lockin and feedback: theory manual

How the lock-in amplifier and the lock box on the Red Pitaya STEMlab 125-14 work: the signal processing, the control loops and their arithmetic

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

Red Pitaya designs `lockin` 1.0 and `feedback` 2.0
October 27, 2026

Abstract

Two FPGA designs for the Red Pitaya STEMlab 125-14 share one signal-processing core. `lockin` is a four-demodulator lock-in amplifier with four oscillators, four control loops and the measurements a lock-in is used for. `feedback` is a lock box: the same core cut to two demodulators and two oscillators, with two PID controllers at the full sample rate, automatic lock acquisition, slow analog outputs and inputs, a TTL reference input and a sync output timed to a nanosecond. This manual explains how every feature of either design works, from the arithmetic in the FPGA to the measurement it performs: the converters and the calibration that makes their numbers volts, the oscillators, the references and the mixers, the decimating and smoothing filters, the conversion to magnitude and phase, noise and resolution, the frame engine that runs the slow computations, the PID controllers and phase-locked loops, the fast controllers and their delay budget, lock acquisition, loop measurement and tuning, the nanosecond timing of digital edges, the slow analog outputs' delta-sigma modulation, and the acquisition paths. The main results are derived; the formulas are those of the designs' bit-exact Python models, which the FPGA's simulation benches match sample by sample; the figures are computed with those models or from the derived expressions. The user's manuals (one for each design) describe how to operate the instruments.

Contents

1 Introduction **4**
About this manual . . . 4
1.1 The lock-in principle . . . 4
1.2 The two instruments . . . 5
1.3 Numbers that recur . . . 6

2 Converters, calibration and real volts **8**
2.1 The converters . . . 8
2.2 Input conditioning . . . 8
2.2.1 AC coupling . . . 9
2.3 The outputs . . . 9
2.4 Delays and the phase reference . . . 10
2.5 Resolution and noise of the converters . . . 11

3 Oscillators and output synthesis **12**
3.1 Phase accumulators . . . 12
3.2 From phase to sine . . . 13
3.3 Output synthesis . . . 14
3.3.1 Dither . . . 14
3.4 Amplitudes in volts . . . 15

4 References and mixing **16**
4.1 The reference phase . . . 16
4.2 Square references . . . 16
4.3 The mixers . . . 17

5 Decimation: the CIC filter **19**
5.1 Integrators and combs . . . 19
5.2 The $2f$ product . . . 20

6 The RC filters **22**
6.1 A first-order section . . . 22
6.2 Cascades . . . 22
6.3 Noise bandwidth and settling . . . 23

7 Magnitude and phase **25**
7.1 The CORDIC . . . 25
7.2 Phase, unwrapped phase and frequency deviation . . . 26

8 Noise and resolution **27**
8.1 The noise of X and Y . . . 27
8.2 Magnitude and phase at low signal-to-noise . . . 28
8.3 The spectrum around the reference: a zoom FFT . . . 28

8.3.1 The noise density at the reference . . . 29
8.3.2 A time constant from the signal . . . 30
8.4 Stability: the Allan deviation . . . 30

9 The frame engine **31**
9.1 Periods and latencies . . . 31
9.2 Configuration and the apply . . . 32
9.3 Readings . . . 32

10 The slow PID controllers **33**
10.1 The controller . . . 33
10.2 The loop . . . 34

11 Phase-locked loops **36**
11.1 The loop . . . 36
11.2 The sampled loop . . . 37
11.3 Tracking and noise . . . 38
11.4 Pull-in . . . 39
11.5 The frequency-locked loop . . . 40
11.6 Variants . . . 41

12 Fast PID controllers [feedback] **42**
12.1 The controller . . . 42
12.2 Latency . . . 43
12.3 The bandwidth a delay allows . . . 43
12.4 The low-pass sections . . . 45
12.5 Product inputs: a demodulated error at the full rate . . . 45
12.6 Fast and slow together . . . 46

13 Lock acquisition [feedback] **47**
13.1 The scan . . . 47
13.2 The state machine . . . 47
13.3 Engaging on a moving scan . . . 48

14 Measuring and tuning a loop [feedback] **50**
14.1 Margins . . . 50
14.2 Measuring the fast loop . . . 50
14.3 Measuring a slow plant from steps . . . 51
14.4 Designing the gains . . . 51
14.5 Applying and verifying . . . 51
14.6 The static methods . . . 53

15 Digital edges timed to a nanosecond [feedback] **54**
15.1 Sampling a pin to 1 ns . . . 54
15.2 The TTL reference . . . 54
15.3 The sync output . . . 55
15.4 The capture trigger and the counter . . . 56
15.5 Status pins . . . 56

16 Slow analog outputs and inputs [feedback] **57**
16.1 The outputs: PWM with a second-order delta-sigma . . . 57
16.2 The inputs: the XADC . . . 58

17 Acquisition and measurement **59**
17.1 The capture . . . 59
17.2 The data stream . . . 60
17.3 The frequency counter . . . 60
17.4 Spectra of an input . . . 61
17.5 Sweeps . . . 61
17.6 Fitting a resonance . . . 61
17.7 Calibrations from loopback . . . 62
17.8 The SR86x port [lockin] . . . 62

A Delays at a glance **63**

B Symbols **65**

C Reproducing the figures **66**

Chapter 1

Introduction

About this manual

Three manuals describe the two designs:

The theory manual (this document, common to both): how the instruments work. Every feature of `lockin` and of `feedback` is explained here, from the arithmetic in the FPGA to the quantity it measures or controls. Passages that concern only one design are marked [lockin] or [feedback]; unmarked passages hold for both.

The user's manual of lockin (`lockin/docs/manual/user_manual.pdf`) and

the user's manual of feedback (`feedback/docs/manual/user_manual.pdf`): how to set each instrument up and operate it, every setting and command, and worked applications with measurements.

The designs' Markdown documents remain the developers' references: `docs/THEORY.md` (the FPGA's schedule and resources), `docs/REGISTERS.md` (the register map), `docs/PROTOCOL.md` (the Python and TCP interfaces) and `docs/VERIFICATION.md` (dated test records).

How the statements here are backed. Each design has a Python model (`control/lockin/model.py`, `control/feedback/model.py`) that computes every signal of the FPGA with the same integer arithmetic. The model is the reference: the simulation benches run the FPGA's Verilog against it and require identical results, sample by sample, frame by frame, and through real register transactions. A formula quoted here is the model's, and a figure labelled *computed* was calculated with the model or with the expression derived in the text. Figures and numbers that were *measured* on a board are labelled so, with the date. Excerpts of the sources are quoted verbatim and checked against them by the manuals' tests.

1.1 The lock-in principle

A lock-in amplifier measures the amplitude and phase of a signal component at a known frequency, however small it is compared with the noise around it (introductions: [1–3]). The input

$$v(t) = A\sin(2\pi f t + \theta) + n(t) \tag{1.1}$$

is multiplied by two references at the same frequency, in quadrature, and the products are low-pass filtered:

$$\overline{v(t)\,\sqrt{2}\sin(2\pi f t)} = \frac{A}{\sqrt{2}}\cos\theta \equiv X, \qquad \overline{v(t)\,\sqrt{2}\cos(2\pi f t)} = \frac{A}{\sqrt{2}}\sin\theta \equiv Y. \tag{1.2}$$

The products' components at $2f$ and every noise component farther from f than the filter's bandwidth B average out; X and Y are the in-phase and quadrature parts of the signal (here in rms volts, the

convention of commercial instruments), and

$$R = \sqrt{X^2 + Y^2} = \frac{A}{\sqrt{2}}, \qquad \theta = \operatorname{atan2}(Y, X) \tag{1.3}$$

its rms amplitude and phase. Only the noise within about $\pm B$ of f remains: with a white input noise density e_n (V/$\sqrt{\text{Hz}}$) the output fluctuates by $e_n\sqrt{\text{ENBW}}$, where ENBW is the filter's equivalent noise bandwidth (chapter 8). A one-second time constant gives ENBW ≈ 0.25 Hz: a signal of a microvolt is then measured to about a percent in a noise density of $20\,\text{nV}/\sqrt{\text{Hz}}$, although the input's noise over its full 62.5 MHz band is 150 μV rms.

Equivalently, multiplying by $\mathrm{e}^{-\mathrm{i}2\pi ft}$ shifts the input's spectrum down by f: the complex output $z = X + \mathrm{i}Y$ is the input's spectrum around f, moved to zero frequency and filtered. That view explains the harmonics and sidebands the designs can detect, the zoom spectrum (section 8.3) and the noise of X and Y.

The reference comes from the instrument's own oscillators, which also drive its outputs: the signal is usually the response of an experiment to the instrument's own excitation, so that the reference is coherent with it by construction. A reference can also follow an external signal: a phase-locked loop (chapter 11) locks an oscillator to it.

1.2 The two instruments

Both designs run in the Red Pitaya's FPGA (a Xilinx Zynq xc7z010) at the board's sample clock of $f_\text{s} = 125$ MHz, between its two 14-bit analog-to-digital converters (inputs IN1, IN2) and its two 14-bit digital-to-analog converters (outputs OUT1, OUT2). The board's ARM processor runs the instrument's driver and a network service; the user's programs talk to either. Figure 1.1 shows the core both designs share and table 1.1 compares them.

Table 1.1: The two designs. Both use the same converters, sample clock and frame rate.

	`lockin` 1.0	`feedback` 2.0
oscillators (48-bit DDS)	4	2
demodulators	4, each on IN1, IN2 or IN1 − IN2, harmonic $m \le 1023$, sideband	2, the same
filters	CIC + 1–8 RC sections, $\tau \ge 0.5\,\mu$s	the same
slow PIDs (976.5625 kHz)	4: from X, Y, R, θ, dθ, the unwrapped phase	4: also the inputs' means, the slow inputs, the TTL reference
fast PIDs (125 MSa/s)	—	2, on IN1, IN2, IN1 − IN2 or a mixer product
lock acquisition	—	2: scan, engage, loss, relock
slow analog I/O	—	4 PWM outputs (2nd-order ΔΣ), 4 XADC inputs
digital pins	capture trigger	capture trigger (1 ns), TTL reference (1 ns), sync output (2 ns edges), 4 status pins
data	stream, 4-channel capture, counter	the same
remote control	TCP, Python, SR86x-compatible port	TCP, Python

The **full-rate part** computes something for every sample: it conditions the inputs (offset, AC coupling, the difference IN1 − IN2), runs the oscillators, forms each demodulator's references and mixes them with its input, integrates the products for the decimating filter, and sums the oscillators' sines, the offsets and the controllers' outputs into the output codes. `feedback`'s fast PIDs belong to this part.

The **frame engine** runs once per *frame* of 128 samples, at $f_\text{o} = f_\text{s}/128 = 976.5625$ kHz. It finishes the decimating filter, runs up to eight first-order low-pass sections on each demodulator's X

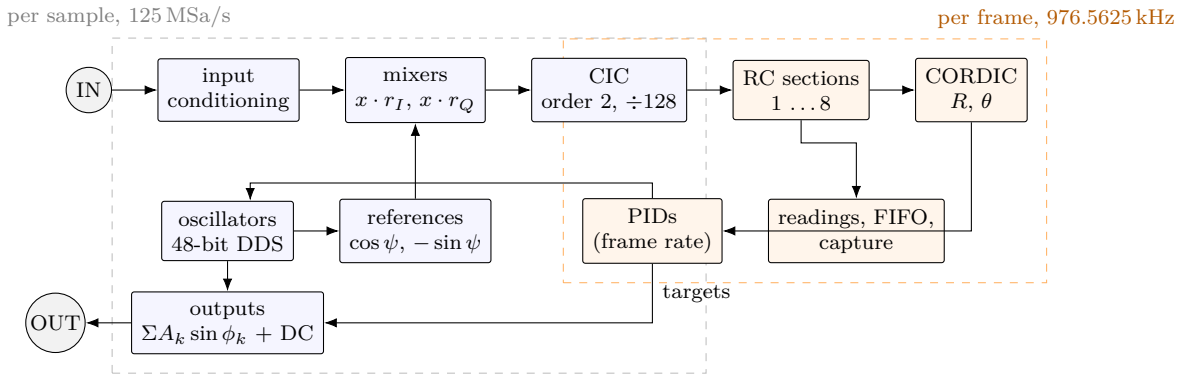


Figure 1.1: The signal-processing core shared by both designs. The full-rate part (grey) handles every 8 ns sample; the frame engine (orange) works once per frame of 128 samples on the decimated results. `feedback` adds, per sample, two fast PID controllers between the input conditioning (or a mixer's product) and the outputs.

and Y, converts them to R and θ, runs the four slow PID controllers (and, in `feedback`, the lock acquisitions), and produces the readings, the data stream and the capture's frame-rate values. Its results act back on the full-rate part as *targets*: corrections to the oscillators' frequencies and phases, to the outputs' amplitudes and offsets.

1.3 Numbers that recur

Table 1.2: Constants of both designs.

sample rate	$f_s = 125\,\text{MHz}$	one sample = 8 ns
frame	128 samples	1.024 μs
frame rate	$f_o = f_s/128 = 976\,562.5\,\text{Hz}$	
frequency resolution	$f_s/2^{48} = 0.444\,\mu\text{Hz}$	48-bit phase accumulators
reference phase	2^{-24} cycle	24-bit reference phase
output latency	14 samples	from the sample counter to the core's output
frame results act	4 frames later	sample $128(k+4)$ for frame k
converters	14 bits	inputs LV $\approx \pm 1$ V, HV $\approx \pm 20$ V

Samples and frames. A sample counter counts the 8 ns clocks since the FPGA was loaded; *sample* n is the input sample that arrives when the counter reads n. Frame k is samples $128k \ldots 128k + 127$. Every time quoted inside the FPGA is a number of samples or of frames from these origins, so that two events in different parts of the design can be compared exactly.

Units. Inputs are converted to *half converter steps* (the code doubled, so that the difference IN1 − IN2 keeps a step's resolution), outputs are written in *DAC steps* and fractions of them. The driver turns every quantity into physical units — volts at the connectors, hertz, degrees, seconds — through the board's calibration (chapter 2). X, Y and R are reported in rms volts by default, or in peak volts (setting `units`).

Notation. f_s the sample rate, f_o the frame rate, f a frequency, ϕ an oscillator's phase in cycles (a fraction of a turn, as the hardware keeps it), ψ a reference's phase, θ a measured phase, τ a filter's time constant, n an RC filter's order, $z = e^{i2\pi f/f_s}$ (or $e^{i2\pi f/f_o}$ for frame-rate signals) the unit-delay variable of sampled systems. $\lfloor\cdot\rfloor$ is the floor and $\gg$ an arithmetic right shift (a floor division by a power of two), as in the FPGA.

How it is verified

The simulation benches `tb_ref`, `tb_datapath`, `tb_engine` and `tb_system` of each design compare the Verilog with the model: the references and sines, every output sample, every frame's filter outputs, R, θ, the PIDs' targets, the readings, the FIFO's rows and the captures, the last through real register transactions. A design is built only from Verilog that passes them; the board tests (`make test-hw`) then check the physical behaviour, mostly as ratios that do not depend on a calibration.

Chapter 2

Converters, calibration and real volts

Every number in the FPGA is a converter code or derived from one. This chapter follows a voltage from the input connector into the arithmetic and from the arithmetic back to the output connector, and shows how the board's calibration turns codes into volts at the connectors.

2.1 The converters

The STEMlab 125-14 has two 14-bit ADCs and two 14-bit DACs, all clocked at 125 MHz from the same oscillator as the FPGA. Each input has a jumper for two ranges, *LV* (nominally ±1 V) and *HV* (nominally ±20 V, a 1:20 divider before the same converter). The converters stream signed 16-bit words (the 14-bit code in the top bits) to the FPGA; the designs take the 14-bit codes.

Real converters differ from their nominal scales by several percent and have offsets of millivolts: on the board used for these manuals the DACs put out 13.5 % and 14.3 % more than nominal and the LV inputs read 11.3 % and 12.0 % low (measured 2026-10-04, `common/CALIBRATION.md`). A loopback hides most of this, because the errors nearly cancel; a calibration against a voltmeter does not.

The board calibration. One calibration file on the board describes the converters for every design (`rpfpga.calibration`):

$$\text{ADC:}\quad c \cdot 4 = K\,V_{\text{in}} + A, \qquad\qquad \text{DAC:}\quad V_{\text{out}} = G\,c_{16} + B, \tag{2.1}$$

with K and A per input and per jumper setting (codes per volt and the code at 0 V, in 16-bit units: $K \approx 32768/\text{V}$ on LV), and G and B per output (volts per 16-bit code and the voltage at code 0). The calibration is measured once with a design that can read its inputs: input offsets with the inputs terminated, the outputs against a voltmeter, and the inputs against the outputs in loopback. Optional fields extend it to AC signals: an amplitude response table and a delay per output and per input (section 2.4). Until measured, the values are nominal, and the designs say so in their status. The designs fold these constants into their register words, so that every user value is a voltage at a connector.

2.2 Input conditioning

Each 14-bit input code c becomes the signal x that the mixers, the fast PIDs and the capture see:

$$x = \text{sat}_{16}(2c - b) \quad \text{(IN1, IN2)}, \qquad x = \text{sat}_{16}(2(c_1 - c_2) - (b_1 - b_2)) \quad \text{(IN1 − IN2)}, \tag{2.2}$$

in *half converter steps*: doubling the code keeps the difference IN1 − IN2 at full resolution without a separate scale. The *background* b is the input's offset word plus, with AC coupling, the engine's estimate of its DC level. The driver puts the converter's own offset into the offset word (from the calibration's A), plus any offset the user asks for, so that $x = 0$ at 0 V. One volt at the connector is therefore $2K/4 = K/2$ half steps of x on that input.

The difference IN1 − IN2 subtracts the codes, not volts: it is exact only if the two inputs have equal gains. The driver scales readings of the difference with the mean of the two inputs' calibrated gains; the residual mismatch (0.75 % between the inputs of this board before calibration) limits the common-mode rejection at DC, and the inputs' different frequency responses limit it at AC.

Clipping. A code at either end of the 14-bit range (−8192 or +8191) counts as clipped: the hardware counts such samples per input and flags the frame in which they occurred (in the data stream's rows and in the counters), because a clipped input distorts every measurement that uses it.

2.2.1 AC coupling

An instrument's input is usually AC coupled by a capacitor. Here the coupling is digital and costs no full-rate filter: the frame engine estimates each input's DC level once per frame and the datapath subtracts the estimate from every sample. The estimate is the input's frame value (the decimating filter's output, a triangle-weighted mean over two frames; chapter 5) through one first-order RC section with the corner f_c (`ac_corner`, 1 μHz to 100 kHz); like every frame result it acts four frames after its frame, held for a whole frame. For a component at frequency f the coupling's response is therefore

$$H_{\mathrm{AC}}(f) = 1 - \operatorname{sinc}^3\!\left(\frac{f}{f_{\mathrm{o}}}\right) \frac{\alpha}{1-(1-\alpha)\,\mathrm{e}^{-\mathrm{i}2\pi f/f_{\mathrm{o}}}}\,\mathrm{e}^{-\mathrm{i}2\pi f\cdot 4.5/f_{\mathrm{o}}}, \qquad \alpha = 1-\mathrm{e}^{-2\pi f_c/f_{\mathrm{o}}}, \tag{2.3}$$

with $\operatorname{sinc} x = \sin(\pi x)/(\pi x)$: the triangle mean contributes sinc^2, the hold over a frame sinc, and the 4.5 frames from the triangle's centre to the held value's centre the delay. Far below f_{o} this is a first-order high-pass, $H_{\mathrm{AC}} \approx \mathrm{i}f/(f_c + \mathrm{i}f)$. Near and above a few kilohertz the estimate's delay makes it ring: a 30 kHz corner over-shoots by 3.0 dB above it (figure 2.1). The corner of an AC coupling is meant to be far below the signals: the default is 10 Hz.

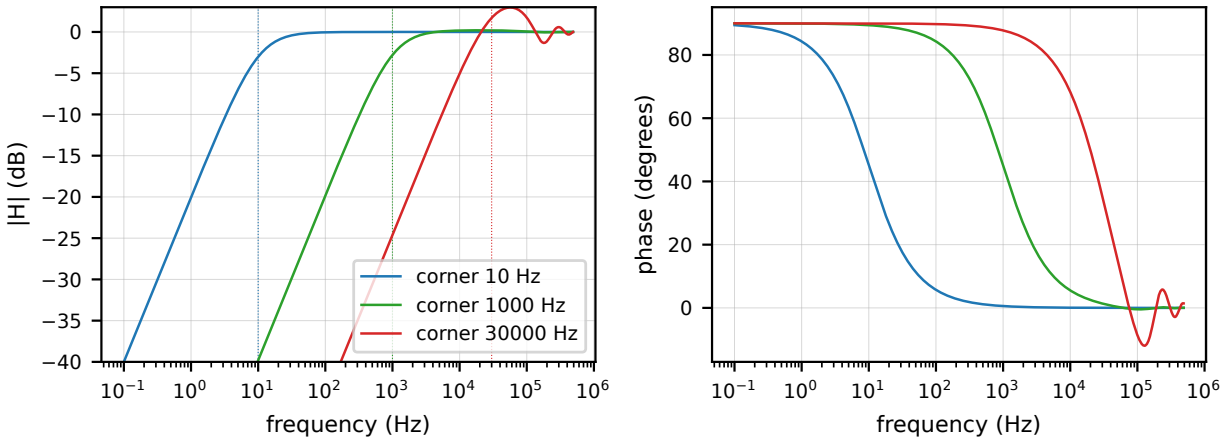


Figure 2.1: AC coupling's response (2.3) for corners of 10 Hz, 1 kHz and 30 kHz (computed; the magnitude agrees with the bit-exact model's to four digits at 5, 30, 100 and 300 kHz). Low corners behave as a first-order high-pass; corners near the frame rate are distorted by the estimate's delay of 4.5 frames.

2.3 The outputs

Each output's arithmetic (chapter 3) ends in a 14-bit code, clamped to the output's limits and sent to the DAC. The DAC holds each code for a whole sample (a zero-order hold), so a sine of frequency f comes out with the amplitude

$$\frac{\sin(\pi f/f_{\mathrm{s}})}{\pi f/f_{\mathrm{s}}} = \operatorname{sinc}(f/f_{\mathrm{s}}) \tag{2.4}$$

of its digital amplitude: −0.092 dB at 10 MHz and −3.92 dB at the Nyquist frequency of 62.5 MHz (figure 2.2). The drivers divide every amplitude by this droop and by the output's calibrated response at the oscillator's frequency, so that the amplitude set is the amplitude at the connector. The analog output stage adds its own response, which the optional response table describes.

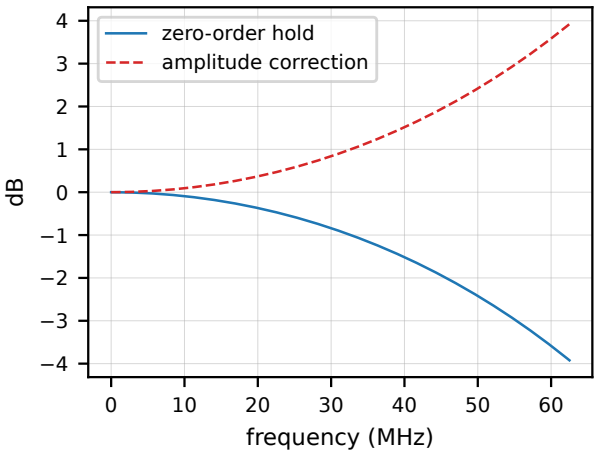


Figure 2.2: The DAC's zero-order hold and the drivers' amplitude correction (computed).

Into which load

The calibration is a DC calibration of the voltage at an unloaded connector. At radio frequencies the outputs behave like a 50 Ω source: a scope's 1 MΩ input sees about twice the set amplitude, a 50 Ω load the set amplitude (checked at 10 MHz). The inputs read about 6 % more at 10 MHz than at DC through a 50 Ω terminator on this board. Measure the response tables for the load you use if the absolute AC amplitude matters.

2.4 Delays and the phase reference

A phase is meaningful only against a reference point. The designs refer every phase to the *sine of the oscillator at the OUT1 connector*: a signal at an input connector that is in phase with OUT1's sine reads $\theta = 0$. To do that, they account for every delay from an oscillator's phase to the input sample that sees it at the connector:

1. the core's output latency: output sample n leaves the core 14 samples after the counter reads n (112 ns);
2. the converters' *loop delay*: from a word on OUT1's DAC port through the connectors, joined directly, to the same value on an input's ADC port (about 120 ns on this board: the DAC's interface and pipeline, the analog paths and the ADC's pipeline). It is part of the calibration (`delay_s` per input and range) and measured in loopback.

A delay T turns into a phase that grows with frequency, $-360° \, f \, T$; the drivers add $+360° f T$ to the demodulator's reference phase, with f its frequency and T the sum of the two delays. The loop delay itself is measured from the phase slope of a loopback (figure 2.3): the phase read at several frequencies falls linearly, and the slope is the delay that the stored value misses (`analysis.measure_input_delay`; 122 ns OUT1 → IN1 with a short cable on this board, the inputs within 0.1 ns of each other). The frequencies are multiples of the frame rate, where the mixers' $2f$ products vanish (section 5.2), so that a short filter gives clean phases quickly.

The convention. The references are $\cos\psi$ and $-\sin\psi$ (section 4.1); a sine in phase with the oscillator would read $\theta = -90°$. The drivers subtract 90° in the reference phase, so that

$$v(t) = V \sin(2\pi f t + \theta) \implies X = V \cos\theta, \quad Y = V \sin\theta \tag{2.5}$$

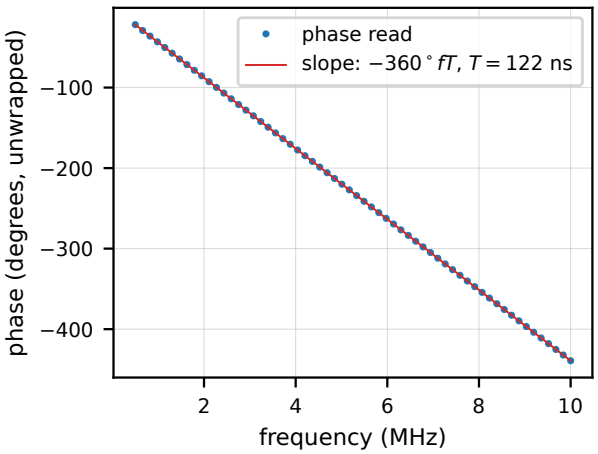


Figure 2.3: A loopback's phase against frequency is a straight line whose slope is the loop delay (illustration; noise of 0.05° added).

(V rms or peak by the setting `units`), with $t = 0$ at a zero crossing of OUT1's sine of the same oscillator. A loop that steers the frequency by Δf leaves the reference's correction at the configured frequency: a phase error of $360° \Delta f\, T$, 0.04° per kHz for 120 ns.

