synth 2.1: reference documentation

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

Rok Zitko, Jozef Stefan Institute

2026-11-10 (commit 6178370)

Contents

synth: a 16 + 16 tone phase-coherent synthesizer **2**
- Before connecting equipment . . . 2
- Quick start . . . 3
- Presets . . . 3
- Specifications . . . 3
- Hardware setup . . . 4
- Documentation . . . 5
- Changing the design . . . 5
- Source layout . . . 5
- Verification . . . 6
- Author . . . 6

synth recipes **6**
- Two outputs with a defined phase difference . . . 6
- Measuring and correcting the OUT1–OUT2 skew . . . 6
- A frequency comb, phases set at t0 . . . 6
- Phase-coherent frequency switching . . . 7
- A sweep or chirp . . . 7
- AOD-style moves: several tones moving together . . . 7
- Triggered pulses . . . 7
- A pulse train repeated by the FPGA . . . 8
- Amplitude modulation from an input . . . 8
- Calibrating the output’s frequency response . . . 8
- A spectrum from the inputs . . . 8
- Checking the board itself (loopback) . . . 9

synth: theory of operation **9**
- Summary . . . 9
- Architecture . . . 9
- Datapath . . . 10
- Units . . . 11
- Spectral purity . . . 12
- Phase and the zero time t0 . . . 12
- The command queue . . . 13
- Sequences . . . 14
- Amplitude modulation . . . 15
- Inputs . . . 15
- Resources and timing (build 0x64bea10e, version 2.1) . . . 16
- Scaling: more tones, longer sequences . . . 16

Control reference **16**
- Configuration . . . 17
- Python methods . . . 17
- Sequences . . . 19
- TCP commands . . . 19
- Command line . . . 20
- Where the state is kept . . . 20

Register map **20**
Control and status . . . 20
Staging window . . . 22
Queue entries . . . 22
Push window . . . 23
Pins . . . 23

Verification record **24**
2026-10-22 (night): version 2.1, fixes of a full review; build 0x64bea10e . . . 24
2026-10-22 (evening): recordings for the user's manual; measure_tones() reference fix . . . 25
2026-10-22: version 2.0 (loops, push window, AM, shaped ramps, AC calibration); build 0xc7afaf46 . . . 26
2026-10-21: live settings after sequences, held staging writes; build 0xa6effd0c . . . 27
2026-10-20 (evening): version 1.0 on the board, loopback . . . 28
2026-10-20: version 1.0, offline (before the board) . . . 29

synth: a 16 + 16 tone phase-coherent synthesizer

synth turns the Red Pitaya into a **multi-tone signal source**. Each output, OUT1 and OUT2, is the sum of **16 independent tones**. Every tone has its own frequency (0 to 62.5 MHz in 0.44 µHz steps), amplitude (volts at the connector) and phase (degrees). All phases refer to **one common zero time** shared by both outputs.

$$OUTn(t) = offset_n + \Sigma_k A_k \sin(2\pi (f_k (t - t0) + \varphi_k / 360°)) \qquad n = 1, 2;\ \ k = 1 \ldots 16$$

Changing settings. Any set of tones on both outputs changes on **one sample** (an APPLY of the FPGA's command queue). In the default coherent mode every tone then has exactly its phase relative to t0. Going f1 → f2 → f1 returns to the same phase, and equal tones on OUT1 and OUT2 stay in phase sample for sample.

Sequences. Frequency and amplitude can ramp linearly or along smooth shapes ($\cos^2$, Blackman, minimum jerk), landing exactly on the new values. Timed sequences (steps, ramps, markers) run in the FPGA, optionally started by a TTL trigger on DIO0_P, with a marker pulse on DIO1_P, and the FPGA can repeat them by itself. Typical uses are sweeps, chirps, multi-tone RF for acousto-optic deflectors and optical tweezers, pulse trains, and phase-coherent switching.

Modulation. Each output can be amplitude-modulated by IN1 or IN2 in real time, for example from a photodiode for intensity stabilization.

Clean output. The digital sine is accurate to 0.57 LSB of 2^-17, with spurs below −120 dBc (computed). Each output is rounded only once to the 14-bit DAC, optionally with dither, so the remaining artefacts are those of the board's DAC and amplifier, which are **not yet measured**.

Units and self-test. Amplitudes and offsets are **real volts at the SMA connectors**, through the **board calibration** shared by every design, which synth can also measure (../common/CALIBRATION.md). The inputs record 8192 samples on the same time base, so in loopback the board measures each tone's amplitude and phase itself.

It is inspired by the Simon lab's "Multi-Chromatic DDS" for the STEMlab 125-14 (10 tones on OUT1, no phase control; see docs/THEORY.md).

Before connecting equipment

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

- **Loading synth leaves the outputs quiet.** All tones are off at 0 in converter units: a few mV from 0 V (the uncalibrated DAC offset). After a reboot, the service restores the last configuration, tones included.
- **Peaks can reach the full DAC range.** By default the driver refuses settings whose worst-case peak (|offset| + the sum of the amplitudes) leaves the DAC range, about ±1.13 V on this board. With `allow_clip` that check is off and the output saturates on peaks. Into a high-impedance load the output is about twice the set amplitude (next item). There is no clamp below the DAC range.
- **Load.** The set amplitude is the level **into 50 Ω**. As measured with atomic_clock at 10 MHz, the outputs behave like a 50 Ω source, so into 1 MΩ you get about twice it. The offset (DC) is right into any load.
- **Changes are steps.** A step in frequency or amplitude changes on one sample. Use ramps (sequences) for gentle changes, e.g. for acousto-optic deflectors.
- **Extension-connector pins are 3.3 V LVCMOS**, not 5 V tolerant. DIO0_P is the trigger input (pull-down); DIO1_P is the marker output. Never drive DIO1_P from outside.

- **Safe state:** `SAFE` (Python `synth.safe()`) mutes every tone at once, stops any sequence and puts both outputs at the calibrated 0 V.

Quick start

The board is set up (../common/SETUP.md) and reachable at `192.168.0.149`. Connect a scope to OUT1 and OUT2 through 50 Ω feed-throughs, or a spectrum analyzer through an attenuator.

- **Jupyter** on the board: open `http://192.168.0.149:9090` (password `xilinx`), then `synth/synth.ipynb`, and run the cells. The panel has three tabs:
  - **Tones:** two tables of 16 tones (frequency, amplitude, phase), offsets and dither, amplitude modulation, the phase mode, **Zero time**, **SAFE**;
  - **Presets**;
  - **Calibration**.
- **Python on the board:**

```
import rpfpga                                   # must come first
from pynq.overlays.synth import DRIVER
synth = DRIVER()                                # loads synth if the board runs something else
synth.set_tone(1, 1, 10e6, 0.25)                # OUT1 tone 1: 10 MHz, 0.25 V
synth.set_tone(2, 1, 10e6, 0.25, -90)           # OUT2: the same, 90 degrees behind
synth.set_tone(1, 2, 13.5e6, 0.1, 45)           # a second tone on OUT1
print(synth.describe())
```

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

```
import sys; sys.path[:0] = ["common/python", "synth/control"]
from synth import SynthClient
with SynthClient("192.168.0.149") as synth:     # the board must run synth
    synth.apply({"outputs": {"1": {"tones": {str(k): {"frequency": 1e6 * k, "amplitude": 0.06}
                                             for k in range(1, 17)}}}})
    synth.play({"steps": [{"at": 0, "ramp": 1e-3, "tones": {"1.1": {"frequency": 2e6}}}]})
    print(synth.hardware()["counts"])
```

  If the board runs another design, switch with `rpfpga.client.connect("192.168.0.149").load("synth")`. Reloading invalidates existing handles; reconnect explicitly.
- **TCP** from anything: `nc 192.168.0.149 5025`, then for example `TONE 1.1 10e6 0.25`, `TONE 2.1 10e6 0.25 -90`, `TONES?`, `ZERO`, `HW?`, `PRESET comb5`, `SAFE`.
- **Command line:** `python3 -m rpfpga.cli --host 192.168.0.149 --tone 1.1=10e6,0.25` (with `common/python` and `synth/control` on `PYTHONPATH`).

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

Presets

Name	Outputs
`off`	all tones off, 0 V (the default)
`10mhz`	10 MHz, 0.5 V on both outputs, in phase
`quadrature`	1 MHz, 0.5 V on both; OUT2 90° behind OUT1
`comb5`	OUT1: 1, 2, 3, 4, 5 MHz, 0.1 V each, in phase at t0
`schroeder16`	OUT1: 1 … 16 MHz, 0.12 V each, Schroeder phases (low crest factor); `allow_clip`

Specifications

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

Quantity	Value
Tones	16 per output, both outputs on one 125 MSa/s time base

Quantity	Value
Frequency	0 … 62.5 MHz in steps of 125 MHz/2^48 = 0.44 µHz; exact relative to the board clock (absolute accuracy and phase noise: the board's oscillator, **not measured**). Images at 125 MHz ± f grow toward Nyquist: keep below about 50 MHz
Amplitude	per tone 0 … about 1.13 V into 50 Ω, steps of 2^-17 of full scale (about 8.6 µV); corrected for the DAC's sinc(f/fs) droop by default, and for the output's frequency response if it is in the board calibration (`measure_response`, `set_output_response`)
Phase	per tone, relative to t0, steps of 2^-16 cycle (0.0055°). Measured in loopback: steps of 90°, −45° and 170° within 0.008°; returning to a frequency restores its phase within 0.005°; a 5-tone comb's tones each within 0.1° and 0.13 % of the same tone alone
Offset	per output, steps of one DAC step (about 138 µV)
Headroom	by default |offset| + Σ amplitudes within the DAC range (about ±1.13 V); `allow_clip` relaxes this, peaks then saturate (counted)
Digital spectral purity	sine core: spurs below −120 dBc; after the single rounding to 14 bits: below −95 dBc (10 MHz: −96 dBc; most frequencies below −103 dBc); with dither a flat floor (**computed** from the bit-exact model, not measured)
Analog spectral purity	measured only through the board's own ADC (loopback, 50 Ω terminators), so these are upper bounds: harmonics of a 0.4 V tone −44 to −58 dBc (H2), two-tone IMD3 −71 to −77 dBc at 0.1–0.2 V per tone, −46 dBc at 0.4 V per tone. With an analyzer: **not measured**
Loopback response	measured / set amplitude 1.00–1.11 from 0.1 to 50 MHz (peak 1.10 at 20 MHz), 0.74 at 55 MHz, 0.33 at 60 MHz, inputs included (docs/VERIFICATION.md); `measure_response` and the board calibration can correct it
Updates	any set of tones on both outputs on one sample; `apply()` takes effect 10 ms (apply_delay) after the call; a call takes about 0.1 s through the service (measured)
Ramps	linear, cos², Blackman or minimum-jerk frequency and amplitude ramps (shaped ones from 16 linear pieces, within 0.41 % of the curve, computed), up to 3600 s, exact endpoints; chirps up to 119 kHz/µs
Sequences	in the FPGA queue (4096 entries, about 680 one-tone steps), steps at least 0.5 µs apart; repeated by the FPGA (K times or until stopped) if they fit in the queue; longer ones streamed by the service at about 60,000 entries/s (measured: one-tone steps every 0.1 ms for 3 s), 20,000 entries/s while another client polls `STATUS?` continuously
Amplitude modulation	per output by IN1 or IN2: OUT = offset + (1 + depth·V_in)·tones, depth up to about ±1.8/V (LV; ±0.1/V on HV); input to output 8 samples (64 ns) inside the FPGA, plus the converters and analog paths (**not measured**)
Trigger	DIO0_P, 3.3 V LVCMOS, pull-down; edge to output change (T + 30) × 8 ns plus 0–8 ns (asynchronous input), plus the analog delay (**not measured**)
Marker	DIO1_P, 3.3 V, on the DAC edge of the step's first sample, 1 µs default; wired back to DIO0_P it arrives as a trigger 31 samples (248 ns) after the step's sample (measured)
OUT1–OUT2 skew at the connectors	digital: none. Measured by cable swap: OUT2 0.14–0.20 ns earlier than OUT1 from 1 to 40 MHz; corrected by the board calibration's output delay (−0.165 ns on OUT2 on this board), leaving about ±0.03 ns
Inputs	capture of 8192 samples of IN1 and IN2 on the same time base; tone measurement (lock-in); means for the board calibration
Settings after loading	all tones off, outputs at 0 in converter units

Hardware setup

- **Outputs.** Use 50 Ω loads for the set amplitudes. Add a low-pass filter for the band you use to remove the DAC images near 125 MHz ± f.
- **Loopback self-test.** Cable OUT1 → IN1 and OUT2 → IN2 (LV jumpers, ±1 V), with 50 Ω terminators at the inputs for the set levels. `synth.measure_tones()` then reports every tone's amplitude and phase relative to t0. Phases include the loopback delay, a constant (docs/THEORY.md "Inputs").
- **Trigger and marker** are on the extension connector E1: DIO0_P trigger in, DIO1_P marker out, ground on the same connector. 3.3 V logic only.
- **Calibration.** Without a calibration, nominal converter constants are used, which on this board are

about 14 % off in amplitude. synth can measure the calibration in loopback (`CAL LOOPBACK`, ../common/CALIBRATION.md), as mult_add can.

