atomic_clock: how the clock simulator works

Theory manual

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

Red Pitaya `atomic_clock` design, version 3.2
October 28, 2026

Abstract

The `atomic_clock` design turns a Red Pitaya STEMlab 125-14 into a 10 MHz source whose phase behaves like that of a real clock: it can carry a frequency offset, a drift, power-law phase noise from white phase noise to random-walk frequency noise, a sinusoidal modulation or a recorded time-error series, with exact arithmetic from one 8 ns sample to years. This manual explains, at the level of the actual arithmetic, how that signal is made: the exact carrier, the 96-bit phase word, the noise generators in the FPGA, the noise played back by the board's software, the conversion to a sine and to the DAC, and the 1 PPS outputs. A worked guide to units connects Allan deviation, phase-noise spectra and time error, and explains how to read OCXO and chip-scale atomic-clock specifications. Key Verilog and Python excerpts are quoted from the sources, the main results are derived, and the standard techniques are named with references.

Contents

About this manual **2**

1 Foundations in one page **3**

2 Units, phase noise, and reading oscillator specifications **4**
2.1 Four coordinates for the same fluctuating clock . . . 4
2.2 What the dimensionless Allan deviation measures . . . 4
2.3 Spectral densities and the meaning of dBc/Hz . . . 5
2.4 Five noise types: units, slopes and conversions . . . 6
2.5 Why the conversion is an integral . . . 8
2.6 Worked example: a sparse phase-noise table . . . 9
2.7 Reading real OCXO and CSAC data sheets . . . 11
2.8 A conditional holdover calculation . . . 14
2.9 Comparing instruments, and using these units on this board . . . 16

3 Architecture **17**

4 The exact carrier **18**

5 The phase word **19**
5.1 Format and resolution . . . 19
5.2 The frequency offset: an exact integer rate . . . 19
5.3 The static phase and its slew limit . . . 20
5.4 The segment FIFO . . . 22
5.5 A coherent snapshot . . . 23

6 Hardware white noise **23**
6.1 Uniform random numbers: xoroshiro128+ . . . 23
6.2 Gaussian-like values: a sum of eight uniforms . . . 24
6.3 Scaling to phase units . . . 25
6.4 White PM and white FM: from h to σ . . . 25

7 Played-back noise **26**
7.1 Exact mirroring: rates from a target trajectory . . . 26
7.2 Flicker noise as a sum of first-order sections . . . 27
7.3 Random walk, drift, sinusoidal modulation and replay . . . 29

8 From phase to sine **30**

9 The output stage **30**
9.1 Scaling and rounding to the DAC . . . 30
9.2 What 14 bits can resolve . . . 31
9.3 Dither . . . 31
9.4 The DAC's hold . . . 32

10 PPS outputs and time of day **32**
10.1 When does the simulated clock's second end? . . . 32
10.2 Time of day . . . 33

11 Numbers and limits **33**

About this manual

Three manuals describe `atomic_clock`:

User's manual (`user_manual.pdf`): setting the board up, operating it, commands and recipes.

Theory manual (this document): how the *generated signal* is made: everything between a setting and the voltage at OUT1, OUT2 and the PPS pins. You need it to know what exactly the simulator produces, with which resolution and which limits. The units section also connects oscillator specifications to the noise model and its settings.

Testing manual (`testing_manual.pdf`): the *measurement* side: the board's phase meter, stability recordings, the Allan-deviation statistics, the floor of the measurement chain, the tests with their results, and the live estimator `rustle`.

The implementation of the phase meter on IN1/IN2 is described in the testing manual and in `docs/THEORY.md`; here we discuss its filtering only where it affects a conversion or comparison. Register addresses are in `docs/REGISTERS.md`, the commands in `docs/PROTOCOL.md`. File paths are relative to the `atomic_clock/` directory of the repository.

Notation. $f_s = 125\,\text{MHz}$ is the sample clock, $f_0 = 10\,\text{MHz}$ the carrier, n the sample index ($t = n/f_s$). In this manual φ is OUT1's phase offset in *cycles* of the carrier (the unit of the hardware); the phase in radians of IEEE Std 1139 is $2\pi\varphi$, and the time error is $x = \varphi/f_0$. "LSB" without a qualifier means 2^{-64} cycle, the unit of the phase word. round rounds to the nearest integer, frac is the fractional part.

1 Foundations in one page

A real oscillator's output is $V(t) = V_0\,[1 + a(t)]\sin[2\pi f_0 t + 2\pi\varphi(t)]$. Following IEEE Std 1139 [1] and the classic papers [2, 3], the random phase is described by

$$x(t) = \frac{\varphi(t)}{f_0}\ \text{(time error, s)}, \qquad y(t) = \frac{\mathrm{d}x}{\mathrm{d}t}\ \text{(fractional frequency)}, \tag{1}$$

and their one-sided power spectral densities. The *power-law model* writes

$$S_y(f) = \sum_{\alpha=-2}^{2} h_\alpha f^\alpha, \qquad S_\phi(f) = \left(\frac{f_0}{f}\right)^2 S_y(f)\ [\mathrm{rad}^2/\mathrm{Hz}], \qquad \mathcal{L}(f) = \tfrac{1}{2}S_\phi(f), \tag{2}$$

with the five terms white PM (h_2), flicker PM (h_1), white FM (h_0), flicker FM (h_{-1}) and random-walk FM (h_{-2}). Their Allan deviations are the well-known asymptotes ($\sigma_y^2 = h_0/2\tau$ for white FM, $2\ln 2\, h_{-1}$ for flicker FM, $(2\pi)^2 h_{-2}\tau/6$ for random-walk FM, and so on [3, 4]); the testing manual (Sections "Definitions" and "Reading Allan-deviation plots") explains them, together with the modified Allan, Hadamard and time deviations, and how to read the plots. Reviews for further reading: [5, 6, 7].

The simulator realizes each term of the model with a mechanism suited to its time scales (Table 1). Terms that are white in time need no memory and are generated in the FPGA anew every 8 ns sample; terms with long memory (flicker, random walk, drift) need state that lives for days and floating-point filters, and are computed by the board's CPU and fed to the FPGA as a sequence of rates.

term	setting	mechanism	offset frequencies
white PM, h_2	`noise.h2`	FPGA: Gaussian-like value added to φ every sample (§6)	up to $f_s/2 = 62.5$ MHz
flicker PM, h_1	`noise.h1`	software: sum of first-order sections, played back (§7.2)	10^{-8} Hz … $f_{seg}/2$
white FM, h_0	`noise.h0`	FPGA: random walk of φ, a new step every sample (§6)	up to 62.5 MHz
flicker FM, h_{-1}	`noise.h_1`	software: sections, integrated (§7.2)	10^{-8} Hz … $f_{seg}/2$
random-walk FM, h_{-2}	`noise.h_2`	software: integrated random walk of y (§7.3)	below $f_{seg}/2$
frequency offset	`frequency_offset`	FPGA: exact integer rate (§5.2)	DC
drift, sine PM	`noise.drift_per_day`, …	software: explicit functions of time (§7.3)	below $f_{seg}/2$
recorded $x(t)$	`playback.replay`	software: interpolated record (§7.3)	below $f_{seg}/2$

Table 1: Which part of the simulator makes which term. f_{seg} is the playback's segment rate, 1000 per second by default.

Relative to what. Everything is generated from the board's own 125 MHz oscillator, so "exact" always means exact relative to that clock. OUT2 is an unmodulated 10 MHz from the same clock: the difference OUT1 − OUT2 contains the programmed $\varphi(t)$ and (to first order) none of the board oscillator's own noise, which has not been measured.

2 Units, phase noise, and reading oscillator specifications

A clock data sheet may promise a fractional stability of 10^{-11}, a phase-noise level of $-140\,\mathrm{dBc/Hz}$, an accuracy of 0.05 ppb, and a time error of a few microseconds after a day. These numbers answer different questions. The first needs an averaging time and a statistic, the second an offset frequency and a carrier, the third a reference and operating conditions, and the fourth an initial state and a prediction model. Comparing the exponents alone is meaningless.

The aim of this section is to make those specifications usable: to convert quantities that describe the same thing, and to identify the additional assumptions needed to connect quantities that describe different things. The conventions follow IEEE 1139, NIST SP 1065 and the modern conversion chart of Rubiola and Vernotte [1, 4, 8]. All numerical examples use a 10 MHz carrier unless another carrier is stated. Analytical illustrations are not measurements of this board or of the commercial oscillators discussed below.

2.1 Four coordinates for the same fluctuating clock

Write the signal as

$$V(t) = V_0[1 + a(t)]\sin[2\pi f_0 t + \phi(t)], \qquad \phi(t) = 2\pi\varphi(t). \tag{3}$$

Here a is fractional amplitude fluctuation, φ is phase offset in *cycles*, and ϕ is phase offset in *radians*. Both phases are unwrapped. An integer number of cycles disappears from a sine but matters to a clock counting elapsed time. The instantaneous frequency $\nu(t)$, time error $x(t)$ and fractional frequency $y(t)$ obey

$$\boxed{x = \frac{\varphi}{f_0} = \frac{\phi}{2\pi f_0}, \qquad y = \frac{\mathrm{d}x}{\mathrm{d}t} = \frac{\nu - f_0}{f_0}, \qquad \Delta\nu = f_0 y.} \tag{4}$$

x is in seconds and y is dimensionless. Positive x means that the clock *leads*: the corresponding edge occurs earlier, with edge-time displacement $-x$ for a constant offset. This distinction fixes the signs of cable-delay and PPS measurements. In this section f_0 is the carrier, f is a positive Fourier offset, and τ is an averaging time; they are not interchangeable frequencies or times.

At 10 MHz, one cycle is 100 ns. A 1 ps phase displacement is 10^{-5} cycle, or $2\pi \times 10^{-5}$ rad. A fractional frequency $y = 10^{-10}$ means a frequency offset of 1 mHz. If constant, that offset adds 8.64 μs of clock lead per day. The conversion to seconds depends on the carrier for phase in cycles or radians; the integration $x(t) - x(0) = \int_0^t y(u)\,\mathrm{d}u$ does not.

Parts per million and billion. $1\,\mathrm{ppm} = 10^{-6}$, $1\,\mathrm{ppb} = 10^{-9}$, and $1\,\% = 10^{-2}$. Thus 0.05 ppb means 5×10^{-11}, not 5×10^{-10}. A tolerance written "±0.05 ppb" is a two-sided interval; it is not automatically a standard deviation. Neither a percentage nor a ppb value says whether it describes accuracy, variation over temperature, or random instability. Read the row heading and its conditions before doing arithmetic.

2.2 What the dimensionless Allan deviation measures

For an ideal contiguous rectangular frequency average,

$$\bar{y}(t;\tau) = \frac{x(t+\tau) - x(t)}{\tau}, \qquad \sigma_y^2(\tau) = \frac{1}{2}\,\mathrm{E}\big[(\bar{y}(t+\tau;\tau) - \bar{y}(t;\tau))^2\big]. \tag{5}$$

Equivalently, Allan variance is the mean squared second difference of x, divided by $2\tau^2$. Its square root, ADEV, is dimensionless. The same statistic in hertz is $\sigma_\nu(\tau) = f_0\sigma_y(\tau)$. For example, $\sigma_y(1\,\mathrm{s}) = 10^{-11}$ corresponds to 100 μHz Allan deviation at 10 MHz. The RMS difference between adjacent 1 s average frequencies is $\sqrt{2} \times 100\,\mu\mathrm{Hz}$, because of the factor 1/2 in the definition.

This is a measure of *changes between averages*, not a bound on instantaneous frequency and not generally the standard error of a frequency estimate. Add a constant y_0 to every average: the

difference is unchanged. An oscillator can have excellent ADEV while running steadily fast by a large amount. A constant phase offset also cancels. A linear frequency drift $y(t) = Dt$ does not: it contributes

$$\sigma^2_{y,\mathrm{drift}}(\tau) = D^2\tau^2/2. \tag{6}$$

Drift removal therefore changes a reported ADEV and must be stated.

Overlapping ADEV uses additional overlapping second differences to estimate the same underlying variance. It improves statistical use of a record; it does not change its units or make the clock physically quieter. Modified Allan deviation (MDEV) adds averaging and has a different frequency weighting. Hadamard deviation (HDEV) uses a third phase difference and rejects ideal linear frequency drift. These statistics must be named when comparing results. At the longest averaging times, a finite record contains few independent pieces of information: confidence intervals, record length and the treatment of drift matter alongside the plotted central value [4, 9].

2.3 Spectral densities and the meaning of dBc/Hz

A power spectral density (PSD) apportions variance over Fourier frequency. We use *one-sided* spectra: for a stationary fluctuation z, $\mathrm{var}(z) = \int_0^\infty S_z(f)\,\mathrm{d}f$ when that integral exists. Its amplitude spectral density (ASD) is $\sqrt{S_z}$, not S_z. Power-law clock phase processes need not have a finite total phase variance; their properly weighted Allan integrals can still exist.

quantity	unit	one-sided PSD unit	ASD unit
phase φ	cycle	$\mathrm{cycle}^2/\mathrm{Hz}$	$\mathrm{cycle}/\sqrt{\mathrm{Hz}}$
phase ϕ	rad	$\mathrm{rad}^2/\mathrm{Hz}$	$\mathrm{rad}/\sqrt{\mathrm{Hz}}$
time error x	s	s^2/Hz	$\mathrm{s}/\sqrt{\mathrm{Hz}}$
fractional frequency y	1	Hz^{-1}	$\mathrm{Hz}^{-1/2}$
frequency error $\Delta\nu$	Hz	$\mathrm{Hz}^2/\mathrm{Hz}$	$\mathrm{Hz}/\sqrt{\mathrm{Hz}}$
ADEV $\sigma_y(\tau)$	1	a time-domain statistic, not a spectral density	

Table 2: Keep the square and the "per hertz." Radians are dimensionless in SI, but retaining rad^2 identifies which phase convention was used.

The conversion dictionary follows from Eq. 4 and the Fourier differentiation factor $i2\pi f$:

$$\begin{aligned} S_\phi &= (2\pi)^2 S_\varphi = (2\pi f_0)^2 S_x, \\ S_y &= (2\pi f)^2 S_x = (f/f_0)^2 S_\phi, \qquad S_{\Delta\nu} = f_0^2 S_y = f^2 S_\phi. \end{aligned} \tag{7}$$

The 2π factors cancel between differentiation and conversion from radian phase to fractional frequency. Inserting another 2π into $S_y = (f/f_0)^2 S_\phi$ is a common normalization error.

The phase-noise quantity. The modern definition is the *linear* quantity

$$\mathcal{L}(f) = \tfrac{1}{2} S_\phi(f), \qquad L_{\mathrm{dB}}(f) = 10\log_{10}\left[\frac{\mathcal{L}(f)}{1\,\mathrm{Hz}^{-1}}\right] \quad \mathrm{dBc/Hz}. \tag{8}$$

The familiar notation "$\mathcal{L} = -140\,\mathrm{dBc/Hz}$" is shorthand for its logarithmic value; the linear $\mathcal{L}$ is positive. The "c" means relative to carrier power, and "per Hz" denotes noise density. The offset is a separation from the carrier: 1 kHz offset from a 10 MHz carrier refers to the neighborhood of 10.001 MHz in the upper sideband, or 9.999 MHz in the lower sideband.

The definition $\mathcal{L} = S_\phi/2$ is not itself a small-angle approximation. Its interpretation as the *noise power in one RF sideband per hertz divided by carrier power* does require a suitable weak-modulation regime and separation of carrier and modulation, with amplitude noise negligible or measured separately [8]. A raw RF spectrum analyzer sees AM as well as PM. Under the usual uncorrelated small AM/PM model the sideband powers add; an RF skirt is not automatically a measurement of pure phase noise. Conversely, a clock's unwrapped phase may wander through many cycles over a long record without making the phase-PSD definition invalid.

One-sided, two-sided, and single-sideband. For a real process, the one-sided PSD is twice the two-sided PSD at positive frequency (with special handling of DC and the sampled Nyquist bin). The division by two in Eq. 8 already relates the *one-sided radian-phase* PSD to $\mathcal{L}$. Do not double it again because an RF signal has two sidebands. Changing between S_ϕ and $\mathcal{L}$ changes their logarithmic numerical values by $10\log_{10}2 = 3.0103\,\mathrm{dB}$. This convention factor is separate from the two-oscillator measurement factor discussed in §2.9.

