply({"outputs": {"1": {"enabled": True, "limits": [-0.5, 0.5]}},
          "fast": {"1": {"enabled": True, "input": "in1", "setpoint": 0.1, "i": 1e6, "output":
           ↪  1}}})
print(fb.readings()["fast"]["1"])              # error and output (V, means over a frame)
```

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

```
import sys; sys.path[:0] = ["common/python", "feedback/control"]
from feedback import FeedbackClient
with FeedbackClient("192.168.0.149") as fb:       # the board must run feedback
    fb.set_output(1, enabled=True, limits=[-0.5, 0.5])
    fb.set_fast(1, enabled=True, input="in1", setpoint=0.1, i=1e6, output=1)
    fb.set_acquisition(1, mode="auto", source={"quantity": "in2"},
                       engage={"level": 0.15}, loss={"level": 0.05, "time": 1e-4},
                       scan={"low": -0.2, "high": 0.2, "frequency": 100.0})
    print(fb.readings()["acquire"]["1"])        # state, relocks, time locked
```

  If the board runs another design, switch with `rpfpga.client.connect("192.168.0.149").load("feedback")`.

- **TCP** from anything: `nc 192.168.0.149 5025`, then for example `FAST 1 {"enabled": true, "input": "in1", "setpoint": 0.1, "i": 1e6}`, `OUT 1 ON`, `MODE 1 AUTO`, `READ?`, `AO 0 0.9`, `AI?`, `DIO 0 locked1`, `COUNT? TTL`, `TTLLOCK {"pid": 1, "oscillator": 2, "bandwidth": 1000}`, `SYNC {"enabled": true}`, `SAFE`.

docs/PROTOCOL.md lists every setting, method, TCP command and command-line option. RECIPES.md has worked examples (Pound–Drever–Hall, side of fringe with scan and relock, slow analog loops, a loop's sensitivity, a PLL, a PLL to a TTL clock, tuning).

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, 2 on IN1 at harmonics 1, 2
`pll`	PID 1 locks oscillator 2 to a signal near 10 kHz on IN2 (100 Hz bandwidth, ±500 Hz, held below 10 mV); OUT2 repeats oscillator 2 at 0.1 V

Specifications

Computed figures come from the bit-exact model. The demodulators, filters, slow PIDs, outputs, stream, capture and counter are lockin 1.0's circuits (cut to two units); their measured figures are in ../lockin/README.md. feedback's own new parts are checked by `make test-hw` (docs/VERIFICATION.md); figures not marked measured are computed.

Quantity	Value
Fast PIDs	2 at 125 MSa/s; input IN1, IN2, IN1 − IN2 (after the inputs' offsets) or a demodulator's I or Q mixer product; two first-order low-pass sections, corners 0 (none) or 13.8 MHz … 607 Hz in octaves (α = 2^{-k}, k = 1 … 15); error 25 bits (2^{-9} converter steps); P, I, D gains m·2^(−8 sh), 18-bit m (P to 1000 V/V, I to about $5 \cdot 10^8$ /s, D to 8 µs for unity scales, computed); integrator clamped at the limits; output limits; hold, reset

Quantity	Value
Fast PID latency	input sample to output sample inside the FPGA: 14 samples (112 ns) from a raw input, 18 (144 ns) from a demodulator's product; plus the converters (about 120 ns: lockin's measured loop delay); about 232 ns and 264 ns from connector to connector (computed); measured 235.6 ns around OUT1 → IN1 with 50 cm of cable (from the oscillation of a P loop at 2.12 MHz)
Lock acquisition	2; triangle scan of fast PID k's output (steps of 2^{-32} DAC steps per sample: 1.8 µHz … 7.6 kHz for a 1 V span); engage when a source passes a level (above or below), lose after 2^w frames beyond a loss level; sources: X, Y, R, θ, dθ/dt of a demodulator, an input's frame mean, a slow input; decided per frame, acting 3 frames after the frame (computed: about 4 µs)
Slow PIDs	4 at 976.5625 kHz; inputs X, Y, R, θ (wrapped), dθ/dt, the unwrapped phase, IN1 or IN2 (frame means), AI0–AI3; targets: an oscillator's frequency or phase, an output's amplitude or offset, AO0–AO3; held unless an acquisition locks (optional)
Slow analog outputs	AO0–AO3 (E2): PWM 488.28125 kHz, duty from a second-order delta-sigma of a 16-bit value (offset + PID targets); 0 is off, otherwise 1/256 … 255/256 of full scale, mean exact; nominal 0 … 1.8 V until measured (`CAL SLOW`); the board's RC filter leaves a 488 kHz ripple: AI reads 0.14–0.17 V p-p at half scale, 0.07–0.10 V at 1/6 and 5/6 (measured through AO → AI; at the pin not measured)
Slow analog inputs	AI0–AI3 (E2): XADC unipolar 12 bits, a sequence of about 14 µs (the supplies and temperature Linux reads included); nominal 0 … 7.0 V until measured (AOk → AIk at nominal scales, all four: slopes 0.98–1.01, measured)
Status pins	DIO0_N–DIO3_N: per pin one of low, high, acquisition locked or scanning, a scan's direction, a PID locked, frame starts, an output or input clipping
TTL reference	DIO0_P, 3.3 V: rising edges sampled at 1 GSa/s (ISERDES, 250 MHz in quadrature); every M-th (M = 1 … 65536) taken; the phase of oscillator 1 or 2 at a taken edge, 2^{-32} turn (the sub-sample term from its frequency's top 24 bits: error below 10^{-7} turn, computed); averages of 2^q edges (q = 0 … 15); a frame's value as a PID or acquisition source (degrees: the oscillator's phase at OUT1 when the edge reaches DIO0_P, through the board calibration's pin delays, ±1 ns; wrapping like θ); lost after 2^w frames without an edge (up to 2^20 ≈ 1 s): its PIDs hold; PLL bandwidth up to 19 kHz (the integral gain's range); measured: an oscillator locked to a 5 MHz clock (and to 3/2 of it) to 10^{-5} Hz on average; edges near a sampling instant toggle by 1 ns
Sync output	DIO1_P, 3.3 V: high while oscillator 1 or 2's phase at OUT1 plus an offset is in its first half cycle (its sine positive: phase 0 rises at OUT1's upward zero crossing, ±1 ns through the board calibration's pin delays), 2 ns edges (OSERDES at 250 MHz, 16-bit phase)
1 ns trigger, counter	the capture's DIO3_P trigger and the frequency counter's TTL source time their edges to 1/8 sample (1 ns); the counter: IN1/IN2 (Schmitt) or DIO0_P, gates of 2^4 … 2^31 samples
Oscillators, outputs, demodulators, filters, readings	as lockin 1.0 with 2 oscillators and 2 demodulators (`docs/THEORY.md`)
Capture	4 × 8192 words; sources: IN1, IN2, OUT1, OUT2, the demodulators' inputs, the fast PIDs' errors and outputs (means over 2^k samples) and X, Y, R, θ, dθ/dt, PID outputs and errors (per frame)
Settings after loading	everything off, outputs at 0 in converter units, status pins low

Hardware setup

- **Outputs.** OUT1/OUT2 drive the fast actuators (a laser's current modulation, a fast piezo or an EOM); set their limits first. The slow outputs AO0–AO3 suit slow actuators through a driver (a piezo amplifier's input).
- **External low-pass on AO0–AO3.** The outputs are PWM at 488.28125 kHz; the board's first-order filter leaves a ripple of about 0.15 V p-p at half scale as seen by AI (measured; at the pin it may be larger: Red Pitaya specifies a 160 kHz bandwidth, which would leave about 0.85 V p-p, not measured). A first-order RC of corner f_c divides the ripple by about 488 kHz / f_c (computed): 10 kΩ and 160 nF (100 Hz) leave well under 1 mV, 10 kΩ and 16 nF (1 kHz) at most about 2 mV. Follow it with a buffer (an op-amp follower) unless the load is high-impedance, and keep a slow PID's bandwidth on that output well below f_c (the filter's delay is in the loop). The mean level and its slow changes pass unchanged; the delta-sigma's noise lies near 488 kHz and above, where the RC removes it.
- **Inputs.** LV jumpers (±1 V) for small signals. An error signal from an external mixer goes straight to a fast PID (`"input": "in1"`); a modulated signal (a photodiode behind a phase-modulated laser) to a demodulator, whose product feeds the fast PID (`"input": "product"`).
- **TTL reference, sync output.** A 3.3 V clock (or any TTL edges) into DIO0_P (E1 pin 3, ground on pin 26); the sync output on DIO1_P (E1 pin 5) drives a 50 Ω or high-impedance input.
- **Loopback self-test** (`make test-hw`): OUT1 → IN1 and OUT2 → IN2 (50 Ω terminators, LV jumpers); optionally the E1 jumpers DIO0_N → DIO3_P (pins 4 → 9: the status pins, the 1 ns trigger) and DIO1_P → DIO0_P (pins 5 → 3: the sync output into the TTL reference), awg's wiring, and E2 jumpers AOk → AIk (pins 17–20 → 13–16) for the slow channels.
- **Calibration.** The board calibration (converters, inputs' delays) is shared with every design; the slow channels have their own entries (`CAL SLOW AO0 <V per unit> <offset>`).

Documentation

Document	Contents
docs/manual/user_manual.pdf	user's manual (21 pages, measured and computed figures): what feedback is, safety, the first loop, the lock-in core, fast PIDs, lock acquisition, measured tuning, slow PIDs and PLLs, the TTL reference, sync output and status pins, slow analog I/O, software, calibration, applications, specifications, command reference
../lockin/docs/manual/theory_manual.pdf	the theory of lockin and feedback, shared with lockin: the lock-in core, PIDs and PLLs, the fast PIDs' delay budget, lock acquisition, loop measurement and tuning, nanosecond pins, the slow outputs' delta-sigma
docs/PROTOCOL.md	every setting, Python methods, TCP commands, the data stream, command line
RECIPES.md	worked examples: a fast hold, Pound–Drever–Hall, side of fringe with scan and relock, slow analog loops, a loop's sensitivity, PLLs
docs/THEORY.md	how it works: the retimed datapath, fast PIDs, acquisition, slow I/O, the frame engine, resources, 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
../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
 ↪  (with the pins), XADC reader
make test-remote            # upload, program the board, check identity and power-up state
make install-service        # the control service, with feedback active
python3 ../common/tests/verify_build.py --design feedback
```

`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 ”FDBK”, RTL, one `fb_core` cell, the E1 and E2 pins
`src/fb_core.v`	top: reset, sample counter, frame strobes, registers, datapath, engine, FIFO, capture, counter, slow I/O, status pins
`src/fb_datapath.v`	the full-rate part: inputs, oscillators (11 samples ahead), sines, two demodulators, fast PIDs and scans, two outputs
`src/fb_fast.v`, `src/fb_scan.v`	a fast PID; an acquisition’s scan
`src/fb_nco.v`, `src/fb_sine.v`, `src/fb_sine_rom.v`	oscillators (48-bit, phase preloads), sines (quarter-wave table with interpolation; the ROM is generated)
`src/fb_ref.v`, `src/fb_demod.v`, `src/fb_dsp_acc.v`	reference phase and its sine/cosine or square; mixers and the CIC’s integrators in DSP48s
`src/fb_input.v`, `src/fb_out.v`, `src/fb_xoro.v`	input conditioning; the output’s sum, rounding, dither, limits; the random generator
`src/fb_engine.v`, `src/fb_iir.v`, `src/fb_cordic.v`, `src/fb_scale.v`, `src/fb_f32.v`	the frame engine: combs and RC filters, CORDIC, PIDs, acquisitions, monitors, readings, FIFO rows, the next targets
`src/fb_ao.v`, `src/fb_xadc.v`	slow analog outputs (PWM, delta-sigma); slow analog inputs (XADC, DRP)
`src/fb_pins.v`, `src/fb_ttl.v`, `src/fb_sync.v`	the 1 ns pins (MMCM, ISERDES); the TTL reference; the sync output (OSERDES)
`src/fb_regs.v`, `src/fb_fifo.v`, `src/fb_capture.v`, `src/fb_counter.v`	AXI4-Lite registers and the apply, the FIFO, the capture, the frequency counter
`constraints/dio.xdc`, `constraints/e2.xdc`	the TTL reference, sync output, trigger and status pins (E1); the slow outputs and inputs (E2)
`control/feedback/`	Python package: bit-exact model (`model.py`), driver, analysis, TCP commands, client, data stream and the service’s worker, command-line options, Jupyter GUI
`control/feedback.ipynb`	the notebook
`scripts/test_hw.py`	the loopback hardware test (`make test-hw`)
`docs/manual/`	the LaTeX user’s manual (`user_manual.tex`, `um_*.tex`), its figures, values and board recordings (`data/`)
`tools/`	host-only: `manual_figures.py` and `record_manual_data.py` (the user’s manual’s figures and its board recordings, through the service); numpy, matplotlib
`tests/`	`tb_ref.sv`, `tb_datapath.sv`, `tb_engine.sv`, `tb_system.sv`, `tb_xadc.sv` (simulation), `vectors.py` → `fb_vectors.sv`, `fb_*.mem` (the model’s vectors), Python tests, `fake_core.py` (the model as a register block, in a loopback)

Verification

Version 2.0 passes the offline tests and the six simulations (bit-exact against the model; the system bench through real AXI transactions, with fast PIDs, acquisitions, slow outputs, the TTL reference, the sync output and 1 ns pins) and builds with timing met (WNS +0.265 ns, 82 % of the LUTs); on the board `make test-hw` passes 66 of 66 checks (2026-10-27, build 0x46632204, with the tuning from the measured loop; again with the software shared with lockin in `common/` and the manual’s recordings). Records: docs/VERIFICATION.md.