Documentation

Document	Contents
docs/manual/user_manual.pdf	user's manual (28 pages, measured and computed figures): getting started, tones, phase and zero time, sequences (shapes, triggers, markers, repetition), amplitude modulation, calibration (with the output response and skew), measuring with the inputs, recipes, use from experiment software, application notes (AOD and tweezer arrays, pulse sequences, phase-coherent RF, network measurements), specifications, troubleshooting, quick reference, command reference
docs/PROTOCOL.md	configuration, sequences, Python methods, TCP commands, command line
RECIPES.md	worked examples: two phase-locked outputs, a frequency comb, sweeps and chirps, AOD-style moves, triggered pulses, a loopback check
docs/THEORY.md	how it works: architecture choices, the tone datapath, sine accuracy, units, spectral purity, phase semantics, queue, sequences, inputs
docs/REGISTERS.md	the hardware registers and queue entries
docs/VERIFICATION.md	what was tested, when, and the build's resource use and timing
../common/CALIBRATION.md	the board calibration, shared by every design
../common/README.md	the shared framework: TCP service, common API, build workflow
NOTICE.md	sources
AGENTS.md	notes for maintainers and coding agents

Changing the design

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

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

`make test-hw` runs `scripts/test_hw.py` on the board. With OUT1 → IN1 and OUT2 → IN2 cabled, it checks the tones' amplitudes and phases in loopback, coherent switching, sequences and SAFE.

Source layout

Path	Contents
`design.tcl`	design description for the common recipe: ID "SYNT", RTL, one `sy_core` cell, DIO pins
`src/sy_core.v`	top: sample counter, registers, queue, two banks, capture, marker
`src/sy_bank.v`	one output: 16 tones, staging RAM, APPLY sequencer, DSP cascade sum, amplitude modulation, rounding, dither, saturation
`src/sy_tone.v`	one tone: phase accumulator, ramps, phase to sine (quarter-wave table + Taylor)
`src/sy_sine_rom.v`	the generated quarter-wave table (`model.rom_verilog`)
`src/sy_queue.v`	the command queue (with BASE and LOOP for repeating sequences) and trigger
`src/sy_capture.v`	input capture and means
`src/sy_regs.v`	AXI4-Lite registers, queue push window, staging window and its mirror
`src/sy_xoro.v`	xoroshiro128+ for the dither
`constraints/dio.xdc`	the trigger and marker pins
`control/synth/`	Python package: bit-exact model (`model.py`), sequence compiler (`sequence.py`), loopback measurements (`measure.py`: response, skew, spectra), driver, worker, TCP commands, client, command-line options, Jupyter GUI
`control/synth.ipynb`	the notebook
`scripts/test_hw.py`	the loopback hardware test (`make test-hw`)
`tools/`	host-only: `record_manual_data.py` (the manual's board recordings, through the service) and `manual_figures.py` (its figures and numbers; numpy, matplotlib)

Path	Contents
`tests/`	`tb_sine.sv`, `tb_system.sv`, `tb_control.sv` (simulation), `vectors.py` → `sy_vectors.sv` (the model's vectors), the Python tests and `fake_core.py` (a register model of the core)

Verification

Version 2.1 (build 0x64bea10e, 2026-10-22, with the fixes of a full code review) passes the offline tests and all simulations (bit-exact against the model on the core streams and the DAC pins). It meets timing at 125 MHz with WNS +0.423 ns. On the board it passed `make test-remote` and all 32 loopback checks of `make test-hw`; levels, loop response, skew, marker timing, streaming rates and ADC-based spectra are recorded. Spectrum-analyzer measurements are not yet made. Details: docs/VERIFICATION.md.

Author

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

synth recipes

Worked examples. Python on the board; on a PC use `SynthClient("192.168.0.149")` with the same calls. Frequencies are in Hz, amplitudes in volts (peak, into 50 Ω), phases in degrees relative to t0, times in seconds.

```
import rpfpga
from pynq.overlays.synth import DRIVER
synth = DRIVER()
```

Two outputs with a defined phase difference

```
synth.safe()
synth.set_tone(1, 1, 5e6, 0.3)            # OUT1: 5 MHz, 0.3 V, 0 degrees
synth.set_tone(2, 1, 5e6, 0.3, 120.0)     # OUT2: 120 degrees ahead
```

Both tones run on the same samples: digitally the difference is exactly 120°. Any skew between the connectors (the DAC, the output amplifiers) adds −360°·f·τ, unless the board calibration holds the outputs' delay (below).

Measuring and correcting the OUT1–OUT2 skew

Use two equal cables. Cable OUT1 → IN1 and OUT2 → IN2, then:

```
from pynq.overlays.synth import measure          # on a PC: from synth import measure
straight = synth.phase_difference(10e6)          # IN1 - IN2 with the same tone on both outputs
# now swap the two cables at the OUTPUTS (OUT1 -> IN2, OUT2 -> IN1), keep them on the inputs
swapped = synth.phase_difference(10e6)
lag = measure.skew(straight, swapped)            # seconds: how much later OUT2 is than OUT1
cal = synth.calibration()
synth.set_output_delay(2, cal["out2"].get("delay_s", 0.0) + lag)
```

The inputs' and the cables' delays cancel between the two measurements. The delay is stored in the board calibration, shared by every design. synth then corrects every phase on OUT2 by 360°·f·τ, so equal phase settings are in phase at the connectors. `straight["spread"]` shows the repeatability of one measurement. The unambiguous range is ±25 ns at 10 MHz; use a lower frequency for long cable differences.

A frequency comb, phases set at t0

```
tones = {str(k): {"frequency": k * 1e6, "amplitude": 0.06, "phase": 0.0} for k in range(1, 17)}
synth.apply({"outputs": {"1": {"tones": tones}}})
synth.zero_time()                          # t0 := the next APPLY sample: all phases 0 there
```

Sixteen tones of 0.06 V each have a worst-case peak of 0.96 V, which the default headroom check accepts.

With equal phases the tones add up to that full peak once per period. Schroeder phases (the preset `schroeder16`) keep the peak of 16 × 0.12 V at 0.65 V instead of 1.43 V, which lets each tone be twice as large. Use `allow_clip` for such combs; the hardware then saturates any rare peak, and `HW?` counts the saturated samples.

Phase-coherent frequency switching

```
synth.set_tone(1, 1, 10e6, 0.3)
synth.set_tone(1, 1, 11e6)                # 11 MHz with the phase 11 MHz would have had since t0
synth.set_tone(1, 1, 10e6)                # back: exactly the phase of an uninterrupted 10 MHz
```

This is the coherent mode (default). For a phase-continuous change instead, with no jump at the switch but the phase relative to t0 lost, use `synth.apply({"phase_mode": "continuous"})`.

A sweep or chirp

```
synth.set_tone(1, 1, 1e6, 0.2)
synth.play({"steps": [
    {"at": 0.0, "ramp": 10e-3, "tones": {"1.1": {"frequency": 20e6}}},       # 1 -> 20 MHz in 10 ms
    {"at": 20e-3, "ramp": 10e-3, "tones": {"1.1": {"frequency": 1e6}}}]})    # and back
```

The sweep is linear and phase-continuous, and lands exactly on 20 MHz. The fastest chirp is 119 kHz/µs. Amplitude ramps the same way (`"amplitude"` in a ramped step), e.g. for smooth turn-on.

AOD-style moves: several tones moving together

Acousto-optic deflectors make one beam per RF tone. To move eight beams by 2 MHz in 300 µs, with smooth amplitude, and come back:

```
base = {"1.%d" % (k + 1): {"frequency": 30e6 + k * 1e6, "amplitude": 0.08} for k in range(8)}
moved = {"1.%d" % (k + 1): {"frequency": 32e6 + k * 1e6} for k in range(8)}
back = {"1.%d" % (k + 1): {"frequency": 30e6 + k * 1e6} for k in range(8)}
synth.play({"steps": [{"at": 0.0, "ramp": 50e-6, "tones": base},
                      {"at": 1e-3, "ramp": 300e-6, "tones": moved, "marker": True},
                      {"at": 2e-3, "ramp": 300e-6, "tones": back}]})
```

All eight tones start each ramp on the same sample.

For moving atoms in optical tweezers, a smooth profile avoids heating. Use `"shape": "min-jerk"` (zero velocity and acceleration at both ends) on the moves:

```
synth.play({"steps": [{"at": 0.0, "tones": base},
                      {"at": 1e-3, "ramp": 300e-6, "shape": "min-jerk", "tones": moved, "marker":
↪  True},
                      {"at": 2e-3, "ramp": 300e-6, "shape": "min-jerk", "tones": back}]})
```

Each move is made of 16 linear pieces that land exactly on the curve (within 0.28 % of the 2 MHz move between pieces, computed). Each piece costs one APPLY per changed tone set, so eight tones and 16 pieces need at least 16 × (0.5 µs + 8 ns × 8 tones × 4 words): a 300 µs move is far above that. `cos2` and `blackman` are the other shapes, mostly for amplitude.

The board's band is DC to 62.5 MHz; for a typical 80 MHz AOD, mix the output up. Add an RF amplifier and filters as the AOD needs. The board's output level is a few dBm into 50 Ω at most.

Triggered pulses

Wait for a TTL edge on DIO0_P, then make a 20 µs, 15 MHz burst 100 µs after it, with a marker on DIO1_P at its start:

```
synth.play({"trigger": True, "steps": [
    {"at": 100e-6, "tones": {"2.1": {"frequency": 15e6, "amplitude": 0.4}}, "marker": True},
    {"at": 120e-6, "tones": {"2.1": {"amplitude": 0.0}}}]})
# ... the edge arrives (or synth.trigger() for a test) ...
print(synth.sequence_status())
```

In a trigger sequence t0 is the trigger edge. Every burst therefore starts with the same phase relative to the trigger, to within the 8 ns sampling of the asynchronous input. Run `play()` again for the next trigger, or add `"repeat": 0` to answer every trigger until `stop()`.

A pulse train repeated by the FPGA

A 20 MHz pulse with 4 µs cos² edges and a 2 µs flat top, every 100 µs, 1000 times, with a marker at the start of each:

```
synth.play({"repeat": 1000, "period": 100e-6, "steps": [
    {"at": 0.0, "tones": {"1.1": {"frequency": 20e6, "amplitude": 0.0}}},
    {"at": 1e-6, "ramp": 4e-6, "shape": "cos2", "segments": 4,
     "tones": {"1.1": {"amplitude": 0.3}}, "marker": True},
    {"at": 7e-6, "ramp": 4e-6, "shape": "cos2", "segments": 4,
     "tones": {"1.1": {"amplitude": 0.0}}}]})
```

The whole sequence is 50 queue entries. A shaped ramp needs about 0.6 µs per piece for one tone; shorter edges get fewer pieces.

The queue repeats the steps by itself: there is no host traffic, and repetition r starts exactly r × 100 µs after the first. In coherent mode every pulse has the same phase relative to its own start. Use `"repeat": 0` to run until `synth.stop()`. A repeating sequence must fit in the queue (at most 4095 entries), and every tone's first change in it must be a step, as here.

Amplitude modulation from an input

Modulate OUT1's tones by the voltage on IN1, e.g. a photodiode error signal:

```
synth.set_tone(1, 1, 25e6, 0.3)
synth.set_am(1, "in1", depth=-0.5)              # OUT1 = (1 - 0.5/V * V_IN1) * 0.3 V tone
```

`depth` is in 1/V of the input's calibrated volts (at most about ±1.8/V on LV). The factor 1 + depth·V stays within 0 … 2 while |depth·V| ≤ 1 and saturates beyond. The headroom check uses the largest factor the input can make (LV spans about ±1.1 V on this board). Inside the FPGA the input reaches the output 8 samples (64 ns) later; the converters and analog paths add their own delays (not measured). `synth.set_am(1, "off")` ends it; sequences can switch it per step (`"outputs": {"1": {"am": {...}}}`).

Calibrating the output's frequency response

The DC calibration makes 0.5 V right at low frequency. At higher frequencies the output amplifier and filter roll off. To correct this in loopback (OUT1 → IN1, OUT2 → IN2, with 50 Ω terminators at the inputs as in the DC calibration):

```
for n in (1, 2):
    table = synth.measure_response(n)           # [[Hz, gain], ...] from 1 to 60 MHz
    synth.set_output_response(n, table)         # stored in the board calibration
```

The table is the loop's response: it includes the input's own front end. With a power meter or a calibrated scope instead, measure the outputs alone and store your own table the same way. `synth.set_output_response(n, [])` removes it.

A spectrum from the inputs

```
from pynq.overlays.synth import measure
f = measure.bin_frequency(10e6)                 # on the FFT grid: no leakage of its own
synth.set_tone(1, 1, f, 0.4)
lines = measure.analyze(synth.spectrum("in1"), [f])
print(lines["fundamentals"], lines["harmonics"], lines["spur"], lines["floor_dbc"])
```

The spectrum (dB re 1 V peak per 15.26 kHz bin) includes the ADC's own distortion and noise, so it shows the output's spurs only down to about −75 dBc. It is a check, not a spectrum analyzer.

Checking the board itself (loopback)

Cable OUT1 → IN1 and OUT2 → IN2 (LV jumpers), with 50 Ω terminators at the inputs.

```
synth.apply({"outputs": {"1": {"tones": {"1": {"frequency": 1e6, "amplitude": 0.2},
                                         "2": {"frequency": 2.7e6, "amplitude": 0.1, "phase": 45}}},
                         "2": {"tones": {"1": {"frequency": 1e6, "amplitude": 0.2}}}}})
result = synth.measure_tones()
for ch, data in result["inputs"].items():
    for k, t in data["tones"].items():
        print(ch, k, "%.4f V (set %.4f), %+.2f deg (set %+.2f)" % (
            t["amplitude"], t["set_amplitude"], t["phase"], t["set_phase"]))
```

How to read the result:

- The measured phases differ from the set ones by the loopback delay, $-360° \cdot f \cdot \tau$, which is constant. Repeat after any change: the differences stay the same, which shows the phase coherence.
- The amplitudes agree to the calibration's accuracy and the inputs' frequency response (not measured).
- Spurs and intermodulation below about −70 dBc need a spectrum analyzer: the ADC's own harmonics are larger (see "A spectrum from the inputs").

synth: theory of operation

This document explains how the FPGA makes the tones, what the numbers mean, and which limits come from the digital design and which from the analog board. The register-level details are in REGISTERS.md. The bit-exact reference for everything here is `control/synth/model.py`. The testbenches compare the RTL against it sample by sample.