The digital pins [feedback]. `feedback` times digital edges to a nanosecond and refers them to OUT1 too; their delays are separate calibration fields (chapter 15).

2.5 Resolution and noise of the converters

A 14-bit step is about 138 μV on LV on this board (122 μV nominal) and twenty times more on HV. A lock-in averages far below one step: the input's own noise (about 1.2 $\mu\mathrm{V}/\sqrt{\mathrm{Hz}}$ from 1 to 10 kHz, 84 $\mathrm{nV}/\sqrt{\mathrm{Hz}}$ at 1 MHz on LV, measured 2026-10-25) dithers the quantization, and the filters average it. Table 2.1 summarizes the board's measured analog properties that limit the measurements; the root `README.md` keeps the full list.

Table 2.1: Measured properties of the board used for these manuals (LV inputs, 50 Ω terminated; from the root `README.md`).

input noise density	5.2 $\mu\mathrm{V}/\sqrt{\mathrm{Hz}}$ at 37 Hz, 1.1–1.3 from 1 to 10 kHz, 0.65 at 100 kHz, 84 $\mathrm{nV}/\sqrt{\mathrm{Hz}}$ at 1 MHz, 90 at 10 MHz
crosstalk OUT2 → IN1	−54 to −55 dB from 100 kHz to 10 MHz
loopback response	0.960 from DC to 1 kHz, a step near 21 kHz, 1.02 from 0.1 to 2 MHz, 1.06 at 10 MHz
converter loop delay	119.91 ns (IN1), 119.99 ns (IN2), cables excluded
loopback stability	amplitude 56–59 ppm rms over 2 h; phase 0.0014° rms at 1 MHz

Chapter 3

Oscillators and output synthesis

The oscillators are the instrument's time base: they drive the outputs and give every demodulator its reference. This chapter describes how they keep phase, how a phase becomes a sine, and how the sines, offsets and controllers' outputs become an output voltage.

3.1 Phase accumulators

Each oscillator is a direct digital synthesizer (DDS) [4]: a 48-bit *phase accumulator* that adds a 48-bit *frequency word* F every sample. Its phase, in cycles, is the accumulator's value divided by 2^{48}; it wraps modulo one cycle by the arithmetic's own overflow. The frequency is

$$f = \frac{F}{2^{48}}\, f_\mathrm{s}, \qquad \Delta f = \frac{f_\mathrm{s}}{2^{48}} = 0.444\,\mu\mathrm{Hz}, \tag{3.1}$$

from 0 to $f_\mathrm{s}/2 = 62.5\,\mathrm{MHz}$. The phase used by the rest of the design is

$$\phi(n) = \mathrm{acc}(n) + \mathrm{POFF}\cdot 2^{16} + dP \pmod{2^{48}}, \tag{3.2}$$

the accumulator plus a 32-bit phase offset (resolution 2^{-32} cycle) plus the slow PIDs' phase target dP. The PIDs' frequency target dF adds to F in the accumulator, so a frequency loop changes the phase's slope while a phase loop shifts it.

Because the accumulator is exact integer arithmetic, an oscillator's phase after N samples is exactly NF modulo 2^{48}: there is no drift relative to the sample clock and no accumulated rounding. Two oscillators with the same F keep their phase difference for ever; oscillators at frequencies in rational ratios keep fixed relations.

Coherent phases. Changing an oscillator's frequency normally continues its phase from where it is (*continuous* phase). In the mode `phase_mode` = `coherent` the driver applies a change at a sample n_a about 2 ms ahead and preloads every accumulator with

$$\mathrm{acc}(n_a) = F\,(n_a - t_0) \pmod{2^{48}}, \tag{3.3}$$

so each oscillator continues as if it had run at its new frequency since the instant t_0 (set with `zero_time`). Returning to a frequency then restores its phase exactly, and oscillators at one frequency keep a fixed phase difference across any sequence of changes (measured: within 0.001°).

The oscillators' lead [feedback]. In `feedback` each oscillator computes its phase $\phi(n)$ eleven samples before sample n, so that a demodulator's reference (eleven clocks from phase to cosine) meets the input sample n as it arrives and the fast PIDs can use the mixers' products with the shortest delay (section 12.2). The outputs use the sines delayed by eleven clocks, so every phase is the same function of the sample counter as in `lockin`; only the lead needed by an apply changes (13 samples instead of 2).

3.2 From phase to sine

The top 24 bits of the 48-bit phase address a sine. A full table of 2^{24} entries would be far too large; the designs store a quarter wave of 1024 entries and interpolate. Writing the 24-bit phase as two quadrant bits, a 10-bit table address a and a 12-bit fraction f, the sine is (`model.sine22`):

$$y = \frac{s_a\, 2^{22} + d_a\, \delta + 2^{17}}{2^{18}}, \qquad s_a = \operatorname{round}(S \sin \alpha_a)\,, \quad d_a = \operatorname{round}\!\left(S \cos \alpha_a \, \frac{2\pi}{4096}\, 2^9\right), \tag{3.4}$$

with $\alpha_a = 2\pi(a + \frac{1}{2})/4096$ the centre of the entry, $\delta = 2f - (2^{12} - 1)$ the signed distance from that centre, and $S = 2^{17} - 1$ the full scale: a first-order Taylor expansion around the entry's centre, $\sin(\alpha + \epsilon) \approx \sin\alpha + \epsilon\cos\alpha$. The quadrant bits mirror the address and the fraction (second and fourth quadrants) and negate the result (third and fourth). The interpolated value has 22 bits; the sine used by the mixers and outputs is rounded to 18 bits.

lockin/control/lockin/model.py

```
def sine22(p):
    """About 16 S sin(2 pi (p + 1/2) / 2^24) for a 24-bit phase p, 22-bit signed."""
    q = (p >> 22) & 3
    a = (p >> FRAC_BITS) & (QUARTER - 1)
    f = p & ((1 << FRAC_BITS) - 1)
    if q & 1:
        a, f = QUARTER - 1 - a, (1 << FRAC_BITS) - 1 - f
    s, d = TABLE[a]
    delta = 2 * f - ((1 << FRAC_BITS) - 1)
    y = ((s << 22) + d * delta + (1 << (Y_SHIFT - 1))) >> Y_SHIFT
    return -y if q & 2 else y
```

The Taylor term's error is the second-order term, $\frac{1}{2}\epsilon^2 \sin\alpha$ with $|\epsilon| \le 2\pi/8192$: at most $S \cdot 2.9 \times 10^{-7}$, about 0.04 of an 18-bit step. The table's own rounding adds up to half a step; together the interpolated sine is within 0.56 LSB of 2^{-17} and the rounded 18-bit sine within 1.06 LSB of the exact value (figure 3.1; the exhaustive check over all 2^{24} phases in `tests/test_model.py` gives 0.57 and 1.07). The phase's truncation to 24 bits is a phase error below 2^{-24} cycle (2×10^{-5} degrees). In a sine's spectrum these errors are spurs: 125 dB below a full-scale sine at an awkward frequency (figure 3.3), far below what the 14-bit output allows.

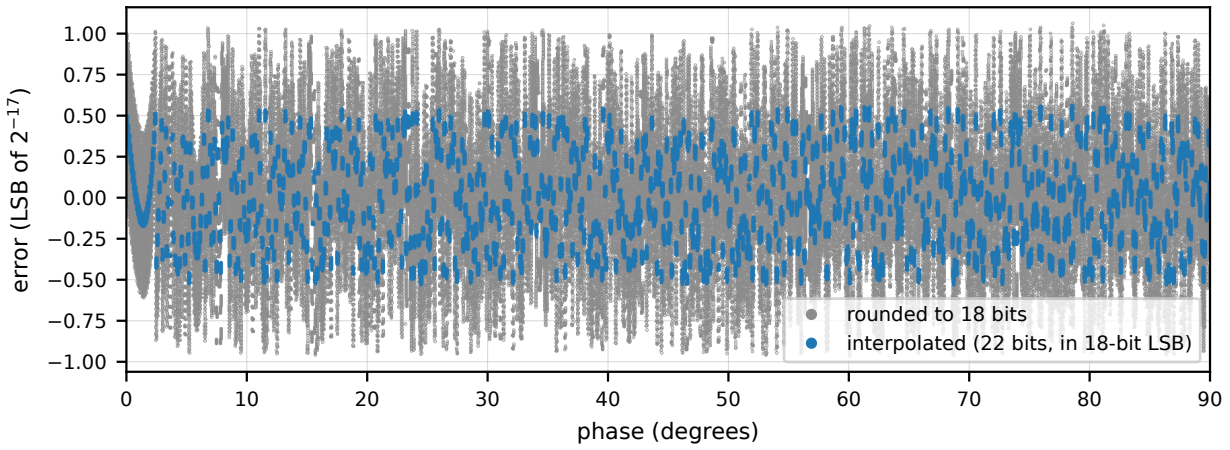


Figure 3.1: The sine's error over a quarter wave (computed with the model at every 37th of the 2^{22} phases): the interpolated 22-bit value (blue, in units of the 18-bit step) and the rounded 18-bit sine (grey).

3.3 Output synthesis

Each output is a sum of the oscillators' sines with their amplitudes, an offset and, in `feedback`, the fast controllers' outputs:

$$c = \mathrm{clamp}\left(\mathrm{sat}_{14}\left\lfloor \frac{\sum_k (A_k + dA_k)\, s_k + (D + U)\cdot 2^{12} + 2^{27} + w}{2^{28}} \right\rfloor,\ c_{\mathrm{lo}},\ c_{\mathrm{hi}}\right) \tag{3.5}$$

(`model.output_code`), where U is the sum of the fast PIDs' outputs and scans aimed at this output, in 2^{-16} DAC steps (`feedback`; zero in `lockin`). The terms:

- A_k, the configured amplitude of oscillator k on this output: a 25-bit word with full scale 2^{24}, so that $A_k = 2^{24}$ with $s_k = S$ is a full-scale sine ($2^{24} \cdot 2^{17}/2^{28} = 2^{13}$ DAC steps). One step of A is 2^{-11} of a DAC step: amplitudes are set to about $0.07\,\mu$V.
- dA_k, the slow PIDs' amplitude target. $A + dA$ is formed in the DSP48's pre-adder, modulo 2^{25}; the driver bounds an amplitude PID's limits so that the sum stays within $\pm(2^{24} - 1)$.
- D, the DC word in 2^{-16} DAC steps: the configured offset, the slow PIDs' offset targets and a *monitor* (any reading, scaled to volts: an analog copy of X, R, θ, a PID's output, ..., updated once per frame).
- w, the dither (below), and 2^{27}, which makes the floor a rounding to the nearest code.
- c_{lo}, c_{hi}, the output's limits: every sample is clamped to them after all the sums, so that no setting and no controller can drive the output beyond them. Clipped samples are counted and flagged. `protect` locks an output's limits until the FPGA is reloaded, a guard for sensitive equipment.

The terms are summed in a cascade of DSP48 blocks, one per oscillator, each computing $(A_k + dA_k)s_k$ and adding the previous block's sum; the whole output stage has a fixed latency of 14 samples from the sample counter to the converter's stream.

3.3.1 Dither

Rounding a slowly varying or periodic value to 14 bits makes an error that repeats with the signal: harmonics and spurs, and dead zones where small changes do not reach the output. *Dither* [5] adds a random value before the rounding so that the error becomes noise, independent of the signal. The designs use triangular-PDF dither of ± 1 DAC step: the sum of two independent uniform values, each $\pm\frac{1}{2}$ step, generated per output and per sample by a xoroshiro128+ generator [6] (two 16-bit uniforms from each 64-bit word). Triangular dither of this width makes the rounding error's mean *and* its variance independent of the signal: the output's average equals the unrounded value exactly, and the noise is white with the variance $\frac{1}{4}$ step^2 ($\frac{1}{12}$ from the rounding and $\frac{1}{6}$ from the dither): 4.8 dB above undithered rounding, the price of removing every signal-dependent artefact.

Figure 3.2 shows a 3.3-step sine at exactly $f_s/64$, whose rounding error repeats every period: without dither its harmonics stand at -27 dB (the third), with dither they vanish into a flat floor. Figure 3.3 shows a full-scale sine at an awkward frequency, where the rounding error is already noise-like: the spurs of the rounded 14-bit code stand 111 dB below the sine, with dither 112 dB, and those of the 18-bit sine itself 125 dB. The DAC's own nonlinearity, which no digital arithmetic can remove and which is not modelled here, is far larger. Dither is worth enabling for small or slowly varying outputs (offsets that a PID moves in sub-step increments, small sines); for large sines at awkward frequencies it changes little.

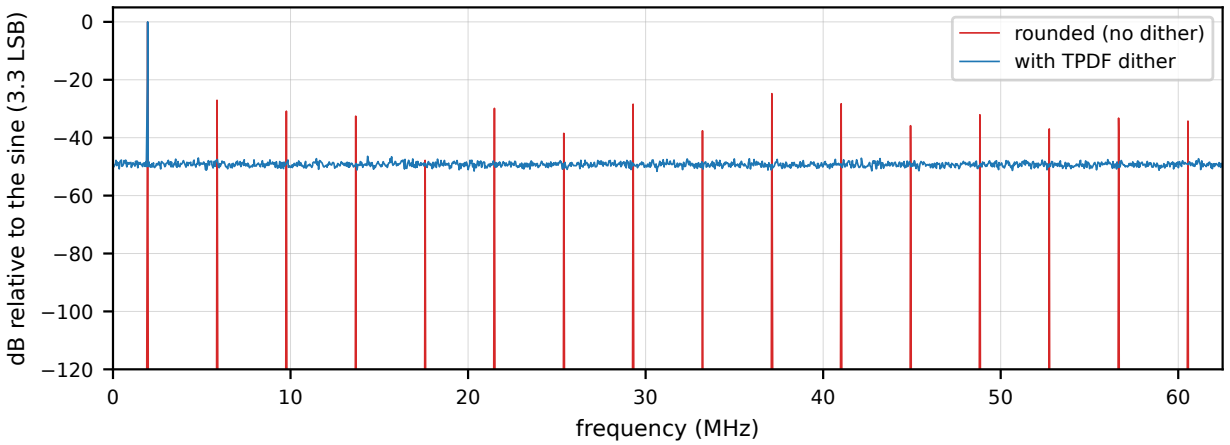


Figure 3.2: A sine of 3.3 DAC steps at $f_s/64$: rounded (red), its error repeats every period and shows as harmonics; with triangular dither (blue) the harmonics turn into a flat floor (computed; powers averaged over 16 records of 2^{14} samples).

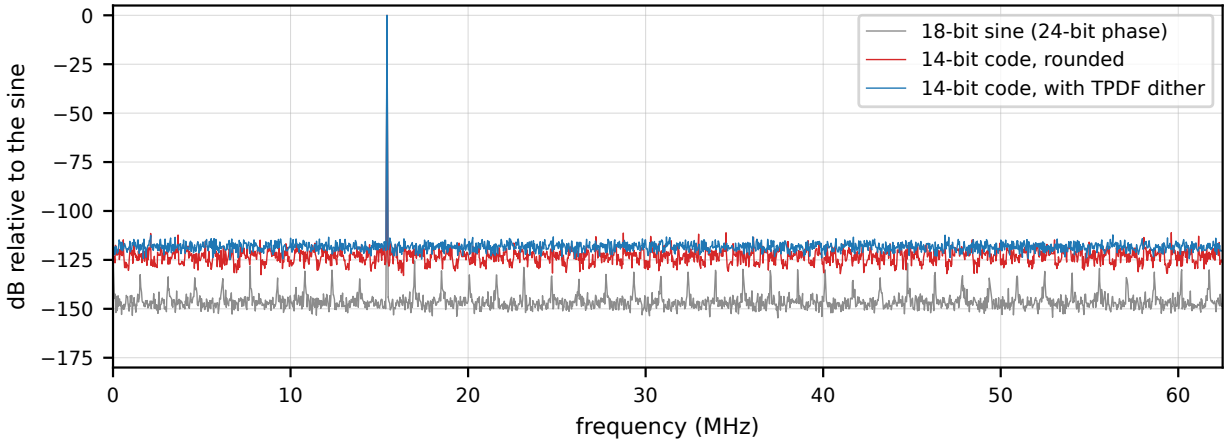


Figure 3.3: A full-scale sine at $0.1234567\,f_s$: the model's 18-bit sine (grey), the 14-bit output code rounded (red) and with dither (blue). Computed over 2^{16} samples with a Kaiser window ($\beta = 20$); the noise floors per bin depend on the record's length, the spurs do not.

3.4 Amplitudes in volts

A full-scale amplitude word 2^{24} gives $\pm 2^{13}$ DAC steps; the driver converts a peak voltage to the word with the output's calibrated gain G, the DAC's droop $\mathrm{sinc}(f/f_s)$ at the oscillator's frequency and the output's calibrated response table:

$$V_{\mathrm{peak}} = A\,\frac{4\,G\,S}{2^{28}}\;\mathrm{sinc}\!\left(\frac{f}{f_s}\right)\,r_{\mathrm{out}}(f), \tag{3.6}$$

where the factor 4 converts the 14-bit code's steps to the calibration's 16-bit units. The realized value (after rounding to the word) is reported back in every configuration readback, so that what the instrument says it does is what it does. Output amplitudes are always given in peak volts; readings follow the setting `units` (rms or peak).

Chapter 4

References and mixing

4.1 The reference phase

Each demodulator d forms its reference from one or two oscillators:

$$\psi_d(n) = \left\lfloor \frac{(m\,\phi_A(n) + n_B\,\phi_B(n)) \bmod 2^{48}}{2^{24}} \right\rfloor + \mathrm{POW}_d \pmod{2^{24}}, \tag{4.1}$$

in units of 2^{-24} cycle (`model.ref_phase`): the *harmonic* $m = 0 \ldots 1023$ of oscillator A, optionally plus or minus oscillator B (the *sideband* $n_B \in \{-1, 0, +1\}$), plus the demodulator's phase offset POW. The products $m\,\phi_A$ are exact 48-bit integers (two DSP48 blocks form the low and high halves), so the reference's phase is exactly m times the oscillator's, at every sample and for ever: a demodulator at the third harmonic of the oscillator that drives an experiment detects the experiment's third-harmonic response with a phase that stays fixed. The reference's frequency is

$$f_{\mathrm{ref}} = m f_A + n_B f_B, \tag{4.2}$$

which may exceed $f_\mathrm{s}/2$: the reference then aliases like any sampled signal, and so does the input component it is compared with, so the product is still right. In practice the board's analog paths limit the signals: the loopback response falls to 0.74 at 55 MHz and 0.33 at 60 MHz (root `README.md`).

Sidebands. With $n_B = \pm 1$ the demodulator detects at the sum or difference of two oscillators: an intermodulation product of two drives, or a modulation sideband (f_A the carrier, f_B the modulation). In a pump–probe experiment, for instance, oscillator A modulates the pump, oscillator B the probe, and a demodulator at $f_A + f_B$ sees only the signal that needs both.

The quadrature pair. The reference phase addresses the sine table twice (`model.ref_iq`):

$$r_I = S\cos\psi, \qquad r_Q = -S\sin\psi, \tag{4.3}$$

from $\psi + \frac{1}{4}$ cycle and ψ. The sign of r_Q makes Y positive for a signal that leads the reference (section 2.4).

4.2 Square references

With the setting `reference = square` the references are $\pm S$ from the reference phase's quadrant: $r_I = S\,\mathrm{sign}\cos\psi$, $r_Q = -S\,\mathrm{sign}\sin\psi$. A square wave of unit amplitude is the Fourier series

$$\mathrm{sign}\cos\psi = \frac{4}{\pi}\sum_{k\ \mathrm{odd}} \frac{(-1)^{(k-1)/2}}{k}\cos(2\pi k\psi), \tag{4.4}$$

so a square reference sees the fundamental $4/\pi$ times larger than a sine reference does, and every odd harmonic of the input with the weight $\pm 1/k$ relative to the fundamental (figure 4.1). The drivers

divide the readings by $4/\pi$, so that a sine at the fundamental reads the same with either reference. A square reference is the classic analog lock-in's (a switching mixer); it is useful when the signal is itself a square wave (a chopped beam), whose odd harmonics, of relative amplitudes $1/k$, then all add to the reading: $\sum_{k\ \mathrm{odd}} k^{-2} = \pi^2/8$ times the fundamental's contribution. With a sine reference only the fundamental is detected: the designs' sine references are clean to 125 dB (section 3.2), so the harmonics' leakage into X and Y is negligible.

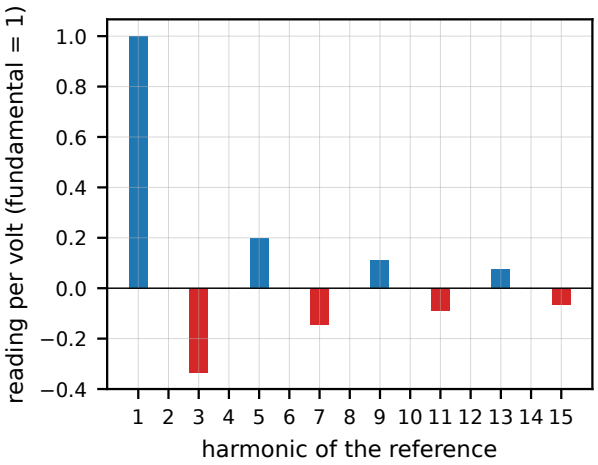


Figure 4.1: The reading of a square reference per volt of input at each harmonic of the reference, relative to the fundamental (the readings are scaled so that the fundamental reads true).

4.3 The mixers

Each demodulator has two mixers, one DSP48 block each, that multiply the conditioned input x (16 bits, chapter 2) by the references (18 bits):

$$p_I(n) = x(n)\, r_I(n), \qquad p_Q(n) = x(n)\, r_Q(n). \tag{4.5}$$

For an input $x = a\sin(2\pi f n/f_\mathrm{s} + \vartheta)$ (in input units) at the reference's frequency, with the reference phase $\psi = fn/f_\mathrm{s}$,

$$p_I = \frac{aS}{2}\left[\sin\vartheta + \sin(4\pi f n/f_\mathrm{s} + \vartheta)\right], \qquad p_Q = -\frac{aS}{2}\left[\cos\vartheta - \cos(4\pi f n/f_\mathrm{s} + \vartheta)\right]: \tag{4.6}$$

a constant, the signal's phasor, and a component at $2f$ that the filters must remove (section 5.2). Everything else in the input — noise, other tones — is moved by f_ref: a component at $f_\mathrm{ref} + \delta$ appears at δ and at $2f_\mathrm{ref} + \delta$. Figure 4.2 shows this on a computed example: the signal, an interferer 60 kHz away and white noise, before and after the multiplication by $\sqrt{2}\,\mathrm{e}^{-\mathrm{i}2\pi f_\mathrm{ref} t}$, with the response of a fourth-order filter that keeps only the signal.

The scale of X and Y. The mixers' products feed the CIC filter (chapter 5), whose output, scaled by 2^{-15}, is the RC filters' input. The CIC's gain over its two frames is 2^{14}, so the filters see the products' mean times $\frac{1}{2}$:

$$X = \frac{S}{4}\, a\cos\theta, \qquad Y = \frac{S}{4}\, a\sin\theta \tag{4.7}$$

in the filters' units (with the drivers' phase convention, $\theta = \vartheta$ referred to OUT1). The drivers convert with the input's calibrated scale: one volt peak at the connector is $a = K/2$ input units, so a reading of X in volts (peak) is $X_\mathrm{units} \cdot 4/(S \cdot K/2)$, divided by $\sqrt{2}$ for rms, by $4/\pi$ for a square reference, and by the input's calibrated amplitude response at f_ref.

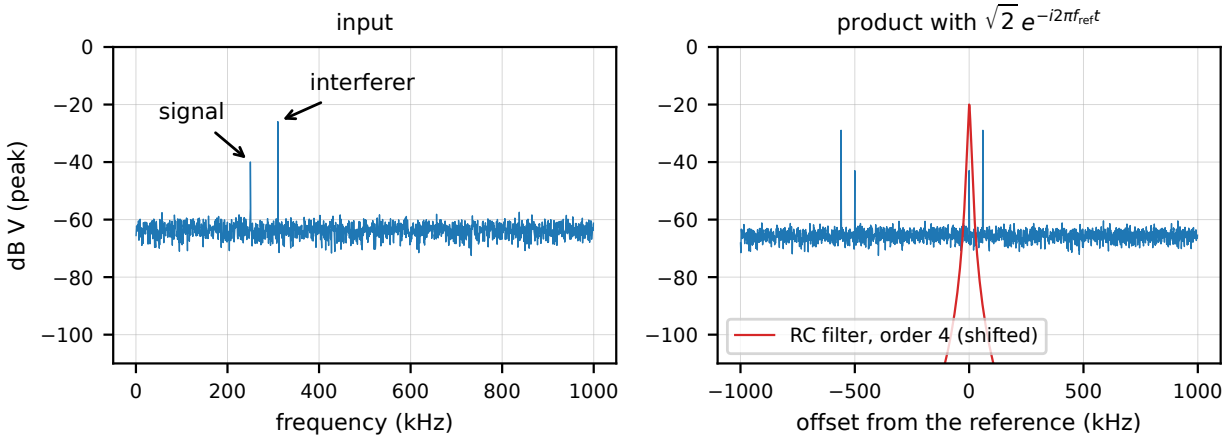


Figure 4.2: Left: an input with a 10 mV signal at 250 kHz, a 50 mV interferer at 310 kHz and white noise. Right: its product with $\sqrt{2}\,\mathrm{e}^{-\mathrm{i}2\pi f_{\mathrm{ref}}t}$: the signal at zero, its image at $-2f_{\mathrm{ref}}$, the interferer at +60 kHz; the red curve is the magnitude of a 4-section RC filter ($\tau = 20\,\mu$s), shifted for display. Computed with floating-point arithmetic to show the principle.

The input of a demodulator. A demodulator's input is IN1, IN2 or the difference IN1 − IN2 (equation (2.2)), selected per demodulator: two demodulators can watch two inputs at one frequency, or one input at two frequencies.

Chapter 5

Decimation: the CIC filter

The mixers make two products per sample per demodulator: 250 million numbers a second for each. The filters that follow need only the products' slow part, so the first step reduces the rate by 128, to one value per frame, with a cascaded integrator-comb (CIC) filter [7]: a decimating filter built from adders alone. (For the sampled-signal background of this and the following chapters, see [8].)

