mult_add 3.0: reference documentation

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

Rok Zitko, Jozef Stefan Institute

2026-11-10 (commit 6178370)

Contents

mult_add: a two-input analog computer for Red Pitaya **2**
Before connecting equipment . . . 2
Quick start . . . 3
Filtering and threshold detection . . . 4
Final clamp and protection . . . 4
Terms used here . . . 5
Presets . . . 5
Specifications . . . 5
Hardware setup . . . 6
Documentation . . . 6
Verification . . . 7
Changing the design . . . 7
Source layout . . . 7
Author . . . 8

Lab recipes **8**
Contents and test status . . . 8
1. Analog multiplier and mixer . . . 9
2. Phase-sensitive detection (lock-in) with the meters . . . 10
3. Ratio meter (X/Y) . . . 10
4. Laser-power normalization (RIN cancellation) . . . 11
5. Conditioning a lock signal for an actuator . . . 12
6. Intensity stabilization: setpoint subtraction . . . 12
7. Feed-forward . . . 13
8. Linearizing a photodiode or AOM response . . . 13
9. Rectifier, envelope detector, clipper . . . 14
10. Sum and difference amplifier . . . 14
11. Absolute difference |X − Y| . . . 15
12. Comparator with a threshold . . . 15
13. Polynomial transfer function . . . 15
14. Peak selector . . . 16
15. Protecting a device under test . . . 16
16. Mean square and RMS . . . 17
17. Checking the setup without a signal generator . . . 17
18. Scripted sweeps from a PC . . . 17
19. Filtered analog lock-in output . . . 18
20. Filtered Schmitt threshold detector . . . 18
21. Envelope detector with hysteretic indication . . . 19

Calibration and testing **19**
Calibration . . . 19
Test suites . . . 20
Manual oscilloscope checklist . . . 21

Theory of operation and numerics **22**
Signal flow . . . 22
Number representation . . . 22

Source mux and input stage . . . 22
Terms . . . 23
Weighted sums . . . 23
Fast and full mode . . . 23
Output stage . . . 24
Configuration updates . . . 25
Monitoring . . . 26
Physical units (the driver) . . . 26
Accuracy and limits . . . 27

Control reference: Python API, command line and TCP protocol **28**
Configuration . . . 28
Python API (`MultAdd` on the board, `MultAddClient` on a PC) . . . 30
Command line . . . 33
TCP protocol . . . 34
Security . . . 35

Register map **36**
Commit semantics . . . 36
Identification and control . . . 36
Configuration (shadowed) . . . 36
Post-processing configuration (shadowed) . . . 37
Status (not shadowed) . . . 38
Meters (not shadowed) . . . 38
Example: raw access from Python on the board . . . 39

Verification record **39**
2026-10-05: MADD 3.0 protected final DAC14 clamp . . . 39
2026-10-05: MADD 2.0 output processing and shared concurrent control . . . 40
2026-10-04: consistency and correctness review . . . 41
2026-10-04: calibrated test levels and X/Y clip (board) . . . 42
2026-10-04: recipes checked on the model (offline) . . . 42
2026-10-04: calibration moved into the common framework . . . 42
2026-10-04: moved onto the common framework . . . 43
2026-10-04: version 1.0, stand-alone . . . 44
Build resources and timing . . . 45

mult_add: a two-input analog computer for Red Pitaya

mult_add turns a Red Pitaya STEMlab 125-14 into a programmable analog computer. The two inputs are conditioned into **X** and **Y**, and each output is its own weighted sum of 13 functions of X and Y: X, Y, |X|, |Y|, X·Y, X/Y, X², Y², min, max, sign(X), sign(Y) and a constant. Each output then has a gain, an offset, an optional **low-pass filter**, an optional **Schmitt comparator**, a mute and a final **protectable voltage clamp**. Version **MADD 3.0** processes every sample at 125 MSa/s with a fixed pipeline delay of 456 ns (608 ns when X/Y is used), plus the filter's response time when enabled. It works as an analog multiplier, mixer, filtered lock-in, ratio meter, envelope detector or error-signal conditioner in a lab setup. All settings are in **volts at the connectors**, using the board calibration, and you control it from Jupyter, Python or a TCP socket. docs/RECIPES.md has worked lab examples.

```
X = gain_x · (IN1, IN2 or test level T1) + offset_x        Y = gain_y · (IN1, IN2 or T2) + offset_y
main_n = gain_n · Σ weight_n,i · term_i + offset_n       (not clipped to the DAC range)
main_n → optional low-pass → optional Schmitt comparator → saturation → mute → final clamp → OUTn
terms: X  Y  |X|  |Y|  X·Y/1V  (X/Y)·1V  X²/1V  Y²/1V  min(X,Y)  max(X,Y)  sign(X)·1V  sign(Y)·1V  1V
```

Before connecting equipment

Read Connecting to lab equipment: read first in the repository README. For mult_add in particular:

- **Power-up and `LOAD` start as a passthrough** (OUT1 = IN1, OUT2 = IN2). This passthrough is code for code, so it is *not* calibrated: on this board it works out to about OUT1 ≈ 1.006·IN1 − 23 mV and OUT2 ≈ 1.005·IN2 − 6 mV (calculated from the calibration, not measured directly). After power-up, the service then restores the last settings, including offsets. `LOAD` clears clamp protection and does not restore settings.

`make upload` copies/installs files; a later attachment may load the new build. `make test-remote` programs, checks power-up state, then applies its preset.

- **Clamp each output** with `set_clamp(1, vmin=..., vmax=..., protect=True)` before connecting an actuator. Protection forces enable and locks the bounds until FPGA bitstream reload; presets, `SAFE` and ordinary core reset retain it. Bounds round inward to achievable DAC14 levels. The configured, clock-running digital stream is bounded from the first DAC capture edge after the commit. Analog settling/ringing, calibration error, load changes, power and reconfiguration are outside this contract. Vendor DAC reset independently forces converter data to zero, even when zero lies outside the window.
- **`MUTE` selects the nearest allowed level to calibrated 0 V.** A positive-only window therefore gives a nonzero muted output (near its lower bound). **`SAFE` preserves protected clamps**, resets other settings and mutes both outputs. Neither command releases protection.
- **Calibration and `make test-hw` drive the outputs themselves:** the reference hold puts OUT1 at about +0.56 V and OUT2 at about +0.57 V on this board, the loopback sweep runs from about −0.83 V to +0.83 V, and `make test-hw` drives up to about ±0.8 V. Disconnect sensitive equipment first.
- **Range:** inputs (LV jumper) and outputs span about ±1.13 V on this board, into a high-impedance load (scope at 1 MΩ, DMM). Nothing has been measured into 50 Ω.

Quick start

The board must already be set up with this design installed and the control service running (../common/SETUP.md). These processing controls require the MADD 3.0 overlay and matching driver/plugin files; the driver checks the major version and build ID. Connect a function generator to IN1 and IN2 (sines, at most 0.8 V amplitude, jumpers on LV) and a scope at 1 MΩ to OUT1 and OUT2.

Jupyter (easiest): open `http://192.168.0.149:9090` (password `xilinx`), then `mult_add/mult_add.ipynb`, and run the cells. The control panel has these tabs:

Tab	Contents
Mix	weights of the 13 terms for OUT1 and OUT2, with the resulting equations
Inputs	source, gain and offset of X and Y, the test levels, the X/Y floor, the input-range (jumper) selector
Outputs	gain, offset, output enable, latency mode; clamp enable/protect, bounds and active readback; protected fields lock until reload
Post-processing	independent LPF/comparator enables, time constant or cutoff, thresholds, low/high levels, reset buttons and applied readback
Meters	mean, RMS, min and max of IN1, IN2, X, Y, OUT1, OUT2
Flags	clipping flags and counters
Presets	load, save and delete configurations
Calibration	the three calibration steps; the hold button shows the real reference voltages

In live mode, changing a control sends only that field. Uncheck **Apply changes immediately** to prepare a draft, then **Apply** it atomically. If another program changes the settings, Apply reports a conflict and keeps your draft; **Refresh** shows current readback, and **Discard draft** explicitly abandons the edits. A reload requires a new driver/client and panel. Meter polling uses a separate connection so a long reading does not block control edits.

Python on the board (a Jupyter cell, or `sudo` Python over SSH):

```
import rpfpga                                   # must come first
from pynq.overlays.mult_add import DRIVER
ma = DRIVER()
ma.load_preset("multiplier")                    # OUT1 = X·Y/1V
print(ma.describe())                            # {1: 'OUT1 = X·Y/1V', 2: 'OUT2 = Y'}
```

Python on your PC, through the board's TCP service (standard library only; `common/python` and `mult_add/control` from this repository on `PYTHONPATH`):

```
from mult_add import MultAddClient
with MultAddClient("192.168.0.149") as ma:
    ma.load_preset("multiplier")
    ma.set_output(1, gain=2, vmin=-0.5, vmax=0.5)
    print(ma.describe()[1])                    # realized equation and inward-rounded clamp window
    print(ma.meters(0.1)["out1"]["mean"])      # mean of OUT1 over 134 ms, in volts
```

The client shares the high-level control API; raw hardware methods and extra calibration measurement controls are

driver-only. Configuration helpers use one network round trip. The 10–40 ms timings measured for short calls with earlier software have not been remeasured for this release (docs/PROTOCOL.md).

TCP from any language or instrument-control program: `nc 192.168.0.149 5025`, then type `PRESET multiplier`, `OUT1 MAX 0.5`, `METERS? 0.1`. Each line gets one reply, `OK ...` or `ERR ...`.

Command line from a PC: `python3 -m rpfpga.cli --host 192.168.0.149 --preset rectifier --meters 0.1` (same `PYTHONPATH`). It also runs on the board without the service, over SSH: `sudo /usr/local/share/pynq-venv/bin/python -m rpfpga.cli mult_add --preset rectifier` (about 11 s per call).

The complete API, command-line and protocol reference is docs/PROTOCOL.md.

Filtering and threshold detection

Both processing stages are **off by default**, independently for each output. For a filtered copy of IN1 with a 10 ms time constant:

```
ma.load_preset("passthrough")
ma.set_lpf(1, enabled=True, tau_s=0.01)
print(ma.status()["lpf"][1])             # realized time constant and digital cutoff
```

To turn that filtered signal into two output levels with hysteresis:

```
ma.set_comparator(1, enabled=True,
                  lower_threshold=0.15, upper_threshold=0.25,
                  low_level=0.0, high_level=0.8, initial_high=False)
```

At or above 0.25 V the state becomes high; at or below 0.15 V it becomes low; between them it keeps its previous state. The comparator sees the filtered signal when the LPF is enabled, otherwise the unfiltered main signal. Its thresholds are in the **virtual output's volts**: the main calculation can exceed the physical output range before comparison. Low/high levels must fit the calibrated DAC range and are rounded to its actual 14-bit steps. The final mute and then the final clamp still act after both stages.

State keeps running. Bypassing a stage, muting, applying settings or loading a preset does not reset its history. Re-enabling selects its current background state. All transition samples are processed after an edit; there is no pipeline flush or settling guard, so their effect can persist for the filter's time constant or in the comparator's remembered state.

Use `ma.reset_processing(1)` deliberately when a fresh start is wanted: it seeds OUT1's filter from the current main signal and sets its comparator to `initial_high`. Threshold decisions resume on the next sample. The call returns a configuration/status snapshot, not a settled measurement. Reset just one stage with `lpf=False` or `comparator=False`; omit the output number to reset both outputs. Worked examples are recipes 19–21.

Final clamp and protection

`clamp_enabled` and `clamp_protect` default to `False` on each output. A sparse `vmin`/`vmax` edit enables clamping unless that patch explicitly sets `clamp_enabled=False`; `clamp_protect=True` always forces enable. `None` selects the corresponding converter-range endpoint. Finite bounds round inward to the physical DAC14 grid (about 138 µV here); an interval with no achievable level is rejected before writes. `status()["clamp"][n]` reports the actual bounds and protection. Protection locks both bounds and enable, even against raw register writes. It does not stop the mix, LPF or comparator from running.

For example, deliberately enable a positive-only protected window:

```
ma.set_clamp(1, enabled=True, protect=True, vmin=0.2, vmax=0.8)
print(ma.status()["clamp"][1])
ma.mute(1)                               # near +0.2 V, not 0 V
ma.safe()                                # retains the protected OUT1 window
```

Presets retain locked fields. DAC calibration constants for a protected output cannot change; automatic loopback calibration refuses any protected clamp. Reference/raw holds outside a protected window are rejected before writes; input-offset measurement may still be used (calibration). To release protection, deliberately reload the FPGA and reconnect:

```
ma.load()                                # uncalibrated passthrough; protection cleared
ma = ma.reconnect()                       # every old handle/panel is stale
```

Reconnect alone does not reload or release protection. The clamp examples are not startup actions; recipe 15 explains the digital bound and physical limits.