Summary

Each output is the sum of 16 tones plus an offset:

```
OUTn(t) = offset_n + sum_k A_k sin(2 pi (f_k (t - t0) + phi_k / 360)),
```

evaluated at every sample, 125 MSa/s, and rounded once to the 14-bit DAC.

Quantity	Value
Frequency step	125 MHz / 2^48 = 0.44 µHz
Phase step	2^-16 cycle = 0.0055°
Amplitude step	2^-17 of full scale (about 8.6 µV on this board)
Spurs of the digital sine	below −120 dBc (computed)
Spurs after the 14-bit rounding	below −95 dBc (computed); with dither, a flat floor instead
Updates	any set of tones on both outputs, at one exact sample (APPLY)
Ramps	linear, or shaped ($\cos^2$, Blackman, minimum jerk) from linear pieces; exact endpoints
Sequences	timed steps from a 4096-entry queue; repeated by the FPGA itself (LOOP); triggered
Modulation	amplitude modulation of each output by IN1 or IN2

Phases refer to one zero time t0, the same for both outputs. Frequency and amplitude can ramp with exact endpoints, and timed sequences run from the FPGA's command queue, optionally started by a TTL trigger and repeated by the queue.

Architecture

The xc7z010 has 80 DSP slices, 60 block RAMs (36 kbit) and 17,600 LUTs. These are the ways to make many tones on it:

1. **A bank of parallel DDS tones (this design).** Each tone has its own phase accumulator, sine table and amplitude multiplier, and the tones of an output are summed.
   - Any frequency, independent phases, exact timing of every change.
   - Cost: 2 DSP slices, half a block RAM, about 245 LUTs and 355 flip-flops per tone (out-of-context implementation, 2026-10-21).

    - 16 tones per output use 64 of the 80 DSP slices. About 18 per output is the ceiling at 125 MHz.
2. **The same bank time-multiplexed at 250 MHz:** 32 tones per output on the same DSP slices. Tried out of context; it needs a restructured tone datapath (see ”Scaling” below) and is not part of this version.
3. **A periodic waveform in memory,** computed on the CPU and played in a loop.
    - Any number of tones, but only on the grid fs/N (block RAM: N ≈ 64k, so a 1.9 kHz grid).
    - Every change means reloading the whole waveform, and phase-coherent switching of single tones is impossible.
    - Streaming from DDR would need an AXI HP port and DMA: a change to the common block design.
4. **IFFT or polyphase band synthesis.** Too large for a 7010, and useful only for narrow bands.
5. **CORDIC sines.** No block RAM, but several times the logic of a table.

Comparison with the Simon-lab ”Multi-Chromatic DDS” (2017)

Facts about the MDDS are from its distributed archive, which contains a compiled bitstream and Python 2 scripts.

	Simon-lab MDDS (STEMlab 125-14)	synth
Outputs × tones	OUT1 × 10	OUT1 × 16 and OUT2 × 16, one time base
Frequency word	32 bits (29 mHz)	48 bits (0.44 µHz)
Phase control	none	per tone, relative to t0
Changes	per-tone tables of ramp segments	atomic APPLY of any set of tones on both outputs; queue
Ramps	piecewise linear	linear, exact endpoints, explicit length
Trigger	DIO0_P, software	DIO0_P, software; APPLY marker on DIO1_P
Overflow of the sum	not documented	saturating, with counters; driver headroom check
Spectral data	none published	computed here; measurements in VERIFICATION.md
Source	Verilog on request, no redistribution	open in this repository

The MDDS installer also adds a daily `curl` that sends the board’s MAC address to the authors.

Datapath

Sample numbering

One counter, `now`, counts the 125 MHz clocks from reset (8 ns each). Every time in the design is a sample index of this counter: APPLY times, trigger times, capture start, t0.

The tones of an output are summed by a chain of DSP slices (a systolic cascade). Tone k of an output is therefore processed k clocks after tone 0, and its state registers hold ”sample n” when the counter reads n + k. The APPLY sequencer walks down the tones one clock at a time, so every tone changes at the same sample n_a. The sum of sample n leaves the core in the DAC stream when the counter reads n + 28 (register LATENCY). The vendor DAC interface adds two more clocks: an input register, and the DDR output that sends OUT2 in the first half and OUT1 in the second half of each 8 ns frame.

One tone

Each tone keeps this state (`sy_tone.v`):

Register	Bits	Meaning
acc	48	phase, 2^-48 cycle
F	72	phase increment, 2^-72 cycle per sample (48 bits + 24 ramp-fraction bits)
A	41	amplitude, 2^-41 of full scale (17 bits to the multiplier + 24 fraction bits)
cnt	24	ramp samples left
dF, dA	56, 32 signed	ramp steps
POW	16	phase offset, 2^-16 cycle

The targets of a ramp are not stored in the tone: they are staged words, used once at the APPLY. Every sample:

```
acc <- acc + F[71:24]
if cnt > 0:  F <- F + dF,  A <- A + dA,  cnt <- cnt - 1
```

On an APPLY at sample n_a the tone takes its staged words (`Tone.apply` in the model):

- acc continues with the old frequency (acc(n_a) = acc(n_a − 1) + F(n_a − 1)). With **LOAD** set, acc(n_a) = P instead, and the change is a step.
- With **count 0** (or LOAD), F(n_a) = F_end·2^24 and A(n_a) = A_end·2^24: a step.
- With **count C > 0**, a ramp starts from the current F and A and moves by dF and dA per sample for C samples. The driver computes the steps as d = ceil((target·2^24 − start)/C), so after C samples the value lies in [target·2^24, target·2^24 + C): for C < 2^24 its upper bits, which are the frequency and amplitude words the tone uses, equal the target **exactly**. The rest, below one LSB of the words, stays in the fraction bits.
- dF, dA and POW take their staged values.

This saves the two target registers per tone (65 flip-flops) of the first version, which landed by overwriting.

Phase to sine

The phase is p = acc[47:24] + POW·2^8, 24 bits (2^-24 cycle). The sine comes from a **quarter-wave table** of 1024 entries, each holding two 18-bit values:

```
s_i = round((2^17 - 1) sin a_i),   d_i = round((2^17 - 1) cos a_i (2 pi / 4096) 2^9),
a_i = (i + 1/2) 2 pi / 4096.
```

How the 24 bits are used:

- The top 2 bits select the quadrant. Odd quadrants invert the 22 lower bits. Because the entries sit at half-step offsets, this mirroring is exact.
- 10 bits address the table.
- 12 bits f give the offset from the entry centre, δ = 2f − 4095 in 1/8192 of an entry.
- One DSP slice adds the first-order Taylor term:

  ```
  y = (s_i 2^22 + d_i delta + 2^17) >> 18          (22 bits, 16 (2^17 - 1) sin)
  ```

- The sign for quadrants 2 and 3 is applied in the output's sum, where the DSP adds or subtracts the product.

One 36-bit table read per tone gives both s and d. A dual-port block RAM serves two tones. Compared with the real sine, the error is at most 0.57 LSB of 2^-17, almost all of it the table's rounding. The Taylor remainder is (π/4096)²/2 = 3·10⁻⁷ of full scale. `tb_sine` checks 2^18 phases over the whole cycle.

Sum and output

Each tone's y is multiplied by its 17-bit amplitude, and the products of the 16 tones are added in the DSP cascade (P = PCIN ± y·A). With amplitude modulation on (see "Amplitude modulation" below) the sum is first multiplied by (1 + m). Then, in `sy_bank.v`:

```
q = floor((sum + offset 2^23 + 2^24 + dither) / 2^25),   saturated to -8192 .. 8191,
```

and the DAC stream is 4q. This is one rounding, half up, of the exact sum to the DAC's 14 bits; the stream carries only codes the DAC resolves.

- Peaks beyond the range saturate instead of wrapping, and every saturated sample is counted (CLIPS1/2).
- The optional dither is triangular, ±1 DAC step, from a xoroshiro128+ generator per output.

Units

Through the board calibration (`rpfpga.calibration`, `../common/CALIBRATION.md`) the driver works in volts at the connectors. With G volts per stream code and B volts of offset for output n:

- **Amplitude.** A tone with amplitude code a has a peak of a·(2^17 − 1)/2^19 stream codes, so its amplitude is G·a·(2^17 − 1)/2^19 volts.
  - The largest single tone (a = 2^17 − 1) is about G·32768 volts: 1.13 V on this board (root README, board specifications).
  - With **droop correction** (default on), the code is divided by sinc(f/fs), the DAC's zero-order-hold response: −0.37 dB at 20 MHz, −2.4 dB at 50 MHz. If the board calibration holds a measured **output response** (`../common/CALIBRATION.md`), the code is also divided by that gain at f. The volts then refer to the tone's actual amplitude at the connector, as well as that response was measured.
- **Offset.** The code is 4·round((V − B)/(4G)), on the DAC grid.

- **Frequency.** The word is round(f·2^48/fs), and the driver reports the realized frequency word·fs/2^48. It is exact relative to the board's 125 MHz clock, and the error from rounding is below 0.22 µHz. The board oscillator's absolute accuracy and phase noise are not measured.
- **Phase.** In degrees, POW = round((φ + 360·f·τ_n)/360·2^16), where τ_n is the output's **delay** in the board calibration (0 unless measured). The phase then refers to the connector, and the reported phase subtracts the correction again. For ramps the correction belongs to the frequency of the step that set the phase.
- **Load.** The calibration is a DC calibration. As measured with atomic_clock at 10 MHz, the output behaves like a 50 Ω source: the set amplitude appears into 50 Ω, and about twice it into high impedance (`../atomic_clock/docs/VERIFICATION.md`).

Headroom. Unless `allow_clip` is set, the driver refuses a configuration whose worst-case peak |offset| + Σ amplitudes (+1 step with dither) can leave the DAC range. With 16 equal tones that limits each to 1/16 of full scale (about 70 mV). Random or Schroeder phases keep the actual peak near 3–4 times the RMS, well below the sum. With `allow_clip` such combs can use larger amplitudes; the hardware saturates rare peaks and counts them (CLIPS1/2, `HW?`).

Spectral purity

These figures come from the bit-exact model (`tests/test_model.py`): Kaiser-windowed FFTs of 2^15 samples, one full-scale tone. They are **computed, not measured**.

Signal	1.23 MHz	10 MHz	12.3 MHz	31.4 MHz	47.7 MHz
sine core alone (22 bits)	−129 dBc	−120 dBc	−125 dBc	−132 dBc	−132 dBc
after the 14-bit rounding	−116 dBc	−96 dBc	−103 dBc	−117 dBc	−106 dBc
14-bit with dither	floor	floor	floor	floor	floor

With dither, no spur rises above the flat floor (−113 dB per FFT bin here).

- **Rounding to 14 bits.** The error is deterministic. If the frequency is a simple fraction of 125 MHz (10 MHz = 2/25 of it), the error repeats with a short period and concentrates into spurs (about −96 dBc). Otherwise it spreads out. Dither turns it into white noise: about −159 dBc/Hz relative to a full-scale tone (the rounding's −164 dBc/Hz plus the triangular dither's own power, 4.8 dB more in all).
- **The DAC and output amplifier.** Their harmonics and intermodulation are not measured. They are probably far above the digital spurs (−60 to −80 dBc is typical for such boards) and will set the spectral purity in practice. Summing tones is exactly linear in the FPGA (the only nonlinearity is saturation), so all intermodulation products between tones are analog.
- **Images and filtering.** Each sample is held 8 ns, so a tone at f has images at 125 MHz ± f. Their attenuation by the board's output filter (about 60 MHz, root README) is not measured. For clean spectra keep f below about 50 MHz, and add an external low-pass for the band in use.
- **Line width.** The DDS adds no random phase noise. Each line is as sharp as the board's 125 MHz clock: its phase noise reaches the output scaled by (f/125 MHz)², plus the DAC's noise floor. With the 14-bit floor, a single full-scale tone has about 86 dB SNR over the Nyquist band.

Phase and the zero time t0

Every tone obeys phase(n) = acc(n) + POW, where n is the shared sample index.

- **Coherent mode (default).** Every APPLY of `apply()` loads every tone's accumulator with P = F·(n_a − t0) mod 2^48. The phase is then exactly F·(n − t0) + POW at every later sample, whatever the tone did before. Switching f1 → f2 → f1 returns to the phase it would have had without the excursion. The same holds for both outputs: two tones of equal frequency and phase on OUT1 and OUT2 are in phase sample for sample.
- **Continuous mode.** The accumulators run on (an immediate APPLY without LOAD). Frequency changes are phase-continuous, and POW shifts the running phase. Phases relative to t0 are then not defined.
- **The value of t0.** t0 is a sample index in the configuration: 0 by default, which is the FPGA's reset. `zero_time()` sets t0 to the next APPLY sample, so that every tone starts at exactly its phase setting there. After a reload of the FPGA the counter starts again at 0.
- **Ramps** are always phase-continuous. A step after a ramp, in coherent mode, loads the accumulator again, so the phase relative to t0 is re-established, with a phase step if the ramp moved it.
- **What "in phase" means at the connectors.** The digital phases are exact. The DAC bus presents OUT1 and OUT2 4 ns apart, and whether that becomes a skew at the connectors depends on the DAC's latching (its data sheet) and the analog paths.

