atomic_clock 3.2: reference documentation

overview, theory, control reference, registers, test records

Rok Zitko, Jozef Stefan Institute

2026-11-10 (commit 6178370)

Contents

atomic_clock: a 10 MHz source with programmable phase and phase noise **2**
- Before connecting equipment . . . 2
- Quick start . . . 3
- Simulating a clock . . . 4
- Presets . . . 5
- Specifications . . . 5
- Hardware setup . . . 6
- Documentation . . . 7
- Changing the design . . . 7
- Source layout . . . 7
- Verification . . . 8
- Author . . . 8

How atomic_clock works **8**
- Signal flow . . . 8
- The exact carrier . . . 9
- The phase word . . . 9
- Phase to sine . . . 9
- Output stage . . . 10
- Physical units . . . 10
- Delays . . . 10
- What the output can resolve . . . 10
- Phase control . . . 11
- PPS outputs . . . 11
- Time of day . . . 12
- Phase meter . . . 12
- Noise generators . . . 13
- Phase-noise playback . . . 13
- Stability recordings and their analysis . . . 14

Control reference **15**
- Configuration . . . 15
- Playback . . . 17
- Stability recordings . . . 17
- Meter stream . . . 18
- Presets . . . 19
- Python methods . . . 19
- TCP commands . . . 21
- Command line . . . 22
- Where the state is kept . . . 22

Register map **22**
- Configuration (shadowed: take effect on commit) . . . 22
- Control and status . . . 23
- Snapshot (read-only; captured by CTRL bit 5 on one clock edge) . . . 24

Verification record **25**
2026-10-17: theory manual units and oscillator specifications . . . 25
2026-10-14: one-day loopback floor repeated on build 0x4bcaf5ab (T0b-long) . . . 25
2026-10-13: live checks before deployment (commit 82845fd) . . . 26
2026-10-13: output and input levels at 10 MHz (oscilloscope) . . . 26
2026-10-13: upload of the second review fixes; loopback hardware test . . . 26
2026-10-12 (night): second review of the review fixes . . . 27
2026-10-12 (late): review fixes — recordings, segmented statistics and RTL . . . 27
2026-10-12 (night): documentation sweep before handoff . . . 29
2026-10-12 (evening): second bug hunt (board chain and rustle) . . . 30
2026-10-12 (afternoon): rustle slow error-bar tests . . . 31
2026-10-12 (afternoon): board upload; frequency offset during noise playback . . . 31
2026-10-12 (afternoon): rustle error bars after the code review . . . 31
2026-10-12 (later): rustle edge cases, dashboard test, long runs, more clocks . . . 32
2026-10-12: six hours of MEAS STREAM; rustle with the measured floor . . . 33
2026-10-11 (evening): 3.2 on the board; MEAS STREAM; rustle on the live meter . . . 33
2026-10-11: rustle, the live estimator (host only; no board involved) . . . 34
2026-10-11: version 3.2 (code-review fixes; built, not yet on the board) . . . 34
2026-10-11: loopback floor over one day (T0b-long, long tier) . . . 35
2026-10-11: adev1-compatible table (host software) . . . 35
2026-10-10: further estimators in the stability tools (host software) . . . 35
2026-10-10: theory and user's manuals (documentation) . . . 36
2026-10-10: oscillator model catalogue (software, 3.1) . . . 36
2026-10-09: stability recordings, analysis tools and the quick tier on the board (software on 3.1, build 0xe2be7288) . . . 36
2026-10-06: version 3.1 (PPS time of day; build 0xe2be7288) . . . 37
2026-10-06: version 3.0 (PPS outputs, phase meter, unwrapped phase; build 0x4c4f8b82) . . . 37
2026-10-06: version 2.0 on the board (build 0x4bd7f0e3) . . . 37
2026-10-05: version 2.0, alignment (build 0x4bd7f0e3) . . . 38
2026-10-05: version 2.0 (smooth phase control, frequency offset, phase noise and playback) . . . 38
2026-10-05: version 1.0 (the MVP: 10 MHz on both outputs, stepped phase) . . . 38

atomic_clock: a 10 MHz source with programmable phase and phase noise

atomic_clock makes the Red Pitaya a **10 MHz** source for testing equipment that expects an atomic-clock reference. Its purpose is to **simulate the phase behavior of a real clock**, from microseconds to a year: OUT1's phase can be programmed with a resolution of 2^{-64} cycle, it moves smoothly, and it can carry a frequency offset, a drift and **phase noise** (white and flicker PM, white and flicker FM, random-walk FM, sinusoidal PM, or a recorded time-error series). The carrier itself has **no drift** relative to the board's sample clock at any time scale. OUT2 carries the same 10 MHz without any of this, as a reference.

```
OUT1 = amplitude1 · sin(2π·(10 MHz · t + φ(t))) + offset1
OUT2 = amplitude2 · sin(2π·10 MHz · t) + offset2           (volts at the connectors)
φ(t) = phase (approached smoothly) + frequency offset · 10 MHz · t + noise
```

White phase and frequency noise are generated in the FPGA at every sample (up to 62.5 MHz offsets); the slower noise types, drift and recorded data are played back by the board's control service through a hardware queue, with the phase integrated exactly (no rounding drift over a year) and logged.

Like a real atomic clock it also has **1 PPS outputs** (the simulated clock's and a reference, on the extension connector), and a built-in **phase meter** on IN1 and IN2: cabled in loopback, the board measures its own outputs, or it compares an external 10 MHz with its internal one.

Amplitudes and offsets are **real volts at the SMA connectors**, through the **board calibration** shared by every design (measured with mult_add; ../common/CALIBRATION.md). The 10 MHz is exactly 0.08 times the board's 125 MHz clock: its absolute accuracy and its own phase noise are those of the board's oscillator, which have **not been measured**. Compare OUT1 with OUT2 to see the programmed phase without the board clock's fluctuations.

Before connecting equipment

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

- **Loading the design starts both outputs as a 10 MHz sine.** Whenever atomic_clock is loaded (at power-up, by `LOAD atomic_clock`, by creating the driver while another design runs, by `make test-remote`), both outputs carry the 10 MHz at amplitude 0.5 V in raw converter units, i.e. without calibration: about **0.57 V** on this board (computed from the calibration, not measured), in phase. After a reboot, the service restores the last configuration.
- **The phase moves by itself** when playback or noise is on, and a new phase setting is approached at the `phase_slew` rate (default 10^{-6}: a quarter cycle in 25 ms). With `phase_slew` 0 a phase change is a step: the waveform jumps on one sample, and equipment locked to the 10 MHz may lose lock.
- **Setting a phase or loading a preset resets the accumulated phase.** A frequency offset or noise moves OUT1 away from OUT2; setting `phase` (or a preset, `RESET`, `SAFE`) brings it to exactly that phase again, at the slew rate. Changing only the frequency offset, noise or playback keeps the accumulated phase. `PHASE 0` preserves the frequency offset, so a running phase ramp continues from the new phase.
- **Playback restarts by itself.** After a reboot the service restores the last settings, including noise and playback (it resumes the saved model). `SAFE` turns the sines, noise and playback off.
- **PPS pins are 3.3 V logic outputs** (DIO1_P, DIO2_P on connector E1), not 5 V tolerant; connect them only to high-impedance logic or counter inputs.
- **There is no clamp.** offset ± amplitude is limited to the DAC range, about ±1.13 V on this board; into a high-impedance load the 10 MHz sine is about twice the set amplitude (next item).
- **Safe state:** `SAFE` (Python `clock.safe()`) turns both sines, the noise and playback off; both outputs sit at the calibrated 0 V.
- **Load:** the set **amplitude is the 10 MHz level into 50 Ω**: 0.5 V gave 1.00 Vpp into a 50 Ω feed-through but 2.12 Vpp into a 1 MΩ scope input (measured 2026-10-13, 1 m coax). At 10 MHz the outputs behave like a 50 Ω source; the **offset** (DC) is right into any load. Many 10 MHz inputs are 50 Ω and expect a particular level (for example +7 dBm, which is 0.71 V amplitude into 50 Ω): check the receiving equipment's specification.

Quick start

The board is set up (../common/SETUP.md) and reachable at `192.168.0.149`. Connect a scope to OUT1 and OUT2 through 50 Ω feed-throughs (the set amplitude; at 1 MΩ you see about twice it); trigger on OUT2.

- **Jupyter** on the board: open `http://192.168.0.149:9090` (password `xilinx`), then `atomic_clock/atomic_clock.ipynb`, and run the cells. The control panel has five tabs: **Outputs** (amplitude, offset and sine on/off per output; the OUT1 phase as slider and text box, with the exact phase word, its slew and frequency offset; **Refresh from hardware**, **Reset both**, **Mute all**), **Noise** (the noise model, playback settings and status, and a preview of the model's Allan deviation and L(f)), **Meter & PPS** (phase measurement, Allan deviation of IN1 − IN2, PPS width, UTC alignment), **Presets**, and **Calibration**.
- **Python on the board:**

```
import rpfpga                                  # must come first
from pynq.overlays.atomic_clock import DRIVER
clock = DRIVER()                              # loads atomic_clock if the board runs something
 ↪  else
clock.load_preset("reference")
clock.set_phase(0.25)                          # OUT1 moves 25 ns ahead of OUT2 (in 25 ms)
clock.set_frequency_offset(1e-11)              # then gains 10 ps per second
print(clock.describe())
```

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

```
import sys; sys.path[:0] = ["common/python", "atomic_clock/control"]
from atomic_clock import AtomicClockClient
with AtomicClockClient("192.168.0.149") as clock:   # the board must run atomic_clock
    clock.set(1, amplitude=0.3, phase=-1e-6)        # 1e-6 cycle = 100 fs behind OUT2
    clock.set_noise(h_1=1e-26, h0=2e-24, seed=1)    # flicker FM + white FM
    clock.set_playback(source="model")              # played by the board's service
    print(clock.playback_status())
```

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

- **TCP** from anything: `nc 192.168.0.149 5025`, then for example `PHASE 0.25`, `FREQ 1e-11`, `NOISE JSON {"h_1": 1e-26}`, `PLAY MODEL`, `PLAY?`, `MEAS? 10`, `PPS UTC`, `PPS?`, `AMP1 0.3`, `OUT2 OFF`, `STATE?`, `PRESET reference`, `SAFE`.

- **Command line:** `python3 -m rpfpga.cli --host 192.168.0.149 --phase 0.25` (with `common/python` and `atomic_clock/control` on `PYTHONPATH`).

docs/PROTOCOL.md lists every method, TCP command and command-line option.

Simulating a clock

The commands below are TCP lines (`nc 192.168.0.149 5025`); every one has a Python equivalent in docs/PROTOCOL.md. Start from `PRESET reference`. OUT2 stays the clean reference throughout; compare OUT1 against it with a phase comparator or time-interval counter, or with the board's own meter in loopback (below).

Deterministic behavior.

- `FREQ 1e-11`: a frequency offset of 10^{-11} (OUT1 gains 10 ps per second, 0.86 µs per day), integrated exactly.
- `NOISE JSON {"drift_per_day": 1e-13}` together with `PLAY MODEL`: a linear frequency drift of 10^{-13} per day.
- `NOISE JSON {"pm_sine_amplitude": 1e-4, "pm_sine_frequency": 1}` with `PLAY MODEL`: sinusoidal phase modulation (here 10 ps at 1 Hz).
- `SLEW 1e-9` then `PHASE 0.5`: a 50 ns phase step taken as a smooth ramp (here over 50 s).

Noise of a given Allan deviation. The noise is set by the power-law coefficients h (IEEE 1139): `h2` white PM, `h1` flicker PM, `h0` white FM, `h_1` flicker FM, `h_2` random-walk FM. White PM and white FM are produced in the FPGA and need nothing else; the others need `PLAY MODEL`, which the board's service plays and logs. Convert Allan deviations to h in Python:

```
from atomic_clock import noise          # on the board: from pynq.overlays.atomic_clock import noise
noise.h_from_adev("h0", 1e-12, 1.0)      # white FM with ADEV 1e-12 at 1 s  -> 2e-24
noise.h_from_adev("h_1", 1e-14, 1.0)     # flicker floor of 1e-14            -> 7.2e-29
noise.h2_from_dbc(-150)                  # white PM floor of -150 dBc/Hz     -> 2e-29
```

then, for example, `NOISE JSON {"h0": 2e-24, "h_1": 7.2e-29, "seed": 1}` and `PLAY MODEL`. `PLAY?` shows whether it plays; the Noise tab of the panel previews the resulting Allan deviation and L(f). The same seed reproduces the same noise.

Replaying a recorded clock. Write the time error of a real clock (seconds, positive = ahead) as two columns, time in seconds and time error in seconds, in a CSV file (comma, semicolon, tab or space separated; # comments), for example `lab.csv`, and upload it (at most 50 MB):

```
with AtomicClockClient("192.168.0.149") as clock:
    clock.upload_replay("lab", "lab.csv")            # TCP: REPLAY PUT ... / REPLAY DONE
```

Then `PLAY REPLAY lab`. Between points the time error is interpolated linearly; after the last point it is held, or with `PLAY JSON {"replay_loop": true}` the record repeats, continuing smoothly. `PLAY JSON {"replay_scale": 10}` multiplies it.

Back to the reference. `RESET` (or `PRESET reference`): playback and noise off, frequency offset 0, and OUT1 exactly in phase with OUT2 again, including any phase accumulated so far. Playback stops synchronously with the settings change, before the phase is realigned. It moves back at the default slew (10^{-6}: 10 cycles = 1 µs per second); `SLEW 0` sent right after it finishes the move at once (`SLEW 1e-6` restores the default). To keep the other settings, use `PLAY OFF`, `NOISE JSON {"h2": 0, "h0": 0}`, `FREQ 0`, `PHASE 0` instead. Starting the same model or replay again continues where it stopped (also after a reboot); changing any of its settings (for example the seed) starts it from the beginning.

Measuring with the board itself. Cable OUT1 → IN1 and OUT2 → IN2 (equal cables), then `MEAS? 100`: IN1 − IN2 is OUT1's phase relative to OUT2 (plus a fixed cable difference, measured at `PRESET reference`). The Meter & PPS tab computes the Allan deviation of a 2 s record; `MEAS LOG ON` lets the service log the phase continuously (`/home/xilinx/rpfpga/atomic_clock/meter/`). The service's `MEAS? n` averages n consecutive worker windows even during logging or a recording; control changes cancel a pending measurement. Input calibration and the configured LV/HV ranges set the reported amplitudes in volts. White PM above the meter's window rate (100 Hz by default) averages out and is not seen.

Long-term stability (Allan deviation over hours to days). The service records the meter on the board while you are away: every 10 ms window for the first hour and one window per second for the whole run, with the board temperature and the complete settings. From a computer with numpy, scipy, matplotlib and allantools (for example the conda environment `base-tools`), in `tools/`:

```
python stability.py start floor1 --duration-h 24     # REC START floor1 JSON {"duration_h": 24}
python stability.py status
python stability.py fetch floor1                     # -> recordings/floor1/
python stability.py analyze recordings/floor1 -o floor1.pdf
python run_tests.py --tier quick --dry-run           # the testing manual's tests, run in sequence
python stability.py hat recordings/clocks            # two external clocks on IN1, IN2:
 ↪  three-cornered hat
python stability.py analyze-file counter.txt --units ns --tau0 1 --oscillator ocxo   # another
 ↪  instrument's data
python adev1.py 1 1e-9 < counter.txt                 # quick table in the format of TvB's adev1
```

Turn the dither on whenever OUT1's phase moves (noise, playback, frequency offset: `NOISE JSON {"dither": true}`): without it the DAC's rounding adds up to 1.5–3× in ADEV at short τ (measured; the test runner does it for you). Before long runs, disable the board's scheduled maintenance jobs (SETUP.md, step 6): they can stall the service. All quantities (h_α, $S_y(f)$, $L(f)$, the deviations) follow the definitions of IEEE Std 1139 (summary in the testing manual), so settings can come from data sheets and results compare with them directly. The PDF report has ADEV, MDEV, Hadamard and time deviations with confidence intervals, the expected curves of the configured noise, a fit of the h coefficients with a pass/fail/inconclusive verdict, $S_y(f)$ and $L(f)$, and the time series; then Totdev and ThéoBR to long τ, MTIE and TIE rms (with ITU-T G.811/G.8262 masks, `--mask`), the dynamic ADEV of long runs, and optionally logarithmic aging (`--drift log`, removing the full fitted aging trajectory before fitting the noise). Incomplete configured-noise checks and phase steps across broken unwrapping branches are inconclusive. The test runner rejects write errors, invalid sample cadence or counts, and acquisition coverage below 95 %. The same report works on phase or frequency files from other instruments (`analyze-file`), and `hat` separates two external clocks and the board's own clock (three-cornered hat). The testing manual describes a series of tests (white and flicker PM, white, flicker and random-walk FM, drift, sinusoidal PM) with suggested settings and the plots to expect. The user's manual has a catalogue of 11 real oscillator types with typical values and ready settings, from a bare crystal through TCXO, OCXO and MEMS to rubidium, caesium, a hydrogen maser and a GPS-disciplined OCXO; the testing manual's M runs play each for three days (`run_tests.py --tier models`) and show their expected ADEV/MDEV against the board's floor. Six deliberately bad clocks (jitter, a crystal next to a hot CPU, 50 Hz ripple, popcorn noise, frequency jumps, a badly installed GPSDO) show their faults within minutes, for demonstrations: `python tools/stability.py play popcorn` plays one on OUT1 (`play off` resets), `run_tests.py --tier demos` records each for one to two hours. D-mains uses 5 ms windows (200 samples/s), keeping its 50 Hz modulation below Nyquist.

Watching the noise live. `python -m rustle` (in `tools/`) is a dashboard in the browser that estimates h_2 … h_{-2}, the frequency offset and the drift while the data arrive, with confidence intervals that tighten as you watch. It takes the demo clocks of the catalogue, downloaded recordings, other instruments' files and the board's meter, live (`MEAS STREAM`). The user's manual, "Watching the noise live", introduces it; see tools/rustle/README.md.

PPS and time of day. DIO1_P carries the simulated clock's PPS and DIO2_P the reference PPS. `PPS UTC` puts the reference on a UTC second (NTP, ms accuracy) and `PPS JSON {"utc": true}` keeps it so after reboots; `PPS?` then reports the time of day, `board_error_s` (board time minus UTC), `simulated_error_s` (the simulated clock minus UTC) and `board_drift.frequency_offset`, the board crystal's measured frequency offset (reliable after some hours).

Presets

Name	Function
`reference`	both outputs 10 MHz, 0.5 V amplitude, in phase, no noise (the default; the board's `make test-remote` preset)
`quadrature`	OUT1 leads OUT2 by a quarter cycle (25 ns)
`off`	both sines off, outputs at 0 V

Specifications

Figures for this board use its calibration (2026-10-04). Board-wide figures are in the root README.