Author

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

feedback recipes

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

checks a fast hold, the product path, an acquisition, AO → AI, a PLL to the TTL reference and the tuners on the board (docs/VERIFICATION.md); the recipes below for the fast PIDs, acquisitions and slow channels have not been tried as written.

The signs of the gains depend on your plant: with the wrong sign a loop runs to a limit. Set the output's `limits` (and `protect` it) to what the actuator tolerates before enabling a loop: a fast PID's output and scan add to the output before those limits.

Hold a voltage with a fast PID

A plant from OUT1 to IN1 (here a cable: a gain of 1, the converters' delay of about 230 ns from connector to connector, computed): hold IN1 at 0.3 V.

```
feedback.apply({"outputs": {"1": {"enabled": True, "limits": [-0.5, 0.5]}},
                "fast": {"1": {"enabled": True, "input": "in1", "setpoint": 0.3, "i": 2e6,
                               "output": 1, "limits": [-0.5, 0.5]}}})
print(feedback.status()["fast"]["1"]["latency"])        # input to output, seconds
print(feedback.readings()["fast"]["1"])                 # mean error and output of the last frame
```

With I alone the loop's bandwidth is about i·G/2π (G: the plant's gain in V/V): 320 kHz here, where the delay costs 30° of phase. Add P (V/V) and D (s) for a plant with its own pole; `"filter": [f1, f2]` puts up to two first-order low-pass sections (13.8 MHz … 607 Hz in octaves) on the input, `"hold": True` freezes the integrator, `feedback.reset_fast(1)` zeroes it.

Pound–Drever–Hall and other demodulated errors

The fast PID can take a demodulator's mixer product (the input times the reference, at the full rate, before the demodulator's own filters) as its error. A phase modulator driven from OUT2 at 20 MHz, the reflected light's photodiode on IN1, the laser's fast actuator on OUT1:

```
feedback.apply({"oscillators": {"1": {"frequency": 20e6}},
                "outputs": {"2": {"enabled": True, "amplitudes": {"1": 0.5}},
                            "1": {"enabled": True, "limits": [-0.2, 0.2]}},
                "demodulators": {"1": {"input": "in1", "oscillator": 1, "phase": 0.0}},
                "fast": {"1": {"enabled": True, "input": "product", "demodulator": 1, "quadrature":
                 ↪  "x",
                               "filter": [1.3e6, 1.3e6], "setpoint": 0.0, "p": 0.5, "i": 1e5,
                               "output": 1, "limits": [-0.2, 0.2]}}})
```

The low-pass sections remove the product's 2f term (40 MHz). Adjust the demodulator's `phase` for the steepest error slope: scan the laser (next recipe, mode `scan`) and capture `fast_error1` with `in1`. The product path adds 4 samples (32 ns) to the raw input's delay.

Side of fringe with scan and relock

A cavity's transmission on IN1 (0 … 1 V at resonance), its piezo driven from OUT1. Acquisition 1 scans fast PID 1's output until the transmission passes 0.6 V upwards, then freezes the scan there and lets the PID hold IN1 at 0.5 V; if IN1 stays below 0.2 V for 1 ms it scans again:

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

The scan engages in either direction, but a side-of-fringe loop holds only one side of the resonance (the PID's sign: here it must raise the piezo when IN1 falls, the side the rising scan meets first). Engaged on the other side, the loop pushes away, loses the lock and the scan resumes from where it froze, in its direction, so it engages on the right side within one more scan period (counted in `relocks`). `MODE 1 LOCK` over TCP freezes the scan and keeps the PID running whatever the source does; `scan` scans with the PID held at 0. DIO0_N goes high while locked (a

scope trigger or an LED, 3.3 V), DIO1_N follows the scan's direction. A slow PID may be held unless acquisition 1 is locked (`"gate": 1`), e.g. a slow loop on R of demodulator 1.

Slow analog outputs and inputs

AO0–AO3 (E2, 0 … 1.8 V nominal, PWM at 488 kHz: an external low-pass, e.g. 10 kΩ and 160 nF, sets the ripple and bandwidth; README, "Hardware setup") for slow actuators, AI0–AI3 (0 … 7.0 V nominal) for slow sensors. A heater driven from AO0 holding a temperature sensor's voltage on AI0 at 1.2 V with PID 4:

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

Until measured, the nominal full scales apply. With AO0 → AI0 and a voltmeter, store each channel's volts per unit and offset: `feedback.set_slow_calibration("ao", 0, G, B)` (or `CAL SLOW AO0 G B` over TCP; `common/CALIBRATION.md`).

A loop's sensitivity

The outputs carry the oscillators' sines next to the fast PIDs: inject a small sine with the loop's actuator output and demodulate the plant's output with the loop off and on; the ratio is |1/(1 + L)| at that frequency:

```
feedback.apply({"outputs": {"1": {"amplitudes": {"1": 0.01}}},
                "demodulators": {"1": {"input": "in1", "oscillator": 1, "time_constant": 1e-3,
                 ↪  "order": 4}}})
values = analysis.sweep_values(1e3, 2e6, 40, "log")
on = analysis.sweep(feedback, "frequency", {"oscillator": 1}, values)
feedback.apply({"fast": {"1": {"hold": True}}})
off = analysis.sweep(feedback, "frequency", {"oscillator": 1}, values)
feedback.apply({"fast": {"1": {"hold": False}}})
```

Gains from a measured plant

Let a PID measure its plant and choose its own gains. For a fast PID, a free oscillator puts a 10 mV sine on its output at 24 frequencies (20 kHz … 8 MHz) while the PID is held. Captures of the output, the PID's error and its output give the plant at each frequency. The PI gains with the largest integral gain within 60° of phase margin and 10 dB of gain margin are applied, and the closed loop is measured to confirm them:

```
feedback.apply({"outputs": {"1": {"enabled": True, "limits": [-0.5, 0.5]}},
                "fast": {"1": {"enabled": True, "input": "in1", "setpoint": 0.3, "output": 1}}})
t = analysis.tune_fast(feedback, 1)
print(t["p"], t["i"], t["crossover"], t["phase_margin"])        # the design
print(t["measured"]["crossover"], t["measured"]["phase_margin"])  # the closed loop, measured
```

The plant here includes whatever is outside the board: an actuator's response and a resonance up to 8 MHz (pass `frequencies` for another range). `margin`, `gain_margin`, `bandwidth` (a ceiling for the crossover) and `kind="i"` (integral only) shape the design. `analysis.fast_response(feedback, 1)` returns the measured plant alone, and with `closed=True` the loop gain of a running loop.

`analysis.tune_pid(feedback, p)` does the same for a slow PID. Its target (an oscillator's frequency or phase, an output's amplitude or offset, a slow output) is stepped a few times while the PID's error is captured per frame. The PI gains come from that step response, and a setpoint step of the closed loop is compared with the one predicted from it (`t["measured"]["deviation"]`, rms relative to the step). The older static tuners (the plant's gain from two steps, an integral-only loop) remain as `method="static"`.

A PLL to a TTL clock

An external 10 MHz reference (3.3 V) into DIO0_P (E1 pin 3): oscillator 2 locks to it and OUT2 repeats it as a sine; with `multiply` and `divide` it runs at N/M times the clock (here 15 MHz):

```
analysis.lock_ttl(feedback, pid=1, oscillator=2, bandwidth=1e3, multiply=3, divide=2)
feedback.apply({"outputs": {"2": {"enabled": True, "amplitudes": {"2": 0.2}}}})
print(feedback.readings()["ttl"], feedback.readings()["oscillators"]["2"])
```

The reference takes every second edge (5 MHz) and measures oscillator 2's phase there; the seed from the frequency counter (edges timed to 1 ns) picks N = 3. While the clock is missing the PID holds the frequency (`readings()["ttl"]["lost"]`). It holds the frequency it reached when the loss was declared: until then, frames without an edge repeat the last reading and the integrator keeps integrating it. `lock_ttl` sets the loss time to 20/bandwidth (20 ms at 1 kHz), which held a 5 MHz clock 250–570 Hz off (docs/VERIFICATION.md, 2026-11-05); for a fast clock set it to a few periods, e.g. `feedback.apply({"ttl": {"loss_time": 4e-6}})` (0.4–2.7 Hz). Demodulators referenced to oscillator 2 then measure against the external clock. The sync output repeats an oscillator as a 3.3 V square wave with 2 ns edges, e.g. to trigger a scope on the modulation: `feedback.apply({"sync": {"enabled": True, "oscillator": 2, "phase": 0.0}})`.

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):

```
feedback.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(feedback.status()["demodulators"]["1"])          # frequency, ENBW, settling time
analysis.auto_phase(feedback, 1)                       # the signal on +X
r = analysis.average(feedback, ("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 and 2 on the first two harmonics.

```
feedback.load_preset("harmonics")
r = feedback.readings()["demodulators"]
print([r[d]["r"] for d in "12"])
```

Two tones f_A and f_B into a mixer, the sum product f_A + f_B on demodulator 2:

```
feedback.apply({"oscillators": {"1": {"frequency": 100e3}, "2": {"frequency": 37e3}},
                "outputs": {"1": {"enabled": True, "amplitudes": {"1": 0.2}},
                            "2": {"enabled": True, "amplitudes": {"2": 0.2}}},
                "demodulators": {"2": {"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)
feedback.apply({"demodulators": {"1": {"time_constant": 10e-3, "order": 4}}})
result = analysis.sweep(feedback, "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
feedback.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(feedback.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:

```
feedback.apply({"outputs": {"2": {"enabled": True, "amplitudes": {"2": 0.1}}},
                "demodulators": {"2": {"input": "in2"}}})
analysis.lock_pll(feedback, 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 with lockin, `../lockin/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(feedback, 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 with lockin: 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):

```
feedback.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):

```
feedback.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 feedback import DataStream                          # (on the board: from
 ↪  pynq.overlays.feedback)
feedback.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 (measured with lockin); 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 = feedback.capture(("in1", "out1", "x1"), rate=125e6 / 64)
spec = analysis.spectrum(feedback, "in1", averages=8, rate=125e6 / 16)   # dBV and V/sqrt(Hz)
```