- `phase_difference()` measures IN1 − IN2 with the same tone on both outputs. Done once with straight cables and once with the cables swapped at the outputs, `measure.skew()` gives how much later OUT2 is than OUT1: the inputs' and cables' own delays cancel.
- The result goes into the common board calibration as a per-output delay (`set_output_delay`, shared by all designs). The driver then adds 360°·f·τ to each phase (see "Units"). The measured value is in VERIFICATION.md.

The command queue

All updates go through a 4096-entry queue in block RAM (`sy_queue.v`). It is read ahead so that it executes one entry per clock:

- **STAGE** writes a staging word.
- **TIME** sets the time register.
- **APPLY** waits for its sample and starts the sequencers of both outputs.
- **WAIT_TRIGGER** waits for a trigger edge.
- **CAPTURE** arms the input capture.
- **BASE** sets T_trig, the reference of relative times, to the time register: a sequence without a trigger uses it as its time 0.
- **LOOP** rewinds the queue by N entries, K times in total (or until stopped), and advances T_trig by the time register each time: the queue plays a repeating sequence by itself, with no host traffic.

Entries are pushed either one at a time (two register writes each) or as a block through the **push window** (`0x0800`–`0x0FFF`), which the driver writes with one array store of up to 256 entries. PUSH_COUNT counts the accepted entries, and the driver checks it after the blocks. If the bus did not deliver all of them, the driver empties the queue and reports an error; from then on it pushes word by word.

How `apply()` uses the queue:

1. It empties the queue, which also stops any sequence.
2. It writes the staging window directly.
3. It reads the counter, writes the preloads P for n_a = now + apply_delay (10 ms by default), and pushes TIME + APPLY with LATE_OK.

The timing-critical part, from reading the counter to pushing the APPLY, is a few hundred register accesses (4.7 µs per read and about 10 µs per write, measured), so the default delay leaves ample margin. If the host is slower, the APPLY comes late. It is then applied at once, the phases are off by f times the lateness, and LATE_COUNT and `HW?` say so; applying again fixes it.

Timing rules:

- An APPLY is late if, when it is reached, its sample is less than 4 clocks ahead.
- In a sequence a late APPLY halts the queue, because the timing would be wrong. The sequence state becomes `halted`, and the outputs keep the last applied settings.
- After an APPLY the sequencers run for 32 clocks. Consecutive APPLYs must therefore be at least 36 clocks + one clock per staged word apart; the sequence compiler requires 64 + one per staged word.

Trigger. DIO0_P is registered in the IOB and synchronized. T_trig is the clock in which the IOB register first holds the high level. The pin edge precedes it by 0–8 ns, because the input is asynchronous to the sample clock. Only edges that arrive while a WAIT_TRIGGER waits count; the others are counted as missed.

A trigger-relative APPLY at T samples puts sample T_trig + T on the DAC pins (T + 30)·8 ns after T_trig (LATENCY 28 plus the DAC interface's 2). The minimum T is set by the staged words between the trigger and the APPLY: 8 samples plus one per word. The compiler requires 16 plus one per word, so a trigger sequence's first step is at least about 0.15 µs after the edge.

Marker. An APPLY with MARKER raises DIO1_P on the DAC clock edge that puts sample n_a on the outputs, for MARKER_WIDTH samples (1 µs by default). `tb_system` checks this with the vendor DAC model. The marker-to-trigger loop delay (DIO1_P wired to DIO0_P) is in VERIFICATION.md; the analog delay of the outputs relative to the marker pin needs a scope and is not measured.

Loops. A repeating sequence is compiled as TIME(start) BASE, the steps relative to the base, then TIME(period) LOOP(K, N). Every LOOP adds the period to T_trig, so repetition r runs its steps at start + r·period exactly, and coherent loads in it refer to its own start. Repetitions with a trigger wait for a new edge each time instead.

Sequences

`play()` (`control/synth/sequence.py`) turns a list of timed steps into queue entries: STAGE words, TIME and APPLY. It tracks every tone with the model's exact arithmetic, so each ramp starts from the tone's exact frequency and amplitude, and the coherent preloads are exact.

Steps:

- A step with `ramp` = 0 changes frequency, amplitude and phase on one sample; in coherent mode it loads the accumulator.
- A ramp moves frequency and amplitude linearly and lands exactly on the target words (see "One tone").
- A **shaped ramp** (`shape`: `cos2`, `blackman`, `min-jerk`) is made of up to `segments` (16) linear pieces. Each piece ends exactly on the shape's value, so the curve is a polygon through the shape, with exact endpoints. In 16 pieces it deviates from the true curve by at most 0.24 % ($\cos^2$), 0.28 % (min-jerk) and 0.41 % (Blackman) of the ramp's total change (computed). Fewer pieces are used if a piece would be shorter than an APPLY needs (about 0.5 µs plus 8 ns per changed word). `min-jerk` ($x^3(10 - 15x + 6x^2)$) is the usual profile for moving optical tweezers.
- A ramp longer than 2^24 − 1 samples (134 ms) is split into equal linear pieces.

Limits:

- The fastest chirp is $2\text{^}55 \cdot 2\text{^}{-72} \cdot fs^2$ = 119 kHz per µs; faster changes are steps.
- The fastest amplitude ramp is full scale in 8 µs.
- Ramps last at most 3600 s (split into pieces of at most 134 ms).

Repetition:

- `repeat` K plays the steps K times (0: until stopped), `period` seconds apart. The queue repeats them by itself, so a repeating sequence must fit in 4095 entries, and its period must exceed its last step by at least 0.5 µs plus the time the first step's staged words need after the last step.
- Every tone's first change in a repeating sequence must be a step (not a ramp), so that each repetition starts from the same state. That first change stages all of the tone's words (and an output's first change all of its words), since a repetition starts from where the previous one ended, not from the words of the first.
- The headroom is checked for the later repetitions too: until a tone changes in a repetition, it still has its amplitude from the end of the previous one.

Start and streaming:

- Without a trigger, time 0 is set after the sequence is compiled: now + delay + an allowance for the rest of play(): 10 µs per entry for relocating and storing the entries (measured 7.4 µs), 5 µs per entry pushed in blocks (measured 1.8 µs; 50 µs word by word, measured 15 µs), and 0.5 s for the worker to take over a long sequence. The start sample, the TIME words and the preloads are then filled in, so the host's compile time cannot make the first step late.
- With `trigger`, the sequence waits for a DIO0 (or software) trigger. Step times are relative to the edge, which is then t0.
- Sequences longer than the queue are streamed by the service's worker as space frees up. play() refuses them when no worker is running.
  - The worker fills all free queue space with block pushes. It sustains about 60,000 entries per second (measured: steps of one tone, 6 entries each, every 0.1 ms for 3 s; from a PC, 6000 steps every 0.07 ms ran, every 0.05 ms fell behind). While another client polls `STATUS?` without pause (about 100 ms each, holding the board), steps every 0.3 ms run and every 0.2 ms fall behind. Denser parts must fit within the 4096 entries already in the queue.
  - A step that changes one tone costs 6 entries (7 when the frequency word's high part changes); each further tone in the same step costs 4–10. The queue alone holds about 680 one-tone steps.
- Compiling takes about 0.85 ms per step on the board's CPU (3000 one-tone steps: 2.5 s, measured).
- Over TCP, long sequences go compressed (PLAYZ), because the service's lines are limited to 64 KB.

During and after a sequence, CONFIG? reads the settings the tones actually run on.

- The staging window always holds one step ahead: the next step's words are staged as soon as the previous APPLY has run.
- The driver therefore replays the sequence's entries up to the last executed APPLY, as counted by APPLY_COUNT, starting from the words staged when play() started.
- When a sequence is stopped or halts, those live words are written back into the staging window, so later reads and the next apply() build on what the outputs carry.
- A ramp that is still running when the sequence stops reads as its target, which it reaches on its own.

Ramps from an earlier sequence. stop() and a halt leave a running ramp running. If a new sequence's first change of such a tone is a ramp, the sequence first steps the tone to that ramp's target (one IMMEDIATE APPLY, no phase change), because its ramps are computed from there.

Steps between ramp pieces. The pieces of a shaped or long ramp are queued in time order with the other steps, so other tones can change during the ramp. The ramping tone itself cannot change until its last piece.

Interactions:

- play() right after apply() keeps the pending APPLY: the sequence starts at least 64 samples after it. If the sequence is stopped before that APPLY has run, the APPLY is scheduled again, so the apply() is not lost.
- play() compiles the new sequence before it stops the running one: a refused sequence changes nothing.
- apply() during a sequence stops it.
- A failed push to the queue (the push-window check) empties the queue and ends the sequence as `halted`, with the error.
- MUTE (`SAFE`) also ends an APPLY that is half-way through its 16-clock walk down the tones, so no later tone takes its staged amplitude.

Amplitude modulation

Each output can be amplitude-modulated by IN1 or IN2:

`OUTn(t) = offset_n + (1 + depth · V_in(t)) · sum_k A_k sin(...),`

with depth in 1/V and V_in in calibrated volts at the input (`"am": {"source": "in1", "depth": 0.5}`). The offset is not modulated.

- The input code c is turned into m = sat18(((c − A_in)·D) >> 14) (2^-17 per unit), where A_in is the input's calibrated offset and D = round(depth·2^31/K_in) the depth word. Then sum' = sum + 2·((sum >> 18)·m). The model (`model.am_factor`, `model.modulated`) is bit-exact with the RTL.
- Input sample s + 20 modulates output sample s (`model.AM_LEAD`). Output sample s leaves the core 28 clocks after the counter reads s, so from the input's samples to the output's the core takes 8 clocks (64 ns); the converters and the analog paths add their delays (not measured).
- The depth word's 18 bits limit |depth| to 131071·K_in/2^31: about 1.8/V on LV (2/V nominal) and 0.1/V on HV. 1 + m saturates at 0 and 2.
- The headroom check assumes the worst case: amplitudes times (1 + |depth|·V_full), where V_full is the input's full scale.
- AM settings change with the output's own words, at an APPLY (OUT flags); sequences can switch them per step.
- Uses: amplitude stabilization from a photodiode, intensity noise eaters, or a modulated test signal. The bandwidth is that of the inputs and outputs (the ADC input filter and the DAC's), not measured here.

Inputs

- **Capture.** IN1 and IN2 are stored for 8192 samples (65.5 µs) from a chosen sample index, on the same time base as the tones. Sample c is the ADC value in the clock where the counter reads c.
- **measure_tones()** captures and runs a Blackman-Harris lock-in at each configured frequency. The reference phase of the first sample is computed exactly from the frequency word and the sample index, F·(start − t0) mod 2^48. It returns the amplitude and the phase relative to t0 at the ADC sample times. The difference from the set phase is −360°·f·τ, where τ is the loopback delay (DAC, cables, ADC): a constant to measure once, and a check of phase coherence after any change.
  - The ADC's own harmonics (about −70 dBc) and noise limit what the board can check itself. Spurs and intermodulation below that need a spectrum analyzer.
- **spectrum()** (in `measure.py`) gives the input's amplitude spectrum from 8192-sample captures: 15.26 kHz bins, a Blackman-Harris window, power averages. `analyze()` reads the fundamental's level, its harmonics, the largest other spur and the floor. The ADC's own distortion and noise set its limits.
- **measure_response()** steps one tone through a list of frequencies (OUTn → INn) and returns the loop's gain table for the board calibration. It includes the input's response, as the DC loopback calibration includes the input's gain.
- Once a step of a trigger sequence has run, the tones' phases refer to the trigger edge, and so do measure_tones()'s (`"reference": "trigger"`, t0 = T_trig). In a repeating sequence without a trigger they refer to the start of the repetition in which the capture starts (`"repetition"`). A coherent apply() ends this (phases from t0 again). A `t0` argument chooses any other sample index.

- **Means.** IN1 and IN2 are summed over MEAN_N samples. These are the `adc_codes`/`hold_dac_codes` hooks, so synth can measure the board calibration (`CAL OFFSETS`, `CAL LOOPBACK`) like mult_add.

Resources and timing (build 0x64bea10e, version 2.1)

Resource	Used	Of	Share
LUTs	11,354	17,600	65 %
Flip-flops	16,201	35,200	46 %
Slices	3,794	4,400	86 %
Block RAM	32.5	60	16 sine tables, 7.5 queue, 8 capture, 1 mirror
DSP	68	80	2 per tone, 2 per output for amplitude modulation

Routed timing at 125 MHz: WNS +0.423 ns, WHS +0.018 ns.

Version 1.0 (build 0x166b29c8) used 11,454 LUTs, 17,489 flip-flops and 4,021 slices (91 %): dropping the ramp targets saved more than amplitude modulation, loops and the push window cost. Slice packing is still the tightest resource.

Scaling: more tones, longer sequences

Per-tone cost (out-of-context implementation of `sy_core`, 2026-10-21): 244 LUTs and 355 flip-flops per tone, 1 DSP for the Taylor term and 1 in the output cascade. The two 16-tone banks with their output stages are 4,740 LUTs and 6,850 flip-flops each.

32 tones per output at 250 MHz (time-multiplexing, two tones per pipeline): an out-of-context implementation of one 16-tone bank (`sy_bank`) at a 4 ns clock (2026-10-21) failed timing by 2.08 ns on 3,751 of 11,895 endpoints. Almost all failing paths run from the ramp counter's zero test to the 72-bit frequency adder: 18 carry stages and 6.05 ns. Sharing a pipeline between two tones adds one register to each of its loops ("C-slow" retiming), so the adders could be split into two 36-bit halves and the ramp decision registered. Estimate for 2 × 32 tones in that form, not built:

Resource	Estimate	Of
LUTs	about 12,000	17,600
Flip-flops	about 26,000	35,200
DSP	68	80
Block RAM	32.5	60

Every tone keeps its own state in flip-flops, so their count barely changes. The other costs are a second clock domain (250 MHz from the 125 MHz clock), a two-slot APPLY walk, a reworked staging bus, and timing closure at 74 % flip-flop use on the slowest speed grade. Decision: not in this version. The way forward is a separate bank prototype with the split datapath, checked out of context before anything else changes.