A complete point conversion. For $L_{\mathrm{dB}}(1\,\mathrm{kHz}) = -140\,\mathrm{dBc/Hz}$ at $f_0 = 10\,\mathrm{MHz}$,

$$\mathcal{L} = 10^{-14}\,\mathrm{Hz}^{-1}, \qquad S_\phi = 2\times10^{-14}\,\mathrm{rad}^2/\mathrm{Hz},$$
$$\sqrt{S_\phi} = 1.414\times10^{-7}\,\mathrm{rad}/\sqrt{\mathrm{Hz}}, \qquad \sqrt{S_x} = 2.251\,\mathrm{fs}/\sqrt{\mathrm{Hz}},$$
$$S_y = (10^3/10^7)^2(2\times10^{-14}) = 2\times10^{-22}\,\mathrm{Hz}^{-1}.$$

The time ASD is *not* a few femtoseconds of total jitter: it still has $1/\sqrt{\mathrm{Hz}}$ in its units. None of these point values determines ADEV without specifying the surrounding spectrum.

Ten log or twenty log? Use $10\log_{10}$ for power, variance and PSD ratios; $20\log_{10}$ for corresponding amplitude, deviation and ASD ratios. A factor ten in ADEV is a factor one hundred in Allan variance. For example, doubling an independent noise amplitude raises its PSD by about 6 dB; adding a second independent process of equal power raises the total by about 3 dB. Add linear densities or variances, not their logarithms and not their square roots.

Bandwidth on an analyzer. For locally flat noise, an analyzer filter with equivalent noise bandwidth B_{ENBW} contains approximately $\mathcal{L}(f)B_{\mathrm{ENBW}}$ times the carrier power in one sideband. If the instrument has not already normalized its reading, the conversion is

$$L_{\mathrm{dB}} \simeq P_N\ (\mathrm{dBm}) - P_c\ (\mathrm{dBm}) - 10\log_{10}(B_{\mathrm{ENBW}}/\mathrm{Hz}). \tag{9}$$

The displayed resolution bandwidth (RBW) is not necessarily the filter's ENBW; detector and logarithmic-averaging corrections may also be instrument-specific. For an FFT window w_n sampled at f_s, $B_{\mathrm{ENBW}} = f_s\sum|w_n|^2/|\sum w_n|^2$ [10]. This resolution-filter width is distinct from the overall bandwidth over which jitter is integrated, and from a frequency counter's averaging filter. An already normalized dBc/Hz display needs no second RBW correction.

As an arithmetic example, a locally flat $-170\,\mathrm{dBc/Hz}$ floor integrates to $-150\,\mathrm{dBc}$ in 100 Hz ENBW. With a $+5\,\mathrm{dBm}$ carrier this is $-165\,\mathrm{dBm/Hz}$, or $-145\,\mathrm{dBm}$ in that filter. Both the flatness and the actual carrier power are required; mixing a typical carrier power with a maximum relative-noise limit does not establish a worst-case absolute-noise specification.

Spurs are integrated lines. A spur specified as $-80\,\mathrm{dBc}$ is a discrete power ratio at a stated offset, not $-80\,\mathrm{dBc/Hz}$. Its displayed density peak changes with resolution bandwidth. For a known small sinusoidal *phase* modulation $\phi(t) = \beta\cos(2\pi f_m t)$, each sideband has integrated ratio $r \simeq \beta^2/4$, hence $\phi_{\mathrm{rms}} = \sqrt{2r}$ and $x_{\mathrm{rms}} = \sqrt{2r}/(2\pi f_0)$. This does not identify an unknown spur as PM: AM produces sidebands too. For the same sinusoid with time-error peak amplitude A, its Allan variance is $4A^2\sin^4(\pi f_m\tau)/\tau^2$. It oscillates with τ and even has zeros; it is not one of the broadband power laws below.

2.4 Five noise types: units, slopes and conversions

The standard model $S_y(f) = \sum_{\alpha=-2}^{2} h_\alpha f^\alpha$ specifies a *fractional-frequency* spectrum. Since S_y has units Hz^{-1}, h_α has units $\mathrm{Hz}^{-(\alpha+1)}$. Thus the five coefficients do not share one unit. Some authors instead use powers of $f/(1\,\mathrm{Hz})$; their coefficients then carry different written dimensions. Check the defining equation rather than transferring a coefficient by name.

For one term,

$$\boxed{\mathcal{L}(f) = \frac{f_0^2}{2} h_\alpha f^{\alpha-2}, \qquad h_\alpha = \frac{2\,\mathcal{L}(f)\, f^{2-\alpha}}{f_0^2}.} \tag{10}$$

The inverse gives a coefficient *conditional on the chosen noise type being responsible for that point.* For a mixture, use the sum of linear spectra and fit enough data to distinguish the terms. A local slope over a transition need not match one canonical exponent.

noise	α	unit of h_α	L slope (dB/decade)	ADEV slope (log deviation / log τ)	MDEV slope (log deviation / log τ)
white PM	2	Hz^{-3}	0	-1	$-3/2$
flicker PM	1	Hz^{-2}	-10	$\simeq -1$	-1
white FM	0	Hz^{-1}	-20	$-1/2$	$-1/2$
flicker FM	-1	1	-30	0	0
random-walk FM	-2	Hz	-40	$+1/2$	$+1/2$

Table 3: Asymptotic signatures of the five power laws. Flicker-PM ADEV has a logarithmic bandwidth factor. The MDEV slopes presume sufficient averaging and the stated sampling/filtering model; at averaging factor $m = 1$, sampled MDEV equals ADEV.

noise	Allan variance $\sigma_y^2(\tau)$	coefficient inferred from $s = \sigma_y(\tau)$
white PM	$3h_2 f_h/(4\pi^2\tau^2)$	$h_2 = 4\pi^2\tau^2 s^2/(3f_h)$
flicker PM	$h_1 C/(4\pi^2\tau^2)$	$h_1 = 4\pi^2\tau^2 s^2/C$
white FM	$h_0/(2\tau)$	$h_0 = 2\tau s^2$
flicker FM	$2\ln 2\, h_{-1}$	$h_{-1} = s^2/(2\ln 2)$
random-walk FM	$2\pi^2 h_{-2}\tau/3$	$h_{-2} = 3s^2/(2\pi^2\tau)$

Table 4: Pure-noise conversions [4]. Here $C = 3\gamma - \ln 2 + 3\ln(2\pi f_h \tau)$, with Euler's constant γ; $3\gamma - \ln 2 \simeq 1.0385$. PM rows are asymptotes for a sharp cutoff f_h and $f_h\tau \gg 1$. FM rows are infinite-band idealizations, or good approximations when relevant cutoffs and filtering are sufficiently remote.

For white FM, $\sigma_y(1\,\mathrm{s}) = 10^{-11}$ gives $h_0 = 2 \times 10^{-22}\,\mathrm{Hz}^{-1}$. At 10 Hz offset, $\mathcal{L} = (10^7)^2 h_0/(2 \times 10^2) = 10^{-10}\,\mathrm{Hz}^{-1}$, or -100 dBc/Hz. If the same ADEV is a *flicker-FM plateau*, then instead $h_{-1} = 7.213 \times 10^{-23}$ and the answer is about -114.4 dBc/Hz. Neither calculation follows from the ADEV value alone.

assumed pure noise	inferred h_α (units in Table 3)	$L(10\,\mathrm{Hz})$, dBc/Hz
white PM	1.32×10^{-23}	-91.82
flicker PM	1.94×10^{-22}	-90.14
white FM	2.00×10^{-22}	-100.00
flicker FM	7.21×10^{-23}	-114.43
random-walk FM	1.52×10^{-23}	-131.19

Table 5: Five models all normalized to ADEV 10^{-11} at 1 s. The two PM models assume an illustrative sharp 100 Hz cutoff, not the board's bandwidth. The different phase-noise predictions are the point of the example.

White and flicker PM both have ADEV slopes near -1: ADEV alone does not reliably separate them. MDEV provides another weighting and can help. Even for white FM, continuous-average MDEV is $1/\sqrt{2}$ of ADEV at the same τ; a lower MDEV number is not evidence of a better oscillator. Tables of slopes are most useful over a resolved range of times and with the measurement filter understood, rather than as a classification of one pair of noisy points.

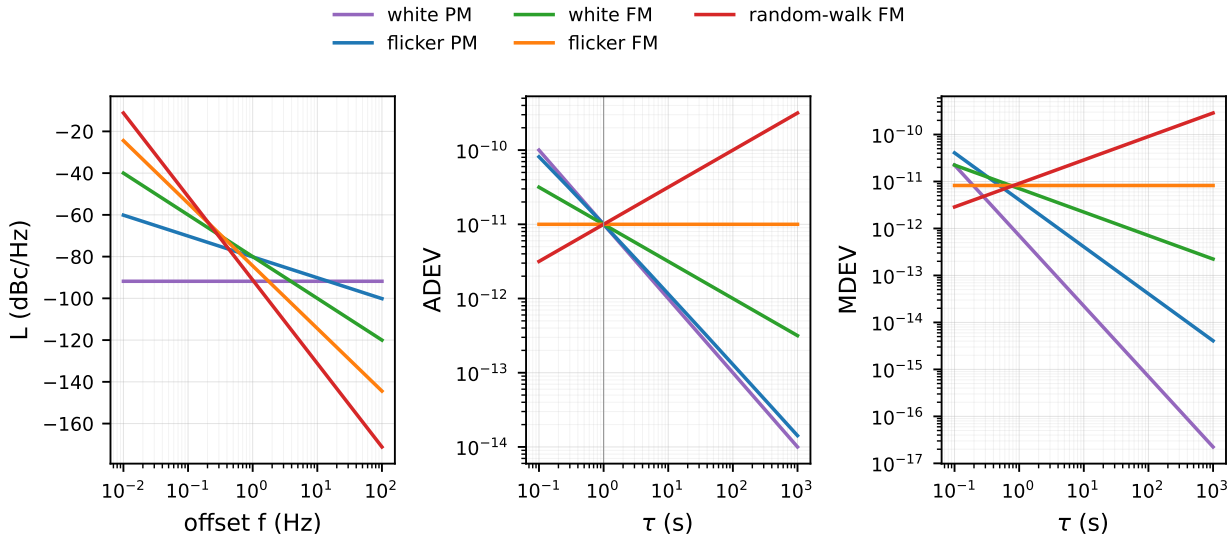


Figure 1: The same five coefficient sets as Table 5, carried through all three panels without renormalizing MDEV. These are textbook asymptotes: PM cutoff 100 Hz, $f_h\tau \geq 10$ on the deviation plots, continuous-average MDEV. The MDEV coefficients differ as well as the white-PM slope. They are not predictions through the board's meter.

2.5 Why the conversion is an integral

The transfer function of adjacent frequency averages gives

$$\boxed{\sigma_y^2(\tau) = 2\int_0^\infty S_y(f)\frac{\sin^4(\pi f\tau)}{(\pi f\tau)^2}\,\mathrm{d}f.} \tag{11}$$

This equation converts a *spectrum* into a family of time-domain variances. Its weighting is broad and has sidelobes (Figure 2); it is not a delta function at $1/\tau$. Substituting $f = 1/\tau$ in a phase-noise number cannot replace integration. Indeed, the Allan weighting is zero exactly there! An additional linear measurement filter contributes $|H(f)|^2$ inside the integral. The ideal integral for a pure $S_y \propto f^\alpha$ converges for $-3 < \alpha < 1$. White and flicker PM therefore need a high-frequency cutoff or filter.

In the continuous-average limit, MVAR has one more squared-sinc factor:

$$\mathrm{mod}\,\sigma_y^2(\tau) = 2\int_0^\infty S_y(f)\frac{\sin^6(\pi f\tau)}{(\pi f\tau)^4}\,\mathrm{d}f. \tag{12}$$

For phase samples spaced τ_0 with $\tau = m\tau_0$, the corresponding extra factor is instead $[\sin(\pi f\tau)/(m\sin(\pi f\tau_0))]^2$. Sampling, anti-alias filtering and the number of samples averaged matter, particularly for broadband PM [4, 8].

The −140 dBc/Hz point at 1 kHz in §2.3 provides a second useful ambiguity. If attributed to white FM, it gives ADEV 10^{-11} at 1 s. If it is instead a white-PM floor from zero to 100 kHz, it gives $h_2 = 2\times10^{-28}\,\mathrm{Hz}^{-3}$ and ADEV 1.233×10^{-12} at 1 s. Extend that *phase-process* floor to the hardware's 62.5 MHz cutoff and the unfiltered value becomes 3.082×10^{-11}. Same point, three different predictions. The last number describes the digital phase process, not an unqualified analog sideband measurement (§2.9).

RMS jitter and TDEV. For a specified band $[f_l, f_h]$, phase-noise integration gives

$$\phi_{\mathrm{rms}}^2[f_l,f_h] = 2\int_{f_l}^{f_h}\mathcal{L}(f)\,\mathrm{d}f, \qquad x_{\mathrm{rms}}[f_l,f_h] = \frac{\sqrt{2\int_{f_l}^{f_h}\mathcal{L}(f)\,\mathrm{d}f}}{2\pi f_0}. \tag{13}$$

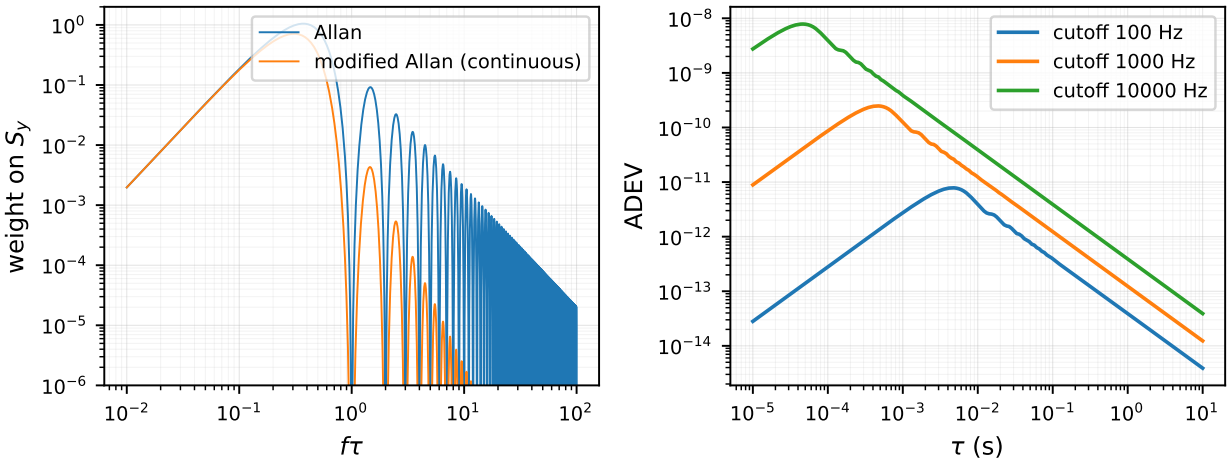


Figure 2: Left: the weighting on S_y, with MVAR in the continuous-average limit. Right: exact Allan deviations of the same $-140\,\mathrm{dBc/Hz}$ white-PM phase process with three ideal sharp cutoffs. A wider band gives larger long-τ ADEV, proportional to $\sqrt{f_h}/\tau$. At $\tau \ll 1/f_h$ the asymptote fails and the smooth band-limited phase gives a turnover. The filtering is illustrative.

Integrate *linear* densities, not dB values. State both limits, any weighting filter, and whether discrete spurs are included. A quoted RMS jitter without its integration band is incomplete. Low-frequency wander excluded from that band may dominate timekeeping. Period jitter and cycle-to-cycle jitter use differences of edge times and have different frequency weightings; they are not automatically equal to this band-limited time fluctuation.