5.1 Integrators and combs

A CIC of order 2 has two integrators running at the full rate and two differences (combs) at the decimated rate:

$$I_1(n) = I_1(n-1) + p(n), \qquad I_2(n) = I_2(n-1) + I_1(n) \qquad \text{(every sample)}, \tag{5.1}$$

$$c_1(k) = I_2(128k+127) - I_2(128k-1), \qquad c_2(k) = c_1(k) - c_1(k-1) \qquad \text{(every frame)}. \tag{5.2}$$

Both integrators wrap modulo 2^{48}: the combs' differences are still exact, because a difference of wrapped values is the wrapped difference, and the result ($|c_2| < 2^{47}$) fits. In the FPGA the first integrator is the mixer's own DSP48 accumulator (the multiplier's output register adds each new product to the previous sum), the second a cascaded DSP48; the frame engine latches I_2 once per frame and runs the combs.

lockin/control/lockin/model.py

```
class Cic:
    """Two integrators (I1 in the mixer's DSP, I2 in the fabric), modulo 2^48, and the
    two combs evaluated per frame; the output is signed 48-bit, gain 2^14."""

    __slots__ = ("i1", "i2", "v_prev", "c1_prev")

    def __init__(self):
        self.i1 = self.i2 = self.v_prev = self.c1_prev = 0

    def push(self, product):
        self.i1 = (self.i1 + product) & MASK48
        self.i2 = (self.i2 + self.i1) & MASK48

    def frame(self):
        v = self.i2
        c1 = (v - self.v_prev) & MASK48
        c2 = (c1 - self.c1_prev) & MASK48
        self.v_prev, self.c1_prev = v, c1
        return signed(c2, 48)
```

Expanding the sums, $c_2(k)$ is the products of the last two frames weighted by a triangle: weight $j+1$ for the j-th sample of frame $k-1$ and $128-j$ for the j-th of frame k (up to the exact indexing),

$$c_2(k) = \sum_{j=0}^{255} w_j \, p(128(k-1)+j), \qquad \sum_j w_j = 128^2 = 2^{14}. \tag{5.3}$$

The triangle is the convolution of two 128-sample boxcars, so its frequency response is the square of a boxcar's:

$$H_{\mathrm{CIC}}(f) = \left[\frac{\sin(128\pi f/f_{\mathrm{s}})}{128\sin(\pi f/f_{\mathrm{s}})}\right]^2 \approx \mathrm{sinc}^2\left(\frac{f}{f_{\mathrm{o}}}\right), \tag{5.4}$$

normalized to 1 at zero frequency (figure 5.1). It has double zeros at every multiple of the frame rate, -7.8 dB at half the frame rate, and a first-sidelobe level of -27 dB. The filters' input is $u = \mathrm{round}(c_2/2^{15})$: the mean of the products times $\frac{1}{2}$.

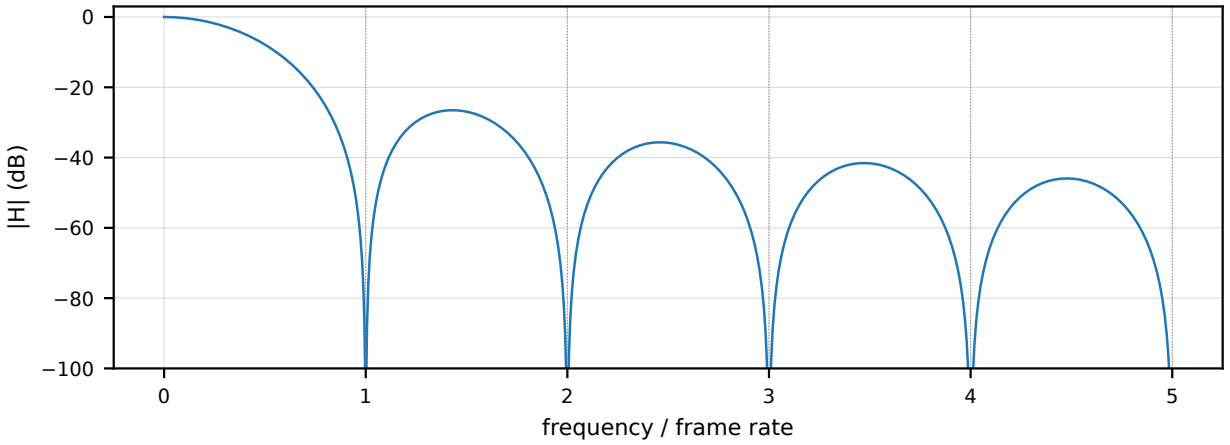


Figure 5.1: The CIC's response (order 2, decimation 128), normalized at zero frequency (computed). Its double zeros at the multiples of the frame rate reject whatever the decimation would fold onto zero frequency.

Decimation and aliasing. Taking one value per frame samples the filtered products at f_{o}: a component at δ appears at δ folded into $\pm f_{\mathrm{o}}/2$. The CIC's double zeros sit exactly where a component would fold onto zero frequency, so components near multiples of f_{o} — which would otherwise alias onto the signal — are suppressed by sinc^2 of their distance from the multiple, about $(\Delta f/f_{\mathrm{o}})^2$ near it. The RC filters after the CIC then remove what aliased elsewhere.

5.2 The $2f$ product

The mixers' products contain, besides the signal's phasor at zero frequency, the image at $2f_{\mathrm{ref}}$ (section 4.3). The CIC attenuates it by $H_{\mathrm{CIC}}(2f_{\mathrm{ref}})$, and after the decimation it lands at $2f_{\mathrm{ref}}$ folded into $\pm f_{\mathrm{o}}/2$, where the RC filters attenuate it again. The residual ripple on X and Y, relative to the signal, is

$$\frac{\text{ripple}}{\text{signal}} = H_{\mathrm{CIC}}(2f_{\mathrm{ref}})\ \left|H_{\mathrm{RC}}(\mathrm{alias}(2f_{\mathrm{ref}}))\right|, \tag{5.5}$$

figure 5.2. Two consequences are useful in practice:

- when $2f_{\mathrm{ref}}$ is a multiple of the frame rate — f_{ref} a multiple of 488.28125 kHz — the image vanishes exactly in the CIC's zeros, and even the shortest filter gives a clean reading (the designs' tests and the loop-delay measurement use such frequencies);
- near those frequencies the image aliases close to zero frequency, where the RC filter passes it: at frequencies just off a multiple of 488.28125 kHz a short filter shows a slow beat on X and Y. A longer time constant or a higher order removes it, as for an analog lock-in at low frequencies.

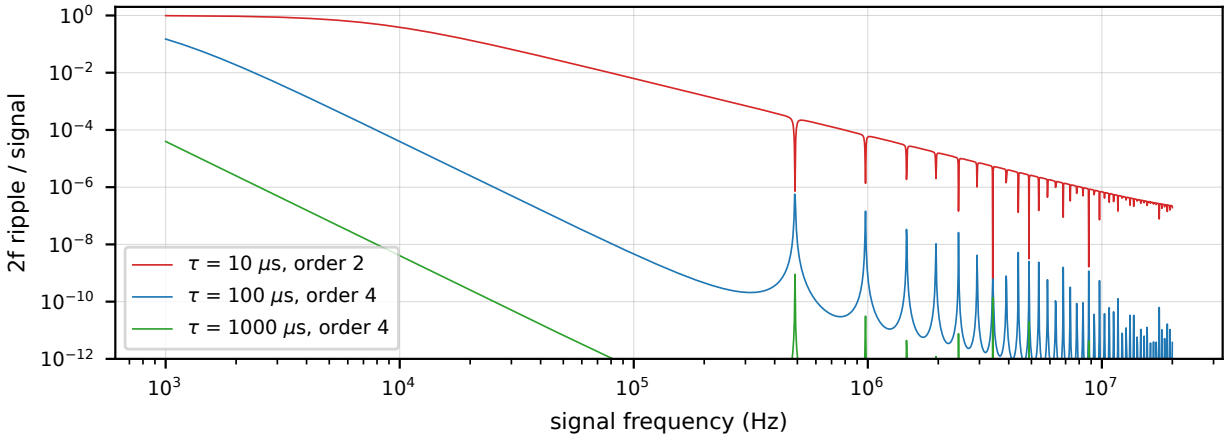


Figure 5.2: The $2f$ product's residue on X and Y relative to the signal, against the signal's frequency, for three filters (computed). The dips at the multiples of 488.28125 kHz are the CIC's zeros; the peaks between them are where the image aliases near zero frequency.

Chapter 6

The RC filters

After the CIC, each demodulator's X and Y pass through a cascade of first-order low-pass sections, the digital equivalent of an analog lock-in's RC filters. Their time constant and order set the measurement's bandwidth, its noise and its response time.

6.1 A first-order section

Once per frame each section moves its state y a fraction α of the way toward its input u:

$$y_k = y_{k-1} + \alpha\,(u_k - y_{k-1}), \qquad H_1(z) = \frac{\alpha}{1-(1-\alpha)z^{-1}}, \quad z = \mathrm{e}^{\mathrm{i}2\pi f/f_\mathrm{o}}. \tag{6.1}$$

This is the exact discrete equivalent of an RC low-pass of time constant τ sampled at f_o when

$$\alpha = 1 - \mathrm{e}^{-1/(\tau f_\mathrm{o})}, \tag{6.2}$$

because then a step decays by $\mathrm{e}^{-1/(\tau f_\mathrm{o})}$ per frame, as $\mathrm{e}^{-t/\tau}$ does in the continuous filter. For $\tau \gg 1/f_\mathrm{o}$, $\alpha \approx 1/(\tau f_\mathrm{o})$; at the shortest setting, $\tau = 0.5\,\mu\mathrm{s}$, $\alpha = 0.87$ and the section is barely a filter.

The arithmetic. The FPGA represents $\alpha = m \cdot 2^{-(17+s)}$ with a 17-bit mantissa $2^{16} \le m < 2^{17}$ and a 6-bit exponent s, which reaches every τ from $0.5\,\mu\mathrm{s}$ to $10^6\,\mathrm{s}$ within a relative error of $2 \cdot 10^{-5}$. The state is a 72-bit number with 40 fraction bits, so that a section with $\alpha \approx 10^{-12}$ (a time constant of a million seconds) still moves: each step adds

$$\Delta y = \mathrm{round}\left(\left\lfloor \frac{u\,2^{40} - y}{2^{32}} \right\rfloor m\,2^{15-s}\right), \tag{6.3}$$

one product of a truncated difference and the mantissa, evaluated by one shared multiplier for all channels in a fixed schedule (`lk_iir.v`, `fb_iir.v`).

lockin/control/lockin/model.py

```
def iir_section(target, y, m, s):
    """One first-order section: y += alpha (target - y), alpha = m 2^-(17+s). target and
    y are state words (value 2^40); the increment is floor(D/2^32) m 2^(15-s), rounded."""
    dt = (target - y) >> D_SHIFT
    return y + shift_round(dt * m, 15 - s)
```

6.2 Cascades

Each channel (X and Y of every demodulator, and each input's DC estimate) has eight sections in series with the same α; the configured *order* $n = 1 \ldots 8$ picks the section whose state is the output

(order 0: the CIC's output itself). Every section adds 6 dB per octave of roll-off: order 4 is the familiar 24 dB/oct. The cascade's response is $H_1(z)^n$; for $\tau \gg 1/f_o$ it approaches the continuous

$$H_n(f) = \frac{1}{(1 + \mathrm{i}2\pi f\tau)^n}, \qquad h_n(t) = \frac{t^{n-1}}{(n-1)!\,\tau^n}\,\mathrm{e}^{-t/\tau}, \qquad s_n(t) = 1 - \mathrm{e}^{-t/\tau}\sum_{j=0}^{n-1}\frac{(t/\tau)^j}{j!}, \tag{6.4}$$

the impulse response an Erlang (gamma) distribution and the step response its cumulative distribution (figure 6.1). The step response's delay grows with the order: the impulse response's mean is $n\tau$, its peak $(n-1)\tau$.

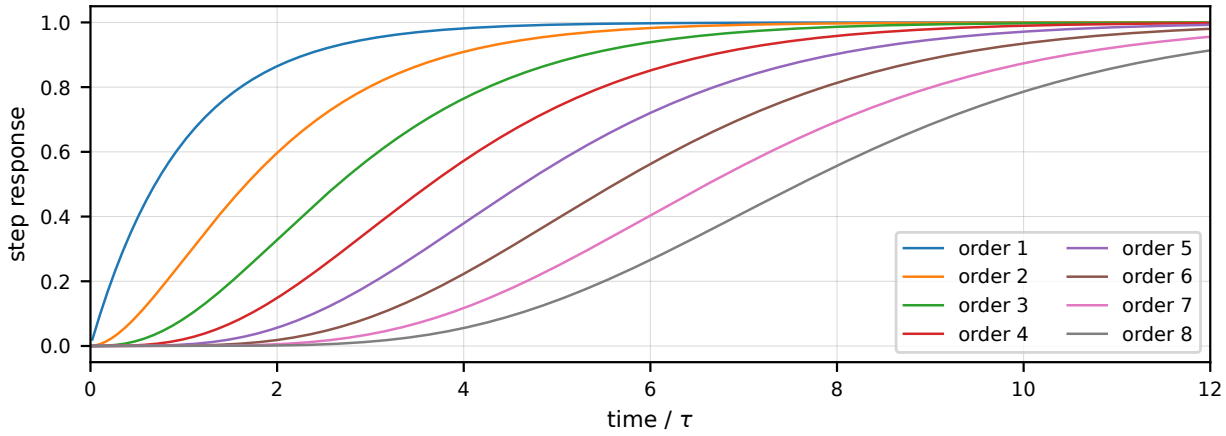


Figure 6.1: Step responses of 1 to 8 sections, computed with the model's arithmetic for $\tau = 50\,\mu$s (49 frames), against t/τ.

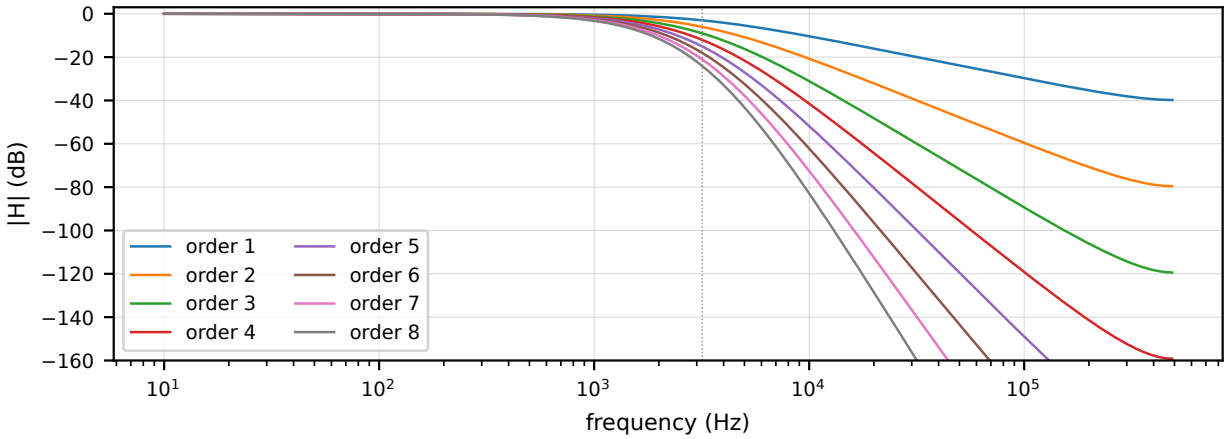


Figure 6.2: Magnitude responses of 1 to 8 sections for $\tau = 50\,\mu$s at the frame rate (computed); the dotted line marks $1/(2\pi\tau)$. Near half the frame rate the discrete sections fall below their continuous counterparts.

6.3 Noise bandwidth and settling

The *equivalent noise bandwidth* is the width of a rectangular filter that passes the same white-noise power:

$$\mathrm{ENBW} = \int_0^\infty |H_n(f)|^2\,\mathrm{d}f = \int_0^\infty \frac{\mathrm{d}f}{(1 + (2\pi f\tau)^2)^n} = \frac{\Gamma(n - \frac{1}{2})}{4\sqrt{\pi}\,\tau\,\Gamma(n)}, \tag{6.5}$$

(substitute $2\pi f\tau = \tan\vartheta$ to reduce the integral to a beta function): $0.2500/\tau$ for $n = 1$, $0.1250/\tau$ for $n = 2$, $0.0781/\tau$ for $n = 4$ and $0.0524/\tau$ for $n = 8$. It is one-sided: the noise of X (or Y) for a white input density e_n is $e_n\sqrt{\mathrm{ENBW}}$ (chapter 8).

The time a reading needs to settle after a step, to a fraction ϵ of the step, solves $1 - s_n(t) = \epsilon$; to 10^{-4} it is $9.21\,\tau$ for order 1, $11.76\,\tau$ for order 2, $15.91\,\tau$ for order 4 and $22.96\,\tau$ for order 8 (figure 6.3). The drivers report it for every demodulator, adding the engine's latency of a few frames (chapter 9), and the sweeps wait for it.

The choice between a higher order and a longer time constant is a trade: for the same noise bandwidth a higher order settles a little later but rejects nearby interferers far better (figure 6.2). Order 4 with τ chosen for the noise is the usual compromise.

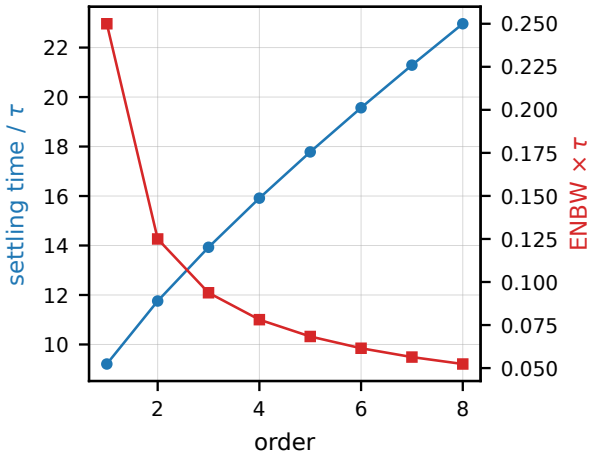


Figure 6.3: Settling time to 10^{-4} (in units of τ) and the noise bandwidth (in units of $1/\tau$) against the order (computed from the formulas above).

Order 0. With order 0 the reading is the CIC's output: a two-frame triangle average, the fastest response the instrument has (about $2\,\mu$s), with the CIC's own response (figure 5.1) as its only filter.

The inputs' DC estimates. The same engine runs one more channel per input: the input's frame value (the CIC of $2c$) through the sections with the AC coupling's corner (section 2.2.1).

Chapter 7

Magnitude and phase

7.1 The CORDIC

The magnitude and phase of each demodulator's (X, Y) come from one CORDIC core [9, 10] shared by the demodulators in turn, once per frame. In its *vectoring* mode a CORDIC rotates the vector (x, y) toward the x axis by a sequence of angles $\pm\arctan 2^{-i}$, each a shift and an add:

$$x_{i+1} = x_i + \sigma_i\, 2^{-i} y_i, \qquad y_{i+1} = y_i - \sigma_i\, 2^{-i} x_i, \qquad z_{i+1} = z_i + \sigma_i \arctan 2^{-i}, \qquad \sigma_i = \operatorname{sign} y_i. \tag{7.1}$$

After N iterations $y_N \approx 0$, z_N is the vector's angle and $x_N = K\sqrt{x^2 + y^2}$ with the gain

$$K = \prod_{i=0}^{N-1} \sqrt{1 + 2^{-2i}} \approx 1.6468. \tag{7.2}$$

The designs use 26 iterations, angles in units of 2^{-32} cycle (a table of 26 rounded arctangents), and a pre-rotation by half a cycle for vectors with $x < 0$, so that the iterations converge in all four quadrants (`model.cordic`). The drivers divide R by K.

Normalization. X and Y span a huge range: a nanovolt signal and a volt signal differ by 10^9. A CORDIC working on fixed-width numbers would lose the small ones' phase in its rounding. Before the core, both are shifted together by

$$\mathrm{sh} = L - 29, \tag{7.3}$$

where L is the bit length of the larger magnitude, so that the core always works with 29 significant bits; R is shifted back afterwards. The phase is then equally accurate for signals of a few nanovolts and of volts, limited only by the readings' own noise.

lockin/control/lockin/model.py

```
    a = (x if x >= 0 else ~x) | (y if y >= 0 else ~y)
    sh = a.bit_length() - CORDIC_BITS
    xn, yn = (x >> sh, y >> sh) if sh >= 0 else (x << -sh, y << -sh)
    z = 0
    if xn < 0:
        xn, yn, z = -xn, -yn, 1 << 31
    for i in range(CORDIC_ITER):
        if yn >= 0:
            xn, yn, z = xn + (yn >> i), yn - (xn >> i), z + ATAN[i]
        else:
            xn, yn, z = xn - (yn >> i), yn + (xn >> i), z - ATAN[i]
```

Accuracy. The last rotation is $\arctan 2^{-25} \approx 3 \times 10^{-8}$ rad, and the 29-bit operands resolve 2^{-29} of the vector's length: the phase error is a few microdegrees. For 20 000 random vectors spanning twelve decades of magnitude the model's θ is within 2.7 μdegrees of atan2 (1.0 μdegrees rms), and R/K within 4.6e-06 of $\sqrt{x^2 + y^2}$ for vectors above a few hundred units (figure 7.1): far below any physical signal's noise.

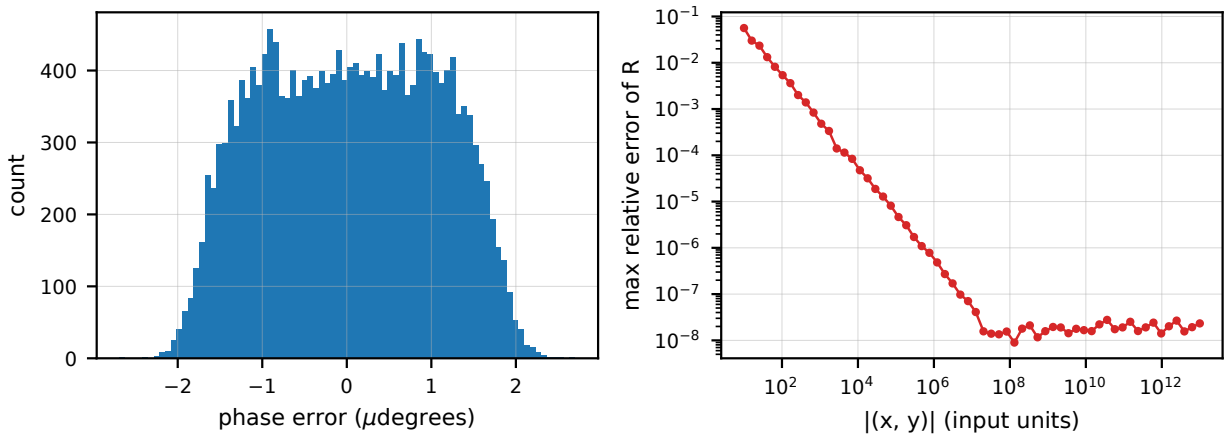


Figure 7.1: The CORDIC's phase error for 20 000 random vectors (left) and the largest relative error of R over 64 angles against the vector's magnitude (right), computed with the model. Small vectors' errors are the integers' own rounding.

7.2 Phase, unwrapped phase and frequency deviation

θ is a 32-bit fraction of a cycle, so it wraps at $\pm 180°$. Two derived quantities follow it across wraps:

- the *phase change* per frame, $d\theta_k = \text{signed}_{32}(\theta_k - \theta_{k-1})$, the difference wrapped to $\pm\frac{1}{2}$ cycle: correct as long as the phase moves by less than half a cycle per frame, i.e. for a signal within $\pm f_o/2 = \pm 488\,\text{kHz}$ of the reference;
- the *unwrapped phase*, the running sum of $d\theta$ in 48 bits (± 32768 cycles).

$d\theta$ per frame is the *frequency deviation* of the signal from the reference:

$$\Delta f = d\theta \, \frac{f_o}{2^{32}}, \qquad \text{resolution } \frac{f_o}{2^{32}} = 0.23\,\text{mHz}, \tag{7.4}$$

updated every frame. The phase detector of a phase-locked loop is θ itself, wrapped to $\pm 180°$; a frequency-locked loop uses $d\theta$ (chapter 11). The unwrapped phase keeps every cycle since the FPGA was loaded: useful to follow a slow drift over many cycles, but a loop on it first winds back the cycles that accumulated before it started.

Both are computed from the filtered X and Y: they inherit the filter's delay and bandwidth. A signal far outside the filter's band is attenuated, but its phase still turns: $d\theta$ reads its frequency offset as long as the signal stays above the noise, which is how the frequency-locked loop acquires signals far from the reference (section 11.5).

Chapter 8

Noise and resolution

A lock-in's purpose is to measure a signal smaller than the noise around it. This chapter quantifies how well it does: the noise of X, Y, R and θ, how the noise density around the reference is measured, how the time constant is chosen from the signal, and how long-term stability is characterized.

8.1 The noise of X and Y

Take the complex view of section 1.1: with rms outputs,

$$z = X + \mathrm{i}Y = \sqrt{2}\,\mathrm{LPF}\left[v(t)\,\mathrm{e}^{-\mathrm{i}\psi(t)}\right]. \tag{8.1}$$

Let the input's noise have the one-sided power spectral density $S_v(f)$ (V^2/Hz). The product with $\mathrm{e}^{-\mathrm{i}\psi}$ moves the spectrum by f_{ref}; a real signal's two-sided density is $S_v/2$, and the factor $\sqrt{2}$ doubles the power, so

$$S_z(\delta) = S_v(f_{\mathrm{ref}} + \delta)\,|H(\delta)|^2 \qquad \text{(two-sided, } \delta \text{ from the reference)}: \tag{8.2}$$

the two-sided density of z at an offset δ is the input's one-sided density at $f_{\mathrm{ref}} + \delta$, shaped by the filter H. For a white input, $S_v = e_n^2$, the variance of z is $e_n^2 \int_{-\infty}^{\infty} |H|^2 \mathrm{d}\delta = 2e_n^2\,\mathrm{ENBW}$, shared equally by X and Y:

$$\sigma_X = \sigma_Y = e_n\sqrt{\mathrm{ENBW}} \qquad \text{(rms units; } \times\sqrt{2} \text{ in peak units)}, \tag{8.3}$$

with the one-sided noise bandwidth (6.5). A time constant of 100 ms at order 4 ($\mathrm{ENBW} = 0.78\,\mathrm{Hz}$) turns an input density of $10\,\mathrm{nV}/\sqrt{\mathrm{Hz}}$ into 8.8 nV rms on X; figure 8.1 shows the general case. The signal-to-noise ratio grows as $\sqrt{\tau}$: four times the time constant halves the noise.

Dynamic reserve. An analog lock-in's front end clips when the noise or an interferer is much larger than the signal. Here the input's 14-bit converter is the only place where that can happen: the arithmetic after it has room for any input that the converter accepts (the mixers' products, the CIC's 48 bits and the filters' 72-bit states never overflow for an unclipped input). An interferer may therefore fill the input's range while the signal is millivolts below; the resolution then rests on the filter, and the reserve is set by the converter's range and its own spurs. Clipped samples are counted and flagged (section 2.2): a clipped input invalidates the measurement.