A DDR-backed queue (sequences from the 512 MB DDR through an AXI HP port and DMA) would change the common block design for every design. It is not needed now:

- repeating sequences run from the queue itself (LOOP), with no host traffic;
- the push window makes queue writes cheap;
- the worker streams about 60,000 entries per second with block pushes (measured).

Only aperiodic sequences that stay denser than the streaming rate for longer than the 4096-entry queue holds would need it.

Control reference

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

- the **driver** `Synth` (Python on the board, e.g. in Jupyter);
- the **client** `SynthClient` (Python on a PC), which talks to the board's control service and has the same methods;

- the **TCP protocol** of that service (port 5025, one text command per line);
- the **command line** `python -m rpfpga.cli`.

The parts common to all designs (the uniform driver API, the TCP service and its core commands, the command line) are in `../../common/README.md`. This page covers what synth adds. Units:

- frequencies in Hz;
- amplitudes in volts (peak) at the connectors, through the board calibration (`../../common/CALIBRATION.md`);
- phases in degrees, relative to the zero time t0 (a sample index);
- times in seconds.

Configuration

Tones are numbered 1–16 on each output. The configuration, as `config()` returns it and `apply()` takes it (sparse patches merge):

```
{"outputs": {"1": {"offset": 0.0,              # volts
                   "dither": False,            # triangular dither of one DAC step
                   "am": {"source": "off",     # amplitude modulation by "in1" or "in2"
                          "depth": 0.0},       #   OUT = (1 + depth V_in) tones + offset; 1/V
                   "tones": {"1": {"frequency": 10e6, "amplitude": 0.25, "phase": 0.0},
                             "2": {...}, ..., "16": {...}}},
             "2": {...}},
 "phase_mode": "coherent",       # or "continuous"
 "t0": 0,                        # zero time: a sample index (8 ns each; 0 = FPGA reset)
 "droop_correction": True,       # amplitudes corrected for the DAC's sinc(f/fs) and the
                                 #   board's measured output response (if calibrated)
 "allow_clip": False,            # True: no worst-case headroom check (the output saturates)
 "apply_delay": 0.01,            # coherent mode: the change takes effect this long after the call
 "marker_width": 1e-6}           # DIO1 marker pulse (sequences)
```

Setting	Range	Resolution
`frequency`	0 ... 62.5 MHz (images grow above about 50 MHz: docs/THEORY.md)	0.44 µHz
`amplitude`	0 ... about 1.13 V on this board (the calibrated full scale)	2^-17 of full scale
`phase`	any (wrapped to −180 ... 180)	0.0055°
`offset`	within the DAC range minus the tones' peak	one DAC step (138 µV)
`am.depth`	about ±1.8 /V on an LV input (±0.1 /V on HV)	about $1.4 \cdot 10^{-5}$ /V on LV (one step of the 18-bit depth word)

How settings take effect:

- **Coherent mode.** `apply()` changes all tones of both outputs on one sample, apply_delay after the call. Each tone then has phase `phase + 360 f (t − t0)` exactly.
- **Continuous mode.** The change is applied at once, and the tones' phases continue (no phase jumps).
- **Headroom.** Unless `allow_clip` is set, `apply()` refuses an output whose |offset| + the sum of its amplitudes can exceed the DAC range. With AM on, the amplitudes count with the largest factor the input can produce.
- **Board calibration.** Amplitudes are volts at the connector through the DC calibration and, if measured, the output's AC response. Phases are corrected for the output's calibrated delay, so that they refer to the connector (`../../common/CALIBRATION.md`).

Returned values are the realized ones: the frequency of the 48-bit word, the amplitude of the 17-bit code, and the phase of the 16-bit word.

Python methods

In addition to the uniform API (`config`, `apply`, `describe`, `status`, `safe`, `presets`, `load_preset`, `calibration`, `calibrate`, ...):

Method	Action
`set_tone(output, tone, frequency=None, amplitude=None, phase=None)`	one tone; returns its applied settings
`tones()`	`{output: {tone: settings}}` of the tones that are on
`zero_time(delay=None)`	t0 := the next APPLY sample (now + delay); re-apply. Every tone then has exactly its phase setting at t0
`trigger()`	software trigger (as a DIO0 edge)
`play(sequence, delay=None)`	play a timed sequence (below); returns `{"start", "entries", "trigger", "coherent", "t0"}`
`stop()`	stop a sequence (empty the queue; running ramps finish; an `apply()` it was waiting for is scheduled again); `config()` then reads the settings of its last executed step
`sequence_status()`	`{"state": none/queued/streaming/done/stopped/halted, "total", "pushed", "error"}`
`hardware()`	sample counter, last APPLY, trigger time, queue state, APPLY/late/trigger/clip counts
`clear_flags()`	clear the late/overflow flags and the counts
`capture(delay=0.001)`	8192 samples of IN1 and IN2: `{"start", "in1", "in2" (codes), "volts"}`
`measure_tones(inputs=None, t0=None)`	capture and measure the configured tones (lock-in): amplitude and phase relative to t0, per input (OUT1→IN1, OUT2→IN2 by default); after a step of a trigger sequence has run, relative to its trigger edge, and in a repeating sequence to the start of the repetition the capture falls in (until the next coherent `apply()`); `t0` (a sample index) chooses another reference
`counter()`	the sample counter now
`set_am(output, source="off", depth=0.0)`	amplitude modulation of an output by IN1 or IN2 (depth in 1/V); returns the applied settings
`spectrum(channel="in1", averages=4)`	the input's amplitude spectrum from captures: `{"frequency", "dbv"}` (dB re 1 V peak per 15.26 kHz bin; numpy); `measure.analyze(spectrum, f)` gives levels, harmonics, largest spur and floor
`measure_response(output, frequencies=None, amplitude=0.2)`	the OUTn → INn loop gain at each frequency, `[[Hz, gain], ...]`, ready for `set_output_response`
`phase_difference(frequency=10e6, amplitude=0.2)`	IN1 − IN2 phase with one tone on both outputs; with cables straight and swapped, `measure.skew(a, b)` is how much later OUT2 is than OUT1 (for `set_output_delay`)

`measure_tones()`, `spectrum()` and the two calibration helpers start their capture only after a scheduled `apply()` has taken effect. The helpers set a test tone and restore the previous configuration afterwards. The functions are in `control/synth/measure.py` (`from synth import measure` on a PC, `from pynq.overlays.synth import measure` on the board). The board-calibration methods `set_output_response(n, points)` and `set_output_delay(n, seconds)` are common to all designs.

`safe()` mutes at once: all amplitudes become 0, the queue empties and any sequence stops. It then applies the default configuration: all tones off, both outputs at 0 V.

```
# on the board (Jupyter, or a script run with sudo /usr/local/share/pynq-venv/bin/python)
import rpfpga                                   # must come first
from pynq.overlays.synth import DRIVER
synth = DRIVER()                                # attaches, or loads synth if needed
synth.set_tone(1, 1, 10e6, 0.25)                # OUT1 tone 1: 10 MHz, 0.25 V
synth.set_tone(2, 1, 10e6, 0.25, -90)           # OUT2 tone 1: 10 MHz, 90 degrees behind OUT1
print(synth.describe())

# on a PC, from the repository root
import sys; sys.path[:0] = ["common/python", "synth/control"]
from synth import SynthClient
with SynthClient("192.168.0.149") as synth:
    synth.apply({"outputs": {"1": {"tones": {str(k): {"frequency": k * 1e6, "amplitude": 0.05}
                                             for k in range(1, 11)}}}})
    print(synth.measure_tones()["inputs"]["in1"])   # with OUT1 cabled to IN1
```

Sequences

`play()` takes a dict:

```
{"trigger": False,           # True: wait for DIO0_P (or trigger()); times are relative to the edge
 "phase_mode": "coherent",    # steps load the phase relative to t0 (the edge, with a trigger)
 "repeat": 1,                 # play the steps this many times (0: until stopped)
 "period": 0.0,               # seconds between repetitions (with a trigger: each waits for an edge)
 "steps": [
    {"at": 0.0, "tones": {"1.1": {"frequency": 80e6 / 2, "amplitude": 0.2}}},
    {"at": 1e-3, "ramp": 2e-4, "tones": {"1.1": {"frequency": 41e6}}, "marker": True},
    {"at": 2e-3, "ramp": 1e-4, "shape": "cos2", "tones": {"1.1": {"amplitude": 0.0}}},
    {"at": 3e-3, "outputs": {"1": {"offset": 0.0, "am": {"source": "in1", "depth": 0.5}}}}]}
```

- Tone keys are `"<output>.<tone>"`.
- A step without `ramp` changes on one sample. With `ramp` (seconds) frequency and amplitude move linearly and land exactly on the new values.
- `shape` (`linear`, `cos2`, `blackman`, `min-jerk`) shapes a ramp from `segments` (default 16, at most 256) linear pieces that land exactly on the curve; with 16 pieces the curve is followed within 0.24–0.41 % of the change (computed). Ramps longer than 134 ms are split into pieces automatically (up to 3600 s).
- `outputs` changes an output's offset, dither and AM at the step; a partial `am` (only `depth`, say) keeps the rest.
- Other tones may change during a shaped or long ramp; the ramping tone itself not until the ramp's end. A tone given twice in one step is refused.
- A tone still ramping from an earlier, stopped sequence is first stepped to that ramp's target if the new sequence's first change of it is a ramp.
- `repeat` plays the steps again by the FPGA itself (no host traffic). The whole sequence must fit in the queue (4095 entries). Every tone's first change must then be a step, and `period` must be longer than the last step. Phases in coherent mode refer to the start of each repetition, so every repetition is identical.
- `marker` pulses DIO1_P on the sample where the step takes effect.
- Steps must be in time order and at least 0.5 µs apart, plus 8 ns per changed word.
- Chirps are limited to 119 kHz/µs; faster changes must be steps.
- The queue holds 4096 entries: about 680 steps that change one tone each. Longer sequences need the service, whose worker streams them at about 60,000 entries per second (measured: one-tone steps every 0.1 ms for 3 s), or about 20,000 while another client polls `STATUS?` without pause (steps every 0.3 ms). Denser parts must fit in the queue.
- A long `play()` holds the board while it compiles (about 0.7–0.85 ms per step); other clients wait meanwhile.
- If the host or the worker falls behind, the late step halts the sequence (state `halted`), and the outputs keep the last applied settings.

Details are in THEORY.md.

TCP commands

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

Command	Reply
`TONE <o>.<k> <Hz> <V> [<deg>]`	set tone k of output o; `OK {...}` as applied
`TONE <o>.<k>?`	`OK {"frequency", "amplitude", "phase"}`
`TONES?`	the tones that are on, `OK {"1": {"3": {...}}, "2": {...}}`
`AM <o> OFF|IN1|IN2 [<1/V>]`	amplitude modulation of output o; `OK {"source", "depth"}` as applied
`ZERO [<s>]`	t0 := the next APPLY sample; `OK <t0>`
`TRIG`	software trigger
`PLAY <json>`	play a sequence; `OK {"start", "entries", ...}`
`PLAYZ <base64>`	the same, zlib-compressed and base64-encoded (the client uses it for sequences above 32 KB of JSON; lines are limited to 64 KB)
`STOP`	stop a sequence
`SEQ?`	sequence state (JSON)
`HW?`	counters, queue state and flags (JSON, as `hardware()`)
`CLEAR`	clear the late/overflow flags and counts
`CAPTURE? [<s>]`	`OK {"start", "in1", "in2"}` (8192 codes each)
`MEASURE? [<t0>]`	the configured tones measured on IN1/IN2 (JSON, as `measure_tones()`), optionally relative to sample index t0

Use the core commands for everything else, e.g. `CONFIG {"outputs": {"2": {"offset": 0.1}}}`, `CONFIG?`, `DESCRIBE?`, `STATUS?`, `PRESET <name>`, `SAFE`, `CAL?`, and for the output response and delay `CAL RESPONSE OUT1 [[1e6, 1.0], ...]` and `CAL DELAY OUT2 <s>`. The loopback helpers (`measure_response`, `phase_difference`, `spectrum`) run on the PC over `CONFIG` and `MEASURE?`/`CAPTURE?`. They are listed in `../../common/README.md`. The service has no authentication: keep the board on a trusted network.

Command line

```
# from a PC, through the service (acts on the running design):
PYTHONPATH=common/python:synth/control python3 -m rpfpga.cli --host 192.168.0.149 \
    --tone 1.1=10e6,0.25 --tone 2.1=10e6,0.25,-90
# on the board itself (e.g. over SSH):
sudo /usr/local/share/pynq-venv/bin/python -m rpfpga.cli synth --preset quadrature
```

synth options, applied after the common ones (`--load`, `--reset`, `--preset`, ...) and before `--safe`:

Option	Action
`--tone O.K=HZ,V[,DEG]`	set tone K of output O; repeatable
`--offset1 V`, `--offset2 V`	output offsets
`--phase-mode coherent|continuous`	the phase mode
`--zero`	t0 := the next APPLY sample
`--am1 SRC[,DEPTH]`, `--am2 SRC[,DEPTH]`	amplitude modulation of an output: SRC off, in1 or in2; DEPTH in 1/V

Each call prints one line per output and a line of hardware counters, e.g.

```
OUT1 = 1 tone: 10000000 Hz 0.25 V +0.00 deg
OUT2 = 1 tone: 10000000 Hz 0.25 V -90.00 deg
Sample 1234567890; queue 0 entries; 3 APPLYs, 0 late; clipped samples OUT1 0, OUT2 0
```

Where the state is kept

As for every design, the state is in `/run/rpfpga/synth/state.json`, which every program on the board shares. It is copied to `/home/xilinx/rpfpga/synth/state.json`, from which the service restores it after a reboot. The sequence job and the worker's heartbeat are in `/run/rpfpga/synth/`. The board calibration is in `/home/xilinx/rpfpga/calibration.json`, shared by every design.