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

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(feedback, "in1", output=1, cable_delay_s=0.5e-9, store=True)
analysis.measure_input_delay(feedback, "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. `feedback.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 feedback works

feedback is one module, `fb_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, except the pins sampled to 1 ns (250 MHz in quadrature, from an MMCM) and the sync output's serializer (the converters' 250 MHz clock). Its core is lockin 1.0's (docs of ../lockin) cut to two demodulators and two oscillators, retimed so that the mixers see each input sample at once, with fast PIDs, lock acquisitions, the slow analog I/O, a TTL reference and a sync output added. The Python model `control/feedback/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) ─► x1, x2, x1 − x2 ──────────┬──────────────────┐
oscillators A, B (11 samples ahead): φ = acc + offset + PID phase             │ raw              │
  ├─► sines (delayed 11) ─► per output Σ (A_k + dA_k) sin φ_k + DC + slow word │                  │
  │                          + Σ fast PIDs and scans aimed at it ─► round, clamp ─► OUT1, OUT2│
  └─► demodulator d: cos ψ, −sin ψ of m φ_A + n φ_B + POW                      ▼                  │
        x · ref ─► I1 ─► I2 ──────────────────────────────── products ─► fast PID f (2 LPF, PID) ─┘
═══════ frame = 128 samples, 976.5625 kHz ═══════════════════════════════════════════════════════
combs ─► RC sections 1..8 on X, Y ─► CORDIC: R, θ ─► unwrap, dθ; the inputs' frame sums
X, Y, R, θ, dθ, input means, AI ─► 4 PIDs ─► targets (frequency, phase, amplitude, DC, AO)
                                  ─► 2 acquisitions ─► scan / hold of fast PID a
readings ─► snapshot, ring of 4 frames (capture), FIFO rows (stream)
AO0..AO3: PWM 488 kHz, 2nd-order ΔΣ        AI0..AI3: XADC (DRP)        status pins DIO0_N..3_N
DIO0_P ─► 8 samples/sample ─► TTL reference: φ_C at every M-th edge, averaged ─► a PID source
DIO3_P ─► 8 samples/sample ─► capture trigger (1 ns)    φ_S ─► 4 bits/sample ─► DIO1_P (sync)
```

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. 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 and acquisitions of frame P − 3; their results act from sample 128 (k + 4) for frame k (L_FRAMES = 4).

Configuration is double-buffered: pending words in flip-flops (the full-rate settings: oscillators, references, outputs, inputs, fast PIDs, scans, slow outputs, status pins) and in the pending half of a block RAM (the engine's bank: filters, PIDs, acquisitions, monitors, stream). An **apply** (CTRL APPLY_NOW: the next frame boundary at least **13 samples** ahead; APPLY_AT: APPLY_TIME rounded up to a frame boundary, late if less than 13 samples ahead) makes all of them live together: the datapath's stages load their pending words at the strobes apl[j], j clocks after clock n_a −

12 for an apply at sample n_a, each stage at the j that makes it switch on the same sample; the bank flips at apl[11] (the frame's last clock) and its new pending half is refilled from the active one (512 clocks). AXI writes and reads wait during an apply's strobes and refill. The apply count (modulo 256) is in every FIFO row.

The oscillators' lead. The oscillators compute phase φ(n) 11 samples before sample n (`fb_nco`: the accumulator holds acc(n + 11) in clock n), so a demodulator's reference (11 clocks from phase to cosine) meets input sample n as it arrives: the product of sample n is in the mixer's DSP48 in clock n + 4, I1 in n + 5, I2 in n + 6. The NCO and the reference stages take their configuration (apl) and the engine's targets (frame strobes fs[0], 11 clocks before the boundary) 11 clocks early; the output stages use the sines delayed by 11 clocks, so the outputs keep lockin's latency of 14 samples and every oscillator phase is the same function of the sample counter as before: the retiming is invisible to the model except for the apply's lead of 13 samples (lockin: 2).

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): two oscillators at one frequency keep their phase difference, and returning to a frequency restores its phase.

Oscillators and sines

Each oscillator adds its 48-bit frequency word F (+ the PIDs' dF) to its accumulator every sample; its phase is acc + POFF·2^{16} + dP. 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 (`fb_sine_rom.v`, generated by the model) with a first-order Taylor correction in a DSP48: s(p) = S sin(2π (p + ½)/2^{24}) for the top 24 bits of the phase, S = 2^{17} − 1, within 1.07 LSB after rounding to 18 bits (computed).

The reference

Demodulator d's reference phase is the top 24 bits of m·φ_A + n·φ_B (48-bit, exact) 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.

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 there, so that x is zero at 0 V) plus, with **AC coupling**, the engine's estimate of the input's DC level (one RC section per input at the frame rate, any corner), subtracted from every sample.

Mixers and the CIC

Each demodulator has two DSP48 mixers, x·r_I and x·r_Q, whose P registers accumulate the products (the CIC's first integrator I1); a cascaded DSP48 is the second integrator I2, both modulo 2^{48}. Once per frame the engine latches I2 (slot 5) and runs the two combs: a CIC of order 2 with decimation 128, DC gain 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 response is sinc²(f / 976.5625 kHz); the mixers' 2f products vanish when f is a multiple of 488.28125 kHz, and the RC filters remove the rest.

Fast PIDs

Two PID controllers run at the sample rate (`fb_fast.v`; `model.FastPid`). Each takes one input per sample: **IN1, IN2 or IN1 − IN2** (x, as above) or a **demodulator's I or Q product** x·r (its in-phase or quadrature part: the DC of x·r_I is the in-phase amplitude a cos θ, the 2f part is the filters' job). The product is the difference of successive I1 words of the selected channel (exact; at a change of the source the first difference is a step), 4 samples behind a raw input. In 2^{-8} half converter steps u = x·2^{8} or x·r >> 8, then two first-order **low-pass sections** with 16 fraction bits, the error and the PID:

```
t1 += (u 2¹⁶ − t1) >> k1;   t2 += (t1 − t2) >> k2;   e = sat25(sp − t2 >> 16)
P = (e kp) >> 8 shp;   D = (e kd − (e kd)_previous) >> 8 shd
J = clamp(J) + (e ki) >> 8 shi        (2⁻²⁴ DAC steps; clamp: J >> 8 within [lo, hi])
out = clamp(sat32(P + J >> 8 + D), lo, hi)      (2⁻¹⁶ DAC steps)
```

A section's α = 2^{-k} gives a corner −ln(1 − 2^{-k})·125 MHz/2π: 13.8 MHz (k = 1) … 607 Hz (k = 15); k = 0 passes the input. Gains are 18-bit mantissas times 2^(−8 sh), sh = 0 … 3, in three DSP48s without M register; the driver picks

the finest sh that holds each gain (P to about 1000, I to about $5 \cdot 10^8$ per second, D to 8 µs for an input and an output of equal scales). **Hold** (the setting, or its acquisition scanning) zeroes P and D and keeps J; a **reset** (CTRL, or the acquisition scanning) zeroes J; disabled, every stage clears. The output adds to its output's sum before rounding (so the output's limits apply after it).

Latency. The pipeline: the input registered 3 clocks after its sample, the sections, the error, the products, the integrator, the sum and the clamp, valid 11 clocks after the sample, which is when the output stage adds it to the output sample of the same index. A raw input sample therefore reaches the output 14 samples later (112 ns inside the FPGA; with the converters' loop delay of about 120 ns, about 232 ns from connector to connector), a product 18 samples (144 ns; about 264 ns). Each stage loads its configuration at its own apply strobe: a new configuration acts from one input sample on.

Readings. Each fast PID's error and output are averaged over every frame (FAST_E, FAST_OUT) and can be captured at full rate (half converter steps and DAC steps).

Lock acquisition

Each fast PID f has an acquisition unit a = f (`model.Acquisition`, `model.Scan`):

- **Scan** (`fb_scan.v`): a triangle between LO and HI (2^{-16} DAC steps, added to fast PID f's output) at STEP per sample (2^{-32} DAC steps): running while the unit scans, frozen while it locks, at LO (rising) and contributing nothing while off.
- **State** (in the frame engine, after the PIDs, per frame): **off**, **scan**, **lock**, or **auto**: scanning until the source passes the **engage level** (above, or below), then locked; locked, a source on the far side of the **loss level** for 2^w frames in a row returns to scanning (a **relock** follows at the next engagement). The source is any PID input (X, Y, R, θ, dθ of a demodulator, an input's frame mean, a slow input) of frame k; the state acts from sample 128 (k + 4), like the PIDs' targets (about 4 µs after the frame).
- **Effect**: while scanning, the fast PID is held and its integrator reset (its output is 0: the output is the scan); at the engagement the scan stops where it is and the PID starts from there; in lock the PID holds the error at 0 around that point.
- **Counts**: engagements in auto since the FPGA was loaded (16 bits, wrapping; not cleared by off) and frames locked (32 bits, saturating; cleared outside lock).

A slow PID can be **gated** by a unit (PSRC bit 11, unit bit 12): held unless the unit locks, e.g. a slow actuator that follows the fast one only while locked.

RC filters

Each of the 6 channels (X and Y of 2 demodulators, the 2 inputs' DC estimates) has 8 first-order sections, evaluated by one multiplier in a fixed schedule (`fb_iir.v`; the round keeps lockin's ten issue positions, four of them idle):

```
Y += round(floor((T − Y) / 2³²) · m · 2^(15 − s)),   α = m · 2^−(17 + s)
```

for τ from 0.5 µs to 10^6 s (α within $2 \cdot 10^{-5}$); the order (1 ... 8) picks the output section. The driver reports ENBW and the settling time to 10^{-4}.

R and θ

One CORDIC core (26 iterations) serves both demodulators. X and Y are first shifted together to 29 significant bits, so θ is accurate for signals of a few nV as for volts; R is shifted back. θ is a 32-bit fraction of a cycle; the unwrapped phase sums each frame's change dθ (48 bits); dθ per frame is the frequency deviation.

PIDs

Each slow PID reads one quantity v of frame k, in register units: X, Y, R, θ, dθ or the unwrapped phase of a demodulator, **an input's frame sum** (IN1 or IN2: 2^{14} times its mean in half converter steps; quantity 6) or **a slow analog input** (AI0–AI3: the XADC's 16-bit word; quantity 7), 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 m·2^(e − 48) (|g| < 2^{17} register units; the integrator keeps 16 more fraction bits). **Hold** (manually, when R of the demodulator is below `hold_below`, or when its gating acquisition does not lock) freezes I and zeroes P and D. **Lock**: |e| ≤ tolerance for 2^w frames sets the locked flag. The outputs of PIDs with the same target are summed: an oscillator's frequency or phase, an output's amplitude of an oscillator (25 bits), an output's DC (into the slow word), a **slow analog output's value** (32 bits; the AO adds its offset and clamps). The targets of frame k act from sample 128 (k + 4). A type-2 PLL (`analysis.pll_settings`) reaches the hardware's integral limit near 19 kHz bandwidth; `analysis.lock_pll` seeds a PLL from the frequency counter or locks dθ/dt first (lockin's THEORY, "PLL acquisition").

Tuning from a measured loop

Software only (`analysis.tune_fast`, `tune_pid`; the capture and the applies are the hardware). A loop's gain is L(f) = C(f) P(f): C is the PID, known exactly from its gains; P is the plant as the PID sees it, from its output word to its error word: the converters, the pipeline, whatever is outside the board, and the PID's own input filters. The tuners measure P, choose the gains for given margins, apply them and measure the closed loop.

Fast PIDs. The capture records the output code (OUT), the PID's error e (after its input filters) and its output u, all of the same samples: the fast PID's output computed from input sample n is part of output sample n (`model.Core.step`), so this numbering puts the whole loop delay in P. A sine from a free oscillator (one with no amplitude on any output and not a product's reference) is added to the PID's output, a few millivolts. At each frequency, a least-squares fit of the sine, at its known frequency, in each channel's 8192 words gives its complex amplitude. k is chosen so that at least 8 periods fit; the means over 2^k samples are common to the channels and divide out of the ratios. With the PID held (its output frozen), the plant is

```
P(f) = −E(f) / OUT(f)        (the PID's input units per output volt; e = setpoint − input)
```

and with the loop closed, OUT is U plus the sine, so the loop gain itself is measured:

```
L(f) = −U(f) / OUT(f)
```

The fast PID's response, with no delay of its own in this numbering, is C(z) = p + (i/f_s)/(1 − z^{-1}) + d·f_s·(1 − z^{-1}), with z = e^(i2πf/f_s) (the integrator includes the sample's own error).

Slow PIDs. No hardware modulates a target, so their plant comes from steps. With the PID held, its target's setting (an oscillator's frequency or phase, an output's amplitude or offset, a slow output) is stepped by ±Δ/2 at a frame boundary, while the capture records the PID's error per frame (`capture_step`). The step's sample is the apply's (LAST_APPLY), and a frame-rate word holds the reading of 4 frames earlier. The mean unit-step response s_j over the steps (frame j = 0 is the first with the new target) gives h_j = s_j − s_(j−1) and the plant per frame