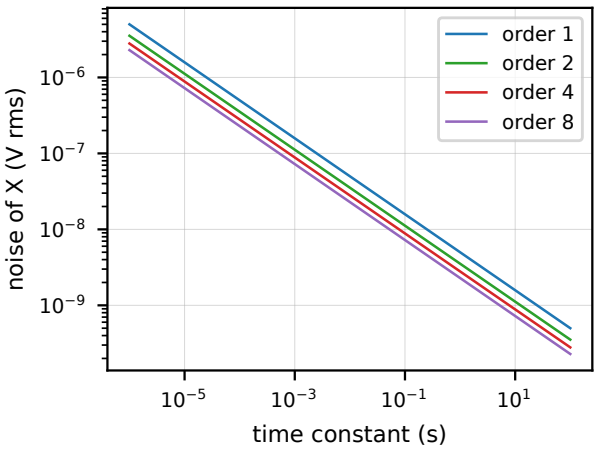


Figure 8.1: The rms noise of X for a white input of $10\,\mathrm{nV}/\sqrt{\mathrm{Hz}}$, against the time constant, for orders 1, 2, 4 and 8 (computed from (8.3)).

8.2 Magnitude and phase at low signal-to-noise

$R = \sqrt{X^2 + Y^2}$ is not a linear function of the input: noise biases it. With a signal R_0 and Gaussian noise σ in each of X and Y, R follows the Rice distribution [11]; its mean exceeds R_0, approximately as $\sqrt{R_0^2 + \sigma^2}$ at high signal-to-noise (its rms is exactly $\sqrt{R_0^2 + 2\sigma^2}$), and tends to $\sigma\sqrt{\pi/2}$ for no signal at all (figure 8.2). The phase's standard deviation is σ/R_0 radians at high signal-to-noise and approaches the uniform distribution's 104° at none.

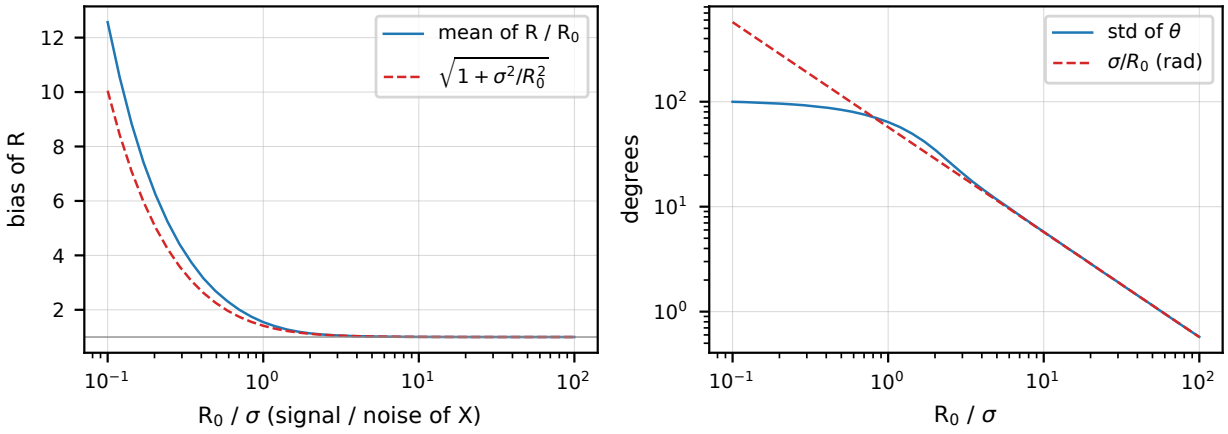


Figure 8.2: Left: the mean of R over its true value against the signal-to-noise ratio R_0/σ (Monte Carlo, 200 000 samples per point) and the high-SNR approximation. Right: the standard deviation of θ and σ/R_0.

The practical rule: *average X and Y, not R.* The vector average converges to the signal, the average of R to its biased mean. When the signal is near the noise, set the phase so that the signal lies on X (auto-phase) and measure X; Y then carries only noise, an estimate of σ.

8.3 The spectrum around the reference: a zoom FFT

Equation (8.2) makes z a window onto the input's spectrum around f_{ref}:

- a tone of rms amplitude A at $f_{\mathrm{ref}} + \delta$ is $z = A\,\mathrm{e}^{\mathrm{i}(2\pi\delta t + \theta)}$, a single line at $+\delta$ (a positive offset for a tone above the reference);
- noise appears with its density at that frequency.

The Fourier transform of a record of z is therefore a spectrum of the input around f_{ref}, whose span and resolution are set by z's sampling, not by f_{ref}: a *zoom FFT*. A 10 MHz signal's neighbourhood can be examined in 0.1 Hz bins, which a direct FFT of the input would need a record of 1.25 billion samples for.

Sampling and aliasing. The designs' capture records X and Y once every 2^j frames ($f_{\mathrm{o}}/2^j$), one frame's reading per word, without averaging between the words. The filter must therefore have removed what lies beyond half that rate, or it folds back. `analysis.zoom_spectrum` picks the rate from the span asked for (by default 8 times the filter's noise bandwidth) and reports $|H|^2$ at half the rate as `aliasing`.

The estimate. Each record of N words is multiplied by a Blackman–Harris window w [12], Fourier transformed, and the powers of several records are averaged (Welch's method [13]). For a rate r:

$$\text{density} = \sqrt{\frac{\overline{|Z_k|^2}}{r\sum w^2}}, \qquad \text{amplitude} = \frac{\sqrt{\overline{|Z_k|^2}}}{\sum w}, \qquad \text{bin width} = \frac{r}{N}, \quad \text{bin ENBW} = \frac{r\sum w^2}{(\sum w)^2}\,: \tag{8.4}$$

a white density reads true in every bin, a tone's amplitude in the bin that holds it. Dividing by $|H(\delta)|$ (`compensate`) undoes the demodulator's filter, so that the result is the input's own spectrum: reliable where $|H|^2$ is well above the arithmetic's floor (figure 8.3).

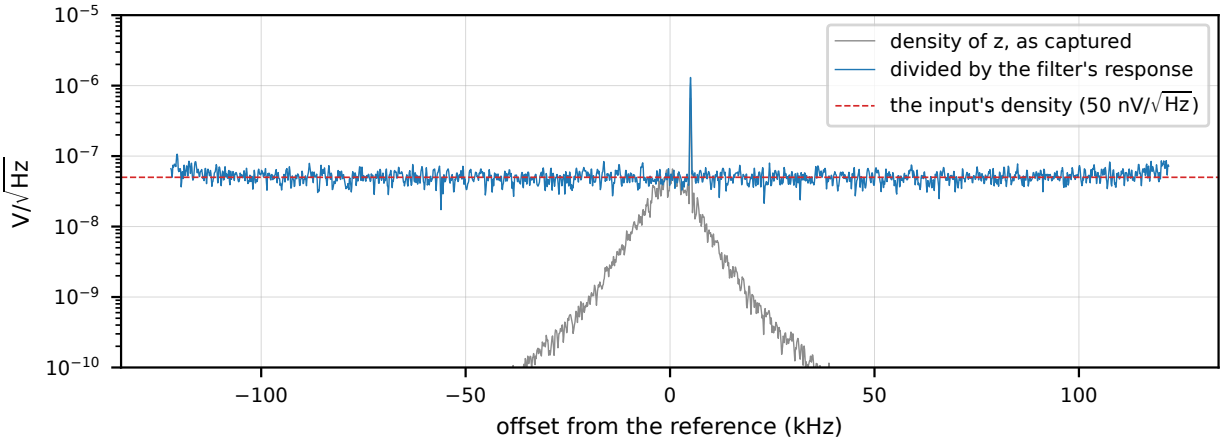


Figure 8.3: `lockin`'s `zoom_spectrum` on a computed signal: a 20 μV tone 5 kHz above the reference in white noise of 50 nV/$\sqrt{\text{Hz}}$, through a 4th-order filter of $\tau = 20\,\mu$s, sampled at $f_{\mathrm{o}}/4$ (2048 words, 8 averages, bins of 119 Hz). Grey: the density of z as captured; blue: divided by the filter's response.

8.3.1 The noise density at the reference

To measure the input's noise around the reference, `analysis.noise_density` removes each record's mean (the signal, which the window's −92 dB sidelobes could otherwise let into the noise of a high-precision measurement), and takes the *median* of the compensated power density over the offsets within ±ENBW/2, excluding ±4 bins around zero. The median ignores spurs and leftover lines, but it is biased: a bin's power averaged over K records is χ^2-distributed with $2K$ degrees of freedom, whose median lies below its mean by the factor

$$m_K: \quad P(K,\, K\,m_K) = \tfrac{1}{2}, \qquad m_1 = \ln 2 = 0.693,\ m_2 = 0.839,\ m_4 = 0.918,\ m_8 = 0.959, \tag{8.5}$$

(P the regularized lower incomplete gamma function; `median_factor`), which the estimate divides out. A second, independent estimate follows from (8.3): $\sqrt{(\sigma_X^2 + \sigma_Y^2)/(2\,\text{ENBW})}$ (`density_xy`). On the board both agree to 1.4 % at 1 MHz (83 and 82 nV/$\sqrt{\text{Hz}}$, measured 2026-10-27).

8.3.2 A time constant from the signal

With the noise density e_n and the signal R measured, the shortest time constant that gives R to a relative precision p follows from (8.3) and (6.5):

$$\mathrm{ENBW} \le \left(\frac{pR}{e_n}\right)^2 \quad\Longrightarrow\quad \tau = \frac{\Gamma(n-\tfrac{1}{2})}{4\sqrt{\pi}\,\Gamma(n)} \left(\frac{e_n}{pR}\right)^2 \tag{8.6}$$

at the chosen order n (`analysis.auto_tau`). On the board a 0.14 V signal at 1 MHz in 82 $\mathrm{nV}/\sqrt{\mathrm{Hz}}$ asks for $\tau = 3.1\,\mu\mathrm{s}$ at order 4 for $p = 10^{-4}$; the readings' scatter was then 1.25×10^{-4} of R (measured 2026-10-27). The rule assumes white noise within the filter's band: drifts and $1/f$ noise call for the Allan deviation instead.

8.4 Stability: the Allan deviation

Over long times a measurement drifts: the signal source, the cables, the converters' temperature. Whether averaging longer still helps is answered by the *Allan deviation* [14–16]. For samples y_i taken at a rate f_s, each the mean over $1/f_s$ (such as the readings in the data stream, or a capture of X),

$$\sigma_y^2(\tau = m/f_s) = \frac{1}{2}\left\langle (\bar{y}_{j+m} - \bar{y}_j)^2 \right\rangle_j, \qquad \bar{y}_j = \frac{1}{m}\sum_{i=j}^{j+m-1} y_i, \tag{8.7}$$

half the mean squared difference of adjacent averages over τ, here with every possible start j (the *overlapping* estimator; `allan_deviation`, from cumulative sums). Unlike the standard deviation it converges for drifting data, and its dependence on τ identifies the kind of fluctuation (figure 8.4):

fluctuation	spectral density	$\sigma_y(\tau)$
white noise	flat	$\propto \tau^{-1/2}$ (averaging helps)
flicker $(1/f)$ noise	$\propto f^{-1}$	flat (averaging no longer helps)
random walk	$\propto f^{-2}$	$\propto \tau^{+1/2}$
linear drift D	—	$D\tau/\sqrt{2}$

The minimum of $\sigma_y(\tau)$ is the longest useful averaging time: a time constant beyond it only adds drift. The estimate at the longest τ rests on few independent differences (the result reports how many were averaged); a record should span ten or more of the longest τ of interest.

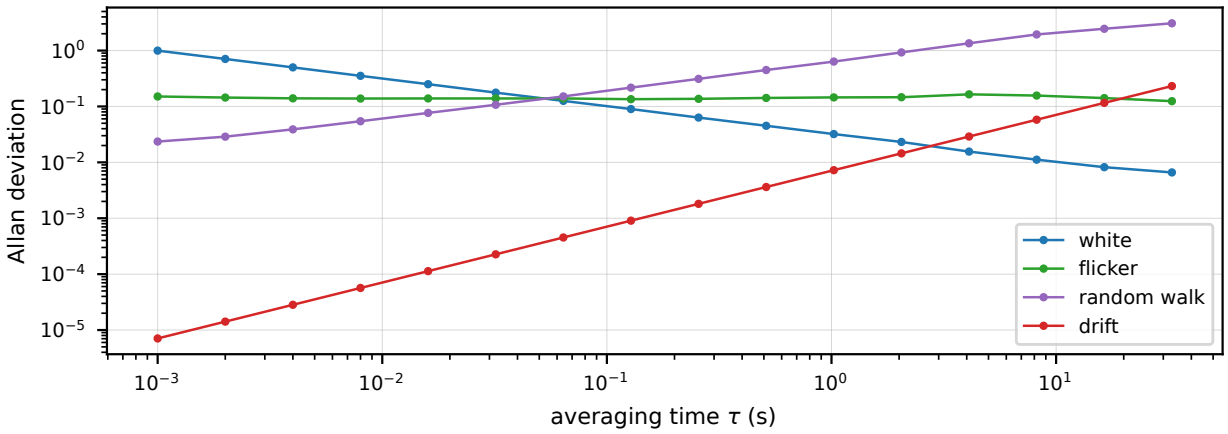


Figure 8.4: Allan deviations of four computed fluctuations at 1000 samples/s, with `lockin`'s `allan_deviation`: white noise ($\tau^{-1/2}$), flicker noise (flat), a random walk ($\tau^{1/2}$) and a linear drift (τ).

Running statistics. `RunningStats` keeps the count, mean, standard deviation and extremes of values as they arrive, with Welford's update [17], which stays accurate for long series with a large mean: for each new value v, $\delta = v - \bar{v}$, $\bar{v}$ += δ/n, M_2 += $\delta\,(v - \bar{v})$, $s^2 = M_2/(n-1)$.

Chapter 9

The frame engine

Everything slower than a sample runs in the *frame engine*: one sequential processor per design that does a fixed list of work in each frame of 128 samples. Knowing its schedule explains every latency the instruments have and why their results are consistent.

9.1 Periods and latencies

During the *period* P (the 128 clocks of samples $128P \dots 128P + 127$) the engine

1. finishes the CIC of frame $P - 1$ (its integrators were latched at the period's start) and runs the RC sections of every channel on it;
2. converts frame $P - 2$'s filtered X, Y to R and θ (the CORDIC, one demodulator after another), and derives $d\theta$ and the unwrapped phase;
3. runs the PIDs (and in `feedback` the lock acquisitions) on frame $P - 3$'s results, and forms the next targets;
4. writes the readings, the stream's row and the capture's frame-rate values.

Each stage meets its deadline within its period, so the pipeline has a fixed latency: *frame k's results act from sample* $128(k + 4)$ (`L_FRAMES` $= 4$), 4.1 μs after the frame's start and about 3.1 μs after its end (figure 9.1). A PID's target computed from frame k changes the outputs from sample $128(k + 4)$ on, and they leave the core 14 samples later.

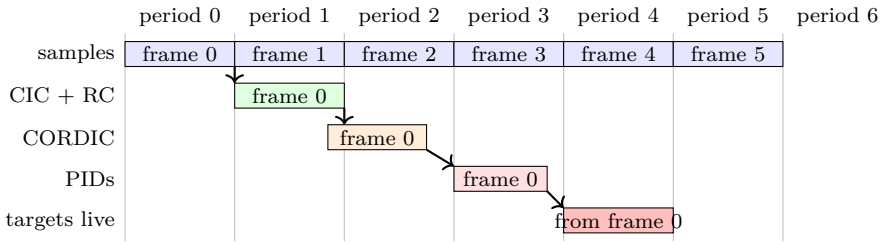


Figure 9.1: The engine's pipeline for one frame: its samples arrive during period 0; the CIC and RC sections run in period 1, the CORDIC from the end of period 1 into period 2, the PIDs in period 3; their targets act from the start of period 4 (sample $128 \cdot 4$). Every frame follows the same path one period later.

The engine is a fixed schedule of 128 clocks, not a processor with a program: each clock does a predetermined step (a coefficient read, a multiplication, an addition, a conversion), so the timing never varies. Its slots and memories are tabulated in each design's `docs/THEORY.md`; the bit-exact model computes the same results at the frame level, and the bench `tb_engine` checks every frame's targets and readings against it.

9.2 Configuration and the apply

A configuration change must reach dozens of registers in two clock domains of the design (the full-rate part and the engine) without a moment where some have the old and some the new values. The designs double-buffer everything:

- the full-rate settings (oscillators, references, outputs, inputs, fast PIDs, scans, slow outputs, pins) are written into *pending* registers;
- the engine's coefficients (filters, PIDs, acquisitions, monitors, the stream) live in a block RAM with two halves: writes go to the pending half.

An *apply* makes all of them live together at a frame boundary: each full-rate stage loads its pending words a fixed number of clocks after the boundary's reference clock, chosen so that every stage switches on the same sample; the engine's bank flips at the same boundary, and its new pending half is refilled from the active one (512 clocks). Register writes wait during the apply's few clocks. `APPLY_NOW` takes the next frame boundary at least a short lead ahead (2 samples in `lockin`; 13 in `feedback`, whose oscillators run 11 samples ahead); `APPLY_AT` takes a given sample, rounded up to a frame boundary, and counts an apply that comes too late. Every apply increments an apply count, which the data stream's rows carry so that a recording shows where the configuration changed.

The drivers apply every configuration change this way. A coherent apply additionally preloads the oscillators' phases (section 3.1).

9.3 Readings

The snapshot. A request (CTRL SNAP) takes the next frame that the engine latches: its X, Y, R, θ, $d\theta$ and unwrapped phase of every demodulator, the PIDs' outputs, errors, integrators and flags, the oscillators' live frequencies (with the PIDs' corrections), the inputs' DC estimates and the frame number — all of one frame, readable when the request's pending flag clears, within about four frames. `readings()` is one such snapshot, decoded into physical units; a PID's error and the demodulator it reads are therefore always of the same frame.

Floats. The engine converts its readings to IEEE single-precision floats as it writes them (one shared converter, three clocks per value): a signed integer v with 16 fraction bits becomes the float $v/2^{16}$ with its mantissa truncated toward zero (`model.f32_bits`), a relative resolution of $2^{-24} \approx 6 \times 10^{-8}$. Phases, frequency deviations and the integrator's flags are passed as raw integers.

The readings ring. Every frame's readings are also written into a ring of four frames; the capture's frame-rate sources and the data stream's rows read from it, so that the stream and the capture carry exactly the readings the snapshot would have shown.

Monitors. An output's DC word can carry a reading (any of X, Y, R, θ, $d\theta$ of a demodulator, a PID's output or error) scaled to volts and saturated: an analog copy, updated once per frame and as late as a PID's target — for an oscilloscope or another instrument.

Chapter 10

The slow PID controllers

Each design has four PID controllers in its frame engine, computed once per frame ($f_o = 976.5625\,\text{kHz}$). They close loops around the instrument's own measurements: an oscillator's frequency locked to an input's phase (a phase-locked loop, chapter 11), an output's amplitude holding a demodulated amplitude constant, an output's offset following a resonance, a slow analog output driving a heater from a slow input [feedback].

10.1 The controller

A PID reads one *input* quantity v of frame k: X, Y, R, θ, $d\theta$ or the unwrapped phase of a demodulator, or [feedback] an input's frame mean, a slow analog input or the TTL reference's phase. In register units it computes (`model.Pid`)

$$e_k = \text{sat}_{48}(v_{\text{set}} - v_k) \qquad (\theta\text{, the TTL phase: } e_k = \text{signed}_{32}(v_{\text{set}} - v_k)), \tag{10.1}$$

$$P_k = k_p e_k, \qquad I_k = \text{clamp}(I_{k-1} + k_i e_k,\ \text{lo, hi}), \qquad D_k = D_{k-1} + 2^{-k_{df}}\left(k_d(e_k - e_{k-1}) - D_{k-1}\right), \tag{10.2}$$

$$u_k = \text{clamp}(P_k + I_k + D_k,\ \text{lo, hi}). \tag{10.3}$$

- The error of a phase wraps to $\pm\frac{1}{2}$ cycle: the phase detector of a PLL, which locks from any starting phase.
- The integrator is clamped to the output's limits: *anti-windup*. While the output is limited, the integrator stops at the limit instead of growing, so the loop recovers at once when the error changes sign.
- The derivative is low-passed by one section of $\alpha = 2^{-k_{df}}$ (`d_filter`), which limits its high-frequency gain.
- The integrator keeps 16 more fraction bits than the output, so very small integral gains still integrate.

Gains are 18-bit mantissas times 2^{e-48} ($|k| < 2^{17}$ register units; the gain stage multiplies and shifts right only). The driver converts the physical gains into these words with the input's and the target's scales: p in target units per input unit, i in target units per input unit per second ($k_i = i \cdot \text{scale}/f_o$), d in seconds ($k_d = d \cdot \text{scale} \cdot f_o$); a gain beyond the range is refused with the limit in physical units. The realized gains are reported back.

Targets. The output u is in the target's units: an oscillator's frequency (Hz; added to its frequency word, $2^{-48} f_s$ per unit) or phase (degrees), an output's amplitude of an oscillator (volts), an output's offset (volts), or [feedback] a slow analog output. PIDs with the same target add: two loops can share an actuator (a fast and a slow one, for instance). The targets of frame k act from sample $128(k+4)$ (chapter 9).

Hold, reset, lock. A held PID (manually, while the demodulator's R is below a level `hold_below` — a lost signal — or [feedback] while its gating acquisition does not lock) keeps its integrator and outputs it alone: P and D are zero. A reset clears the integrator and the derivative at the next frame; a disabled PID outputs zero and clears its state. *Lock detection*: $|e| \leq$ `lock_tolerance` for 2^w frames in a row sets the PID's locked flag, which the readings, the stream's rows and the status show, and which [feedback] a status pin can carry to other equipment.

The arithmetic's schedule. The engine computes each PID in a 27-step micro-sequence: its 15 coefficient words read in steps 0–14, one shared gain stage for P, I and D, an adder per step for the sums and comparisons, the state update in step 22 and its target's sum in step 23. PID $p+1$ starts 22 steps after PID p; the four take $88 + 5$ clocks of the period.

10.2 The loop

A slow PID's loop runs at the frame rate. Its error of frame k acts on the target four frames later, so in the frame-rate z-domain ($z = \mathrm{e}^{\mathrm{i}2\pi f/f_\mathrm{o}}$) the controller is

$$C(z) = \left(k_p + \frac{k_i}{1 - z^{-1}} + D(z)\right) z^{-4} \tag{10.4}$$

and the plant $P(z)$ is the target's effect on the input quantity: the experiment's own response followed by the demodulator's filters (the CIC and the RC sections, chapters 5 and 6) when the input is a demodulated quantity. The loop gain is $L = CP$ and the closed loop's response to a setpoint step is $T = L/(1 + L)$ (chapter 14).

The sampled plant. A target is held constant over each frame, from the sample where it acts, and reaches the input d samples later: the core's 14 samples of output latency and the converters' loop delay (about 15 samples, 120 ns, with a short cable; section 2.4), plus whatever the experiment adds. The CIC's frame value j is a triangle-weighted mean centred on frame j's first sample, so a target changed for one frame enters the frame values with the weights of a quadratic B-spline shifted by d: for $d = 29$ samples 0.3021, 0.6731, 0.0248 in the frames m, $m + 1$, $m + 2$ (figure 11.2, section 11.2). An experiment with a flat response in the band therefore has, read through a demodulator,

$$P(z) = K \left(h_0 + h_1 z^{-1} + h_2 z^{-2}\right) H_\mathrm{RC}(z), \tag{10.5}$$

and the figures below use this exact form; `feedback`'s tuning measures it instead (chapter 14).

The demodulator's filter usually dominates the plant: the loop cannot be faster than the measurement it acts on. Figure 10.1 shows the largest crossover frequency that keeps 60° of phase margin and 10 dB of gain margin when the plant is a demodulator's filter alone. With an integral-only controller the crossover stays a fraction of the filter's corner $1/(2\pi\tau)$; a PI controller can cancel the dominant pole of an order-1 or order-2 filter and cross over above it, until the four frames of delay and the CIC limit the loop near 18 kHz. Higher filter orders, which reject the $2f$ product and interferers better, cost loop bandwidth: with order 4 the crossover stays below the corner.

The margins decide the transient: figure 10.2 shows setpoint steps of PI loops on an order-2 filter of $\tau = 20\,\mu$s designed for three phase margins. 60° is the usual compromise between speed and overshoot.

Typical loops.

Amplitude stabilization: input R, target an output's amplitude: holds a resonance's or a transmission's amplitude constant; p and i are in volts of output per volt of R.

Offset following: input X (the in-phase part of a dispersive signal, which crosses zero at the resonance), target an output's offset: keeps a resonance centred, a side-of-fringe or Pound–Drever–Hall-style lock at the frame rate.

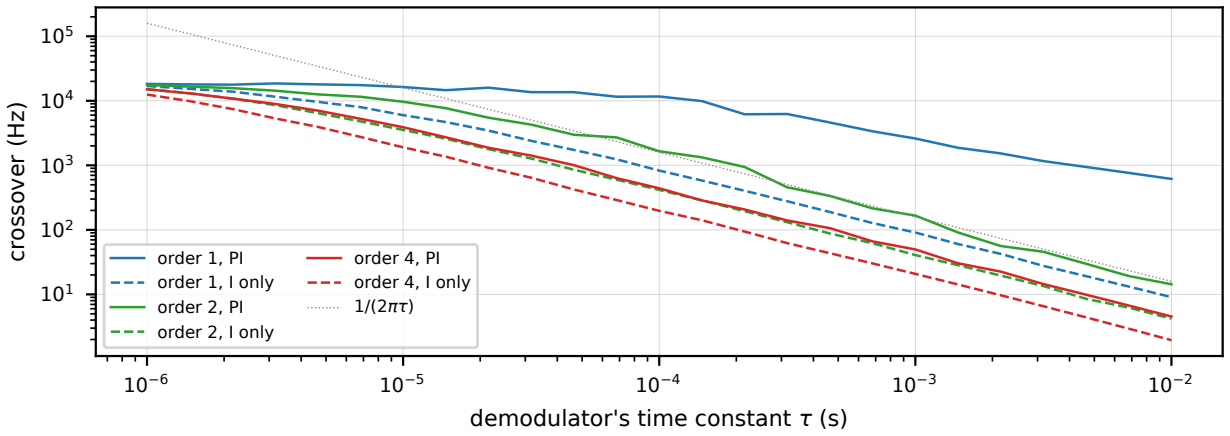


Figure 10.1: The largest crossover of a slow PID loop with 60° phase margin and 10 dB gain margin, against the demodulator's time constant, for filters of order 1, 2 and 4 and for PI (solid) or integral-only (dashed) control (computed with `feedback`'s `design_pi` on the sampled plant above, $d = 29$ samples, with the controller's four frames of delay).