Register map

One AXI4-Lite slave, `sy_core`, at **0x43C00000** with a **64 KiB** window (16-bit addresses). All registers are 32 bits. Writes must be whole words: partial writes are ignored. **Never access these registers outside `rpfpga.Overlay`.** An access to an unconfigured FPGA hangs the board (`../../common/README.md`, "Safe loading").

Range	Contents
`0x0000-0x00FF`	control, status, counters, queue push, capture and mean settings
`0x0800-0x0FFF`	the queue push window (write only)
`0x1000-0x1FFF`	the staging window: per-tone and per-output words
`0x8000-0xFFFF`	the capture buffer: 8192 words `{IN2, IN1}`

Reading the low word of a 64-bit pair latches its high word. Read LO, then HI.

Control and status

Offset	Name	Access	Meaning
0x000	ID	R	`0x53594E54` ("SYNT")
0x004	VERSION	R	`0x00020001` (2.1)
0x008	CTRL	W	strobes, see the table below
0x00C	STATUS	R	flags, see the table below

Offset	Name	Access	Meaning
0x010 / 0x014	COUNT_LO / HI	R	the sample counter: clocks since reset (8 ns each)
0x018	NTONES	R	tones per output (16)
0x01C	LATENCY	R	clocks from a counter value n to output sample n in the DAC stream (28)
0x020	QUEUE_LO	W	low word of the next queue entry
0x024	QUEUE_HI	W	high word; the write pushes `{QUEUE_HI, QUEUE_LO}`
0x028	QUEUE_LEVEL	R	entries not yet executed (at most 4096)
0x02C	QUEUE_DEPTH	R	4096
0x030 / 0x034	LAST_APPLY	R	the sample index n_a of the last APPLY
0x038 / 0x03C	TRIG_TIME	R	T_trig: the sample of the edge taken by the last WAIT_TRIGGER
0x040	TRIG_COUNT	R	trigger edges (pin and software) since reset
0x044	LATE_COUNT	R	late APPLYs (cleared by CLEAR_FLAGS)
0x048	APPLY_COUNT	R	APPLYs executed since reset
0x04C	MISSED_COUNT	R	trigger edges that no WAIT_TRIGGER was waiting for (cleared by CLEAR_FLAGS)
0x050 / 0x054	CLIPS1 / CLIPS2	R	saturated output samples of OUT1 / OUT2 (saturating at 2^32 − 1; cleared by CLEAR_FLAGS)
0x058	MARKER_WIDTH	RW	marker pulse length in samples (power-up 125: 1 µs; 0 counts as 1)
0x060 / 0x064	CAP_TIME	RW	start sample of a direct capture (48 bits: HI holds bits 47:32)
0x068 / 0x06C	CAP_START	R	the sample index of the first captured sample
0x070	MEAN_N	RW	samples per input mean
0x074 / 0x078	SUM1	R	sum of MEAN_N IN1 codes (signed, 48 bits sign-extended)
0x07C / 0x080	SUM2	R	the same for IN2
0x084	LOAD_PENDING	R	LOAD flags: bit k is OUT1 tone k+1, bit 16+k is OUT2 tone k+1
0x088	CAPTURE_SAMPLES	R	8192
0x08C	LIVE_OFFSETS	R	`{OUT2 offset, OUT1 offset}` as applied (16-bit stream codes)
0x090–0x09C	SCRATCH0–3	RW	driver words, no hardware function (see below)
0x0A0	PUSH_COUNT	R	queue entries accepted since reset (both push paths; a push into a full queue is not counted)

CTRL bits (write 1):

Bit	Name	Action
0	FLUSH	empty the queue, abort the entry that is waiting, end a running LOOP, clear HALTED and the LOAD flags
1	MUTE	FLUSH, plus every tone’s amplitude and ramp count set to 0 at once, a running APPLY sequence ended, LOAD flags cleared, a running LOOP ended
2	TRIGGER	software trigger, the same as an edge on DIO0_P
3	CLEAR_FLAGS	clear LATE and OVERFLOW, and the late, missed-trigger and clip counts
4	CAPTURE	arm the capture at CAP_TIME (a capture that is running is abandoned)
5	MEAN	start the input means over MEAN_N samples
6	RESUME	clear HALTED: the queue continues after the dropped entry
7	CAPTURE_STOP	abort a capture and the means

STATUS bits:

Bit	Meaning
0	queue empty
1	an APPLY is waiting for its time
2	a WAIT_TRIGGER is waiting
3	HALTED: a late APPLY without LATE_OK was dropped and the queue stopped
4	LATE (sticky): an APPLY came late
5	OVERFLOW (sticky): a push found the queue full and was dropped
6, 7, 8	capture busy, capture done, capture started late
9, 10	means busy, means done
11	the DIO0_P trigger input (synchronized level)
12	an APPLY sequence is running (staging writes wait)
13, 14	OUT1, OUT2 dither on (applied)
15	queue idle: empty and nothing executing or waiting

The driver keeps its own settings in the SCRATCH words:

- SCRATCH0 is a set of flags: bit 0 continuous phase mode, bit 1 no droop correction, bit 2 allow_clip.
- SCRATCH1 and SCRATCH2 hold t0, the low and the high word.
- SCRATCH3 holds the apply delay in µs; 0 means the default, 10 ms.

Staging window

The address of a word is `0x1000 + (output − 1)·0x800 + slot·0x40 + word·4`. Slots 0–15 are the tones (tone k+1 is slot k). Slot 16 holds the output's own words.

The rules for writes and reads:

- Writing a word stages it. While an APPLY sequence runs (at most 32 clocks), the write is held and answered (BVALID) only once it has been made. No sequence ever reads a slot that is half-written, and the queue's own STAGE entries wait the same way.
- The driver still writes the staging window directly only while the queue is idle.
- Reads return the words as written: a block-RAM mirror written by the same port as the staging RAM.
- Word 10 reads the tone's LOAD flag. Words 11–15 read 0.

Tone slot words:

Word	Name	Bits	Meaning
0 / 1	F_END_LO / HI	48 (HI: 47:32)	target frequency word, 2^-48 cycle per sample: f = F·125 MHz/2^48
2 / 3	DF_LO / HI	56 signed (HI: 55:32)	frequency step per sample during a ramp, 2^-72 cycle/sample²
4	A_END	17	target amplitude, of 2^17 − 1
5	DA	32 signed	amplitude step per sample during a ramp, 2^-41 of full scale
6	COUNT	24	0: a step; C: a ramp of C samples
7	POW	16	phase offset, 2^-16 cycle
8 / 9	P_LO / HI	48	phase-accumulator preload, used if LOAD
10	LOAD	1	writing 1 sets the LOAD flag; the next APPLY of this tone loads P

A ramp adds DF and DA for COUNT samples, starting from the tone's current values (frequency with 24 and amplitude with 24 fraction bits). F_END and A_END are used only by a step. A ramp lands exactly on the target words when the steps are DF = ceil((F_END·2^24 − F)/C) and DA = ceil((A_END·2^24 − A)/C), as the driver computes them (docs/THEORY.md, "One tone").

Output slot (16) words:

Word	Name	Meaning
0	OFFSET	signed 16-bit stream code, added before rounding to the 14-bit grid
1	FLAGS	bit 0: triangular dither of ±1 DAC step
2	AM_SOURCE	0 off, 1 IN1, 2 IN2: the input that modulates this output's tones
3	AM_DEPTH	signed 18-bit depth word D
4	AM_OFFSET	signed 16-bit input code subtracted before the product (the input's offset)

Amplitude modulation: m = sat18(((in − AM_OFFSET)·D) >> 14), and the tones' sum becomes sum + 2·((sum >> 18)·m), that is sum·(1 + m/2^17). Input sample s + 20 modulates output sample s (output sample s leaves the core 28 clocks after the counter reads s, so the path from input to output is 8 clocks). All output words change at an APPLY with the output's OUT flag.

Queue entries

Each entry is 64 bits: `[63:60]` op, `[59:32]` argument, `[31:0]` data. The executor takes one entry per clock and waits only where noted below.

Op	Name	Argument	Data	Action
0	NOP			none

Op	Name	Argument	Data	Action
1	STAGE	`[9]` output − 1, `[8:4]` slot, `[3:0]` word	the word	stage one word, as a register write would; waits while an APPLY sequence runs
2	TIME	`[15:0]` time bits 47:32	time bits 31:0	set the 48-bit time register T
3	APPLY	flags (see below)	tone mask	n_a = T (+ T_trig with REL); waits until the counter reads n_a − 4, then applies
4	WAIT_TRIGGER			wait for a trigger edge that arrives after this entry is reached; T_trig = its sample
5	CAPTURE	`[0]` REL		arm the capture at T (+ T_trig with REL)
6	BASE			T_trig := T (bits 63:48 from the counter)
7	LOOP	`[15:0]` K	`[11:0]` N	rewind by N entries (to the entry N before this one) and add T to T_trig; K repetitions in total, 0 = until FLUSH/MUTE

The APPLY tone mask has bits 0–15 for OUT1 tones 1–16 and bits 16–31 for OUT2 tones 1–16.

APPLY flags:

Bit	Name	Meaning
0	REL	T is relative to T_trig
1	MARKER	pulse DIO1_P on the DAC edge that puts sample n_a out
2	IMMEDIATE	ignore T: n_a = the clock of execution + 4
3	LATE_OK	if late, apply at once (and count it); otherwise drop the entry and halt
4, 5	OUT1, OUT2	the output's own words (offset, dither, AM) also change at n_a

A LOOP keeps the entries it repeats in the queue. QUEUE_LEVEL then counts them as well.

- The span from the loop's first entry to the LOOP may be the whole queue (4096 entries, N = 4095) when nothing else is in it.
- From the first rewind on, pushes that would overwrite the loop's entries are refused (counted as OVERFLOW). The first pass is not protected: the queue does not know the loop before it reaches the LOOP.
- The LOOP after the last repetition passes on to the next entry.
- LOOPs do not nest: one repetition counter.

Times are sample indices modulo 2^48, which wraps after 26 days. An APPLY is late if, when the executor reaches it, n_a − 4 has already passed.

After an APPLY the sequencers run for 32 clocks. STAGE and APPLY wait until they finish, while TIME, WAIT_TRIGGER and CAPTURE do not. Two APPLYs can therefore be as close as 35 clocks plus one clock per STAGE between them (a TIME entry is taken in the clock of the go); the driver's sequence compiler requires 64 plus one per STAGE. An APPLY takes its tones' LOAD flags in its first clock: a FLUSH during its walk down the tones (16 clocks) clears only the flags of tones not in it.

After a trigger edge, a REL APPLY needs T of at least 8 + s samples, where s is the number of STAGE entries between the WAIT_TRIGGER and the APPLY (7 + s after the software trigger). The compiler requires 16 + s.

Push window

Writing word 2i of `0x0800–0x0FFF` (address 0x0800 + 8i) stores the low word of an entry; writing word 2i + 1 (0x0804 + 8i) pushes `{that word, the low word}`. Every address is distinct, so the driver can write up to 256 entries with one block store (PYNQ's array assignment) instead of two separate writes per entry. The order of the writes within a block must be the address order, which a block store keeps. The driver pushes blocks of 32 entries or more this way and reads PUSH_COUNT before and after. If the count is short, it empties the queue and the call fails with an error; from then on it pushes word by word (QUEUE_LO/QUEUE_HI).

Pins

- DIO0_P (G17) is the trigger input. It has a pull-down, is registered in the IOB, and is synchronized by a second register. T_trig is the clock in which the IOB register first holds the high level.
- DIO1_P (H16) is the marker output: 3.3 V LVCMOS, 8 mA, slow slew, registered in the IOB.

- Both are 3.3 V only and not 5 V tolerant (root `README.md`, equipment safety).

Verification record

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

To repeat the checks, run `make test`, `make sim`, `make`, `python3 ../common/tests/verify_build.py --design synth` and `make test-remote` (see "Changing the design" in ../README.md). `make test-hw` (loopback OUT1 → IN1, OUT2 → IN2) then checks amplitudes, phases, coherent switching, sequences and SAFE with the board's own capture. Check spurs and intermodulation with a spectrum analyzer.

2026-10-22 (night): version 2.1, fixes of a full review; build 0x64bea10e

A review of the RTL, the driver and compiler, the model, the protocol and the tools (reproduced on XSim testbenches and the register model) found these bugs; all are fixed, each with a regression check.