Terms used here

Word	Meaning
X, Y	the conditioned inputs: X = input gain · source + input offset, likewise Y. The source is IN1, IN2 or a test level.
input gain, input offset	the settings of X and Y (gain −128…+127.996; offset in volts)
test level (T1, T2)	an internal constant that can replace IN1 or IN2 as the source of X (T1) or Y (T2), quantized on that input's calibrated LV scale; it has no analog input noise.
term	one of the 13 functions of X and Y. Products and quotients are referred to 1 V: the term X·Y/1V is 0.25 V for X = Y = 0.5 V.
weight	the coefficient of a term in an output's sum; **mix** is the set of weights of one output
output gain, output offset	main signal = output gain · (weighted sum) + output offset, before LPF/comparator
LPF	first-order low-pass filter; time constant `tau_s`, or alternative setting `cutoff_hz`
Schmitt comparator	threshold detector with memory between its lower and upper thresholds; selects `low_level` or `high_level`
clamp	enabled lower/upper limits (`vmin`, `vmax`) on the final DAC14 stream; protection locks bounds and enable until bitstream reload
mute	selects the DAC level nearest calibrated 0 V, then clips it to an enabled clamp window
floor	the smallest |Y| that X/Y divides by (`y_floor`)
fast / full mode	456 ns without X/Y, 608 ns with it, excluding LPF response and converters; chosen automatically per output
meters, flags	built-in averaging voltmeters; sticky indicators that a signal was clipped

Presets

Name	Function
`passthrough`	OUT1 = X, OUT2 = Y
`inverted`	OUT1 = −X, OUT2 = −Y
`sum_difference`	OUT1 = X + Y, OUT2 = X − Y
`multiplier`	OUT1 = X·Y/1V, OUT2 = Y
`squarer`	OUT1 = X^2/1V, OUT2 = Y^2/1V
`ratio`	OUT1 = 0.125·(X/Y)·1V (1 V at X/Y = 8), OUT2 = Y
`rectifier`	OUT1 = |X|, OUT2 = |Y|
`comparator`	OUT1 = ±0.5 V by the sign of X (threshold set with X's input offset), OUT2 likewise for Y
`min_max`	OUT1 = min(X, Y), OUT2 = max(X, Y)
`quadratic`	OUT1 = 0.1 V + 0.5·X + 0.25·X^2/1V, OUT2 = Y

Built-in presets start from the default settings: input gains 1, offsets 0, no clamp, outputs on, LPF and Schmitt stages bypassed. The `comparator` preset uses the original sign terms and has no hysteresis; enable the separate post-processing comparator for Schmitt behavior. Presets store processing settings, not signal history, and loading one does not reset the running stages. Loading a preset preserves all currently protected clamp fields. Save your own with `ma.save_preset("name")` (TCP `PRESET SAVE name`); saved presets are JSON files in `/home/xilinx/rpfpga/mult_add/presets/` on the board.

Specifications

Values for this board with its calibration (LV jumpers, high-impedance load). The board itself (bandwidth, impedance, noise, accuracy) is described in the repository README.

Quantity	Value
Processing delay	456 ns (fast), 608 ns with X/Y (full): 57/76 sampling-edge intervals, 58/77 register stages. Fixed for all bypass combinations; excludes LPF response, converters and analog front ends. Total input-to-output delay is **unmeasured**.
Output low-pass filter	full-rate first-order response; `tau_s` about 11.54 ns … 8796 s, default about 1 ms while bypassed. `cutoff_hz` is the exact digital pole's −3 dB setting, approximately `1/(2π·tau_s)` only at low frequencies. Read realized values with `status()`.
Schmitt comparator	independent enable, lower/upper thresholds on the wide virtual main/filtered signal; configurable low/high output levels on the 14-bit DAC grid (about 138 µV here); equality at a shared threshold selects high
Input and output range	about ±1.13 V
X and Y range	about ±1.13 V whatever the input gain, so the usable input range is ±1.13 V / |input gain|. Beyond it, X or Y clips (flag `x_clip`, `y_clip`).
Input gain	−128 … +127.996, realized exactly (the hardware's 1/256 steps are compensated). On this board, gains above about 73 on IN1 are rejected, because the input offset correction no longer fits (IN2: up to 128).
Input offset	about −1.14 … +1.11 V on IN1 and −1.14 … +1.13 V on IN2 at gain 1 (the sum of offset and gain × ADC offset must stay within the X range)
Weights	each output's coefficients (output gain × weight) up to about ±500 per term; resolution $7.6 \cdot 10^{-6}$ to $1.5 \cdot 10^{-5}$ of the largest coefficient of that output (coarser only if that coefficient is below about 0.016)
Output offset	about ±1.13 V, in internal code steps of about 35 µV; physical output resolves about 138 µV
Final clamp	disabled/unprotected by default; bounds intersect the DAC range and round inward to DAC14 steps (about 138 µV), including negative calibration polarity. Empty achievable intervals are rejected. Protection persists until bitstream reload.
Test levels T1, T2	about ±1.13 V, steps of 35 µV
X/Y	saturates at ±8 (flag `div_clip`); floor (smallest |Y| used) 35 µV … 1.13 V, default 7.8 mV
Meters	windows of 2^n samples, 8.2 µs … 1.07 s (a request is rounded to the nearest: 0.01 s → 8.4 ms, 0.02 s → 16.8 ms, 0.1 s → 134 ms)
Settings update	unlocked configuration commits on one clock edge; the live final clamp reaches DAC capture on the following edge. In-flight samples can combine old/new processing settings for the pipeline transit; LPF and comparator process them without flushing and may retain their effects longer. Ordinary edits preserve state. Concurrent edits cancel pending acquisitions; the 100 ms responsiveness target is unmeasured on the board.
Resolution	14-bit converters, about 138 µV per step

Hardware setup

- **Input jumpers:** LV (about ±1.13 V on this board) gives the best resolution and is the calibrated range. HV (±20 V nominal, gain not calibrated here) is for larger signals. After moving a jumper, tell the software: `ma.set_input_range("in1", "hv")`, TCP `RANGE IN1 HV`, CLI `--range in1=hv`, or the selector on the Inputs tab. Settings stay in volts. The setting belongs to the board, so every design uses it (../common/CALIBRATION.md).
- **min/max** need X and Y on the same jumper setting (both LV or both HV) and input gains within a factor of about 10 of each other; otherwise they raise `RangeError`. X/Y clips at exactly ±8 under the same conditions (beyond them it does not raise, but the clip shifts by the ratio of X's and Y's volts per code). The floor is independently converted using Y's scale (docs/THEORY.md).
- **Outputs** are calibrated into a high-impedance load: a scope at 1 MΩ, a DMM or another Red Pitaya input. Into 50 Ω the voltage will be lower; it has not been measured.
- **Calibration and `make test-hw`:** connect OUT1 → IN1 and OUT2 → IN2 with short coax cables, nothing else on those connectors (docs/CALIBRATION.md, ../common/CALIBRATION.md).
- **State:** the live settings are kept in memory on the board (`/run/rpfpga/mult_add/state.json`) and copied to the SD card (`/home/xilinx/rpfpga/mult_add/state.json`) in the background; the SD copy restores them after a reboot.

Documentation

Document	Contents
docs/RECIPES.md	worked lab configurations: mixer, lock-in, laser-power normalization, error-signal conditioning, linearization and more
docs/PROTOCOL.md	configuration format, Python API, command line, TCP commands
docs/CALIBRATION.md	how mult_add measures the board calibration, test suites, oscilloscope checklist
docs/THEORY.md	how it works inside: exact definition of every term, numerics, divider, latency, volts-to-hardware mapping, accuracy
docs/REGISTERS.md	the hardware register map, for raw access and debugging
docs/VERIFICATION.md	what was tested, when, with which result; build resources and timing
../common/CALIBRATION.md	the board calibration: model, procedure, accuracy, input ranges
../common/README.md	the shared framework: build workflow, safe loading, service, common API
AGENTS.md	notes for maintainers and coding agents

Verification

2026-10-05: MADD 3.0 build 0xcc5f666c, seven simulation benches, eight build-workflow stages and 273 Python tests passed; physical-board deployment/testing is pending (record and earlier board results).

Not measured yet

- the total input-to-output delay (the 456/608 ns core edge delays exclude LPF response);
- MADD 3.0 on a physical board, including protected clamp, LPF/Schmitt waveforms and concurrent-control response time;
- the output voltage into 50 Ω;
- the HV input gain (nominal value used);
- the outputs before the FPGA is configured at power-up;
- frequency response and noise spectrum through mult_add;
- waveforms from an external generator on a scope (none recorded);
- recipes 4–8 and 19–21 of docs/RECIPES.md on hardware.

Changing the design

For maintainers. Building needs Vivado 2025.1 (setup: ../common/SETUP.md). The targets are the common ones (`make help`, ../common/README.md):

```
source <Vivado install>/2025.1/Vivado/settings64.sh
make                        # create the project (first time), build and export the overlay
make test-remote            # upload, program, verify ID/build and power-up state, apply passthrough
make install-service        # the common TCP service (port 5025) with mult_add active
```

- `make test` runs the offline Python tests; `make sim` runs `tb_sysid`, `tb_divider`, `tb_core`, `tb_system`, `tb_post_lpf`, `tb_post` and `tb_clamp`.
- `make test-hw` runs the loopback test on the board (OUT1→IN1 and OUT2→IN2 cabled); `HW_CALIBRATE=1` calibrates first.
- `make test-remote` applies `PRESET` (default `passthrough`) at the end.
- `python3 ../common/tests/verify_build.py --design mult_add` checks recipe-only creation and the build's dependency handling.
- `make upload` copies the notebook only if the board does not have it yet.

TCP security: the service has no authentication. Keep the board on a trusted network, or use `SERVER_BIND=127.0.0.1` and an SSH tunnel (../common/README.md).

Source layout

Path	Contents
`src/mult_add_core.v`	top datapath, the block design's module reference, with AXI4-Lite and AXI-Stream ports
`src/ma_regs.v`	registers, shadow/commit, flags, counters
`src/scale_offset.v`	input stages (from analog_echo, plus a clip output)
`src/ma_terms.v`, `src/ma_divider.v`	term bank and pipelined divider

Path	Contents
`src/ma_mixer.v`, `src/ma_out_stage.v`, `src/ma_meter.v`	weighted sums, output stages, meters
`src/ma_post_lpf.v`, `src/ma_post_processor.v`	eight-sample lookahead LPF, Schmitt state and aligned bypass paths
`src/ma_final_clamp.v`	live final DAC14 clamp after processing, saturation and mute, with no downstream data mux
`design.tcl`	design description for the common recipe: ID "MADD", RTL, testbenches, the core's block-design cell
`control/mult_add/`	plugin package: driver, TCP commands, client, CLI options, Jupyter GUI
`control/mult_add.ipynb`	the notebook
`scripts/test_hw.py`	loopback hardware test (`make test-hw`)
`tests/`	testbenches, Python tests, the bit-exact board model
`docs/`	the documents listed above

Author

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

Lab recipes

Worked configurations for mult_add. Each recipe has the same four parts:

- **Wiring:** what goes where, the input jumper setting and the termination;
- **Settings:** Python (the on-board driver or the PC client, which have the same high-level controls) and the equivalent TCP lines;
- **Expected:** numbers you should read, and what the scope shows;
- **Limits:** what the configuration cannot do, and how far it has been tested.

All voltages are real volts at the connectors, using the board calibration. Unless a recipe says otherwise: inputs on the **LV jumper** (about ±1.13 V on this board), signals at most 0.8 V amplitude, and outputs into a **high-impedance load** (a scope at 1 MΩ, a DMM, or the high-impedance input of the next instrument). Before connecting an actuator, read Connecting to lab equipment.

```
import rpfpga                                    # must come first (on the board)
from pynq.overlays.mult_add import DRIVER
ma = DRIVER()
# or, from a PC:  from mult_add import MultAddClient; ma = MultAddClient("192.168.0.149")
```

Recipes use presets or `replace=True` (TCP `CONFIG REPLACE`) for calibrated defaults where shown; partial examples build on the selected mix. **Protected clamps survive all of these actions**, so their bounds can constrain later recipes, and an incompatible bounds edit is rejected. Only a deliberate FPGA reload releases protection (recipe 15). `ma.describe()` (TCP `DESCRIBE?`) prints the equations actually realized. The words X, Y, weight, clamp and so on are defined in the README.

Contents and test status

Recipes 1–18's settings and expected numbers were checked on the bit-exact model of the hardware, with this board's converter errors and calibration (VERIFICATION.md). The earlier hardware run of `make test-hw` checked all 13 terms on **OUT1** at DC through the loopback cables, with selected mix/mute/clamp/input-path checks on OUT2; waveforms from an external generator have not been recorded with a scope yet.