Quantity	Value
Frequency	10 MHz = 2/25 of the sample clock, exactly: no drift relative to the board clock at any time scale. Absolute accuracy and stability: those of the board's 125 MHz oscillator, **not measured**
Phase (OUT1 relative to OUT2)	unwrapped 96-bit word: $\pm 2^{31}$ cycles (±214 s) in steps of 2^{-64} cycle; the sine uses 2^{-28} cycle = 0.37 fs. Approached at a set maximum fractional frequency (default 10^{-6}), or as a step
PPS outputs	DIO1_P: simulated clock, DIO2_P: reference; 3.3 V LVCMOS, 20 µs default width; ideal edge shift −phase/10 MHz, rounded to the 8 ns grid. At +1 cycle: 96 ns advance and a fixed +4 ns remainder. Remainder reported, normally below 8 ns but larger on jumps; digital alignment verified, analog alignment with the sine not measured
Time of day	`PPS UTC` puts the reference PPS on a UTC second via the board's NTP-synchronized clock (ms accuracy at arming); afterwards it follows the board crystal, whose drift against UTC is measured and logged (not yet measured on this board)
Phase meter	I/Q at 10 MHz on IN1 and IN2, windows of 200 ns … 1 s (default 10 ms), exact DC rejection; measured white phase noise $2.9 \cdot 10^{-7}$ cycle (29 fs) per 10 ms window at 0.5 V in loopback (one-day floor recording)
Frequency offset	$\pm 10^{-3}$, resolution $6.8 \cdot 10^{-19}$, integrated exactly
Hardware noise	white PM (h2) and white FM (h0), new every 8 ns sample, Gaussian-like (tails at 4.9σ), seeded; optional dither for sub-step phase linearity
Played-back noise	flicker PM (h1), flicker FM (h−1), random-walk FM (h−2), drift, sinusoidal PM, or a recorded time error; up to half the segment rate (default 1000 segments/s: 500 Hz) in offset frequency; the realized phase is within $3.4 \cdot 10^{-15}$ cycle of the programmed one at every segment start
Sine computation	within 1 LSB of 2^{17} ($7.6 \cdot 10^{-6}$ of full scale) in simulation, below one DAC step ($1.2 \cdot 10^{-4}$)
Amplitude	0 to about 1.13 V peak into 50 Ω (about 2.1× into 1 MΩ, measured at 0.5 V), steps of about 35 µV on this board
Offset	steps of one DAC step, about 138 µV; offset ± amplitude within about −1.143 … +1.128 V (OUT1, this board)
Phase resolved by a single sample	one DAC step at the zero crossing: $3.9 \cdot 10^{-5}$ cycle (3.9 ps) at 0.5 V nominal amplitude
Spurious lines	14-bit rounding gives lines at multiples of 5 MHz, an estimated −90 dBc at 0.5 V (not measured); DAC images at 115 and 135 MHz, about −21 dBc before the output's analog filter (not measured)
Harmonics, phase noise, OUT1–OUT2 skew at the connectors	**not measured**
Delay of a setting change	To the FPGA's output stream: 104 ns from static/integrated phase, 96 ns from white-FM phase, 88 ns from white PM; 24 ns for amplitude/offset. Phase alignment adds a settling interval (THEORY.md); software calls are not real-time (milliseconds)
Settings after loading	10 MHz on both outputs at 0.5 V uncalibrated (about 0.57 V on this board), in phase, no noise, playback off

Hardware setup

- **Inputs: the phase meter.** IN1 and IN2 measure 10 MHz signals (LV jumpers, at most about ±1 V). Match the software's input ranges to the physical jumpers for correct amplitude readings. For the self-test cable OUT1 → IN1 and OUT2 → IN2, both the same way: in direct loopback a 0.5 V sine reads about 0.67 V (the input loads the output), with 50 Ω terminators 0.5 V; a terminator on one input only shifts its phase by about 0.1 cycle.
- **PPS outputs** on the extension connector E1: **DIO1_P** the simulated clock's 1 PPS, **DIO2_P** the reference 1 PPS (board timebase); 3.3 V LVCMOS, **not 5 V tolerant**, pulses of 20 µs by default. Ground is on the same

connector. Never drive these pins from outside.

- **Calibration:** measured once with mult_add (../common/CALIBRATION.md). Without it, nominal converter constants are used, which on this board are about 14 % off in amplitude.
- **Filtering:** for a clean spectrum, add a low-pass filter (cut-off about 20–30 MHz) at the output to remove the DAC images near 115 and 135 MHz.
- **A quick check:** with `PRESET quadrature`, a scope triggered on OUT2 shows OUT1 25 ns earlier, both 10.000 MHz.

Documentation

Document	Contents
docs/manual/user_manual.pdf	user's manual: setup from an empty SD card, getting started, controls, recipes, oscillator catalogue, application notes (counter check, NTP server on the PPS, GNSS satellite clock replay, holdover, teaching the noise types)
docs/manual/theory_manual.pdf	theory manual: how the simulated clock is made, at the level of the arithmetic (exact carrier, phase word, noise generators, playback, output stage, PPS), with derivations, code excerpts and references
docs/PROTOCOL.md	configuration, presets, Python methods, TCP commands, command line
docs/THEORY.md	how it works: the exact carrier, phase word, sine computation, error budget, phase control, noise generation and playback
docs/manual/testing_manual.pdf	long-term stability tests: recording, analysis, the tests with suggested settings and expected Allan-deviation plots; applications of the phase meter (PLL transfer, the board's crystal, two external clocks, cable delay, amplifier residual noise)
docs/REGISTERS.md	the hardware registers (for debugging)
docs/IDEAS.md	ideas for future expansion, ranked
docs/VERIFICATION.md	what was tested, when, and the build's resource use and timing
../common/CALIBRATION.md	the board calibration, shared by every design
../common/README.md	the shared framework: TCP service, common API, build workflow
NOTICE.md	sources
AGENTS.md	notes for maintainers and coding agents

Changing the design

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

```
make                        # create the project (first time), build, export the overlay
make test                   # offline tests: tables, driver, playback worker, protocol, client,
↪  command line, GUI
make sim                    # simulation: system ID, sine, noise, phase path, PPS, meter, system
make test-remote            # upload, program the board, check identity and power-up state
make install-service        # the control service, with atomic_clock active
python3 ../common/tests/verify_build.py --design atomic_clock
```

`make test-hw` runs `scripts/test_hw.py` on the board: with OUT1 → IN1 and OUT2 → IN2 cabled and the service installed, it measures phase steps, the slew, the frequency offset, white FM and played-back flicker FM, and the PPS counters with the board's own phase meter. Check the outputs with a scope, a frequency counter or a spectrum analyzer.

Source layout

Path	Contents
`design.tcl`	design description for the common recipe: ID "ACLK", RTL, one `ac_core` cell
`src/ac_core.v`	top: registers, noise, phase path, carrier, two sine paths, two output stages
`src/ac_phase.v`	OUT1 phase: slewed static phase, frequency offset and segment FIFO accumulator, noise terms, snapshot
`src/ac_noise.v`	white PM and white FM noise generators and the dither
`src/ac_pps.v`, `src/ac_meter.v`	the PPS outputs; the I/Q phase meter
`constraints/pps.xdc`	the PPS pins
`scripts/test_hw.py`	the loopback hardware test (`make test-hw`)
`src/ac_carrier.v`	the exact 25-sample carrier phase table and the OUT1 phase adder

Path	Contents
src/ac_sine.v, src/ac_sine_rom.v	phase to sine (table + linear interpolation); the generated table
src/ac_out.v	amplitude, offset, rounding to the DAC grid
src/ac_regs.v	AXI4-Lite registers with shadow and commit
control/atomic_clock/	Python package: driver, playback worker (playback.py), stability recordings (recording.py), noise conventions (noise.py), TCP commands, client, command-line options, Jupyter GUI
tools/	host-only stability tools: stability.py (record, fetch, analyze to PDF, other instruments' files, three-cornered hat, play a catalogue or demonstration clock), adev1.py (adev1-compatible ADEV table, standard library only), run_tests.py (the testing manual's tests), manual_figures.py and app_figures.py (the manuals' figures), rustle/ (live dashboard: h coefficients, offset and drift as the data arrive), and the library clocklib/ (meter-aware noise model, recordings, synthetic recordings, test plan, replays, runner) on the common stability library ../common/tools/stabilitylib (analysis, further estimators, ITU-T masks, external data, report)
docs/manual/	the user's, theory and testing manuals (LaTeX sources, shared references.bib and style.tex, generated figures and values.tex, the PDFs)
control/atomic_clock.ipynb	the notebook
tests/	tb_sine.sv, tb_noise.sv, tb_phase.sv, tb_pps.sv, tb_meter.sv, tb_system.sv (simulation), the Python tests and fake_core.py (a register model of the core)

Verification

Version 3.2 (build 0x4bcaf5ab, 2026-10-12) runs on the board; the review fixes passed offline regressions, all seven simulations, clean-build verification and targeted live checks. The quick tier passed on 3.1 (2026-10-07/09); the one-day loopback floor (repeated on this build, 2026-10-13) gives $1.2{\cdot}10^{-13}$ at 1 s and $2.9{\cdot}10^{-17}$ at 18 h (ThéoBR), and the rest of the long tier is not yet run. Details: docs/VERIFICATION.md.

Author

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

How atomic_clock works

atomic_clock turns the Red Pitaya into a 10 MHz source whose phase can be set with extreme resolution, as the basis of a simulated atomic-clock output with programmable phase noise (from sub-millisecond to year-long time scales). This page explains the signal path, the number formats and the error budget; the theory manual (manual/theory_manual.pdf) explains the same at the level of the arithmetic, with derivations, code excerpts and references. User controls are in PROTOCOL.md, registers in REGISTERS.md.

Signal flow

Everything runs at the converters' sample clock, f_s = 125 MHz (8 ns per sample).

```
mod-25 counter k -> C[k] --+--> θ  = C[k] + φ -> phase→sine -> × A1, + offset1 (+ dither) -> round to
 ↪  DAC -> OUT1
                           |      (64-bit)      (top 28 bits)
OUT1 phase φ ----------------+
                           └-> θ_ref = C[k] --> phase→sine -> × A2, + offset2 -> round to DAC -> OUT2

φ = φ_s + φ_d + φ_w + w_pm
    φ_s  static phase, moved toward its setting at a limited rate (the slew)
    φ_d  integral of the frequency offset and of the played-back segment rates
    φ_w  white-FM noise (random walk), φ_d and φ_w 96-bit unwrapped
    w_pm white-PM noise, new every sample
```

OUT1 and OUT2 go through identical stages with identical delays, so with φ = 0, no dither and equal settings the two outputs are the same sample for sample. The inputs feed the phase meter; the unwrapped φ also drives the simulated clock's PPS output.

The exact carrier

The carrier frequency is f0 = 10 MHz = **(2/25)·f_s exactly**. A conventional direct digital synthesizer adds a binary phase increment every sample, but 2/25 has no finite binary representation (25 contains the factor 5). With the nearest 48-bit increment, for example, the frequency would be wrong by up to $2 \cdot 10^{-14}$ of f0, which accumulates to several whole cycles per year: a simulated clock with a built-in drift.

atomic_clock does not accumulate the carrier phase. Over 25 samples the 10 MHz sine runs through exactly 2 cycles, so the carrier phase at sample n is 2n/25 mod 1, which takes only 25 values. A counter k = n mod 25 selects a constant from a 25-entry table:

```
C[k] = round(frac(2k/25) · 2^64)          (64-bit phase, units of 2^-64 cycle)
```

The rounding error of each entry is at most 2^{-65} cycle and repeats every 25 samples; it never accumulates. The output frequency is therefore exactly 0.08·f_s at every time scale, from one sample to the 4677 years after which the 64-bit sample counter would wrap. `tests/test_control.py` checks every table entry against the exact fraction and that the 25 steps add up to exactly two cycles; `tests/tb_system.sv` checks the hardware sequence.

Absolute frequency. "Exactly" is relative to the board's 125 MHz clock, which comes from the ADC clock oscillator on the board. Its absolute accuracy (a crystal oscillator, typically tens of ppm) and its own phase noise have not been measured here. The phase offsets programmed in OUT1 are **relative to the board clock**, and OUT2 shares that clock: comparing OUT1 against OUT2 removes the board oscillator's own fluctuations to first order (both outputs carry them equally) and leaves the programmed phase. Referencing the board to an external 10 MHz or 125 MHz standard needs a hardware modification of the board's clock input and is not part of this design.

The phase word

OUT1's public offset φ is an **unwrapped** phase: a 96-bit signed number of 2^{-64} cycle, i.e. 32 bits of whole cycles ($\pm 2^{31}$ cycles = ±214 s of time) and a 64-bit fraction. The sine only needs φ modulo one cycle (its low 64 bits, added to the carrier as θ = C[k] + φ), but the PPS output and the meter need the whole cycles: φ = 3 cycles requests a 300 ns PPS advance (rounded to its 8 ns grid) although the sine looks the same. The fraction's resolution, 2^{-64} cycle = $5.4 \cdot 10^{-20}$ cycle ($5.4 \cdot 10^{-27}$ s at 10 MHz), is far finer than any output can show, but it is the right size for the phase integrator (see "Phase control") so that a constant frequency offset can be integrated for a year without rounding drift:

Quantity	Value
phase word	96 bit signed (32 whole-cycle bits), 2^{-64} cycle
one LSB per sample, as a fractional frequency offset	$2^{-64} \cdot$ f_s / f0 = $6.8 \cdot 10^{-19}$
samples in a year	$3.9 \cdot 10^{15} \approx 2^{52}$
phase used by the sine	the top 28 bits: 2^{-28} cycle = $3.7 \cdot 10^{-9}$ cycle = 0.37 fs

The internal static phase φ_s and its goal `target − origin` are signed 97-bit values, because subtracting two signed 96-bit phases needs one more bit. A move between opposite such goals needs a signed 98-bit difference. The total phase and all public phase snapshots retain only the low 96 bits modulo 2^{96}; the snapshot's signed interpretation does not expose φ_s's extra bit.

The sign convention: OUT1 = A·sin(2π(f0·t + φ)), so a positive φ makes OUT1 **lead** OUT2: its zero crossings come φ·100 ns earlier.

Phase to sine

The top 28 bits p of θ address a 4096-entry full-wave table S[i] = round($(2^{17} - 1) \cdot \sin(2\pi i/4096)$) (18-bit signed, in block RAM, generated into `src/ac_sine_rom.v`). With i = p[27:16] and the 16-bit fraction f = p[15:0], the stage interpolates linearly between S[i] and S[i+1]:

```
y = S[i] + round((S[i+1] − S[i]) · f / 2^16)
```

The interpolation error is at most $(2\pi/4096)^2/8 = 2.9 \cdot 10^{-7}$ of full scale; with the table and product rounding, the result is within one LSB of $(2^{17} - 1) \cdot \sin(2\pi p/2^{28})$. `tests/tb_sine.sv` checks 112 282 phases (every table point,

midpoints and random phases): the worst error is 0.999 LSB of 2^{17}, i.e. $7.6{\cdot}10^{-6}$ of full scale, far below one DAC step ($1.2{\cdot}10^{-4}$ of full scale).

Output stage

The 16-bit stream to the DAC uses only its top 14 bits, so the output stage rounds directly to that grid:

```
q = sat14( floor( (y · A + offset · 2^17 + 2^18 + d) / 2^19 ) )    stream = 4·q
```

d is the optional dither on OUT1 (triangular over $\pm 2^{19}$, i.e. ±1 DAC step; 0 when off).

A (amplitude, 0 … 65535) and offset (signed) are in 16-bit stream codes, nominally 32768 per volt; the board calibration (below) gives the real volts. The peak of the sine is $A{\cdot}(1 - 2^{-17})$ codes. With the sine switched off (`enabled` false) the output is the offset alone. Without dither, rounding is half up; saturation is at the DAC limits (stream codes −32768 and +32764).

Physical units

The board calibration (../../common/CALIBRATION.md) gives each output as V = G·code + B (G volts per stream code, B volts). The driver uses

```
A      = round(amplitude / (G · (1 − 2^-17)))       applied amplitude = G·A·(1 − 2^-17)
offset = 4 · round((offset_V − B) / (4·G))          applied offset    = G·offset + B
```

The offset is placed on the DAC grid (4 stream codes, about 138 µV on this board) so that the DC level is the same with the sine on or off. A configuration whose peak would exceed the DAC range (offset ± A beyond −32768 … +32764 codes) is refused. Nominally (no calibration) 0.5 V is A = 16384, the power-up value; on this board that is about 0.57 V (computed from the calibration, not measured).

Delays

To the core's 16-bit output stream the delays are 13 clocks (104 ns) from φ_s or φ_d, 12 clocks (96 ns) from φ_w, and 11 clocks (88 ns) from w_pm: these terms join successive stages of the phase sum, followed by the carrier adder, 6 sine stages and 3 output stages. From a committed amplitude, offset or on/off: 3 clocks. ALIGN adds the settling sequence below before φ_s moves; at the default slew it then moves at most $10^{-6}{\cdot}f0 = 10$ cycles per second.

The vendor DAC's input register and SAME_EDGE ODDR add two clock stages, verified in simulation and included in the PPS alignment. OUT1 and OUT2 are presented on alternate half cycles of the 125 MHz clock, 4 ns apart. The analog output delay and the skew at the connectors remain unmeasured.

What the output can resolve

- **Single samples.** At the zero crossing one DAC step (1/8192 of full scale) corresponds to a phase of $1/(2\pi{\cdot}A_DAC)$, where A_DAC is the amplitude in DAC steps: $3.9{\cdot}10^{-5}$ cycle (3.9 ps) at 0.5 V nominal, $1.9{\cdot}10^{-5}$ cycle (1.9 ps) at full scale.
- **Averages.** A phase meter averages many samples, so finer phase changes are visible on average, down to roughly the 2^{-28} cycle of the sine input. Without dither the 14-bit rounding error is a fixed pattern that repeats every 25 samples and depends on φ, so the averaged phase is not perfectly linear in φ: a deviation of order 10^{-5} cycle (about 1 ps) is expected. A static phase only sees a constant offset, but a moving phase (noise, playback, an offset) gains short-term error: measured with white FM at $7{\cdot}10^{-25}$, the meter's ADEV was 1.5× the expected value at 10 ms (1.05× at 35 s). The optional dither ("Noise generators") removes it (measured: 1.00×); use it whenever OUT1's phase moves.
- **Spurs.** The same 25-sample rounding pattern puts the quantization error into lines at multiples of 5 MHz instead of a noise floor: an estimated −90 dBc each at 0.5 V (not measured). Analog harmonics of the DAC and output amplifier are probably larger (not measured).
- **DAC images.** The DAC holds each sample for 8 ns: the 10 MHz line is 0.09 dB low, and images appear at 115 MHz and 135 MHz, about −21 dBc before the output's analog low-pass (about 60 MHz, unmeasured attenuation at 115 MHz). For a clean spectrum add an external low-pass filter (about 20–30 MHz cut-off) at the output.

Phase control

Static phase with slew. The `phase` setting is a target. Every 2 clocks the hardware moves φ_s toward it by at most 2R and then lands on it exactly. Whole cycles count: a move from +0.4 to −0.4 cycle goes 0.8 cycle down (it does not take the ”shorter way” through ±0.5), because the PPS distinguishes them. R = round(y_s·(f0/f_s)·2^{64}), where y_s is the `phase_slew` setting, the largest fractional frequency deviation the move may cause (default 10^{-6}: a quarter-cycle move, 25 ns, takes 25 ms). The output phase is continuous; its frequency changes by y_s during the move. y_s = 0 restores steps (version 1.0 behavior). The loop registers the target and up/down candidates together with the slew bounds and step mode used to select them. A setting change between the two stages cannot mix a new limit with an old candidate. The signed 98-bit difference preserves the correct direction even at opposite phase extremes.

Alignment. The frequency offset, the played-back segments and the white-FM random walk accumulate phase in φ_d and φ_w, which are never reset (the playback mirrors φ_d exactly). Setting a phase therefore also *aligns* OUT1: an ALIGN strobe establishes origin = φ_d + φ_w (low 96 bits), and the static phase moves toward the signed 97-bit difference `phase` − origin. The new phase and ALIGN go in one CTRL write. Edge A is the edge at which the phase path sees ALIGN and the committed settings, one clock after COMMIT:

- At A, φ_s freezes and any old slew candidate is discarded, including one due on that edge. The requested target is saved.
- RATE/STOP drain by A+2. The five-register noise scaler first samples the new gain at A; φ_w consumes the last old FM value at A+4.
- A+5 captures the settled φ_d + φ_w; A+6 publishes origin and `target − origin` together. Neither accumulator is paused or reset.
- A+7 prepares the first coherent slew candidate; A+8 first updates φ_s. Repeating ALIGN restarts the settling interval.

Thus OUT1 never uses a new target with an old origin. The accumulated phase at capture is compensated smoothly at the slew rate; any offset, noise or playback left running continues to accumulate through and after the move. With those terms off, the total reaches exactly `phase` relative to OUT2. The driver aligns whenever a patch sets `phase` and whenever a whole configuration is applied (presets, reset, SAFE); changing only the frequency offset, the noise or playback keeps the phase accumulated so far. Whenever the applied configuration has playback off, its COMMIT also carries STOP|CLEAR. In particular RESET stops queued rates synchronously before origin capture; it does not wait for the worker. Before starting a prepared trajectory, the worker rechecks the current model, rate and segment length so a superseding reset or edit is respected.

A SNAP requested during ALIGN waits until publication. One common strobe then captures phase, sample count and PPS state together; the AXI read response also waits until that capture is complete.

Frequency offset. φ_d gains ν_base = round(y0·(f0/f_s)·2^{64}) every sample, where y0 is the `frequency_offset` setting: an exact integer, so the offset is exactly y0 to within $3.4\cdot10^{-19}$ (half an LSB per sample; one LSB is $6.8\cdot10^{-19}$), and it never drifts. For example y0 = 10^{-9} moves OUT1 by 1 ns relative to OUT2 every second, 31.6 ms per year.