- **RTL** (hardware 2.1):
  - A LOOP spanning all 4096 queue entries played once and left its counters for the next loop; the span is now 4096 when the pointers meet.
  - Pushes during a repeating loop could overwrite its entries; they are now refused from the first rewind on.
  - A FLUSH during an APPLY's walk down the tones let the remaining tones skip their phase preload; the LOAD flags are now taken at the APPLY's first clock.
  - A capture re-armed in its last write clock read done and not busy.
  - ASYNC_REG on the trigger's IOB flip-flop (still packed in the IOB: 29 ILOGIC).
  - `tb_control` gained four checks: the 4096-entry trigger loop (3 passes, then a K=2 loop runs twice), 4092 pushes accepted during a 4-entry loop and the loop intact, FLUSH 6 clocks into an APPLY (16 of 16 tones loaded), and the re-armed capture.
- **Sequence compiler:**
  - From the second repetition on, words that the first pass had not restaged kept their end-of-pass values. The first change of a tone or output in a loop now stages all its words.
  - The headroom of later repetitions was not checked.
  - The loop boundary did not allow for the first step's staged words.
  - A step's out-of-range offset wrapped (−0.5 V for 1.5 V).
  - A ramp to 62.5 MHz went the "shorter way" (down to 21 MHz and back).
  - A ramp started on a tone still ramping from a stopped sequence landed far off: such tones are now first stepped to their target.
  - Steps of other tones during a shaped or long ramp were refused; the ramp pieces now go in time order with the steps.
  - Duplicate tone keys were accepted.
  - A partial `am` switched AM off.
  - The P words were restaged without LOAD.
  - Loops are limited to 4095 entries.
- **Driver:**
  - 62.5 MHz read back as −62.5 MHz, and then broke `apply(config())`.
  - A scheduled `apply()` that ran during compile, or was dropped by a FLUSH, made `config()`/`stop()` show the wrong step; `stop()` could drop the apply. The count is now read consistently, `apply.json` is forgotten on every FLUSH, and a dropped apply is scheduled again.
  - A refused `play()` stopped the running sequence; it now compiles first.
  - A failed block push while streaming skipped about 4000 entries silently; the job now halts with the error.
  - `measure_tones()` used the wrong reference in repeating sequences, before a trigger, and after a continuous-mode apply.
- **Service and protocol:**
  - CAPTURE? and MEASURE? held the board for the whole acquisition (now LONG_COMMANDS).
  - PLAYZ had no limit on the decompressed size.
  - `CAL RESPONSE` accepted an infinite frequency.
  - The client's `capture()` lacked `volts`, and the calibration setters returned different things on the driver and the client.
- **Speed:**

  - `capture()` reads the window as one block and converts with numpy; `measure_tones()` caches its window.
  - On the board, one `measure_tones()` took 152 ms (about 480 ms before).
- **Fake core:** a capture spanning an APPLY is rendered in parts.
- **Docs and manual:**
  - AM timing corrected: input sample s + 20 modulates output sample s, 8 clocks (64 ns) inside the core.
  - The manual's pulse spectrum now shows the edges as played (13 linear pieces per 8 µs edge): far sidelobes −66 dB ($\cos^2$) and −63 dB (Blackman), against −84 and −108 dB for the ideal curves.
- **Build 0x64bea10e:**
  - 11,354 LUTs, 16,201 registers, 3,794 slices (86 %), 32.5 block RAM, 68 DSP.
  - WNS +0.423 ns, WHS +0.018 ns; no errors or critical warnings.
  - `make sim`: all four testbenches passed.
  - `verify_build.py --design synth`: passed, build ID reproduced.
  - `make test` (all designs) OK; synth 82 tests.
- **On the board:**
  - `make test-remote` passed (SYNT 2.1).
  - `make test-hw`: 32 loopback checks passed, twice. IN1 − IN2 is −0.030° at 1 MHz and +0.027° at 7 MHz with the stored skew correction (−0.092° and −0.363° before).
  - Board checks of the fixes, all passed:
    - a raw 4096-entry trigger loop with K=3 ran 3 passes, then was idle, with the fourth trigger missed;
    - 62.5 MHz round trip;
    - a ramp after a stopped ramp reads 0.1025 V, against 0.1024 V for the same tone applied directly;
    - pass 2 of a repeating sequence restarts with the tone off (0.0014 V rms, against 0.1448 V in pass 1);
    - the capture block read equals word-by-word reads.
  - Streaming 6000 one-tone steps every 0.3 ms through the service ran to its end with no late APPLY:
    - while another client polled MEASURE? without pause (303 ms per poll; before, every 0.3 ms halted);
    - while another client polled STATUS? (106 ms per poll), twice, and at 0.4 ms.
- **Not fixed (low):**
  - `zero_time()` with a very short delay can come late, and t0 is then off by the lateness (LATE_OK).
  - A `resetn` pulse without reloading leaves the staging mirror's output words different from the reset live words (does not happen in use).
  - CLEAR_FLAGS in the same clock as a counted event keeps the event.

2026-10-22 (evening): recordings for the user's manual; measure_tones() reference fix

Build 0xc7afaf46 unchanged. Setup: OUT1 → IN1, OUT2 → IN2, equal cables, 50 Ω terminators at the inputs, LV. Recorded with `tools/record_manual_data.py` into `docs/manual/data/` and drawn by `tools/manual_figures.py`:

- **Two tones:** 5 MHz, 0.3 V, OUT2 −90°: IN1 − IN2 = 89.83°, amplitudes 0.309 and 0.310 V.
- **Repeating sequences, captured:** a 10 MHz pulse with $\cos^2$ and with Blackman edges every 40 µs; a chirp 2 → 5 MHz in 30 µs.
- **AM by IN2** (depth 1/V): the carrier amplitude follows the IN2 level linearly, with slope over intercept 1.000/V.
- **Loop response:** 0.1–62 MHz, both outputs; the same as the earlier record.
- **Spectra:** H2 −46 dBc and H3 −59 dBc at 10 MHz, 0.4 V; IMD3 −72 dBc at 0.2 V per tone; sum product (21 MHz) −46 dBc.
- **16-tone comb:** crest factor 1.93 measured with Schroeder phases (1.91 computed), 4.20 with equal phases (4.22 computed).
- **Phase repeatability:** 40 applies, each after an excursion. The phase vs t0 scatters by 4.3 m° RMS; IN1 − IN2 by 4.9 m°.
- **Loop transfer function:** 0.5–62 MHz in 124 steps. Group delay 349.2 ns, fitted over 1–40 MHz.
- **Bug found and fixed:** a first transfer-function sweep showed nonsense phases.
  - After a trigger or repeating sequence, `measure_tones()` kept that sequence's trigger time as its reference even after a later `apply()`, which sets every phase from t0.
  - An `apply()` now marks the job superseded, and the reference is t0 again (new check in `test_phases_after_a_trigger_sequence_refer_to_the_edge`).
  - The affected recordings (two tones, repeatability, transfer function) were made again.
- **Docs corrected:** a trigger-relative step reaches the DAC pins (T + 30)·8 ns after T_trig (LATENCY 28 + 2). The 1.0 figure T + 28 had been left in THEORY and the README. This agrees with the measured marker loop of 31 samples.

2026-10-22: version 2.0 (loops, push window, AM, shaped ramps, AC calibration); build 0xc7afaf46

Test setup: OUT1 → IN1 and OUT2 → IN2 through two equal cables with 50 Ω terminators at the inputs (LV), a wire from DIO1_P to DIO0_P; for the skew, the cables swapped at the outputs. No spectrum analyzer: spectra are from the board's own ADC.

- **Hardware 2.0.**
  - The tone keeps no ramp targets (F 72 bits with 24 fraction bits, dF 56 bits, COUNT 24 bits); ramps land exactly on the target words by rounding the steps up.
  - Queue ops BASE and LOOP: sequences repeated by the FPGA.
  - The push window (0x0800) and PUSH_COUNT (0x0A0).
  - Amplitude modulation of each output by IN1 or IN2. LATENCY 28.
- **Out of context (sy_core alone, 2026-10-21):** 244 LUTs and 355 flip-flops per tone.
  - Before the loop fix: a 19 ps violation in the queue's LOOP rewind; the loop start is now computed per buffer entry.
- **250 MHz experiment (one 16-tone bank out of context, 4 ns clock):** WNS −2.08 ns, 3,751 failing endpoints. The ramp counter's zero test feeding the 72-bit frequency adder takes 6.05 ns (18 carry stages). Not pursued (docs/THEORY.md, "Scaling").
- **Builds.**
  - First 2.0 build (0xf137db9d): WNS −0.067 ns on 12 endpoints, the LOOP counter (from the read-ahead buffer through the count tests to `loop_left`). The count tests are now decoded when an entry enters the buffer.
  - **0xc7afaf46:** 11,327 LUTs, 16,159 registers (1,330 fewer than 1.0), 3,893 slices (88.5 %), 32.5 block RAM, 68 DSP. WNS +0.390 ns, WHS +0.012 ns; no DRC violations, no errors or critical warnings in `runme.log`.
  - `make sim`: `tb_sysid`, `tb_sine`, `tb_system`, `tb_control` passed. `verify_build.py --design synth`: all checks passed, build ID reproduced (0xc7afaf46).
- **`make test`** (all designs) OK; synth: model (randomized exact landing, back-to-back ramps, shaped and long ramps, AM bit-exact), driver (loops, block pushes with a lossy fake, AM, response/delay, skew by cable swap, spectra), protocol, client, manual examples. `tb_control` and `tb_system` pass with the new vectors, which include AM and a LOOP section.
- **On the board** (`make upload`, `make install-service`):
  - `make test-remote` passed: SYNT 2.0, build 0xc7afaf46, power-up state verified.
  - `make test-hw`: **32 loopback checks passed.** 0.2 V at 1 MHz reads 0.2048 V (IN1) and 0.2053 V (IN2) through the terminators; loopback delay 346.1/345.8 ns; phase steps within 0.003°; coherent switching −0.005°; the 5-tone comb within 0.13 % and 0.08° of each tone alone; 3 × 2 APPLYs of a repeating sequence with no late one; a cos² amplitude ramp lands within 0.25 %; AM with depth 0.5/V by IN2 within 0.2 %.
  - The first run failed one check: a 3003-entry sequence had not run to its end 0.1 s after `play()`. The start allowance assumed word pushes (50 µs per entry); it now follows the push mode, and the check passed.
- **Block pushes (2.3):** 4000 entries in 6.9–7.2 ms (1.8 µs per entry), PUSH_COUNT +4000 every time; word pushes 14.9 µs per entry (8.3× slower).
- **Streaming (2.6)**, one-tone steps (6 entries each) streamed by the service's worker:
  - from the board: 30,000 steps every 0.1 ms (180,181 entries, 3.0 s) done, no late APPLY: 60,000 entries/s sustained;
  - from a PC, 6000 steps: every 0.07 ms done; every 0.05 ms halted;
  - with a second client polling `STATUS?` continuously (about 100 ms per poll): every 0.3 ms done (twice); every 0.2 ms halted. With `SEQ?` polling (29 ms per poll), 0.4 ms done.
  - Fixed along the way:
    - the start allowance now covers relocating every entry (measured 6.4 µs each), and 0.5 s for the worker to take over (with 50 ms the worker found the sequence halted: the service's reply snapshot and a slow first step under load came first);

    - `config()` during a sequence replays only the APPLYs since the last call (it replayed the whole job each time);
    - the worker fills all free queue space with block pushes.
  - A long `play()` holds the board while it compiles (6000 steps: about 5 s; other clients wait).

- **Marker to trigger (1.6):** DIO1_P wired to DIO0_P, a marked APPLY at n_a returns as T_trig = n_a + 31 samples (248 ns), for marker widths of 1, 3 and 125 samples (20 trials each, no spread). A 1-sample (8 ns) marker is caught. The pin states during FPGA configuration were not measured.

- **Levels and loop response (1.1)**, OUT → IN through 50 Ω terminators, measured / set amplitude (droop corrected; the input's response included):

MHz	0.1	1	2	5	10	15	20	25	30	35	40	45	50	55	60
OUT1→IN1	1.021	1.022	1.023	1.030	1.056	1.087	1.101	1.085	1.055	1.035	1.048	1.069	0.999	0.740	0.335
OUT2→IN2	1.021	1.025	1.025	1.033	1.062	1.095	1.107	1.082	1.042	1.024	1.052	1.103	1.051	0.758	0.319

  The table was not stored in the board calibration: it includes the inputs' response (the drop above 50 MHz is probably the inputs' anti-alias filter). Store a table measured at the outputs alone, or this one knowingly (RECIPES.md).

- **Skew (1.5)**, by cable swap. IN1 − IN2 with straight and with swapped cables:

MHz	1	5	10	20	30	40
straight	−0.083°	−0.276°	−0.500°	−0.443°	−1.197°	−3.220°
swapped	+0.015°	+0.212°	+0.625°	+2.405°	+2.916°	+1.723°
OUT2 − OUT1	−137 ps	−135 ps	−156 ps	−198 ps	−190 ps	−172 ps

  - Spread of each value (5 captures) 0.005–0.021°.
  - OUT2 is 0.14–0.20 ns earlier than OUT1, far less than the 4 ns between them on the DAC bus. The DAC latches both together.
  - Stored in the board calibration as a −0.165 ns delay of OUT2. With it, the swapped difference at 10 MHz went from +0.625° to +0.035° (predicted +0.03°), and at 30 MHz from +2.916° to +1.137° (predicted +1.14°).

- **Spectra with the board's ADC (1.2, 1.3)**, IN1, 8 averages, Blackman-Harris:

  - No tone: a line at 214 kHz of about −61 dBV and a floor near −94 dBV per bin. The line is there without any output, so it belongs to the input side or the board's supply; its origin is not known.
  - One tone of 0.4 V: harmonics H2 −58 (1 MHz), −50 (5), −46 (10), −44 (20), −50 (31.4), −54 dBc (45 MHz); H3 −42.5 dBc at 45 MHz, otherwise −58 to −80 dBc. At 0.1 V: H2 −57 to −72 dBc. Dither on or off makes no visible difference, as expected: the floor is the ADC's (−84 to −86 dBc per bin at 0.4 V).
  - Two tones at 10 and 11 MHz: IMD3 −67/−68 dBc at 0.05 V each, −71 dBc at 0.1 V, −72/−77 dBc at 0.2 V, −46/−47 dBc at 0.4 V each (0.8 V peak).
  - 16 tones of 0.05 V (1–16 MHz): crest factor 2.37 with Schroeder phases, 4.27 with equal phases (peaks 0.35 and 0.64 V at the input).
  - These figures include the ADC's distortion, so they are upper bounds for the outputs. The outputs alone, phase noise and the images were not measured.