Time deviation is a specifically defined statistic,

$$\mathrm{TDEV}(\tau) = \frac{\tau}{\sqrt{3}} \bmod \sigma_y(\tau), \tag{14}$$

not $\tau\,\mathrm{ADEV}/\sqrt{3}$ and not integrated RMS jitter over an arbitrary band. Here we use $x_{\mathrm{rms}}[f_l, f_h]$ for the latter to avoid assigning both meanings to the symbol σ_x. Neither quantity alone is a guaranteed maximum time error after a day.

Changing the carrier. An ideal noiseless multiplier by N gives $f_0' = Nf_0$ and $\phi' = N\phi$, so $L'_{\mathrm{dB}} = L_{\mathrm{dB}} + 20\log_{10} N$, but $x' = x$ and $y' = y$. Thus ideal multiplication from 10 to 100 MHz raises phase noise by 20 dB while preserving fractional-frequency stability and time jitter *over the same absolute offset band*. Real multipliers, dividers and PLLs contribute noise and filtering; division can also introduce aliasing. Use the actual transfer functions when comparing different output frequencies, not just the ideal scaling law.

2.6 Worked example: a sparse phase-noise table

Suppose the following *synthetic* table is all we know between 1 Hz and 100 kHz offset. It is deliberately not a commercial specification. Interpolate linearly in dB versus $\log f$, which means a power law in the linear density [4, pp. 34–35]:

$$\beta_j = \frac{L_{j+1,\mathrm{dB}} - L_{j,\mathrm{dB}}}{10\log_{10}(f_{j+1}/f_j)}, \qquad \mathcal{L}(f) = \ell_j (f/f_j)^{\beta_j}, \quad \ell_j = 10^{L_{j,\mathrm{dB}}/10}\,\mathrm{Hz}^{-1}. \tag{15}$$

Each segment's area is analytic: for $r = f_{j+1}/f_j$,

$$\int_{f_j}^{f_{j+1}} \mathcal{L}(f)\,\mathrm{d}f = \ell_j f_j \begin{cases} (r^{\beta_j+1} - 1)/(\beta_j + 1), & \beta_j \neq -1, \\ \ln r, & \beta_j = -1. \end{cases} \tag{16}$$

offset interval (Hz)	start (dBc/Hz)	end (dBc/Hz)	β_j
1–10	−100	−130	−3
10–100	−130	−150	−2
100–1000	−150	−155	−0.5
1000–10000	−155	−155	0
10000–100000	−155	−155	0

Table 6: Input to the worked example. The exponent −0.5 is an interpolation between two points, not evidence of another canonical noise mechanism.

Summing gives $\int \mathcal{L}\,\mathrm{d}f = 8.214 \times 10^{-11}$ and $x_{\mathrm{rms}}[1\,\mathrm{Hz}, 100\,\mathrm{kHz}] = 0.204\,\mathrm{ps}$. This small jitter number describes only that band.

For Allan variance the same spectrum gives the finite-band contribution

$$V_{\mathrm{band}}(\tau) = \frac{4}{\pi^2 f_0^2 \tau^2} \int_{1\,\mathrm{Hz}}^{100\,\mathrm{kHz}} \mathcal{L}(f) \sin^4(\pi f \tau)\,\mathrm{d}f. \tag{17}$$

The second column of Table 7 is $\sqrt{V_{\mathrm{band}}}$. It is not the full oscillator ADEV. The integrand is nonnegative, so omitting frequencies omits nonnegative contributions *if the in-band spectrum is actually known*. Isolated vendor limits do not prove our interpolated curve is an upper or lower envelope between the listed points.

τ (s)	1 Hz–100 kHz only	+ assumed sub-Hz white FM	+ assumed sub-Hz flicker FM
1.00×10^{-3}	2.197×10^{-10}	2.197×10^{-10}	2.197×10^{-10}
0.01	2.216×10^{-11}	2.216×10^{-11}	2.216×10^{-11}
0.1	2.733×10^{-12}	2.756×10^{-12}	2.768×10^{-12}
1	3.386×10^{-13}	9.851×10^{-13}	1.680×10^{-12}
10	3.531×10^{-14}	3.158×10^{-13}	1.665×10^{-12}

Table 7: ADEV contributions calculated from Table 6. The last two columns extend the spectrum below 1 Hz by f^{-2} or f^{-3} in $\mathcal{L}$, continuously matched at 1 Hz. Both still omit offsets above 100 kHz. These are assumptions, not extra information extracted from the table.

At $\tau = 10\,\mathrm{ms}$, the 10–100 kHz segment contributes 88.1 % of the listed band's Allan variance, even though $1/\tau = 100\,\mathrm{Hz}$. The high-frequency white-PM floor matters greatly. At long averaging times, unlisted *low* offsets matter instead: extending the same table below 1 Hz with white FM or flicker FM produces the distinct results shown in Figure 3. A data sheet beginning at 1 Hz cannot by itself specify stability over hours.

The numerical calculation splits at each spectral breakpoint. It uses positive quadrature where $f\tau$ is small and the expansion $\sin^4 z = 3/8 - \cos(2z)/2 + \cos(4z)/8$ with oscillatory quadrature at large $f\tau$. A sparse logarithmic grid applied directly to the sine factor can alias the oscillations and give a convincing but wrong result. The generator and its independent numerical checks are named at the end of this section.

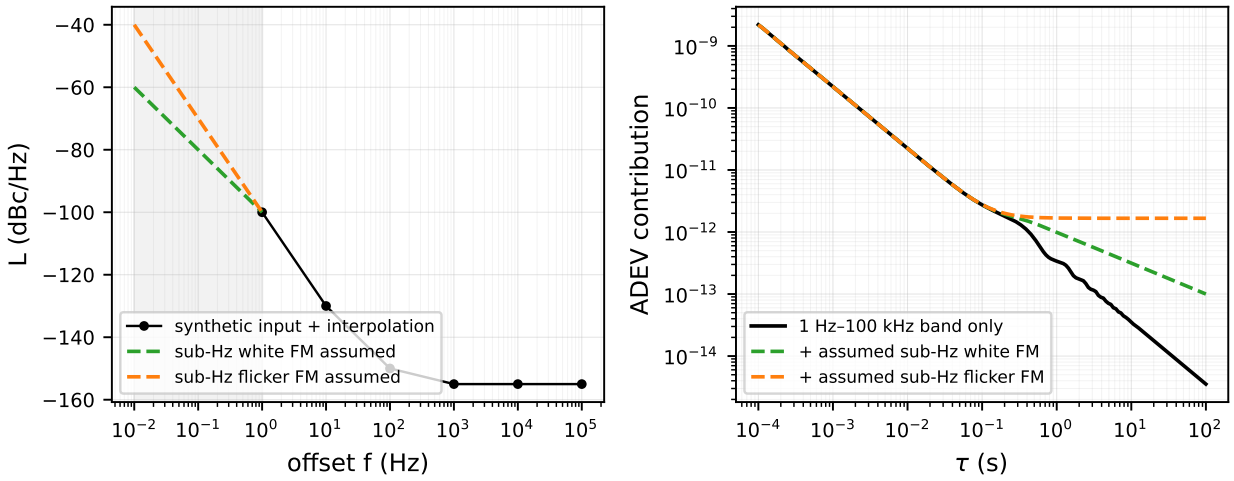


Figure 3: An explicitly synthetic spectrum and its conditional ADEV predictions. The solid black spectrum is specified only over the listed band. Shading marks unlisted sub-Hz offsets; dashed continuations are alternative hypotheses. The in-band curve is identical in all three calculations.

2.7 Reading real OCXO and CSAC data sheets

An *OCXO* is an oven-controlled crystal oscillator. The oven reduces temperature-driven frequency variation; it does not remove calibration error, aging or noise. A *CSAC* is a chip-scale atomic clock: a compact electrical oscillator is controlled using an atomic transition as its reference. In the Microchip SA65 the oscillator is a TCXO; in the SA65-LN it is an ovenized evacuated miniature crystal oscillator (EMXO) [11, 12]. The atoms and the quartz/control electronics play different roles. Atomic referencing does not imply that every Fourier offset is atomically limited, nor that an atomic clock must have lower short-term phase noise than a good quartz oscillator.

Which documents and variants? The following examples reproduce selected entries from the manufacturer's OX-171 and OX-174 sheets with printed revisions "5-6-19 jar" and "5-6-2019 jar," the SA65 sheet DS00003478B (July 2021), and SA65-LN sheet DS00004687A (January 2025) [13, 14, 15, 16]. These are identified document revisions, not an assertion about every current ordering variant. All four examples have 10 MHz outputs. For the OCXO phase-noise comparison use sine-wave outputs, with OX-174 phase-noise *code B*, available at 5 V. Its headline −175 dBc/Hz belongs to *code C at 12 V*, and must not be assigned to this variant. For SA65 use the industrial ordering variants 090-02789-002/-012.

offset (Hz)	OX-171	OX-174 code B	SA65	SA65-LN
1	−95	−100	−44	−85
10	−125	−130	−64	−120
100	−140	−155	−110	−140
1000	−145	−165	−128	−145
10000	−145	−170	−135	−150
100000	–	−170	−140	−155

Table 8: Published single-sideband phase-noise upper levels, in dBc/Hz, at 10 MHz. OX-171 p. 3 and OX-174 p. 1 give maximum values; SA65 p. 3 and SA65-LN p. 2 use strict "<" entries. A dash means no selected entry. These are not measured spectra or guaranteed interpolated envelopes.

τ (s)	OX-171	OX-174	SA65	SA65-LN
0.1	–	5.00×10^{-12}	–	–
1	5.00×10^{-12}	3.00×10^{-12}	3.00×10^{-10}	3.00×10^{-11}
10	8.00×10^{-12}	4.00×10^{-12}	–	5.00×10^{-11}
100	1.00×10^{-11}	5.00×10^{-12}	3.00×10^{-11}	3.00×10^{-11}
1000	5.00×10^{-11}	5.00×10^{-12}	–	–

Table 9: Selected published ADEV upper levels, dimensionless. OCXO columns use *maxima*, not OX-174's separate typical values. SA65 uses industrial ordering limits ($\leq$) at 1 and 100 s; SA65-LN uses its "<" table. SA65's conflicting 10 s entries and its main table's unqualified 1000 s entry are omitted. A dash does not imply zero noise.

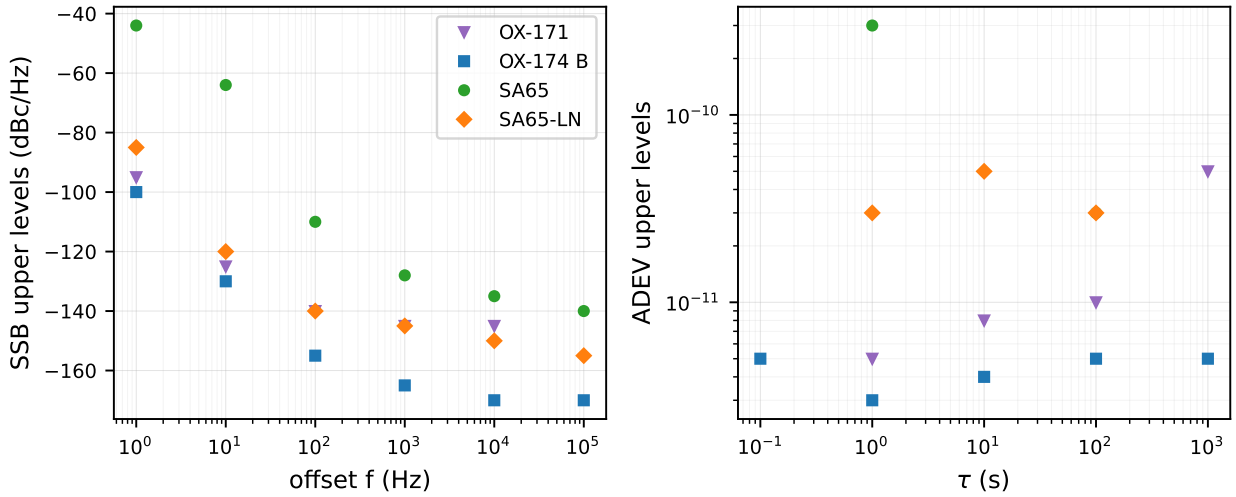


Figure 4: The specification points in Tables 8 and 9; markers identify devices, not measurement runs. Points are intentionally unconnected: bounds at isolated offsets or averaging times do not determine a curve between them. In particular, slopes between specification limits need not be the noise slopes of any individual oscillator.

The OCXO comparison. OX-174 code B has a phase-noise maximum 25 dB lower than OX-171 at 10 kHz offset, and a lower 1 s ADEV maximum. Yet its shipment tolerance and aging limits are looser (Table 10). This compares published limits, not a prediction that every two purchased units differ by those ratios.

specification	OX-171	OX-174	conditions / meaning
initial tolerance	±25 ppb	±50 ppb	at shipment, nominal frequency-control voltage
temperature variation	±0.6 ppb	±5 ppb	−20 to +70 °C options; relative to +25 °C
supply variation	±0.5 ppb	±2 ppb	supply changed by ±5 %
load variation	±0.2 ppb	±2 ppb	load changed by ±5 %
daily aging	±0.06 ppb/day	±0.5 ppb/day	after seven days operation
annual aging	±15 ppb	±100 ppb	OX-171 after 72 h; OX-174 first year after seven days
warm-up, maximum	5 min	5 min	within ±5 / ±10 ppb of the one-hour frequency, at +25 °C
typical output	+8 dBm	+5 dBm	selected 5 V sine outputs into 50 Ω

Table 10: Selected OCXO specifications [13, 14], pp. 1–3 and notes on p. 6. Unless stated otherwise, values apply after warm-up at nominal supply, tuning, load and 25 °C. The temperature row explicitly selects the −20 to +70 °C grades, not every grade of each family.

At 10 MHz, the initial ±25 and ±50 ppb tolerances mean ±0.25 and ±0.50 Hz. If those offsets persisted unchanged, their time-error magnitudes after one day would be 2.16 and 4.32 ms. Compare that with microhertz-scale short-term Allan deviations: accuracy and stability are clearly different.

Frequency calibration or tuning can remove a static offset; it does not by itself improve the oscillator's phase-noise spectrum.

The temperature row is a frequency *excursion relative to the value at* $25\,^{\circ}C$, not accuracy relative to exactly 10 MHz, and not ppb per degree. The difference between two temperatures could span both sides of the quoted interval. Likewise, supply and load entries specify changes for a stated perturbation; they are not RMS noise densities or necessarily linear derivatives. Supply sensitivity is often called *pushing*, load sensitivity *pulling*. Frequency-control range and tuning resolution are also distinct from accuracy: a very fine digital step does not establish that the frequency is known to one step.

Aging conditions matter. OX-171's daily entries improve from 1 ppb/day after 24 h to 0.1 after 72 h and 0.06 after seven days. Multiplying the final daily figure by 365 does not replace the independently specified annual figure. Warm-up to a tolerance around the one-hour frequency does not mean calibration to nominal frequency within that tolerance. Output-amplitude startup is yet another time: a waveform can be present before its frequency has settled.

Retrace describes the change after a power cycle and has its own protocol. OX-171 specifies ±2 ppb comparing a 24 h powered reading with a reading after 24 h off and 60 min on. OX-174 specifies ±10 ppb comparing the ends of two 24 h powered periods separated by 24 h off. These are not identical tests.

Output level and termination. A dBm level is absolute power relative to 1 mW, whereas dBc is relative power. For a sine delivering power P into resistance R, $V_{\mathrm{rms}} = \sqrt{PR}$ and $V_{\mathrm{pp}} = 2\sqrt{2PR}$. The typical OCXO levels above imply 1.589 Vpp and 1.125 Vpp into 50 Ω. These are loaded voltages. A high-impedance voltage depends on the source impedance and cannot be inferred by assuming every output is an ideal 50 Ω source. CMOS logic levels, duty cycle and rise/fall times describe a different interface; use the manufacturer's recommended buffer and load. Harmonics specified in dBc are discrete spectral components, not the broadband phase-noise floor.