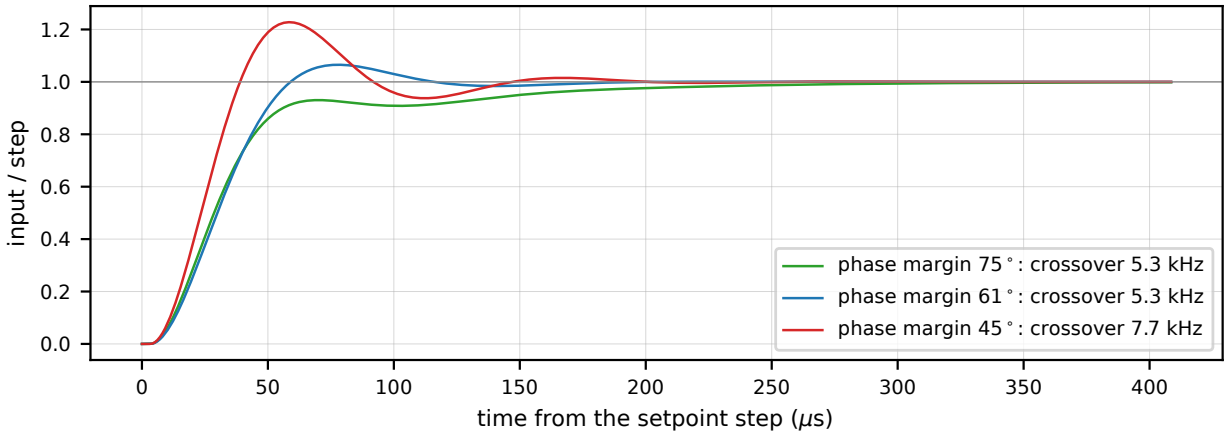


Figure 10.2: Setpoint steps of a slow PI loop whose plant is a demodulator's filter (order 2, $\tau = 20\,\mu$s), with gains designed for three phase margins (computed with `design_pi` and `predict_setpoint_step`).

Phase-locked loop: input θ, target an oscillator's frequency (chapter 11).

Slow actuators [feedback]: input a slow analog input (a temperature sensor's voltage) or an input's mean, target a slow analog output (a heater or a piezo driver).

`feedback` measures a slow loop's plant from steps of its target and designs the gains for chosen margins (chapter 14).

Chapter 11

Phase-locked loops

A phase-locked loop (PLL) makes an oscillator follow an input signal: its frequency, and its phase up to a constant. In the designs it is one of the slow PIDs of chapter 10, reading a demodulator's phase θ and acting on the frequency of the oscillator that demodulator uses as its reference. The oscillator then *is* a copy of the input: its frequency, read from the PID's output, measures the input's frequency every frame, and the other demodulators and outputs can use it as a clean reference (a phase-coherent drive of a resonator at its own frequency, a tracking measurement of a drifting resonance, a clean-up of a noisy clock). `analysis.lock_pll` sets the loop up from one number, its bandwidth; this chapter derives its design, its margins, its noise, how far it pulls in and how the frequency-locked loop extends that range.

General references are Gardner's and Best's books [18, 19]; what follows is specific to a loop sampled at the frame rate, with a sawtooth phase detector and a few frames of delay.

11.1 The loop

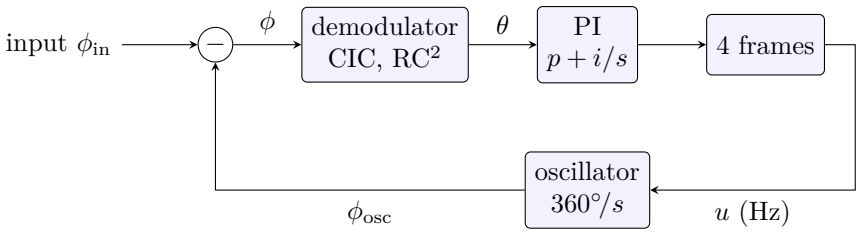


Figure 11.1: The PLL. The demodulator compares the input's phase with its reference's, the oscillator's (the phase θ of $X + \mathrm{i}Y$); the PI turns the phase error into a frequency correction u, added to the oscillator's set frequency four frames later; the oscillator integrates the frequency into its phase.

The oscillator is an integrator: a frequency correction u (Hz) turns its phase by $360°u$ per second. The demodulator's phase θ is the difference between the input's phase and the oscillator's, filtered by the CIC and the RC sections. A PI controller closes the loop:

$$u = -\left(p + \frac{i}{s}\right)\theta, \qquad L(s) = \left(p + \frac{i}{s}\right)\frac{360°}{s}\,H_{\text{RC}}(s)\,\mathrm{e}^{-sT} \tag{11.1}$$

(continuous approximation; p in Hz per degree, i in Hz per degree per second, T the loop's delay). The loop is of *type 2*: two integrators, the oscillator's and the PI's, so it follows a constant frequency offset with zero phase error and a frequency ramp with a constant one. `pll_settings` places the crossover at the requested bandwidth bw ($\omega_c = 2\pi\,\text{bw}$) and the PI's zero a factor four below it:

$$p = -\frac{\omega_c}{360°}, \qquad i = p\,\frac{\omega_c}{4} \qquad \Longrightarrow \qquad |L(\mathrm{i}\omega_c)| \approx \left|1 - \tfrac{\mathrm{i}}{4}\right| = 1.03. \tag{11.2}$$

The minus sign: the PID's error is setpoint $-\,\theta$, and an input running ahead of the oscillator ($\theta > 0$) must raise its frequency.

common/python/rpfpga/demod.py

```
    wc = 2 * math.pi * bandwidth
    p = -wc / 360.0
    span = frequency_range if frequency_range is not None else 5 * bandwidth
```

`pll_filter` sets the demodulator's filter for the loop: order 2 with its corner eight times above the bandwidth, $\tau = 1/(2\pi \cdot 8\,\mathrm{bw})$, which suppresses the $2f$ product and the out-of-band noise by 40 dB per decade above 8 bw while costing $2\arctan\frac{1}{8} = 14°$ of phase at the crossover. The phase margin is then

$$\mathrm{PM} \approx 90° - \arctan\tfrac{1}{4} - 2\arctan\tfrac{1}{8} - 360°\,\mathrm{bw}\,T = 61.7° - 360°\,\mathrm{bw}\,T\,: \tag{11.3}$$

the integrators take 180°, the PI's zero gives back 76°, the filter takes 14°, and the delay takes the rest. The delay is what limits a PLL's bandwidth.

11.2 The sampled loop

The loop runs at the frame rate, and three of its parts are not continuous at all: the PID's output acts four frames after the frame whose error it used, the oscillator's frequency is held constant over each frame, and the demodulator samples the phase through the CIC's triangle (chapter 5). Their combination has an exact frame-rate description. A correction held for one frame from sample $128m$ turns the reference's phase linearly during that frame (by $1/f_\mathrm{o}$ cycles per hertz) and leaves it turned afterwards. The CIC's frame value j is the triangle-weighted mean of the phase over samples $128j - 127 \ldots 128j + 127$, centred on frame j's first sample. The phase's increments between successive frame values are therefore the triangle convolved with the ramp's derivative (a box of one frame), convolved once more with a box: four boxes, the *cubic B-spline*, sampled at its knots:

$$\frac{\theta(z)}{u(z)} = \frac{360°}{f_\mathrm{o}}\,\frac{g(z)}{1 - z^{-1}}, \qquad g(z) = \tfrac{1}{6} + \tfrac{2}{3}z^{-1} + \tfrac{1}{6}z^{-2} \qquad \text{(exactly: 0.1706, 0.6666, 0.1628)}, \tag{11.4}$$

the exact values being the sums over the integer samples (`frame_fir` in `lockin/tools/theory_figures.py`; figure 11.2). At low frequencies $g(z) \approx z^{-1}$: the sampling costs one frame. A target that *adds* to the input instead (an amplitude, an offset: the slow loops of chapter 10) sees three boxes, the quadratic B-spline, $\approx \frac{1}{2}(1 + z^{-1})$: half a frame, plus the time the target takes to reach the input (section 10.2).

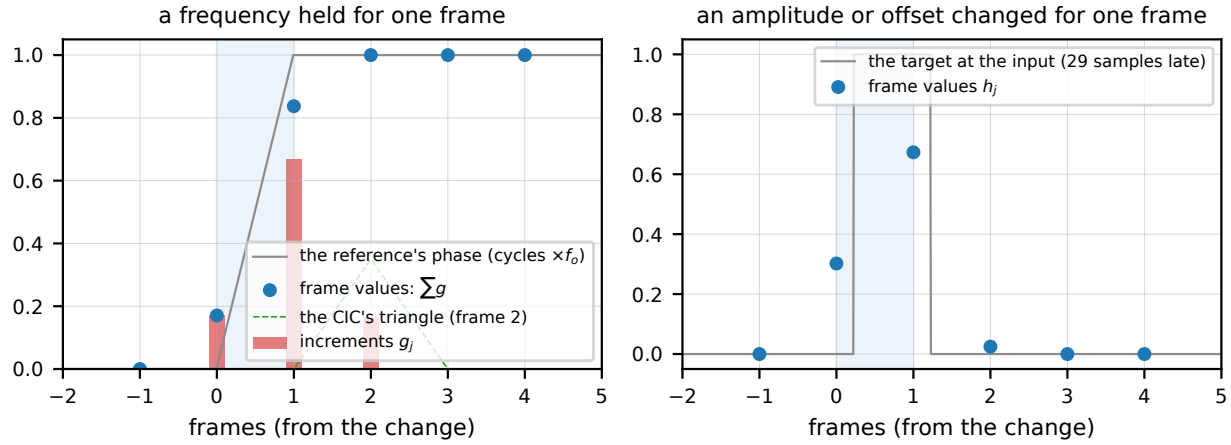


Figure 11.2: How the CIC samples a target that changes for one frame (shaded). Left: a frequency; the reference's phase ramps during the frame, and the frame values (the triangle-weighted means) rise by the cubic B-spline's $\frac{1}{6}, \frac{2}{3}, \frac{1}{6}$. Right: an amplitude or offset that reaches the input 29 samples after it changes (the core's 14 and the converters' 15); the frame values are the quadratic B-spline's, shifted: 0.3021, 0.6731, 0.0248.

The PLL's open loop in the frame-rate z-domain is then

$$L(z) = \left(|p| + \frac{|i|/f_\mathrm{o}}{1-z^{-1}}\right) z^{-4}\, \frac{360^\circ}{f_\mathrm{o}}\, \frac{g(z)}{1-z^{-1}} \left(\frac{\alpha}{1-(1-\alpha)z^{-1}}\right)^2, \qquad \alpha = 1 - \mathrm{e}^{-1/(\tau f_\mathrm{o})}. \tag{11.5}$$

Its delay, $T \approx 4.1/f_\mathrm{o} = 4.2\,\mu\mathrm{s}$ in equation (11.3), is the pipeline's four frames plus the sampling's one, less the frame by which two discrete RC sections lead the continuous ones ($(1-\alpha)/\alpha$ frames of group delay each instead of $-1/\ln(1-\alpha)$). Figure 11.3 shows L for bandwidths of 1 and 10 kHz.

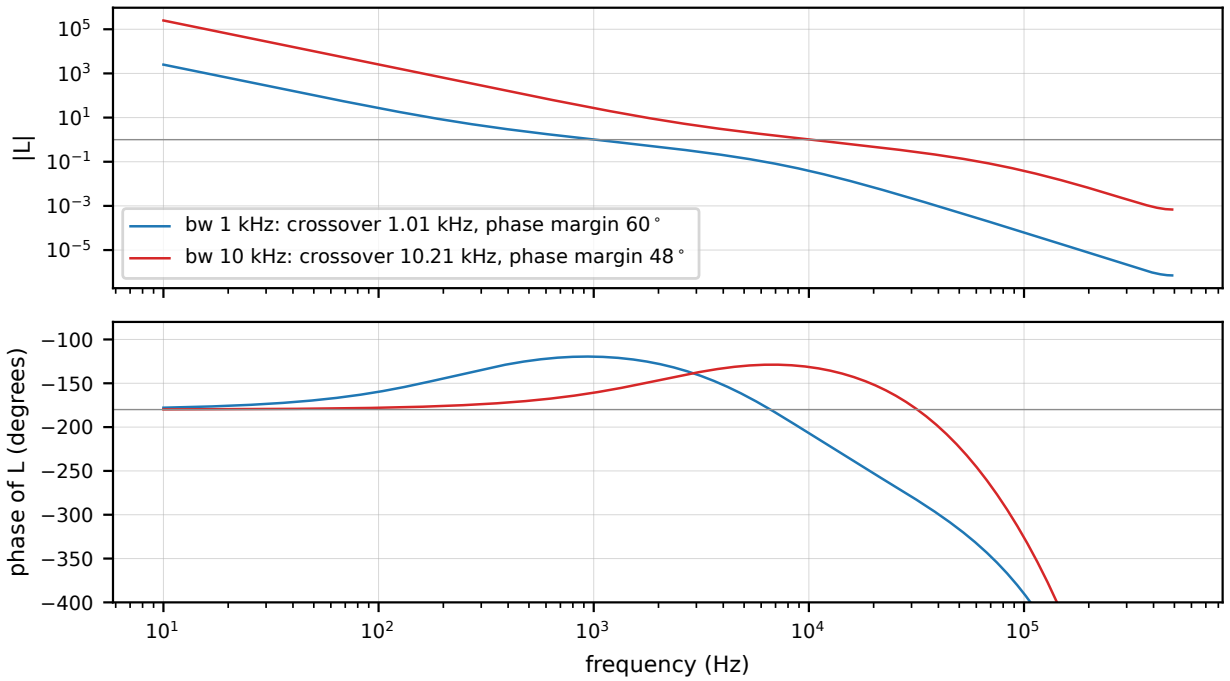


Figure 11.3: The open loop of `lock_pll`'s PLL for bandwidths of 1 kHz and 10 kHz (the exact sampled loop above). The phase rises from -180° toward the PI's zero at bw/4 and falls again through the filter and the delay.

bandwidth	1 kHz	10 kHz	19 kHz
crossover (kHz)	1.01	10.21	
phase margin	60°	48°	36°
gain margin (dB)	21.0	11.3	6.9
peak of $\lvert T\rvert$ (dB)	1.6	2.2	

Table 11.1: Margins of `lock_pll`'s loop from the sampled model.

The top bandwidth. The integral gain $i = \omega_c^2/(4 \cdot 360)$ grows as the square of the bandwidth and reaches the PID's gain range near 19 kHz, where the driver refuses it. The margins there are already thin (table 11.1): 10 kHz, with 48°, is the practical top. A wider loop would need a different structure (less filtering, which lets the $2f$ product into the loop) rather than more gain.

11.3 Tracking and noise

Closed, the oscillator's phase follows the input's through

$$T = \frac{L}{1+L}, \tag{11.6}$$

a low-pass with a 3 dB corner at 1.66 bw and a peak of 1.6 dB for `lock_pll`'s design (figure 11.4). The reading θ shows the part the loop does not follow, $H_{RC}/(1+L)$: slow phase changes of the input are suppressed in proportion to $|L|$, which grows as $1/f^2$ below the PI's zero. A locked loop's θ is therefore the *phase error* and should stay near the setpoint (the lock detector of section 10.1 watches it), while its frequency, the PID's output, is the measurement.

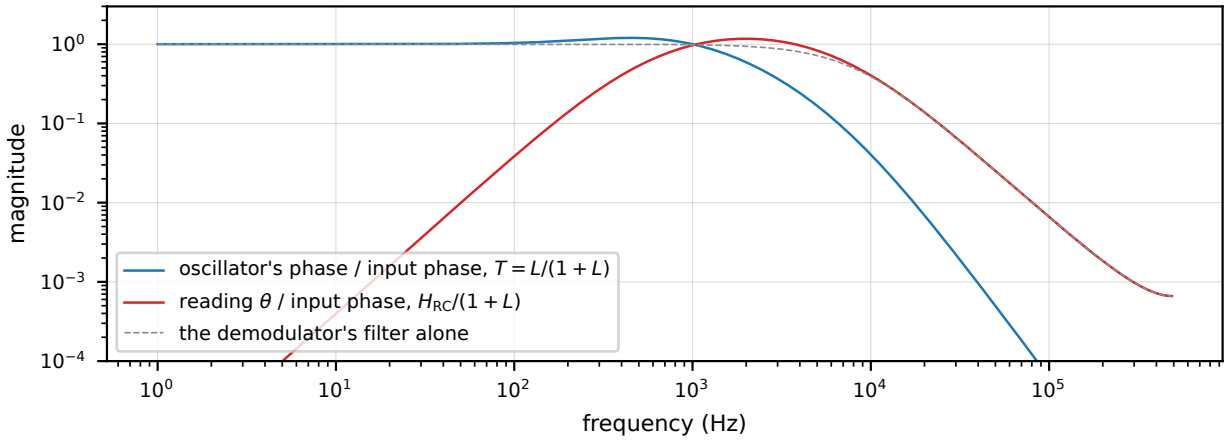


Figure 11.4: The closed 1 kHz loop: the oscillator follows the input's phase below about 1.7 bw (blue); the reading θ shows the input's faster phase changes (red), filtered by the demodulator above 8 bw.

Noise. An input of rms amplitude R with white noise of density e_n near its frequency has a phase noise of one-sided density e_n/R in rad/$\sqrt{\text{Hz}}$ (chapter 8: Y's density is e_n and $\theta = Y/R$). Through the loop, the oscillator's phase and frequency (the PID's output, read every frame) scatter by

$$\sigma_\phi = \frac{e_n}{R}\sqrt{B_L}, \quad B_L = \int_0^{f_o/2} |T|^2 \mathrm{d}f, \qquad \sigma_u = \frac{e_n}{R}\sqrt{\int_0^{f_o/2} f^2 |T|^2 \mathrm{d}f} \quad \text{(rad, Hz)}. \tag{11.7}$$

For `lock_pll`'s loop $B_L = 2.28$ bw and $\int f^2|T|^2\mathrm{d}f = 7.5\,\text{bw}^3$ (1 kHz; 3.02 and 9.0 at 10 kHz): the phase noise grows as $\sqrt{\text{bw}}$, the frequency reading's as $\text{bw}^{3/2}$. A narrow loop measures the frequency more quietly but follows the input's own frequency changes more slowly; the right bandwidth is the frequency of the fastest changes one wants to follow. The oscillator's own noise is none: the DDS is exact (chapter 3), and the loop adds only the input's noise.

11.4 Pull-in

The phase detector is θ itself, wrapped to $\pm 180°$: unlike a mixer's $\sin\phi$ it is linear over the whole cycle, a sawtooth. From any phase within one cycle the loop locks without slipping, and the error of a phase PID wraps the same way, so that no cycle history matters (section 7.2). From a frequency offset Δ the beat drives the sawtooth around at Δ, and the loop must *pull in*: the PI's output carries the sawtooth at the beat frequency, which modulates the oscillator's frequency and with it the beat. The beat slows down in the half of each cycle where the modulation brings the two frequencies closer, so the sawtooth spends longer there: its mean is no longer zero, and the integrator follows that mean toward lock. The mean's sign depends on the phase with which the modulation comes back: it pulls in while the path from the detector to the oscillator's frequency (PI, delay, sampling, filter) lags the beat by less than 90°, and pushes away beyond [18].

The pull-in range is therefore where that lag reaches 90°: near the filter's corner, 8 bw, where the two RC sections alone lag 90°, for narrow loops; closer for wide ones, where the delay's lag adds. Figure 11.5 shows the nonlinear loop simulated sample by sample (the reference's phase, the CIC's triangle, the RC sections, atan2 and the clamped PI) and the range found by it, against the 90°

criterion and the board's measurements: 8, 7 and 3.25 bandwidths at 100 Hz, 1 kHz and 10 kHz (criterion 7.6, 6.8, 3.5; board 7–8, 7–8, 3–5).

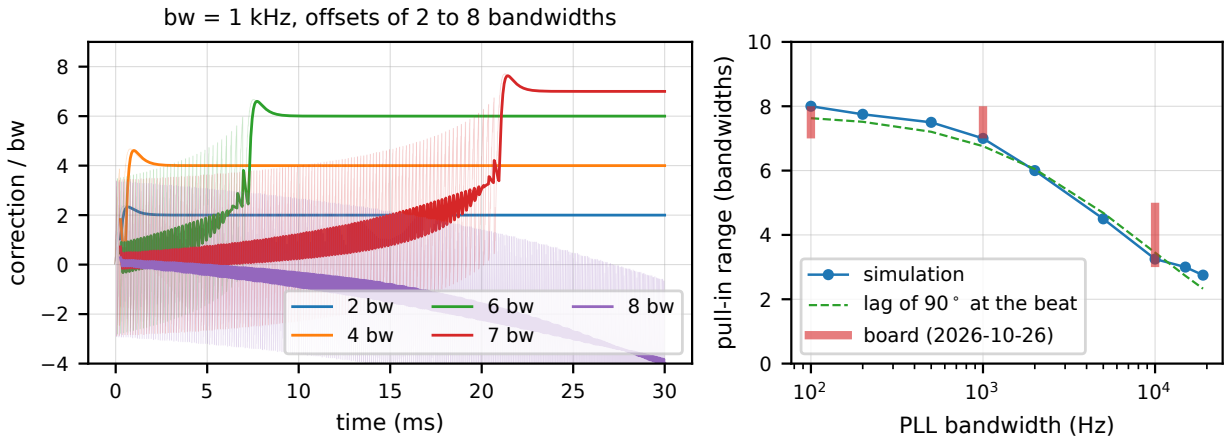


Figure 11.5: Left: the oscillator's correction while a 1 kHz loop pulls in from 2 to 8 bandwidths (thin: every frame; thick: a running mean over 0.26 ms). From 7 bw the pull-in takes 20 ms; from 8 bw the mean drifts away. Right: the pull-in range against the bandwidth: the simulation, the beat at which the detector-to-oscillator path lags 90°, and the board's records (`lockin/docs/VERIFICATION.md`).

Limits. The PID's limits, ±5 bw around the set frequency by default (`frequency_range`), lie inside the pull-in range. A loop that lost its input winds its integrator to a limit; when the input returns near the set frequency, the beat from the limit is within the range and the loop pulls back in by itself. Wider limits would let a lost loop wander out of its own range.

Lock time. Inside the range, the time to lock scales as 1/bw: from one bandwidth, $|\theta| < 5°$ after 14.5, 1.46 and 0.145 ms at 100 Hz, 1 kHz and 10 kHz on the board, about 1.5/bw. From farther, pull-in is slower: the mean that drives it shrinks as the beat approaches the range's edge.

Starting near the input. `lock_pll`'s `seed` first measures the input's frequency with the frequency counter (section 17.3) and sets the oscillator to it: the loop then starts within a fraction of a hertz, far inside the range, and only the phase remains to lock. A demodulator `hold_below` level holds the PID (its integrator, i.e. the frequency) while R is below it: a lost input leaves the oscillator at its last frequency instead of letting the loop run to a limit on noise.

11.5 The frequency-locked loop

Signals the counter cannot measure (a harmonic, a sideband, the difference IN1 − IN2, a signal in noise) can still be acquired from far outside the pull-in range by first locking the *frequency*. With `acquire` set, `lock_pll` first gives the same PID the demodulator's frequency deviation $d\theta$ (section 7.2) as its input, a type-1 loop:

$$u_k = u_{k-1} + \frac{|i_F|}{f_o}\,\Delta f_k, \qquad |i_F| = \min(\pi\,\mathrm{bw},\ 3000)\ \mathrm{s}^{-1}, \tag{11.8}$$

so that the beat Δf decays as $\Delta f(t) = \Delta f(0)\,\mathrm{e}^{-t/\tau_F}$ with $\tau_F = 1/|i_F|$. Its detector reads the beat's frequency correctly however strongly the filter attenuates the beat, as long as the beat stays within $\pm f_o/2 = \pm 488$ kHz and the signal above the noise: $d\theta$ counts the phase's turns, not its amplitude. The beat therefore decays exponentially from any offset in that range (figure 11.6). The gain is capped at $3000\,\mathrm{s}^{-1}$ (a frequency-to-frequency gain reaches the PID's range near $3800\,\mathrm{s}^{-1}$), which sets $\tau_F \geq 0.33$ ms for bandwidths above 1 kHz.

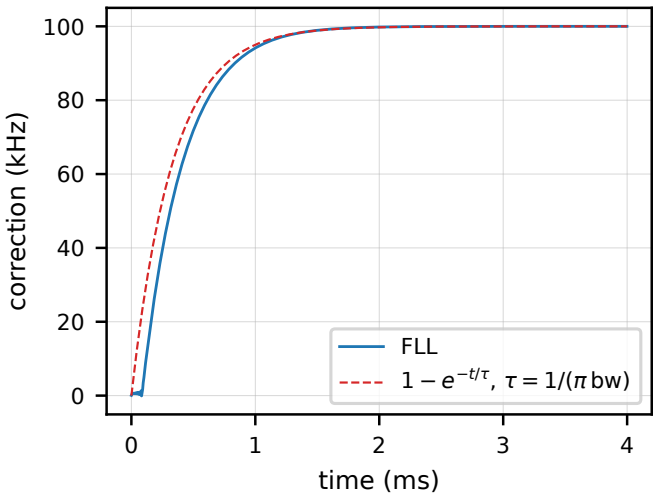


Figure 11.6: The FLL of a 1 kHz loop acquiring an input 100 kHz (100 bw) away (simulated sample by sample): the correction approaches the offset with the time constant $1/(\pi\,\mathrm{bw})$; the beat is below bw/2 after 1.7 ms.

Once the beat stays below bw/2 for three readings, one apply turns the PID into the PLL: input θ, the PLL's gains, its limits centred on the correction found. The integrator is kept in the target's units (hertz), so it carries the frequency across the switch: no reset, no transient beyond the new P term, and the phase locks from inside the range. On the board the whole acquisition, from 100 bandwidths at 100 Hz and 1 kHz and from 30 at 10 kHz, took about 0.3 s, set by the software's polling of the readings rather than by the loop.

11.6 Variants

Harmonics and sidebands. A demodulator at harmonic m of its oscillator, or at a sideband $mf_A \pm f_B$ (section 4.1), sees m times the oscillator's phase: a PLL through it has m times the loop gain. Dividing p and i by m restores the design; the pull-in range in the input's frequency scales accordingly.

A phase target. A PID may act on an oscillator's *phase* instead of its frequency: a type-1 loop without the oscillator's integration that holds a phase relation against slow drifts, but cannot follow a frequency offset.

Locking to the TTL reference [feedback]. In `feedback` a PID can read the phase of the TTL reference at DIO0_P, measured to a nanosecond, and lock an oscillator to an external lab clock, also at a ratio N/M (chapter 15).