```
P(f) = Σ_j h_j e^(−i2πf j / f_o)
```

which is valid down to any frequency once the response has settled. The PID's error of frame k acts on the target 4 frames later: C(z) = (p + (i/f_o)/(1 − z^{-1})) z^{-4}, with z = e^(i2πf/f_o). An input's frame value is the CIC2's, a triangle over two frames. A step 20 samples into a frame therefore shows ½ (108/128)² of itself in that frame and 1 − ½ (20/128)² in the next.

Design (`design_pi`). P is interpolated on 240 frequencies (log |P| and the unwrapped phase linear in log f). For P = 0 and 48 proportional gains, the integral gain is solved that puts the gain crossover at each frequency. Among the loops that meet all of the following, the one with the largest integral gain is taken (the best rejection of slow disturbances):

- every gain crossover inside the measured range, with a phase margin ≥ φ_m (default 60°);
- every phase crossover with a gain margin ≥ g_m (default 10 dB);
- |L| below the gain margin at the top of the range (beyond it, P is unknown);
- the crossover below an optional bandwidth.

A plant with a negative low-frequency gain gets negative gains. `kind="i"` keeps P at 0. The result reports the crossover, both margins and the peak sensitivity max |1/(1 + L)|.

Verification. The gains change bumplessly: the integrator's output carries over, and no reset is made. For a fast PID, L is then measured on the closed loop at the same frequencies, and its crossover and phase margin are compared with the design's. For a slow PID, four setpoint steps of the closed loop (up and back) are captured and averaged, and compared with the step predicted from the measured step response and the new gains. The new setpoint (an engine word) acts from the control frame 3 before the apply. The comparison is their rms difference relative to the step.

In the model (`tests/test_control.py`), the loopback has 20 samples of delay and a first-order low-pass (a = 0.3). The measured P agrees with it within 2 %, and the closed loop's measured L agrees with C·P within 5 % (the phase margin within 3°, the crossover within 5 %). The slow loop's setpoint step agrees with its prediction to 0.6 % rms of the step.

Outputs

Each output sums its two oscillators' sines in a DSP48 cascade ((A_k + dA_k)·s_k, the pre-adder modulo 2^{25}), the DC word (configured offset + the PIDs' DC targets + the monitor, 2^{-16} DAC steps), the rounding constant and optional TPDF dither; two registers keep lockin's latency of 14 samples. Before rounding the sum of the **fast PIDs and scans** aimed at the output is added (2^{-16} DAC steps, not saturated before the code; 0 while the output is off). The rounded 14-bit code is clamped to the output's **limits** (CLAMP; clipped samples counted and flagged), lockable by PROTECT. A **monitor** puts any reading on an output's DC word.

Slow analog outputs

AO0–AO3 (`fb_ao.v`, `model.AoChannel`) are the PWM pins of extension connector E2 behind the board's RC filters. Per PWM period (256 samples: 488.28125 kHz, aligned to the sample counter) each output's value v = clamp(offset + the PIDs' AO targets, 0, 65535) goes through a second-order delta-sigma with error feedback (noise transfer $(1 - z^{-1})^2$):

```
0: off (duty 0, errors cleared);  else y = clamp(v, 256, 65280) + 2 e1 − e2,
d = (y + 128) >> 8,  e = y − 256 d
```

One datapath serves the four outputs, output k in sample 256 j + k (the offsets and targets change only at frame boundaries, so this is its value at the period's start): its sum, then its clamped value, then its delta-sigma; the duties take effect together in sample 256 j + 6, and the pin of sample n leaves the core in clock n + 8. Within those bounds d stays in 0 … 256 and |e| ≤ 128, so the mean duty is exactly v / 65536 and the quantization noise is pushed above the RC filters' band (the bounds cost 1/256 of the range at each end; a delta-sigma that saturates its duty, as near the rails without them, biases its mean). The pin is high for the first d samples of the period. Volts: V = G·v + B from the board calibration (`slow` entries: nominal G = 1.8 V / 65536, B = 0 until measured with `CAL SLOW`).

The ripple at the PWM frequency is left to the filters: the board's first-order filter passes 0.14–0.17 V p-p at half scale as AI sees it (0.07–0.10 V at 1/6 and 5/6; measured 2026-10-26, the same on all four channels), so an external RC is required before an actuator (README, "Hardware setup"). A first-order RC of corner f_c attenuates the 488 kHz fundamental by about 488 kHz / f_c and the delta-sigma's shaped noise, which rises as f^4 up to the PWM rate, by more. A shorter PWM period (32 samples: 3.9 MHz, about 8× less ripple behind the same filters, at the cost of 1/32 of the range at each end) was considered and not adopted (2026-10-26).

Slow analog inputs

AI0–AI3 (`fb_xadc.v`) are the XADC's auxiliary inputs VAUX8, VAUX0, VAUX1, VAUX9 behind the board's dividers. The XADC runs in continuous sequence mode at ADCCLK = 25 MHz (DCLK / 5; at most 26 MHz) over the calibration, the temperature, the supplies (VCCINT, VCCAUX, VCCBRAM, VCCPINT, VCCPAUX, VCCO_DDR, VREFP, VREFN) — the channels Linux reads through its own XADC interface (`/sys/bus/iio/devices/iio:device0`), which shares the converter — and the four inputs: one sequence in about 14 µs. After each end of sequence the four results are read over the DRP port and held: the 16-bit word with the 12-bit result in [15:4], unipolar 0 … 1 V at the pins. The PIDs and acquisitions read the latest word when they run. Volts: V = G·word + B (nominal G = 7.0 V / 65536 until measured).

Status pins

DIO0_N–DIO3_N each show one source (the DIO word, 4 bits per pin), registered in the I/O block: low, high, acquisition 1 or 2 locked or scanning, a scan's direction (rising: a scope trigger at its turning points), PID 1–4 locked, a frame's first sample, an output or an input clipping in the sample. They are levels for equipment and tests, not aligned to samples.

Pins to 1 ns

`fb_pins.v` (awg's trigger front end) samples DIO0_P and DIO3_P with ISERDESE2 in oversample mode, clocked at 250 MHz in 0° and 90° from an MMCM driven by the sample clock: four samples per 4 ns, eight per sample, bit

0 the oldest. The word of sample m arrives in clock m + 4. In simulation (as in awg) an edge set k + 0.5 ns after the clock edge where the counter became n is seen from pin sample 8 n − 3 + k (the benches drive edges this way): on average an edge at time t is stamped t − 2.5 ns. On the board the pins' buffers and traces add delays, which the board calibration describes against OUT1 (`common/CALIBRATION.md`): `pins_delay_s`, an output pin against OUT1 (−28.6 ns: pins lead), and `pins_loop_s`, an output pin to a 1 ns input through a jumper (4.5 ns, measured with the status pin's frame pulse on DIO0_N → DIO3_P: `analysis.measure_pins_loop`).

Referred to OUT1. Taking the clock edge where the counter reads n as the origin of sample n, OUT1 shows sample n from the edge n + 16 (14 samples in the core, 2 in the DAC's interface) plus the DAC's and the analog path's delay; an output pin registered on an edge appears `pins_delay_s` before OUT1 shows the sample of that edge. Hence an edge arriving at an input pin is timestamped, relative to OUT1's sample at that moment, by

```
ttl_delay = pins_loop_s − pins_delay_s + 16 samples − 2.5 ns     (158.6 ns here)
```

and the sync output's edges leave relative to OUT1's sine of the same phase by

```
sync_delay = 109.1 ns − 0.3 ns − 16 samples + pins_delay_s       (−47.8 ns here)
```

(109.1 ns: an edge's mean delay through the serializer: 108.1 ns for a 2 ns slot in simulation, plus 1 ns as an edge takes the first slot after its crossing; −0.3 ns: the sync pin's fast output buffer against the standard pins', measured with the sync looped into the TTL reference at frequencies whose edges sweep the 1 and 2 ns grids: the mean over five frequencies and three phases, ±0.1 ns). The driver refers the TTL reference's readings, its PIDs' setpoints and acquisition levels, and the sync's phase to OUT1 with these: φ_OUT1 = φ_read − 360°·f·ttl_delay, and SPH = phase + 360°·f·sync_delay. Uncertainty about 1 ns (the 1 ns timestamps of the loop measurement, the pins' differences).

TTL reference

`fb_ttl.v`, `model.TtlRef`. Per sample m the first rising edge of DIO0_P (its sub-sample k) is counted; every M-th is taken and measured against oscillator C:

```
e = (φ_C(m) >> 16) + ((f24_C · k) << 5)        (32 bits: 2⁻³² turn, wrapping)
```

with φ_C(m) its phase of sample m and f24_C = F[47:24] + dF[47:24] (its frequency's top 24 bits; the truncations cost less than 10^{-7} turn). The phase and frequency come from the oscillator 10 clocks before sample m and are delayed with it; the product is one DSP48. 2^q taken edges are summed and the sum shifted by q (one bit per clock: the reading changes q samples after its last edge). A frame's value is the reading at its end; it is lost when no edge was taken in that frame and the 2^w frames before (counted up to 2^20). A ring of four frames gives the engine the value of its control frame: a PID (or acquisition) with the source "ttl" (quantity bit 3) sees it in 2^{-32} turns, its error wrapping like θ, and holds while the reference is lost (PSRC loss bit). The edge count and the last frame's value and loss flag are registers.

A PID from "ttl" to the oscillator's frequency is a PLL: the oscillator's phase at the taken edges grows by 360° · (f_C − N f_TTL / M) per second, so the loop locks f_C to an integer N times the taken edges' rate (the seed decides N: `analysis.lock_ttl` sets the oscillator from the counter's measurement times N/M). P = 2π·bw/360 Hz per degree and I = P·2π·bw/4 give a type-2 loop crossing over at bw; the integral gain's range ends near 19 kHz.

Sync output

`fb_sync.v`, `model.sync_bits`. Per sample n four bits, bit j for n + j/4:

```
bit j = not MSB16((φ_S(n) >> 32) + (SPH >> 16) + (((f24_S >> 8) · j) >> 2))
```

high while oscillator S's phase plus SPH is in its first half cycle (its sine positive). They enter an OSERDESE2 (DDR, 4 bits per sample clock, CLK the converters' 250 MHz clock from the same clocking as the sample clock) in clock n + 12 and leave on DIO1_P with 2 ns edges. Looped into the TTL reference (DIO1_P → DIO0_P) the sync's edges appear 107.5 ns after the phase they were computed from (measured 2026-10-26: the 12 samples to the serializer, the OSERDES, the pins and the ISERDES).

Phase reference and units

As lockin's: an input V sin(ωt + θ) at the connector reads X = V cos θ, Y = V sin θ (rms by default); phases are referred to the OUT1 connector through the core's 14 samples and the inputs' calibrated converter loop delays (`analysis.measure_input_delay`). A fast PID's error is in volts at its input (a product: the in-phase or quadrature amplitude in the configuration's units, rms or peak); its output and limits in volts added to its output.

The frame engine's schedule

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

Slots	Work
1 ... 6	the filter words (2 demodulators, 2 inputs; words 2, 3 unused) are read
5	the I2 words of the 6 channels are latched
16 ... 24	the combs: read (16 + c), subtract (17 + c), the IIR input (18 + c); the inputs' frame sums kept by frame mod 4
19 + 10 r + c	section r of channel c issued (results 10 clocks later, forwarded); the last tap leaves in slot 103
14 + 22 p	PID p of frame P − 3: steps 0 … 26
99, 101, 107	the FIFO's word; the monitors' words (issued in 105 and 111)
103 ...	the FIFO row of frame P − 3 (every 2^j frames)
105 ... 118	the next targets: the PIDs' sums (frequency, phase, amplitude, AO) read, saturated and written
106	the taps' frame is latched; the snapshot's frame is taken
107 ... 110	X and Y of frame P − 1 read from the taps' block RAM, converted to floats and kept for the normalizer
109 + 27 d	demodulator d of frame P − 1: normalization, the CORDIC from 112 + 27 d, done D = 11 + 27 d of period P + 1; the results written to the results' block RAM in D + 1 ... D + 6
14, 41 (+ 0 ... 4)	R, θ, dθ and the unwrapped phase of frame P − 2 converted (from the write-back's registers)
110 + 7 a	acquisition a of frame P − 3: its words (110 + 7 a … 114 + 7 a), its source through the PIDs' input path (114 + 7 a), the engage and loss comparisons (116 + 7 a, 118 + 7 a), its state (119 + 7 a)
115 ... 126	the snapshot's oscillator words, DC estimates and frame number
125, 127	the monitors' slow words; the acquisitions' next states

The engine's memories are block RAMs with one read port each, shared by schedule: the taps (16 × 48 bits, read in slots 107 … 110), the results X, Y, R, θ, dθ and the unwrapped phase by frame parity (64 × 48 bits; read for the write-back's previous θ and unwrapped phase in D − 1 and D, the PIDs' sources and R in steps 3 and 4, the monitors in slots 104 and 110, the acquisitions in 113 and 120), and the PIDs' state (I, 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). Their slots never coincide.

Readings, FIFO, capture, counter

Snapshot. CTRL SNAP takes the next latched frame: 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, readable when SNAP_PENDING clears. The fast PIDs' frame means, the acquisitions' states and counts, the slow inputs and outputs are direct registers read beside it.

FIFO. Every 2^j frames the engine pushes a row: the frame number, 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. The board's service drains it every 50 ms (`DATA STREAM`).

Capture. Four channels of 8192 32-bit words. Full-rate sources (IN1, IN2, OUT1, OUT2 codes, the demodulators' inputs, the fast PIDs' errors (half converter steps) and outputs (DAC steps)) store the sum over 2^k samples; frame-rate sources (X, Y, R, θ, dθ, PID outputs and errors) the reading of the frame. Triggers: immediate, DIO3_P's edges, IN1/IN2 Schmitt levels, software, with pretrigger. DIO3_P's edges are found in its eight samples per sample (fb_pins, delayed to the capture's stage): the trigger's sub-sample (status [10:8]) times it to 1 ns.

Frequency counter. Rising edges of IN1 or IN2 through Schmitt levels, or of DIO0_P (timed to 1/8 sample), in a gate of 2^g samples; its stage runs four clocks after the sample counter (the converters' codes and the strobes delayed three clocks) to meet the pin's samples.

Resources

Build 0x46632204 of 2026-10-26 (Vivado 2025.1, xc7z010-1, the whole design): 14 479 LUTs (82.3 %), 16 842 flip-flops (47.8 %), 4 371 of 4 400 slices (99 %), 51 of 60 block RAM tiles, both MMCMs, two ISERDES and an OSERDES; WNS +0.265 ns at 125 MHz, WHS +0.024 ns.

1.0 used 14 902 LUTs (84.7 %, 1 366 as memory) at 99 % of the slices. To make room for 2.0 the engine's LUT memories moved to block RAM (the taps, the results, the PIDs' state: the schedule above) and the capture's and slow outputs' per-channel arithmetic was shared (one shifter for intervals of 2^17 samples and more; one delta-sigma for the four outputs): 13 600 LUTs (77.3 %, 444 as memory), WNS +0.165 ns. The 2.0 hardware added about 860 LUTs. docs/VERIFICATION.md has the numbers of each build.

Limits

feedback is complete for the xc7z010 (2026-10-27): 82 % of the LUTs and 99 % of the slices. Freeing logic for anything more would take cuts. The cheapest, in order: the oscillators' phase accumulators in DSP48s (about 100 LUTs), the fast PIDs' D terms, and their second low-pass sections. lockin's unbuilt ideas (a synchronous filter, a mains canceller) would need at least one of them.

feedback: 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

`feedback.config()` returns, and `feedback.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, "2": ...},
 "demodulators": {"1": DEMODULATOR, "2": ...},
 "inputs": {"in1": {"offset": 0.0, "ac": False, "ac_corner": 10.0}, "in2": {...}},
 "outputs": {"1": OUTPUT, "2": OUTPUT},
 "pids": {"1": PID, ..., "4": ...},
 "fast": {"1": FAST, "2": FAST},          # fast PIDs (125 MSa/s)
 "acquire": {"1": ACQUIRE, "2": ACQUIRE}, # lock acquisition a: scans fast PID a's output
 "ao": {"0": {"offset": 0.0}, ..., "3": ...},   # slow analog outputs AO0..AO3 (E2)
 "dio": {"0": "low", ..., "3": "low"},          # status pins DIO0_N..DIO3_N
 "ttl": TTL,                                    # the TTL reference on DIO0_P
 "sync": SYNC,                                  # the sync output on DIO1_P
 "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 (1–2)
`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

Key	Default	Meaning
`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)
`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) of `demodulator`; `in1`, `in2` (V: the input's mean over a frame), `ai0` … `ai3` (V: a slow input), `ttl` (degrees: the TTL reference's value, the oscillator's phase at OUT1 when the edge reaches DIO0_P; the error wraps to ±180°; the PID holds while the reference is lost)
`pids.p.setpoint`	0.0	the input's units; error = setpoint − input
`pids.p.target`	`{"kind": "none", "oscillator": 1, "output": 1, "channel": 0}`	`frequency` or `phase` of `oscillator`; `amplitude` of `oscillator` on `output`; `offset` of `output`; `ao`: slow output `channel` (V, added to its offset)
`pids.p.p`	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 (demodulator quantities only)
`pids.p.gate`	None	1 or 2: hold unless acquisition 1 or 2 is locked
`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)
`fast.f.enabled`	False	off: output 0, state cleared
`fast.f.input`	`"in1"`	`in1`, `in2`, `diff` (IN1 − IN2), or `product`: a demodulator's mixer product
`fast.f.demodulator`, `fast.f.quadrature`	1, `"x"`	the product's demodulator and part: `x` (I, in phase) or `y` (Q)
`fast.f.filter`	[0.0, 0.0]	Hz: the two low-pass sections' corners (0: none; realized in octaves, 13.8 MHz … 607 Hz)

Key	Default	Meaning
`fast.f.setpoint`	0.0	V at the input (raw inputs: absolute, with the input's offset setting; a product: the in-phase amplitude, in `units`); error = setpoint − input
`fast.f.p`, `fast.f.i`, `fast.f.d`	0.0	V/V; per second; seconds (the output's volts per input volt)
`fast.f.output`	1	the output it adds to
`fast.f.limits`	[−1.0, 1.0]	V: its output's range (the integrator is clamped to it); the output's own limits apply after the sum
`fast.f.hold`	False	freeze the integrator, zero P and D
`acquire.a.mode`	`"off"`	`off`, `scan` (scan; fast PID a held at 0), `lock` (scan frozen, fast PID a runs), `auto` (scan until the source engages, lock, scan again after a loss)
`acquire.a.source`	`{"quantity": "r", "demodulator": 1}`	as a PID's input
`acquire.a.engage`	`{"level": 0.0, "direction": "above"}`	engage when the source passes the level (`above` or `below`)
`acquire.a.loss`	`{"level": 0.0, "time": 0.0}`	lost when the source stays on the far side of the level for `time` (s, 2^w frames)
`acquire.a.scan`	`{"low": 0.0, "high": 0.0, "frequency": 10.0}`	V added to fast PID a's output; triangles per second
`ao.k.offset`	0.0	V of slow output AO k (0: off; nominal 0 … 1.8 V until calibrated)
`dio.k`	`"low"`	status pin DIOk_N: `low`, `high`, `locked1`, `locked2`, `scanning1`, `scanning2`, `rising1`, `rising2` (a scan's direction), `pid1` … `pid4` (locked), `frame` (a frame's first sample), `out_clip`, `in_clip`
`ttl.enabled`	False	the TTL reference on DIO0_P (off: its state cleared, lost)
`ttl.oscillator`	1	the oscillator whose phase it measures at the taken edges
`ttl.divide`	1	M: every M-th rising edge is taken (1 … 65536)
`ttl.average`	1	edges averaged per reading (a power of 2 up to 2^15; others rounded)
`ttl.loss_time`	0.001	s: lost after this long without a taken edge (2^w frames, up to 1.07 s)
`sync.enabled`, `sync.oscillator`, `sync.phase`	False, 1, 0.0	the sync output on DIO1_P: high while the oscillator's phase at OUT1 plus `phase` (degrees) is in its first half cycle (its sine at OUT1 positive): phase 0 rises at OUT1's upward zero crossing; low while disabled
`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`, per fast PID its `latency` (s, connector to connector: the core's samples and the input's calibrated loop delay), per slow output whether it is `calibrated`, and for `ttl` and `sync` their `delay` against OUT1 (s: `ttl_delay()`, `sync_delay()`) and whether the board calibration's pin delays are measured (`calibrated`).

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

Method	
`apply(patch)`, `config()`, `safe()`, `describe()`, `status()`, presets, calibration	common (`rpfpga.Overlay`)

Method	
`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"}, "fast": {f: {"error", "output"}} (V, means over the last frame), "acquire": {a: {"state", "relocks" (engagements since the FPGA was loaded, modulo 2^16), "locked_s"}}, "ai": {k: V}, "ao": {k: V}, "ttl": {"phase" (degrees: the last frame's value, referred to OUT1), "lost", "edges" (taken since enabled)}}`
`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)`, `reset_fast(f)`, `protect(o)`, `zero_time()`	clip and drop counts to 0; clear a PID's state; zero a fast PID's integrator; lock an output's limits; t0 := now
`capture(sources, rate, trigger, pretrigger, level, timeout_s)`	8192 words of up to 4 sources (`in1`, `in2`, `out1`, `out2`, `xin<d>`, `fast_error<f>`, `fast_out<f>` (full rate), `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" (s from the trigger; a DIO3_P edge to 1 ns), "rate", "trigger_sample", "trigger_subsample" (eighths of a sample), source: values}`
`capture_step(sources, patch, rate, after, timeout_s)`	an immediate capture armed, `patch` applied (at the next frame boundary), the capture stopped `after` words past it: capture()'s values with `"t"` (s from the step), `"step_sample"`, `"step_word"`, `"first"`, `"span"`, `"config"`; a frame-rate word holds the reading of 4 frames earlier
`count_frequency(channel, gate_s, hysteresis, level)`	the frequency (Hz) of IN1, IN2 or the TTL pin (`ttl`: edges timed to 1 ns), 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(...)`, `fast(f)`, `set_fast(f, ...)`, `reset_fast(f)`, `acquisition(a)`, `set_acquisition(a, ...)`, `set_mode(a, mode)`, `set_ao(k, V)`, `ao()`, `ai()`, `set_dio(k, source)`, `ttl()`, `set_ttl(...)`, `lock_ttl(...)`, `sync()`, `set_sync(...)`, `tune_fast(f, ...)`, `tune_pid(p, ...)`, `fast_response(f, ...)`, `pid_step_response(p, ...)`, `set_stream(rate, demodulators)`, `record(seconds, rate, demodulators)` (the stream for that long: `{"t", "x<d>", "y<d>", "rate", "lost", "gaps"}`), `stream_settings()`, `units()`, `set_units(u)`, `value(quantity, d)`, `average(...)`, `auto_phase(d)`, `sweep(...)`, `spectrum(...)`; `DataStream(host)` reads the stream.

`control/feedback/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)

Function	
`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)
`ttl_pll_settings(bandwidth, oscillator, frequency_range)`	a PID patch: a type-2 PLL on the source `ttl` (P = 2π·bw/360 Hz/°, I = P·2π·bw/4 /s), limits ±`frequency_range` (default 5·bw)
`measure_pins_loop(device, store, count, timeout_s)`	the board calibration's `pins_loop_s` from the status pin's frame pulse looped into DIO3_P (needs the jumper DIO0_N → DIO3_P)
`lock_ttl(device, pid, oscillator, bandwidth, multiply, divide, frequency_range, average, seed, gate_s)`	the TTL reference (every `divide`-th edge, `average` edges) and the PLL (up to 19 kHz); with `seed` the oscillator set first to `multiply/divide` times the counter's TTL frequency
`tune_fast(device, fast, bandwidth, step, count, margin, apply, method, gain_margin, kind, frequencies, amplitude, oscillator, settle, verify)`	`method="response"` (default): `fast_response` (the PID held), `design_pi` (`kind` `pi` or `i`; phase margin ≥ `margin`°, default 60, gain margin ≥ `gain_margin` dB, default 10, crossover ≤ `bandwidth` if given), the gains applied bumplessly and (`verify`) the closed loop measured: `{"p", "i", "crossover", "phase_margin", "gain_margin", "sensitivity", "f", "plant", "loop", "oscillator", "measured": {"loop", "crossover", "phase_margin", "gain_margin", "sensitivity"}}`. `method="static"`: the output's offset stepped by ±step/2 (V): the plant's static gain G; an integral-only loop crossing over at `bandwidth` (default: `margin`° against the computed delay): I = 2π·bw/G; `{"gain", "delay", "bandwidth", "i"}`
`tune_pid(device, pid, bandwidth, step, count, apply, method, margin, gain_margin, kind, frames, steps, verify)`	`method="step"` (default): `pid_step_response` (`steps` steps, default 4, of `frames` frames), `design_pi` with the slow PID's form, the gains applied bumplessly and (`verify`) a closed-loop setpoint step compared with its prediction from the measured plant: `{"p", "i", "crossover", "phase_margin", "gain_margin", "sensitivity", "f", "plant", "t", "response", "gain", "measured": {"t", "error", "predicted", "deviation", "step", "count"}}`. `method="static"`: its target's setting stepped (frequency 10 Hz, phase 1°, amplitude or offset 10 mV, AO 20 mV): G; I = 2π·bw/G with bw by default a quarter of the demodulator's bandwidth, 10 kHz for an input's mean, 1 kHz for a slow input or the TTL reference; `{"gain", "bandwidth", "i"}`
`fast_response(device, fast, frequencies, amplitude, oscillator, closed, settle, periods, count)`	a sine of `amplitude` (default 10 mV) from `oscillator` (default: a free one) on the fast PID's output at `frequencies` (default 24 from 20 kHz to 8 MHz): the plant P = −E/OUT from captures of the output, the error and the PID's output (the PID held), or with `closed` also the loop gain L = −U/OUT: `{"f", "plant", "loop", "oscillator", "amplitude", "k"}`, complex values as `[re, im]` pairs (numpy)
`pid_step_response(device, pid, step, frames, count, rate)`	the PID held, its target's setting stepped `count` times by ±step/2, its error captured per frame: the unit-step response (`"t"`, `"response"`, `"gain"`: the mean of its last, settled quarter, which also stands for that quarter in the plant) and the plant per frame (`"f"`, `"plant"`) (numpy)
`pid_setpoint_step(device, pid, step, frames, plant, rate, count)`	`count` (default 4) setpoint steps of the closed loop, up and back, captured and averaged; with `plant` (a `pid_step_response`), the predicted error and the rms `"deviation"`

Function	
`design_pi(f, plant, controller, margin, gain_margin, kind, bandwidth)`	PI gains for a measured plant (docs/THEORY.md, "Tuning from a measured loop"); `controller`: `fast_controller` or `pid_controller`
`loop_margins(f, loop)`	`{"crossover", "phase_margin", "gain_margin", "sensitivity"}` of a sampled loop gain
`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)
`spectrum(device, channel, averages, rate)`	an input's spectrum from captures (Blackman-Harris): `{"frequency", "dbv", "density", "bin", "enbw"}` (numpy)
`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`
`FAST <f> <json>`, `FAST <f>?`, `FASTRESET <f>`	fast PID f (1, 2); zero its integrator
`ACQ <a> <json>`, `ACQ <a>?`	acquisition a (1, 2)
`MODE <a> OFF|SCAN|LOCK|AUTO`	acquisition a's mode
`AO <k> <V>`, `AO?`	slow output k's offset (0 … 3); all four
`AI?`	the slow inputs (V, JSON)
`DIO <k> <source>`, `DIO?`	status pin DIOk_N's source (see `dio.k`)
`TTL <json>`, `TTL?`	the TTL reference
`PINSLOOP? [STORE]`	`measure_pins_loop` (jumper DIO0_N → DIO3_P): the pins' loop delay in seconds; STORE: into the board calibration
`TTLLOCK <json>`	`{"pid", "oscillator", "bandwidth"[, "multiply", "divide", "range", "average", "seed", "gate"]}`: `lock_ttl`
`SYNC <json>`, `SYNC?`	the sync output
`TUNE <json>`	`{"kind": "fast"|"pid", "unit"[, "method", "bandwidth", "step", "count", "margin", "gain_margin", "kind", "apply", "verify"; fast: "frequencies", "amplitude", "oscillator", "settle"; pid: "frames", "steps"]}`: `tune_fast`, `tune_pid`

Command	Reply
`RESPONSE? <json>`	`{"kind": "fast", "unit"[, "frequencies", "amplitude", "oscillator", "closed", "settle", "periods", "count"]}`: `fast_response`; `{"kind": "pid", "unit"[, "step", "frames", "count", "rate"]}`: `pid_step_response`
`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|TTL] [<gate s>]`	the frequency, or NaN
`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`
`DATA STREAM [<interval s>]`	the stream (below), on its own connection

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

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 11 float64 fields:

```
frame  t_s  x1 y1  x2 y2  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)
`--fast F JSON`, `--acquire A JSON`	patch fast PID F, acquisition A (repeatable)
`--ao K=V`	slow output AOK's level (repeatable)
`--ttl JSON`, `--sync JSON`	patch the TTL reference, the sync output
`--autophase D`	auto-phase demodulator D

It prints the description, the readings of every demodulator, the active PIDs and fast PIDs, the acquisitions not off,

the slow inputs, and the clip and drop counts (`--json` for everything).

feedback 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, the fast PIDs' means, the acquisitions, the slow inputs and outputs
0x00100–0x001FF	configuration words in flip-flops (pending; live at an apply)
0x00200–0x002FF	more of them: the fast PIDs, the scans, the slow outputs' offsets, the status pins
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	0x4644424B ("FDBK")
0x004	VERSION	R	0x00020000 (2.0)
0x008	CTRL	W	strobes: [0] APPLY_NOW (the next frame boundary at least 13 samples ahead), [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), [17:16] FAST_RESET (fast PID f's integrator from the write's sample)
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 (less than 13 samples ahead: applied at the next boundary)
0x038, 0x03C	AO_VALUES	R	the slow outputs' values of the last PWM period: {AO1, AO0}, {AO3, AO2} (16 bits each)
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	[1:0]: 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)

Offset	Name	Access	Contents
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; [10:8] a DIO3_P trigger's sub-sample (eighths of a sample: its edge to 1 ns); [28:16] the trigger's word
0x070, 0x074	CAP_TRIG_LO/HI	R	the trigger's sample
0x078, 0x07C	CAP_FIRST_LO/HI	R	the first interval's start sample
0x080	CNT_CFG	RW	frequency counter: [1:0] input (0 IN1, 1 IN2, 2 the TTL pin DIO0_P), [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; the TTL pin: in eighths of a sample, 8 n + k)
0x09C, 0x0A0	CNT_START	R	the gate's start sample
0x0A4	TTL_PHASE	R	the TTL reference's value of the last frame (signed, 2^{-32} turn)
0x0A8	TTL_STATUS	R	[0] lost (in the last frame)
0x0AC	TTL_TAKEN	R	edges taken since enabled (32-bit, wrapping)
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), k = 0, 1
0x0D0 + 4k	AI[k]	R	slow input AI k's XADC word (the 12-bit result in [15:4]), k = 0 … 3
0x0E0 + 8f	FAST_E[f]	R	fast PID f's error, mean over the last frame (signed, 2^{-8} half converter steps)
0x0E4 + 8f	FAST_OUT[f]	R	its output, mean over the last frame (signed, 2^{-16} DAC steps)
0x0F0 + 8a	ACQ_STATUS[a]	R	acquisition a: [1:0] state (0 off, 1 scan, 2 lock), [31:16] engagements (auto) since the FPGA was loaded (wrapping)
0x0F4 + 8a	ACQ_LOCKED[a]	R	its frames in lock (saturating)

Configuration words in flip-flops (0x100–0x1C8, 0x200–0x28C)

Pending words, live at an apply; read back from a mirror. The layout is lockin's: the words of oscillators and demodulators 2, 3 and amplitudes A2, A3 are absent (read 0), and an oscillator or demodulator index field has one bit used.

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), [2] oscillator A, [4] oscillator B (bits 3, 5 unused), [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, A1 (+0, +4)	output o: amplitude words of oscillators 0, 1 (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
0x200 + 32f	FCFG	fast PID f: [2:0] input (1 IN1, 2 IN2, 3 IN1 − IN2, 4 + 2d + q: demodulator d's I (q 0) or Q (q 1) product; 0 none), [3] enable, [4] output (0 OUT1, 1 OUT2), [5] hold, [11:8] k1, [15:12] k2 (the low-pass sections: $\alpha = 2^{-k}$, 0 none), [17:16] shp, [19:18] shi, [21:20] shd (the gains' shifts 8 sh)
0x204 + 32f	FSP	its setpoint (signed 25-bit, 2^{-8} half converter steps)
0x208, 0x20C, 0x210 + 32f	FKP, FKI, FKD	the gains' mantissas (signed 18-bit): P = (e·m) >> 8 shp (2^{-16} DAC steps), the integrator's increment (e·m) >> 8 shi (2^{-24} DAC steps), D = (e·m − previous) >> 8 shd

Offset	Name	Contents
0x214, 0x218 + 32f	FLO, FHI	its output's limits (signed 32-bit, 2^{-16} DAC steps; the integrator is clamped to them)
0x240 + 16a	SCAN_LO, SCAN_HI (+4)	acquisition a's scan limits (signed 32-bit, 2^{-16} DAC steps, added to fast PID a's output)
0x248 + 16a	SCAN_STEP	its step per sample (unsigned 32-bit, 2^{-32} DAC steps)
0x260 + 4k	AO[k]	slow output AO k's offset (0 … 65535: 0 off; the PIDs' AO targets add to it)
0x270	DIO	the status pins' sources: [4k+3:4k] DIOk_N: 0 low, 1 high, 2, 3 acquisition 1, 2 locked, 4, 5 scanning, 6, 7 its scan rising, 8 … 11 PID 1 … 4 locked, 12 a frame's first sample, 13 an output clipped, 14 an input clipped
0x280	TCFG	the TTL reference: [0] enable, [1] oscillator (0, 1), [7:4] q: averages of 2^q taken edges, [12:8] w: lost after 2^min(w, 20) frames without a taken edge
0x284	TDIV	M − 1 (16 bits): every M-th rising edge is taken
0x288	SCFG	the sync output: [0] enable, [1] oscillator
0x28C	SPH	its phase offset (2^{-32} turn; bits [31:16] used)

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 (d = 0, 1): [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), [17:16] the demodulators in each row (bits 18, 19 unused), [20] enable
0x560 + 32a	ACFG	acquisition a: [3:0] the source quantity and [5:4] its demodulator or input (as PSRC), [9:8] mode (0 off, 1 scan, 2 lock, 3 auto), [10] engage below the level (else above), [15:11] w: lost after 2^w frames
0x564, 0x568 + 32a	ENGAGE_LO/HI	the engage level (signed 48-bit, the source's register units)
0x56C, 0x570 + 32a	LOSS_LO/HI	the loss level

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	[3:0] quantity (0 X, 1 Y, 2 R, 3 θ: the error is signed32(SP − θ), within ±½ cycle; 4 dθ, 5 the unwrapped phase, 6 an input's frame sum, 7 a slow input's XADC word; 8 (bit 3) the TTL reference's value: wraps as θ, [10] holds while it is lost), [5:4] demodulator (an index beyond 1 reads zeros), input (6: [4]) or slow input (7: 0 … 3), [8] enable, [9] hold, [10] hold when R < RMIN, [11] hold unless acquisition [12] locks
1, 2	SP_LO, SP_HI	setpoint (signed 48-bit, the quantity's register units)
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 a slow output; [5:4] oscillator (frequency, phase; beyond 1: no target), output ([4]: amplitude, DC) or slow output (0 … 3); [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; an input's frame sum: 2^{14} × its mean in half converter steps; a slow input: its XADC word. Targets: frequency 125 MHz / 2^{48}; phase 2^{-48} cycle; amplitude the A words; DC the DC words; a slow output its value (summed, then its offset added and the result clamped to 0 … 65535).

Readings (0x800–0x8FF)

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

Offset	Contents
0x800 + 32d	demodulator d (0, 1): 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, 37 + f fast PID f's error (half converter steps, saturated to 16 bits), 39 + f its output (DAC steps); 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). Codes of demodulators 2, 3 read 0. Full-rate words are the sum of 2^k samples for k ≤ 16, else the sum / 2^(k − 16).

feedback verification

Dated records, newest first. Board: the Red Pitaya STEMlab 125-14 used for every design here (PYNQ, 192.168.0.149). The circuits feedback shares with lockin 1.0 (cut to two demodulators and two oscillators) were measured there: `../lockin/docs/VERIFICATION.md`.

2026-11-05 (evening): the TTL reference locked to awg on a second board

feedback 2.0 as released, loaded for this test on 192.168.0.102 (the board then ran counter), with awg 3.0 on 192.168.0.104: awg's D0 (DIO0_N) into the TTL reference (DIO0_P) with a 20 cm jumper, grounds joined; awg's D1–D3 were also wired (to the sync and status pins 0 and 3) and stayed in high impedance (awg's enable: D0 only). The two boards' crystals are unlocked. Receipt: `../counter/docs/data/board-1.0-awg-tests-2026-11-05.json` (`t4_feedback`; the guide `guides/combining`, "awg + feedback").