- **AOD recipe** (8 tones, min-jerk moves of 2 MHz in 300 µs): 1705 entries, `play()` 208 ms, 33 APPLYs, no late one, the marker returned through the wire. After each move the tones read 0.081–0.083 V on IN1, and their frequencies land on the target words (within 0.2 µHz of the nominal values).

- **Not measured:** phase noise, absolute frequency, DAC images and output spurs below the ADC's, the AM path's analog delay, pin states during configuration.

2026-10-21: live settings after sequences, held staging writes; build 0xa6effd0c

- **RTL.** A staging write (AXI) is now held until no APPLY sequence runs, and is answered only then. Before, its handshake was checked two clocks before the write.
  - `tb_control` gained a monitor (no staging write while a sequence runs) and a sweep of the write's handshake across the APPLY's start.
  - With the previous `sy_regs.v` the monitor caught 2 writes inside a sequence; with the new one, none.

- **Build 0xa6effd0c.** 11,468 LUTs, 17,489 registers, 4,024 slices (91.5 %), 32.5 block RAM, 64 DSP. WNS +0.284 ns, WHS +0.021 ns.
- **`make sim`:** `tb_sysid`, `tb_sine`, `tb_system` and `tb_control` passed. **`make test`:** 49 tests, OK.
- **Driver.**
  - `config()` during and after a sequence reads the settings of the last executed step. It replays the job's entries up to APPLY_COUNT, from the words staged when the sequence started.
  - `stop()` and a halt write those live settings back into the staging window.
  - `measure_tones()` refers phases to the trigger edge after a trigger sequence, or to a given `t0` (`MEASURE? [<t0>]`).
  - The compiler's headroom bookkeeping is now incremental (a new test checks it against a full recount), and tone words are cached.
- **On the board** (atomic_clock was restored afterwards, with its configuration and playback):
  - `make test-remote` passed: build 0xa6effd0c, power-up state verified.
  - `make test-hw`: 26 loopback checks passed. Loopback delay 319.22/319.17 ns; IN1 − IN2 −0.015° at 1 MHz; coherent switching −0.0004°.
  - Compiling 3000 one-tone steps: 2.54 s (was 3.4 s).
  - Through the service: 700 steps every 0.2 ms done (play 0.86 s), and 3000 steps every 0.3 ms done (play 3.1 s), no late APPLY. In both, `config()` afterwards read the last step's frequency.
  - Stopped half-way, a 10-step sequence (1, 2, … MHz every 50 ms) read 7 MHz in `config()`, and `measure_tones()` found the tone at 7 MHz (0.119 V) on IN2.
- **Note:** restarting the service loads its last design (here atomic_clock); load synth explicitly afterwards.

2026-10-20 (evening): version 1.0 on the board, loopback

Build **0x166b29c8** (built 2026-10-20 21:27). This is the offline build below plus three RTL fixes from a code review and from timing work:

- MUTE now ends a running APPLY sequence;
- FLUSH clears the LOAD flags;
- MUTE cancels a pending marker;
- each tone's step and load decisions are registered in the bank (the previous critical path).

Implementation.

- Resources: 11,454 LUTs (65.1 %), 17,489 registers (49.7 %), 4,021 slices (91.4 %), 32.5 block RAM, 64 DSP48E1.
- Timing: WNS +0.230 ns, WHS +0.023 ns. The worst path is now in the input means.
- `make sim` through the common flow: `tb_sysid`, `tb_sine`, `tb_system` and `tb_control` passed. `tb_control` now also checks MUTE in the middle of a sequence; it fails without the fix. `make test`: 45 tests, OK, with the GUI test run in a Python with ipywidgets.
- `python3 ../common/tests/verify_build.py --design synth`: all 8 stages passed (recipe build, unchanged, missing export, vendor-HDL edit, design-RTL edit, failed build blocks upload, clean, saved-project regeneration). The recipe build reproduces build ID 0x166b29c8.
- Repository-wide `make test`: all designs, OK.

On the board (192.168.0.149, PYNQ, Cortex-A9):

- `make upload` and `make test-remote` passed: synth loaded through its driver; design SYNT 1.0, build 0x166b29c8 and the power-up state verified; preset `off`.
- `make test-hw`: **26 loopback checks passed.** OUT1 → IN1 and OUT2 → IN2 in direct loopback, without 50 Ω terminations; LV jumpers.

Check	Measured
0.2 V at 1 MHz read back	IN1 0.2142 V, IN2 0.2139 V (ratio 1.0014)
Loopback delay OUT → IN, from the 1 MHz phase	319.25 ns (OUT1), 319.19 ns (OUT2); the 7 MHz phases agree within 1.6°
IN1 − IN2 phase (both outputs set equal)	−0.019° at 1 MHz (−53 ps), −0.204° at 7 MHz (−81 ps)
Phase steps +90°, −45°, +170°	+89.997°, −45.007°, +169.994°
Coherent switching 1 MHz → 2.3 MHz → 1 MHz	1 MHz phase changed by −0.0008°
5-tone comb (1.1 … 5.5 MHz, 0.08 V, phases 20° … 100°) vs each tone alone	amplitude ratios 1.0003 … 1.0009; phase differences within 0.10° of the set ones

Check	Measured
Sequence: a step and a 10 ms ramp	both APPLYs executed, none late; the ramp's target level reached
Trigger sequence (software trigger)	waited, then ran
SAFE	all tones off; IN1 RMS 1.3 mV (noise and offset)

The IN1 − IN2 figure combines the outputs' and the inputs' skews. That they are equal within 0.1 ns means that the 4 ns between OUT1 and OUT2 on the DAC bus does not reach the connectors. Measuring with the cables swapped would separate the two skews; not done.

Loopback frequency response. One tone of 0.2 V on both outputs, in the same direct loopback (no terminations; cable type and length not recorded). This is the board-plus-cable response, **not** a property of synth:

f (MHz)	0.1	0.5	1	2	5	10	15	20	25	30	40	50	60
IN1 (V)	0.213	0.214	0.214	0.216	0.227	0.274	0.401	0.912	1.243	0.449	0.235	0.299	0.125
IN1 − IN2 (°)	+0.03	−0.01	−0.02	−0.05	−0.13	−0.36	−0.77	−2.7	+1.2	−1.3	−1.2	−0.7	−2.0

The strong peak near 20–25 MHz is consistent with an unterminated cable resonance (at 10 MHz the 1.37× agrees with atomic_clock's 0.5 V → 0.67 V in direct loopback). Measure levels with 50 Ω terminations. Not repeated with terminations yet.

Host timing (measured):

- `apply()` through the service: 109 ms (one tone) and 123 ms (all 32 tones) round trip, none late. The board-side cost was cut from 1.5 s, where the calibration file was read 136 times per call, by reading it once per compile.
- Queue pushes: 21 µs per entry, 2 writes. Before the fix: 111 µs, because of PYNQ `MMIO.write` plus a linear membership test on the 265 configuration registers.
- Register reads: 4.7 µs per word.
- Compiling 3000 one-tone steps: 3.4 s.
- A vectorized numpy store of a whole chunk (1.7 µs per entry) lost queue entries on the board (sequences then halted at their first step) and was reverted to word-by-word stores.

Sequences through the service, beyond the queue. One tone per step, 6 entries each:

Steps × spacing	Entries	Result
700 × 0.2 ms	4238	done, 700 APPLYs (4 of 4 runs)
3000 × 0.3 ms	18159	done, 3000 APPLYs
3000 × 0.2 ms	18160	halted at step 2706 (the worker fell behind)
3000 × 10 µs	18160	halted at step 676, where the queue's first 4096 entries end

So the worker sustains about 20,000 entries/s; anything denser must fit in the queue. A late step halts the queue, as intended. Earlier failures at the chunk boundary had two causes, both fixed: the worker's heartbeat was starved by the board lock, and its 0.2 s idle poll was too slow.

Also found on the board and fixed: sequences of more than about 700 steps exceeded the service's 64 KB line limit. The client now sends them compressed (`PLAYZ`).

Not yet done: the levels with 50 Ω terminations; the external trigger pin and the marker on a scope; spectrum-analyzer measurements of harmonics, intermodulation, images and the noise floor; the OUT1–OUT2 skew by cable swap.

2026-10-20: version 1.0, offline (before the board)

Build **0xd83c35ef** (`overlay/synth/synth.json`, built 2026-10-20 20:48), Vivado 2025.1, xc7z010clg400-1.

Implementation

Resource	Used	Of	Share
Slice LUTs	11,986	17,600	68.1 %
Slice registers	17,415	35,200	49.5 %
Slices	4,107	4,400	93.3 %
Block RAM tiles	32.5	60	54.2 %
DSP48E1	64	80	80 %

- **Timing at 125 MHz:** WNS +0.262 ns, TNS 0, WHS +0.017 ns, THS 0, across 40,504 endpoints (`design_1_wrapper_timing_summary_routed.rpt`).
- **Routed DRC:** two DPIP-1 warnings. The DSP cascade of each output is split into two parts at tone 9, whose C input comes from the fabric unpipelined; the path meets timing.
- **Methodology warnings:** TIMING-18, no input/output delays on the converter and DIO ports, as in the other designs.
- **Pins:** trig_i on G17 (LVCMOS33, pull-down) and marker_o on H16 (LVCMOS33, 8 mA, slow), both registered in the IOB.
- Before the full build, an out-of-context synthesis of `sy_core` measured 369 LUTs per tone. Restructuring the tone updates as single adders with multiplexed operands, applying the quadrant sign in the DSP cascade and decoding "count = 0" once per output brought this to 290 LUTs. That cut the core from 13.5k to 11.3k LUTs and raised its WNS from +0.084 ns to +0.293 ns.

Simulation

Run with the standalone XSim flow on the design's sources; see the entry above for `make sim`. Results:

- **`tb_sine`.** The model's 512 vectors match bit-exactly. A 2^18-phase sweep (step 61 of 2^-24, every table entry) matches the integer algorithm read back from the ROM. The worst error against the real sine is 9.06 LSB of 2^-21 (0.57 LSB of 2^-17).
- **`tb_system`** (`sy_core` + the vendor DAC VHDL): the scenario of `tests/vectors.py`.
  - Contents: 264 AXI writes; five APPLYs; a 300-sample frequency and amplitude ramp; coherent LOADs; a full-amplitude pair that clips OUT2 (401 clipped samples); dither; offsets; WAIT_TRIGGER on the pin; a trigger-relative APPLY; a 100-sample fade.
  - Result: all 1700 samples of both outputs equal the model's codes, on the core's streams and on the DAC pins (OUT2 high half, OUT1 low half).
  - Each APPLY started at n_a − 3. The trigger time was the pin's clock + 1. The marker rose on the DAC edge of sample n_a and lasted 3 samples, as set.
- **`tb_control`.** Passed:
  - constant and scratch registers, the latched counter, staging read-back and LOAD flags;
  - an immediate APPLY;
  - a late APPLY halting the queue, then RESUME;
  - LATE_OK;
  - FLUSH of a waiting APPLY;
  - MUTE stopping a ramp and zeroing amplitudes;
  - WAIT_TRIGGER with the software trigger, and missed-edge counting;
  - queue overflow at 4096 entries, and CLEAR_FLAGS;
  - capture at a scheduled sample (first word = sample T), a late capture, and a CAPTURE entry;
  - input means.
- **Fixed during development:**
  - the FIFO read pointer (the first entry was skipped);
  - the queue's throughput: one entry per 5 clocks made a step late, now one per clock with a read-ahead buffer;
  - the queue level now counts the buffered entries.

Offline Python tests

`make test`: 40 tests, OK (GUI test skipped: no ipywidgets in that Python).

- **Model.**
  - The generated ROM and vectors are current.
  - Sine accuracy and symmetry are as specified.

  - SFDR by FFT (2^15 samples): the sine core is above 120 dBc and the 14-bit output above 100 dBc at 1.23, 12.3, 31.4 and 47.7 MHz, with and without dither. At 10 MHz: sine 120.2 dBc, 14-bit 95.5 dBc, dithered 113.0 dBc (the floor).
  - floor_sum and the closed-form advance are exact.
  - Ramps land exactly without overshoot, and coherent loads return to the phase.
  - Output rounding, saturation and dither are as specified.
  - Sequence compiling and relocation agree with the sample model.
- **Driver**, against the register model `tests/fake_core.py`:
  - APPLY at now + apply_delay with every accumulator at F·(n_a − t0);
  - the return-to-phase property;
  - continuous mode;
  - headroom refusal and allow_clip;
  - droop correction;
  - configuration derived from the staging mirror;
  - presets, SAFE, zero_time and status;
  - sequences: steps, ramps, marker and offsets at the right samples; a trigger sequence; long sequences refused without the worker and streamed with it; a late step halting;
  - capture and loopback tone measurement (amplitudes within 0.2 mV, phases relative to t0 within 0.05°);
  - a loopback calibration through the hooks;
  - loading through FakePynq.
- **Protocol, client and CLI** over localhost.

`common/tests`: 175 tests; the uniformity test passes after this file was added.

The board steps and the later fixes are in the entry above.