Chapter 12

Fast PID controllers [feedback]

The slow PIDs act once per frame, about a microsecond apart and four frames late: enough for loops of tens of kilohertz. Stabilizing a laser to a cavity, a cavity to a laser, or an interferometer's path difference needs loops of hundreds of kilohertz, which only a controller working on every sample can reach. `feedback` has two such fast PIDs. Each takes one input per sample and adds its output to one output's sum on the same sample index, with a fixed latency of 14 samples inside the FPGA.

12.1 The controller

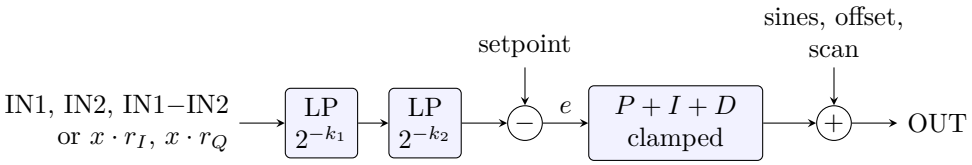


Figure 12.1: A fast PID. Every sample: an input or a mixer product, two low-pass sections, the error, the PID; the output adds to an output's sum (the oscillators' sines, the offset, a scan) before it is rounded to the DAC's 14 bits.

Each fast PID takes one of five inputs per sample: IN1, IN2 or IN1 − IN2 (the input words x of section 2.2, AC-coupled if configured), or a demodulator's in-phase or quadrature *product* $x \cdot r_I$, $x \cdot r_Q$ (section 12.5). In register units it computes (`model.FastPid`; $\gg$ floors)

$$t_1 \mathrel{+}= (u\,2^{16} - t_1) \gg k_1, \qquad t_2 \mathrel{+}= (t_1 - t_2) \gg k_2, \qquad e = \mathrm{sat}_{25}(\mathrm{sp} - (t_2 \gg 16))\,, \tag{12.1}$$

$$P = (e\,k_p) \gg 8s_p, \qquad D = (e\,k_d - (e\,k_d)_{\mathrm{prev}}) \gg 8s_d, \qquad J = \mathrm{clamp}(J) + (e\,k_i) \gg 8s_i, \tag{12.2}$$

$$\mathrm{out} = \mathrm{clamp}(\mathrm{sat}_{32}(P + (J \gg 8) + D),\ \mathrm{lo},\ \mathrm{hi}) \tag{12.3}$$

with the input u in 2^{-8} half converter steps, the integrator J in 2^{-24} DAC steps and the output in 2^{-16} DAC steps.

feedback/control/feedback/model.py

```
        self.t1 += ((u << FAST_T_FRAC) - self.t1) >> f["k1"]
        self.t2 += (self.t1 - self.t2) >> f["k2"]
        e = sat(w["sp"] - (self.t2 >> FAST_T_FRAC), 25)
```

Gains. Each gain is an 18-bit mantissa times 2^{-8s}, $s = 0 \ldots 3$; the driver picks the finest step that holds the gain (`fast_gain`), so small gains keep 17 significant bits. In physical units, with v_e the volts per error unit and v_o the volts per output unit,

$$k_p = p\,\frac{v_e}{v_o}, \qquad k_i = i\,\frac{v_e}{v_o}\,\frac{2^8}{f_\mathrm{s}}, \qquad k_d = d\,\frac{v_e}{v_o}\,f_\mathrm{s} \tag{12.4}$$

(p in V/V, i in s^{-1}, d in s): for an input and an output of equal scales p reaches about 1000, i about $5 \times 10^8\,\mathrm{s}^{-1}$ and d about $8\,\mu\mathrm{s}$. The integral gain i is the loop's unity-gain frequency of an integrator alone, in radians per second: $i = 2\pi \cdot 1\,\mathrm{MHz} = 6.3 \times 10^6\,\mathrm{s}^{-1}$ crosses over at 1 MHz on a plant of unit gain.

The integrator and the limits. The output is clamped to the PID's limits `limits` (in volts at the output); the integrator is clamped to the same range before each addition (*anti-windup*: it never runs beyond the limits, so the loop recovers as soon as the error changes sign). The output adds to its output's sum (the oscillators' sines, the offset, the slow PIDs' DC targets, a scan) *before* the sum is rounded to 14 bits and clamped to the output's own limits (equation (3.5)), so the output's protection holds whatever the PID does. *Hold* (the setting, or its acquisition scanning) zeroes P and D and keeps J; a *reset* zeroes J; a disabled PID clears every stage.

Resolution. The error has 25 bits in 2^{-8} half converter steps: the sections' averaging resolves an input far below one converter step. The integrator keeps 8 bits more than the output (2^{-24} DAC steps), so that the increments of a slow integrator, far below a DAC step per sample, still accumulate. The output itself is resolved to 2^{-16} DAC steps, but the DAC takes 14 bits: a locked integrator's output toggles between neighbouring DAC codes (a limit cycle of one step, 0.12 mV on the ± 1 V range), which the output's dither (section 3.3.1) randomizes into white noise.

12.2 Latency

The pipeline registers the input 3 clocks after its sample, runs the sections, the error, the gains' products, the integrator, the sum and the clamp, and presents the output 11 clocks after the sample, 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. The converters and the analog paths add their loop delay of about 120 ns (section 2.4): **232 ns from the input connector to the output connector**. A product input is formed from the mixer's integrator four samples later: 18 samples, 144 ns, about 264 ns connector to connector. The status reports each fast PID's latency (`latency`) from the calibration's delays.

A delay T lags by $360^\circ fT$: 28° at 338 kHz for 232 ns. Every degree of the phase margin it takes is a degree the controller cannot give back, which makes the delay the fast loop's limit.

12.3 The bandwidth a delay allows

Take a plant that is flat (its own response fast compared with the loop) and the delay T: $P(f) = K\mathrm{e}^{-\mathrm{i}2\pi fT}$. With an integral controller, $L = (Ki/\mathrm{i}2\pi f)\,\mathrm{e}^{-\mathrm{i}2\pi fT}$: its phase crosses -180° at $f_{180} = 1/(4T)$, where $|L| = f_c/f_{180}$. A gain margin GM therefore puts the crossover at

$$f_c = \frac{1}{4T \cdot 10^{\mathrm{GM}/20}}, \qquad \mathrm{PM} = 90^\circ - 360^\circ f_c T, \tag{12.5}$$

341 kHz and 61.5° for 232 ns and 10 dB: on a pure delay the gain margin binds before the phase margin. A proportional term adds phase near the crossover and moves it up by about 15 %. Figure 12.2 shows the largest crossover with 60° and 10 dB from `design_pi` against the delay: 388 kHz for the raw path (63.4°, 10.1 dB; I only: 338 kHz) and 338 kHz for a product. On the board, `tune_fast` on an OUT1 → IN1 cable designed 385.2 kHz and 62.9° and measured 387.8 kHz and 62.7° in the closed loop: the board's loop is a pure delay of 232 ns to within the measurement.

Suppression. Below the crossover the loop divides disturbances by $|1 + L| \approx |L|$: an integrator gives 20 dB per decade, a PI the same with a higher crossover. Above it the loop cannot help, and between the crossover and f_{180} it amplifies: at f_{180} by $1/(1 - 10^{-\mathrm{GM}/20})$, 3.3 dB for 10 dB of gain margin, at most 3.8 dB for the design of figure 12.2. A plant with its own lag (a piezo's resonance, a

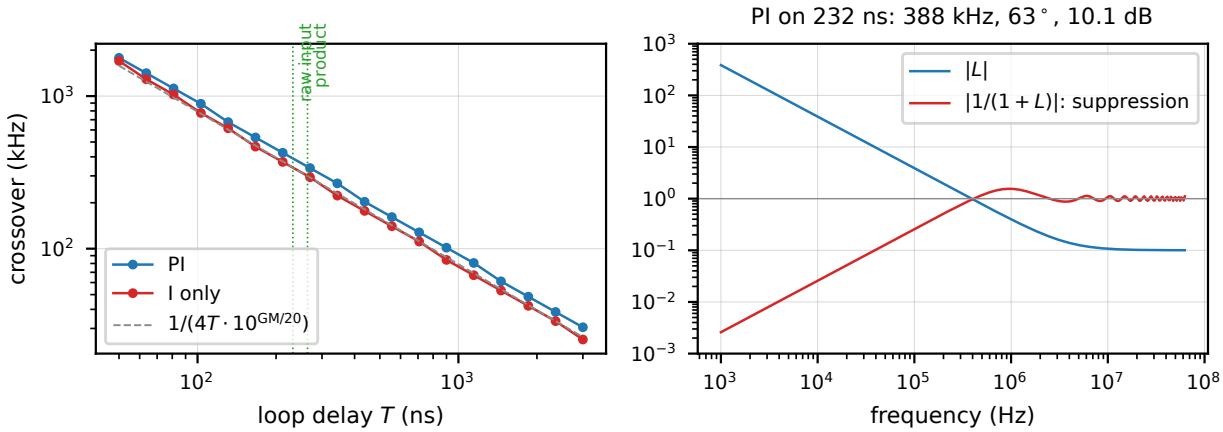


Figure 12.2: Left: the largest crossover of a fast loop with 60° phase margin and 10 dB gain margin on a pure delay T (PI and integral-only control; `design_pi`), with the I-only formula. Right: the PI loop on 232 ns and its suppression of disturbances $|1/(1+L)|$: -52 dB at 1 kHz, -32 dB at 10 kHz, and an amplification of up to 3.8 dB above the crossover.

current driver's bandwidth, a long cable: 5 ns per metre) lowers all of these; `tune_fast` measures the actual plant and designs the gains on it (chapter 14).

12.4 The low-pass sections

Two first-order sections $t \mathrel{+}= (u - t)\,2^{-k}$, $k = 1 \ldots 15$ (0: none), filter the input before the error. A section is an RC of

$$f_c = -\ln(1-2^{-k})\frac{f_s}{2\pi}: \quad 13.8\,\text{MHz}\ (k=1),\ 1.28\,\text{MHz}\ (4),\ 77.9\,\text{kHz}\ (8),\ 4.86\,\text{kHz}\ (12),\ 607\,\text{Hz}\ (15), \tag{12.6}$$

about an octave per step of k (figure 12.3). The configuration takes the corners in hertz (`filter`) and rounds them to the nearest k; the applied configuration reports the realized corners. The sections limit the noise bandwidth of the error and, for a product input, suppress the $2f$ product; they cost phase, $\arctan(f_x/f_c)$ each at the crossover f_x, which comes out of the phase margin: 11° per section at $f_c = 5f_x$.

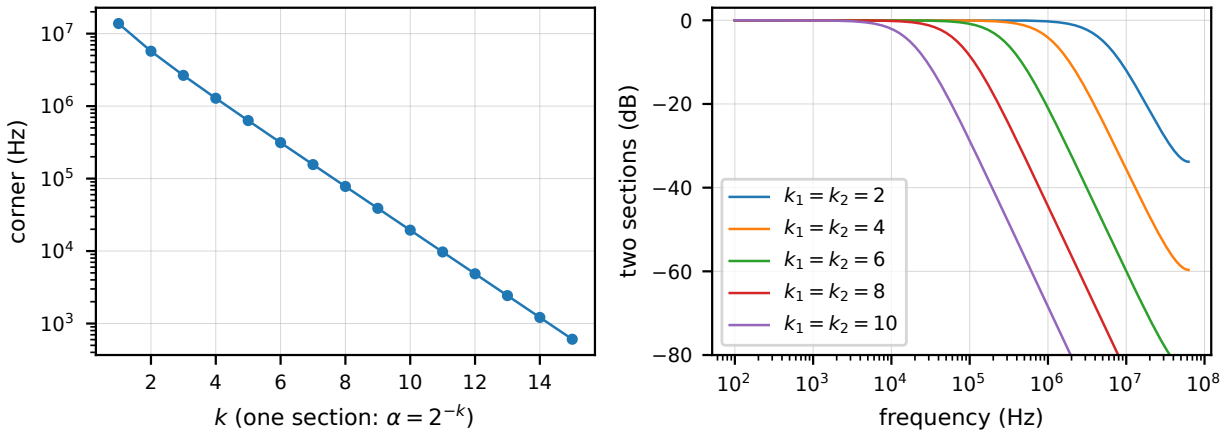


Figure 12.3: Left: the corner of one section against k. Right: two equal sections for $k = 2 \ldots 10$.

12.5 Product inputs: a demodulated error at the full rate

Many locks use an error signal obtained by demodulation: the Pound–Drever–Hall technique [20, 21] phase-modulates a laser at a radio frequency and demodulates the light reflected from a cavity at that frequency; its in-phase part is a dispersive error around the resonance. A fast PID can take such an error directly from a demodulator's mixer, without the frame-rate filtering: the product of the input and the reference,

$$x\,r_I = a\cos(\omega t + \theta)\cdot S\cos\omega t = \frac{aS}{2}\left[\cos\theta + \cos(2\omega t + \theta)\right], \tag{12.7}$$

whose DC term is the in-phase amplitude and whose $2f$ term the sections must remove. The product is taken exactly: the mixer's DSP48 accumulates the products (the CIC's first integrator, section 4.3), and the difference of two successive accumulator words is the latest product, available four samples after the raw input. The driver scales the error like the demodulator's X (rms or peak volts of the in-phase amplitude, a square reference's gain included), so that a setpoint and a gain mean the same as for X.

With two sections of corner $f_c \ll 2f$ the ripple left is about $(f_c/2f)^2$ of the DC and the lag at the crossover $f_x \ll f_c$ about $2f_x/f_c$ radians: their product is fixed by the carrier,

$$\frac{\text{ripple}}{\text{DC}} \approx \left(\frac{f_x}{f\cdot\text{lag}}\right)^2. \tag{12.8}$$

Exactly (two sections, their corner chosen for 10^{-3} of ripple), the lag at a 100 kHz crossover is 9° for a 25 MHz carrier, 18° for 10 MHz, 35° for 5 MHz and 115° for 1 MHz: a fast loop on a low

carrier must accept ripple or a lower crossover (figure 12.4). The $2f$ product of a carrier above 31.25 MHz folds back below the Nyquist frequency of 62.5 MHz; what matters is the folded frequency, $\min(2f \bmod f_\mathrm{s},\ f_\mathrm{s} - 2f \bmod f_\mathrm{s})$.

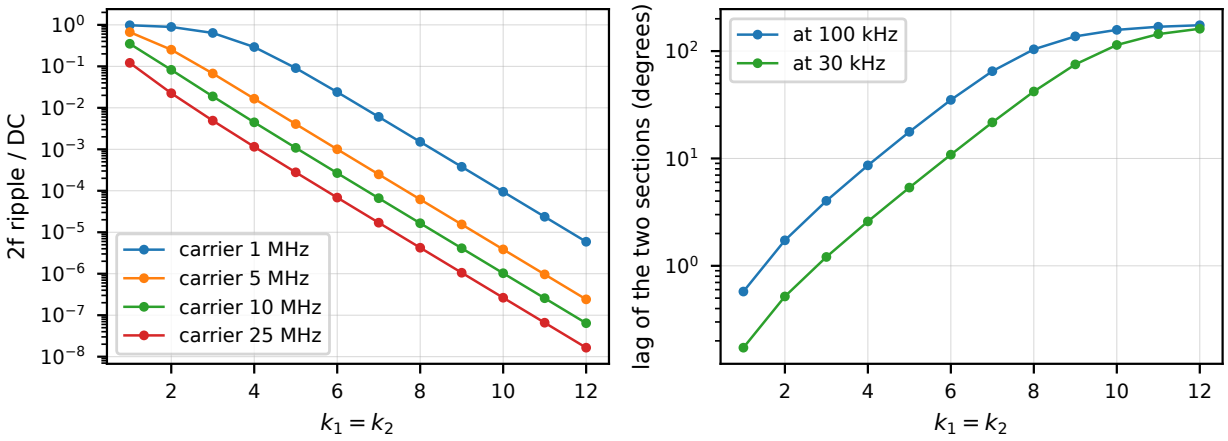


Figure 12.4: A product input. Left: the $2f$ ripple left by two equal sections against k, for carriers of 1 to 25 MHz (the folded $2f$). Right: the lag of the same sections at 30 kHz and 100 kHz, which the loop's phase margin pays.

12.6 Fast and slow together

A fast PID has the range of its output's limits and of the DAC, a volt or so; a laser's current, a piezo's voltage, a cavity's length drift by more over minutes. A second, slow actuator with a large range (an output's offset, a slow analog output to a piezo driver or a temperature controller) should take the slow part. The slow PIDs cannot read a fast PID's output, so the split is made in the controller instead: the fast PID gives the proportional (and high-frequency) action on the fast actuator, with little or no integral gain, and a slow PID integrates the same error (an input's frame mean for a raw input, the demodulator's X for a product) onto the slow actuator. The combined controller is

$$C(f) = p_\mathrm{fast}\, G_\mathrm{fast}(f) + \frac{i_\mathrm{slow}}{\mathrm{i}2\pi f}\, G_\mathrm{slow}(f)\, \mathrm{e}^{-\mathrm{i}2\pi f T_\mathrm{slow}}, \tag{12.9}$$

(G the actuators' responses): below the frequency where the two terms are equal the slow integrator dominates, takes the large excursions and removes the steady error; above it the fast actuator acts alone, around the middle of its range. A fast PID with a large integral gain of its own would hold the error at zero by itself and leave the slow loop nothing to see. The crossover between the two must lie well below the fast loop's crossover and within the slow loop's own bandwidth (the frame-rate delay and the slow actuator's filter), and the two actuators must act with the same sign at that frequency. A slow PID can be *gated* by a lock acquisition, so that it acts only while the fast loop is locked (chapter 13).

Chapter 13

Lock acquisition [feedback]

An error signal is useful only near its lock point. The Pound–Drever–Hall signal of a cavity, the transmission slope of a side-of-fringe lock and an atomic line's dispersive signal are linear within about a linewidth of the resonance and flat (or of the wrong sign) far from it. A loop closed elsewhere runs its integrator to a limit. Lock acquisition finds the resonance first: it scans the actuator until a monitor signal shows the resonance, closes the loop there, watches the lock, and scans again when it is lost. `feedback` has one acquisition unit per fast PID, in the frame engine, with its scan in the fast datapath.

13.1 The scan

The scan is a triangle added to the fast PID's output, between `scan.low` and `scan.high` (volts at the output) at `scan.frequency` triangles per second. Its position p moves by a step per sample, in 2^{-32} DAC steps:

$$\text{step} = \text{round}\left(\frac{2(\text{hi} - \text{lo}) \cdot 2^{16}\, f_{\text{scan}}}{f_{\text{s}}}\right), \tag{13.1}$$

so even a slow scan (a millihertz triangle over a millivolt) moves smoothly, without a visible staircase. The scan runs while its unit scans, *freezes* where it is while the unit locks, and rests at `scan.low` contributing nothing while the unit is off (`model.Scan`).

13.2 The state machine

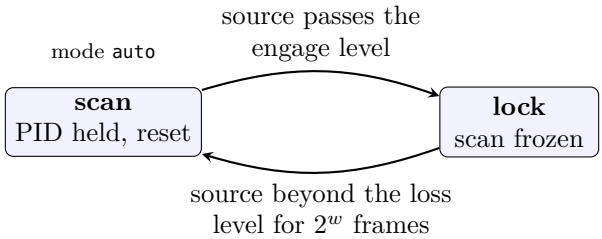


Figure 13.1: An acquisition in mode `auto`. The modes `off`, `scan` and `lock` force the state; `auto` switches between scan and lock by the source's levels.

Once per frame, after the PIDs, the engine compares the unit's *source* with two levels (`model.Acquisition`). The source is any PID input of frame k: X, Y, R, θ, $d\theta$ of a demodulator, an input's frame mean, a slow analog input or the TTL reference's phase. In mode `auto`:

- *scanning*, the unit locks in the first frame whose source passes the *engage level* (above it, or below with `engage.direction below`); each engagement is counted (`relocks`, 16 bits);

- *locked*, the unit returns to scanning when the source has been on the far side of the *loss level* for 2^w frames in a row (`loss.time`, up to 2000 s, rounded up to a power of two frames: 1.02 μs, 2.05 μs … 2^{31} frames); a frame back inside restarts the count.

The engage and loss levels form a hysteresis: a transmission peak might engage above half its height and be lost only below a fifth of it, so that noise near a level does not toggle the state. The window ignores brief dips (a mechanical knock, a mode hop) that the loop rides through.

Like a PID's target, the state of frame k acts from sample $128(k+4)$. *While scanning*, the fast PID is held and its integrator reset: its output is zero and the output is the scan. *At the engagement* the scan freezes where it is and the PID starts from zero around it: the transition is bumpless, the output continuing from the scan's position. A count of the frames spent locked (32 bits, saturating) measures how long a lock has held.

Gating. A slow PID can be *gated* by a unit (`gate`): held unless the unit locks. A slow loop that keeps the fast PID centred in its range (section 12.6) should act only while the fast loop is locked; while scanning, the fast PID's output means nothing. A status pin can carry a unit's lock state to other equipment (section 15.5).

13.3 Engaging on a moving scan

Between the frame in which the source passes the engage level and the sample from which the lock acts, the scan keeps moving: up to one frame for the source's frame average, four frames of the engine's pipeline, and the loop delay of a few hundred nanoseconds, together about $T_e = 5.4\,\mu$s at most. A scan of speed $v = 2(\mathrm{hi} - \mathrm{lo})f_{\mathrm{scan}}$ (volts per second) moves vT_e in that time, and the loop starts that far from the point where the source crossed the level. It captures the resonance if that distance is inside the error signal's capture range, about its linear half-width w:

$$v\,T_e \lesssim w \qquad \Longrightarrow \qquad f_{\mathrm{scan}} \lesssim \frac{w}{2(\mathrm{hi} - \mathrm{lo})\,T_e}. \tag{13.2}$$

The source must also stay past the level for a good part of a frame to be seen in the frame mean: the resonance's width $2w$ should take a few frames to cross, $2w/v \gtrsim 3\,\mu$s, which the first condition already ensures ($2w/v \geq 2T_e \approx 11\,\mu$s). For a resonance of $w = 4$ mV within a 1 V scan, $f_{\mathrm{scan}} \lesssim 370$ Hz.

Figure 13.2 shows a complete cycle, simulated sample by sample with the model's state machine and scan: a fast integral loop (crossover 50 kHz) on the dispersive error $\delta/(1+\delta^2)$ of a resonance at V_0 ($\delta = (V - V_0)/w$, $w = 4$ mV), the transmission $1/(1+\delta^2)$ as the source, engage above 0.5, loss below 0.2 for 2^7 frames, a 400 Hz scan over ± 0.5 V ($vT_e \approx w$). The scan finds the resonance at 0.90 ms; the loop follows its drift (20 mV at 1 kHz) within 375 μV; a jump of 150 mV at 2 ms loses the lock, which the unit notices at 2.134 ms (2^7 frames later); the scan resumes from its frozen position and relocks at 2.36 ms.

Which source? The source should show the resonance unambiguously: a transmission or a demodulated amplitude R peaks there, while the error signal itself crosses zero at the resonance and also far from it. Engaging on a transmission level and locking on a dispersive error is the usual pairing. A demodulated source carries its filter's delay (chapter 6), which adds to T_e: a short time constant or an input's frame mean keeps the engagement prompt.

Scan direction after a loss. The scan resumes from where it froze, in the direction it had, after the PID's integrator has been reset: the output steps back from the lost lock point to the scan's frozen position. A resonance that jumped the other way is found on the return sweep, within one scan period.

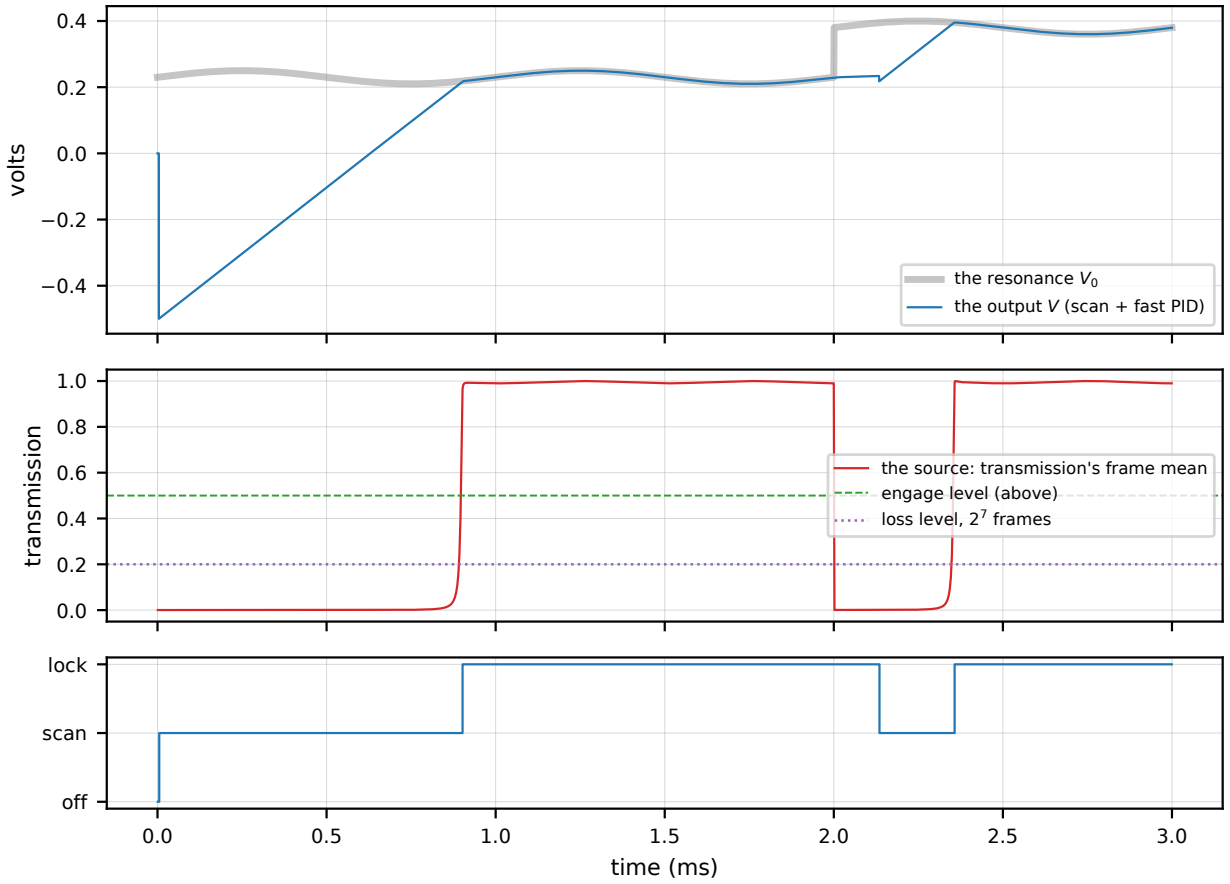


Figure 13.2: Lock acquisition in mode `auto` on a resonance (simulated sample by sample with `model.Acquisition` and `model.Scan`). Top: the resonance's position and the output. Middle: the source (the transmission's frame mean) with the engage and loss levels. Bottom: the state.