- **Counter:** awg's 25-word square wave (exactly 5 MHz of awg's clock) read 5000025.63, 5000025.22, 5000025.24 Hz with 0.01, 0.1, 1 s gates: +5.05 ppm, the crystals' difference (counter and timetagger on the same boards: 5.02 and 5.00 ppm at other times).
- **PLL** (`lock_ttl`, PID 1, oscillator 2, 1 kHz): at 1, 2 and 3/2 times the clock, 200 of 200 readings locked, none lost; the mean frequency +4.92, +5.10, +4.89 ppm (±0.4 ppm: 9–18 Hz rms per reading over 200). TTL phase 0.54° rms at 5 MHz (300 ps): the 1 ns quantization of the edges (289 ps for a uniform phase). The phase of awg's edges against feedback's sampling moves by 1 ps per edge (5 ppm of 200 ns), so the quantization error is a slow sawtooth (one cycle per ~1000 edges): averaging 8 and 64 edges per reading left it at 273 and 278 ps, 512 at 138 ps.
- **Holdover:** awg's clock switched off and on: lost after the loss time, the PID held, relocked within 13 ms. But the held frequency was 250–570 Hz (three trials each, also averaging 64 edges) from the locked one with

`lock_ttl`'s loss time 20/bandwidth = 20 ms: until the loss is declared, frames without an edge carry the last reading (THEORY, "TTL reference") and the integrator integrates it. Loss time 1 ms: 6–15 Hz; 4 µs (w = 2): 0.4–2.7 Hz, within the readings' noise. Not changed yet: `lock_ttl` could derive the loss time from the clock's measured rate (a few taken-edge periods, at least two frames) instead of the bandwidth.

2026-10-27 (afternoon): the consolidated software on the board; the user's manual

Software only (bitstream 0x46632204), after the consolidation of lockin's and feedback's shared software into `rpfpga.demod` and `rpfpga.rows`.

Board (`make test-hw`, the service reinstalled with feedback active): **66 of 66**.

- `tune_fast`: P 0.0888, I $2.36 \cdot 10^{6}$ /s, crossover 385.2 kHz, phase margin 62.9°, gain margin 10.1 dB; the closed loop measured 384.2 kHz, 63.1°, 10.1 dB. Static: 231.9 ns, 359.3 kHz.
- `tune_pid`: I $1.16 \cdot 10^{5}$ /s, crossover 18.8 kHz, phase margin 60.4°; the setpoint steps against the prediction 5.6 % rms of the step (the board's spur, as before).
- TTL PLL frequency jitter $8 \cdot 10^{-5}$ and $4 \cdot 10^{-5}$ Hz rms (1 and 3/2 times the 5 MHz sync).
- Stream through the service: none lost up to 3815 rows/s; at 7629 rows/s 16 rows dropped at the FIFO in 2 s (none in lockin's run).
- A first run failed in the stream checks with a JSON error: the service had been reinstalled with lockin active, so `DATA STREAM` answered `ERR STALE` while the test drove feedback. The stream checks now report that as INFO; run `make install-service` from the design under test.

User's manual (`docs/manual/user_manual.pdf`, figures from `tools/record_manual_data.py` on this board, 2026-10-27): `tune_fast` designed 385 kHz and 63.0°, measured 389 kHz and 62.8°; `tune_pid` 21.5 kHz, 60.8°, 5.8 % from the prediction; lock acquisition engaging once per cycle of a 50 Hz source, IN1 held within 110 µV rms; the sync of oscillator 1 read by the TTL reference within 4.2° peak to peak over all phase settings at 5 MHz (the 1 and 2 ns grids), the TTL reading being minus the sync's phase (both conventions documented); AO0/AO1 → AI0/AI1 slopes 0.989, 0.993. `tests/test_manual.py`: the examples on the model, the figures regenerated from the recordings, the PDF built. Tests: feedback 72 (1 skipped), common 233 (5 skipped).

2026-10-27: tuning from the measured loop; an earlier stream's rows

Software only (bitstream 0x46632204; docs/THEORY.md, "Tuning from a measured loop").

Model tests. The loopback has 20 samples of delay and a first-order low-pass (a = 0.3).

- `fast_response` measures that plant within 2 % at 0.2 … 16 MHz.
- `tune_fast`'s design meets its 60° and 10 dB, and the closed loop's measured gain matches C·P within 5 % (the phase margin within 3°, the crossover within 5 %).
- `pid_step_response` of IN1's frame value shows the CIC2's triangle: ½ $(108/128)^2$ in the step's frame, 1 − ½ $(20/128)^2$ in the next.
- `tune_pid`'s setpoint steps match their prediction to 0.6 % rms of the step.
- Tests: feedback 65 (1 skipped), common 227 (5 skipped).

Board (`make test-hw` on OUT1 → IN1): **66 of 66**.

	Design	Closed loop, measured
fast PID 1 (plant 0.992 V/V at 20 kHz)	P 0.0891, I $2.37 \cdot 10^{6}$ /s: crossover 385.2 kHz, phase margin 62.9°, gain margin 10.0 dB, peak sensitivity 1.56	crossover 387.8 kHz, phase margin 62.7°, gain margin 10.0 dB
slow PID 1, IN1's frame value → OUT1's offset (settled gain 0.979)	P 0.0927, I $1.02 \cdot 10^{5}$ /s: crossover 16.5 kHz, phase margin 75.1°, gain margin 10.0 dB	4 setpoint steps against the prediction: 5.5 % rms of the step

The static tuner on the same plant chose an integral-only loop at 359 kHz (I $2.30 \cdot 10^{6}$ /s).

Found on the board and fixed:

- **The step came too late.** An apply takes about 97 ms on the board (compile, writes, readback, state), longer than a frame-rate capture's 8.4 ms. `capture_step` now arms the capture inside the apply, just before its APPLY write, with 16 words before the step.
- **The gain was noisy.** One averaged frame carries the board's 214 kHz spur, about 7 % of a 10 mV step. The

step response's gain is now the mean of its last, settled quarter (which also stands for it in the plant), and `tune_pid` takes 4 steps.
- **An earlier stream's rows** (both designs). With the stream off, the worker stops draining and rows stay in the FIFO. A service restart keeps the FPGA, so the next stream delivered yesterday's rows first: the 1000 rows/s check read 0.9 % of the rate. An apply that starts a stream now clears the FIFO before its APPLY (`FifoTests` fails without it).

Common: the service listens on a second port, 23, for a design's compatible protocol (lockin's SR86x subset). It answers nothing for feedback.

2026-10-26 (late night): a level trigger's first sample (simulation, build, board)

Found while porting this capture's precomputed trigger decodes back to lockin. The level comparison, registered in the clock before the sample it judges, used in an ARM's clock the **previous** capture's levels and source (IN1 or IN2). When the first interval starts at the ARM's sample (k = 0, or an ARM on an interval boundary), a level trigger (`in1_up` … `in2_down`) could fire on that sample. The bug has been there since 1.0 (8d7dad7). In an ARM's clock, the comparison now takes the new levels and source.

`tb_system` now arms such a capture: IN1 falling through levels ±100 codes around its value at the ARM, after the pin-triggered capture (levels 0, 0 on IN2). The committed RTL triggered on the ARM's sample (15197, word 0); the model and the fixed RTL trigger 69 samples later. All six benches pass (`tb_system`: 38 400 samples, 915 reads). In the feedback tests (61 pass, 1 skipped), one capture now has a 60 s timeout: its 1024 frames of the model took 11 s on a busy host, over the 10 s default.

Build 0x46632204: 14 479 LUTs (82.3 %), 16 842 flip-flops, 4 371 of 4 400 slices (99 %), 51 block RAM tiles; WNS +0.265 ns, WHS +0.024 ns; DRC warnings as before.

Board: `make test-remote` (FDBK 2.0, build 0x46632204, loaded through the driver, power-up state verified), the service reinstalled, `make test-hw`: **60 of 60** (the sync at phase 0 read by the TTL reference at −0.04 ns at 5.004 MHz and −0.13 ns at 6.004 MHz). `make test-hw` has no level trigger, so a separate check ran through the service (a short script with `FeedbackClient.capture`):

- Control: `in1_up` (levels ±0.1 V) on a 100 kHz, 0.3 V sine fired three times out of three where the captured IN1 crosses 0.1 V (0.098–0.100 V on the word before, 0.101–0.102 V on the trigger's word).
- The bug's case: capture A (immediate, levels −0.2/−0.3 V, IN2 at 0 V) leaves the level state high; OUT2's offset then puts IN2 at −0.48 V. Capture B (`in1_down`, levels 0.55/0.35 V, IN1 at 0.43 V between them) never triggered in five trials, as the model says. The old logic would have judged B's first sample by IN2 against A's levels and fired on it every time.

The two records below were first dated 2026-10-27; the date was wrong.

2026-10-26 (late night): the TTL reference and the sync output referred to OUT1

Software only (bitstream 0xc057ecbb). The TTL reference's readings, its PIDs' setpoints and acquisition levels, and the sync's phase are now referred to the oscillator's sine at the OUT1 connector (docs/THEORY.md "Pins to 1 ns"), from the board calibration's `pins_delay_s` (−28.6 ns, measured with a scope for awg, 2026-10-24) and a new board quantity `pins_loop_s` (common: `CAL PINSLOOP`, `analysis.measure_pins_loop`), the delay from an output pin to a 1 ns input: **4.5 ns** with the frame pulse on DIO0_N → DIO3_P (4.5–4.8 ns in repeated measurements: the timestamps' resolution, the pulse cannot be dithered). TTL delay against OUT1 158.6 ns; sync delay −47.8 ns (it leads OUT1).

The sync's constant was first taken from the 5 MHz loop of the previous record (107.5 ns), whose edges sit at fixed places of the 2 ns output and 1 ns input grids; with the sync at phase 0 the TTL reference then read +3.0 ns. At frequencies whose edges sweep the grids (1.2345 … 9.9971 MHz, three phases, averages of 1024 edges) the residual was the same everywhere: +3.26 ns (±0.07 ns; one reading of 15 discarded, 5.4 ns with a 1.3° spread). The sync's model now uses its mean edge delay (the simulated 108.1 ns of a 2 ns slot plus 1 ns: an edge takes the first slot after its crossing) and a pad difference of −0.3 ns. `make test-hw`: **60 of 60**; the sync at phase 0 read by the TTL reference at +0.01 ns (5.0037 MHz) and −0.09 ns (6.0041 MHz), the raw loop 110.2 ns; the stored loop delay re-measured within 0.25 ns. These checks show the two corrections agree; their absolute reference to OUT1 rests on `pins_delay_s` and `pins_loop_s` (about ±1 ns): a scope comparison of the sync with OUT1's zero crossing would check it independently (not done).

Also fixed on the way (common): a long TCP command that edits the configuration and then acquires (`PINSLOOP?`: the status pin's source, then captures) was cancelled by its own edit; a request's own edits now move its admission

point, other clients' edits still cancel it (`test_concurrency`).

2026-10-26 (late night): feedback 2.0 on the board

Build 0xc057ecbb, `make test-remote` (FDBK 2.0 loaded through the driver, power-up state verified), the service reinstalled, `make test-hw`: **57 of 57 checks pass** (third run; the first two exposed three test flaws, below). Wiring as 1.0's record plus DIO1_P → DIO0_P (E1 pins 5 → 3): the sync output into the TTL reference.

- **1.0's checks** pass as before (fast-loop delay 235.7 ns; AOk → AIk slopes 0.993–1.008; stream lossless to 7.6 k rows/s).
- **TTL reference**: the counter on the TTL pin reads the 5 MHz sync output to 0 ppm (10 ms gate, 1 ns timestamps); the TTL phase of oscillator 1 at its own sync is steady (8 readings, averages of 16 edges: $5{\cdot}10^{-6}$° p-p at 5 MHz, 0.36° at 6 MHz). The phase at 5 and 6 MHz (−160.0°, −121.3°) gives the **sync → TTL loop delay: 107.5 ns** (the sync bits enter the serializer 12 samples after their phase, plus the OSERDES, the pins and the ISERDES).
- **PLL to the TTL reference** (`lock_ttl`, 1 kHz, oscillator 2 locked to oscillator 1's sync, seeded by the counter): 5 MHz to $+3{\cdot}10^{-5}$ Hz, 3/2 of it (every second edge, N = 3) to $+7{\cdot}10^{-6}$ Hz (means of 64 readings); phase error under 10^{-5}°. The edges are quantized to 1 ns: when they fall near an ISERDES sampling instant (it depends on oscillator 1's phase on the sample grid, set anew with its frequency) they toggle between neighbouring samples, ±0.9° at 5 MHz per edge, and a frame's frequency carries P times that: 4–6 Hz rms in one run, 10^{-4} Hz in the next. Without edges the reference is lost and the PID holds; it relocks when they return.
- **1 ns trigger**: three captures triggered by the status pin's frame pulse (DIO0_N → DIO3_P) report the same sub-sample (2/8).
- **Tuning on OUT1 → IN1**: `tune_fast` measures a gain of 0.970–0.988 V/V and sets I = $2.3{\cdot}10^{6}$ /s (359 kHz against a computed delay of 231.9 ns); IN1 held to −0.009 mV over 4 ms. `tune_pid` (IN1's frame mean → OUT1's offset): gain 0.95–1.00, IN1 held to −0.002 mV over 4 ms.