Recipes 19–21 use the processing APIs introduced in MADD 2.0 and retained in 3.0. Recipe 15 now uses the MADD 3.0 protected final clamp. Their expected numbers are calculations, not board measurements; the configurations and waveforms have not yet been tested on a physical board. LPF/Schmitt RTL simulation and digital response verification are recorded separately in VERIFICATION.md.

#	Recipe	On hardware
1	Analog multiplier and mixer	terms checked at DC
2	Phase-sensitive detection (lock-in) with the meters	terms checked at DC

#	Recipe	On hardware
3	Ratio meter (X/Y)	terms checked at DC
4	Laser-power normalization (RIN cancellation)	**not yet tested**
5	Conditioning a lock signal for an actuator	**not yet tested**
6	Intensity stabilization: setpoint subtraction	**not yet tested**
7	Feed-forward	**not yet tested**
8	Linearizing a photodiode or AOM response	**not yet tested**
9	Rectifier, envelope detector, clipper	terms checked at DC
10	Sum and difference amplifier	terms checked at DC
11	Absolute difference $\|X - Y\|$	terms checked at DC
12	Comparator with a threshold	terms checked at DC
13	Polynomial transfer function	terms checked at DC
14	Peak selector	terms checked at DC
15	Protecting a device under test	3.0 protection/positive-only mute **not yet tested**; earlier clamp/mute checked at DC
16	Mean square and RMS	terms checked at DC
17	Checking the setup without a signal generator	this is what `make test-hw` does
18	Scripted sweeps from a PC	sweep code not run on hardware
19	Filtered analog lock-in output	**not yet tested**
20	Filtered Schmitt threshold detector	**not yet tested**
21	Envelope detector with hysteretic indication	**not yet tested**

Limits of mult_add

- **No integrator or PID.** There is a first-order output low-pass filter and a Schmitt comparator per output; a PI/PID controller must still be external.
- **No internal oscillator.** A lock-in reference must come in on an input, so I/Q demodulation with two references in quadrature is not possible.
- **Two inputs only.** X and Y are built from IN1, IN2 or constant test levels.
- **Filtering and metering differ:** the optional output LPF runs at the full 125 MSa/s; meter windows provide separate software readout averages.
- **Software changes are slow:** earlier software measured about 10–40 ms per short call over the network, under 1 ms on the board; acquisitions/load take longer. The current concurrent-control path is unmeasured on hardware.

1. Analog multiplier and mixer

Wiring: signal A → IN1, signal B → IN2 (LV), OUT1 → scope (1 MΩ).

Settings:

```
ma.load_preset("multiplier")                    # OUT1 = X·Y/1V, OUT2 = Y
```

TCP: `PRESET multiplier`

Expected:

- DC: IN1 = 0.5 V and IN2 = −0.4 V give OUT1 = −0.200 V; 0.8 V × 0.8 V gives 0.640 V.
- Two sines $A\cdot\sin(2\pi f_1 t)$ and $B\cdot\sin(2\pi f_2 t)$ give $(AB/2)\cdot[\cos 2\pi(f_1-f_2)t - \cos 2\pi(f_1+f_2)t]$ / 1V. With 0.8 V at 1 MHz and 0.8 V at 1.1 MHz, the difference (100 kHz) and sum (2.1 MHz) components are each 0.32 V in amplitude. On the scope you see the 2.1 MHz signal riding on the 100 kHz beat; the whole waveform swings between −0.64 V and +0.64 V (±AB/1V).
- **Small signals:** an input gain is digital, so it does not make the ADC step (about 138 µV) or the ADC noise any smaller; only an analog preamplifier in front of the input does. It reduces only the rounding of the product (½ code, about 20 µV), which matters only for very small signals (X = 10 mV gives X^2 of about 2.6 codes). To let X fill its range for signals up to 0.1 V on IN1:

  ```
  ma.set_input("x", gain=10)                    # X = 10·IN1
  ma.set_output(1, gain=0.1)                    # OUT1 = 0.1·X·Y/1V = IN1·IN2/1V
  ```

  TCP: `INX GAIN 10`, `OUT1 GAIN 0.1`. IN1 = 0.05 V and IN2 = 0.5 V give OUT1 = 0.025 V, computed from a 0.5 V X instead of a 0.05 V one.

Limits:

- The output range is about ±1.13 V, so |A·B| / 1V must stay below that.
- With an input gain g, X clips at about ±1.13 V, so IN1 must stay below 1.13 V / g.
- Components above 62.5 MHz (half the sample rate) fold back to lower frequencies; $f_1 + f_2$ should stay well below that. The inputs have no anti-aliasing filter beyond the analog front end (about 60 MHz), so input signals and noise above 62.5 MHz alias too.
- The DAC holds each sample for 8 ns: the output droops (−0.37 dB at 20 MHz, −2.4 dB at 50 MHz) and carries images at 125 MHz − f.

2. Phase-sensitive detection (lock-in) with the meters

Wiring: the signal → IN1; a reference at the same frequency → IN2 (LV), at most ±1 V: a second generator channel (a sine), or the generator's sync output attenuated to at most 1 V. A sync output is usually a unipolar TTL square wave, so it needs the square-wave formulas below.

Settings:

```
ma.load_preset("multiplier")
m = ma.meters(0.1)                              # 134 ms average acts as the low-pass filter
mean = m["out1"]["mean"]                         # (A·B/2)·cos φ / 1V
rms_x, rms_y = m["x"]["ac_rms"], m["y"]["ac_rms"]   # A/√2, B/√2
cos_phi = mean * 1.0 / (rms_x * rms_y)           # the 1.0 is the 1 V of the product term
```

TCP: `PRESET multiplier`, then `METERS? 0.1` and read `out1.mean`, `x.ac_rms`, `y.ac_rms` from the reply.

Expected:

- For a signal A·sin(ωt + φ) and a sine reference B·sin(ωt): mean = (A·B/2)·cos φ / 1V, rms_x = A/√2, rms_y = B/√2, so cos φ = mean · 1V / (rms_x · rms_y). With a known reference amplitude B, A·cos φ = 2 · mean · 1V / B.
- Example: A = 0.5 V, B = 0.8 V, φ = 40°: mean = 0.153 V, rms_x = 0.354 V, rms_y = 0.566 V, cos φ = 0.766.
- With a square-wave reference of amplitude B (±B after removing its DC part with Y's input offset, e.g. `ma.set_input("y", offset=-0.5 * level)` for a 0…level sync signal, which gives B = level/2), X·Y is a switching demodulator: mean = (2/π)·A·B·cos φ / 1V, so A·cos φ = (π/2) · mean · 1V / B. With B = 1 V (scaled with Y's input gain): mean = 0.244 V in the example. The `cos_phi` line above assumes a sine and does not apply.
- With the LPF bypassed, OUT1 shows the product at 2ω on top of the DC value. For an on-board filtered analog lock-in output, see recipe 19.

Limits:

- The meter window is a moving average with its first null at 1/T (7.5 Hz for the 134 ms window). Choose T much longer than one signal period.
- One reference only, so you get cos φ but not the sign of φ, and no simultaneous X/Y (in-phase and quadrature) outputs.
- A DC part on X or Y adds ⟨X⟩·⟨Y⟩/1V to the mean: subtract `m["x"]["mean"] * m["y"]["mean"]`, or remove the DC with the input offsets.
- `ac_rms` includes noise and harmonics, so normalizing by it underestimates cos φ when the signal-to-noise ratio is low. With a known reference amplitude, use A·cos φ = 2 · mean · 1V / B (sine) instead.
- A reading waits for a fresh window, so it takes one to two windows plus a round trip (about 0.15–0.3 s here): this is a slow lock-in.

3. Ratio meter (X/Y)

Wiring: the signal → IN1; the normalizing signal → IN2 (LV); OUT1 → scope.

Settings:

```
ma.load_preset("ratio")                          # OUT1 = 0.125·(X/Y)·1V
ma.set_y_floor(0.05)                             # |Y| below 50 mV: divide by ±50 mV instead
ma.set_mix(1, x_div_y=1.0)                       # OUT1 = (X/Y)·1V
```

TCP: `PRESET ratio`, `YFLOOR 0.05`, `MIX1 x_div_y 1`

Expected:

- IN1 = 0.4 V, IN2 = 0.8 V → OUT1 = 0.500 V; IN1 = −0.3 V, IN2 = 0.6 V → −0.500 V.

- Below the floor: IN1 = 0.02 V, IN2 = ±0.01 V → OUT1 = ±0.400 V (0.02/0.05, with Y's sign), and the flag `div_floor` is set (`ma.flags()`).

Limits:

- X/Y saturates at ±8 (flag `div_clip`), and OUT1 at about ±1.13 V, so with weight 1 the useful ratio range is about ±1.1.
- Outputs that use X/Y run in full mode: 608 ns instead of 456 ns (automatic).
- Noise in Y is amplified by X/Y^2: choose the floor above Y's noise.

4. Laser-power normalization (RIN cancellation)

Divide an absorption or fluorescence signal by a monitor of the laser power, so that laser intensity noise (RIN) and drifts cancel.

Wiring:

- signal photodiode (after the experiment) → IN1;
- monitor photodiode (a pick-off before the experiment) → IN2;
- both LV; the photodiode outputs must stay below about 1.1 V. Amplified photodiodes rated for high-impedance loads drive the 1 MΩ inputs directly;
- OUT1 → scope or data acquisition (1 MΩ).

Settings: first measure the dark levels with the beam blocked:

```
m = ma.meters(0.1)
dark1, dark2 = m["in1"]["mean"], m["in2"]["mean"]    # e.g. 0.005 V and 0.008 V
```

Then subtract them with the input offsets and divide:

```
ma.apply({"inputs": {"x": {"offset": -dark1}, "y": {"offset": -dark2}},
          "y_floor": 0.05,                              # beam off: divide by 50 mV, not by noise
          "outputs": {"1": {"mix": {"x_div_y": 1.0}, "vmin": -0.1, "vmax": 1.0},
                      "2": {"mix": {"y": 1.0}}}}, replace=True)
# OUT1 = (X/Y)·1V with X = IN1 - dark1, Y = IN2 - dark2; OUT2 = Y (monitor)
```

TCP (with the example dark levels): `CONFIG REPLACE {"inputs":{"x":{"offset":-0.005},"y":{"offset":-0.008}},"y_floor":0.05,"outputs":{"1":{"mix":{"x_div_y":1},"vmin":-0.1,"vmax":1},"2":{"mix":{"y":1}}}}`

For an output of k·X/Y, use the weight k (for example `"x_div_y": 2.0` when the ratio is about 0.5 and you want about 1 V).

Expected (dark levels 5 mV and 8 mV):

- IN1 = 0.305 V, IN2 = 0.608 V → OUT1 = 0.500 V.
- Laser power 10% lower (IN1 = 0.275 V, IN2 = 0.548 V) → OUT1 = 0.500 V again; 5% higher → 0.500 V.
- A real change of the signal (IN1 = 0.245 V, IN2 = 0.608 V) → 0.400 V.
- Beam blocked → the mean of OUT1 within about 1 mV of 0 V. OUT1 is then noisy: X's noise divided by the floor, σ_X/floor · 1V (about 20 mV RMS for 1 mV of input noise and the 0.05 V floor), and an error δ in the dark level of IN1 appears as δ/floor · 1V.
- To measure the rejection, modulate the laser power by a few percent (for example AM on the AOM at 1 kHz) and compare the modulation line in a spectrum of `in1` and of OUT1 (FFT on a scope, a spectrum analyzer or an external lock-in). The broadband `ac_rms / mean` of the meters includes the photodiode noise, which does not cancel: with 3% modulation and 1 mV RMS of noise on each photodiode it shows only about 15 dB, however well the line itself is suppressed.

Limits:

- The cancellation is only as good as the two photodiodes track each other: matched bandwidths, and matched delays. mult_add delays both channels equally; a cable or photodiode delay difference τ limits the rejection at frequency f to roughly $2\pi f\tau$ (1 ns: about −44 dB at 1 MHz).
- Keep the monitor signal large (a few hundred mV): the quotient's noise grows as 1/Y.
- Division cancels only multiplicative intensity noise common to both beams, and only for a signal linear in the laser power (not, for example, saturated fluorescence or optical pumping).
- Additive noise of the monitor (detector and shot noise) does not cancel; it adds in quadrature, $\sigma^2_{rel,out} = \sigma^2_{rel,signal} + \sigma^2_{rel,monitor}$. Give the monitor enough photocurrent that its relative noise is well below the signal's.

- Dark levels or offsets that are not subtracted leave a residual of the intensity noise in the ratio.
- The pick-off ratio must be stable: polarization drifts and etalons in the monitor path appear as signal.
- Full mode: 608 ns core edge-to-edge delay, excluding converters and LPF response.
- **Not yet tested on hardware**; the numbers above are from the model.

5. Conditioning a lock signal for an actuator

Scale and shift a controller or error signal into an actuator's input window, and bound the DAC stream with a final clamp. Example: a PID controller putting out ±0.2 V, and a piezo driver that accepts 0…1 V.