The CSAC comparison. SA65-LN improves the stated 1 s ADEV by a factor ten and greatly reduces the phase-noise levels, while retaining the same listed shipment accuracy, retrace and typical aging [15, 16, 12]. Its low-noise oscillator also costs power: SA65 specifies less than 120 mW, SA65-LN at most 295 mW. The standard SA65 supplies a 10 MHz CMOS signal with a specified 1 MΩ load; SA65-LN supplies a 6–9 dBm sine into 50 Ω. This is not a direct interchangeable output-level specification.

SA65 quantity	quoted value	qualification
frequency offset	$\pm 5 \times 10^{-11}$	maximum at shipment
retrace	$\pm 5 \times 10^{-10}$	maximum after 48 h off
monthly aging	$< 9 \times 10^{-10}$	typical, after 30 days continuous operation
yearly aging	$< 10^{-8}$	same aging qualification
temperature sensitivity	$\pm 3 \times 10^{-10}$	industrial −40 to +80 °C; temperature rate ≤ 0.5 °C/min
supply sensitivity	$\pm 4 \times 10^{-10}$	over 3.2–5.1 V
magnetic sensitivity	$\pm 9 \times 10^{-11}$/gauss	field up to 2 gauss
PPS synchronization	±100 ns	coarse alignment to external pulse
digital tuning	range $\pm 10^{-6}$	resolution 10^{-12}

Table 11: Reading the qualifications is part of reading the units. SA65 sheet DS00003478B, pp. 2–3, including the temperature-ramp and aging footnotes [15]. Default conditions are 3.3 V and 25 °C. These rows are different kinds of specification and cannot all be treated as standard deviations.

The shipment accuracy 5×10^{-11} corresponds to 0.5 mHz, whereas the 1 s ADEV 3×10^{-10} corresponds to 3 mHz Allan deviation. There is no contradiction: an accurately calibrated mean can still fluctuate appreciably over a short averaging interval. Shipment accuracy is also not a permanent bound; the guide discusses changes during transport, environmental excursions and aging [11]. The

temperature and supply figures can exceed the shipment offset and must be considered for the actual installation.

Per-field or per-acceleration sensitivities need their own unit conversion. One gauss is 10^{-4} T; the magnetic row is therefore 9×10^{-7} per tesla in magnitude within its stated conditions. An oscillator's acceleration sensitivity expressed per g uses g as a unit of acceleration, not grams. A sinusoidal fractional-frequency perturbation $y = A_y \cos(2\pi f_v t)$ produces radian-phase amplitude $f_0 A_y / f_v$ and hence PM sidebands. A vibration sensitivity cannot be turned into a broadband dBc/Hz curve without an acceleration spectrum and the relevant transfer response. The same principle applies to temperature-induced frequency variation.

Reading a specification includes checking consistency

SA65's main table gives 1.5×10^{-10} at 10 s, while its industrial ordering table on the same page gives $\leq 10^{-10}$. Its 1000 s main-table entry has no explicit typical/maximum label. We have not silently selected the stronger interpretation. OX-174's extended-temperature row prints -10 in both limit columns, while the ordering key says ±10 ppb; the comparison above uses the unambiguous -20 to $+70$ °C grade. For procurement or acceptance testing, resolve such discrepancies against the exact ordered revision with the supplier. A smooth typical plot, a table of limits and a production test guarantee are different evidence, even when printed on the same page.

Atomic lock, synchronization and disciplining. Four states need distinguishing. An atomically locked CSAC can be free-running with respect to UTC: the internal atomic loop is active, but no external epoch has been supplied. PPS synchronization aligns a divider edge to an external pulse; it does not by itself calibrate the RF frequency. External disciplining uses a control loop to steer phase and frequency toward an external reference. If that reference disappears, the SA65 holds its last steering value. This external-reference *holdover* is different from losing atomic lock, when the quartz oscillator is no longer being corrected to the atoms [11, pp. 15–24].

The output waveform and PPS can exist before atomic lock. SA65's sheet specifies lock in less than 120 s, while the cited guide describes startup within three minutes at 25 °C; use the applicable document and status indication rather than equating a visible waveform with settled performance. Cable delay, reference accuracy and loop settling affect externally synchronized time. The guide allows disciplining time constants from 10 to 10 000 s and explains why a short loop can import a noisy reference's short-term fluctuations. The RF carrier's long-term accuracy, its phase noise and the PPS epoch accuracy remain separate specifications even when all outputs share one oscillator.

2.8 A conditional holdover calculation

Let T be elapsed time since the external reference was lost, rather than the ADEV averaging time τ. A useful decomposition is

$$x(T) = x_0 + y_0 T + \tfrac{1}{2} D T^2 + x_{\text{noise}}(T). \tag{18}$$

Here x_0 is initial epoch error, y_0 residual frequency offset after calibration, and $D = \mathrm{d}y/\mathrm{d}t$ an assumed constant frequency drift in s^{-1}. Aging is internal evolution; measured drift can include environmental effects too.

For a deliberately conditional example, assume continuous atomic lock, constant environment, at least 30 days prior operation, and a fresh calibration at the start of holdover. Use the SA65's *typical monthly-aging scale* to choose

$$D = \frac{9 \times 10^{-10}}{30 \times 86400\,\mathrm{s}} = 3.472 \times 10^{-16}\,\mathrm{s}^{-1}. \tag{19}$$

Treating this as constant linear drift is a modeling choice. The monthly specification neither guarantees this instantaneous slope nor its sign or indefinite persistence. Set $x_0 = 100$ ns and $y_0 = 5 \times 10^{-11}$ for example A. The latter is an assumed *newly calibrated residual*, chosen at the scale of the shipment accuracy, not a claim that shipment accuracy still holds after 30 days. Example B assumes a better initial calibration, $x_0 = 5$ ns and $y_0 = 10^{-12}$, with the same drift.

	initial x_0	offset y_0T	drift $DT^2/2$	same-sign total
Example A	0.1	4.32	1.296	5.716
Example B	5.0000×10^{-3}	0.0864	1.296	1.3874

Table 12: Calculated time errors after 24 h, all in μs. These are deterministic scenarios with positive contributions and no stochastic residual included; they are not CSAC holdover limits.

Improving initial frequency knowledge greatly reduces the linear contribution; it does not remove aging. A persistent additional frequency error of 3×10^{-10} would add 25.92 μs in a day, but the temperature-range specification does not say that any particular temperature change causes that error. A thermal trajectory and oscillator response are needed.

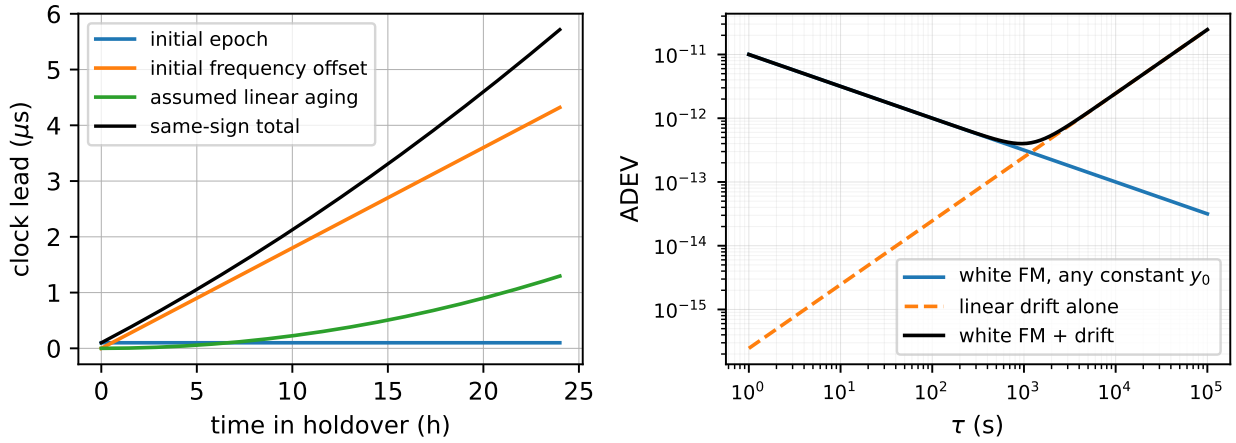


Figure 5: Left: example A of Table 12, with explicit analytic components. Right: a separate illustrative white-FM model (ADEV 10^{-11} at 1 s), with the same assumed drift added. Any constant frequency offset leaves its ADEV unchanged, while drift eventually dominates. Neither panel is a measured clock record or a vendor holdover guarantee.

If each deterministic magnitude were a true bound, their absolute contributions could be added for a conservative envelope. Here the drift scale is typical and model-based, so even that interpretation is unavailable. Root-sum-square addition is justified for suitably independent zero-mean random uncertainties, not merely because several numbers have the same units. Correlated temperature, supply and aging effects require their covariance or an explicit bound calculation.

Random time-error growth also requires a noise and estimation model. For pure white FM with $\sigma_y(\tau_r) = s$, exact initial epoch, and no offset or drift, independent increments give time-error RMS $s\sqrt{\tau_r T}$. It is not sT. Flicker and random-walk FM, uncertain initial frequency, and a frequency estimate based on a finite pre-holdover record give different answers. TDEV and ADEV characterize particular statistics; neither supplies a universal maximum error or a confidence level for a holdover prediction.

Real system-level holdover claims can also include GNSS estimation, temperature compensation and adaptive aging correction. For example, the manufacturer of the SA65-based MD-015 module lists 1.5 μs after 24 h with a 2 °C change, but its footnote calls this *estimated* performance after 30 days locked at 25 °C, zero initial phase and frequency error, and otherwise static conditions [17, p. 2]. That is a specification of a conditioned system under stated assumptions, not a property to assign to a bare free-running SA65.

2.9 Comparing instruments, and using these units on this board

The reference is part of the result. For independent sources A and B and independent instrument noise,

$$S_{y,A-B} = S_{y,A} + S_{y,B} + S_{y,\text{instrument}}, \qquad \sigma^2_{y,A-B} = \sigma^2_{y,A} + \sigma^2_{y,B} + \sigma^2_{y,\text{instrument}}. \tag{20}$$

Only for *equal independent* sources with negligible instrument noise does the pair have $\sqrt{2}$ times either ADEV, or 3.0103 dB more phase-noise density. It is not a universal correction. For correlated sources the spectrum contains $-2\,\mathrm{Re}\,S_{AB}$; shared-reference noise can cancel. An excellent relative result then says little about absolute performance against UTC or an independent frequency standard. Subtract a known floor in linear power/variance, with its uncertainty; subtracting deviations or dB numbers is wrong. A negative residual indicates an unresolved estimate, not negative physical noise [4].

What was averaged? A counter's gate duration, spacing between readings, dead time and front-end bandwidth are separate quantities. Rectangular (Π), triangular (Λ) and regression-weighted counters filter differently [18]. Dead time changes the relation to contiguous-average Allan variance and can bias noise identification; a correction depends on noise type. Similarly, a 10 ms phase average saved once per second is not a one-second phase average. An instrument result should identify statistic, window, cadence, bandwidth, reference, record length, confidence interval, and any drift or outlier removal. This is what makes two laboratories' numbers comparable.

The simulator's bandwidths. Hardware white PM and FM are generated at 125 MSa/s. The native white-PM phase sequence has a 62.5 MHz Nyquist bandwidth; flicker and random-walk playback extend only through the modeled segment band, 500 Hz at the default rate. These bands describe the noise mechanisms. At 10 MHz carrier, interpreting a 62.5 MHz-wide phase process as two isolated analog RF sidebands would violate the carrier/modulation separation used in the elementary RF interpretation. The phase-to-sine conversion, overlapping sideband contributions, DAC hold and analog output filter must be accounted for. A generated h_α is not a measurement of connector phase noise, and the board oscillator's absolute phase noise remains unmeasured.

The meter averages phase over its window; the idealized window contributes $|H(f)|^2 = \mathrm{sinc}^2(f\tau_w)$, where $\mathrm{sinc}(u) = \sin(\pi u)/(\pi u)$. Broadband white PM has an additional I/Q-demodulation variance factor of approximately 1.5 in this meter's model. That is not a universal conversion factor between ADEV and dBc/Hz. The testing manual derives the model and compares it with measurements. For actual recordings, use the window- and cadence-aware analysis, rather than the unfiltered asymptotes of Table 4.

A practical conversion recipe. First choose the quantity to reproduce: a noise spectrum, a justified noise-type region of an ADEV curve, or a deterministic offset/drift. Record its carrier, bandwidth and conditions. Convert to linear densities before fitting or adding terms. Infer h_α only within the range supported by that model. Then predict what the simulator and the chosen measurement window will actually show. The existing helpers make the algebra short:

```
from atomic_clock import noise

# Pure white FM: textbook (unfiltered) ADEV 1e-11 at 1 s.
h0 = noise.h_from_adev("h0", 1e-11, 1.0)
level = noise.phase_psd_dbc({"h0": h0}, 10.0)  # -100 dBc/Hz

# A flat white-PM phase-noise density, not total jitter.
h2 = noise.h2_from_dbc(-140.0)                # 2e-28 Hz^-3

# An explicitly chosen sharp cutoff for a pure flicker-PM example.
h1 = noise.h_from_adev("h1", 1e-11, 1.0, fh=500.0)
```

`phase_psd_dbc` assumes this design's 10 MHz carrier. The default `fh` in `h_from_adev/adev` is 62.5 MHz, appropriate to the native white-PM generator, not automatically to played-back flicker PM or a filtered measurement. It is not a tool for inferring an arbitrary mixture from one ADEV point. For meter-aware levels use the host model `clocklib.model.h_for_adev` with the actual window, cadence and segment rate. Turning a fitted spectrum into settings also requires checking which parts lie inside the playback and hardware bandwidths.

A final reading order for a data sheet. Identify the exact variant, carrier and output interface; separate typical curves from stated limits; read temperature, supply, warm-up and aging footnotes; compare like statistics at like averaging times or offsets; and only then perform the conversion. Keep a measured result, a specification limit and a model prediction visibly distinct. Doing so lets the same equations serve a quartz oscillator, an atomic reference and this simulator without claiming information that the data do not contain.

How it is verified

`tools/metrology_figures.py`, run through `tools/manual_figures.py -only metrology`, generates the five figures and `metrology_values.tex` independently of the measured-floor tables. `tests/test_metrology_figures.py` compares its oscillatory integration with independent positive quadrature, the analytic FM limits and exact band-limited white PM; checks the continuous MDEV coefficients, carrier/power scaling and the worked band integrals; and verifies generated-number freshness and selective generation. Vendor points retain their document/page provenance in the generator and captions. Verification runs are recorded in `docs/VERIFICATION.md`.

3 Architecture

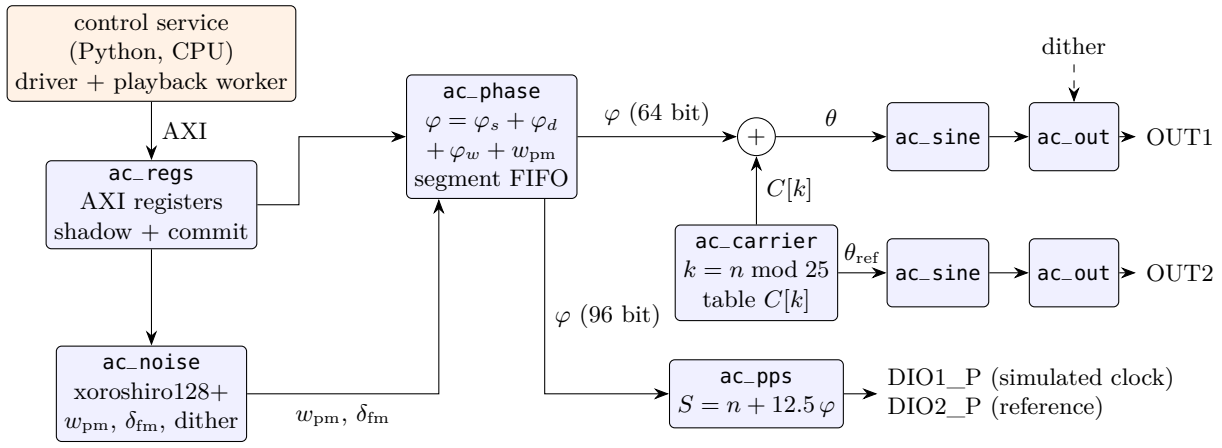


Figure 6: Signal flow of `src/ac_core.v`. Everything runs on the one 125 MHz sample clock; the CPU writes settings and FIFO rates and reads snapshots over AXI. The carrier adder uses the low 64 bits of φ (one cycle), the PPS all 96 bits; the dither comes from `ac_noise` and is added on OUT1 only. φ, and with it every noise term and the playback, enters only OUT1's adder: OUT2's phase is the bare table $C[k]$. OUT2 passes through identical carrier, sine and output stages with $\varphi = 0$ and no dither, so with $\varphi = 0$ the two outputs are equal sample for sample. The phase meter (`ac_meter`, on IN1/IN2) is not shown.