Segment FIFO. For anything else the hardware integrates a sequence of rates: a 4096-entry FIFO holds signed 64-bit rates ν_i, each applied for one segment of L samples (`segment_rate` = f_s/L, default 1000 segments per second, so the FIFO holds 4 s). Segments follow one another on exact sample boundaries. If the FIFO runs empty the last rate is held (the phase stays continuous) and a sticky UNDERFLOW flag is set. The hardware latches, together, φ_d at the start of each segment, the segment’s index, the FIFO entries taken and the segment’s rate, and software can read them (with the sample counter) as one snapshot. Because

```
φ_d(start of segment i+1) = φ_d(start of segment i) + L·(ν_base + ν_i)
```

holds exactly in integers, the software mirrors φ_d without error and checks the mirror against the snapshot. The 96-bit φ_d keeps 32 integer cycles: ±2^{31} cycles, ±214 s of time error, before it wraps (the output only uses φ mod 1).

PPS outputs

Two 1 PPS outputs, 3.3 V LVCMOS on the extension connector E1, pulses of `pps.width_us` (default 20 µs):

- **DIO2_P: reference PPS**, the board timebase (OUT2): a pulse every $125\cdot10^6$ samples, the first one second after the design starts.
- **DIO1_P: simulated clock’s PPS** (OUT1). The simulated clock has run 0.08·n + φ(n) cycles after n samples, so its second k ends when S(n) = n + 12.5·φ(n) (φ in cycles, S in samples) reaches $125\cdot10^6$·k. The hardware computes S every sample from the unwrapped φ (12.5·φ as 8φ + 4φ + φ/2, exact) and starts a pulse on the

first sample with S ≥ the next second, then advances that second by $125 \cdot 10^6$. A phase that moves back (noise) cannot produce a second pulse; a jump over several seconds produces one merged pulse and counts every second.

At φ = 0 both pulses rise on the same sample. The ideal edge shift is −φ/f0, but edges lie on the 8 ns grid: φ = +1 cycle requests a 100 ns advance and gives a **96 ns advance with a fixed +4 ns remainder**. That remainder repeats every second; it does not average away. For a constant phase the remainder is in [0, 8 ns); with phase jumps it can exceed one sample. The hardware captures the latest rising edge's unsigned remainder (whole samples saturated at $2^{32}-1$ plus a 2^{-64} fraction), and `pps_status()` reports it.

S(n) uses the φ of carrier sample n (the carrier adds the phase of the following clock, so the PPS takes the index one clock later). Both pulses are delayed with `DELAY = 9` (the simulated pulse uses DELAY − 1 for its later comparison), to align with the same sample's vendor DAC input register and SAME_EDGE ODDR frame. These digital latencies are verified; the analog delays of the outputs and pins are not measured.

Time of day

The reference PPS normally starts one second after the design starts. `PPS UTC` (`align_utc()`) places it on a UTC second instead, using the board's system clock, which `systemd-timesyncd` keeps on UTC via NTP (on this board: internet server, offset ≈ 0.3 ms, jitter ≈ 5 ms when checked). The driver reads the sample counter between two readings of the system clock (in integer nanoseconds), picks the UTC second U0 at least 1.5 s ahead, and arms both PPS on the sample `epoch = n + (U0 − t)·125 MHz`, rounded to a multiple of 25 samples (200 ns), where the carrier table restarts, so the reference pulse stays on OUT2's positive zero crossing; the hardware restarts both seconds at the epoch and resets the counts, which then count seconds since U0. So the reference pulse k marks UTC U0 + k − 1, and the simulated clock's pulses keep their offset −φ/f0 (12.5 samples per cycle). Accuracy at arming: the NTP offset (ms) plus the read time (µs).

After arming, the PPS follow the **board's crystal**, not UTC: a crystal error y makes them drift by y·t (20 ppm would be 1.7 s per day). The service's worker measures this every `pps.log_interval` (default 60 s) as the board time minus UTC, e = (n − epoch)·8 ns − (t − U0), logs it to `pps/<date>.csv` and fits a line to the last day: the slope is the board clock's frequency offset against UTC (NTP-limited: about 5 ms / span, i.e. $1.4 \cdot 10^{-6}$ over an hour, $6 \cdot 10^{-8}$ over a day). `PPS?` reports `board_error_s`, the drift fit and `simulated_error_s` = e + φ/f0, the simulated clock reading minus UTC. Positive clock error means earlier PPS edges: their edge-time error has the opposite sign, plus the grid remainder. Re-align with `PPS UTC` whenever wanted; with `pps.utc` on, the worker aligns by itself after every reload or reboot. A reload resets the hardware epoch, so an alignment is valid only for the session it was made in.

Phase meter

The ADC inputs feed an I/Q demodulator at exactly 10 MHz. For each input x and each window of N = 25·M samples (whole periods of the 2/25 carrier, `meter.window`, default 10 ms):

```
I = Σ x[n]·C[k],  Q = Σ x[n]·S[k],   k = n mod 25
C[k], S[k] = 131071·cos, sin(2π·2k/25) rounded (C[12] one higher, so that Σ C = Σ S = 0)
x = A·cos(2π·0.08·n + ψ)   →   I ≈ N·A·131071/2·cos ψ,   Q ≈ −N·A·131071/2·sin ψ
```

Because the window holds whole carrier periods and the tables sum to exactly zero, a DC offset cancels exactly. Cancellation of the 2f product is only approximate with the quantized tables. ψ is the input's phase relative to the internal 10 MHz (k counts from the design's start, like the carrier), plus the fixed delays of the analog input and ADC. A is converted to volts with the board's input calibration and configured LV/HV ranges. Windows follow each other without gaps; each pushes {index, I1, Q1, I2, Q2} (64-bit sums) into a 512-entry FIFO, which software drains. The index is 32 bits (it wraps after 2^{32} windows, 14 min at the shortest window); the driver extends it from the sample counter. The service's worker unwraps the phases window by window for every reader (recordings, the meter log, `MEAS STREAM`, `MEAS?`): IN1 against OUT1's predicted phase, dropping windows without signal or whose phase turned within ("Stability recordings" below). A script without the service unwraps against OUT1's frequency offset alone (`meter_advance`). Changing raw M restarts the windows but preserves old FIFO entries: issue METER CLEAR before interpreting data with the new window length/start. The service's `MEAS? n` averages n consecutive worker windows, also while recording or logging, under the common acquisition token. A second acquisition is BUSY, a superseding mutation cancels the first, and reload makes its handle STALE. A direct driver can use a cached single window; averaging multiple windows of an active recording requires the service.

Loopback. With OUT1 → IN1 and OUT2 → IN2, IN1 − IN2 is OUT1's phase relative to OUT2 plus a fixed difference (cables, the two output and input channels), which `scripts/test_hw.py` measures at φ = 0 and subtracts. The

board clock's own noise is common to both and cancels. Expected per-window resolution at 0.5 V with about 1 mV rms input noise: $\sigma \approx 1\ \text{mV} / (0.5\ \text{V} \cdot 2\pi \cdot \sqrt{(N/2)}) \approx 4 \cdot 10^{-7}$ cycle (40 fs) for 10 ms windows — an estimate; the one-day loopback floor measured $2.9 \cdot 10^{-7}$ cycle (29 fs) per 10 ms window. White PM above the window rate averages out; white FM, flicker and random-walk noise, frequency offsets and phase steps are measured directly.

Noise generators

Two Gaussian-like noise sources and a dither source run in the FPGA, from xoroshiro128+ generators seeded by the 64-bit `seed` (the same seed gives the same sequence after each commit that changes it, and at power-up). Each variate is the sum of 8 uniform values of the form 2r+1 (r a 16-bit integer), so its mean is exactly zero; its standard deviation is $\sqrt{8 \cdot (2^{32}-1)/3} = 107\ 019$ and its tails end at 4.9σ (excess kurtosis -0.15). It is scaled by a 32-bit gain and a power of two to units of 2^{-64} cycle, with a relative resolution better than 10^{-9}, rounded half away from zero (so the mean stays exactly zero) and saturated at $\pm(2^{63} - 1)$, including clamping the representable but asymmetric endpoint -2^{63}.

- **White PM (h2):** w_pm is added to the phase every sample. With $S_\varphi(f) = h2 \cdot f0^2$ (one-sided, rad²/Hz) up to f_s/2, the per-sample standard deviation is $\sigma = f0 \cdot \sqrt{(h2 \cdot f_s/2)}/(2\pi)$ cycles. Example: a floor of L(f) = −140 dBc/Hz is $h2 = 2 \cdot 10^{-28}$, $\sigma = 1.78 \cdot 10^{-4}$ cycle (17.8 ps). The floor extends to 62.5 MHz offsets before the DAC's hold and the analog output filter.
- **White FM (h0):** each sample adds a random increment of standard deviation $0.08 \cdot \sqrt{(h0 \cdot f_s/2)}$ cycles to φ_w, a random walk of phase. Allan deviation $\sigma_y(\tau) = \sqrt{(h0/(2\tau))}$; for example 10^{-11} at 1 s is $h0 = 2 \cdot 10^{-22}$.
- **Dither (`dither`, OUT1 only):** a triangular random value spanning ±1 DAC step (the sum of two odd, hence symmetric, 19-bit values: mean exactly zero) is added before the 14-bit rounding. The average output is then linear in φ below one DAC step (no fixed rounding pattern), at the cost of a white amplitude and phase floor, roughly −156 dBc/Hz at 0.5 V (a digital estimate; the board's analog noise is unmeasured and probably higher). It is off at power-up (the configuration then equals the FPGA's power-up registers); turn it on with `NOISE JSON {"dither": true}` whenever OUT1's phase moves.

The standard deviations are limited to 0.05 cycle per sample, far above any clock. `tests/tb_noise.sv` checks the moments, the absence of correlation and the exact scaling.

Phase-noise playback

The noise types that need memory, the deterministic parts and recorded data are generated by a background task in the board's control service (`control/atomic_clock/playback.py`), which keeps the segment FIFO 3.5 s full (at most its 4096 entries). For each segment boundary $t_n = n \cdot L/f_s$ it computes a target phase $p(t_n)$ from:

- **flicker FM (h−1)** and **flicker PM (h1):** a sum of first-order sections with corner frequencies spaced two per decade from 10^{-8} Hz (about three years) to f_seg/2, each driven by Gaussian noise and weighted so that the sum has the 1/f spectrum ($S_y = h{-}1/f$; for flicker PM $S_x = h1/((2\pi)^2 f)$), started in its stationary state. The sections are filtered in blocks of 256 segments with numpy/scipy (one filter call per section; a pure Python step per segment and section was too slow for 1000 segments/s on the board's CPU), driven by a PCG64 generator seeded with `seed`;
- **random-walk FM (h−2):** Gaussian frequency steps of $2\pi \cdot \sqrt{(h{-}2 \cdot T/2)}$ per segment of length T;
- **drift:** y(t) = D·t/86 400 s for `drift_per_day` D, i.e. phase $f0 \cdot D \cdot t^2/(2 \cdot 86\ 400\ \text{s})$;
- **sinusoidal PM:** amplitude (cycles)·sin(2πf t), f at most f_seg/10;
- **replay:** a recorded time error x(t) (seconds, two-column CSV in `/home/xilinx/rpfpga/atomic_clock/replay/<name>.csv`) interpolated linearly, times f0 and `replay_scale`; looped with the end-to-start difference carried over, so the phase stays continuous, or held after its end.

The worker then picks $\nu_n = \text{round}((B_{n+1} + p(t_{n+1}) - P_n)/L) - R$, where P_n is its exact mirror of φ_d at the start of segment n, R the frequency offset's rate and $B_n = B + n \cdot L \cdot R$ the baseline, which moves with the offset (so ν carries the trajectory alone and the offset keeps acting during playback). The realized phase at every boundary is within L/2 LSB ($3.4 \cdot 10^{-15}$ cycle at 1 kHz) of the target and the error never accumulates. B is chosen when playback starts so that the trajectory continues from the present phase, and re-derived from a snapshot on a resynchronization or a change of the offset: starting, stopping and resynchronizing never make the output jump.

Software-generated content only reaches f_seg/2 (500 Hz by default) in offset frequency; above that, flicker PM is missing and the hardware's white PM and white FM take over, which matches real clocks, whose phase noise at high offsets is dominated by white PM. `tests/test_playback.py` checks the Allan deviation of each software noise type against the formulas below, within 10 % from 10 ms to 100 ms over a 60 s record (within 5 % for the seeds tried).

Allan deviation of the power-law model (standard asymptotic forms, f_h the white/flicker PM bandwidth, used by the GUI preview and `noise.adev`):

Type	$\sigma_y^2(\tau)$
white PM, h2	$3 \cdot f_h \cdot h2 / ((2\pi)^2\tau^2)$
flicker PM, h1	$h1 \cdot (1.038 + 3 \ln(2\pi f_h \tau)) / ((2\pi)^2\tau^2)$
white FM, h0	$h0 / (2\tau)$
flicker FM, h−1	$2 \ln 2 \cdot h{-}1$
random-walk FM, h−2	$(2\pi)^2 \cdot \tau \cdot h{-}2 / 6$

Robustness. The worker refills about 20 times per second, at most 0.25 s of computing per step, pushing 128 entries at a time so that the FIFO is fed while it computes; a (re)start queues 1 s of zero-rate warm-up segments first. If the FIFO ran empty (for example while the service restarts) it restarts the sequence from the present phase and logs a resynchronization; the held rate kept the phase continuous in the meantime. A gap of more than 1 s between worker steps is logged (`playback: worker stalled ...`, with the time each part of the previous step took). SD-card writes can block for a second or two: the checkpoint and the recordings' CSV files are written by background threads, and the 3.5 s of FIFO content rides out the rest. On the board the service then uses about 39 % of one CPU core with model playback, 67 % with a meter stream and about 50 % with a recording at 100 windows/s (measured before the predicted-phase unwrapping). At least every 10 s (`VERIFY_SECONDS`) it reads the snapshot and checks that the hardware's φ_d at a segment start equals its mirror exactly (any difference triggers a logged resynchronization); every `log_interval` (default 10 s) it appends the segment's φ_d and target to `/home/xilinx/rpfpga/atomic_clock/playback/<date>.csv`. Every minute it saves the model state (random generator, filter states, model time) to `checkpoint.json` in the same folder; after a reboot it resumes the same model from there. A change of the noise or playback settings (or a replay record uploaded again) starts a new model; a change of the frequency offset needs no restart: the queued rates carry the trajectory alone, so the worker re-derives its mirror from a snapshot and playback goes on without a jump or a pause.

What is simulated relative to what. All of this is relative to the board's 125 MHz oscillator, whose own noise and drift are not measured here. OUT2 shares that oscillator: compare OUT1 against OUT2 to measure the programmed phase alone.

Stability recordings and their analysis

All quantities follow the definitions of IEEE Std 1139-2022 (the testing manual summarizes it and lists the canonical references).

A recording (`control/atomic_clock/recording.py`, run by the same worker) drains every meter window: a fast series of every window for the first hour and a slow series of one window per second for the whole run (see PROTOCOL.md). Each sample is x averaged over one window, so the series see the noise through the window filter $|H(f)|^2 = \mathrm{sinc}^2(f \cdot \tau_w)$. IN1's phase is unwrapped window by window against OUT1's phase as the worker predicts it (the frequency offset, the played trajectory from its exact mirror, the slew toward a set phase). The meter itself limits the motion: the I/Q sum over a window of a phase that moves r cycles within it shrinks by $|\mathrm{sinc}(r)|$, so only windows moving less than about a quarter cycle keep their amplitude (more than 90 %); the others are dropped. A new run starts (the row is flagged) where the phase is not continuous: lost windows (the 512-entry FIFO overflowing, about 5 s at 10 ms), a change of OUT1's frequency offset, phase, slew or alignment, or of the window, windows without signal (below 0.05 V at either input) or whose amplitude fell below 90 % of the run's (not written), and windows more than a quarter cycle from their prediction; the analysis uses the longest gap-free segment. Fast and slow series retain discontinuity flags independently, so a gap between slow samples still flags the next slow row. The slow interval must be an integer multiple of the quantized window. Acquisition duration follows the board sample counter even when all windows lack signal. A window change ends the recording without restoring over the new setting; a write error persists in header/status and gives a failed stop reason.

The host tools (`tools/clocklib` on the common `common/tools/stabilitylib`; numpy, scipy, matplotlib, allantools) compute the overlapping ADEV, MDEV, TDEV and Hadamard deviation with 68 % confidence intervals (allantools' noise identification and Greenhall's equivalent degrees of freedom; where noise identification needs more points, a χ^2 interval with one degree of freedom per non-overlapping τ interval). The expected curves come from the spectral integrals with the meter's window and each series' spacing τ0:

$$\mathrm{AVAR}(\tau) = 8/\tau^2 \quad \int S_x |H|^2 \sin^4(\pi f\tau)\, df$$
$$\mathrm{MVAR}(\tau) = 8/\tau^2 \quad \int S_x |H|^2 \sin^6(\pi f\tau) / (m \sin(\pi f\tau_0))^2\, df \qquad (\tau = m\,\tau_0)$$
$$\mathrm{HVAR}(\tau) = 32/(3\tau^2) \int S_x |H|^2 \sin^6(\pi f\tau)\, df, \quad \mathrm{TVAR} = \tau^2/3 \cdot \mathrm{MVAR}$$

up to 62.5 MHz for the FPGA's white FM and to f_seg/2 for played-back terms. The FPGA's white PM is taken as independent window errors of σ_x = √(1.5·h2/(2τ_w))/(2π) (AVAR = 3σ_x²/τ², MVAR = 3σ_x²/(mτ²)). The factor 1.5: the demodulator's product x·e^(−iθ) contains, besides the wanted ½·iφ, the term −½·iφ·e^(−2iθ); for phase noise that is white up to the Nyquist frequency the second term averages down no faster than the first and adds half its variance (a simulation of the demodulator gives 1.54; T1 on the board, 1.52). Played-back noise (below 500 Hz) is not affected; drift adds D²τ²/2 to AVAR and MVAR and nothing to HVAR; a sinusoidal PM of amplitude A (s) is a line S_x = A²/2 at f_m. The model reproduces the standard asymptotes (white, flicker and random-walk FM within 3 %) and simulated noise (MDEV/ADEV and HDEV/ADEV ratios).

The fit is a weighted non-negative least-squares fit of AVAR and MVAR of both series to Σ h_α A_α(τ) + D²τ²/2 + floor (a measured floor recording, interpolated in log-log), with the drift D from a quadratic fit to x(t) of the slow series; the weights come from the confidence intervals, and the parameter uncertainties are widened by a factor 3 for the correlation between neighbouring τ points (checked on synthetic recordings: terms are then judged only where resolved). On synthetic recordings with the meter model's noise, the fit recovers each single term within 10 % (white PM, flicker PM, white FM, flicker FM, random-walk FM) and the drift within a few percent. A configured sine (e.g. a daily temperature cycle) is fitted together with the drift polynomial, and its amplitude and the drift are judged only where the run covers at least two periods and the noise at the sine's frequency (σ_A = √(2 S_x(f)/T)) allows 30 %. For a replay run the known term uses the actual replay scale, linear interpolation, and each series' sample spacing, window and analyzed segment. Its cache includes file identity and sampling; unavailable τ remain unavailable. Generated replay records are reused only when duration, parameters and algorithm identity match.

Configured-noise checks without a resolved noise coefficient or at least three useful above-floor ADEV points are inconclusive. A phase step across a broken unwrapping branch is not identifiable and is also inconclusive. The runner rejects write errors, invalid cadence or row counts, and acquisition coverage below 95 %. TDEV masks use only τ for which the actual analyzed segment spans at least 12τ, rather than counting time across gaps.

For external point-sampled data, a finite white-PM bandwidth uses the sampled covariance R(k) = h2·f_h·sinc(2f_h kτ0)/(2π)², retaining correlations when samples are not independent. Logarithmic aging removes the full fitted aging trajectory before the noise fit; its final frequency-drift rate is reported separately. The reports add Totdev and ThéoBR to long τ (picking every kth phase sample for long records, preserving white PM), MTIE and TIE rms with ITU-T masks, the dynamic ADEV, the three-cornered hat of two external clocks and the board's clock, and the same analysis of other instruments' files (testing manual, "Further estimators and tools"). The tests and their expected plots are in the testing manual (manual/testing_manual.pdf).