Wiring: PID controller output → IN1 (LV); OUT1 → piezo driver input (high-impedance). Set the clamp *before* connecting the driver.

Settings:

```
ma.apply({"outputs": {"1": {"mix": {"x": 1.0}, "gain": 2.5, "offset": 0.5,
                            "vmin": 0.0, "vmax": 1.0}}}, replace=True)
# OUT1 = 2.5·X + 0.5 V, clamped to 0 … 1 V
```

TCP: `CONFIG REPLACE {"outputs":{"1":{"mix":{"x":1},"gain":2.5,"offset":0.5,"vmin":0,"vmax":1}}}`

Use a negative gain to flip the sign of the loop.

Expected:

- IN1 = 0 → OUT1 = 0.500 V; +0.1 V → 0.750 V; −0.1 V → 0.250 V.
- IN1 = +0.3 V → 1.000 V and −0.3 V → 0.000 V: clamped, and the flag `out1_clamped` is set.

Limits:

- **No integrator:** the PID (or PI) must be an external instrument. mult_add can sit before it (conditioning the error signal) or after it (scaling and clamping the control signal). Its optional LPF adds a first-order response.
- Delay adds phase lag of 360°·f·τ: the 456 ns inside the FPGA alone give 16.4° at 100 kHz. The converters and any enabled LPF add more.
- This example auto-enables an editable clamp. Use `ma.set_clamp(1, protect=True)` deliberately to lock its realized bounds until reload (recipe 15). Analog settling/ringing, calibration and load error are not constrained by the clamp; the stated 0/1 V endpoints are ideal values before inward DAC14 quantization.
- At power-up and after `LOAD` the output is an uncalibrated passthrough of IN1, with no clamp, until the settings are applied; a negative controller output then reaches the actuator. See the README.
- **Not yet tested on hardware**; the numbers above are from the model.

6. Intensity stabilization: setpoint subtraction

Turn a photodiode signal into an error signal, error = gain · (setpoint − PD), for an external controller that stabilizes laser power through an AOM.

Wiring: photodiode → IN1 (LV); OUT1 → the external PI controller's input (or an integrator in front of the AOM driver's amplitude input).

Settings: the constant term is the setpoint (0.4 V here):

```
ma.apply({"outputs": {"1": {"mix": {"one": 0.4, "x": -1.0}, "gain": 2.0,
                            "vmin": -0.5, "vmax": 0.5}}}, replace=True)
# OUT1 = 2·(0.4 V − X), clamped to ±0.5 V
ma.set_mix(1, replace=True, one=0.45, x=-1.0)          # new setpoint 0.45 V (one round trip)
```

TCP: `CONFIG REPLACE {"outputs":{"1":{"mix":{"one":0.4,"x":-1},"gain":2,"vmin":-0.5,"vmax":0.5}}}`, then `MIX1 SET one=0.45,x=-1`

Expected (setpoint 0.4 V):

- IN1 = 0.40 V → OUT1 = 0.000 V; 0.35 V → +0.100 V; 0.45 V → −0.100 V.
- Beam blocked (IN1 = 0) → +0.500 V (clamped).
- After the setpoint change to 0.45 V, IN1 = 0.45 V gives 0 V.

Limits:

- **Proportional only.** With loop gain G, a proportional loop leaves a fraction 1/(1+G) of a disturbance; an external integrator is needed for zero steady-state error.
- Setpoint changes are atomic and not ramped. Earlier short-call TCP timings were about 10–40 ms; the current control path has not been timed on hardware.
- **Not yet tested on hardware**; the numbers above are from the model.

7. Feed-forward

Add a scaled copy of a known disturbance or scan signal to a control signal. Example: a lock's control signal plus 0.3 times a scan ramp, or a diode laser's piezo voltage with a proportional copy for the current-modulation input.

Wiring: control signal (e.g. PID output) → IN1; feed-forward source (e.g. the scan ramp) → IN2; both LV; OUT1 (and OUT2) → the actuators (high-impedance).

Settings:

```
ma.apply({"outputs": {"1": {"mix": {"x": 1.0, "y": 0.3}, "vmin": -0.8, "vmax": 0.8},
                      "2": {"mix": {"x": 0.15}}}}, replace=True)
# OUT1 = X + 0.3·Y (clamped to ±0.8 V); OUT2 = 0.15·X
ma.set_mix(1, replace=True, x=1.0, y=0.32)            # tune the feed-forward gain
```

TCP: `CONFIG REPLACE {"outputs":{"1":{"mix":{"x":1,"y":0.3},"vmin":-0.8,"vmax":0.8},"2":{"mix":{"x":0.15}}}}`, then `MIX1 SET x=1,y=0.32`

Expected:

- IN1 = 0.2 V, IN2 = 0.5 V → OUT1 = 0.350 V, OUT2 = 0.030 V.
- IN1 = −0.1 V, IN2 = −0.6 V → OUT1 = −0.280 V.
- IN1 = 0.7 V, IN2 = 0.8 V → OUT1 = 0.800 V (clamped).
- Tune the weight of Y while watching the residual error signal on a scope.

Limits:

- This configuration uses fixed gains with the LPF bypassed. An enabled LPF would filter the entire output sum, not just the feed-forward term.
- **Not yet tested on hardware**; the numbers above are from the model.

8. Linearizing a photodiode or AOM response

Make a nonlinear actuator respond linearly to a command, by passing the command through a fitted quadratic: OUT1 = a + b·X + c·X²/1V. Example: an AOM whose diffracted power is not proportional to its amplitude-control voltage.

Wiring: command (from your sequencer or a generator) → IN1; OUT1 → the AOM driver's amplitude input (0…0.85 V here, high-impedance); photodiode in the diffracted beam → IN2. All LV.

Settings: step 1, measure the response by stepping OUT1 and reading the photodiode with the meters (the AOM is driven during the sweep):

```
import numpy as np
ma.apply({"outputs": {"1": {"mix": {}, "vmin": 0.0, "vmax": 0.85}}}, replace=True)   # OUT1 = 0 V,
 ↪  clamped
u = np.linspace(0, 0.8, 17)                           # drive voltages
p = []
for v in u:
    ma.set_mix(1, replace=True, one=v)                # OUT1 = v
    p.append(ma.meters(0.02)["in2"]["mean"])          # photodiode reading (V)
c, b, a = np.polyfit(p, u, 2)                        # drive needed for a reading: u ≈ a + b·p + c·p²
```

Step 2, apply the inverse, with the command in photodiode volts:

```
ma.apply({"outputs": {"1": {"mix": {"one": a, "x": b, "x2": c},
                            "vmin": 0.0, "vmax": 0.85}}}, replace=True)
```

TCP (example coefficients below): `CONFIG REPLACE {"outputs":{"1":{"mix":{"one":0.0022,"x":0.9582,"x2":0.3352},"vmin":0,"vmax":0.85}}}`

Expected: for a compressive response p = u − 0.2·u²/1V (an example, not a measured AOM), the fit gives a = 0.0022 V, b = 0.9582, c = 0.3352 with a worst residual of 2.6 mV. The configured OUT1 is then:

Command IN1	OUT1 (drive)	Exact inverse
0.10 V	0.101 V	0.102 V
0.30 V	0.320 V	0.321 V
0.50 V	0.565 V	0.564 V
0.65 V	0.767 V	0.768 V

so the photodiode reads the command to within about 3 mV. Check it by stepping IN1 and comparing `ma.meters(0.02)["in1"]["mean"]` with `["in2"]["mean"]`.

Limits:

- Quadratic only, and only within the fitted range; a strongly curved response (an AOM near saturation) needs a narrower range.
- Static: the AOM's response time and thermal effects are not corrected.
- The x^2 term is rounded to ½ code, an absolute error of about ½ code · |c| (a few µV here): negligible.
- **Not yet tested on hardware**; the numbers above are from the model.

9. Rectifier, envelope detector, clipper

Wiring: signal → IN1 (LV); OUT1 → scope.

Settings: full-wave:

```
ma.load_preset("rectifier")                    # OUT1 = |X|, OUT2 = |Y|
```

Half-wave, with Y as the internal test level 0 V and max(X, Y):

```
ma.apply({"inputs": {"y": {"source": "test"}}, "test": {"t2": 0.0},
          "outputs": {"1": {"mix": {"max": 1}}}}, replace=True)   # OUT1 = max(X, 0 V)
ma.set_test_levels(t2=0.2)                     # precision clipper: max(X, 0.2 V)
ma.set_mix(1, replace=True, min=1)             # or min(X, 0.2 V)
```

TCP: `PRESET rectifier`; half-wave: `CONFIG REPLACE {"inputs":{"y":{"source":"test"}},"test":{"t2":0},"outputs":{"1":{"mix":{"max":1}}}}`, then `TEST 0 0.2` and `MIX1 SET min=1`.

Expected:

- Full-wave: IN1 = ±0.6 V → OUT1 = 0.600 V. A 0.6 V sine gives |sin| between 0 and 0.6 V with sharp cusps at the zero crossings; its mean (`ma.meters(0.01)["out1"]["mean"]`) is (2/π)·0.6 V = 0.382 V. For small amplitudes, input noise biases this: noise alone gives a mean |n| of σ·√(2/π), about 0.8 mV for σ = 1 mV.
- Half-wave: IN1 = 0.5 V → 0.500 V, −0.5 V → 0.000 V.
- Clipper at 0.2 V: max gives 0.500 V and 0.200 V for IN1 = ±0.5 V; min gives 0.200 V and −0.500 V.
- **AM envelope:** enable OUT1's LPF with a corner between the modulation and carrier frequencies. Recipe 21 also adds a hysteretic amplitude indication.

Limits: the cusps contain harmonics above 62.5 MHz that fold back, so rectify signals well below about 10 MHz. The test level and IN1 must be on the same footing: this works with IN1 on LV, not HV.

10. Sum and difference amplifier

Wiring: signal A → IN1, signal B → IN2 (LV); OUT1, OUT2 → scope.

Settings:

```
ma.load_preset("sum_difference")               # OUT1 = X + Y, OUT2 = X − Y
ma.set_output(1, gain=0.5)                     # OUT1 = (X + Y)/2, the common mode
```

TCP: `PRESET sum_difference`, `OUT1 GAIN 0.5`

Expected: IN1 = 0.3 V, IN2 = 0.1 V → OUT1 = 0.200 V, OUT2 = 0.200 V. IN1 = −0.4 V, IN2 = 0.6 V → OUT1 = 0.100 V, OUT2 = −1.000 V.

Limits:

- The match between IN1 and IN2 needs the DMM step of the calibration (../../common/CALIBRATION.md). With it, X − Y for IN1 = IN2 = 0.5 V is within about 0.1 mV of 0 V. Starting from nominal ADC gains without it leaves their true gain difference (0.75% on this board): X − Y reads +3.7 mV (a common-mode rejection

of about 43 dB) and X + Y reads 0.996 V instead of 1.000 V. A subsequent loopback without a new DMM reference retains existing calibrated ADC gains.
- The calibration is at DC: the two front ends' frequency responses are not matched, so the rejection of AC common-mode signals is not calibrated.
- For more gain, use the input gains, keeping X and Y within ±1.13 V.

11. Absolute difference |X − Y|

Wiring: A → IN1, B → IN2 (LV); OUT1 → scope.

Settings: max(X, Y) − min(X, Y) = |X − Y|:

```
ma.apply({"outputs": {"1": {"mix": {"max": 1, "min": -1}}}}, replace=True)
```

TCP: `CONFIG REPLACE {"outputs":{"1":{"mix":{"max":1,"min":-1}}}}`

Expected: IN1 = 0.3 V, IN2 = −0.2 V → OUT1 = 0.500 V; IN1 = −0.1 V, IN2 = 0.6 V → 0.700 V.

Limits: min and max need both inputs on the same jumper setting. To detect "|A − B| > 0.1 V", enable OUT1's post-processing comparator, for example with lower/upper thresholds 0.08/0.10 V and low/high levels 0/0.8 V. The same hysteresis rules as recipe 20 apply to the absolute-difference signal.

12. Comparator with a threshold

Wiring: signal → IN1 (LV); OUT1 → scope or a high-impedance gate input.

Settings:

```
ma.load_preset("comparator")                    # OUT1 = 0.5·sign(X)·1V
ma.set_input("x", offset=-0.2)                  # X = IN1 − 0.2 V: threshold at +0.2 V
ma.set_mix(1, sign_x=0.5, one=0.5)              # 0 V below, +1 V above the threshold
```

TCP: `PRESET comparator`, `INX OFFS -0.2`, `MIX1 SET sign_x=0.5,one=0.5`

Expected: OUT1 = 1.000 V for IN1 above 0.2 V (0.21 V is enough), 0.000 V below. A slow triangle on IN1 gives a 0/1 V square wave switching at +0.2 V.