Figure 6 shows the data path. Each 8 ns sample, the FPGA

1. forms OUT1's phase offset φ from four parts: the *static* phase φ_s (the `phase` setting, approached at a limited rate), the *integrated rate* φ_d (frequency offset plus the played-back segments), the white-FM random walk φ_w and the white-PM sample w_{pm} (§5);
2. adds the exact carrier phase $C[n \bmod 25]$ (§4);
3. converts the top 28 bits of the phase to a sine value (§8);

4. scales it by the amplitude, adds the offset (and the dither) and rounds it to the 14-bit DAC grid (§9);
5. compares the unwrapped phase with the next second to drive the PPS pins (§10).

From φ_s or φ_d to the core's DAC stream the pipeline is 13 clocks (104 ns): 3 for the phase sum, 1 for the carrier adder, 6 for the sine and 3 for the output stage (`PHASE_LATENCY` in `src/ac_core.v`). The later sum inputs φ_w and w_{pm} take 12 and 11 clocks respectively. The vendor DAC input register and SAME_EDGE ODDR add two clock stages, included in the PPS alignment (§10); analog delays at the connectors remain unmeasured.

Settings change on one clock edge. The CPU writes 32-bit AXI registers; many settings are 64 or 96 bits wide. All configuration registers are therefore *shadowed*: writes go to a shadow copy, and writing 1 to bit 0 of the CTRL register copies all of them to the active registers on one clock edge (`src/ac_regs.v`). A new phase, rate, amplitude or noise level never appears half-written. The other CTRL bits are requests for strobes (start/stop playback, clear the FIFO, snapshot, align, arm the PPS); a snapshot waits through an alignment (§5.5). A commit that changes the noise seed restarts the random generators. Byte strobes are honored; a write with all byte strobes zero has no data, dirty-bit or control-strobe effect.

The software side. The board's control service (`rp-overlay.service`, Python on the 667 MHz ARM CPU) owns the hardware through the driver `control/atomic_clock/driver.py`, which converts physical settings into register words (§5.2, §6, §9). Its background worker (`control/atomic_clock/playback.py`) computes the slow noise and keeps the segment FIFO filled (§7). The hardware never depends on the software for continuity: if the worker stops, the FIFO's last rate is held and the phase stays continuous.

4 The exact carrier

The standard technique and its problem. The usual way to make a sine of programmable frequency from a clock is the *direct digital synthesizer* (DDS), also called a *numerically controlled oscillator* (NCO) [19, 20, 21]: an N-bit *phase accumulator* adds a tuning word M every clock, so its value is the phase in units of 2^{-N} cycle, and $f = M f_{\mathrm{s}}/2^N$. For this design $f_0/f_{\mathrm{s}} = 10/125 = 2/25$, and since 25 contains the factor 5, no finite binary fraction equals 2/25. The best tuning word $M = \mathrm{round}(2^N \cdot 2/25)$ misses by up to half an LSB per clock, a fractional frequency error of up to

$$|\Delta y| \le \frac{2^{-(N+1)}}{2/25} = 6.25 \cdot 2^{-N} \quad (2.2 \cdot 10^{-14} \text{ for } N = 48,\ 3.4 \cdot 10^{-19} \text{ for } N = 64). \tag{21}$$

For a 48-bit accumulator this is a built-in frequency offset that accumulates to several whole carrier cycles per year: a simulated clock with a drift nobody asked for.

The solution: a modulus-25 table. Over 25 samples the 10 MHz carrier completes exactly 2 cycles. The carrier phase at sample n is therefore $\mathrm{frac}(2n/25)$, which takes only 25 values, so instead of accumulating it the design looks it up:

$$k = n \bmod 25, \qquad C[k] = \mathrm{round}\big(\,\mathrm{frac}(2k/25) \cdot 2^{64}\big) \quad (\text{units of } 2^{-64} \text{ cycle}). \tag{22}$$

Each entry is off by at most half an LSB, $|C[k]\,2^{-64} - \mathrm{frac}(2k/25)| \le 2^{-65}$ cycle. (No entry is a tie: $2^{64} \bmod 25 = 16$, so $2k \cdot 2^{64}/25$ is never an integer plus one half.) The error depends only on k: it is the same in every 25-sample period and *never accumulates*. The carrier has exactly the frequency $0.08\,f_{\mathrm{s}}$ at every time scale, from one sample to the 4677 years after which the 64-bit sample counter of the PPS would wrap. Its tiny periodic error is a set of lines at multiples of $f_{\mathrm{s}}/25 = 5$ MHz with an amplitude of order $2\pi \cdot 2^{-65}$ rad, about 10^{-19}, which no instrument can see.

In the DDS literature the general idea, an accumulator whose modulus is not a power of two so that rational frequency ratios are exact, is called a *programmable-modulus* (or rational) DDS [22]. Here it degenerates to its simplest form: a counter modulo 25 with increment 1 indexes a table that holds the phase $2k/25$ with 64-bit precision.

atomic_clock/src/ac_carrier.v

```
always @(posedge clk) begin
    if (!resetn) begin
        k <= 0; c <= 0; theta <= 0; theta_ref <= 0;
    end else begin
        k <= k == 5'd24 ? 5'd0 : k + 5'd1;
        case (k)
            5'd0: c <= 64'h0000000000000000;
            5'd1: c <= 64'h147ae147ae147ae1;
            5'd2: c <= 64'h28f5c28f5c28f5c3;
// ...
            5'd23: c <= 64'hd70a3d70a3d70a3d;
            default: c <= 64'heb851eb851eb851f;
        endcase
        theta <= c + phi;
        theta_ref <= c;
```

The last two lines are where OUT1's phase offset enters: $\theta = C[k] + \varphi \bmod 2^{64}$ for OUT1 and $\theta_{\text{ref}} = C[k]$ for OUT2. The sum wraps modulo one cycle, which is all a sine needs; the whole cycles of φ are kept elsewhere (§5).

How it is verified

`tests/test_control.py` recomputes every table entry from the exact fraction with integer arithmetic and checks that the 25 steps add up to exactly two cycles. `tests/tb_system.sv` checks the hardware sequence and that OUT1 equals OUT2 sample for sample at $\varphi = 0$. Records in `docs/VERIFICATION.md`.

5 The phase word

5.1 Format and resolution

OUT1's offset is an *unwrapped* phase, a 96-bit signed integer in units of 2^{-64} cycle: 32 bits of whole cycles and a 64-bit fraction,

$$\varphi = \varphi_s + \varphi_d + \varphi_w + w_{\text{pm}}. \tag{23}$$

The range is $\pm 2^{31}$ cycles, ± 214.7 s of time error. The sine uses only the fraction, but the whole cycles matter for the PPS output (§10): $\varphi = 3$ cycles looks the same on a scope as $\varphi = 0$, yet the simulated clock is 300 ns ahead. The resolution, 2^{-64} cycle $= 5.4 \cdot 10^{-20}$ cycle ($5.4 \cdot 10^{-27}$ s), is far finer than any output can show; it is chosen for the integrator: a frequency offset is an integer number of LSB per sample, and 2^{-64} makes that integer fine enough (§5.2). A year has $3.9 \cdot 10^{15} \approx 2^{52}$ samples.

Internally, the static phase φ_s and its goal *target* − *origin* are signed **97-bit** values: subtracting two signed 96-bit phases needs the extra bit. Their slew difference is signed **98-bit**, so even a move between opposite extreme goals has the correct direction. The total phase and public snapshots keep the low 96 bits modulo 2^{96}, including the snapshot of φ_s.

Sign convention: OUT1 $= A \sin 2\pi(f_0 t + \varphi)$, so a positive φ makes OUT1 *lead* OUT2; its zero crossings come $\varphi \cdot 100$ ns earlier.

5.2 The frequency offset: an exact integer rate

A constant fractional frequency offset y_0 is a phase that grows by $y_0\, f_0/f_{\text{s}}$ cycles per sample. The driver converts it to the integer

$$R = \text{round}(y_0 \cdot 0.08 \cdot 2^{64}) \quad \text{LSB per sample}, \tag{24}$$

which φ_d accumulates every clock. One LSB per sample is a fractional frequency of $2^{-64}/0.08 = 6.8 \cdot 10^{-19}$, so the realized offset $R \cdot 2^{-64}/0.08$ equals the request to within $3.4 \cdot 10^{-19}$ and, being an integer added exactly, never drifts. Example: $y_0 = 10^{-9}$ gives $R = 1\,475\,739\,526$ and moves OUT1 by 1 ns per second, 31.6 ms per year. The driver accepts $|y_0| \le 10^{-3}$.

5.3 The static phase and its slew limit

The `phase` setting is a target. A phase *step* makes the waveform jump on one sample, which a receiver locked to the 10 MHz may not tolerate, so by default the hardware approaches the target at a limited rate: every two clocks φ_s moves by at most $2R_s$, with $R_s = \mathrm{round}(y_s \cdot 0.08 \cdot 2^{64})$, and lands on the target exactly when it is closer than that. y_s (the `phase_slew` setting, default 10^{-6}) is the largest fractional frequency deviation the move may cause: a quarter-cycle move takes 25 ms at the default. In control language this is a *slew-rate limiter*; the two-clock loop exists for timing closure (the wide compare and add do not fit in one 8 ns clock). $y_s = 0$ (or an absurdly large value) gives steps. The target, up/down candidates, slew bounds and step mode are latched as one operation: a configuration change between stages cannot select an old candidate with a new limit.

atomic_clock/src/ac_phase.v

```
reg signed [96:0] ps, t_a, up, dn, t_eff;
reg [95:0] origin;
reg [95:0] pd, pw;
reg signed [97:0] d;
// ...
always @(posedge clk) begin
    if (!resetn) begin
        ps <= 0; t_a <= 0; up <= 0; dn <= 0; d <= 0; half <= 0;
        r2_a <= 0; r2n_a <= 0; step_a <= 1;
    end else if (align_busy) begin
        // Discard even a candidate due on A: it may carry the old step mode.
        // Restart with a fresh coherent operation after the new origin/goal.
        half <= 0;
    end else if (!half) begin
        t_a <= t_eff;
        d <= $signed({t_eff[96], t_eff}) - $signed({ps[96], ps});
        up <= ps + $signed({31'd0, r2});
        dn <= ps - $signed({31'd0, r2});
        r2_a <= r2; r2n_a <= r2n; step_a <= step_mode;
        half <= 1;
    end else begin
        half <= 0;
        if (step_a) ps <= t_a;
        else if (d > $signed({32'd0, r2_a})) ps <= up;
        else if (d < $signed({{32{r2n_a[65]}}, r2n_a})) ps <= dn;
        else ps <= t_a;
    end
end
```

Because φ is unwrapped, 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 for the PPS these are different clocks.

Alignment. The integrated parts φ_d and φ_w are never reset during operation (the playback mirrors φ_d exactly, §7). Setting a phase must nevertheless put OUT1 at exactly that phase relative to OUT2. The ALIGN strobe (CTRL bit 8) does this: after settling it captures $\mathit{origin} = \varphi_d + \varphi_w$ (low 96 bits), and publishes it together with the signed 97-bit goal $\mathit{target} - \mathit{origin}$. The driver sends COMMIT and ALIGN in one CTRL write. Let A be the edge at which the phase path sees ALIGN and the committed configuration, *one clock after COMMIT*:

1. At A, hold φ_s and discard any old candidate, including one due on that edge. Save the requested target.

2. RATE/STOP drain by A+2. The five-register noise scaler samples the new gain at A and produces it at A+4; φ_w consumes the last old FM value at A+4.
3. A+5 captures the settled $\varphi_d + \varphi_w$. A+6 publishes the origin and goal together, using separate add/subtract stages for timing.
4. A+7 prepares the first new slew candidate; A+8 first updates φ_s.

Neither accumulator is paused or reset; a repeated ALIGN restarts the wait. Thus no candidate uses a new target with an old origin. With the rate and noise off, $\varphi_s + \varphi_d + \varphi_w$ reaches the target exactly at the slew rate; terms left running continue to accumulate through and after the move. The driver strobes ALIGN whenever a setting changes `phase` and whenever a whole configuration is applied (presets, reset, `SAFE`); changing only the frequency offset, the noise or the playback keeps the phase accumulated so far.

atomic_clock/src/ac_phase.v

```
reg [2:0] align_wait;
reg [95:0] align_target, align_origin;
assign align_busy = align || align_wait != 0;
always @(posedge clk) begin
    if (!resetn) begin
        origin <= 0; t_eff <= 0;
        align_wait <= 0; align_target <= 0; align_origin <= 0;
    end else if (align) begin
        align_wait <= 6;
        align_target <= target;
    end else if (align_wait != 0) begin
        align_wait <= align_wait - 3'd1;
        if (align_wait == 2) align_origin <= pd + pw;
        if (align_wait == 1) begin
            origin <= align_origin;
            t_eff <= $signed({align_target[95], align_target}) - $signed({align_origin[95],
    ↪ align_origin});
        end
    end else begin
        t_eff <= $signed({target[95], target}) - $signed({origin[95], origin});
    end
end
```

Whenever the applied configuration has playback off, the driver's COMMIT also carries STOP and CLEAR. RESET therefore stops the queued rates synchronously before origin capture, rather than waiting for the worker's next wake-up. The write is:

atomic_clock/control/atomic_clock/driver.py

```
_align_next = False
_stop_next = False

def commit(self):
# ...
    self.write(REG_CTRL, CTRL_COMMIT | (CTRL_ALIGN if self._align_next else 0)
               | (CTRL_STOP | CTRL_CLEAR if self._stop_next else 0))

def _apply_unlocked(self, config):
    # Stop in the commit itself, before ALIGN establishes the new origin. Waiting
    # for the worker would leave a permanent phase error from the queued rates.
    self._stop_next = config["playback"]["source"] == "off"
    try:
        return super()._apply_unlocked(config)
    finally:
        self._stop_next = False
```

5.4 The segment FIFO

Everything slower than the hardware's white noise reaches the phase as a sequence of rates: a 4096-entry FIFO in block RAM holds signed 64-bit rates ν_i; while playing, each is added to R for one *segment* of L samples ($L = f_s/f_{seg}$, 125 000 at the default 1000 segments per second, so the FIFO holds 4.1 s). Segments follow each other on exact sample boundaries. When the FIFO runs empty, the last rate is *held* (the phase stays continuous) and a sticky UNDERFLOW flag is set.

atomic_clock/src/ac_phase.v

```
end else if (playing) begin
    if (remain == 1) begin
        remain <= seg_eff;
        seg_count <= seg_count + 64'd1;
        if (next_ready) begin
            nu_seg <= fetched_nu;
            have_next <= 0;
            taken <= taken + 64'd1;
        end else begin
            underflow <= 1;
        end
    end else begin
        remain <= remain - 32'd1;
    end
end
// ...
nu_sum <= zero_d ? 64'sd0 : rate + nu_seg;          // zeroing drops the increment in flight
pd <= zero_d ? 96'd0 : pd + {{32{nu_sum[63]}}, nu_sum};
pw <= zero_w ? 96'd0 : pw + {{32{fm[63]}}, fm};
b1 <= boundary;
b2 <= b1;
if (zero_d) seg_pd <= 0;
else if (b2) seg_pd <= pd;
if (b2) begin
    lat_seg <= seg_count; lat_taken <= taken; lat_nu <= nu_seg;
end

// phi = phi_s + phi_d + phi_w + w_pm (unwrapped)
s1 <= ps[95:0] + pd;
s2 <= s1 + pw;
phi_tot <= s2 + {{32{pm[63]}}, pm};
```

The essential property is an *integer invariant*. Writing P_i for φ_d at the first sample of segment i (the register `seg_pd`, latched together with the segment's index, the number of FIFO entries taken

and its rate),

$$P_{i+1} = P_i + L\,(R + \nu_i) \tag{25}$$

holds exactly modulo 2^{96}, with no rounding anywhere. Software can therefore predict φ_d at every segment start without error, and check the prediction against a snapshot (CTRL bit 5 captures all these registers on one clock edge). The last three lines form the total φ; the low 64 bits go to the carrier adder, all 96 bits to the PPS. A BRAM fetch finishing on the boundary is forwarded into that segment; the first segment need not wait for a later boundary. A new underflow or overflow on the same edge as CLEAR_FLAGS keeps its sticky flag set.