Control reference

atomic_clock is controlled in the same ways as every design in this repository:

- the **driver** `AtomicClock` (Python on the board, e.g. in Jupyter);
- the **client** `AtomicClockClient` (Python on a PC), which talks to the board's control service and shares its high-level controls;
- the **TCP protocol** of that service (port 5025, one text command per line);
- the **command line** `python -m rpfpga.cli`.

The parts common to all designs are described in `../../common/README.md` (the uniform driver API, the TCP service and its core commands, the command line). This page gives what a user of atomic_clock needs, and everything it adds. Amplitudes and offsets are volts at the connectors, converted with the board calibration (`../../common/CALIBRATION.md`); phases are in cycles of the 10 MHz carrier (1 cycle = 100 ns).

Configuration

```
{"outputs": {"1": {"amplitude": 0.5, "offset": 0.0, "enabled": True, "phase": 0.0,
                   "phase_slew": 1e-6, "frequency_offset": 0.0},
             "2": {"amplitude": 0.5, "offset": 0.0, "enabled": True}},
 "noise": {"h2": 0.0, "h1": 0.0, "h0": 0.0, "h_1": 0.0, "h_2": 0.0, "drift_per_day": 0.0,
           "pm_sine_amplitude": 0.0, "pm_sine_frequency": 0.0, "seed": 0, "dither": False},
 "playback": {"source": "off", "replay": "", "replay_loop": False, "replay_scale": 1.0,
              "segment_rate": 1000.0, "log_interval": 10.0},
 "pps": {"width_us": 20.0, "utc": False, "log_interval": 60.0},
 "meter": {"window": 0.01, "log": False, "log_interval": 10.0}}
```

```
OUT1 = amplitude1 · sin(2π·(10 MHz · t + φ(t))) + offset1      (sine part only if enabled1)
OUT2 = amplitude2 · sin(2π·10 MHz · t) + offset2                (the reference)
```

`φ(t) = phase (reached at phase_slew) + frequency_offset·10 MHz·t + noise and playback`

`apply(patch)` merges a partial configuration, e.g. `clock.apply({"outputs": {"1": {"phase": 0.25}}})`; `apply(config, replace=True)` starts from the default above. All hardware settings are committed on **one clock edge**.

Outputs

- **amplitude** (V, peak, ≥ 0): steps of one stream code, G·(1 − 2^{-17}) ≈ 35 µV on this board (30.5 µV nominal).
- **offset** (V): placed on the DAC grid, steps of 4·G ≈ 138 µV on this board, so that the DC level is the same with the sine on or off.
- **enabled** (true/false): with false the output is the offset alone.
- **phase** (cycles, OUT1 only): the target phase, **unwrapped**: whole cycles count (±2^{31} cycles = ±214 s; 1 cycle = 100 ns), because they move the simulated clock's PPS. Positive values make OUT1 **lead** OUT2. 96-bit word (2^{-64} cycle steps), of which the sine uses the fraction's top 28 bits (0.37 fs); the exact word is `state()[1]["phase_word"]` (signed int). The output moves to a new phase smoothly at most at `phase_slew` (from +0.4 to −0.4 is a move of 0.8 cycle). Setting `phase` (and loading a preset, `reset()`, `safe()`) puts OUT1 at exactly that phase relative to OUT2, discarding the phase accumulated so far by the frequency offset, noise and playback; these continue from there. Changing only those keeps the accumulated phase. In particular `PHASE 0` preserves `FREQ`: the phase ramp continues after the realignment.
- **phase_slew** (fractional frequency, 0 … 10^{-3}, default 10^{-6}): the largest frequency deviation a phase move may cause. A move of Δφ cycles takes Δφ/(phase_slew·10 MHz): 25 ms for a quarter cycle at the default. 0 makes phase changes steps (the 1.0 behavior).
- **frequency_offset** (fractional, |y| ≤ 10^{-3}): OUT1's frequency is 10 MHz·(1 + y) relative to the board clock, integrated exactly (resolution 6.8·10^{-19}, no drift over any time).

offset ± amplitude must stay within the DAC range, about −1.143 … +1.128 V on this board's OUT1 (computed from the calibration); otherwise the request is refused with the exact range, and nothing is written.

Noise (power-law model $S_y(f) = h_2 f^2 + h_1 f + h_0 + h_{-1}/f + h_{-2}/f^2$, one-sided, THEORY.md):

- **h2** (white PM) and **h0** (white FM) are generated in the FPGA at every sample, at most 0.05 cycle rms per sample each (h2 ≤ 1.6·10^{-23}, h0 ≤ 6.2·10^{-9}). Applied values are reported after the gain quantization (relative error below 10^{-8}). h2 for a floor of L dBc/Hz: `noise.h2_from_dbc(L)`.
- **h1** (flicker PM), **h_1** (flicker FM), **h_2** (random-walk FM), **drift_per_day** (linear frequency drift, fractional per day, |D| ≤ 10^{-6}) and **pm_sine_amplitude** (cycles) / **pm_sine_frequency** (Hz, at most a tenth of the segment rate) take effect through playback (`source` "model").
- **seed**: integer 0 … 2^{64}−1, for the hardware generators and the playback model; the same seed reproduces the same noise.
- **dither**: OUT1's triangular dither before the 14-bit rounding (linear sub-step phase; adds a white floor).

`atomic_clock.noise` converts between conventions: `h_from_adev(kind, sigma, tau)` (e.g. `h_from_adev("h_1", 1e-13, 1)` for a flicker floor of 10^{-13}), `adev(h, tau)`, `phase_psd_dbc(h, f)`, `h2_from_dbc(level)`.

Playback (runs in the board's control service; see "Playback" below):

- **source**: "off", "model" (the noise model's h1, h_1, h_2, drift and sinusoidal PM) or "replay" (a recorded time error).
- **replay**: the record's name, a file `/home/xilinx/rpfpga/atomic_clock/replay/<name>.csv` with two columns, time (s) and time error (s), increasing times, # comments; **replay_loop** repeats it (continuously), otherwise the last value is held; **replay_scale** multiplies it; the record's largest slope times replay_scale must stay within 10^{-3} (fractional frequency).
- **segment_rate** (1 … 10 000 per second, default 1000): the trajectory's sample rate; software noise reaches half of it in offset frequency.
- **log_interval** (s, 0 = no log): how often the worker logs the programmed phase (default 10 s).

Values out of range raise `ValueError` (TCP: `ERR ...`) and nothing is written. Getters return the **applied** values after rounding. A phase given as a floating-point number has 53 significant bits; reapplying the reported value can move the word by up to 2^{-54} cycle, far below what the sine uses.

After loading (power-up, `LOAD`, or the driver downloading the design), both outputs carry the 10 MHz sine at amplitude 16384 codes and offset 0 in raw converter units, i.e. without calibration: about 0.57 V amplitude on this board (computed, not measured), in phase, no noise, playback off. `config()` reports the calibrated interpretation of those raw words. Apply any configuration (e.g. `load_preset("reference")`) for calibrated values. After a reboot, the service restores the last configuration by itself, including playback, which resumes the saved model.

PPS — `width_us`: pulse width of both PPS outputs (0.008 … 999 999 µs, default 20 µs); `utc`: the service's worker aligns both PPS to UTC by itself after every reload or reboot (`align_utc()` / `PPS UTC` does it on request); `log_interval`: seconds between the worker's measurements of the board clock against UTC (`/home/xilinx/rpfpga/atomic_clock/pps/<UTC date>.csv`).

Meter — `window`: seconds per meter window, rounded to a whole number of 200 ns (25 samples), 200 ns … 1 s, default 10 ms; `log`: the service's worker drains the meter, logs the means every `log_interval` seconds to `/home/xilinx/rpfpga/atomic_clock/meter/<UTC date>.csv` (unwrapped phases), and keeps a summary for direct-driver readback. The service's `MEAS? [n]` averages n consecutive worker windows of one run, including during meter logging, recordings and open streams. A direct driver may return a cached single-window reading; multiwindow averaging during a recording requires the service. Direct FIFO acquisition (`meter_read()`) is refused while a recording owns the meter, and, within the service's own process, while the worker drains for a log or stream. A driver in another process (a notebook on the board) cannot see the service's streams: do not read the FIFO directly while the service logs or streams the meter. Without the service, fresh measurements unwrap against the frequency offset alone. Input calibration and the configured LV/HV ranges determine amplitudes in volts. The meter measures a phase that moves less than about a quarter cycle per window: faster, the I/Q sum of the turning phasor loses its amplitude (|sinc| of the cycles moved) and the worker drops the window; use a shorter window for faster motion.

Measurements use the shared acquisition token and release the hardware lock while waiting. A second acquisition gets `BUSY`; a superseding configuration, calibration or input-range change cancels the pending measurement (`CANCELLED`), and a reload makes it `STALE`. A gap restarts the service's collection of n consecutive windows. Use a separate client connection for control while a measurement waits, as in the common concurrency reference.

Playback

The board's control service (`rp-overlay`) runs the playback worker whenever atomic_clock is the active design; Jupyter and command-line drivers only change the settings. With `source` "model" or "replay" the worker keeps the hardware's segment FIFO 3.5 s full, mirrors the hardware's phase exactly, and:

- starts (and restarts after a model change or a FIFO underflow) from the present phase, so the output never jumps; a frequency-offset change goes on without a restart (the mirror is re-derived);
- logs the programmed phase every `log_interval` and every event (start, stop, resynchronization, errors) to `/home/xilinx/rpfpga/atomic_clock/playback/<UTC date>.csv`: columns `utc, samples, segment, model_time_s, phase_d, target, baseline, event`, phases as integers in 2^{-64} cycle (`target` = `baseline` + the trajectory's value, the baseline of that segment; `phase_d` − `target` is the realized minus programmed phase, at most $3.4 \cdot 10^{-15}$ cycle at 1 kHz);
- saves its model every minute to `checkpoint.json` in that folder and resumes it after a reboot (logged as "resume at model time …").

Starting the same model or replay again (same noise and playback settings) continues where it stopped, also after a reboot (the checkpoint); changing any of those settings, for example the seed, starts it from t = 0. In both cases the output continues from the present phase without a jump.

Resetting the phase. `RESET` (`reset()`) or `PRESET reference` turns playback and noise off, sets the frequency offset to 0 and puts OUT1 exactly in phase with OUT2, discarding the phase accumulated so far (whole cycles included); it moves there at the default slew (10^{-6}), which it sets itself: `SLEW 0` sent right after it finishes the move at once (`SLEW 1e-6` restores the default). To keep the other settings: `PLAY OFF`, `NOISE JSON {"h2": 0, "h0": 0}`, `FREQ 0`, `PHASE 0`. Applying playback off stops and clears the hardware queue in the same commit as the settings. RESET therefore stops queued rates before the phase realignment, independently of the worker's next wake-up.

`playback_status()` (TCP `PLAY?`) reports the source, whether segments are playing, the FIFO level, the underflow flag, the latest segment, φ_d in cycles, the sample count and the last logged event. Without the service (for example a driver in Jupyter with the service stopped) nothing feeds the FIFO: it runs empty and holds the last rate.

Stability recordings

For long-term stability (Allan deviation over hours or days) the service's worker records the phase meter on the board; the host tools in `atomic_clock/tools/` download and analyze the records (the testing manual, `manual/testing_manual.pdf`). One recording at a time:

- the meter window is set to `fast_window_s` (default 10 ms) and every window is drained: `fast.csv` keeps every window for the first `fast_duration_s` (default 3600 s), `slow.csv` the first window of every `slow_`

interval_s (default 1 s) for the whole run, plus the Zynq die temperature. The fast window is quantized to 200 ns; the slow interval must be an integer multiple of that quantized window;
- columns t_s (from the sample counter, since the first window), in1_phase, in2_phase (unwrapped cycles of 10 MHz), difference (IN1 − IN2, cycles), in1_amp, in2_amp (V), flags (value 1: the phase is not continuous with the previous row: windows were lost, OUT1's frequency offset, phase, slew or alignment changed, there was no signal (windows below 0.05 V at either input are not written), or the phase moved other than predicted; value 2: the playback resynchronized or started, stopped or changed its trajectory), temp_c (slow series); header.json holds the options, the start and stop, the complete configuration, the versions, the gap count, the stop reason and any write_error. Fast and slow flags are retained independently: discontinuities between two slow rows flag the next slow row even if already flagged in fast data, and a slow row that follows the previous one by other than slow_interval_s (recovery partway through an interval) has value 1 too;
- it stops after duration_h (0: until REC STOP), on SAFE, or when the service restarts (marked "service restart"); a closed connection does not stop it. Duration is board-clock acquisition time, including windows rejected for missing signal; elapsed_s therefore advances even with no usable rows;
- changing the meter window ends the recording ("meter window changed"), preserving the new setting. Normal completion restores the previous meter settings only if they have not been superseded. A write error, including one found on final flush, persists in both header.json and status and sets stop_reason to "write failed: …";
- while it runs the meter belongs to it: MEAS? [n] averages n of its windows and meter_read() is refused; other settings may be changed and flag the next rows;
- a start is refused while another recording runs, if the name exists, or with less than 1 GB free on the SD card. A day takes about 40 MB, the fast hour about 34 MB.

Recordings are kept in /home/xilinx/rpfpga/atomic_clock/recordings/<name>/ until deleted. REC GET sends a file in chunks of 256 kB, zlib-compressed and base64-encoded; download_recording() assembles them.

Meter stream

MEAS STREAM [interval_s] pushes every meter window to a client as it is measured, for live analysis on a PC (tools/rustle). It is sent on a connection of its own, as the first line, not inside REQUEST. The reply is one header line, then one line per interval (default 0.2 s; 0.05 to 5 s). Sending STOP, or closing the connection, ends the stream.

- **Header:** OK {"fields": [...], "format": ..., "f0_hz": 1e7, "interval_s": 0.2, "window_s": 0.01, "queue_rows": 6000, "session": ..., "utc": ...}.
- **Data:** DATA {"rows": n, "lost": k, "data": "<base64>"}, with k the windows dropped for this client since the stream started (cumulative). The data are n rows of 8 little-endian float64. An empty batch also comes every interval, as a heartbeat.
- **Rows:** index (window number), t_s (window start, s on the sample counter), in1_phase, in2_phase (unwrapped, cycles of 10 MHz), in1_amp, in2_amp (V peak), window_s and flags.
- **Flags:** flags = 1 means windows were lost before this row, or the unwrapping restarted: the first row; a recording started or stopped; the meter log was switched; the window, or OUT1's frequency offset, phase, slew or alignment changed; the playback resynchronized, started, stopped or changed its trajectory; windows without signal (below 0.05 V at either input: not sent) came before; or the window was more than a quarter cycle from OUT1's predicted phase. The phase is continuous only within a run of unflagged rows. IN1 is unwrapped against OUT1's predicted phase (frequency offset, played trajectory, slew). Windows in which the phase moved more than about a quarter cycle (a fast slew, a phase step, PM or an offset too fast for the window) lose their amplitude and are not sent.
- **Who drains the meter:** the service's worker drains it for a running recording, for the meter log, or only for the stream. A stream therefore runs alongside a recording and gets the same windows.
- **A slow client:** each client has a queue of about a minute of windows (6000). One that falls further behind loses the oldest windows; lost counts them, and the first row kept after them is flagged.
- **End:** a reload of the overlay (LOAD) ends the stream with ERR STALE …; reconnect.
- **Board CPU:** measured on the board with model playback running, after the unwrapping against OUT1's predicted phase was added (2026-10-12).

Running	Service CPU
playback alone	38.7 %

Running	Service CPU
with a stream	66.8 %

No windows were lost. With a recording (not remeasured since): 50 % alone, 54 % with a stream, before the prediction.

From Python: `MeterStream` (`from atomic_clock import MeterStream`):

```
with MeterStream("192.168.0.149") as s:
    rows, lost = s.read()       # every interval: [(index, t_s, in1, in2, amp1, amp2, window_s,
 ↪  flags)]
```

Presets

Name	Function
`reference`	both outputs 10 MHz, 0.5 V amplitude, 0 V offset, in phase, no noise, playback off (the default configuration)
`quadrature`	as `reference`, OUT1 leading OUT2 by a quarter cycle (25 ns)
`off`	both sines off, both outputs at 0 V

`presets()` returns `{"builtin": {name: description}, "user": [names]}`. `save_preset(name)` stores the current configuration on the board (in `/home/xilinx/rpfpga/atomic_clock/presets/`), `load_preset(name)` applies a built-in or saved preset, and `delete_preset(name)` removes a saved one. `safe()` turns both sines off, puts both outputs at the calibrated 0 V, turns the noise and playback off and empties the segment FIFO.

Python methods

On the driver `AtomicClock` (board) and the client `AtomicClockClient` (PC):

Method	Description
`set(n, amplitude=None, offset=None, enabled=None, phase=None, phase_slew=None, frequency_offset=None)`	patch output n (1 or 2; the last three only for 1) in one commit; returns its applied values
`set_amplitude(n, volts)`, `set_offset(n, volts)`, `enable(n, on=True)`	set one value, keeping the others; return `get(n)`
`set_phase(cycles)`	OUT1 phase relative to OUT2 (reached at `phase_slew`); returns `get(1)`
`set_frequency_offset(y)`	OUT1 fractional frequency offset; returns `get(1)`
`set_noise(**settings)`	patch the noise settings; returns them as applied
`set_playback(**settings)`	patch the playback settings (the service's worker follows); returns them
`playback_status()`	source, playing, FIFO level, underflow, latest segment, φ_d (cycles), samples, last logged event
`snapshot_phase()`	coherent phase/count/PPS snapshot as exact integers; waits for a pending alignment, with public phases modulo 2^{96} (REGISTERS.md)
`measure(windows=1)`	phase meter: `{"in1"/"in2": {"amplitude" (V), "phase" (cycles)}, "difference" (IN1 − IN2, cycles), "t", "window_s", "windows"}`; service: average of 1 … 1000 consecutive worker windows, including REC/logging; direct driver: cached single window or fresh FIFO acquisition as described above
`meter_read(windows)`	`windows` fresh windows, each as above with its time `t` (driver; on a PC use `measure`)
`meter_status()`	the latest summary written by the service's worker while it logs the meter (`meter.log`), or None (driver)
`set_meter(**settings)`, `set_pps(width_us=None, utc=None, log_interval=None)`	meter and PPS settings

Method	Description
`pps_status()`	PPS counts, the latest simulated-clock edge (`last_edge_late_s`, normally below 8 ns, possibly larger on jumps), `time_offset_s`; when aligned to UTC also `utc0`, `utc_of_last_reference_pulse`, `board_error_s` (board − UTC), `simulated_error_s` (clock reading − UTC), `board_drift` (fitted frequency offset)
`start_recording(name, duration_h=0, fast_window_s=0.01, fast_duration_s=3600, slow_interval_s=1, note="")`	start a stability recording (above); returns its status. Ranges: `duration_h` ≤ 8784 (a year), `fast_window_s` 200 ns … 1 s, `fast_duration_s` ≤ 7 days, `slow_interval_s` from `fast_window_s` to 3600 s and an integer multiple of the quantized window
`stop_recording()`, `recording_status()`	request a stop; read its state (`requested`, `running`, `stopped`), board-clock elapsed time, rows, gaps, latest point, stop reason, `write_error`, free disk; inspect the stopped status for write failure
`list_recordings()`, `delete_recording(name)`	the recordings on the board (`name`, `start_utc`, `duration_s`, `stop_reason`, `note`, `bytes`); delete one
`download_recording(name, folder, progress=None)`	(client) copy `header.json`, `fast.csv`, `slow.csv` into `folder/name/`
`upload_replay(name, path, progress=None)`	(client) upload a replay record (two-column CSV, at most 50 MB) for `PLAY REPLAY <name>`; returns `list_replays()`
`list_replays()`, `delete_replay(name)`	the replay records on the board (`name`, `bytes`); delete one (refused while it plays)
`align_utc()`	arm both PPS on the next UTC second at least 1.5 s ahead; returns `{"utc0", "utc0_iso", "epoch", "uncertainty_s", ...}`
`get(n)`	applied values of output n: `{"amplitude", "offset", "enabled", "word"}`, for OUT1 also `"phase"`, `"phase_word"` (int), `"phase_slew"`, `"frequency_offset"`
`state()`	`{1: get(1), 2: get(2)}` from one register read
`sample_count()`	samples (8 ns each) since the core left reset (64 bit)
`reset()`	the default configuration (calibrated); returns the resulting state
`config()`, `apply(patch, replace=False, expected_revision=None)`	the applied configuration; apply a patch, optionally checking the revision of a draft
`snapshot()`, `apply_snapshot(...)`	atomic config/revision/session/description/status/calibration; or apply and return it
`describe()`	one line per output, e.g. `{1: 'OUT1 = 10 MHz sine, 0.5 V amplitude, phase +0.25 cycle (+25 ns)', 2: 'OUT2 = 10 MHz sine, 0.5 V amplitude (reference)'}`
`status()`	design name, ID, hardware/driver/rpfpga versions, build ID and date (`rpfpga.version_text(clock)` prints them on one line), register words (hex), `phase_word` (decimal string), `frequency_hz`, `rate_word`, `segment_samples`, the white PM/FM standard deviations per sample, calibration method
`safe()`	both sines off, outputs at 0 V, noise and playback off
`presets()`, `load_preset(name)`, `save_preset(name)`, `delete_preset(name)`	presets (above)
`calibration()`, `reset_calibration()`	the board calibration in use; back to nominal constants
`load()`, `reconnect()`	reprogram (power-up sine, uncalibrated), invalidating all handles including the caller; obtain a new handle with reconnect