Chapter 14

Measuring and tuning a loop [feedback]

A PID's gains are only as good as what is known about the plant. `feedback` measures its loops itself: the fast PIDs' plants as frequency responses, with a sine injected at the output, and the slow PIDs' plants as step responses. It then designs the gains for chosen margins and verifies the closed loop by measuring it again. This chapter describes the margins, the two measurements, the design rule and the checks. Åström and Murray [22] and Skogestad and Postlethwaite [23] cover the general theory.

14.1 Margins

A loop $L(f) = C(f)P(f)$ is stable, with robustness to spare, when $|L|$ falls through 1 (the *crossover* f_c) with its phase well above $-180°$, and when $|L|$ is well below 1 where the phase passes $-180°$:

$$\mathrm{PM} = 180° + \arg L(f_c), \qquad \mathrm{GM} = -20\log_{10}|L(f_{180})|\ \mathrm{dB}. \tag{14.1}$$

The margins say how much the plant may change (its phase, its gain) before the loop oscillates, and they set the transient: about 60° and 10 dB give a step response with little overshoot (figure 10.2). The *sensitivity* $S = 1/(1+L)$ is the factor by which the loop multiplies a disturbance. It is small below the crossover, and its peak $\max|S|$ above the crossover is the price: the closer L passes to -1, the more the loop amplifies noise there. `loop_margins` computes all of them from L sampled at increasing frequencies (interpolating $\log|L|$ and the unwrapped phase in $\log f$), taking the smallest margins when $|L|$ or its phase crosses more than once.

14.2 Measuring the fast loop

`fast_response` adds a sine of amplitude a (10 mV by default) from a free oscillator to the PID's output and captures, of the same samples, the output OUT, the PID's error E (its filtered input, $e = \text{setpoint} - \text{input}$) and its output U at a rate that holds eight periods in the capture's 8192 words. The sine's complex amplitude $\hat{z}$ in each, a least-squares fit of $\mathrm{Re}(\hat{z}\,\mathrm{e}^{\mathrm{i}2\pi ft})$ plus an offset and a slope (`phasor`), rejects everything else in the capture: noise, other signals, the loop's own fluctuations and drift.

- *The plant*, with the PID held (its output frozen): the output carries the injection only, and the error responds to it through the converters, the pipeline, the outside plant and the PID's input sections:
$$P(f) = -\frac{\hat{E}}{\widehat{\mathrm{OUT}}} \qquad \text{(input units per output volt)}. \tag{14.2}$$
- *The loop*, with the PID running: at the summing point the output is the injection plus the PID's response, $\mathrm{OUT} = a + U$, and the PID's response is $U = C\,e = -CP\,\mathrm{OUT}$. The ratio
$$L(f) = -\frac{\hat{U}}{\widehat{\mathrm{OUT}}} = C(f)P(f) \tag{14.3}$$

is the open loop's gain, measured while the loop is closed: the loop never has to be opened, and the result does not depend on the injection's amplitude.

The frequencies are a log grid (24 points from 20 kHz to 8 MHz by default); the oscillator, the output's amplitude and the PID's state are restored afterwards.

14.3 Measuring a slow plant from steps

A slow PID's plant is measured where it acts: its target's setting (an oscillator's frequency or phase, an output's amplitude or offset, a slow output) is stepped by ±step/2 with the PID held, and its input quantity captured at the frame rate (`pid_step_response`, `capture_step`). The capture is armed inside the apply that makes the step, so that the step's frame is known exactly. The mean over the steps (alternately up and down, each less the frames before it) is the response s_j to a unit step at frame 0, which acts from that frame's first sample. Its differences $h_j = s_j - s_{j-1}$ are the plant's frame-rate impulse response, and

$$P(f) = \sum_j h_j \, \mathrm{e}^{-\mathrm{i}2\pi f j / f_\mathrm{o}} \tag{14.4}$$

its frequency response. The first frames show the CIC's sampling (section 10.2): for an input's frame mean that follows the output 20 samples later, $\frac{1}{2}(108/128)^2$ in the step's frame and $1 - \frac{1}{2}(20/128)^2$ in the next. The response's last quarter is taken as settled: its mean is the plant's static gain and stands for the tail, which suppresses the noise of the frames where nothing happens. Steps of the default size (10 Hz, 1°, 10 mV, 20 mV for the frequency, phase, amplitude, offset and slow-output targets) and four of them average the board's spurs to a percent.

14.4 Designing the gains

`design_pi` chooses the PI gains for a measured plant. Among all loops $L = C(p, i)P$ whose crossovers lie inside the measured range, with every phase margin ≥ `margin` (60°), every gain margin ≥ `gain_margin` (10 dB), $|L|$ below the gain margin's limit at the top of the range (beyond it the plant is unknown) and the crossover below a `bandwidth` if one is given, it takes the one with the *largest integral gain*. A unit step disturbance at the plant's output leaves an integrated error $\int e \, \mathrm{d}t = 1/(Ki)$ (for a plant of static gain K), so the largest i is the best rejection of slow disturbances that the margins allow, in the spirit of Åström and Hägglund's MIGO design [24] (which constrains the sensitivity's peak instead of the margins).

The search is exhaustive and vectorized. The plant is interpolated on 240 log-spaced frequencies. For each of 49 proportional gains ($p = 0$ and 48 values over four decades around $1/\mathrm{median}|P|$), the integral gain that puts the crossover at each grid frequency solves the quadratic $|p\,c_0 + i\,c_1|^2 |P|^2 = 1$ (c_0, c_1: the controller's proportional and integral responses per unit gain). The margins of all these candidate loops are computed at once, and the largest admissible i wins. A plant whose low-frequency gain is negative gets negative gains. The controller's form is the hardware's:

$$C_\mathrm{fast}(f) = p + \frac{i/f_\mathrm{s}}{1 - z^{-1}}, \quad z = \mathrm{e}^{\mathrm{i}2\pi f/f_\mathrm{s}}; \qquad C_\mathrm{slow}(f) = \left(p + \frac{i/f_\mathrm{o}}{1 - z^{-1}}\right) z^{-4}, \quad z = \mathrm{e}^{\mathrm{i}2\pi f/f_\mathrm{o}}, \tag{14.5}$$

the fast one without a delay of its own (the measured plant holds the loop's), the slow one with its four frames. `kind i` restricts the search to $p = 0$.

14.5 Applying and verifying

The gains are applied *bumplessly*: the integrator is kept, so a running loop changes its gains without a transient. With `verify`, after $20/(2\pi f_c)$ the tuner measures the result:

- *fast*: the closed loop's L at the same frequencies, and its margins, which should agree with the design's;
- *slow*: four setpoint steps of the closed loop, captured at the frame rate and compared with the error predicted from the measured step response and the applied gains (`predict_setpoint_step`: the PID's arithmetic per frame, its output acting four frames later through the measured response); the rms difference relative to the step is reported.

Figures 14.1 and 14.2 show both on the bit-exact model with a plant of 29 samples of delay (the core's and the converters') and one pole. Fast, a pole at 400 kHz: the design crosses over at 340 kHz with 60.3° and 10.1 dB; the closed loop measures 342 kHz and 60.1°. Slow, a pole at 20 kHz on OUT1's offset read as IN1's frame mean: 16.7 kHz, 61.5°, 10.2 dB, and the setpoint steps follow the prediction within 0.4 % of the step. On the board (OUT1 → IN1, 2026-10-27) the fast design gave 385.2 kHz and 62.9° and the closed loop 387.8 kHz and 62.7°. The slow steps matched their prediction within 5.5 %, limited by a 214 kHz spur that carries about 7 % of a 10 mV step into single frames.

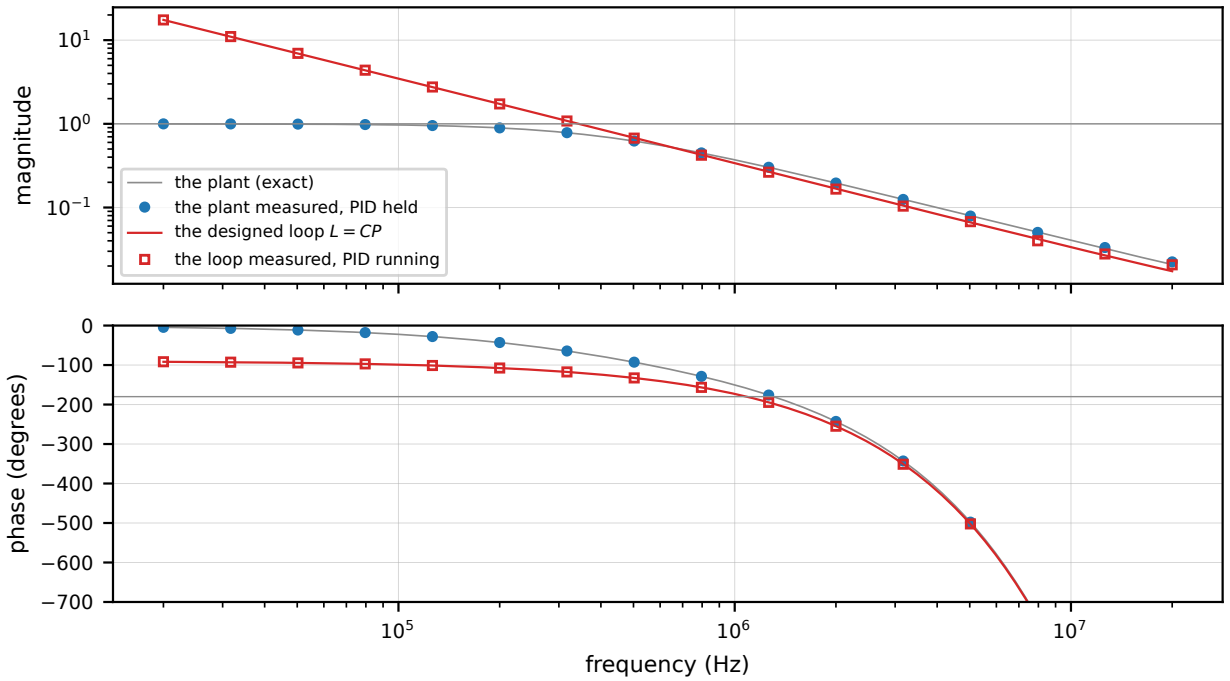


Figure 14.1: `tune_fast` on the model: the plant measured with the PID held (dots; the line is the exact plant), the designed loop (line) and the loop measured with the PID running (squares). The highest points scatter because the plant attenuates the 10 mV probe to a few converter steps.

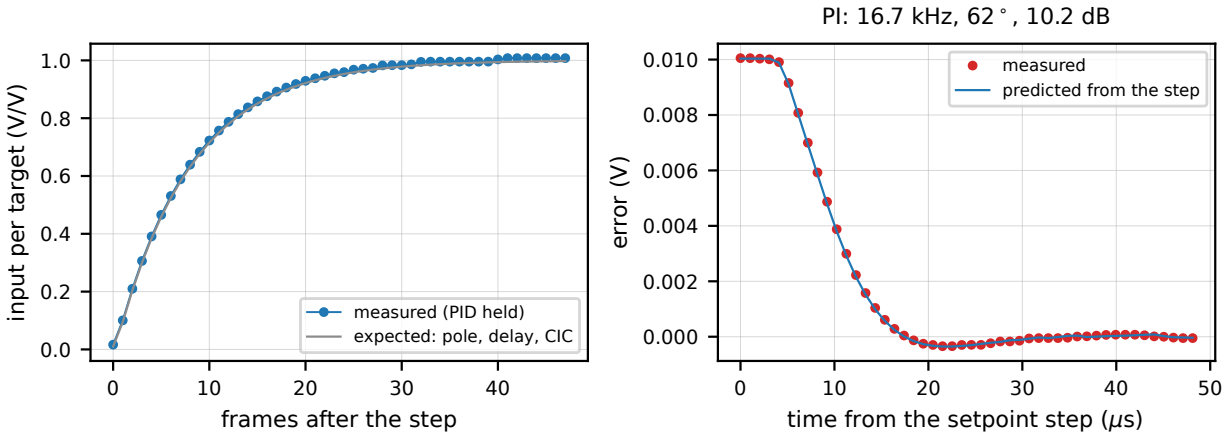


Figure 14.2: `tune_pid` on the model. Left: the measured unit step response of the plant (OUT1's offset to IN1's frame mean through a 20 kHz pole and 29 samples), against the response computed sample by sample through the CIC. Right: the closed loop's error after a 10 mV setpoint step, measured (four steps averaged) and predicted from the step response.

14.6 The static methods

Without the time or the excitation for a measurement, `method static` uses only the plant's static gain G, from two steps of the PID's output (fast) or target (slow) with the PID held, and closes an integral-only loop $i = 2\pi\,\mathrm{bw}/G$:

- *fast*: the bandwidth is the one the computed loop delay T (the status's latency: pipeline and converters, not the outside plant) allows with the requested phase margin, $\mathrm{bw} = (90° - \mathrm{PM})/(360°T)$, at most a third of an input section's corner; on the board 359 kHz;
- *slow*: a quarter of the demodulator's bandwidth $1/(2\pi\tau n)$, 10 kHz for an input's frame mean, 1 kHz for a slow input or the TTL reference.

The static methods assume the plant is flat up to the crossover. A plant with its own lag (a resonance, a slow driver) needs the measured methods, whose design sees it.

Chapter 15

Digital edges timed to a nanosecond

[feedback]

A lock box sits among other equipment: a lab's 10 MHz reference, a pulsed laser's trigger, an oscilloscope that should trigger in phase with the drive. `feedback` connects to them through the extension connector's digital pins. DIO0_P (the TTL reference) and DIO3_P (a capture trigger) are timed to 1 ns, eight times finer than a sample. DIO1_P carries a square wave in phase with an oscillator, its edges placed to 2 ns. All three are referred to the OUT1 connector like every analog phase. The pins are 3.3 V LVCMOS, not 5 V tolerant (see the root README's equipment safety).

15.1 Sampling a pin to 1 ns

The pins DIO0_P and DIO3_P are sampled by the FPGA's input serializers (ISERDESE2 [25, 26]) in oversample mode: two 250 MHz clocks in quadrature (0° and 90°, from an MMCM locked to the sample clock), each edge of each sampling, four samples per 4 ns. Eight pin samples per 8 ns sample; bit j of sample n's word is the pin's level at $n + j/8$. The first rising edge in a word, at sub-sample k (`model.first_edge`), timestamps the edge to $8n + k$ nanoseconds. In simulation an edge is seen, on average, 2.5 ns before the timestamp's nominal time; the board's buffers and traces add their own delays, which the board calibration measures.

Referred to OUT1. Two calibration fields describe the pins against OUT1 (`common/CALIBRATION.md`): `pins_delay_s`, an output pin's lead over OUT1's sample of the same clock edge (−28.6 ns: the pins lead, since OUT1's sample passes the core's 14 samples, the DAC's interface and the analog path), and `pins_loop_s`, an output pin's delay into a 1 ns input through a jumper (4.5 ns, measured with a status pin's frame pulse looped into DIO3_P: `measure_pins_loop`). OUT1 shows sample n 16 samples after the clock edge where the counter reads n, plus its own delay, so an edge arriving at a 1 ns input pin is timestamped, against OUT1's sample at that moment, with

$$T_{\mathrm{TTL}} = \texttt{pins_loop_s} - \texttt{pins_delay_s} + 16/f_{\mathrm{s}} - 2.5\,\mathrm{ns} = 158.6\,\mathrm{ns}, \tag{15.1}$$

and the driver turns a phase read at the pin into one at OUT1: $\phi_{\mathrm{OUT1}} = \phi_{\mathrm{read}} - 360°\, f T_{\mathrm{TTL}}$. The uncertainty is about 1 ns: the timestamps' resolution in the loop measurement and the pins' differences.

15.2 The TTL reference

The TTL reference measures an external clock's phase against an oscillator. Every M-th rising edge of DIO0_P (`divide`) is *taken*, and the phase of oscillator C at that edge is computed from its phase at the edge's sample and the edge's sub-sample k (`model.TtlRef`):

$$e = \mathrm{signed}_{32}((\phi_C(n) \gg 16) + ((f_{24}\, k) \ll 5)) \qquad (2^{-32}\ \mathrm{cycle}), \tag{15.2}$$

f_{24} being the top 24 bits of the oscillator's frequency word with its PIDs' corrections: the phase advances by $f_{24} \cdot 2^5$ per eighth of a sample. The truncations cost less than 10^{-7} cycle. 2^q taken edges (`average`) are averaged; a frame's value is the latest average at the frame's end, and the reference counts as *lost* when no edge was taken in that frame and the 2^w before (`loss_time`). A PID or an acquisition reads it as the source `ttl`, in degrees, its error wrapping at $\pm 180°$ like θ's, and a PID reading it holds while the reference is lost. The sign is the oscillator's: an edge that leads the oscillator by p comes when the oscillator's phase is $-p$ and reads $-p$, the opposite of a demodulator's θ (the input's phase against the reference). The sync output below follows θ's convention, so the TTL reference reads minus the sync's phase setting.

Resolution. One edge's timestamp is uniform within its nanosecond: an rms error of $1/\sqrt{12} = 289\,\text{ps}$, $360° f \cdot 289\,\text{ps}$ in phase (0.10° at 1 MHz, 1.0° at 10 MHz). Averaging $N = 2^q$ edges helps only if the edges fall at different places within the nanosecond (figure 15.1, left):

- a clock whose period is a whole number of nanoseconds (10 MHz: 100 ns) puts every edge at the same place: the error stays at its first value, up to ± 0.5 ns, whatever N;
- a period that is not (9.7 MHz) walks the edges across the nanosecond, and the average's error falls as $1/N$ until the walk's own pattern limits it;
- jitter dithers the edges: the error falls as $1/\sqrt{N}$. Jitter well below a nanosecond dithers incompletely: 0.3 ns rms leaves an error of about 40 ps on a 10 MHz clock.

A PLL to a lab clock. A PID from `ttl` to the oscillator's frequency is a PLL: the oscillator's phase at the taken edges grows by $360°(f_C - N f_{\text{TTL}}/M)$ per second, so the loop locks f_C to N/M times the clock. `lock_ttl` seeds the oscillator from the frequency counter on DIO0_P (section 17.3) times N/M, which picks N, and sets up a type-2 loop like `lock_pll`'s (chapter 11): $p = 2\pi\,\text{bw}/360$ (positive: the reading grows while the oscillator runs fast), $i = p\,\omega_c/4$, bandwidths up to the integral gain's limit of 19 kHz. There is no demodulator filter in this loop, only the delay from the frame's end to the target (three frames) and the average's age ($2^{q-1}M/f_{\text{TTL}}$), so its margins are better than a demodulator PLL's:

$$\text{PM} \approx 76° - 360°\,\text{bw}\,(3/f_\text{o} + 2^{q-1}M/f_{\text{TTL}}), \tag{15.3}$$

about 65° at 10 kHz and 55° at 19 kHz for a 10 MHz clock without averaging. The loss detection holds the loop when the clock disappears; it pulls back in when the clock returns within its limits (± 5 bw).

15.3 The sync output

DIO1_P carries a square wave that is high while an oscillator's phase plus an offset is in its first half cycle (its sine positive). Four bits per sample (`model.sync_bits`), bit j for $n + j/4$:

$$b_j = \neg\,\text{MSB}_{16}((\phi_S(n) \gg 32) + (\text{SPH} \gg 16) + (((f_{24} \gg 8)\,j) \gg 2))\,, \tag{15.4}$$

leave through an output serializer (OSERDESE2, double data rate on the converters' 250 MHz clock) as 2 ns slots. An edge takes the first slot after its phase crossing, so its timing error against the ideal crossing is uniform over 2 ns: 0.58 ns rms, a mean of 1 ns that the calibration removes. As for the TTL input, a frequency whose period is a whole number of slots puts every edge at the same offset (no jitter, a phase-dependent offset up to 2 ns), while others walk across the slot in a sawtooth whose period depends on the frequency (figure 15.1, right): 0.58 ns rms at 9.7 MHz. The pattern repeats, so the sync's spectrum has spurs rather than a noise floor. Equipment that needs a cleaner edge should lock to it with its own PLL.

The phase setting is referred to OUT1's sine of the same oscillator. An edge leaves, against the OUT1 sine of the same phase, by

$$T_{\text{sync}} = 109.1\,\text{ns} - 0.3\,\text{ns} - 16/f_\text{s} + \texttt{pins_delay_s} = -47.8\,\text{ns} \tag{15.5}$$

(109.1 ns: an edge's mean delay through the serializer, 108.1 ns for a 2 ns slot in simulation plus 1 ns for the slot after the crossing; −0.3 ns: the sync pin's fast output buffer, measured with the sync looped into the TTL reference at frequencies whose edges sweep both grids), and the driver sets SPH = phase + 360° fT_{sync}. Looped into the TTL reference (DIO1_P → DIO0_P), the sync's edges appear 107.5 ns after the phase they were computed from (2026-10-26).

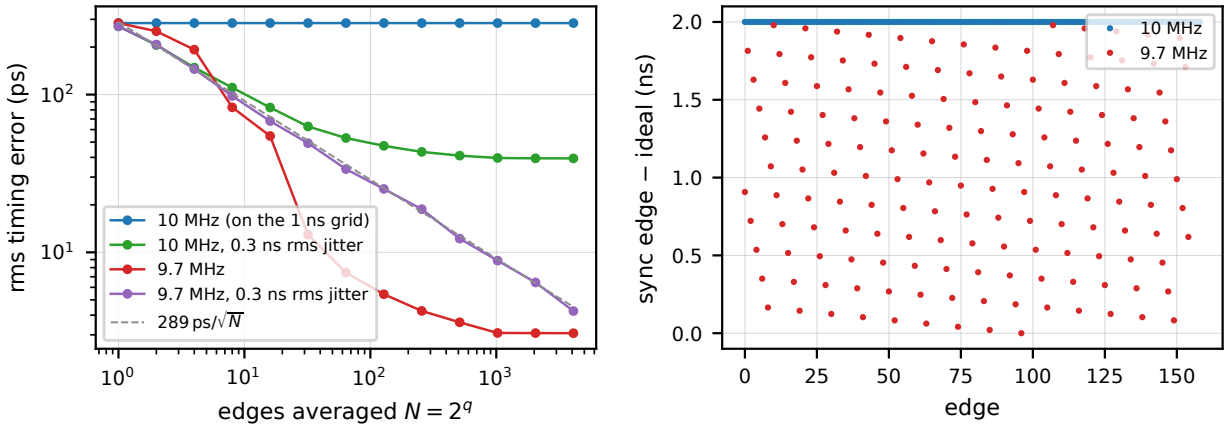


Figure 15.1: Left: the rms timing error of the TTL reference's average of N edges (1 ns timestamps, random starting phases): on the 1 ns grid (10 MHz) averaging does not help without jitter; off the grid (9.7 MHz) or with jitter it does. Right: the sync output's edges against the ideal crossings (`model.sync_bits`): at 10 MHz all edges have the same offset; at 9.7 MHz they walk across the 2 ns slot.

15.4 The capture trigger and the counter

A capture can trigger on DIO3_P's edges (section 17.1). The trigger's sample is where the first edge appears and its sub-sample k (in the capture's status) times it to 1 ns, so captures of a repetitive experiment can be aligned to its trigger to a nanosecond. The frequency counter can count DIO0_P's rising edges with their 1 ns timestamps instead of IN1's or IN2's level crossings (section 17.3).

15.5 Status pins

DIO0_N–DIO3_N show one state each, as levels for other equipment and tests: low, high, an acquisition locked or scanning, a scan rising (a trigger at the scan's turning points for an oscilloscope), a PID locked (section 10.1), a frame's first sample (the frame pulse), an output or an input clipping in the sample. They are registered in the I/O block, not aligned to samples or calibrated in time, except the frame pulse, which the pins' loop measurement uses.

Chapter 16

Slow analog outputs and inputs [feedback]

The Red Pitaya's extension connector E2 has four slow analog outputs (AO0–AO3: pulse-width modulated pins behind RC filters, 0 to 1.8 V) and four slow analog inputs (AI0–AI3: the Zynq's XADC behind dividers). They are too slow for the signal path, but they suit the actuators and sensors of the slow loops: a piezo driver's or a current controller's input, a heater, a temperature sensor, a photodiode's DC level. `feedback` makes them PID targets and sources, calibrated in volts like the fast converters.

16.1 The outputs: PWM with a second-order delta-sigma

Each output is a pin that is high for the first d samples of every PWM period of 256 samples (488.28125 kHz), $d = 0 \dots 256$: on its own an 8-bit converter. Its 16-bit value $v = \text{clamp}(\text{offset} + \text{the PIDs' targets}, 0, 65535)$ is resolved finer by modulating d from period to period with a second-order delta-sigma with error feedback [27] (`model.AoChannel`):

$$y_j = \text{clamp}(v, 256, 65280) + 2e_{j-1} - e_{j-2}, \qquad d_j = (y_j + 128) \gg 8, \qquad e_j = y_j - 256\, d_j. \quad (16.1)$$

Since $256\, d_j = v + 2e_{j-1} - e_{j-2} - e_j$, the duty's error is the quantization error filtered by

$$256\, d_j - v = -(1 - z^{-1})^2 e_j : \quad (16.2)$$

- its mean is exactly zero (the sum telescopes and e stays bounded): the mean duty is exactly $v/65536$, 16 bits of resolution from an 8-bit PWM;
- its spectrum rises as $|2\sin(\pi f/f_p)|^4$, as f^4 at low frequencies: the error is pushed toward the PWM rate, where the filters remove it (figure 16.1, left).

The clamp to $[256, 65280]$ keeps d within $0 \dots 256$ and $|e| \le 128$. A delta-sigma that saturates its duty, as near the rails without the clamp, loses the exact mean. The clamp costs 1/256 of the range at each end. A value of 0 switches the output off: duty 0 and the errors cleared. With a constant input the modulator's error is not white but tonal (the idle tones of a second-order loop), at frequencies that depend on v; they lie above a few kilohertz for values away from multiples of 256, and the filters suppress them with the PWM ripple.

The ripple. The PWM's fundamental at 488 kHz carries most of the AC power: a first-order RC of corner $f_c \ll f_p$ leaves a triangle of peak-to-peak height $V_{\text{FS}}\, d(1-d)\, 2\pi f_c/f_p$ (d the duty fraction), and the delta-sigma's shaped noise, steeper than the RC's slope, adds little (figure 16.1, right). Through the board's own filter the slow inputs read 0.14–0.17 V peak to peak at half scale (AO → AI, measured 2026-10-26 on all four channels); how much the inputs' own front end smooths it is not known, and the ripple at the AO pin has not been measured. Red Pitaya specifies the outputs' filter as a 160 kHz first-order low-pass, which would leave about 0.85 V peak to peak at half scale ($V_{\text{FS}} \tanh(1/(4 f_p \tau))$ for a corner comparable with the PWM rate; computed); the ripple seen by AI would correspond to a corner near 26 kHz (151 mV computed). Either way an actuator needs an

external RC (or a second order) well below the loop's needs: a 100 Hz corner leaves well under a millivolt.