Limits:

- The output is **0 V / 1 V into a high-impedance load, not a TTL or CMOS logic level.** For a TTL trigger, follow OUT1 with an external comparator or logic buffer.
- This sign-term preset has no hysteresis: noise near the threshold toggles it at up to the sample rate. Use the separate Schmitt stage on a passthrough or other continuous-valued mix for hysteresis (recipe 20).
- With the post-processing stages bypassed, the output switches within one sample (8 ns) after the 456 ns core processing delay.

13. Polynomial transfer function

Wiring: signal → IN1 (LV); OUT1 → scope.

Settings:

```
ma.load_preset("quadratic")                     # OUT1 = 0.1 V + 0.5·X + 0.25·X²/1V
ma.set_mix(1, replace=True, one=-0.05, x=1.02, x2=-0.08)   # any a + b·X + c·X²/1V
```

TCP: `PRESET quadratic`, `MIX1 SET one=-0.05,x=1.02,x2=-0.08`

Expected:

- Preset: IN1 = 0.5 V → 0.1 + 0.25 + 0.0625 = 0.4125 V.
- After the change: IN1 = 0.5 V → −0.05 + 0.51 − 0.02 = 0.440 V; −0.5 V → −0.580 V.
- Cross terms are available too: `xy` gives X·Y/1V and `y2` gives Y²/1V. Recipe 8 shows how to fit the coefficients.

Limits: second order at most; with Y = X (`set_input("y", source="in1")`), `xy` just duplicates `x2`.

14. Peak selector

Wiring: A → IN1, B → IN2 (both LV, or both HV); OUT1, OUT2 → scope.

Settings:

```
ma.load_preset("min_max")                        # OUT1 = min(X, Y), OUT2 = max(X, Y)
```

TCP: `PRESET min_max`

Expected: IN1 = 0.3 V, IN2 = 0.5 V → OUT1 = 0.300 V, OUT2 = 0.500 V. IN1 = −0.2 V, IN2 = −0.6 V → −0.600 V and −0.200 V. OUT2 follows whichever input is higher, with no diode drop.

Limits: X and Y must come from inputs on the same jumper setting (or test levels with LV inputs), with input gains within a factor of about 10 of each other. Otherwise (e.g. LV against HV) the driver raises `RangeError`.

15. Protecting a device under test

Wiring: whatever drives the device → IN1 (LV). Set/read back OUT1's clamp before connecting it to the device's high-impedance input. This example assumes a calibrated board and an initially unprotected OUT1.

Settings: a deliberately positive-only allowed window:

```
ma.load_preset("passthrough")
ma.set_clamp(1, enabled=True, protect=True, vmin=0.2, vmax=0.8)
print(ma.status()["clamp"][1])                # actual inward-rounded window and protection
ma.mute(1)                                    # near +0.2 V, NOT 0 V
ma.safe()                                     # keeps the protected window; both outputs muted
# ma.unmute(1)                                # deliberately resume the mix inside that window
```

Equivalent TCP:

```
PRESET passthrough
OUT1 CLAMP SET {"enabled":true,"protect":true,"vmin":0.2,"vmax":0.8}
OUT1 CLAMP?
MUTE 1
SAFE
```

CLI: `--clamp1 enabled=true,protect=true,vmin=0.2,vmax=0.8 --mute`. The Outputs tab has the same controls and locks protected fields with active readback. Preset loads preserve the locked window.

Expected: before muting, IN1 = 0.5 V gives about 0.5 V; −0.5 V selects the lower bound and +1.0 V the upper bound. Bounds round inward on the DAC14 grid (about 138 µV per step here). Muting and `SAFE` select the nearest permitted level to zero, near **+0.2 V**. For a wholly negative window, mute selects its upper voltage bound instead. `ma.flags()["out1_clamped"]` records whether a candidate was limited (clear with `ma.clear_flags()`). These are predictions, not physical measurements of MADD 3.0.

Release only by deliberate bitstream reload: `protect=False`, `SAFE`, presets, processing reset and ordinary core reset cannot unlock it.

```
# Run only when the resulting uncalibrated passthrough is appropriate:
ma.load()                                     # protection cleared; no settings restore
ma = ma.reconnect()                           # old handles/panels are stale
```

TCP: `LOAD mult_add`. Reconnect alone preserves the clamp. Uploading files does not itself program the FPGA; a later attachment to a changed build may do so.

Limits: the clamp follows processing, saturation and mute, with no later core mux. The configured, clock-running DAC14 stream is bounded from the first capture edge after the commit. It does not constrain analog settling/ringing, calibration or load error, power-up/loss or reconfiguration. Vendor DAC reset independently forces converter data to zero, even for this positive-only window. At boot the output is uncharacterized before configuration, then raw passthrough until settings are restored. The digital protection survives ordinary core reset.

Automatic loopback calibration refuses any protected clamp; out-of-window reference/raw holds and changes to its DAC calibration constants are also refused. Measure calibration before locking (CALIBRATION.md). **Protection and this positive-only mute example have not been tested on a physical board**; RTL/build evidence is in VERIFICATION.md.

16. Mean square and RMS

Wiring: signal → IN1 (LV); OUT1 → scope.

Settings:

```
ma.load_preset("squarer")                       # OUT1 = X²/1V, OUT2 = Y²/1V
ma.set_output(1, gain=1/50)                     # OUT1 = X²/50 Ω, in volts ≙ watts
```

TCP: `PRESET squarer`, `OUT1 GAIN 0.02`

Expected:

- IN1 = 0.5 V → OUT1 = 0.250 V; IN2 = −0.7 V → OUT2 = 0.490 V.
- With the gain 1/50 and X the voltage across 50 Ω: IN1 = 0.5 V → 5.0 mV, numerically the 5 mW dissipated. The inputs are 1 MΩ, so the 50 Ω load must be external (a feed-through terminator or a tee with a 50 Ω load).
- The meters give RMS directly: `ma.meters(0.1)["x"]["rms"]`, and `ac_rms` excludes DC.

Limits: noise and offsets bias a mean square: for a signal s with a residual offset δ and noise σ, $E[(s + \delta + n)^2] = s^2 + 2s\cdot\delta + \delta^2 + \sigma^2$, so the RMS of a small signal reads high (1 mV RMS of noise adds 1 µV to X²/1V, about 10% of the X²/1V of a 3 mV signal). The rounding of X² (½ code, about 20 µV) matters only for signals of about 10 mV and below.

17. Checking the setup without a signal generator

The internal test levels replace the inputs, so the arithmetic can be checked with only a DMM or the loopback cables.

Wiring: DMM or scope on OUT1/OUT2; or OUT1 → IN1 and OUT2 → IN2 to read back with the meters.

Settings:

```
ma.apply({"inputs": {"x": {"source": "test"}, "y": {"source": "test"}},
          "test": {"t1": 0.6, "t2": -0.4},
          "outputs": {"1": {"mix": {"xy": 1}}, "2": {"mix": {"x_div_y": 0.25}}}}, replace=True)
m = ma.meters(0.01)
print(m["out1"]["mean"], m["out2"]["mean"])
```

TCP: `CONFIG REPLACE {"inputs":{"x":{"source":"test"},"y":{"source":"test"}},"test":{"t1":0.6,"t2":-0.4},"outputs":{"1":{"mix":{"xy":1}},"2":{"mix":{"x_div_y":0.25}}}}`, `METERS? 0.01`

Expected: OUT1 = 0.6 × (−0.4) = −0.240 V, OUT2 = 0.25 × (0.6/−0.4) = −0.375 V, approximately on the DMM and in the meters. Internal test settings are quantized on the calibrated LV input scales; they are not exact real voltages.

Limits: the output meters show the values sent to the converters; only a DMM or the loopback shows the real voltages. With OUT1→IN1 and OUT2→IN2 cabled, `make test-hw` checks all terms on OUT1, plus selected OUT2 paths.

18. Scripted sweeps from a PC

Wiring: A → IN1, B → IN2 (LV); OUT1 → scope.

Settings:

```
from mult_add import MultAddClient            # common/python and mult_add/control on PYTHONPATH
import numpy as np
with MultAddClient("192.168.0.149") as ma:
    ma.load_preset("sum_difference")
    for a in np.linspace(0, 1, 11):              # crossfade OUT1 from X to Y
        ma.set_mix(1, replace=True, x=1 - a, y=a)
        print(a, ma.meters(0.01)["out1"]["mean"])
```

TCP: `MIX1 SET x=0.7,y=0.3`, then `METERS? 0.01`, per step.

Expected: with IN1 = 0.4 V and IN2 = −0.2 V, OUT1 goes from 0.400 V (a = 0) through 0.220 V (a = 0.3) to −0.200 V (a = 1).

Limits:

- Each `set_mix(..., replace=True)` is one round trip and one atomic change (a `MIX1` command took about 36 ms over TCP on this network with earlier software). Without `replace=True`, `set_mix` now uses an atomic server-side patch, also one round trip. `apply()` with a full patch is one round trip.
- Each meter reading adds a round trip plus one to two windows (a window is 8.4 ms for 0.01 s), because it waits for a window that starts after the call.
- Earlier on-board measurements (Jupyter, or Python over SSH) were under 1 ms per short call; this has not been remeasured with the new control path.

19. Filtered analog lock-in output

Produce a continuously available demodulated voltage on OUT1. The meter is useful for reading that voltage, but is not part of the output signal path.

Wiring: signal → IN1; phase-coherent sine reference → IN2 (LV); OUT1 → scope or acquisition input at high impedance. Example: both 10 kHz, signal amplitude A = 0.5 V, reference B = 0.8 V, phase difference 40°.

Settings:

```
ma.load_preset("multiplier")
ma.set_lpf(1, enabled=True, tau_s=0.01)          # 10 ms, about 15.9 Hz cutoff
print(ma.status()["lpf"][1]["tau_s"])
print(ma.meters(0.1)["out1"]["mean"])
```

TCP: `PRESET multiplier`, `OUT1 LPF TAU 0.01`, `OUT1 LPF EN ON`, then `METERS? 0.1 FRESH 5` when a reading is wanted.

Expected: after settling, OUT1 ≈ (AB/2)·cos(40°)/1V = **0.153 V**. The 20 kHz product component has an ideal digital residual amplitude about 0.16 mV with this time constant, before analog/converter effects. A slowly changing demodulated level approaches its new value exponentially: about 63% of a step in 10 ms and 99.3% in 50 ms, in addition to fixed pipeline/converter delay. OUT2 remains the unfiltered reference Y.

Limits: one phase-sensitive channel; no internal oscillator or quadrature reference. This is a first-order filter, so increase the time constant if the double-frequency ripple is too large, at the cost of slower response. `cutoff_hz` is an alternative to `tau_s`, not a second simultaneous setting. Presets and enable changes preserve filter history; a fresh meter window is not a settling wait. An explicit LPF reset seeds from the instantaneous product, which need not equal its mean. **Not yet tested on a physical board.**

20. Filtered Schmitt threshold detector

Detect a slowly changing voltage without repeated switching while it remains inside a chosen hysteresis band.

Wiring: signal → IN1 (LV), OUT1 → scope at 1 MΩ. Use a slow triangle between 0.1 and 0.3 V for an initial demonstration.

Settings: one patch sets the whole threshold band and levels together:

```
ma.apply({"outputs": {"1": {
    "mix": {"x": 1},
    "lpf": {"enabled": True, "tau_s": 0.001},
    "comparator": {"enabled": True,
                   "lower_threshold": 0.15, "upper_threshold": 0.25,
                   "low_level": 0.0, "high_level": 0.8, "initial_high": False}
}}}, replace=True)
# Optional explicit restart; omit this to preserve the running history:
# ma.reset_processing(1)
```

TCP:

```
CONFIG REPLACE {"outputs":{"1":{"mix":{"x":1},"lpf":{"enabled":true,"tau_s":0.001},"comparator":{"
 ↪  enabled":true,"lower_threshold":0.15,"upper_threshold":0.25,"low_level":0,"high_level":0.8,"in
 ↪  itial_high":false}}}}
```

Optional restart: `PROCESS RESET 1 BOTH`. `OUT1 CMP?` reads settings; `STATUS?` includes the running high/low bit.

Expected: the filtered voltage reaching 0.25 V switches OUT1 to about 0.8 V; it must fall to 0.15 V to switch back to about 0 V. Between thresholds, the previous state remains. For a settled 0.1 V input stepped to 0.3 V, the ideal filtered signal reaches 0.25 V after `tau·ln(4)` ≈ **1.39 ms**, plus fixed pipeline/converter delay. That switching time is calculated, not measured.