The client also has `idn()`, `design()`, `designs()` and `load(design)`. atomic_clock cannot run the calibration's measuring steps (its meter measures 10 MHz phase and amplitude, not DC); measure the calibration with mult_add. The input range settings (`ranges()`, `set_input_range`) select the calibrated LV/HV scale of meter amplitudes and must match the physical jumpers. Concurrent control, revisions and reload semantics are those of every design (common reference).

```python
# on the board (Jupyter, or a script run with sudo /usr/local/share/pynq-venv/bin/python)
import rpfpga                                   # must come first
from pynq.overlays.atomic_clock import DRIVER
clock = DRIVER()                                # attaches, or loads atomic_clock if needed
clock.set(1, amplitude=0.3, phase=0.1)
print(clock.state())

# on a PC, from the repository root
import sys; sys.path[:0] = ["common/python", "atomic_clock/control"]
from atomic_clock import AtomicClockClient
with AtomicClockClient("192.168.0.149") as clock:
    clock.set_phase(-1e-6)                      # OUT1 lags by 1e-6 cycle = 100 fs
    clock.set_noise(h_1=1e-26, h0=2e-24, seed=1) # flicker FM + white FM
    clock.set_playback(source="model")          # the service plays the trajectory
    print(clock.describe(), clock.playback_status())
```

TCP commands

Connect with `nc 192.168.0.149 5025` and send one command per line; keywords are case-insensitive. Each reply is one line, `OK <value>` or `ERR <message>`.

atomic_clock commands (n = 1 or 2):

Command	Reply
`AMP<n> <volts>`, `AMP<n>?`	`OK <applied amplitude>`
`OFFS<n> <volts>`, `OFFS<n>?`	`OK <applied offset>`
`OUT<n> ON|OFF`, `OUT<n>?`	sine on or off; `OK ON` / `OK OFF`
`PHASE <cycles>`, `PHASE?`	OUT1 phase; `OK <applied phase>` with 17 significant digits
`SLEW <y>`, `SLEW?`	OUT1 phase slew (0: steps)
`FREQ <y>`, `FREQ?`	OUT1 fractional frequency offset
`NOISE JSON <patch>`, `NOISE?`	noise settings, e.g. `NOISE JSON {"h_1":1e-24, "seed":3}`; `OK {...}` as applied
`PLAY MODEL`, `PLAY OFF`, `PLAY REPLAY <name>`, `PLAY JSON <patch>`	playback settings; `OK {...}` as applied
`PLAY?`	playback status (JSON, as `playback_status()`)
`SNAP?`	the phase path's snapshot (JSON, exact integers)
`MEAS? [n]`	phase meter, averaged over n windows, 1 ... 1000 (JSON as `measure()`)
`MEAS WINDOW <s>`, `MEAS LOG ON|OFF`, `MEAS JSON <patch>`	meter settings; `OK {...}` as applied
`MEAS STREAM [interval_s]`	every meter window, pushed: a header line, then `DATA {...}` lines until `STOP` (own connection; see "Meter stream")
`PPS?`, `PPS WIDTH <µs>`	PPS status (JSON, as `pps_status()`); pulse width
`PPS UTC`	align both PPS to UTC (JSON record)
`PPS JSON <patch>`	PPS settings (`width_us`, `utc`, `log_interval`)
`REC START <name> [JSON <options>]`	start a stability recording, e.g. `REC START floor1 JSON {"duration_h": 2}`; `OK {...}` status
`REC STOP`, `REC?`	stop it; its status (JSON)
`REC LIST`, `REC DELETE <name>`	the recordings (JSON list); delete one
`REC GET <name> <file> <offset>`	a chunk of `header.json`, `fast.csv` or `slow.csv`: `{"data", "offset", "next", "size", "eof"}`
`REPLAY PUT <name> <offset> <chunk>`	append a chunk (zlib-compressed, base64-encoded) to the replay record being uploaded; offset 0 starts it; `OK {"name", "size"}`
`REPLAY DONE <name> <size>`	check the size and publish it as `replay/<name>.csv`; `OK [...]` the records
`REPLAY LIST`, `REPLAY DELETE <name>`	the replay records (JSON list); delete one
`SET<n> JSON <patch>`	several settings in one commit, e.g. `SET1 JSON {"phase":0.25,"amplitude":0.3}`; `OK {...}` with the applied values
`STATE?`	`OK {"1": {...}, "2": {...}}` (JSON, as `state()`)
`COUNT?`	`OK <samples since reset>`
`RESET` / `RESET?`	the default configuration; `OK` / `OK <state JSON>`

The core commands (`*IDN?`, `HELP`, `LOAD`, `CONFIG?`, `DESCRIBE?`, `STATUS?`, `PRESET ...`, `SAFE`, `CAL?`, ...) are listed in `../../common/README.md`. The service has no authentication: keep the board on a trusted network.

Command line

```
# from a PC, through the service (acts on the running design):
PYTHONPATH=common/python:atomic_clock/control python3 -m rpfpga.cli --host 192.168.0.149 --phase
 ↪  0.25 --amplitude2 0.2
# on the board itself (e.g. over SSH):
sudo /usr/local/share/pynq-venv/bin/python -m rpfpga.cli atomic_clock --preset quadrature
```

atomic_clock options, applied after the common ones (`--load`, `--reset`, `--preset`, ...; see the common README) and before `--safe`:

Option	Action
`--amplitude1 V`, `--amplitude2 V`	sine amplitude
`--offset1 V`, `--offset2 V`	DC offset
`--sine1 on|off`, `--sine2 on|off`	sine on or off
`--phase CYCLES`	OUT1 phase relative to OUT2
`--slew Y`, `--frequency-offset Y`	OUT1 phase slew, frequency offset
`--noise JSON`	noise settings, e.g. `--noise '{"h_2": 1e-26}'`
`--play model|off|replay:NAME`	playback source
`--pps-width US`, `--meter-window S`	PPS pulse width, meter window
`--pps-utc`	align both PPS to the next UTC second
`--measure N`	also print a phase measurement averaged over N windows

Each call prints one line per output and the applied values, e.g.

```
OUT1 = 10 MHz sine, 0.5 V amplitude, phase +0.25 cycle (+25 ns)
OUT2 = 10 MHz sine, 0.5 V amplitude (reference)
OUT1: sine on, amplitude 0.50000 V, offset +0.00000 V, phase +0.250000000 cycle (word
 ↪  4611686018427387904)
OUT2: sine on, amplitude 0.50000 V, offset +0.00000 V
```

Where the state is kept

As for every design: `/run/rpfpga/atomic_clock/state.json` (shared by every program on the board), copied to `/home/xilinx/rpfpga/atomic_clock/state.json`, from which the service restores it after a reboot. The playback log, checkpoint, replay records and stability recordings are in `/home/xilinx/rpfpga/atomic_clock/playback/`, `.../replay/` and `.../recordings/`. The board calibration is in `/home/xilinx/rpfpga/calibration.json`, shared by every design.

Register map

For debugging only: normal use goes through the driver, client or TCP commands (PROTOCOL.md), which work in volts, cycles and fractional frequency. Phases are in units of 2^{-64} cycle; 64-bit values are split into LO (bits 31:0) and HI (bits 63:32) words.

Base	Size	Block
0x40000000	4 KiB	common sysid (read-only identification), design ID **"ACLK"**, version 3.2: see `../../common/README.md`
0x43C00000	4 KiB	`atomic_clock_0` (`src/ac_core.v`, registers in `src/ac_regs.v`)

Configuration (shadowed: take effect on commit)

Offset	Name	Contents	After loading
0x008	MODE	bit 0: OUT1 sine on, bit 1: OUT2 sine on (off: the offset alone), bit 2: OUT1 dither	`0x3`

Offset	Name	Contents	After loading
0x00C	OUT1	bits 31:16 offset (signed, stream codes), bits 15:0 amplitude (unsigned, stream codes)	`0x00004000`
0x010	OUT2	as OUT1	`0x00004000`
0x014, 0x018	PHASE_LO/HI	OUT1 phase target relative to OUT2 after ALIGN: fraction of a cycle (bit 63 = half a cycle)	0
0x0E0	PHASE_INT	its whole cycles (signed; with PHASE_HI/LO a 96-bit signed phase)	0
0x01C, 0x020	SLEW_LO/HI	largest change of the static phase per sample, R (0: steps)	`0x00000157_98ee2309` (y = 10^{-6})
0x024, 0x028	RATE_LO/HI	frequency offset ν_base, signed, per sample	0
0x02C	SEG_LEN	samples per FIFO segment (values below 16 act as 16)	125 000 (1 ms)
0x030	PM_GAIN	white-PM gain (unsigned): σ = 107 019 · gain · 2^(exp−16) LSB, rounded half away from zero, saturated at ±(2^{63} − 1)	0
0x034	PM_EXP	white-PM exponent, bits 4:0: σ = 107 019·gain·2^(exp−16) LSB	0
0x048	FM_GAIN	white-FM gain, as PM_GAIN	0
0x04C	FM_EXP	white-FM exponent	0
0x050, 0x054	SEED_LO/HI	noise seed; a commit that changes it restarts the generators	0
0x0E4	PPS_WIDTH	PPS pulse width in samples (0 acts as 1)	2500 (20 µs)
0x0E8	METER_M	meter window in units of 25 samples (2 carrier cycles; 0 acts as 1)	50 000 (10 ms)

Writes go to shadow copies, which read back what was written; writing 1 to CTRL bit 0 copies all of them to the datapath on one clock edge, so 64-bit values and related settings change together. Byte strobes are honored: in CTRL, byte 0 carries bits 0–7 and byte 1 bits 8–11 (each acts only with its strobe). A write with no byte strobes has no effect on data, dirty status or control strobes, including FIFO pushes. Write COMMIT and ALIGN together when setting a phase: at edge A, one clock after COMMIT, the phase path sees ALIGN with the new configuration. It holds the static phase and discards old slew candidates while the rate and noise pipelines drain. At A+5 it captures φ_d + φ_w, at A+6 it publishes origin and PHASE − origin together, and at A+8 it first updates the static phase. A repeated ALIGN restarts this wait; neither accumulator is paused or reset. The driver adds STOP|CLEAR to COMMIT whenever the applied configuration has playback off, so queued rates stop before origin capture.

Control and status

Offset	Name	Access	Contents
0x000	ID	R	`0x41434C4B` (”ACLK”)
0x004	CTRL	R/W	write: bit 0 commit, bit 1 PLAY, bit 2 STOP (segment rate 0), bit 3 CLEAR the FIFO, bit 4 clear the UNDERFLOW/OVERFLOW flags, bit 5 SNAPSHOT, bit 6 zero φ_d, bit 7 zero φ_w, bit 8 ALIGN (origin = φ_d + φ_w, 96 bits; the static phase then moves to PHASE − origin), bit 9 METER POP (the oldest meter window into METER_DATA), bit 10 METER CLEAR (empty the meter FIFO, clear its overflow), bit 11 PPS ARM (both PPS restart at sample PPS_EPOCH, counts reset) (strobes, several at once allowed). Read: bit 0 = uncommitted writes pending
0x038	LATENCY	R	clocks from φ_s/φ_d to the core’s DAC stream: 13 (φ_w: 12, white PM: 11); excludes ALIGN settling and the vendor DAC interface
0x03C	VERSION	R	`0x00030002` (3.2)
0x040, 0x044	COUNT_LO/HI	R	free-running 64-bit sample counter; reading LO latches HI
0x058	STATUS	R	bit 0 playing, bit 1 UNDERFLOW (sticky), bit 2 OVERFLOW (sticky), bits 28:16 FIFO level (0 … 4096)
0x05C	FIFO_DEPTH	R	4096
0x060	FIFO_LO	W	low half of the next segment rate
0x064	FIFO_HI	W	high half; the write pushes {FIFO_HI, FIFO_LO} (signed rate per sample)
0x0EC	PPS_COUNT	R	simulated-clock seconds counted (DIO1_P)
0x0F0	PPS_REF_COUNT	R	reference seconds counted (DIO2_P)

Offset	Name	Access	Contents
0x144, 0x148	PPS_EPOCH	R/W	sample index (COUNT timebase) of the second armed by CTRL bit 11; written directly, not a shadowed setting
0x0F4	METER_STATUS	R	bit 0 METER_DATA valid, bit 1 overflow (sticky), bits 25:16 windows in the FIFO (0 … 512)
0x0F8, 0x0FC	METER_START	R	sample index at which window 0 began
0x100	METER_INDEX	R	window index of METER_DATA
0x104 … 0x120	METER_I1, Q1, I2, Q2	R	that window's sums (64-bit signed, LO/HI each)

Segments. After PLAY the first segment starts on the next sample; each segment of SEG_LEN samples takes the next FIFO entry as its rate ν_i, back to back. An empty FIFO holds the last rate and sets UNDERFLOW; segments continue on the same grid and take entries again as soon as there are some. STOP sets the rate to 0 (φ_d then advances by RATE only); CLEAR empties the FIFO. Pushes must be at least 2 clocks apart (every AXI write is). A BRAM fetch completing at a segment boundary is forwarded into that segment. A new underflow/overflow on the same edge as CLEAR_FLAGS keeps the corresponding sticky flag set.

Static phase. PHASE and origin are signed 96-bit words, but their difference and the internal static phase are signed 97-bit. The slew difference is signed 98-bit to handle moves between opposite extremes without overflow. Each two-clock slew operation latches its target, candidates, bounds and step mode together.

Snapshot (read-only; captured by CTRL bit 5 on one clock edge)

CTRL bit 5 requests a snapshot. During ALIGN the request waits until the new origin is published, then a common strobe captures phase, sample count and PPS state on one edge. The capture is at least one clock after the CTRL write, so a SNAP written together with other strobes (CLEAR_FLAGS, PLAY, zero φ_d, …) sees their effect: write SNAP alone to see the state before them. An AXI read whose address is accepted after the SNAP write's response waits for the capture to complete; the read and write channels are independent, so issue the read after the write completes. Public phase words retain their low 96 bits modulo 2^{96}, including the internally 97-bit static phase; signed readback does not expose its extra bit.

Offset	Name	Contents
0x080, 0x084	S_COUNT	sample counter
0x088, 0x08C	S_SEGMENT	index of the latest segment (segments started since reset)
0x090, 0x094	S_TAKEN	FIFO entries taken up to and including that segment
0x098	S_REMAIN	samples left in the current segment
0x09C	S_STATUS	as STATUS
0x0A0, 0x0A4	S_NU	that segment's rate (signed)
0x0A8 … 0x0B0	S_SEGMENT_PHASE	φ_d at that segment's start (96-bit signed, 3 words)
0x0B4 … 0x0BC	S_PHASE_D	φ_d now (96-bit signed)
0x0C0, 0x0C4	S_PHASE_S	the static phase now (low 64 bits; whole cycles at 0x124)
0x0C8 … 0x0D0	S_PHASE_W	φ_w, the white-FM random walk (96-bit signed)
0x0D4, 0x0D8, 0x128	S_ORIGIN	the origin latched by the last ALIGN (96-bit signed; 0x128 the whole cycles)
0x124	S_PHASE_S_INT	bits 95:64 of the internal static phase; with S_PHASE_S, its low-96-bit modular value
0x12C, 0x130	S_PPS_FIRE	sample index of the latest simulated-clock PPS edge
0x134, 0x138, 0x13C	S_PPS_OVER, S_PPS_FRAC	unsigned remainder S_PPS_OVER + S_PPS_FRAC/2^{64} samples beyond the second threshold at that edge; whole samples saturate at 0xffffffff; may exceed one sample on jumps
0x140	S_PPS_COUNT	PPS_COUNT at the snapshot

The latest segment's fields (S_SEGMENT, S_TAKEN, S_NU, S_SEGMENT_PHASE) are latched together when that segment starts contributing, two clocks after its boundary, so they are always consistent with each other: S_SEGMENT_PHASE(i+1) = S_SEGMENT_PHASE(i) + L·(RATE + S_NU(i)), L = max(SEG_LEN, 16). Once the latch has happened (L − S_REMAIN ≥ 2), also S_PHASE_D = S_SEGMENT_PHASE + (L − S_REMAIN − 1)·(RATE + S_NU). The current ν contributes S_REMAIN + 1 further times through the accumulator pipeline (with no intervening rate/control change); software uses this convention when rebuilding its mirror. A snapshot within the latch's two clocks (L − S_REMAIN < 2 while playing) pairs the new segment's count with the previous segment's fields; the driver then takes another.

Stream codes are the 16-bit signed samples sent to the DAC interface, which uses their top 14 bits; nominally 32768 codes per volt. **Power-up**: both outputs carry the 10 MHz sine with amplitude 16384 codes (0.5 V nominal; about 0.57 V on this board, computed from the calibration), offset 0, φ = 0, no noise, the default slew.

Phase meter. Windows of 25·METER_M samples start on carrier index 0 and follow each other without gaps; a new METER_M (on commit) restarts them at index 0 but preserves old FIFO entries. Issue METER CLEAR when changing the raw window setting before interpreting entries with the new METER_START and window length. For each window, I = Σ x·C[k] and Q = Σ x·S[k] per input (THEORY.md "Phase meter"). To read: METER_STATUS for the count, then for each window CTRL bit 9 (POP) and the nine words at 0x100 … 0x120.

Verification record

What was tested on atomic_clock, when, and with what result, newest first. Only results that were actually obtained are listed; anything not yet done is said so.

How to repeat the checks: `make test`, `make sim`, `make`, `python3 ../common/tests/verify_build.py --design atomic_clock` and `make test-remote` (see "Changing the design" in ../README.md). `make test-hw` (loopback OUT1 → IN1, OUT2 → IN2) checks phases, slew, frequency offset, noise and PPS counters with the board's phase meter; check the outputs with an oscilloscope, a frequency counter or a spectrum analyzer.

2026-10-17: theory manual units and oscillator specifications

Added the units/specification-reading section (`docs/manual/metrology.tex`), about 14 pages with five generated figures and eleven tables. It covers phase/time/frequency units, one-sided spectra and dBc/Hz, the five noise types, ADEV/MDEV/TDEV and bandwidth, finite-band spectral integration, revision-qualified OX-171/OX-174 and SA65/SA65-LN specifications, and conditional holdover budgets. The white-PM 0.05-cycle limit's conversion was corrected from −115 to −91.03 dBc/Hz (phase-PSD convention).

- `base-tools` Python, `tools/manual_figures.py --only metrology`: all five figures and `metrology_values.tex` generated. The independent selection preserves existing measured-floor/catalogue files; it does not need a floor recording.
- `test_metrology_figures.py`: **7 tests passed**, including direct positive quadrature versus oscillatory integration, analytic FM limits, exact finite-band white PM, continuous-MDEV coefficients, carrier/power scaling, generated-number freshness, and preservation of the other generated tables. The synthetic 1 Hz–100 kHz example gives 0.203990 ps integrated time RMS and ADEV contribution 3.38559e-13 at 1 s; it is explicitly not a full-spectrum prediction or a vendor guarantee.
- `test_manuals.py`: **3 tests passed** (source excerpts and theory/user builds).
- `test_stability_tools.ManualTests` (`base-tools`): **3 tests passed**, including existing measured-floor/catalogue freshness, slope normalization, and testing-manual build.
- `../common/tests/test_designs.py`: **4 tests passed** (framework uniformity).
- Theory PDF rebuilt with `latexmk -pdf -interaction=nonstopmode -halt-on-error`; no undefined citations/references or overfull boxes. The new section was rendered and visually reviewed, with the final OCXO/CSAC table/figure layout checked again. The conflicting OX-174 temperature and SA65 ADEV entries were also checked against rendered manufacturer pages; the text identifies them rather than inventing a correction.