5.5 A coherent snapshot

CTRL bit 5 requests a snapshot of the phase path, sample count and PPS. During ALIGN the register block holds the request until the origin is published. One common strobe then captures all three on the same edge, at least one clock after the CTRL write (a SNAP written with other strobes sees their effect). A read whose address is accepted after the SNAP write waits until the capture is complete; AXI's read and write channels are independent, so software reads after the write's response:

atomic_clock/src/ac_regs.v

```
if (snap) s_count <= count;           // the same edge as the phase path's snapshot
// A snapshot requested during ALIGN waits for the settled origin. One
// common strobe still captures phase, PPS and sample count together.
if (snap_pending && !align_busy && !align_write) begin
    snap <= 1;
    snap_pending <= 0;
end
// ...
if (wdata[5]) snap_pending <= 1;
// ...
end else if (rd_pending && !snap_pending && !snap) begin
    // Even an immediate AXI read after SNAP waits for all snapshot
    // registers, including a snapshot deferred by ALIGN, to be stable.
    rd_pending <= 0;
    rvalid <= 1;
```

The phase snapshot is modular, including the low 96 bits of the 97-bit static phase:

atomic_clock/src/ac_phase.v

```
if (snap) begin
    s_seg <= lat_seg; s_taken <= lat_taken; s_remain <= remain;
    s_status <= {overflow, underflow, playing, level};
    s_nu <= lat_nu; s_seg_pd <= seg_pd; s_pd <= pd; s_ps <= ps[95:0]; s_pw <= pw; s_origin <= origin;
    ↪
end
```

How it is verified

`tests/tb_phase.sv`: step mode and the slew limit in both directions, landing exactly; φ_d checked clock by clock; FIFO entries consumed in order on the segment grid, hold-last and underflow, overflow at 4096, clear; the invariant (25) at 47 segment starts; snapshots consistent. `tests/tb_system.sv`: the same through the AXI registers. On the board, test T4 of the testing manual: an offset of 10^{-9} measured to $1.6 \cdot 10^{-15}$, a 937.5 ps step measured as 934.1 ps.

6 Hardware white noise

6.1 Uniform random numbers: xoroshiro128+

Five generators run in the FPGA, each producing a 64-bit pseudo-random word per clock. They are *xoroshiro128+* generators of Blackman and Vigna [23]: a 128-bit state (s_0, s_1) updated by XOR,

shift and rotate (an $\mathbb{F}_2$-linear recurrence of period $2^{128}-1$), and the output scrambled by a 64-bit addition. In hardware the update is wiring plus XOR gates and the addition is one carry chain:

atomic_clock/src/ac_noise.v

```
reg reseed_r;
always @(posedge clk) reseed_r <= reseed;
always @(posedge clk) begin
    if (reseed_r) begin
        s0 <= seed ^ K0;
        s1 <= {seed[31:0], seed[63:32]} ^ K1 | 64'd1;    // never the all-zero state
        r <= 0;
    end else begin
        r <= s0 + s1;
        s0 <= {s0[39:0], s0[63:40]} ^ t ^ (t << 16);     // rotl(s0, 24) ^ t ^ (t << 16)
        s1 <= {t[26:0], t[63:27]};                       // rotl(t, 37)
    end
```

All generators are seeded from the one 64-bit `seed` XOR-ed with different constants K (odd 64-bit constants of the kind used in hash mixers), so the same seed reproduces the same noise after each commit that sets it and at power-up (each generator restarts one clock after the `reseed` strobe, from its own copy of it, which keeps the strobe's routing short), while the generators are mutually independent for practical purposes. The lowest bits of xoroshiro128+ are its weakest [23]; the design uses 16-bit fields across the whole word, which is uncritical for noise that is summed and scaled (the measured correlations are below 10^{-3}, see the box below).

6.2 Gaussian-like values: a sum of eight uniforms

Each noise value is the sum of eight uniform 17-bit values of the form $u = 2r + 1$, r a 16-bit field (two generators supply 128 bits per clock):

atomic_clock/src/ac_noise.v

```
// u = 2r + 1 for each 16-bit r: odd, so symmetric about zero.
wire signed [16:0] u [0:7];
genvar g;
generate for (g = 0; g < 8; g = g + 1) begin : odd
    assign u[g] = $signed({r[16 * g +: 16], 1'b1});
end endgenerate
reg signed [17:0] a0, a1, a2, a3;
reg signed [18:0] b0, b1;
always @(posedge clk) begin
    a0 <= u[0] + u[1]; a1 <= u[2] + u[3]; a2 <= u[4] + u[5]; a3 <= u[6] + u[7];
    b0 <= a0 + a1; b1 <= a2 + a3;
    n <= b0 + b1;
```

Read as a signed 17-bit number, $\{r, 1\}$ is uniform on the $M = 2^{16}$ odd integers from -65535 to 65535: symmetric, so the *mean is exactly zero* (a plain two's-complement field would have mean $-1/2$, a bias that the white-FM random walk would integrate into a frequency offset). Its variance is $(M^2-1)/3$, so the sum of eight has

$$\sigma_N = \sqrt{8\,(2^{32}-1)/3} = 107\,019, \qquad |N| \le 8 \cdot 65535 = 4.90\,\sigma_N, \tag{26}$$

and an excess kurtosis of $-\frac{6}{5}\cdot\frac{1}{8} = -0.15$ (that of one uniform, divided by the number of terms). This is the *central-limit* (or Irwin–Hall) method, the oldest and cheapest way to make Gaussian-like numbers, and a common choice in FPGAs [24]. Its distribution is very close to Gaussian in the core but has no tails beyond 4.9σ. For clock simulation this is harmless: Allan-type statistics depend on second moments, which are exact, and the deviation from Gaussian shows only in events rarer than 10^{-6} per sample.

6.3 Scaling to phase units

A value N is scaled to the phase word by a 32-bit gain g and a power of two:

$$w = \mathrm{round}(N \cdot g \cdot 2^{e}/2^{16})\ \mathrm{LSB}, \qquad \sigma_w = \sigma_N\, g\, 2^{e-16}\ \mathrm{LSB}, \tag{27}$$

rounded half away from zero, so that the rounding is odd-symmetric and the mean stays exactly zero (rounding ties up would bias white FM by a quarter LSB per sample), and saturated at $\pm(2^{63}-1)$ instead of wrapping:

atomic_clock/src/ac_noise.v

```
n1 <= n; g1 <= gain; e1 <= exp;
p2 <= n1 * $signed({1'b0, g1}); e2 <= e1;
p3 <= p2; e3 <= e2;
// round half away from zero (symmetric: the mean stays exactly zero)
wide <= ($signed(p3) <<< e3) + (p3[52] ? 96'sd32767 : 96'sd32768);
// Symmetric limits: -2^63 itself fits in signed 64 bits but is outside
// +-(2^63 - 1), so it too must clamp to -2^63 + 1.
if (wide[95:79] != {17{wide[95]}} || wide[79:16] == 64'h8000000000000000)
    out <= wide[95] ? 64'sh8000000000000001 : 64'sh7fffffffffffffff;
else out <= wide[79:16];
```

The driver (`noise.encode_gain`) picks the exponent so that $g \in [2^{31}, 2^{32})$: the relative resolution of the noise level is then better than $5 \cdot 10^{-10}$. $g = 0$ gives exactly zero. Even the representable value -2^{63} is clamped, to preserve symmetric limits. Gain/exponent changes take five scaler registers; ALIGN waits for their old FM values to drain before establishing its origin. The largest supported level, 0.05 cycle per sample, keeps the 4.9σ tails (0.245 cycle) inside the signed 64-bit word.

6.4 White PM and white FM: from h to σ

White PM. A new independent value w_{pm} with standard deviation σ (cycles) is added to φ every sample. A white sequence of variance $(2\pi\sigma)^2\,\mathrm{rad}^2$ sampled at f_s has the one-sided PSD $2(2\pi\sigma)^2/f_s$, flat up to $f_s/2$. Equating it to $S_\phi = h_2 f_0^2$ (Eq. 2 with $S_y = h_2 f^2$):

$$\sigma_{\mathrm{pm}} = \frac{f_0}{2\pi}\sqrt{\frac{h_2 f_s}{2}}\ \mathrm{cycles}. \tag{28}$$

Example: a floor $\mathcal{L} = -140\,\mathrm{dBc/Hz}$ is $S_\phi = 2 \cdot 10^{-14}$, $h_2 = 2 \cdot 10^{-28}$, $\sigma = 1.78 \cdot 10^{-4}$ cycle (17.8 ps). The 0.05-cycle limit is $h_2 \le 1.579 \times 10^{-23}$ ($-91.03\,\mathrm{dBc/Hz}$ in the phase-PSD convention). The digital phase process is white up to $f_s/2 = 62.5\,\mathrm{MHz}$; its unfiltered Allan-deviation formulas use $f_h = 62.5\,\mathrm{MHz}$. This is not a promise of flat analog RF sidebands through that offset: the carrier is only 10 MHz, the phase-to-sine conversion is nonlinear, and the DAC's hold and output filter also matter (§2.9, §9).

White FM. A new value δ with standard deviation σ_δ is added to the accumulator φ_w every sample, so φ_w is a random walk. The frequency during sample n is $y_n = \delta_n f_s/f_0$, a white sequence of variance $(\sigma_\delta f_s/f_0)^2$ and one-sided PSD $2(\sigma_\delta f_s/f_0)^2/f_s = h_0$:

$$\sigma_{\mathrm{fm}} = \frac{f_0}{f_s}\sqrt{\frac{h_0 f_s}{2}} = 0.08\sqrt{\frac{h_0 f_s}{2}}\ \mathrm{cycles}, \qquad \sigma_y(\tau) = \sqrt{h_0/2\tau}. \tag{29}$$

Example: $\sigma_y(1\,\mathrm{s}) = 10^{-11}$ is $h_0 = 2 \cdot 10^{-22}$, $\sigma_{\mathrm{fm}} = 8.9 \cdot 10^{-9}$ cycle per sample. Because δ has mean exactly zero and φ_w is a 96-bit integer, the random walk has no bias and no rounding drift.

How it is verified

`tests/tb_noise.sv`, 200 000 variates: standard deviation within 0.1 % of 107 019, kurtosis 2.859 (expected 2.85), lag-1 and PM–FM correlations below 0.0012, the dither within $\pm 2^{19}$ with the triangular standard deviation, the scaling bit-exact, the seed reproducing the sequence. On the board (testing manual, tests

T1 and T3): white PM and white FM fitted at 0.98–1.03 of the setting.

7 Played-back noise

Flicker noise, random-walk FM, drift, a slow sinusoidal modulation and recorded data all need memory over time scales up to years. They are computed by the *playback worker* in the board's control service and reach the hardware through the segment FIFO. This section explains the two halves of that job: turning a target phase trajectory into FIFO rates without error, and generating the trajectory.

7.1 Exact mirroring: rates from a target trajectory

Let $p(t)$ be the wanted phase trajectory in LSB, sampled at the segment boundaries $t_n = nL/f_s$. The worker keeps an exact copy P_n of the hardware's φ_d at the start of segment n, as a Python integer (unbounded), and chooses

$$\nu_n = \mathrm{round}\left(\frac{B_{n+1} + p(t_{n+1}) - P_n}{L}\right) - R, \qquad B_n = B + nLR, \tag{30}$$

where B is a constant baseline (below) and B_n moves with the frequency offset, so that ν carries the trajectory alone and the hardware's $R + \nu$ adds the two. By the invariant (25),

$$P_{n+1} - (B_{n+1} + p(t_{n+1})) = L\ \mathrm{round}(z) - Lz, \qquad z = \frac{B_{n+1} + p(t_{n+1}) - P_n}{L}, \tag{31}$$

so $|P_{n+1} - B_{n+1} - p(t_{n+1})| \le L/2$ LSB at *every* boundary, for every n. The error at boundary $n+1$ is fixed by the target at $n+1$ alone; the errors of earlier segments are absorbed because each ν_n is computed from the realized P_n, not from the previous target. Rounding errors *never accumulate*, however long the run. At 1000 segments per second, $L/2 = 62\,500$ LSB $= 3.4 \cdot 10^{-15}$ cycle ($3.4 \cdot 10^{-22}$ s). Compare a naive scheme that rounds each segment's frequency independently: its errors would add up as a random walk.

Between boundaries the phase is linear (the rate is constant during a segment). The realized trajectory is therefore $p(t)$ sampled at f_{seg} and linearly interpolated; equivalently, the frequency is held constant over each segment, a zero-order hold on y that multiplies the played spectrum by $\mathrm{sinc}^2(f/f_{\mathrm{seg}})$. Software-made noise thus ends at about $f_{\mathrm{seg}}/2$ (500 Hz by default); above that the FPGA's white PM and white FM take over, as in real clocks, whose high-offset phase noise is dominated by white PM.

atomic_clock/control/atomic_clock/playback.py

```
def _round_div(a, b):
    """round(a / b) for integers, halves away from zero (b > 0)."""
    q, r = divmod(abs(a), b)
    q += 2 * r >= b
    return q if a >= 0 else -q
# ...
    def _entry(self):
        start = (self.next_entry, self.P, self.target, self.traj.time(), self.B)
        # the baseline moves with the frequency offset (RATE): nu carries the trajectory only
        self.B += self.L * self.rate
        target = self.target = self.B + self.traj.next()
        nu = _round_div(target - self.P, self.L) - self.rate
# ...
        self.P += self.L * (self.rate + nu)
        self.next_entry += 1
        return nu
```

The hardware adds $R + \nu$ in signed 64 bits, so ν is limited to keep that sum in range (an event is logged if a trajectory ever demands it); the settings are validated so that a playable trajectory never does (a sinusoidal PM or a replay may not exceed the 10^{-3} frequency limit).

Baseline, start and resynchronization. When playback starts, the worker stops and clears the FIFO, pushes about 1 s of zero-rate *warm-up* segments and starts playing. Once a snapshot shows the warm-up running, it knows the hardware's φ_d at a segment start and the number of segments still to go, and computes the P at which the trajectory's first segment will start. The baseline is then $B = P - p(t_0)$ (counting n from that segment): the trajectory begins at the present phase, so starting never makes the output jump. A change of the frequency offset (a new R) needs no restart: the queued ν carry the trajectory alone, so the worker only re-derives the mirror's P and B for the segments not yet started from a snapshot (the hardware's φ_d now and the current segment's remaining samples), and playback goes on. The full procedure *resynchronizes* after an underflow (while the FIFO was empty the hardware held the last rate, so φ_d continued smoothly but no longer as planned), or if a periodic check finds the hardware's latched P_i different from the mirror by even one LSB; the trajectory then continues from the present phase with a new B. Every event is logged. Trajectory preparation can take seconds on the board. Before starting it, the worker rechecks the current model, rate and segment length under the transaction lock, so a reset or edit accepted during preparation supersedes that start.

Timing on the CPU. The worker wakes every 50 ms. Each step it refills the FIFO to 3.5 s of content, 128 entries at a time and at most 0.25 s of computing per step, so the FIFO is fed while it computes; the 3.5 s ride out stalls of the service (an SD-card write can block for seconds; log files and checkpoints are therefore written by a background thread). At least every 10 s it compares its mirror with the snapshot, and every `log_interval` (10 s) it logs P_i and the target; every minute it saves the complete model state to a checkpoint, from which a reboot resumes the same trajectory. With model playback the service uses about 40 % of one CPU core (measured).