Limits: output levels are rounded to the calibrated DAC14 grid, and any final clamp still acts on them. These are analog 0/0.8 V levels, not standard TTL/CMOS levels. To remove filtering while keeping hysteresis, use `ma.set_lpf(1, enabled=False)` (`OUT1 LPF EN OFF`). The comparator immediately follows the selected unfiltered stream after the aligned pipeline delay; its state is not reset by that change. Reset selects `initial_high` only for the reset sample; the next threshold decision can change it. **Not yet tested on a physical board.**

21. Envelope detector with hysteretic indication

Show a smoothed rectified amplitude on OUT1 and a two-level indication on OUT2. Both outputs use X so the same input drives the two calculations.

Wiring: amplitude-modulated carrier → IN1 (LV); OUT1 and OUT2 → scope (1 MΩ). Example: a 20 kHz sine whose peak amplitude changes slowly between 0.3 and 0.6 V.

Settings:

```
ma.apply({"outputs": {
    "1": {"mix": {"abs_x": 1}, "lpf": {"enabled": True, "tau_s": 0.005}},
    "2": {"mix": {"abs_x": 1}, "lpf": {"enabled": True, "tau_s": 0.005},
          "comparator": {"enabled": True,
                         "lower_threshold": 0.20, "upper_threshold": 0.30,
                         "low_level": 0.0, "high_level": 0.8}}
}}, replace=True)
```

TCP:

```
CONFIG REPLACE {"outputs":{"1":{"mix":{"abs_x":1},"lpf":{"enabled":true,"tau_s":0.005}},"2":{"mix"
↪  :{"abs_x":1},"lpf":{"enabled":true,"tau_s":0.005},"comparator":{"enabled":true,"lower_threshol
↪  d":0.2,"upper_threshold":0.3,"low_level":0,"high_level":0.8}}}}
```

Expected: for sine peak amplitude A, settled OUT1 ≈ $2 \cdot A/\pi$: **0.191 V** at A = 0.3 V and **0.382 V** at A = 0.6 V. OUT2 goes high when the smoothed rectified signal reaches 0.30 V (equivalent steady sine peak ≈ 0.471 V), and low at 0.20 V (peak ≈ 0.314 V). Between those levels it retains its state.

Limits: this measures mean absolute voltage, not a peak hold or RMS. Waveform shape and input noise change the amplitude interpretation. The 5 ms filter smooths the envelope as well as the carrier ripple; choose its corner between the envelope and rectified-carrier frequencies. The two filters have independent histories, so allow them to settle before comparing outputs. Edits, presets and mute do not clear those histories. This digital filter cannot remove aliases already created at the ADC or by rectification. **Not yet tested on a physical board.**

Calibration and testing

Calibration

The calibration belongs to the board and is shared by every design; see `../../common/CALIBRATION.md` for the model, the three-step procedure, accuracy and the input ranges. mult_add is a design that can **measure** it (`status()["calibrates"]` is true):

- `adc_codes()` reads the IN1/IN2 meters (raw ADC codes, a fresh window);
- `hold_dac_codes()` sets all weights to 0 and the output offsets to the requested codes, bypassing both LPF/comparator stages and unprotected clamps. Protected final clamps remain active; every requested code is checked before writes to either output.

What the outputs do during calibration (real volts on this board, with its calibration): the reference hold (step 2) puts OUT1 at about 0.560 V and OUT2 at about 0.573 V (code 16384, nominally 0.5 V; the Calibration tab shows the expected values on its hold button). The loopback (step 3) steps both outputs through 9 levels from about −0.83 V to +0.83 V (codes ±24000, nominally ±0.73 V), with a 16.8 ms meter window each (about 20–35 ms per level in the earlier software). The reference hold persists until `CAL RELEASE` or a superseding edit; successful loopback restores the configuration. These steps bypass only unprotected clamps. Disconnect sensitive equipment first.

MADD 3.0 protection: automatic loopback rejects if any output is protected. Reference holds and raw DAC holds outside a protected window reject before hardware writes. Changing a protected output's DAC gain/offset constants is refused, including resetting calibration if it would change those constants. Input-offset measurement may still proceed, provided the resulting configuration fits. Protection fixes the digital bounds and their voltage

interpretation until an intentional FPGA bitstream reload; SAFE, CAL RELEASE and presets do not unlock it. Reload starts an uncalibrated passthrough and requires explicit reconnect. See recipe 15.

Loopback without a new DMM reference retains stored ADC gains. They are nominal only if still uncalibrated or reset; omitting dmm does not discard an existing DMM-based gain calibration.

In mult_add, run the steps from the Jupyter **Calibration** tab, from Python (ma.measure_input_offsets(), ma.hold_reference(), ma.calibrate(dmm=...)), over TCP (CAL ...), or as part of make test-hw HW_CALIBRATE=1 (loopback step only).

Successful calibration completion (and explicit reference release) restores the processing settings and resets the LPFs from the live main signals and comparators from initial_high. This differs from ordinary edits/presets, which preserve history. The measurement does not reserve the controls: another accepted edit cancels it, and cancelled calibration does not restore old settings over that edit. A reload invalidates the handle. See the shared concurrency rules.

mult_add folds the constants into its input offsets, weights, output offsets, clamp limits, mute codes and comparator thresholds/levels (THEORY.md, "Physical units"), so its meters and settings use connector volts. Mute selects the nearest-to-zero DAC14 level and then clips it to the enabled clamp, so a protected window excluding zero gives nonzero output even on SAFE. Meters report the digital stream, not an independent measurement of analog settling, ringing or calibration error.

Test suites

Command	What it checks	Needs
make test	mult_add's offline tests (see below)	Python 3
make sim	seven XSim benches: sysid, divider, core, ADC/core/DAC system, post LPF, post processor, final clamp	Vivado 2025.1
python3 ../common/tests/verify_build.py --design mult_add	recipe-only project creation and the build dependency handling	Vivado, ~25 min
make test-remote	program the board, verify ID/version/power-up registers, apply PRESET with readback	board, SSH
make test-hw	every term on OUT1 through DAC → cable → ADC, plus selected OUT2, clamp, mute, flags, input and processing paths	board, loopback cables
make test-hw HW_CALIBRATE=1	the same, after a loopback calibration	as above

make test in this directory (offline) covers:

- the driver's volt-to-register mapping, against a bit-exact Python model of the hardware with simulated converter errors (tests/fake_board.py): every preset and random configurations match their formulas within a few ADC steps;
- calibration with hidden gain and offset errors;
- range errors, state sharing, presets, meters, flags and the CLI;
- LPF coefficient mapping, wide calibrated thresholds, DAC14 levels, dynamic processing models, explicit resets and background-state preservation;
- final-clamp inward quantization (including negative DAC gain), protected configuration/preset/SAFE behavior and calibration restrictions;
- interruptible meter/calibration acquisition, session/revision conflicts and processing GUI drafts (with the common concurrency tests);
- every TCP command and error, the client against the real server over localhost, and the hardware test script itself;
- the GUI, when ipywidgets is installed (run it on the board).

make test at the repository root runs these together with the framework tests, which include the Make/SSH workflow with fake tools (common/tests/test_workflow.py).

make sim covers:

- **tb_divider:** about 250,000 operands, random and directed (zeros, ±full scale, ties, the ×8 boundary, floors), against an exact integer reference.
- **tb_core:** over 100,000 output samples, checked one by one against an independent SystemVerilog model of

the whole datapath. The configurations:

  - every term alone at extreme weights, in fast and full mode;
  - division edge cases;
  - 100 random configurations, including extreme gains, offsets, clamps, mutes and test sources.

  It also checks the shadow/commit semantics, sticky flags (both directions) and counters, write-1-to-clear, and the meters (sums, squares, min/max, freeze, sequence). Deliberate RTL mutations (rounding, divider, clamp, commit, sign, meter freeze, flag gating) all make it fail.

- **`tb_system`:** the actual ADC/DAC VHDL around the core: power-up passthrough, DDR channel order and a configuration change through AXI.
- **`tb_post_lpf`:** lookahead LPF against the sample arithmetic model, including coefficient extremes, changing inputs and reset history.
- **`tb_post`:** wide output processing, aligned bypass, Schmitt decisions, resets, saturation and mute candidates before the external final guard.
- **`tb_clamp`:** final DAC14 bounds on both outputs through commits, raw and partial-strobe writes, invalid intervals, mute, reset, processing and valid gaps. Protected active/shadow retention is checked separately from the vendor DAC's reset-to-zero behavior in `tb_system`.
- **`tb_sysid`:** the common identification block.

`make test-hw` uses **internal test levels** as X and Y, quantized on the calibrated LV input scales, without ADC input noise in those sources. It checks the analog return via the input meters:

- all 13 terms at 3 operating points on **OUT1**, not every term on both outputs;
- combined mixes with gains and offsets;
- full-mode X/Y, clipping at ±8 and the floor flags;
- clamp limits and flags, mute;
- the input path (OUT1 → IN1 → X → OUT2);
- LPF DC gain on both outputs;
- wide-domain Schmitt thresholds, history, reset and independent inverted levels.

Tolerances are 3 mV + 0.5% when calibrated, and 30 mV + 5% when not. The outputs reach up to about ±0.8 V during the test. It restores your configuration at the end, then restarts processing from the live signal. A concurrent configuration edit cancels the test; its new settings take precedence and are not overwritten by the test's cleanup. The current script performs 59 checks. This expanded script has model-test evidence; it has **not been run on a physical board for MADD 3.0**, and is not a physical verification of protected clamps. Existing protection can constrain or reject its requested configurations. LPF dynamics and analog switching still require the scope checks below.

Manual oscilloscope checklist

Use a two-channel function generator and a scope, with inputs on LV, outputs into 1 MΩ, and amplitudes ≤ 0.8 V.

1. **Passthrough** (`PRESET passthrough`): OUT1 = IN1 and OUT2 = IN2, with a delay of a few hundred ns (fast mode).
2. **Multiplier** (`PRESET multiplier`): IN1 = 0.8 V at 1 MHz, IN2 = 0.8 V at 1.1 MHz. OUT1 shows the 2.1 MHz sum component (0.32 V amplitude) riding on the 100 kHz difference component (0.32 V amplitude); the waveform swings between −0.64 V and +0.64 V.
3. **Rectifier** (`PRESET rectifier`): a 100 kHz sine on IN1 gives |sin| on OUT1, 0 to 0.8 V, with sharp cusps at the zero crossings.
4. **Comparator** (recipe 12): a slow triangle on IN1 gives a 0/1 V square wave on OUT1 that switches at +0.2 V on the triangle.
5. **Ratio** (`PRESET ratio`, `MIX1 x_div_y 1`): IN1 = 0.4 V DC and IN2 = 0.8 V DC give OUT1 = 0.5 V. Sweep IN2 toward 0: OUT1 = 0.4 V/IN2 · 1V reaches the output limit (about 1.13 V, flag `out1_saturated`) once IN2 is below about 0.35 V. `div_clip` sets below IN2 ≈ 0.05 V (X/Y = 8), and `div_floor` only below the floor (default 7.8 mV).
6. **Clamp** (`OUT1 MAX 0.3`): any IN1 above 0.3 V is cut flat at 0.3 V.
7. **Fast vs full latency**: a square-wave edge on IN1 with `OUT1 FAST ON`, then `OUT1 FAST OFF`. The edge moves later by 152 ns (19 clocks).
8. **Filtered output** (recipe 19): observe demodulated DC and residual carrier ripple; vary `tau_s`, recording settling and digital-cutoff readback separately from the measured analog response.

9. **Filtered Schmitt** (recipes 20–21): observe both thresholds, memory between them, and independent LPF/comparator bypass. Check explicit reset and background tracking through mute/unmute.

None of these scope checks has been recorded yet (VERIFICATION.md).

Theory of operation and numerics

This document defines exactly what the hardware computes, how the driver maps physical settings (volts) onto it, and what limits the accuracy. It is meant for readers who want to know what happens inside; using mult_add does not require it. The hardware description (RTL, register-transfer level: the Verilog source of the FPGA logic) is in `src/`; the reference models that the tests check it against are `tests/tb_core.sv` (SystemVerilog) and `tests/fake_board.py` (Python).

Two notations are used throughout:

- **Fixed-point formats** Qm.n: a signed integer read as a number with n fractional bits, e.g. Q8.8 is a 16-bit value in steps of 1/256 from −128 to +127.996.
- **Codes:** the integer values the hardware works with; the driver converts them to and from volts.

Signal flow

Everything runs at **125 MSa/s** on the 125 MHz ADC clock, one sample per clock.

```
IN1, IN2, T1/T2 → source muxes → input gain/offset/sat16 → X, Y → 13-term bank
  → independent weighted sum per output (fast or full)
  → output gain/offset: unclipped 49-bit code-Q15 main
  → LPF or aligned main bypass → Schmitt or aligned signal bypass
  → round/sat16 → mute → live final DAC14 clamp → OUT1/OUT2

meters (mean, RMS, min, max): IN1, IN2, X, Y, OUT1, OUT2
sticky flags/counters: input clipping, divider limits, final output saturation/clamp
```

The LPF and comparator are independently bypassable and both continue running in the background. The comparator follows the **selected** LPF/bypass signal. No DAC-range saturation occurs between the weighted sum and the post-processing stages; input-stage saturation and the divider's own limits still apply.