2026-10-14: one-day loopback floor repeated on build 0x4bcaf5ab (T0b-long)

`run_tests.py --tier long --tests T0b-long --prefix long-20261013 --unattended`, same setup as 2026-10-10 (loopback, 0.5 V, no noise, 24 h): **PASS**, 86 401 slow rows, no gaps, no resyncs, no write error, download coverage 100 %; IN1 0.676 V, IN2 0.668 V. Report: `tools/stability-tests/long-20261013/reports/T0b-long.pdf`.

ADEV of IN1 − IN2	10 ms	0.1 s	1 s	10 s	100 s	1000 s	10^4 s
2026-10-10 (previous build)	7.04e-12	9.96e-13	1.16e-13	1.93e-14	2.38e-15	2.76e-16	5.0e-17
2026-10-13 (this build)	7.13e-12	9.26e-13	1.20e-13	1.76e-14	2.08e-15	3.17e-16	7.3e-17

Fit: h2 4.45e-28 (was 3.68e-28), h1 2.90e-26 (3.03e-26), h0 2.87e-28 (1.87e-28). The two floors agree within the scatter expected of one-day runs (the 10^4 s point has only a few degrees of freedom); the ALIGN/SNAP RTL changes did not change the floor. The Zynq die temperature varied more than on 2026-10-10 (70.5–76.9 °C against 71.5–73.3

°C), and the time error followed it (3.0 ps peak to peak, correlation −0.73, against 2.0 ps and 0.04).

Manuals switched to this floor (`manual_figures.py --floor stability-tests/long-20261013/recordings/long-20261013-T0b-long`, all figures, `values.tex`, `apps.tex`; the three manuals rebuilt): white PM $2.9 \cdot 10^{-14}$ s per 10 ms window (was 2.6), test levels T5–T7 about 1.2–1.5× higher. The testing manual's floor paragraph now says the time error follows the die temperature. The cable note's single run gives 3.31 ± 0.21 ps/K and 21 min (ensemble 2.58 ± 0.44 ps/K, 9–17 min; true 2.5 ps/K, 10 min).

2026-10-13: live checks before deployment (commit 82845fd)

The board's `control/atomic_clock/*.py` match the commit (md5, file by file); build 0x4bcaf5ab.

- **Recording** `deploy-smoke-20261013` (3 min, 10 ms windows, slow 1 s, loopback): stopped "duration reached" at 180.06 s, write_error null; 181 slow rows exactly 1.000 s apart, 18 005 fast rows 10 ms apart, no flags, no gaps, no resyncs; header complete.
- **rustle** `--headless --board` during and after the recording (126 s, no bootstrap): ADEV $7.1 \cdot 10^{-12}$ at 10 ms (as `make test-hw`), $9.3 \cdot 10^{-13}$ at 0.1 s, $1.2 \cdot 10^{-13}$ at 1 s; two segments, the break at the recording's end (the meter handed back); JSON export written.
- **Root `make test`:** framework 159, common 18, atomic_clock 228, mult_add 121, all OK.
- The service journal had no underflow, stall or error during these checks; worker CPU about 12 %.
- Not repeated after the ALIGN/SNAP RTL changes: the long stability tiers (the manuals' measured floor is from 2026-10-10).

2026-10-13: output and input levels at 10 MHz (oscilloscope)

Scope with a 1 MΩ input, DC coupled, 1× (direct coax, 1 m), with and without a 50 Ω feed-through; default configuration (0.5 V amplitude, offset 0), or amplitude 0 and an offset for the DC readings.

Signal, load	Set	Measured
OUT1 / OUT2 sine, 1 MΩ	0.5 V amplitude	2.08 / 2.12 Vpp (scope amplitude measurement); 2.12 Vpp read directly on a recheck
OUT1 / OUT2 DC, 1 MΩ	+0.5 V / −0.5 V	+0.53 V / −0.46 V
OUT1 sine, 50 Ω feed-through	0.5 V amplitude	**1.000 Vpp**
OUT1 DC, 50 Ω feed-through	+0.5 V	0.5 V
IN1 meter, OUT1 through the feed-through	(0.5 V peak by the scope)	0.528 V
IN2 meter, OUT2 → IN2 direct		0.668 V

- At DC the levels are right into either load (the DC calibration, made with a DMM; the scope's ±35 mV offset is within its accuracy). At 10 MHz the set amplitude is the level into 50 Ω; a 1 MΩ load sees about 2.1×: the outputs behave like a 50 Ω source there. Other frequencies, and the mechanism, were not investigated.
- The LV input reads 1.06× at 10 MHz (scope reference). In direct loopback it reads 0.67 V for the 0.5 V setting, i.e. about 0.63 V at the connector: the input loads the output much more than the scope does.
- The feed-through on IN1 alone moved the meter's IN1 − IN2 difference from −0.001 to −0.105 cycle: both inputs must be cabled and terminated alike.
- Changes: `scripts/test_hw.py` checks the input amplitudes between 0.45 and 0.8 V and IN1/IN2 within 5 % (was 0.5 ± 0.05 V); the load statements in the root README (safety: up to about ±2.4 V into high impedance at 10 MHz), README, CALIBRATION.md, the notebook and the user's and testing manuals.
- `make test-hw` afterwards, direct loopback: **PASS, 15 checks** (IN1 0.676 V, IN2 0.668 V, ratio 1.011; meter floor 8.5e-12 at 10 ms; no worker underflow or stall).

2026-10-13: upload of the second review fixes; loopback hardware test

`make upload` (build 0x4bcaf5ab unchanged; the uploaded driver, recording and client files match the local ones), service restarted (no recording running). `make test-hw`, loopback OUT1 → IN1, OUT2 → IN2, LV inputs, run twice:

Check	Result
Phase steps +0.25, −0.30, +0.10 cycle	PASS (within 0.6 mcycle)
Slew y = 1e-8; frequency offset 1e-9	PASS (0.09975 cycle/s; 0.009967 cycle/s)
Meter floor, ADEV of IN1 − IN2	7.2e-12 at 10 ms, 0.85–1.05e-12 at 0.1 s

Check	Result
White FM ADEV 1e-9 at 1 s, at 10 ms / 50 ms	6.49e-9 / 3.86e-9: PASS against the window-averaged expectation 7.07e-9 / 4.24e-9
Played-back flicker FM (flat 1e-9) at 20 ms / 0.1 s	PASS (0.92–0.94e-9; 0.85–0.90e-9)
PPS counts (reference and OUT1)	PASS
IN1 / IN2 amplitude of the 0.5 V sine	**FAIL** against the old 0.5 ± 0.05 V: 0.674 V / 0.667 V (correct, see below; check changed)

- The white FM check expected the point-sampled ADEV; the meter averages the phase over each window, which multiplies the Allan variance of white FM by $1 - \tau 0/(2\tau)$ (checked by simulation: 0.495 and 0.899 for $\tau = \tau 0$ and $5\tau 0$). `scripts/test_hw.py` now includes the factor; the first run, with the old expectation, failed at 10 ms (6.48e-9 against 1e-8 ± 3e-9).
- The amplitudes read 0.67 V; the one-day floor recording (T0b-long, 2026-10-10) read the same (0.6756 / 0.6681 V). The scope measurements below show that this is the correct reading of a direct loopback; the check expected the wrong level.
- The worker logged one 5.9 s stall in its config step when the first test run's driver attached right after the service restart; none during the second run.

2026-10-12 (night): second review of the review fixes

A read-only review of the entry below (RTL clock by clock, control, host tools) confirmed its RTL changes and statistics; these defects were fixed (software and documentation only, no RTL change, so the bitstream is unchanged):

- **Rustle:** the exact segment-layout trend constants were recomputed from scratch on every analysis of a run with gaps (13.2 s for 400 segments, about 230 s for 3000). Same formulas, shared per-segment Fourier basis, exponentials by blocks on the uniform grid, closed-form spherical-Bessel integrals: **1.1 s and 9.5 s**, constants equal to the old ones within $1.3 \cdot 10^{-13}$ (relative). New tests: direct Fourier sums, and a time bound. The published snapshot is checked against the model revision (a toggle back and forth during an analysis no longer publishes the old one); file sources reject time tags that go backwards.
- **Recordings:** slow-row cadence and `t_s` from integer window indices (absolute float times would flag ordinary slow rows after about two months of board uptime; tested at 200 days); a failed `header.json` write at the stop still reports "stopped" with the error; slow interval compared with the quantized window; numpy numbers accepted.
- **Driver:** a snapshot within two clocks of a segment boundary (the segment fields' latch) is taken again.
- **Analysis files:** delimiter chosen from data lines (a comma in a header no longer breaks whitespace data); aligned tab columns; blank lines before/after the data. The runner closes a replaced client and rejects another design after reconnecting.
- **Cable application note:** the sign-convention change had made its single simulated run read 3.07 ± 0.15 ps/K and 18 min (true 2.5 ps/K, 10 min). Over 16 further runs the fit scatters ±0.37 ps/K about 2.57 ps/K, lag 9–16 min: the method is unbiased, one run's error is optimistic. The manual now quotes the ensemble too.
- **Documentation:** SNAP with other strobes and read ordering (REGISTERS, theory manual); `meter_read()` refusal within the service's process only; the irregular slow-row flag (PROTOCOL); Node ≥ 22.13 for the web unit tests (AGENTS).

Check	Result obtained
`make test` (system Python)	**228 tests OK** (123 skipped: optional host dependencies)
`test_rustle` (base-tools)	**68 tests OK** (15 opt-in skipped)
`test_stability_tools` + `test_adev1` (base-tools)	**61 tests OK**
Manuals	testing and theory manuals rebuilt; `test_manuals.py` OK

Uploaded on 2026-10-13 (entry above).

2026-10-12 (late): review fixes — recordings, segmented statistics and RTL

The code/documentation review was followed by implementation and regression checks.

- **Recording and control:** fast/slow series retain discontinuity flags independently; recovery partway through a slow interval also flags the irregular timestamp step. Timed recordings use board time even without signal; write errors survive in the downloaded header and fail the stop. Non-integral slow/window cadences are rejected; a changed meter window ends the recording without restoring over the user's edit. Service `MEAS? n`

averages n worker windows during recordings and meter logging, with acquisition ownership and cancellation. RESET stops/clears the FIFO in the alignment commit, and trajectory preparation rechecks the current settings before PLAY. PPS status sums modular snapshot words modulo 2^96; static-phase predictions lift the modular snapshot using observed motion, including long unwrapped slews.

- **Host analysis:** empty CSV fields no longer shift columns; missing quantitative checks cannot pass configured noise tests. The runner adopts a reconnected client, checks write errors, cadence, row counts and acquisition coverage. Replay expectations include scale, interpolation and actual windows/cadence; cached records check duration and generator identity. Added correlated bandlimited-white-PM kernels, full logarithmic aging removal for the noise fit, identifiability checks for phase steps/sines, and mask eligibility based on the actual analyzed segment. D-mains uses 5 ms windows.
- **Rustle:** count-weighted segment EDFs, discrete segment-aware trend errors, gap-safe historical tau backfill, a fixed fit grid independent of display choices, and internally consistent fit/EDF/drift debiasing. Stop/error flush pending MDEV terms; bootstrap jobs track model revisions and actual estimator counts. Exact segmented replicas are used through 2^21 timeline samples; longer runs explicitly report the EDF-rescaled approximation. Source validation, websocket cleanup, exports, sampled PSDs, strip detrending and uncertainty/frozen-cell displays were corrected.
- **RTL:** coherent slew candidates, signed 97-bit static/effective phase and 98-bit difference, settled ALIGN origin capture with deferred coherent SNAP/read response, first-entry FIFO forwarding, symmetric noise saturation, and side-effect-free null write strobes. PPS delay is 9 clocks: the integrated vendor DAC/ODDR test reproduced the old 8 ns early edge and passes with the correction. Analog delays remain unmeasured.

Offline checks

Check	Result obtained
`make test` (system Python)	**221 discovered, 102 passed, 119 skipped**, 27.230 s; skips are optional host dependencies and ipywidgets
Shared `common/tests` discovery	**159 discovered, 154 passed, 5 skipped**, 12.257 s
`test_stability_tools` + `test_adev1` (base-tools)	**60 passed**, including generated values, report/manual builds and byte-identical adev1 reference output
`test_rustle` (base-tools)	**65 discovered, 50 passed, 15 opt-in skipped**, final run 73.669 s
Selected `RUSTLE_SLOW=1` coverage checks	**11 passed**, 125.303 s + 163.413 s; cases and bootstrap approximation comparison in `tools/rustle/README.md`
Rustle web	**3 unit tests + 25 browser checks passed**; Node 26.10.0 TypeScript/Vite build passed and `static/` rebuilt
`make sim` (Vivado 2025.1)	**All seven PASS receipts**: sysid, sine, noise, phase, PPS, meter, system
`make`	**Build 0x4bcaf5ab**, setup slack **+0.565 ns**, hold **+0.022 ns**; TNS/THS zero; DRC warnings only (2 DPIP-1, 10 DPOP-2, 1 RTSTAT-10)
`verify_build.py --design atomic_clock`	**All eight stages passed**, 1879.4 s; fresh recipe, reuse/export recovery, preserved-mtime vendor edit, RTL edit, failed-build upload block, clean and saved-project regeneration
Manuals	Affected figures/tables regenerated with the measured `long-20261010-T0b-long` floor; all three PDFs rebuilt, excerpts match sources, no undefined citations/references or duplicate labels

The full design uses **6300 LUTs, 9449 registers, 16.5 BRAM tiles and 12 DSPs**. The independent recipe verification restored the same build ID, with setup slack +0.566 ns and hold +0.022 ns; its logs/receipt are under `.build/verification-yvu8nbfx/`. No unconstrained internal endpoints were reported; the existing external-I/O methodology warnings remain.

The first combined host run found a timing-sensitive dashboard test: it assumed a pause snapshot was published within 0.6 s while a fit was in progress. It now polls for the asynchronous result with a bounded timeout; the complete Rustle suite then passed. The full throughput benchmark and the two remaining long ErrorBar cases were not rerun.

Board checks

`make upload`, service restart, and safe client attachment confirmed **0x4bcaf5ab**. The saved configuration was restored; final deployed driver/playback/recording SHA-256 hashes match the working tree. Targeted live checks obtained:

- `MEAS? 5` returns five worker windows, including during a recording and while a stream is open. The review stream received **519 windows, lost = 0**.
- A competing acquisition returned **BUSY**; a frequency edit cancelled the pending 1000-window acquisition in **0.092 s**.
- `review-gap-20261012-215401`: a short input mute produced flags in both CSV series; **273 fast rows, 4 slow rows**, slow flags **0, 1, 0, 0**, no write error.
- `review-no-signal-20261012-215401`: no valid input windows, **0 fast rows**; the 2 s recording stopped with `duration reached` at **2.118 s** board time.
- RESET from segment playback, a frequency offset and hardware white FM, followed by `SLEW 0`, left **exactly zero modular phase**, playback stopped and FIFO level zero.
- `top`: **66.0–72.0 %** of one CPU while flicker-FM playback and meter clients were active (another TCP client was connected, so this is not an isolated playback-only comparison). After the final software deployment, the restored playback-off state used **10.0–11.0 %**. The journal had no underflow or mirror mismatch in the live check interval. One **2.60 s configuration-stage stall** was logged just after the first restart, before model playback was enabled; none followed the final restart in the observed interval. Original settings were restored after the targeted checks.

These checks are targeted live regressions, not a repeat of the long stability tiers or an analog PPS/DAC timing measurement.

2026-10-12 (night): documentation sweep before handoff

Three read-only reviews compared the documentation with the code (commands, settings, ranges, register map, internals, host tools). The register map, version 3.2, the testbench list, command syntax, option names and defaults agreed.

- **Corrected:**
  - `SLEW 0` *before* `RESET` or a preset has no effect, because they set the default slew themselves. The README, PROTOCOL and the user's manual now say to send it after.
  - The README's verification summary (3.2 "not yet on the board") and the meter resolution ("not yet measured": 26 fs per 10 ms window from the one-day floor).
  - THEORY's playback formula (it now has the moving baseline B_n = B + nLR) and the meter unwrapping (the worker's prediction; `meter_advance` only without the service).
  - The mirror-check interval (at least every 10 s, independent of `log_interval`), and `MEAS?` during a recording (averages n windows).
  - Limits added to PROTOCOL: PPS width, recording options, h2/h0, replay slope, MEAS? n.
  - In the stream: `lost` is cumulative, and a reload ends a stream with `ERR STALE`. Recording flags are given as values 1 and 2.
  - rustle: the README path, tail size and workers default; `--floor` without a source now gives an error instead of being ignored; the headless message names `--board`.
  - The notebook measured with white FM still on.
  - "Power-up state" said where the default configuration is meant.
- **RTL comments:** the header of `src/ac_regs.v` now lists every shadowed register and all strobes; `ac_meter.v` and `ac_regs.v` give METER_M in units of 25 samples (2 carrier cycles), not "carrier periods". Comments only.
- **Build** 0xeb1fac7b (`make`): timing met, worst setup slack +0.189 ns, hold +0.018 ns (as 0x2570da4a); routed DRC: warnings only (DSP pipelining, one unloaded net); 6267 LUTs, 9112 registers, as before. `make sim`: all seven testbenches PASS.
- **Board:** `make upload`, service restart (no recording running). `STATUS?` reports build 0xeb1fac7b; the flicker-FM playback resumed from its checkpoint; `MEAS? 5` from the worker; `PRESETS?` shows the corrected text; the deployed driver, playback and protocol files match the repository; service CPU 35 %; no errors in the journal.
- **Tests:**
  - `make test` OK, `test_rustle` OK, `test_stability_tools` + `test_adev1` 41 OK, and `test_manuals` OK (the three manuals rebuilt).
  - The board runs the committed service code (file hashes compared). Its journal since the last restart has no stall, underflow or mismatch lines; service CPU 41 % (top).

2026-10-12 (evening): second bug hunt (board chain and rustle)

Three read-only reviews (board playback and meter drains, rustle statistics, rustle data path and dashboard) found about 35 problems, most reproduced offline; each fix has a test that fails without it (checked against the previous commit for the board side).