7.2 Flicker noise as a sum of first-order sections

A $1/f$ spectrum cannot be produced by a finite linear system, but over a finite band it can be approximated arbitrarily well by a sum of first-order low-pass (relaxation) processes with geometrically spaced corner frequencies. This is the classic method of Barnes and Jarvis for clock noise [25]; the physical picture (a superposition of relaxation processes with a $1/\tau$ distribution of time constants) is reviewed by Keshner [26], and Kasdin [27] surveys this and the other discrete methods for $1/f^\alpha$ noise.

Each section is a first-order autoregressive (AR(1)) process, the discrete form of an Ornstein–Uhlenbeck process, updated once per segment of length $\Delta t = 1/f_{\mathrm{seg}}$:

$$u_{k,n+1} = a_k u_{k,n} + s_k e_{k,n}, \qquad a_k = e^{-2\pi f_k \Delta t}, \tag{32}$$

with independent standard normal $e_{k,n}$. Its one-sided spectrum is close to a Lorentzian with plateau $S_k(0)$ and corner f_k. Choose the plateau K/f_k for all sections, with corners $f_k = f_{\mathrm{top}}\, r^{-k}$. For f well inside the band, the sum over the geometric ladder is a Riemann sum in $\ln f_k$ with spacing $\ln r$:

$$\sum_k \frac{K/f_k}{1 + (f/f_k)^2} \approx \frac{K}{\ln r} \int_0^\infty \frac{\mathrm{d}g}{g}\, \frac{1/g}{1 + f^2/g^2} = \frac{K}{\ln r} \int_0^\infty \frac{\mathrm{d}g}{g^2 + f^2} = \frac{K}{\ln r}\, \frac{\pi}{2f}. \tag{33}$$

Setting this equal to the wanted level h/f gives the section weight

$$K = \frac{2h \ln r}{\pi}. \tag{34}$$

The discrete section's low-frequency plateau is $2\Delta t\, s_k^2/(1 - a_k)^2$; setting it to K/f_k gives the noise amplitude $s_k = (1 - a_k)\sqrt{K/(2\Delta t f_k)}$. The design uses two sections per decade ($r = \sqrt{10}$), corners from $f_{\mathrm{seg}}/2$ down to below 10^{-8} Hz (about three years): 23 sections at the default rate. A numerical evaluation of the sum shows a ripple of only ± 0.005 dB about h/f inside the band; the spectrum flattens below 10^{-8} Hz (invisible in any run shorter than years) and falls off above $f_{\mathrm{seg}}/2$. Each section starts in its stationary state (variance $s_k^2/(1 - a_k^2)$), so there is no start-up transient even for the three-year section.

atomic_clock/control/atomic_clock/playback.py

```
r = 10 ** (1 / SECTIONS_PER_DECADE)
f_high = 0.5 / dt
corners, f = [], f_high
while f > F_LOW / r:
    corners.append(f)
    f /= r
k = 2 * level * math.log(r) / math.pi
self.a = np.array([math.exp(-2 * math.pi * fk * dt) for fk in corners])
self.s = (1 - self.a) * np.sqrt(k / (2 * dt * np.array(corners)))
if state is not None:
    self.u = np.array(state, float)
else:                                       # start stationary
    self.u = rng.standard_normal(len(self.a)) * self.s / np.sqrt(1 - self.a ** 2)
# ...
noise = rng.standard_normal((count, len(self.a))) * self.s
if lfilter is not None:                     # one filter call per section
    out = np.zeros(count)
    for k, a in enumerate(self.a):
        y, zi = lfilter([1.0], [1.0, -a], noise[:, k], zi=[a * self.u[k]])
        out += y
        self.u[k] = y[-1]
    return out
```

Flicker FM and flicker PM. For flicker FM the sum is the fractional frequency y with $S_y = h_{-1}/f$ (level $h = h_{-1}$); it is integrated into the phase, $y\, f_0 \Delta t$ cycles per segment. For flicker PM the sum is the phase itself, in cycles, with $S = h_1 f_0^2/((2\pi)^2 f)$ (from $S_\phi = (f_0/f)^2 h_1 f$ in rad^2/Hz).

Speed. A pure-Python loop over 23 sections at 1000 segments per second was too slow for the board's CPU. The sections are therefore advanced in blocks of 256 segments, one call of scipy's `lfilter` (a C implementation of the recursion) per section and block, about 20 times faster. The random numbers come from numpy's PCG64 generator [28], seeded with the same `seed` as the hardware; the checkpoint stores the generator state and the filter states at the start of the current block plus the position in it, so a resumed trajectory regenerates the block and continues bit for bit.

Why not Kasdin–Walter or Timmer–König? The usual way to simulate power-law clock noise is the method of Kasdin and Walter [29, 27]: white noise convolved with the fractional-integration filter $h_0 = 1$, $h_k = h_{k-1}(k - 1 + \alpha/2)/k$, which gives an exact discrete $1/f^\alpha$ spectrum for any α, including non-integer values. The other common method, by Timmer and König [30], draws Gaussian Fourier amplitudes with variance proportional to $S(f)$ and transforms them back to time; the spectrum is exact on the record's frequency grid. Both build a record of a length fixed in advance. The simulator needs an unbounded stream on a small CPU, which decides the choice (Table 13).

A stream can be made from either finite method by joining blocks, as the generator `sumi` [31] does with Timmer–König blocks cross-faded into each other. But successive blocks are independent, so the $1/f$ spectrum ends near the block rate: about six decades for blocks of 2^{20} samples, while a clock simulation needs eleven (10^{-8} Hz to 500 Hz), that is blocks of some 10^{11} samples.

Both finite methods remain in use for what they do best. Kasdin–Walter is the standard for finite test records in frequency-stability work (allantools [32], for example, includes it); Timmer–König is the other common choice and makes the synthetic recordings of this design's host tools. For long streams, sums of first-order filters are the established method [33]. Clock models in timescale and navigation work are usually stochastic differential equations for white and random-walk FM [34], an exact state-space form to which flicker is added by approximations of this kind, or left out.

	Kasdin–Walter filter	Timmer–König (FFT)	sum of first-order sections (here)
spectrum	exact discrete $1/f^{\alpha}$ down to $1/(N\Delta t)$, any α	exact on the grid $k/(N\Delta t)$, any $S(f)$	$1/f$ within ± 0.005 dB from 10^{-8} Hz to $f_{\text{seg}}/2$, flat below
length	N fixed in advance	N fixed; the record is periodic	unbounded (a year at 1 kHz is $3 \cdot 10^{10}$ segments)
cost, memory	$O(N \log N)$ by FFT; $O(N)$ memory	$O(N \log N)$; $O(N)$ memory	23 multiply-adds per segment; 23 numbers of state
start	from zero: non-stationary, like a clock just switched on	stationary	stationary, like a clock that has run for years
as a stream	blocks; nothing below the block rate	blocks; nothing below the block rate	continuous
after a reboot	needs the whole past	needs the whole record	checkpoint of 23 numbers and the generator state

Table 13: Three methods for flicker noise. White noise and random walk need none of them: they are exact recursions.

7.3 Random walk, drift, sinusoidal modulation and replay

Random-walk FM. y takes a Gaussian step of standard deviation $q^{1/2}$ every segment. A continuous random walk with increment variance q per Δt has the one-sided spectrum $S_y = q/(2\pi^2 \Delta t\, f^2)$; equating to h_{-2}/f^2 gives $\sqrt{q} = 2\pi\sqrt{h_{-2}\Delta t/2}$.

Drift. A linear frequency drift D per day, $y = Dt/86\,400\,\text{s}$, is the phase $f_0 Dt^2/(2 \cdot 86\,400\,\text{s})$ cycles, an explicit function of time.

Sinusoidal PM is another explicit function, amplitude (cycles)$\cdot \sin 2\pi f_m t$, with f_m at most $f_{\text{seg}}/10$. A daily temperature cycle of a crystal, $y = A\sin(2\pi t/\text{day})$, is a sinusoidal PM of amplitude $f_0 A/(2\pi/\text{day})$.

Replay. A recorded time error $x(t)$ (seconds, two columns) is interpolated linearly at the segment boundaries and multiplied by f_0 and a scale factor. A looped record carries the end-to-start difference over to the next lap, so the phase stays continuous.

Assembly. At each boundary the worker adds the integrated frequency terms (flicker FM and random walk, accumulated as an integer number of LSB), the explicit functions of time, the flicker-PM value and the replay:

atomic_clock/control/atomic_clock/playback.py

```
def next(self):
    """Advance one segment; the target at the new boundary."""
    if self._index == BLOCK:
        self._block()
    i = self._index
    y = 0.0
    if self._ffm is not None:
        y += self._ffm[i]
    if self._rw is not None:
        y += self.y_rw
        self.y_rw += self._rw[i]
    if y:
        self.acc += round(y * F0 * self.dt * CYCLE)
    if self._fpm is not None:
        self._fpm_value = self._fpm[i]
    self._index = i + 1
    self.n += 1
```

```
    return self.value()
```

The floating-point parts are converted to integers at each boundary; their rounding ($\leq \frac{1}{2}$ LSB, $2.7 \cdot 10^{-20}$ cycle) is irrelevant next to any noise level a clock can have.

How it is verified

`tests/test_playback.py` (with `tests/fake_core.py`, a register-level model of the core): Allan deviation of flicker FM, random-walk FM and flicker PM within 10 % of the formulas from 10 to 100 ms over 60 s records; drift and sine exact; exact mirroring (every predicted segment start equal to the model's, within $L/2$ of the target) through starts, a 3 s stall with underflow, a frequency-offset change, a model change and `SAFE`; seeded reproducibility and resume from a checkpoint. On the board: flicker PM (test T2) fitted at 1.00–1.07 of the setting, the sinusoidal PM (T8) at 1.02–1.05.

8 From phase to sine

The top 28 bits p of θ (resolution 2^{-28} cycle = 0.37 fs at 10 MHz) address a 4096-entry full-wave table $S[i] = \text{round}\left((2^{17}-1)\sin(2\pi i/4096)\right)$ in block RAM, and the 16-bit remainder f interpolates linearly between neighbours:

$$y = S[i] + \text{round}\big((S[i+1] - S[i])\, f/2^{16}\big), \qquad i = p_{27:16},\ f = p_{15:0}. \tag{35}$$

This is the *phase-to-amplitude converter* of a DDS, here with linear interpolation [21]. The interpolation error of a function with second derivative bounded by M_2 over an interval h is at most $M_2 h^2/8$; for the sine, $h = 2\pi/4096$ and $M_2 = 1$ (relative to full scale), so

$$|\Delta y| \leq \frac{(2\pi/4096)^2}{8} = 2.9 \cdot 10^{-7} \text{ of full scale}, \tag{36}$$

below the table's own rounding of $\frac{1}{2}$ LSB of 2^{17} ($3.8 \cdot 10^{-6}$). The result is within one LSB of $(2^{17}-1)\sin(2\pi p/2^{28})$, i.e. $7.6 \cdot 10^{-6}$ of full scale, sixteen times finer than one DAC step ($1.2 \cdot 10^{-4}$). The table is generated by `tests/test_control.py` into `src/ac_sine_rom.v`.

atomic_clock/src/ac_sine.v

```
addr_a <= phase[27:16];                      // 1
addr_b <= phase[27:16] + 12'd1;
f1 <= phase[15:0];
f2 <= f1;                                    // 2, 3: table read
f3 <= f2;
s0_4 <= s_a;                                 // 4
d4 <= s_b - s_a;
f4 <= f3;
m5 <= d4 * $signed({1'b0, f4});              // 5
s0_5 <= s0_4;
y <= s0_5 + ((m5 + 36'sd32768) >>> 16);      // 6: round half up
```

How it is verified

`tests/tb_sine.sv` checks 112 282 phases (every table point, the midpoints and random phases): worst error 0.999 LSB of 2^{17}.

9 The output stage

9.1 Scaling and rounding to the DAC

The DAC is 14 bits wide, driven by a 16-bit stream whose two lowest bits it ignores. The output stage computes directly on the DAC grid:

$$q = \text{sat}_{14}\left(\left\lfloor \frac{y\,A + \mathit{offset} \cdot 2^{17} + 2^{18} + d}{2^{19}} \right\rfloor\right), \qquad \text{stream} = 4q, \tag{37}$$

with y the sine ($\pm(2^{17}-1)$), A the amplitude and *offset* in 16-bit stream codes (nominally 32 768 per volt), and d the optional dither. Dividing yA by 2^{17} gives stream codes, and a further factor 4 gives DAC steps: hence 2^{19}. Adding 2^{18} (half a step) before the floor makes it a rounding (half up); saturation is at the DAC limits.

atomic_clock/src/ac_out.v

```
product <= y * $signed({1'b0, amp});                                    // 1
offset1 <= offset;
enable1 <= enable;
dither1 <= dither;
sum <= (enable1 ? product : 35'sd0) + ($signed(offset1) <<< 17) + 37'sd262144 + dither1;   // 2
if (sum >>> 19 > 37'sd8191) out <= 16'sh7ffc;                           // 3
else if (sum >>> 19 < -37'sd8192) out <= 16'sh8000;
else out <= {sum[32:19], 2'b00};
```

Volts. The board calibration, shared by all designs, gives each output as $V = G \cdot \text{code} + B$ (`common/CALIBRATION.md`). The driver sets $A = \text{round}\,(V_{\text{amp}}/(G(1-2^{-17})))$ and places the offset on the DAC grid, $\mathit{offset} = 4\,\text{round}\,((V_{\text{off}} - B)/4G)$, so that the DC level is the same with the sine on or off. Settings whose peak would leave the DAC range are refused.

9.2 What 14 bits can resolve

At the zero crossing one DAC step corresponds to a phase of $1/(2\pi A_{\text{DAC}})$ cycle, where A_{DAC} is the amplitude in DAC steps: $3.9 \cdot 10^{-5}$ cycle (3.9 ps) at 0.5 V nominal ($A_{\text{DAC}} = 4096$). A phase meter averages many samples and resolves far finer phase changes on average, but without dither the rounding is *deterministic*: the carrier repeats every 25 samples, so for a fixed φ the rounding errors form a fixed 25-sample pattern. Its consequences:

- **Spurs instead of a noise floor.** The quantization error is a periodic signal: lines at multiples of $f_s/25 = 5$ MHz, estimated at about -90 dBc each at 0.5 V (not measured).
- **A nonlinear phase.** The pattern, and hence the averaged phase of the output, depends on φ: the averaged phase deviates from φ by a periodic function of φ of order 10^{-5} cycle (about 1 ps). A static phase sees only a constant offset, but a moving phase sweeps through this nonlinearity and gains short-term error. Measured on the board without dither: ADEV up to 1.5× the expected value for white FM and 3× for a sinusoidal PM at $\tau = 0.01$–2 s (testing manual).

9.3 Dither

The cure is the textbook one: add a random *dither* before the quantizer. With *triangular* (TPDF) dither spanning ±1 step, the sum of two independent uniform values, the first and second moments of the total error become independent of the signal (Lipshitz, Wannamaker and Vanderkooy [35]). In particular the *average* output equals the input exactly, however finely the input moves, and the error power is constant, $\Delta^2/4$ (the quantizer's $\Delta^2/12$ plus the dither's $\Delta^2/6$). The design makes the dither from two odd 19-bit values of one generator word (odd, as in the noise generator, so that each is symmetric and the mean is exactly zero):

atomic_clock/src/ac_noise.v

```
// two odd 19-bit values (mean exactly zero): triangular over +-(2^19 - 2)
always @(posedge clk) dither <= $signed({rd[17:0], 1'b1}) + $signed({rd[49:32], 1'b1});
```

In the sum of `ac_out`, 2^{19} is one DAC step, so this is TPDF dither over ±1 step. The price is a white noise floor. Additive white noise of one-sided density N_0 on a carrier of amplitude A gives $\mathcal{L} = N_0/A^2$ (half of it is phase noise); with $N_0 = 2\sigma^2/f_s$, $\sigma^2 = 1/4$ step^2 and $A = 4096$ steps,

$$\mathcal{L}_{\text{dither}} = \frac{1}{2 f_s A_{\text{DAC}}^2} \approx 2.4 \cdot 10^{-16}\,\text{Hz}^{-1}, \text{ i.e. } -156 \text{ dBc/Hz} \quad \text{at 0.5 V nominal}, \tag{38}$$

a digital estimate; the board's analog noise is not measured and probably higher. The dither is off at power-up and should be on whenever OUT1's phase moves; with it, the measured ADEV ratios above become 1.00.