Number representation

Internally every signal is a signed integer **code**. On the nominal scale, **32768 codes = 1 V**:

- A 16-bit signal spans −32768…32767 codes, i.e. −1 V…+0.99997 V nominal.
- The ADC (analog-to-digital converter) delivers 14-bit samples in the top 14 bits of 16, in steps of 4 codes.
- The DAC (digital-to-analog converter) uses the top 14 bits of the 16-bit output code.

The real converters differ from nominal. On this board, the LV inputs give 29,050 and 28,831 codes per volt and the DACs put out 13.5% and 14.3% more than nominal, so the 16-bit range corresponds to about ±1.13 V at the connectors and one 14-bit step to about 138 µV. With the input jumper at HV (±20 V nominal), one ADC code is 20× larger. The driver accounts for all of this with the board calibration (see Physical units).

Notation: `round(a / 2^k)` below always means `floor((a + 2^(k−1)) / 2^k)`, i.e. round half up. It is computed exactly with an arithmetic shift.

Source mux and input stage

`MODE[1:0]` selects X's source: 0 = IN1, 1 = IN2, 2 or 3 = test level T1. `MODE[3:2]` selects Y's source: 0 = IN1, 1 = IN2, 2 or 3 = T2. The test levels are 16-bit registers that stand in for an ADC sample. The driver fills them so that Tn uses the calibrated LV scale of INn: T1 = t1 volts uses IN1's calibrated LV codes per volt, with no offset (likewise T2 with IN2). Their range is therefore about ±1.13 V on this board. These constants are quantized to whole internal codes (about 35 µV); requested volts are not exact.

Each input stage is analog_echo's `scale_offset`:

```
X = sat16( round( src · gx / 2^8 ) + ox )        gx: signed Q8.8 (−128 … 127.996), ox: signed code
```

`sat16` (saturation to 16 bits) clamps to −32768…32767 and raises the `x_clip` / `y_clip` flag. So X and Y themselves are limited to the 16-bit range, about ±1.13 V when the source is an LV input, whatever the input gain. Latency: 1 clock for the mux, 4 for the stage.

Terms

All terms are **signed 20-bit** values on the same 1.0 = 2^15 scale, so nothing saturates inside the term bank. Even |−1| = +1 and (−1)·(−1) = +1 are exact.

#	Name	Integer definition	Physical meaning (1 V reference)
0	`x`	X	X
1	`y`	Y	Y
2	`abs_x`	|X|	|X|
3	`abs_y`	|Y|	|Y|
4	`xy`	round(X·Y / 2^15)	X·Y / 1 V
5	`x_div_y`	see Division	(X / Y) · 1 V, limited to ±8 V
6	`x2`	round(X² / 2^15)	X² / 1 V
7	`y2`	round(Y² / 2^15)	Y² / 1 V
8	`min`	min(X, Y)	min(X, Y)
9	`max`	max(X, Y)	max(X, Y)
10	`sign_x`	+32768 / 0 / −32768 for X > 0 / = 0 / < 0	±1 V (0 V at exactly 0)
11	`sign_y`	as above for Y	±1 V
12	`one`	+32768	1 V (a constant inside the sum)

`sign` is a comparator with no hysteresis. Near its threshold, input noise toggles it at up to the sample rate. To compare X against a level V, set X's input offset to −V.

Division

`ma_divider` is a fully pipelined, **exact** restoring divider (one quotient bit per stage, 23 clocks):

```
d = max(|Y|, floor)            (floor: FLOOR register, 1…32768 codes; 0 counts as 1)
q = 2^15 · |X| / d             computed exactly, rounded to nearest (ties away from zero)
q = min(q, 8 · 2^15)           saturates at ±8.0 (flag div_clip)
t5 = sign(X)·sign(Y)·q         sign(Y) taken as + when Y = 0
```

The floor prevents the quotient from becoming noise near Y = 0. When |Y| < floor, the denominator is the floor, with Y's sign (flag `div_floor`). The divider flags are raised only while some output in full mode uses X/Y.

Weighted sums

Each output has 13 weights `w` in **Q4.14** (18-bit signed, −8 … +7.99994, step 2^-14 ≈ 6.1·10^-5). Weight slots 13–15 are reserved. The sum is

```
S = round( Σ t_i · w_i / 2^7 )          S: 1.0 = 2^22 (7 extra fraction bits beyond the DAC)
```

There are 13 multiplies per output, each in one of the FPGA's dedicated multiplier blocks (DSP48), and a registered adder tree. The sum **cannot overflow**: |S| < 2^29, even with every weight at −8 and X/Y at 8.

Fast and full mode

The divider takes 23 clocks while the other terms take 4. In **full** mode (`MODE[6]`/`MODE[7]` = 0) all terms are delayed to line up with X/Y. In **fast** mode the undelayed terms are used and X/Y reads as 0. The driver picks fast mode automatically unless the output uses X/Y (`fast: "auto"`).

Path	Register stages (LATENCY register)	Sampling edge → output edge
fast	58	57 clock intervals = **456 ns**
full	77	76 clock intervals = **608 ns**

N register stages span N−1 edge-to-edge clock intervals. The table covers MADD 3.0's core only; every

LPF/comparator bypass combination has this same pipeline depth. It excludes the LPF's frequency-dependent response time. The ADC/DAC interface logic, the converter chips' own pipelines and the analog front ends add more delay that is not modeled here. Measure the total on a scope if it matters.

Output stage

```
main_q15 = S · g + off · 2^15                 g: signed Q8.8 gain register; off: signed code
processed_q15 = selected LPF/comparator result
v = sat16( round( processed_q15 / 2^15 ) )    flag outN_saturated
candidate = output_enabled ? v : mute_code    mute_code: DAC14 level nearest calibrated 0 V
dac14 = candidate >>> 2                       signed top 14 bits (truncate toward −∞)
guarded = max(lower14, min(dac14, upper14))
out = clamp_enabled ? 4·guarded : candidate   flag outN_clamped if dac14 was outside bounds
```

`main_q15` is a **signed 49-bit integer with 15 fractional bits per DAC code** (code-Q15). This is distinct from the nominal signal convention 32768 codes/V: one nominal volt is 2^30 in `main_q15`. Gain/offset take four stages; the LPF and Schmitt/bypass paths take 29 + 3; final rounding, saturation, an identity stage and mute take four. The output stage therefore keeps 40 register stages. The new `ma_final_clamp` is combinational after that pipeline; the overall 58/77-stage latency is unchanged. Clamp/mute do not feed back into either processor.

Final DAC14 clamp and protected bounds

The final guard takes the **live active** bounds, not delayed copies carried through the processing pipeline. No data mux follows it in the core. Every candidate, including a muted, reset or calibration candidate, goes through it. When enabled it asserts output valid even during input-valid gaps or pipeline refill, so the DAC can replace an out-of-window previously held sample.

The signed DAC14 range is **−8192…8191**; its 16-bit stream representation is **−32768…32764 in multiples of 4**. Candidate conversion happens before the comparison. For raw bound codes, commit uses `lower = 4·ceil(raw_lower/4)` and `upper = 4·floor(raw_upper/4)`, including negative codes. If that interval is empty/reversed (including a lower bound rounding up to +32768), the hardware retains the last valid bounds. Enable/protect requests and unrelated register changes still commit. Shadow readback is normalized to the accepted active bounds/control. The voltage API instead rejects an empty interval before writes.

Per-output protection forces enable and latches both bounds and control in **active and shadow** storage until FPGA bitstream reload. Subsequent writes, including partial byte strobes, cannot alter them. Ordinary core `resetn` leaves the protected storage intact and the final guard active; unprotected banks reset to disabled/full range. LPF/comparator resets also cannot release the guard.

If commit occurs at edge C, the new guard acts on the stream after C and is captured by the DAC interface on **C+1**, 8 ns later at 125 MHz. Thus the bound covers the configured, clock-running digital stream from that capture edge; it does not wait for the 58/77-stage processing transit. It is **not an absolute physical-voltage guarantee**: analog settling/ringing, calibration error, loading, power and reconfiguration are outside the guard. In particular the vendor DAC reset independently forces converter data to zero, irrespective of the core clamp. A positive-only or negative-only protected stream can remain bounded during core reset while the reset converter data is zero.

Mute chooses the achievable DAC level nearest calibrated zero and then clips that candidate to the enabled window. It is nonzero when the window excludes zero. `SAFE` and preset loading retain protected fields. The driver refuses changes to DAC calibration constants that would reinterpret protected bounds; automatic loopback refuses any protected clamp, and reference/raw holds outside the window are rejected before writes (CALIBRATION.md).

Full-rate low-pass filter

The intended first-order response, with sample interval T = 8 ns, is

```
y[n] = y[n−1] + alpha · (x[n] − y[n−1])
a = 1 − alpha
H(z) = alpha / (1 − a·z^−1)
tau = −T / ln(a)
fc = fs/π · asin(alpha / (2·sqrt(1−alpha)))       fs = 125 MHz
```

The driver reports the pole's exact exponential time constant and digital −3 dB frequency. `1/(2π·tau)` is only a low-frequency approximation to `fc`. The supported range is approximately alpha = 2^−40 … 0.5, or **tau = 11.54 ns … 8796 s**. Requests outside the coefficient range are rejected.

The feedback multiplication needs pipelining at 125 MHz. Simply delaying the feedback would create the wrong transfer function. `ma_post_lpf.v` instead uses an **eight-sample lookahead**:

```
b = 1 − a^8 = M · 2^−P
c_k = a^k / (1 + a^k)                              k = 1, 2, 4
F_k(z) = 1 + c_k·(z^−k − 1)
v = F_1 · F_2 · F_4 · x
y[n] = y[n−8] + b · (v[n] − y[n−8])
```

In exact arithmetic, `b·F_1·F_2·F_4 / (1−(1−b)·z^−8)` equals `H(z)`. It still accepts and produces **one sample every clock**, with no decimation. Three seven-stage FIR sections and an eight-stage feedback path account for the 29-stage pipeline. Fixed pipeline delay is separate from the response described by these sample-index equations.

The implemented coefficient set is:

- Feedback: normalized **unsigned 24-bit M**, 2^23 … 2^24−1, and **P = 24…60**; `b <= 1−0.5^8`. The driver chooses the nearest representable `b`.
- It then derives `a = (1−b)^(1/8)` from that **quantized b**, and rounds each `c_k` independently to unsigned Q24 (`C_k = round(2^24·c_k)`). Coefficients are committed together. Registers store **b**, not alpha.
- Each FIR evaluates `x + floor(C_k·(x_delayed−x)/2^24)` on code-Q15 data. This form gives exactly unity DC gain even after coefficient quantization.
- The feedback state is **signed 89-bit code-Q55**, with 40 guard bits beyond the main signal. The difference operand uses the state's code-Q15 portion; the update retains the guard bits, so very slow poles continue accumulating sub-Q15 changes. Final processor output takes the code-Q15 portion.

Q24 rounding means pole cancellation is not exact: the actual response has small residues from the eight-sample realization. The reported `tau_s` and `cutoff_hz` describe the real pole a and its corresponding one-pole response, not an assertion of a mathematically perfect one-pole transfer at every frequency. Numerical frequency-response verification across **124 tested coefficient settings**, including alpha = 2^−40, found response residue **no greater than 0.605 ppm** in that sweep. This is digital numerical verification, not a measured analog response or a bound proven over every setting. Arithmetic truncation adds small code-Q15 errors as well; converter noise, calibration error and the 14-bit output grid are separate effects.

Schmitt comparator

Threshold decisions use calibrated virtual output volts, for the selected filtered or unfiltered main signal:

```
if signal >= upper_threshold: high = True
elif signal <= lower_threshold: high = False
else: retain high
selected level = high_level if high else low_level
```

Lower must not exceed upper. Equality at a zero-width band chooses **high**. `low_level` and `high_level` name states; their voltages may be inverted. The comparator state updates every sample, including in bypass and mute. Its bypass path is aligned to its three-stage decision/selection pipeline.

Thresholds use the full signed 49-bit code-Q15 range. With DAC calibration `V = G·code+B`, the threshold payload is `round((V_threshold−B)/G · 2^15)`. Thus thresholds may lie outside the physical DAC range without clipping the signal to be compared. A reverse-comparison bit handles negative G while retaining the physical lower/upper meanings. Levels instead use signed codes −32768 … 32764, rounded to **multiples of 4**: the physical DAC14 grid. Readback reports realized threshold and level volts. Selected levels subsequently pass through final saturation, mute and clamp.

Configuration updates

Every configuration register is double-buffered. Writes go to a shadow copy, and writing CTRL bit 0 (commit) updates the active set on one clock edge. Protected clamp fields remain locked; unprotected clamp fields are normalized as above. The other registers copy their shadow values together.