- **Board, wrong or unflagged data (fixed):** windows without signal, or in which the phase turned by many cycles (a fast slew or PM, a phase step: the I/Q sum loses its amplitude), were unwrapped like any other, so their noise-like phases added whole cycles unflagged, and a phase step (ALIGN) was not flagged. Such windows are now dropped and start a new run, and OUT1's phase, slew and alignment are in the continuity key. IN1 is also unwrapped against OUT1's phase as the worker predicts it (the played segments, the slew); within the meter's limit (less than about a quarter cycle per window) the old offset-only unwrapping could not slip, so this is a safeguard more than a fix. (The offline figure first quoted here, 88.5 cycles shown for 3628.5 real, came from a fake meter that kept its amplitude at 12.3 cycles per window; on the board those windows have none. The fake now loses amplitude as the board does.) A replay uploaded again under the same name kept playing the old record. A model change racing the worker's step was marked as played but not played. A frequency-offset change restarted the playback (1 s of zero rates: the trajectory's phase lost); it now re-derives the mirror and goes on. Model changes and play stop/start were not flagged; the meter log's overflow column stuck at 1 and its lines mixed runs; MEAS? could be stale; the playback log's baseline column was wrong; a recording left the meter window changed; the rate clamp could leave int64; verification stopped with log_interval 0.
- **Found only on the board:** the snapshot's `remain` is one less than the samples the current rate still adds (ac_phase: boundary at remain = 1, next nu two clocks later). The first re-derivation was off by one sample of rate + nu, and the board reported "mirror mismatch by 147573977217 LSB" (1e-7 offset) and "by 31039 LSB" (ν alone). Fixed, and the fake core now counts `remain` as the hardware does; no mismatch since.
- **Live, after the fixes** (10 ms windows, flicker FM playing): OUT1 offset +1e-6 then 0: one flag each, steps 10.001 ns (9.98–10.03) and 0.000 ns; −3e-6: −29.998 ns (−30.06…−29.94); no mismatch, no stall. A 500-cycle slew at 10 cycles per window: the windows during the slew have no usable amplitude and are dropped; one flag, the run after it continuous (no whole-cycle jump). A phase step back to 0 after 1e-7: flagged; the following run advances 3.67 cycles in 3.7 s (the offset). Replay re-upload: record +1e-6, then −3e-6 under the same name: measured 1.000e-6 and −3.000e-6 (twice; a first attempt right after a service restart read 0 and −4e-6 and was not reproduced: the hardware's φ_d moved at 1.000e-6 throughout when checked directly).
- **Worker stalls after a restart:** `gc.freeze()` after the worker's first step; no stall in the journal after three restarts (before: two of 2.5–3 s, one underflow).
- **rustle statistics (fixed):** intervals where white PM and white FM cross were far too narrow (coverage 0.08–0.27; the EDF of the dominant term alone): now the EDF combines the terms (with cross products), coverage 0.67–0.81 at m = 500–2048 (0.52–0.62 at a few τ below). Offset and drift errors ignored segments (ratios 0.18–13 in 20-segment runs): now computed for the segment layout, model 0.98–1.49, bootstrap 1.02–1.43. Under a strong drift h_{-2} was fitted to 0 and the drift error was 4× too small: HVAR is now fitted too, ratio 0.99–1.00. The drift term is $D^2 - \sigma^2$ unclipped (h_{-2} in 10-min runs 0.79 of true, was 0.63). A one-term MDEV point had EDF 0 and crashed every bootstrap replica; MDEV terms held back at gaps were dropped; gap padding counted as samples.
- **rustle data path (fixed):** time-tagged files split into segments depending on the read size (1, 40 or 2 segments for 100, 5000, 131072): now one. Strip chart points across a gap; two concurrent starts left an orphan producer; a board stream opened during close() was leaked; CLI options ignored in server mode; the UI's true offset after a reset; a restarted server left the old run on screen; Play in the error state did nothing; bad inputs gave HTTP 500; recordings' windows without signal were used; paused board windows were counted as lost; a floor shorter than the run was over-subtracted (now extrapolated along its slope, never rising, with a note); the PSD model lacked aliasing (measured/model near Nyquist 1.9–2.4, now 1.00–1.02).
- **Tests:** `make test` OK; `test_playback/test_control` 77 OK; `test_rustle` OK; `RUSTLE_SLOW=1` ErrorBarTests 11 OK (31 min); `test_stability_tools` 35 OK.
- **Known limits (accepted, documented in the rustle README):** h_{-2} 10–40 % low in short runs (the non-negative fit; within one run's error); drift error up to 2.5× conservative in runs of many short segments (a spurious h_{-2}). The one-off `MEAS?` path (no stream, no log) unwrapped against the frequency offset alone: fixed the same evening, it now averages the worker's windows (`test_measure_goes_through_the_worker`), also while streams are open (it returned the latest single window then). Live, during a replay at 0.2 cycle per window: `MEAS? 5` = the first window + 0.40 cycle each time (source "worker"). Board CPU after reading the mirror on demand: 38.7 % without a stream, 66.8 % with one (56 % before this round; 80 % with the first version of the prediction).

2026-10-12 (afternoon): rustle slow error-bar tests

`RUSTLE_SLOW=1 python -m unittest tests.test_rustle.ErrorBarTests` (base-tools python; about 20 min on 4 workers, under 1 GB). All 7 pass:

- 68 % intervals of ADEV, MDEV and HDEV over 40 runs each, pooled coverage within 60–78 % and every τ within 40–95 %: white PM (up to m = n/3), white FM, flicker FM, random-walk FM; gappy runs (3000-sample segments, 500 lost between); after the store halved (r up to 32).
- Offset and drift model errors within 0.7–1.4× the scatter for each of the four noise types.
- Bootstrap against the scatter (h terms present 0.6–1.7×, offset and drift 0.6–2.2×): a recording's 1 s slow series, a file of instantaneous samples, one hour of 10 ms windows.
- The bootstrap serves every long τ at decimations k = 2, 3, 6, 7.
- Found and fixed on the way: after the store halved, white-PM intervals at long τ were 4–5× too narrow (coverage 0.18–0.25 at m = 16384–20000: the EDF counted the store's strided terms as all the overlapping ones). A segment's EDF is now that of the run sampled at its terms' mean stride r, at m/r (reported 1250 against 1060 measured; coverage 0.68–0.72).

2026-10-12 (afternoon): board upload; frequency offset during noise playback

- `make upload` (build reused, build ID 0x2570da4a) and a service restart; no recording was running.
- **Bug found on the board and fixed:** with noise playing (here flicker FM, $h_{-1} = 5.6 \cdot 10^{-30}$), a frequency-offset change held for only about 1.1 s. After the worker's resynchronization the played rates cancelled it: the stream's IN1 phase advanced 10 ns per 10 ms window for 110 windows at $y = 10^{-6}$, then 0. The mirror's target lacked the L·rate term (THEORY "Exact mirroring"; theory manual updated). Without noise playback the offset was never affected (T4, `test_hw.py`). Offline test: `test_frequency_offset_holds_while_noise_plays`.
- After the fix, live, 10 ms windows, offset switched on for 5 s and off again, IN1 phase step per window between the stream's flags:

y	expected	measured (median; all windows within)
$+1 \cdot 10^{-6}$	10 ns	10.00 ns (±0.03 ns)
$-3 \cdot 10^{-6}$	−30 ns	−29.99 ns (±0.06 ns)
0 (off)	0	0.000 ns (±0.005 ns)

  Each change, and the worker's resynchronization, started a new segment (flagged); no whole-cycle slips. The off-switch takes effect in the hardware a few windows after the stream's flag (one partial window). Near $y = 5 \cdot 10^{-6}$ (half a carrier cycle per 10 ms window) unwrapping cannot work, by principle.
- Twice, just after the restart, the worker stalled 2.5–3 s in a refill (one underflow, resynchronized and logged); not reproduced in 8 further runs. Watch the journal.

2026-10-12 (afternoon): rustle error bars after the code review

Independent demo runs (seeds), each fitted as rustle does; "ratio" = reported error / true scatter of the estimates over the runs (1 is right).

Source (simulated)	Runs	h terms present: fit / bootstrap ratio	Offset: model / bootstrap	Drift: model / bootstrap
recording slow series (1 s points of 10 ms windows; white PM + white FM, 12 h)	30	h_2 3.1 / 1.00, h_0 1.7 / 0.80	1.03 / 1.01	1.23 / 1.06
file of instantaneous samples (1 s, 0.5 Hz bandwidth; white PM + flicker FM, 24 h)	30	h_2 5.6 / 1.10, h_{-1} 1.7 / 0.90	0.86 / 0.90	1.06 / 1.17
OCXO model, 10 ms windows, 4 h	20	h_0 4.6 / 1.68, h_{-1} 1.3 / 1.21	1.14 / 1.90	0.86 / 1.47
OCXO model, 10 ms windows, 15 h	12	h_0 2.9 / 1.15, h_{-1} 1.3 / 1.04	1.57 / 1.92	1.60 / 1.57

- The bootstrap matches the scatter for the terms the data determine; the fit's own errors are conservative

(2–6×, stabilitylib's factor 3), up to 50× for absent terms.

- Non-negative fitting biases terms the data cannot resolve upwards by up to about 1σ: the OCXO's h_{-2} ($1.8 \cdot 10^{-30}$) comes out 3–$5 \cdot 10^{-29}$ in 4–15 h. The replicas then carry that random walk, so offset and drift errors are up to 1.9× too large. In the 12 h slow series, the absent flicker PM takes $1.4 \cdot 10^{-22}$ ($\pm 1.1 \cdot 10^{-22}$) and h_0 comes out 24 % low (0.8σ of one run).
- Fixed on the way: the trend constants took 6.4 GB per process (an OOM of the PC during the first attempt at these runs); now 0.2 GB and 0.2 s, values unchanged to 10^{-12}. The bootstrap's offset error was zero for one build of the fixes (a variable shadowed); the test now requires distinct replica errors.
- Not yet done: the board-side fixes (stream discontinuities, session check) are tested offline against `fake_core` only; not yet uploaded to the board.

2026-10-12 (later): rustle edge cases, dashboard test, long runs, more clocks

- **Board edge cases** (live stream into a rustle session; nothing was lost except where stated):

Case	Result
a second client streaming at the same time	3014 windows in 30 s, none lost
recording start and end	100/s throughout; a new segment at each meter handover
meter log on, then off	a new segment at each switch
a client that read nothing for 100 s	it lost 870 windows (reported, flagged); rustle unaffected
service restart	the new automatic reconnection: "closed", "refused", streaming again after about 16 s, as a new segment
a network stalled for 25 s (frozen TCP relay on the PC; the board has no iptables)	noticed after 10 s, reconnected 0.3 s after the network came back
`SAFE`, then the configuration restored	the stream went on
a meter-window change	the run ended with "The meter window changed from 0.01 s to 0.02 s: start again"

  Reconnection also has an offline test (`test_rustle.py`).

- **Dashboard test** (`tools/rustle/web/tests/ui.mjs`, Playwright, Chromium): all 29 checks pass, with the downloaded recording and the board as sources. It found missing accessible names on the icon buttons, now added.

- **Long runs** (demo OCXO at maximum speed, 1500–2500× real time):
  - 81 days of data (6.98×10^8 samples, store halved 7 times to 1:128): the statistics are continuous across every halving (ADEV at 1 s, 100 s, 10^4 s, 10^5 s and 10^6 s stable to 0.1–0.5 %).
  - h_0 and h_{-1} were recovered within 0.1 %.
  - The server's memory grew by about 7 MB per day of data, to 863 MB. The cause: the model curves' τ grid moved with every analysis, so the kernel cache grew without bound. Fixed with a fixed lattice. A 31-day rerun stayed at 250–375 MB, and the analysis time fell from 0.2–0.4 s to 10–70 ms.

- **More clocks** (coverage, 12–20 seeds each, with the fixes below):

Clock	Window	Length	White FM h_0	Flicker FM h_{-1}	Bootstrap errors vs scatter
hydrogen maser	10 ms	4 h	within 2 %	−14 % (not resolved in 4 h: 1.3σ)	within about 20 %
crystal (with drift)	1 s	72 h	−2.8 %	within 0.6 %	within about 30 %
rubidium	1 ms	1 h	exact	not resolved	right

  Two fixes came out of these runs:
  - **The fit** now runs a second pass with weights from the first fit's model. With measured weights, the maser's flicker FM came out 40 % low.
  - **The demo generator** now simulates the flicker terms 16× finer (was 4×), with slow sections stepped once per window to keep the cost. At 4×, flicker FM was 4 % low at τ_0 and pulled the crystal's white FM 7 % low.

2026-10-12: six hours of MEAS STREAM; rustle with the measured floor

- **Stream** (22:28–04:28, 10 ms windows, model playback of T5's flicker FM, no recording):
  - 2 159 982 windows in 21 600 s (99.998/s): no index jumps, none lost, only the first row flagged.
  - The service CPU was 56 % in all 70 samples, every 5 minutes.
  - No underflow, stall or error lines in the journal.
  - The data are kept on the PC as `tools/recordings/stream-soak-20261011.f64`, with its `.log` (not in git).
- **The fit on real data**: without help, rustle fitted the meter floor as white and flicker PM plus white FM, and missed T5's flicker FM, although the ADEV flattens at 2.9×10^{-15} as expected ($h_{-1} = 0$, reduced χ^2 702). Two changes:
  - the fit's points get a 3 % systematic uncertainty;
  - the measured floor (T0b-long) can be subtracted in the fit.

  Now the floor terms come out 0, as they should.
- **h_{-1} against the configured 5.6×10^{-30}**:

Data	Fitted h_{-1}	Bootstrap error	Reduced χ^2
the 6 h stream	$(3.4 \pm 2.5)\times10^{-30}$	±0.6	7.2
the stopped T5 recording (10.7 h, slow series)	$(6.2 \pm 1.3)\times10^{-30}$	±0.5	1.6

  - stabilitylib's analysis of the same T5 recording, with the same floor, gives $(4.6 \pm 7.2)\times10^{-30}$, which it calls "not resolved".
  - The stream's value is 3.7 bootstrap σ low: the bootstrap does not include the floor recording's own uncertainty (documented in tools/rustle/README.md).
- **Tests**: a floor-subtraction test (`test_rustle.py`) was added; all suites pass.

2026-10-11 (evening): 3.2 on the board; MEAS STREAM; rustle on the live meter

`make upload` (overlay 3.2, build ID 0x2570da4a; the build stage found the export current), then a service restart. This ended the long-tier recording `long-20261010-T5-x10` at 10.7 h of 24 h (marked "service restart"). The service came back on 3.2 with its configuration (T5 flicker-FM playback resumed). No underflow or stall lines in the journal.

- **`MEAS STREAM`** (new, board software only):
  - 60 s from the PC: 5990 windows (99.7/s), none lost, no index jumps, exactly 10 ms apart, only the first row flagged.
  - Service CPU (top, 5 s samples), with model playback running in every case:

Running	Service CPU
playback alone	36 %
with a stream	56 %
recording and a stream	54 %

- **Stream during a recording**:
  - A 12-minute recording (`rustle-check-20261011`) started 30 s into a 12-minute rustle run on the stream. rustle received all 72 000 windows in two segments; the break is where the recording took over the meter, flagged as designed. The recording itself had no gaps.
  - The live ADEV agreed with rustle's analysis of the downloaded recording within 0.2–1 % at $\tau \le 10$ s, and within the confidence intervals beyond. The fitted h_2, h_1 and h_0 agreed within their errors.
  - The fit measured the meter floor: white PM $h_2 \approx 5.6\times10^{-28}$, flicker PM $h_1 \approx 2.3\times10^{-26}$. Twelve minutes do not resolve T5's flicker FM ($h_{-1} = 5.6\times10^{-30}$).
- **Dashboard**: `python -m rustle --board 192.168.0.149` ran live at 1.0× real time (screenshot checked).
- **Tests**:
  - offline tests for the hub (`test_playback.py`) and the TCP stream with `MeterStream` (`test_control.py`);
  - the generic streaming command (`common/tests/test_cli.py`, the demo plugin's `TICKS`);
  - the rustle board source against the board model (`test_rustle.py`).

- **Not tested**: a stream running for an hour or more.

2026-10-11: rustle, the live estimator (host only; no board involved)

`tools/rustle` (milestones 1 and 2: engine and dashboard) checked against references on this PC, with the `base-tools` python; `tests/test_rustle.py` passes (16 tests, plus the slow one).

- **Deviations**:
  - the incremental overlapping ADEV, MDEV and HDEV equal allantools to 10^{-8} (ADEV and HDEV to 10^{-16}), with random batch sizes and accumulators added mid-run;
  - adev1's τ sequences and values agree for every `bins` setting;
  - with the store halved the short τ stay exact, and the long τ differ only statistically.
- **Demo clock against the meter-aware model**: each term's ADEV and MDEV ($4{\cdot}10^5$ samples) agrees within 2 % at short τ, and within the statistics at long τ.
- **Fit and its uncertainties**: the demo OCXO (h_0, h_{-1}, drift) was run 30 times for 4 h and 20 times for 8 h, each with its own seed.
  - The mean fitted h_0 and h_{-1} are within 0.3 % of the truth.
  - The parametric bootstrap reproduces the true scatter within 10 % for h_0, h_{-1}, h_{-2}, offset and drift (8 h; for runs longer than 2^{21} samples it uses two records).
  - The covariance errors are about 3× too large for h_0 (stabilitylib's correlation factor), and right for h_{-1}.
  - At 4 h, the bootstrap's offset error was 3× the scatter: h_{-2} was not resolved, and the fitted value (10× the true one) went into the replicas.
- **Speed**:
  - ingest about 660 kS/s (one thread; four threads measured no faster);
  - a snapshot analysis takes 50–300 ms;
  - the demo runs at a sustained 1000× with 10 ms windows, while the bootstrap runs in the background.
- **Shutdown**: SIGTERM ends the server in under 0.5 s together with its worker processes; after SIGKILL the workers exit by themselves within seconds.
- **Not tested** (at the time; the board stream is in the entry above): the dashboard was checked by screenshots at 1920×1080 and 390×844 in Chromium only, with no horizontal overflow.

2026-10-11: version 3.2 (code-review fixes; built, not yet on the board)

A review of the RTL, the board software and the host tools found the defects below; the core arithmetic (carrier, phase word, noise scaling, playback mirror, flicker sections, PPS arithmetic, model kernels, estimators, ITU-T masks) checked out. The long tier is running on the board with 3.1: 3.2 is built and simulated but not uploaded yet.

- **RTL** (`make sim`: all seven benches pass; tb_noise checks the new rounding and a saturating case, tb_phase a commit and ALIGN on one clock, which failed on the old code):
  - ALIGN was not atomic with the commit: with slew 0, OUT1 stepped for a moment to the new phase plus everything accumulated since the last alignment (the driver also committed and aligned in two writes, about a µs apart). Fixed in the hardware and the driver.
  - a sticky UNDERFLOW (OVERFLOW) set on the clock of a CLEAR_FLAGS was lost: set now wins;
  - noise scaling wrapped instead of saturating for out-of-range raw gains; ties rounded up (+¼ LSB mean): now halves away from zero, mean exactly 0, saturated at $\pm(2^{63}-1)$;
  - the dither had a mean of −1 (two two's-complement fields): now two odd values;
  - the PPS compared n with the φ of the previous carrier sample; a merged catch-up pulse dropped low for a clock and its "late" value could wrap: fixed;
  - PLAY while playing latched a spurious segment; CTRL bits 8–11 needed byte strobe 0 and a FIFO_HI write without strobes pushed: fixed;
  - the reseed strobe's fan-out (five generators) was the critical path: registered per generator.
- **Build** 0x2570da4a: timing met, worst setup slack +0.189 ns (snapshot strobe fan-out), hold +0.018 ns; routed DRC: no errors; 6267 LUTs (36 %), 9112 registers (26 %), 16.5 BRAM tiles. **`verify_build.py --design atomic_clock`** passed all eight stages, reproducing 0x2570da4a.
- **Board software** (`make test`: 89 tests, 27 skipped with the system python): `pps_status()` used only the static phase for `time_offset_s`/`simulated_error_s` (now φ_s + φ_d + φ_w); `align_utc()` now puts the epoch on a multiple of 25 samples (OUT2's zero crossing; it could be up to 96 ns off); the 32-bit meter window index is extended from the sample counter (it wrapped after 14 min at the shortest window); `measure()` and the worker's meter log unwrap window by window with the expected advance; a recording could stay "requested" forever (REC STOP before the worker started it, or a failed start); playback: rate + ν kept within 64 bits, sine PM and replays validated against the 10^{-3} frequency limit, non-finite replay values rejected, a

failing restart halts the FIFO instead of holding the last rate, mirror compared modulo 2^{96}; PPS and meter logs and the NTP check no longer write or wait on the worker's thread; recording write errors reported; a playing replay record cannot be replaced; meter logging restored after a recording.

- **Host tools** (`tests/test_stability_tools.py`: 26 tests pass): the measured floor is now per series (fast and slow differ by up to 100× in MDEV for white PM); the frequency- offset and step check lost a rank on multi-day records (now conditioned); the expected white-PM spectrum lacked the meter's ×1.5; the simulator held y over each slow-series interval (flicker and random-walk FM about 10 % high at τ0; now from an oversampled series); Totdev/ThéoBR decimate by picking (block averaging biased white-PM-dominated data low by √k); the model's tail integral was 1.5–2.3 % low for flicker PM (now within 0.1 % of brute force); smaller fixes (fit curve with the sine, per-segment drift on the series page, drift column per deviation kind, external FM cutoffs, mask TDEV range, noise type with a dominant floor, results labels). The manuals' figures and tables were regenerated: test results changed at most in the second decimal (T2-x10 1.07 → 1.06).

2026-10-11: loopback floor over one day (T0b-long, long tier)

Conditions. Loopback OUT1 → IN1, OUT2 → IN2 as in the quick tier, 0.5 V, no noise, no dither, the board's scheduled jobs disabled. Recording `long-20261010-T0b-long`, 2026-10-10 09:17 to 2026-10-11 09:17 UTC: 24 h, no lost windows, no playback resyncs.