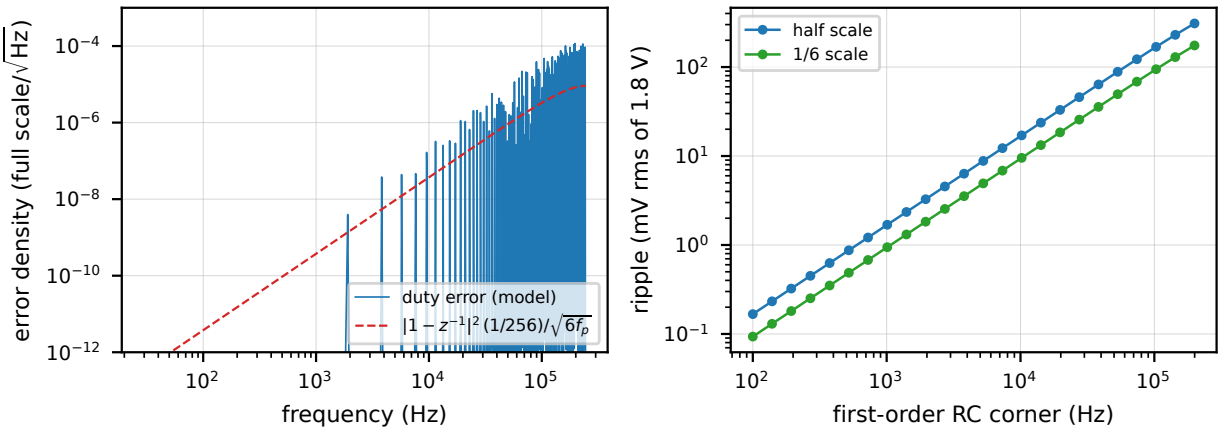


Figure 16.1: Left: the error density of the duty sequence for a constant value (computed with `model.AoChannel`): tonal, below the white-noise approximation of a second-order modulator at low frequencies. Right: the ripple and noise left behind a first-order RC (pin waveform at the sample rate, 1.8 V full scale), against its corner.

Timing. One datapath serves the four outputs in turn (output k in sample $256j + k$); the offsets and targets change only at frame boundaries, so this is the value at the period's start. The four duties take effect together. A PID's AO target therefore acts with up to one PWM period (2 μs) of extra delay, plus the RC's.

Volts. $V = G\,v + B$ from the board calibration's `slow` entries (nominal $G = 1.8\,\mathrm{V}/65536$, $B = 0$ until measured with `CAL SLOW`).

16.2 The inputs: the XADC

AI0–AI3 are the XADC's [28] auxiliary inputs behind the board's dividers. The XADC runs a continuous sequence at a 25 MHz conversion clock over its calibration, the temperature, the supplies (the channels Linux reads through its own XADC driver, which shares the converter) and the four inputs: one sequence in about 14 μs. After each sequence the four results are read over the dynamic reconfiguration port and held; PIDs and acquisitions read the latest word when they run. Each is a 12-bit result for 0 to 1 V at the pins, which the dividers map to about 0 to 7.0 V at the connector: steps of 1.7 mV. Volts: $V = G\,\mathrm{word} + B$ from the calibration (nominal $G = 7.0\,\mathrm{V}/65536$ until measured). The inputs' noise and bandwidth behind the dividers have not been measured.

A slow PID on a slow input is sampled every 14 μs at best, held between sequences: its loop should cross over far below 70 kHz, which is anyway the domain of slow actuators. `tune_pid` measures such a loop from steps like any other (chapter 14).

Chapter 17

Acquisition and measurement

Besides the readings of section 9.3, both designs record signals (the capture), deliver every frame's results continuously (the data stream), count frequencies, and build measurements on these in software: spectra, sweeps, resonance fits, the input-delay calibration. `lockin` also answers a subset of the SR860's remote commands.

17.1 The capture

The capture records four channels of 8192 32-bit words each, of the same samples, into block RAM. Its sources:

- *full-rate*: the converters' codes of IN1, IN2, OUT1, OUT2, the demodulators' inputs, and [feedback] the fast PIDs' errors (half converter steps) and outputs (DAC steps). The capture stores the *sum* over 2^k samples, $k = 0 \ldots 24$ (beyond 16, the sum divided by 2^{k-16}): spans of 8192 words from 65.5 μs to 18 minutes;
- *frame-rate*: X, Y, R, θ, $d\theta$, the PIDs' outputs and errors, the reading of each frame from the readings ring, the same values the snapshot and the stream see.

Summing 2^k samples is a boxcar filter followed by decimation. Its response,

$$|H(f)| = \left| \frac{\sin(\pi f 2^k / f_\mathrm{s})}{2^k \sin(\pi f / f_\mathrm{s})} \right|,$$

has nulls at multiples of the decimated rate but sidelobes of only -13 dB, so components beyond the decimated Nyquist frequency fold back, attenuated by the sidelobes (figure 17.1). For a spectrum of a decimated capture, the top half of the band is affected; a slow capture of a signal with fast components should be read with that in mind (or the demodulator's filtered outputs captured instead).

Triggers. Immediate; software; IN1 or IN2 crossing Schmitt levels (rising or falling, with hysteresis); DIO3_P's edges (`lockin`: through two synchronizer stages; `feedback`: timed to 1 ns, section 15.4). A pretrigger keeps a chosen number of words before the trigger; the trigger is accepted once they are recorded. The capture stage runs with every source of the same sample (15 clocks after the sample counter in `lockin`), so the channels are simultaneous to the sample.

A step captured exactly. `capture_step` arms a frame-rate capture inside the apply that makes a configuration change, just before the apply's write, with 16 words before it: the step's frame is known exactly, whatever the host's latency (an apply takes about 97 ms on the board, longer than a fast capture). The tuning of chapter 14 measures its step responses this way.

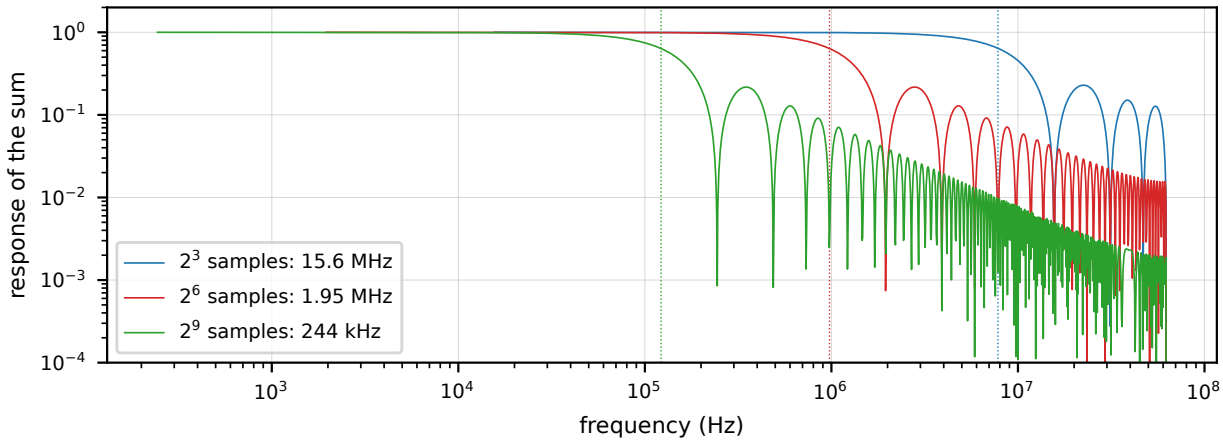


Figure 17.1: The response of the capture's sum over 2^k samples for $k = 3, 6, 9$; dotted: the decimated Nyquist frequencies.

17.2 The data stream

Every 2^j frames the engine pushes a row into a FIFO of 8192 words: the frame number (48 bits), the apply count, the PIDs' locked flags, the frame's clip flags, and X, Y as floats of the chosen demodulators. Rates are $f_o/2^j$: 976 562.5, 488 281.25, … rows per second. The board's service drains the FIFO every 50 ms and sends the rows to its clients; a row that does not fit is dropped and counted, and every row carries its frame number, so a gap is always visible. The sustained rate through the service is about 8000–9000 rows per second (it varies with the board's load); faster streams lose rows, and short bursts are better taken with the capture. An apply that starts a stream clears the FIFO first, so that no earlier stream's rows arrive first. The apply count in each row shows where the configuration changed.

17.3 The frequency counter

The counter counts rising edges of IN1 or IN2 through Schmitt levels (`level` ± `hysteresis`), or [feedback] of DIO0_P with its 1 ns timestamps, in a gate of 2^g samples, and records the samples of the first and the last edge. The frequency is the *reciprocal* count

$$f = \frac{(N-1)\, f_s}{n_{\text{last}} - n_{\text{first}}}, \tag{17.1}$$

whose error is the timestamps' error over the time between the first and last edges, not ±1 count over the gate: a resolution of about

$$\frac{\delta f}{f} \approx \frac{\sqrt{2}\,\sigma_t}{T_{\text{gate}}}, \qquad \sigma_t = \frac{8\,\text{ns}}{\sqrt{12}}\ \text{(IN1, IN2)}, \quad \frac{1\,\text{ns}}{\sqrt{12}}\ \text{(DIO0_P)}, \tag{17.2}$$

independent of the frequency's size relative to the gate. Noise on a sine adds a trigger jitter of $\sigma_v/(2\pi f A)$ per edge (A the sine's amplitude, σ_v the noise's rms); the hysteresis prevents multiple triggers on one noisy crossing but does not remove the jitter (figure 17.2). A 10 ms gate resolves 1 MHz to about 0.3 Hz on an input and 0.04 Hz on DIO0_P. `lock_pll`'s and `lock_ttl`'s seeds use the counter to start their oscillators near the input.

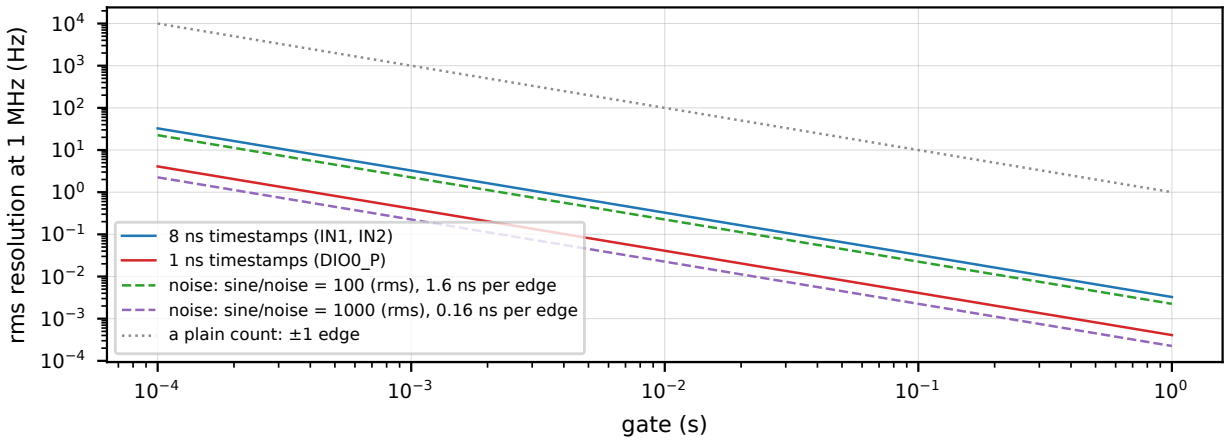


Figure 17.2: The counter's rms resolution at 1 MHz against the gate: the timestamps' quantization on an input and on DIO0_P, the trigger jitter of noise on a sine, and a plain count's ±1 edge.

17.4 Spectra of an input

`spectrum` takes captures of an input (8192 samples at $f_s/2^k$), applies a 4-term Blackman–Harris window (sidelobes below −92 dB, equivalent noise bandwidth 2.00 bins) and averages the power of several captures. It returns the power per bin in dBV rms and the density in $\mathrm{V}/\sqrt{\mathrm{Hz}}$ (the power divided by the bin's ENBW). A tone reads its rms amplitude in its bin, noise its density, both correctly through the window's ENBW. The spectrum around a demodulator's reference with a finer resolution is the zoom FFT of section 8.3.

17.5 Sweeps

A sweep steps one parameter (an oscillator's frequency, amplitude or phase, an output's offset) through a list of values; after each step it waits for the demodulators to settle and averages a few readings of their X, Y, R, θ. The default wait is the slowest demodulator's settling time to 10^{-4} of a step (section 6.3, the engine's latency included), so that every point is a steady-state measurement: a sweep through a resonance of linewidth Γ must also wait for the resonance itself, about $1/(\pi\Gamma)$, which `settle` or `settle_factor` extends. Sweeps nest (an outer parameter stepped around inner sweeps), estimate their duration beforehand, run on the board as an acquisition (another client's configuration edit cancels them) and restore the swept parameters afterwards.

17.6 Fitting a resonance

A driven resonance seen through a lock-in is a circle in the X–Y plane: near its frequency the response of a damped oscillator is

$$z(f) = \frac{a}{1 + 2\mathrm{i}Q(f - f_0)/f_0} + c, \tag{17.3}$$

a complex amplitude a (its size and phase), the quality factor Q (FWHM $= f_0/Q$) and a complex background c (crosstalk, a non-resonant path). `fit_resonance` fits it to a sweep's $X + \mathrm{i}Y$. For given f_0 and Q, a and c enter linearly and follow from linear least squares; a Levenberg–Marquardt iteration adjusts f_0 and Q alone (a variable projection), started from the largest deviation from the median and the half-power width. Figure 17.3 fits a simulated 32.768 kHz tuning fork of $Q = 20\,000$: f_0 within 2.1 mHz, $Q = 19953$, from 121 points with 0.4 mV of noise on a 50 mV resonance. Fitting both quadratures at once uses all the information: R alone loses the phase, and the background distorts a magnitude-only fit.

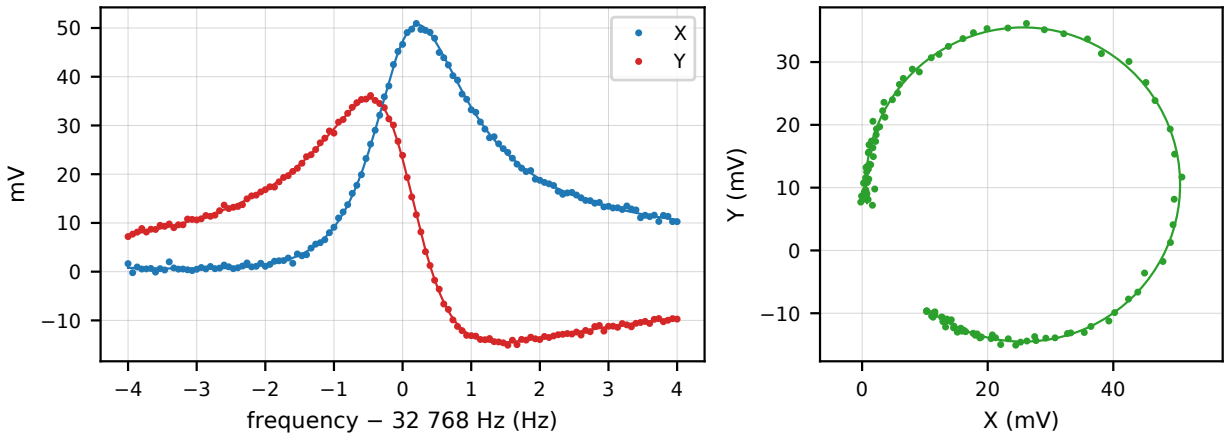


Figure 17.3: A swept resonance (simulated: $f_0 = 32\,768$ Hz, $Q = 20\,000$, phase 30°, background, noise) and the fit (lines): X and Y against frequency (left) and the circle in the X–Y plane (right).

17.7 Calibrations from loopback

The input delay. `measure_input_delay` drives an output into an input through a cable of known delay at five multiples of the frame rate and fits the phase's slope against frequency: the slope is the converter loop delay that the calibration misses (section 2.4, figure 2.3). At multiples of the frame rate the mixers' $2f$ products fall on the CIC's nulls, so a short time constant gives clean phases.

Auto-phase. `auto_phase` averages a demodulator's readings and turns its reference phase by the measured θ, so that the signal lies on $+X$ and Y reads zero: the classic lock-in procedure for reading a signal's in-phase amplitude with all of the filter's noise rejection.

17.8 The SR86x port [lockin]

Programs written for a Stanford Research SR860/SR865 [29] can drive `lockin` on the service's second TCP port (23, the SR860's telnet port). The port maps a subset of the SR860's commands onto demodulator 1, oscillator 1 and OUT1 (`control/lockin/compat.py`; the table is in `lockin/docs/PROTOCOL.md`). The mapping follows the SR860's conventions, not `lockin`'s:

- values are in base units, X, Y, R in V rms, and `SLVL` sets the sine's *rms* amplitude: OUT1's amplitude (a peak value in `lockin`) is $\sqrt{2}$ times it;
- the time constant is chosen from the SR860's table (1 µs to 30 ks in 1–3 steps), the slope 6–24 dB/oct is the filter order 1–4; `ENBW?` reports the realized filter's equivalent noise bandwidth (section 6.3);
- the sensitivity is a per-connection setting used only by `ASCL` (the smallest range at least 1.1 R) and by the status bits for X, Y, R beyond it: the readings are always in volts, the board's input range being set by its jumpers;
- errors set the standard event status bits (command, execution) and answer nothing, as an SR860's do.

Long operations (`APHS`, `ASCL`, `ILVL?`) run as acquisitions on the board, like the native protocol's.

Appendix A

Delays at a glance

Every delay the manual derives, in one place. Times are from the sample counter's origin of sample n (the clock edge where it reads n) unless stated; "board" values were measured on the board used throughout (2026-10).

what	from → to	delay
output latency	sample counter → the core's output word	14 samples (112 ns)
DAC interface	core output → DAC	2 samples
converter loop delay	OUT1's DAC port → an input's ADC port (board, short cable)	119.9–120.0 ns
frame results act	frame k → sample $128(k+4)$	4 frames (4.10 μs)
CIC sampling	a target changed for one frame → the frame values	B-splines: $\frac{1}{2}$ frame (hold), 1 frame (frequency)
slow loop, effective	a slow PID's error → its effect, with the CIC	about 4.5 frames + the plant's delay
PLL, effective	`lock_pll`'s loop (equation 11.3)	about 4.1 frames (4.2 μs)
fast PID, raw input	input connector → output connector	14 samples + 120 ns = 232 ns
fast PID, product	input connector → output connector	18 samples + 120 ns = 264 ns
engagement	source past the level → lock acts	up to 5.4 μs
slow outputs	a PID's AO target → the PWM's duty	up to one PWM period (2.05 μs)
slow inputs	XADC sequence	about 14 μs per update
1 ns pins	an edge → its timestamp (simulation)	−2.5 ns on average
`pins_delay_s`	an output pin → OUT1's sample of the same edge	−28.6 ns (board)
`pins_loop_s`	an output pin → a 1 ns input (jumper)	4.5 ns (board)
TTL reference	an edge at	$T_{\text{ref}} = 158.6$ ns

Appendix B

Symbols

symbol	meaning
f_s, f_o	sample rate (125 MHz), frame rate ($f_\mathrm{s}/128$)
n, k, j	a sample, a frame, an index (a PWM period, a capture word, an edge)
F, ϕ	an oscillator's 48-bit frequency word and phase (cycles)
ψ, m	a demodulator's reference phase; its harmonic
S	the sine's and the reference's full scale ($2^{17}-1$)
x, r_I, r_Q	an input word (half converter steps); the in-phase and quadrature references
X, Y, R, θ	a demodulator's outputs: in-phase, quadrature, magnitude, phase
$d\theta$, Δf	the phase change per frame; the frequency deviation $d\theta\, f_\mathrm{o}$
τ, n, α	an RC section's time constant, the number of sections, $1-\mathrm{e}^{-1/(\tau f_\mathrm{o})}$
ENBW	equivalent noise bandwidth (one-sided, Hz)
e_n	a white noise's one-sided density ($\mathrm{V}/\sqrt{\mathrm{Hz}}$)
σ_X, $\sigma_y(\tau)$	the standard deviation of X; the Allan deviation at averaging time τ
z	the unit-delay variable, $\mathrm{e}^{\mathrm{i}2\pi f/f_\mathrm{s}}$ or $\mathrm{e}^{\mathrm{i}2\pi f/f_\mathrm{o}}$
C, P, L, T, S	controller, plant, open loop CP, closed loop $L/(1+L)$, sensitivity $1/(1+L)$
p, i, d	proportional, integral (s^{-1}), derivative (s) gains in physical units
k_p, k_i, k_d	the same in register units
f_c, PM, GM	crossover frequency, phase margin, gain margin
bw	a PLL's bandwidth (its crossover)
$g(z)$, h_j	the frame-rate sampling of a frequency (cubic B-spline) and of a held target
B_L	a PLL's noise bandwidth $\int_0^{f_\mathrm{o}/2} \lvert T\rvert^2 \mathrm{d}f$
T, T_e	a loop's delay; the engagement delay of a lock acquisition
w, q, M, N	a window (2^w frames); an average (2^q edges); the TTL reference's divider and multiplier
d, v, e	a slow output's duty, value and delta-sigma error
$\gg$, $\lfloor\cdot\rfloor$, sat_b, clamp	arithmetic right shift (floor), floor, saturation to b bits, clamping

Appendix C

Reproducing the figures

Every computed figure and every number quoted with it comes from one host program, `lockin/tools/theory_figures.py`, which imports the designs' bit-exact models (`control/lockin/model.py`, `control/feedback/model.py`), their analysis functions, and for the tuning figures `feedback`'s driver on its model (`feedback/tests/fake_core.py`). It writes `figures/th_*.pdf` and `theory_values.tex`, whose `\val` macros the text quotes:

```
python3 lockin/tools/theory_figures.py              # all figures (several minutes)
python3 lockin/tools/theory_figures.py --only pll_pullin fast_delay
cd lockin/docs/manual && latexmk -pdf theory_manual.tex
```

It needs numpy and matplotlib. The board's measurements quoted in the text are from the designs' `docs/VERIFICATION.md`, dated.

Bibliography

[1] M. L. Meade. *Lock-in amplifiers: principles and applications*. Peter Peregrinus, London, 1983.

[2] Stanford Research Systems. About lock-in amplifiers. Application note 3, https://www.thinksrs.com.

[3] Zurich Instruments. Principles of lock-in detection and the state of the art. White paper, https://www.zhinst.com, 2016.

[4] J. Tierney, C. M. Rader, and B. Gold. A digital frequency synthesizer. *IEEE Transactions on Audio and Electroacoustics*, 19(1):48–57, 1971.

[5] Stanley P. Lipshitz, Robert A. Wannamaker, and John Vanderkooy. Quantization and dither: a theoretical survey. *Journal of the Audio Engineering Society*, 40(5):355–375, 1992.

[6] David Blackman and Sebastiano Vigna. Scrambled linear pseudorandom number generators. *ACM Transactions on Mathematical Software*, 47(4):36:1–36:32, 2021.

[7] E. B. Hogenauer. An economical class of digital filters for decimation and interpolation. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 29(2):155–162, 1981.

[8] Alan V. Oppenheim and Ronald W. Schafer. *Discrete-time signal processing*. Pearson, Upper Saddle River, NJ, 3rd edition, 2010.

[9] Jack E. Volder. The CORDIC trigonometric computing technique. *IRE Transactions on Electronic Computers*, EC-8(3):330–334, 1959.

[10] Ray Andraka. A survey of CORDIC algorithms for FPGA based computers. In *Proceedings of the 1998 ACM/SIGDA Sixth International Symposium on Field Programmable Gate Arrays*, pages 191–200, 1998.

[11] S. O. Rice. Mathematical analysis of random noise. *Bell System Technical Journal*, 23(3): 282–332, 1944.

[12] Fredric J. Harris. On the use of windows for harmonic analysis with the discrete Fourier transform. *Proceedings of the IEEE*, 66(1):51–83, 1978.

[13] Peter D. Welch. The use of fast Fourier transform for the estimation of power spectra: a method based on time averaging over short, modified periodograms. *IEEE Transactions on Audio and Electroacoustics*, 15(2):70–73, 1967.

[14] David W. Allan. Statistics of atomic frequency standards. *Proceedings of the IEEE*, 54(2): 221–230, 1966.

[15] W. J. Riley. Handbook of frequency stability analysis. NIST Special Publication 1065, National Institute of Standards and Technology, 2008.

[16] Enrico Rubiola. *Phase noise and frequency stability in oscillators*. Cambridge University Press, Cambridge, 2008.

[17] B. P. Welford. Note on a method for calculating corrected sums of squares and products. *Technometrics*, 4(3):419–420, 1962.

[18] Floyd M. Gardner. *Phaselock techniques*. Wiley, Hoboken, NJ, 3rd edition, 2005.

[19] Roland E. Best. *Phase-locked loops: design, simulation, and applications*. McGraw-Hill, New York, 6th edition, 2007.

[20] R. W. P. Drever, J. L. Hall, F. V. Kowalski, J. Hough, G. M. Ford, A. J. Munley, and H. Ward. Laser phase and frequency stabilization using an optical resonator. *Applied Physics B*, 31(2): 97–105, 1983.

[21] Eric D. Black. An introduction to Pound–Drever–Hall laser frequency stabilization. *American Journal of Physics*, 69(1):79–87, 2001.

[22] Karl Johan Åström and Richard M. Murray. *Feedback systems: an introduction for scientists and engineers*. Princeton University Press, Princeton, 2008.

[23] Sigurd Skogestad. Simple analytic rules for model reduction and PID controller tuning. *Journal of Process Control*, 13(4):291–309, 2003.

[24] Karl Johan Åström and Tore Hägglund. *Advanced PID control*. ISA, Research Triangle Park, NC, 2006.

[25] *7 Series FPGAs SelectIO resources user guide*. Xilinx, . UG471.

[26] *Vivado design suite 7 series FPGA and Zynq-7000 SoC libraries guide*. Xilinx, . UG953.

[27] Richard Schreier and Gabor C. Temes. *Understanding delta-sigma data converters*. Wiley-IEEE Press, Piscataway, NJ, 2005.

[28] *7 Series FPGAs and Zynq-7000 SoC XADC dual 12-bit 1 MSPS analog-to-digital converter user guide*. Xilinx, . UG480.

[29] *SR860 500 kHz DSP lock-in amplifier, user manual*. Stanford Research Systems, Sunnyvale, CA, 2016.