9.4 The DAC's hold

The DAC holds each value for 8 ns, a zero-order hold with response $\mathrm{sinc}(f/f_s)$: the 10 MHz line is 0.09 dB low, and images appear at $f_s \pm f_0 = 115$ and 135 MHz, at about -21 dBc before the board's analog output filter (attenuation there not measured) [20]. For a clean spectrum, add an external low-pass filter (cut-off 20–30 MHz).

How it is verified

`tests/tb_system.sv`: OUT1 and OUT2 bit-exact at $\varphi = 0$, the dither only on OUT1. On the board: tests T3 and T8 of the testing manual with and without dither (figure "dither").

10 PPS outputs and time of day

10.1 When does the simulated clock's second end?

The reference PPS (DIO2_P) marks every $125 \cdot 10^6$ samples of the board clock. The simulated clock (OUT1) has completed $0.08\,n + \varphi(n)$ cycles after n samples, so its k-th second ends when that reaches $10^7 k$:

$$0.08\,n + \varphi(n) = 10^7 k \iff S(n) \equiv n + 12.5\,\varphi(n) = 125 \cdot 10^6\,k \qquad (\varphi \text{ in cycles, } S \text{ in samples}). \quad (39)$$

This is why φ is unwrapped: one cycle of phase is 12.5 samples (100 ns) of PPS. The hardware computes S every sample from the 96-bit φ, with $12.5\varphi = 8\varphi + 4\varphi + \varphi/2$ (shifts and adds, exact except the last bit of $\varphi/2$), and starts a pulse on the first sample with $S \geq$ the next second, then advances that second:

atomic_clock/src/ac_pps.v

```
// S = n + 12.5 phi: 12.5 phi = 8 phi + 4 phi + phi / 2 (units 2^-64 sample)
n <= n + 64'd1;
n1 <= n;
a1 <= ($signed(phi_tot) <<< 3) + ($signed(phi_tot) <<< 2);
h1 <= $signed(phi_tot) >>> 1;
n2 <= n1;
a2 <= a1 + h1;
n3 <= n2;
n4 <= n3;
// The carrier pairs sample j with phi of the clock after it (ac_carrier), so
// S(j) takes n one clock later than phi: n3 here belongs to the phi in a2.
s3 <= $signed(n3) + $signed(a2[99:64]);          // whole samples (floor)
f3 <= a2[63:0];

// Simulated clock
if (arm) begin
    next <= epoch; count <= 0; left <= 0;
end else if ($signed(s3) >= $signed(next)) begin
    next <= next + SECOND;
    count <= count + 32'd1;
    left <= w;                                   // a merged pulse stays high
    if (left == 0) begin
        fire_n <= n4; fire_frac <= f3;
        fire_over <= over[63:32] != 0 ? 32'hffffffff : over[31:0];
```

Comparing against a *target that only advances* makes the output monotonic: phase noise that moves S back and forth across a second cannot produce a second pulse, and a jump over several seconds produces one merged pulse while counting every second. The ideal edge-time shift is $-\varphi/f_0$,

rounded to the 8 ns sample grid. For a constant $\varphi = +1$ cycle, the pulse advances **96 ns**, with a **fixed +4 ns remainder** from the ideal 100 ns advance. This repeats every second and does not average away. For a constant phase the remainder is in $[0, 8\,\text{ns})$; during jumps it may exceed one sample. The latest rising edge's unsigned whole-sample remainder (saturated at $2^{32} - 1$) and 2^{-64} fraction are captured for software. $S(n)$ uses the same φ as the carrier sample n (the carrier adds the phase of the following clock, so the index is taken one clock later). Both pulses are delayed inside the FPGA (`DELAY` = 9 clocks, the simulated one a clock less 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 remain unmeasured.

10.2 Time of day

The reference second normally starts when the design starts. To put it on a UTC second, the driver uses the board's system clock, kept on UTC by NTP [36]: it reads the 64-bit sample counter between two readings of the system clock (integer nanoseconds), picks the UTC second U_0 at least 1.5 s ahead, computes

$$epoch = n + \text{round}\left((U_0 - t) \cdot 125\,\text{MHz}\right), \tag{40}$$

rounds it to a multiple of 25 samples (200 ns), where the carrier table restarts and OUT2 crosses zero upwards, and arms both PPS on that sample. The accuracy is that of NTP at that moment (milliseconds over the internet) plus the read time (microseconds).

Afterwards the PPS follow the *board's crystal*, not UTC: a crystal error y_b makes them drift by $y_b t$ (20 ppm would be 1.7 s per day). The worker therefore measures the board time minus UTC, $e = (n - epoch) \cdot 8\,\text{ns} - (t - U_0)$, every minute, logs it and fits a line: its slope is the board crystal's frequency offset against UTC, limited by NTP's jitter (milliseconds per span). The simulated clock reading minus UTC is $e + \varphi/f_0$. Its PPS edge-time error has the opposite sign, plus the sample-grid remainder.

How it is verified

`tests/tb_pps.sv`: the reference every $125 \cdot 10^6$ samples, the simulated pulse with its grid-rounded advance, monotonic under noise, merged pulses on jumps, arming at an epoch (also one in the past), the latched remainder. `scripts/test_hw.py` checks the counts on the board.

11 Numbers and limits

quantity	value
carrier	$0.08\,f_\mathrm{s}$ exactly; periodic phase error $\leq 2^{-65}$ cycle, never accumulating
phase word	96-bit signed, 2^{-64} cycle ($5.4 \cdot 10^{-27}$ s), range $\pm 2^{31}$ cycles (± 214.7 s)
frequency offset	steps of $6.8 \cdot 10^{-19}$, realized within $3.4 \cdot 10^{-19}$, $\lvert y_0 \rvert \leq 10^{-3}$, no drift
phase slew	$0 \ldots 10^{-3}$ (fractional frequency), same resolution; 0 = steps
latency	to the core stream: φ_s/φ_d 13 clocks, φ_w 12, w_pm 11; amplitude/offset 3; ALIGN settling is additional
sine	input 2^{-28} cycle (0.37 fs); value within 1 LSB of 2^{17} ($7.6 \cdot 10^{-6}$ FS)
DAC	14 bits; one step at the zero crossing $3.9 \cdot 10^{-5}$ cycle at 0.5 V
without dither	5 MHz spurs (est. -90 dBc), averaged-phase nonlinearity $\sim 10^{-5}$ cycle
with dither	linear on average; white floor ≈ -156 dBc/Hz at 0.5 V (digital estimate)
hardware noise	white PM and white FM, a new value every 8 ns, to 62.5 MHz offset; $\sigma \leq 0.05$ cycle per sample ($h_2 \leq 1.6 \cdot 10^{-23}$); Gaussian-like, tails at 4.9σ; level resolution $5 \cdot 10^{-10}$
segment FIFO	4096 rates of 64 bit, $L \geq 16$ samples; segment rate $1 \ldots 10\,000$ per second (default 1000); kept 3.5 s full
played-back noise	to $f_\mathrm{seg}/2$ (500 Hz); realized within $L/2$ LSB ($3.4 \cdot 10^{-15}$ cycle at 1 kHz) of the target at every segment start
flicker synthesis	2 sections per decade, 10^{-8} Hz … $f_\mathrm{seg}/2$, ripple ± 0.005 dB
PPS	edge on the 8 ns grid (remainder reported); UTC arming to NTP accuracy (ms); then the board crystal

Table 14: Resolutions, ranges and limits of the generated signal. "Not measured" items (absolute frequency, the board oscillator's noise, analog harmonics and delays) are listed in the README's specifications.

References

[1] IEEE standard definitions of physical quantities for fundamental frequency and time metrology — random instabilities. IEEE Std 1139-2022, 2022. Earlier editions 1988, 1999, 2008.

[2] D. W. Allan. Statistics of atomic frequency standards. *Proc. IEEE*, 54(2):221–230, 1966. The two-sample (Allan) variance.

[3] J. A. Barnes, A. R. Chi, L. S. Cutler, et al. Characterization of frequency stability. *IEEE Trans. Instrum. Meas.*, IM-20(2):105–120, 1971. The power-law model and its time-domain measures.

[4] W. J. Riley. *Handbook of Frequency Stability Analysis*. NIST Special Publication 1065. National Institute of Standards and Technology, 2008. The practical reference for the deviations, their confidence and noise identification.

[5] J. Rutman. Characterization of phase and frequency instabilities in precision frequency sources: fifteen years of progress. *Proc. IEEE*, 66(9):1048–1075, 1978.

[6] J. Levine. Introduction to time and frequency metrology. *Rev. Sci. Instrum.*, 70(6):2567–2596, 1999.

[7] E. Rubiola. *Phase Noise and Frequency Stability in Oscillators*. Cambridge University Press, 2008.

[8] E. Rubiola and F. Vernotte. The companion of Enrico's chart for phase noise and two-sample variances. *IEEE Trans. Microwave Theory Tech.*, 71(7):2996–3025, 2023. A compact modern summary of the relations.

[9] C. A. Greenhall and W. J. Riley. Uncertainty of stability variances based on finite differences. In *Proc. 35th Annual Precise Time and Time Interval (PTTI) Meeting*, 2003. The confidence intervals used in the reports.

[10] SciPy developers. `scipy.signal.welch`: power spectral density and spectrum normalization. https://docs.scipy.org/doc/scipy/reference/generated/scipy.signal.welch.html. Density/spectrum scaling and equivalent noise bandwidth; accessed 17 October 2026.

[11] Microchip Technology. *Chip-Scale Atomic Clock (CSAC) SA65 User's Guide*, 2026. DS50003159E, July 2026, pp. 15–24 and 49–50. Accessed 17 October 2026. https://ww1.microchip.com/downloads/aemDocuments/documents/FTD/ProductDocuments/UserGuides/Chip-Scale-Atomic-Clock-CSAC-SA65-Users-Guide-DS50003159.pdf.

[12] Microchip Technology. *LN-CSAC SA65-LN: Frequently Asked Questions*, 2025. DS00005984A, May 2025. https://ww1.microchip.com/downloads/aemDocuments/documents/FTD/ProductDocuments/Brochures/LN-CSAC-SA65-LN-FAQs-DS00005984.pdf.

[13] Microchip Technology / Vectron. *OX-171: Oven Controlled Crystal Oscillator*, 2019. Printed revision 5-6-19 jar, pp. 1–3 and 6. Accessed 17 October 2026. https://ww1.microchip.com/downloads/aemDocuments/documents/VOP/ProductDocuments/DataSheets/OX-171.pdf.

[14] Microchip Technology / Vectron. *OX-174: Ultra Low Noise Oven Controlled Crystal Oscillator*, 2019. Printed revision 5-6-2019 jar, pp. 1–2 and 6; sine-wave phase-noise code B. Accessed 17 October 2026. https://ww1.microchip.com/downloads/aemDocuments/documents/VOP/ProductDocuments/DataSheets/OX-174.pdf.

[15] Microchip Technology. *CSAC SA65: Chip-Scale Atomic Clock*, 2021. DS00003478B, July 2021, also 900-00744-008 rev D, pp. 2–3. Accessed 17 October 2026. https://ww1.microchip.com/downloads/en/DeviceDoc/00003478.pdf.

[16] Microchip Technology. *LN-CSAC: Low-Noise Chip-Scale Atomic Clock*, 2025. SA65-LN, DS00004687A, January 2025, pp. 2–3. Accessed 17 October 2026. https://ww1.microchip.com/downloads/aemDocuments/documents/FTD/ProductDocuments/Brochures/LN-CSAC-Sell-Sheet-DS00004687.pdf.

[17] Microchip Technology. *MD-015: Disciplined GNSS Atomic Oscillator Module*, 2026. DS00005668B, September 2026, p. 2 holdover table and double-asterisk footnote. Accessed 17 October 2026. https://ww1.microchip.com/downloads/aemDocuments/documents/FTD/ProductDocuments/Brochures/MD-015-Sell-Sheet-DS00005668.pdf.

[18] E. Rubiola. On the measurement of frequency and of its sample variance with high-resolution counters. *Rev. Sci. Instrum.*, 76(5):054703, 2005.

[19] J. Tierney, C. M. Rader, and B. Gold. A digital frequency synthesizer. *IEEE Trans. Audio Electroacoust.*, 19(1):48–57, 1971. The phase-accumulator synthesizer (DDS, NCO).

[20] Analog Devices. A technical tutorial on digital signal synthesis. Analog Devices, Norwood MA, 1999. Tuning equation, truncation spurs, DAC images.

[21] J. Vankka and K. A. I. Halonen. *Direct Digital Synthesizers: Theory, Design and Applications*. Kluwer Academic, Boston, 2001. Phase-to-amplitude conversion, interpolation, spurs.

[22] Analog Devices. Direct digital synthesis (DDS) with a programmable modulus. Application Note AN-953. Exact rational frequencies with a non-binary accumulator modulus.

[23] D. Blackman and S. Vigna. Scrambled linear pseudorandom number generators. *ACM Trans. Math. Softw.*, 47(4):36:1–36:32, 2021. xoroshiro128+ and relatives; arXiv:1805.01407.

[24] D. B. Thomas, W. Luk, P. H. W. Leong, and J. D. Villasenor. Gaussian random number generators. *ACM Comput. Surv.*, 39(4):11, 2007. Survey including the central-limit (sum of uniforms) method and hardware tails.

[25] J. A. Barnes and Jr. S. Jarvis. Efficient numerical and analog modeling of flicker noise processes. NBS Technical Note 604, National Bureau of Standards, 1971. Flicker noise from a cascade of first-order filters; doi:10.6028/NBS.TN.604.

[26] M. S. Keshner. 1/f noise. *Proc. IEEE*, 70(3):212–218, 1982. 1/f spectra as superpositions of relaxation processes.

[27] N. J. Kasdin. Discrete simulation of colored noise and stochastic processes and $1/f^{\alpha}$ power law noise generation. *Proc. IEEE*, 83(5):802–827, 1995.

[28] M. E. O'Neill. PCG: A family of simple fast space-efficient statistically good algorithms for random number generation. Technical Report HMC-CS-2014-0905, Harvey Mudd College, 2014. The PCG64 generator of numpy.

[29] N. J. Kasdin and T. Walter. Discrete simulation of power law noise. In *Proc. 1992 IEEE Frequency Control Symposium*, 1992. The fractional-integration filter for $1/f^{\alpha}$ noise of any α.

[30] J. Timmer and M. König. On generating power law noise. *Astron. Astrophys.*, 300:707–710, 1995. Spectral shaping of random Fourier amplitudes.

[31] R. Žitko. sumi: noise generation tool. https://github.com/rokzitko/sumi. Timmer–König $1/f$ noise in blocks, streamed without end.

[32] A. E. E. Wallin et al. AllanTools. Python library for Allan deviation and related statistics, https://github.com/aewallin/allantools.

[33] S. Plaszczynski. Generating long streams of $1/f^{\alpha}$ noise. *Fluct. Noise Lett.*, 7(1):R1–R13, 2007. Sums of first-order filters for unbounded streams; arXiv:astro-ph/0510081.

[34] L. Galleani, L. Sacerdote, P. Tavella, and C. Zucca. A mathematical model for the atomic clock error. *Metrologia*, 40(3):S257–S264, 2003. Clock error as the solution of stochastic differential equations.

[35] S. P. Lipshitz, R. A. Wannamaker, and J. Vanderkooy. Quantization and dither: a theoretical survey. *J. Audio Eng. Soc.*, 40(5):355–375, 1992. Why triangular (TPDF) dither makes the mean output linear.

[36] D. Mills, J. Martin, J. Burbank, and W. Kasch. Network time protocol version 4: Protocol and algorithms specification. RFC 5905, IETF, 2010.