Result. ADEV $7.0{\cdot}10^{-12}$ at 10 ms, $1.1{\cdot}10^{-13}$ at 1 s, $2.4{\cdot}10^{-15}$ at 100 s, $2.3{\cdot}10^{-16}$ at 1000 s, $5.5{\cdot}10^{-17}$ at 8640 s (a tenth of the record); ThéoBR $5.6{\cdot}10^{-17}$ at 10^4 s and $1.5{\cdot}10^{-17}$ at $6.5{\cdot}10^4$ s (18 h), Totdev alike. Fit: white PM h2 = $3.7{\cdot}10^{-28}$ ($2.6{\cdot}10^{-14}$ s per 10 ms window), flicker PM h1 = $3.0{\cdot}10^{-26}$, white FM h0 = $1.9{\cdot}10^{-28}$; no flicker or random-walk FM. The time error stayed within 2.0 ps peak to peak over the day (a constant frequency removed); the Zynq die was at 71.5–73.3 °C, with no correlation (0.04) between it and the time error. The 2-hour T0b of the quick tier agrees where both exist (within 10 % to 100 s); its fit, extrapolated, predicted $1.8{\cdot}10^{-16}$ at 10^4 s, three times the measured $5.6{\cdot}10^{-17}$.

Manuals. The testing manual (floor table and figure, test levels, models figures) and the user's manual (catalogue figure) now use this floor (`manual_figures.py --floor`); the fit lies up to about twice above the measured floor beyond an hour, so levels scaled to it are conservative there.

2026-10-11: adev1-compatible table (host software)

- `tools/adev1.py`: Tom Van Baak's `adev1` interface and output in standard-library Python. Against `adev1.c` (09-Jun-2000) compiled with gcc on Linux: byte-identical output on 20 000 random-walk samples for bins 1, 2, 3, 4, 5, 10, 29, 50 and 0, the same errors. On HP 53132A lines with digit-group commas ("1.239,7 us") the compiled C program loses the unit suffix (its overlapping `strcpy` is undefined behaviour; values come out 10^6 too large); `adev1.py` reads them as documented. Normal and overlapping deviations agree with allantools' `adev`/`oadev` to 10^{-9}; 360 000 samples take 1.3 s.
- Tests: `tests/test_adev1.py` (6, in `make test`) and one in `test_stability_tools.py`.

2026-10-10: further estimators in the stability tools (host software)

- **Added** (`tools/stabilitylib/extras.py`, `masks.py`, `external.py`): Totdev and ThéoBR with SP 1065's bias corrections and degrees of freedom; MTIE and TIE rms with ITU-T G.811 and G.8262 masks (values transcribed from the recommendations); `analyze-file` for other instruments' phase or frequency files; the three-cornered hat (`stability.py hat`); the dynamic ADEV; logarithmic aging (`--drift log`). Testing manual section "Further estimators and tools". No change on the board.
- **Checked**: on simulated white, flicker and random-walk FM, bias-corrected Totdev and ThéoBR agree with the ADEV formulas within their scatter at long τ (white FM: 0.96 and 0.89 of the formula, mean over 6 records, the largest τ having 2–3 degrees of freedom); MTIE identical to allantools' on random data and 20× faster (the T3-x10 report: 84 s → 27 s); the hat recovers three clocks within 15 %; log aging recovers A and B within 5 % and 10 %; an external counter file (1 day, white PM, white and flicker FM) gives h2 1.00 and h_1 0.94 of the setting.
- Not provided: modified and Hadamard total deviations (allantools' `mtotdev` is marked preliminary: Mtot/MVAR = 0.77 on white FM where SP 1065 gives 1.27).
- **Tests**: `tests/test_stability_tools.py` 25 tests (10 new) pass with the base-tools python; `make test` passes.

2026-10-10: theory and user's manuals (documentation)

- `docs/manual/theory_manual.pdf` (18 pages) and `user_manual.pdf` (11 pages); all three manuals share `references.bib`. Their code excerpts are checked against the sources by `tests/test_manuals.py`, which also builds both manuals; `make test` (69 tests, 16 skipped with the system python) and `tests/test_stability_tools.py` (base-tools) pass.
- Not checked on a fresh board: the user's manual's setup chapter follows `common/SETUP.md` (image: dspsandbox/Pynq-Redpitaya-125).

2026-10-10: oscillator model catalogue (software, 3.1)

- **Catalogue**: 11 oscillator models with typical values (XO, MEMS, TCXO, OCXO, double-oven OCXO, CSAC, rubidium, caesium standard and high performance, active H maser, GPS-disciplined OCXO), tier `models` (3 days each, floor M0 first, 36 days); the long tier's floor-scaled clock shapes are replaced by it. Values, settings and expected ADEV/MDEV in the testing manual, "Oscillator models". Not yet run on the board.
- **GPSDO**: a generated replay record (type-2 loop, 1000 s, GPS 10 ns rms; ADEV $1.1{\cdot}10^{-12}$ at 1 s, $1.3{\cdot}10^{-12}$ at 1000 s, $1.8{\cdot}10^{-13}$ at 10^4 s). Uploaded over TCP (`REPLAY PUT/DONE`, 6.0 MB in 7 s) and played on the board: no underflow, the meter following, the service at ~28 % of a core. Starting a replay stalls the worker while it reads the record: 3.8 s for 3 days at 1 s (numpy parsing), absorbed before a run starts.
- **Analysis**: drift fitted together with a configured sine (a small drift under a daily cycle over 2.5 days: 1.00 of the setting, the quadratic fit alone 10×); a sine's amplitude and the drift judged only where the run covers two periods and the noise allows; replay runs' expected curves include the record's own deviations.
- **Tests**: `make test` (replay upload, play and refusals over TCP; the replay readers), `tests/test_stability_tools.py` (the models tier end to end with a replay against a fake board, the joint drift fit, generated tables and the manual build) pass.

2026-10-09: stability recordings, analysis tools and the quick tier on the board (software on 3.1, build 0xe2be7288)

Conditions. Loopback OUT1 → IN1, OUT2 → IN2 with two equal cables (T0a: OUT2 through a BNC T to both inputs), 0.5 V, Zynq die about 71 °C, the board's scheduled maintenance jobs disabled (`common/SETUP.md`). Noise levels scaled to the T0b floor; dither on in every test except the floors. Series `quick-20261007` (`tools/run_tests.py --tier quick`).

Floor. T0b (2 h): ADEV $7.0{\cdot}10^{-12}$ at 10 ms, $1.2{\cdot}10^{-13}$ at 1 s, $2.3{\cdot}10^{-15}$ at 100 s, still falling at 720 s (fitted: white PM $2.9{\cdot}10^{-14}$ s per 10 ms window, flicker PM h1 = $2.9{\cdot}10^{-26}$, white FM h0 = $6.3{\cdot}10^{-28}$; no flicker-FM plateau within 2 h). T0a (meter only) is close to T0b.

Tests (fitted/configured; "points": the expected ADEV inside the measured 68 % intervals widened by 1.3). All 11 runs pass.

Run	Setting	Result
T1-x10, T1-x100	white PM h2 = $1.87{\cdot}10^{-25}$, $1.26{\cdot}10^{-23}$ (the hardware limit with margin)	1.02, 1.03; 38/38 points each
T2-x10, T2-x100	flicker PM h1 = $3.48{\cdot}10^{-24}$, $3.48{\cdot}10^{-22}$	1.07, 1.00; 42/42 each
T3-x10, T3-x100	white FM h0 = $7{\cdot}10^{-25}$, $7{\cdot}10^{-23}$	0.98, 0.99; 37/37, 42/42
T4	frequency offset $1{\cdot}10^{-9}$, phase step at 960 s	offset error $1.6{\cdot}10^{-15}$; step 934.1 ps fitted, 937.5 ps set
T8-x10, T8-x100	sinusoidal PM 0.93 ps and 9.3 ps at 0.05 Hz	amplitude 1.054, 1.021; 14/14, 25/25

Properties of the system established by these measurements:

- The phase meter reads the FPGA's broadband white PM with 1.5 times the variance of the window-averaged phase (the demodulator's 2f product; simulation 1.54, T1 1.52). The analysis model includes it (`METER_WPM_GAIN`).
- Without the dither, a moving phase picks up extra short-term error from the 14-bit rounding: ADEV up to 1.5× the expected value for white FM (T3-x10) and 3× for the sinusoidal PM (T8-x100) at 0.01–2 s; with the dither 1.00 (manual, Figure "dither").

- The FPGA's white-PM generator is limited to h2 ≤ $1.6 \cdot 10^{-23}$ (0.05 cycle per sample).
- Board load: model playback about 36 % of one CPU core, with a recording at 100 windows/s about 50 %; no lost windows and no playback underflow during the series; service stalls of up to 2.4 s are absorbed by the 3.5 s playback FIFO and the 5 s meter FIFO. (Before this version, model playback was CPU-bound and could underflow.)

Offline tests. `make test`: 63 atomic_clock tests (14 skipped with the system python: the stability-tool tests need allantools, the GUI test ipywidgets) — recordings at two rates with temperature, gaps, resync flags and restart, REC commands and download over TCP, the trajectory blocks and resume; framework 121 tests including `read_burst`/`write_burst`. `tests/test_stability_tools.py` (base-tools python): 13 tests — the model's asymptotes and ratios, the fit recovering each term on synthetic recordings, verdicts and exact checks, the report and CLI, the runner end to end against a fake board (floor scaling, relative phase steps, caps, refused settings, resume), the manual's generated numbers and its LaTeX build.

Not yet done: the long tier (flicker FM, random-walk FM, drift, clock shapes) and the floor beyond 2 h.

2026-10-06: version 3.1 (PPS time of day; build 0xe2be7288)

- **`make sim`** passed (seven benches). `tb_pps` adds arming: the reference edge lands on the armed epoch with the same fixed offset as on the free-running grid, counts restart, the simulated pulse keeps its −12.5·φ offset, and an epoch 5.5 seconds in the past is caught up (6 seconds counted).
- **`make test`**: 45 tests passed (GUI with ipywidgets): `align_utc()` with a fake UTC clock in integer nanoseconds (the epoch on the UTC second ≥ 1.5 s ahead, the reference count turning 1 at that second, board error 0, simulated error +5 µs for φ = 50 cycles, invalid after a reload); the worker aligning by itself with `pps.utc` (also after a reload) and logging the board error for an hour of a board clock 10 ppm fast: fitted frequency offset $1 \cdot 10^{-5}$ within 10^{-8}; PPS UTC and PPS JSON commands, client, CLI, GUI button; a robust PPS count check in `scripts/test_hw.py`.
- **Build** 0xe2be7288: timing met, setup slack +0.359 ns, hold +0.008 ns; DRC warnings only; 6116 LUTs (35 %), 9039 registers (26 %), 16.5 BRAM tiles, 12 DSP. **`verify_build.py`** passed all eight stages, reproducing 0xe2be7288.
- **Not yet done:** anything on the board with 3.0/3.1; the board crystal's offset against UTC has not been measured.

2026-10-06: version 3.0 (PPS outputs, phase meter, unwrapped phase; build 0x4c4f8b82)

- **`make sim`** passed (seven benches). New: `tb_pps` (shortened 1000-sample second: both pulses aligned at φ = 0, period and width; φ = +2.5 cycles → 31 samples early with remainder 0.25; φ = −1 cycle → 13 samples late with remainder 0.5; no double pulse when φ wobbles ±0.1 cycle across a boundary; a 3-second jump counted 3 times); `tb_meter` (synthetic sines: amplitude within $2 \cdot 10^{-4}$, phase within $2–3 \cdot 10^{-5}$ cycle, exact DC rejection, window index and length, FIFO pop/overflow/clear, restart on a new window length); `tb_phase` with 96-bit unwrapped φ_s/origin and whole-cycle targets; `tb_system` with a digital loopback into the meter (OUT1 − OUT2 measured at 0.25 cycle within 10^{-4}) and the new registers. Mutations (12φ instead of 12.5φ in the PPS; the uncorrected cosine table) make `tb_pps` and `tb_meter` fail.
- **`make test`**: 43 tests passed (GUI with ipywidgets): unwrapped phase words, PPS settings and status, meter readout and `measure()` in model loopback, MEAS/PPS commands, client and CLI, meter logging by the worker (unwrapped, growing under a frequency offset), the deterministic checks of `scripts/test_hw.py` against the core model; framework: `design(xdc)` checked for every design (150 tests).
- **Build** 0x4c4f8b82 from a regenerated project (new block-design ports): timing met, setup slack +0.656 ns, hold +0.024 ns; DRC warnings only; PPS pins H16 (DIO1_P) and J18 (DIO2_P), LVCMOS33, bank 35. 6001 LUTs (34 %), 8961 registers (25 %), 16.5 BRAM tiles (27.5 %), 12 DSP48E1 (15 %). **`verify_build.py`** passed all eight stages (with the design constraints in the source copy), reproducing 0x4c4f8b82.
- **Not yet done:** anything on the board with 3.0 (`make install-service`, `make test-hw` in loopback, PPS on a scope or counter, the meter's noise floor).

2026-10-06: version 2.0 on the board (build 0x4bd7f0e3)

- Driver attached as root without reloading. In `reference` the phase path read exactly zero (static phase, φ_d, φ_w, origin; PHASE, RATE and noise gains 0, no dither).
- Phase steps of −0.42 and +0.25 cycle (slew 0) changed the OUT1–OUT2 delay on a scope by the programmed amounts (−42 ns and +25 ns, within about 1 ns).

- Simultaneous DC steps on both outputs (sines off, 0 ↔ +0.4 V): OUT2 about 3.2 ns earlier than OUT1 (strong ringing, approximate), consistent with the DAC's half-clock channel interleave (4 ns).
- Simulation of the synthesized `ac_core` netlist (Vivado functional netlist, out of context): OUT1 and OUT2 identical sample for sample at power-up.

2026-10-05: version 2.0, alignment (build 0x4bd7f0e3)

- **New:** setting a phase aligns OUT1. An ALIGN strobe (CTRL bit 8) latches origin = φ_d + φ_w; the static phase slews to PHASE − origin, so the phase accumulated by the frequency offset, noise and playback is absorbed. The driver aligns when a patch sets `phase` and for presets, reset and SAFE.
- **`make sim`** passed (all five benches). New checks: `tb_phase` aligns with left-over φ_d and φ_w, in step mode and with slew, total phase exactly equal to the target and φ_d unchanged; `tb_system` through the registers (rate, FIFO, noise, then "reference" + ALIGN: φ = 0, S_ORIGIN read back, OUT1 == OUT2 sample for sample).
- **`make test`**: 37 tests passed (GUI with ipywidgets), including 42 s at y = 10^{-9} followed by `reference` giving OUT1 exactly in phase (core model).
- **Build** 0x4bd7f0e3: timing met, setup slack +0.376 ns, hold +0.032 ns; DRC warnings only (as before); 4372 LUTs, 6260 registers, 12.5 BRAM tiles, 8 DSP. **`verify_build.py`** passed all eight stages, reproducing 0x4bd7f0e3.

2026-10-05: version 2.0 (smooth phase control, frequency offset, phase noise and playback)

- **`make sim`** passed: `tb_sysid`, `tb_sine` (unchanged), and:
  - `tb_noise`: 200 000 variates: mean 626 (2.6 standard errors), standard deviation within 0.1 % of 107 019, kurtosis 2.859 (expected 2.85), lag-1 and PM–FM correlation below 0.0012; the dither within $\pm 2^{19}$ with the triangular standard deviation; the scaling round(n·gain·2^exp/2^{16}) bit-exact for four gain/exponent pairs, gain 0 exactly 0; the same seed reproducing the sequence, another seed not.
  - `tb_phase`: step mode; slew rate limits in both directions and across the wrap (shorter way), landing exactly; φ_d checked clock by clock; 40 FIFO entries consumed in order on the segment grid, hold-last and underflow, refill taken on the same grid, stop, overflow at 4096 and clear; the segment recurrence φ_d(i+1) = φ_d(i) + L·(rate + ν_i) at 47 segment starts; periodic snapshots consistent (segment fields and the live φ_d formula); the φ sum with noise inputs.
  - `tb_system`: the 1.0 checks with the new latency (13) and power-up values (including the default slew), plus through the AXI registers: the slew rate, rate + FIFO playback with snapshot readback, white PM/FM noise with dither on OUT1 while OUT2 stays bit-exact, and zeroing back to φ = 0.
- **`make test`**: 36 tests passed (GUI with ipywidgets installed). New: the 2.0 driver registers (frequency offset and slew words, white-noise gain encoding to within 10^{-8}, segment length, seed, dither), FIFO/snapshot through a register model of the core (`tests/fake_core.py`), and the playback worker: Allan deviation of flicker FM, random-walk FM and flicker PM within 10 % of the formulas at 10–100 ms over 60 s records (an error of 2× in h is detected); drift and sinusoidal PM exact; seeded reproducibility and checkpoint resume; replay interpolation, looping and holding; exact mirroring (every predicted segment-start phase equal to the model's, within L/2 of the target) through starts, a 3 s stall with underflow and resynchronization without a phase jump, a frequency-offset change, a model change, off/on and SAFE. Common: the new plugin `WORKER` hook (start/stop on LOAD, no worker for other designs, a broken worker not blocking the design); all 150 framework tests pass.
- **Build** 0xfdfd5c36: timing met, worst setup slack +0.605 ns, hold +0.012 ns (an earlier build had +0.24 ns on the FIFO-full and slew-compare paths; both are now registered). Routed DRC: no errors (warnings DPIP-1/DPOP-2 on DSP pipelining, RTSTAT-10 on unused SmartConnect reset nets). 4242 LUTs (24 %), 6073 registers (17 %), 12.5 BRAM tiles (21 %: sine tables, carrier table, FIFO), 8 DSP48E1 (10 %).
- **`verify_build.py --design atomic_clock`** passed all eight stages; the source-only rebuild reproduced build ID 0xfdfd5c36.
- **Not yet done** (phase steps were later checked on the board, see 2026-10-06): the service's worker on the board (the service there still runs an older rpfpga; CPU load and refill timing at 1 kHz have not been measured), slew and frequency offset on a phase comparator, and the noise levels and Allan deviation measured from OUT1 against OUT2.

2026-10-05: version 1.0 (the MVP: 10 MHz on both outputs, stepped phase)

- **On the board** (scope, reported by the user): both outputs showed the 10 MHz sine, with a relative delay of 25 ns between them as expected for the quarter-cycle phase. Frequency, amplitude, spectrum and the sign

of the delay were not recorded.

- **`make sim`** passed: `tb_sysid`; `tb_sine` (112 282 phases: every table point, every midpoint, the last fraction before each next point, and 100 000 pseudo-random phases; worst error 0.999 LSB of 2^{17} against $(2^{17} - 1)\cdot\sin(2\pi p/2^{28})$); `tb_system` (core with the vendor DAC interface: power-up sine on both outputs, identical, amplitude in range; the carrier phase repeating every 25 samples with steps of 2/25 cycle to 1 LSB of 2^{-64}; the output stage bit-exact against its integer model and within 2.6 codes of a real-valued sine for φ = 0, ¼, $\pm 10^{-6}$ and ½ cycle; the 64-bit phase not changing before the commit; amplitude, offset, sine off, saturation at both limits; ID, VERSION, LATENCY = 10 and the sample counter; the DAC's channel order and code mapping; reset restoring the power-up configuration).
- A deliberate change of the output stage's rounding constant (the `verify_build.json` edit) made `tb_system` fail with 490 errors, so the bench detects it.
- **`make test`**: 17 tests passed (the generated sine table and the carrier table against their formulas, driver, protocol, client over localhost, command line, model loading); the GUI test passed with ipywidgets installed.
- **Build** (`make`, Vivado 2025.1): build ID **0x0d248fe2**. Routed timing met at 125 MHz: worst setup slack +0.878 ns, worst hold slack +0.026 ns, no failing endpoints. Routed DRC: no errors (warnings DPIP-1 and DPOP-2: DSP pipelining suggestions). Utilization of the whole design (with the PS interconnect and converter interfaces): 1034 LUTs (5.9 %), 1687 registers (4.8 %), 5 RAMB36 (8.3 %: two per sine table, one for the carrier table), 4 DSP48E1 (5 %).
- **`verify_build.py --design atomic_clock`** passed all eight stages (recipe build, unchanged reuse, missing export, vendor HDL edit, the design RTL edit, failed build blocking upload, clean, regeneration from the saved project); the source-only rebuild reproduced build ID 0x0d248fe2.
- **Not yet done:** quantitative measurements on the board (frequency, amplitude, spectrum, harmonics, OUT1–OUT2 phase and skew).