Test flaws fixed: the PLL's frequency check took 16 single readings without averaging (±30 Hz per frame from the edges' quantization): now averages of 16 edges and the mean of 64 readings; the loop delay used 5 and 10 MHz (ambiguous: it showed −88 ns instead of +112 ns): now 5 and 6 MHz; the slow PID's hold used its error readings, single frames taken at software-timed moments that sample the board's 214 kHz spur unevenly (a bias of 0.2–0.5 mV either way, while a capture showed IN1 at the setpoint): now a 4 ms capture.

2026-10-26 (night): feedback 2.0 (logic freed; offline, simulation, build)

Freeing logic (1.0's hierarchy: the engine 6 680 LUTs, 1 028 of them LUT memory; the datapath 4 061; the registers 1 546; the capture 1 407; the AO 329). Each change bit-exact against the unchanged model, all six benches passing after each:

- the engine's results (X, Y, R, θ, dθ, unwrapped) in one block RAM with one read port, written one per clock after the CORDIC; the conversions from the write-back's registers; the taps in a block RAM read once at slots 107 … 110 (the CORDIC two clocks later);
- the PIDs' state in a block RAM (`pid_e` was a copy of `e_prev`);
- the capture's words for k > 16 through one shifter (a channel per clock): a throwaway bench compared it with the old capture at k = 17 and 20 with random inputs (equal), as the benches cannot reach 2^17-sample intervals;
- one delta-sigma for the four slow outputs (the pins 8 samples late instead of 3);
- the vectors now also monitor R and a PID's error. Builds: 0x8893ee53 13 890 LUTs, WNS −0.064 ns (the AO's pin compare, the target sums); after registering those, 0xb7d4ce32: **13 600 LUTs (77.3 %)**, 444 as memory, slices 96.8 %, BRAM 51 tiles, WNS +0.165 ns.

2.0 hardware: the pins sampled to 1 ns (awg's MMCM and ISERDES), the TTL reference (edge, ÷M, the phase in a DSP48, averages of 2^q with a serial shifter, a frame ring for the engine, loss), the sync output (OSERDES on the converters' 250 MHz clock), the capture's trigger sub-sample, the counter's TTL source (1/8-sample timestamps), the PID and acquisition source "ttl". The system bench drives both pins with edges k + 0.5 ns after a clock edge (734 edges: a 1.37 MHz + 300 Hz TTL clock with a gap, two trigger edges) through the UNISIM MMCM and ISERDES, and compares bit for bit: every output sample (PID 3 in the first configuration is a PLL from the TTL reference to oscillator 1, held during the gap), the AO pins, the sync output's bits of every sample, the TTL registers (phase, lost — reached in the gap —, edges), the counter on the TTL pin, the capture's trigger sub-sample (3); all six benches pass (912 reads). This also confirms awg's pin-timing calibration for these pins.

Builds: 0xc09f514b 14 458 LUTs, WNS −0.091 ns (the AO sum → clamp); with the sum in a stage of its own, **0xc057ecbb: 14 493 LUTs (82.4 %), 16 841 flip-flops, slices 98.4 %, BRAM 51 tiles, both MMCMs, two**

ISERDES, an OSERDES; WNS +0.346 ns, WHS +0.011 ns (the pin words into the sample clock +1.13 ns). No unclocked pins; the pins' false paths as intended. `common/tests/verify_build.py --design feedback`: all 8 checks pass, reproducing build ID 0xc057ecbb from a source-only copy.

Software: the sections `ttl`, `sync`; the source `ttl` (held while lost); readings `ttl`; `count_frequency("ttl")`; the capture's `trigger_subsample` (its time axis from the edge); `analysis.lock_ttl` (up to 19 kHz: the integral gain's range), `tune_fast`, `tune_pid`; TCP `TTL`, `TTLLOCK`, `SYNC`, `TUNE`, `COUNT? TTL`; client, GUI, CLI, notebook. Python tests 61 (the fake core with a TTL square wave: readings, drift sign and size, the PLL seeded and locked, the sync's cycles, a pin trigger's sub-sample, both tuners), common `test_designs` and `test_workflow` 116.

2026-10-26 (evening): feedback 1.0 on the board

Build 0x8c1bde6d, `make test-remote` (loaded through the driver, power-up state verified) and `make test-hw`: **43 of 43 checks pass** with all four slow channels wired (41 and 42 in the runs before, with two and three). Wiring: OUT1 → IN1 and OUT2 → IN2 through 50 cm RG316 with a T and 50 Ω at each input, LV jumpers, DIO0_N → DIO3_P (E1 pins 4 → 9), and on E2 wires AOk → AIk: pins 17 → 13, 18 → 14, 19 → 15, 20 → 16. Board calibration as lockin's (2026-10-04, the inputs' loop delays of 2026-10-25).

- **lockin's circuits** (two demodulators): R against the captured amplitude 1 kHz … 10 MHz within 0.16 %, linearity to 1 mV, harmonic, sideband, square reference, IN1 − IN2, RC step, captured R, coherent phase, counter (0.22 ppm), PLL (1 mHz, phase error 0.0004°), amplitude and offset PIDs, limits and clip counts, SAFE: as lockin's figures. Converter loop delay from the phase slope 122.1 ns with the cable (stored 119.9 ns).
- **Fast PID** (IN1 → OUT1, I = 10^6 /s): IN1 held at 0.1 V, mean error −0.009 mV over 4 ms. A P-only loop (gain 1.6) oscillates at 2.122 MHz: **loop delay 235.6 ns** around OUT1 → IN1 with the cable (computed: 232 ns from connector to connector plus about 2.5 ns of cable). Through demodulator 1's product (harmonic 0): IN1 held at 0 V (+0.02 mV) against OUT1's 0.1 V offset.
- **Acquisition** (OUT1 scanned ±0.2 V at 1 kHz, engage on IN2 above 0.15 V, loss below 0.05 V for 0.1 ms, IN2 stepped by OUT2's offset): scan, lock, scan, lock; two engagements; IN1 held at the setpoint while locked (−0.03 mV over 4 ms). The status pin DIO0_N (`locked1`) read on DIO3_P follows the state.
- **Slow channels**: AOk → AIk slopes 0.985, 0.996, 1.005, 0.995 (k = 0 … 3; earlier runs 0.982–0.984, 1.000, 0.987–0.996) at the nominal scales once the inputs' nominal full scale is 7.0 V (Red Pitaya's specification; the first run, at 3.5 V, read 0.499). AI's spread over 20 readings (the PWM ripple sampled by the XADC), alike on the four channels: 0.14–0.17 V p-p at 0.9 V, 0.07–0.10 V at 0.3 and 1.5 V. Linux's XADC temperature reads 69.5 °C beside the PL's XADC.
- **Stream**: 953.7 rows/s exact and lossless; lossless up to 7.6 k rows/s, about 9–10 k rows/s delivered above (as lockin).

Fixed on the way (software only): the test's fast-PID error was a single frame's mean (1 µs), which samples the board's 214 kHz spur (about 0.9 mV rms; seen equally in IN1, OUT1 and the fast PID's error and output): it now averages a 4 ms capture. The relock counter counts since the FPGA was loaded (not cleared by off): the test counts the difference, the documents say so. The slow inputs' nominal full scale 3.5 → 7.0 V (`rpfpga.calibration`; unmeasured entries always take the current nominal).

2026-10-26: feedback 1.0 (offline, simulation, build)

Copied from lockin at commit 856a010 (renamed `fb_`, ID FDBK, version 1.0, dither seed 0x4644424B44495448) and cut to 2 demodulators and 2 oscillators (lockin's register layout kept: indices beyond the units read zeros or are ignored, which the vectors cover). Then, each against the bit-exact model first:

1. **Oscillators 11 samples ahead** (A = 11): the references meet their input samples without lockin's 11-clock input delay (input to mixer product 4 clocks instead of 15); 27-bit apply and frame strobes, APPLY_NOW/APPLY_AT with 13 samples of lead, the outputs on the sines delayed by A (LATENCY 14 kept).
2. **Fast PIDs** (`fb_fast.v`, `model.FastPid`): raw inputs or a demodulator's product, two low-pass sections, P/I/D, clamps, joined to the outputs' sums before rounding; frame means and capture sources of their errors and outputs.
3. **Lock acquisition** (engine slots 110 + 7a, `fb_scan.v`, `model.Acquisition`, `model.Scan`): off/scan/lock/auto, engage and loss levels, relocks, locked time; PID gating by an acquisition; the inputs' frame means as PID and acquisition sources.
4. **Slow I/O and status pins** (`fb_ao.v`, `fb_xadc.v`): AO0–AO3 PWM with a second-order delta-sigma, AI0–AI3 from the XADC's sequence over DRP, AI and AO as PID sources and targets, DIO0_N–DIO3_N status sources; common board-calibration fields `slow` (`CAL SLOW`).

Simulation (`make sim`, XSim, Vivado 2025.1): all six benches pass, bit-exact against the model's vectors: `PASS: sysid`, `PASS: reference phase and sines`, `PASS: sample datapath` (random fast PID, scan and acquisition configurations), `PASS: frame engine` (an acquisition losing and regaining lock on X of demodulator 1, slow inputs, a gated PID, an AO target), `PASS: feedback core system (38400 samples, 879 reads)` (fast paths, scans, acquisitions, AO offsets and pins, through real AXI transactions), `PASS: slow analog inputs` (the UNISIM XADC with a stimulus file through `fb_xadc`).

Bugs the benches found on the way: an unsigned sum in `fb_out` that zero-extended the fast term; the UNISIM DSP48E1's 100 ns GSR (the benches now release reset after 16 clocks); a register-mask select on the wrong address bit (`awaddr[8]`, now `[9]`); reads during an apply's 512-clock refill in the system bench's command list (moved).

Python: feedback's tests 53 (and 66 subtests: model, driver, protocol, client, stream, GUI, CLI on the fake core), including the vectors' staleness test (also added to lockin); common `test_calibration` 10 (slow fields, `CAL SLOW`); `test_designs` and `test_workflow` 116.

Builds (Vivado 2025.1, xc7z010-1, post-route phys_opt):

Build	LUT	FF	Slices	DSP48	WNS	
lockin 1.0 (for comparison)	12 953 (73.6 %)	14 283	97.2 %	56	+0.100 ns	
cut to 2 + 2 (0xaadccc84)	11 324 (64.3 %)	11 724	88.2 %	32	+0.047 ns	worst: the IIR comb
retimed, comb pipelined	11 258	11 883	86.8 %	32	+0.063 ns	worst: the capture's mode decode
fast PIDs, acquisition	14 370 (81.7 %)		98.1 %		−0.422 ns	183 endpoints failing
1.0, 0x8c1bde6d	**14 902 (84.7 %)**	15 707	**98.95 %**	40	**+0.102 ns**	WHS +0.015 ns; 48 BRAM tiles

What closed the −0.422 ns: the acquisition in a 7-step pipeline with precomputed counters and thresholds and a registered loss comparison; the scans' thresholds precomputed; the fast PID reworked out of context (1 062 → 810 LUTs, WNS −0.044 → +0.883 ns: y-form sections, saturations by sign bits, select-then-difference for products, one clamp); the capture's trigger thresholds precomputed at ARM; the register write decode precomputed in the accept clock. The worst paths of the final build are lockin's (an output's offset sum in the engine, 16 logic levels). The routed DRC has no errors (the expected DSP48 pipelining and REQP-1723/1724 warnings). The new ports are placed as constrained: AO0–3 on T10, T11, P15, U13 (LVCMOS18), VAUX8/0/1/9 on B19/A20, C20/B20, E17/D18, E18/E19, DIO0_N–3_N on G18, H17, H18, K18 (LVCMOS33).

`common/tests/verify_build.py --design feedback`: all 8 checks pass (recipe build, unchanged reuse, missing export, vendor HDL edit, the RTL edit of `tests/verify_build.json`, a failed build blocking uploads, clean, regeneration from the saved project), reproducing build ID 0x8c1bde6d from a source-only copy (34 minutes).

The plan's area target (≤ ~82 % of the LUTs, room for 2.0) is missed by 3 points; timing is met. 2.0 has to free logic first (docs/THEORY.md, "Roadmap").

Board: see the record above.