Settings act at different stages of the pipeline. In-flight samples can therefore combine old and new settings during its 58/77-stage transit (for example new output gain with old weights). The implementation **processes these transitional samples**: there is no flush, hold or settling guard. Their effects can remain in the LPF for its response time, or in the Schmitt state until a threshold is crossed. Waiting one pipeline transit is not a filter-settling guarantee, and muting does not freeze the state.

Ordinary commits, preset loads, bypass changes and mute/unmute preserve all processing history. An explicit processing reset is different:

- **LPF reset:** a seed token initializes FIR histories and all eight feedback histories to that token's current, unclipped main value. Following samples continue normally; none are dropped. It is not a reset to zero.
- **Comparator reset:** the corresponding token sets the bit to `initial_high`, overriding the threshold decision for that sample. The next sample resumes normal threshold decisions; changing `initial_high` alone does not reset it.
- Each output/stage can be reset independently. Reset pulses travel with the data/settings. Power-up clears hardware history and leaves both stages bypassed. Successful calibration restores the configuration and explicitly resets processing from the live signal.

Software configuration/reply transactions and revision checks are described in PROTOCOL.md. They coordinate programs; they do not flush the sample pipeline or wait for analog settling.

Monitoring

Flags (FLAGS register) are sticky: they stay set until cleared. **Counters** count limited samples per output (saturated or clamped), clipped X and Y samples, and X/Y clip events. They saturate at 2^32−1.

Meters: one window counter (2^k samples, k = 10…27, so 8.2 µs…1.07 s) covers all six signals at once. For each window the hardware latches Σx (48 bits), Σx² (64 bits), min and max, plus a sequence number. Reading sets a freeze bit so the snapshot cannot change midway. The driver converts to volts:

```
mean = Σx/N · scale + offset        RMS = sqrt( E[(x·scale + offset)²] )        AC RMS = |scale| ·
↪  std(x)
```

A window mean is a moving-average (boxcar) low-pass filter with its first null at 1/T. A requested window is rounded to the nearest power of two in log2: `meters(0.01)` uses 2^20 samples = 8.4 ms (first null at 119 Hz), 0.1 s gives 2^24 samples = 134 ms. A boxcar's sidelobes fall only as 1/(π·f·T), about −28 dB at 1 kHz for 8.4 ms, and the windows are not synchronized with the signal, so a 2f ripple leaks into the mean at about that level. That turns the multiplier into a phase-sensitive detector (lock-in) and `x2` into a mean-square power meter; see RECIPES.md.

Physical units (the driver)

The hardware works in codes; the driver (`control/mult_add/driver.py`, function `compile_config`) maps user settings in volts onto it.

Inputs. Each source has a calibrated conversion: ADC code = K·V + A, with K ≈ 32768 per volt (LV) or 1638.4 per volt (HV), from the board calibration shared by all designs (`../../common/CALIBRATION.md`). The user's input settings are X = g·V + o.

- The hardware gain is gx = round(g·256)/256, Q8.8 (unless min/max or X/Y pairing chooses the code, below), so relative steps of 1/(256·|g|): 0.4% at g = 1.
- Rather than leave that quantization in X, the driver defines X's **volts per code** as `s_X = g / (gx·K)`.
- The input offset code is `ox = o/s_X − gx·A`.

Gain quantization and ADC gain error then end up in s_X. The much finer Q4.14 weights downstream absorb s_X exactly. So X in volts is exactly g·V + o (within one code) whatever g is.

Terms. Each term's hardware value converts to physical volts with a factor built from s_X and s_Y:

Term	Volts per code
x, abs_x	s_X
y, abs_y	s_Y
xy	32768 · s_X · s_Y
x2	32768 · s_X²
y2	32768 · s_Y²
x_div_y	s_X / (32768 · s_Y)
min, max	s_X
sign_x, sign_y, one	1/32768

min/max compare X and Y codes directly, so X and Y need the same volts per code. When an output uses min, max or X/Y, the driver therefore chooses the two hardware gain codes gx and gy as a pair so that s_X and s_Y agree to

better than $2 \cdot 10^{-4}$ (in practice about 10^{-5}). Each code may range from half the requested code up to the largest that still fits the whole input range plus the offset into 16 bits without clipping; so for $|g| < 1$ the code may exceed round(g·256). The user gains stay exact, because s_X and s_Y absorb the chosen codes. This works for negative gains too and between IN1, IN2 and the test levels, as long as the inputs used are on the same jumper range (the two inputs' different ADC gains are compensated) and the two input gains are within a factor of about 10 of each other (tested up to about 14×). Beyond that, and for LV against HV (a factor of 20), no match is possible and min/max raise `RangeError`.

X/Y is matched the same way. The quotient itself is exact for any scales, but its ±8 code-ratio clip represents ±8 V only when s_X = s_Y. When no match is possible (LV against HV, or very different input gains) the clip sits at ±8·s_X/s_Y V; no error is raised. **The floor does not scale with this ratio:** the driver separately encodes it as `round(y_floor/s_Y)` (1…32768), so its realized voltage is `FLOOR·s_Y`, subject only to Y-scale quantization/range.

Test levels. Tn = t volts is written as the code round(t·K_n), with K_n the calibrated LV codes per volt of INn and no offset. A test level therefore enters the input stage on the calibrated LV scale of INn, quantized to a whole internal code, and compares with real inputs (for example in max(X, T2)).

Outputs. With the calibrated DAC (V = G·code + B), output n must satisfy

```
G · g_out · w_i = g_user · u_i · scale_i      for every term i        off = (o_user − B) / G
```

The product g_out·w_i is fixed, but the split between them is free. The driver picks the power of two g_out (1/256 … 64) that puts the largest |w_i| between 4 and 8. That gives the largest weight a relative precision of 2^-14/|w_max|, between 7.6·10^-6 and 1.5·10^-5; smaller weights of the same output keep the same absolute step.

Example (nominal scales; on this board w ≈ 7.23): OUT1 = 0.05·X·Y/1V is realized as w = 6.4 with g_out = 1/128, not as w = 0.05 (819 steps of 2^-14) with g_out = 1. Only mixes whose largest coefficient is below 4/256 ≈ 0.016 hit the g_out = 1/256 limit, and those lose relative precision in proportion.

If a request needs more than 64 × 8 = 512 times the term's natural scale, the driver raises `RangeError`, naming the term and the closest achievable value. The input offset register must also hold the ADC offset correction, scaled by the input gain (ox = o/s_X − gx·A): on this board (A = −444.6 codes on IN1), input gains above about 73 on IN1 therefore raise `RangeError` even with zero offset. Clamp limits are intersected with the physical DAC14 range and rounded **inward** to multiples of four stream codes. For positive G this is a ceiling at the lower voltage bound and a floor at the upper; for negative G the code ordering reverses. The driver searches the monotone calibrated voltage grid so no outward tolerance admits an out-of-window level, including on endpoint readback/reapply. Bounds beyond the converter range give its endpoints if the intersection is nonempty; otherwise the request is rejected. Output gain 0 keeps the requested mix in the applied configuration, so setting the gain back restores the output. The mute code is the nearest DAC14 level to calibrated 0 V, subsequently constrained by the final clamp.

Loading and attaching follow the common rules for every design (download unless the driver loaded this bitstream in the current boot; no register access while the processor–FPGA bus is down; the design ID and build ID in the common system-ID block (sysid) are checked). See "Safe loading" in `../../common/README.md`.

Every getter returns the **applied** values, recomputed from the quantized registers. The state file stores those values and the register image they correspond to:

- the live copy is on tmpfs (a file system in memory), `/run/rpfpga/mult_add/state.json`, shared by every process on the board;
- a copy on the SD card, `/home/xilinx/rpfpga/mult_add/state.json`, is written in the background (SD writes can stall for seconds) and restores the configuration after a reboot.

If another program changes the registers, the driver notices the mismatch and derives the configuration from the registers instead. After a reload the registers hold their power-up values, and the driver reports the power-up configuration (derived from the registers with the current calibration). That power-up passthrough is code for code and therefore not calibrated: on this board it works out to about OUT1 ≈ 1.006·IN1 − 23 mV and OUT2 ≈ 1.005·IN2 − 6 mV (calculated from the calibration constants).

Accuracy and limits

Contribution	Size (LV range)
ADC/DAC resolution	14 bits: about 138 µV steps over about ±1.13 V on this board (122 µV nominal)
Weight quantization	7.6·10^-6 … 1.5·10^-5 relative to the largest weight of an output (power-of-two allocation; coarser only if its largest coefficient is below about 0.016)
Term rounding	½ code per product (about 20 µV on this board, 15 µV nominal); the divider is exact to ½ LSB

Contribution	Size (LV range)
Uncalibrated gain/offset	large on this board: DAC gains +13.5% and +14.3%, LV ADC gains −11.3% and −12.0%, offsets up to about 15 mV per converter, about 20 mV around the OUT→IN loop
After the board calibration	the OUT→IN loop is accurate to the meter noise; without a new DMM reference, loopback retains stored ADC gains. Starting from nominal gains gave X − Y = +3.7 mV for IN1 = IN2 = 0.5 V on this board (0.08 mV with the DMM reference), see `../../common/CALIBRATION.md`
DAC truncation	discarding the two lowest stream bits introduces 0 to −3 codes (mean ≈ −52 µV for this board's positive DAC gain). The enabled clamp performs this conversion before clipping and emits multiples of four; there is no further truncation error at its bounds.
Bandwidth	with LPF bypassed, set by the analog front ends and converters; enabled LPF adds its response (still one result per 8 ns); see above for lookahead residues and below for aliasing and DAC hold

Products and noise. Products and squares of small signals lose relative resolution. 0.01 V × 0.01 V = 100 µV is less than one DAC step (output). The digital input gain does not make the ADC step (about 138 µV) or the ADC noise smaller; it only reduces the product's rounding (½ code), which matters only for very small signals (X ≈ 10 mV gives $X^2 \approx 2.6$ codes). For really small signals use an analog preamplifier. Noise and offsets also bias squares: $E[(s + n)^2] = s^2 + \sigma^2$.

Division and noise. Division amplifies the denominator's noise by $|X|/Y^2$ (the relative noise of Y passes 1:1 into the ratio). Choose the floor above the noise level of Y.

Discrete samples, no anti-aliasing. The inputs have no anti-aliasing filter beyond the analog front end (about 60 MHz), so input signals and noise above the 62.5 MHz Nyquist frequency alias. Nonlinear terms create harmonics: for example, |X| and sign(X) of a 20 MHz sine have components above 62.5 MHz, which fold back. The output reconstruction filter does not remove them. The DAC holds each sample for one clock (zero-order hold): the output droops as sinc(f/125 MHz), −0.37 dB at 20 MHz and −2.4 dB at 50 MHz, and carries images at 125 MHz − f.

Output loading. Calibration and all voltages refer to the outputs as seen by a high-impedance input. A 50 Ω load changes the voltage according to the board's output impedance; this has not been measured.

Control reference: Python API, command line and TCP protocol

mult_add uses the interfaces every design shares (../../common/README.md): the driver API (config, apply, snapshot, describe, status, safe, presets, load), the core TCP commands, the command line, the service and the client. This page lists what mult_add **3.0** adds. Timings measured with the earlier software are identified below; the new control path has not yet been timed on the board.

All interfaces go through one driver (`control/mult_add/driver.py`, class `MultAdd`) and share one state and one set of presets on the board, plus the board calibration and input ranges that all designs share (../../common/CALIBRATION.md). Values are **real volts at the connectors**; the meaning of each term is in THEORY.md.

Interface	Runs where	Time per call	Setup
`pynq.overlays.mult_add.DRIVER` in Python or Jupyter	on the board, as root	earlier software: < 1 ms for short calls; current path unmeasured	none (Jupyter runs as root)
`MultAddClient` (Python) or any TCP program	PC → board, port 5025	one round trip per configuration helper; earlier network timings about 10–40 ms	the service (`make install-service`)
`python -m rpfpga.cli mult_add` over SSH	on the board, via `sudo`	about 11 s (PYNQ import)	the passwordless sudo rule of the board setup

Configuration

A configuration is a JSON-compatible dict. Every key is optional in a patch.

```
{
  "inputs": {
```

```
    "x": {"source": "in1", "gain": 1.0, "offset": 0.0},   # X = gain·source + offset; source: in1,
     ↪  in2, test
    "y": {"source": "in2", "gain": 1.0, "offset": 0.0}    # source test: X uses T1, Y uses T2
  },
  "test": {"t1": 0.0, "t2": 0.0},     # test levels (V): T1 acts like a calibrated LV signal on IN1,
   ↪  T2 on IN2
  "y_floor": 0.0078125,               # smallest |Y| that X/Y divides by (V)
  "outputs": {
    "1": {
      "mix": {"x": 1.0},              # term -> weight; terms below
      "gain": 1.0,                    # main = gain · Σ weight·term + offset (before processing)
      "offset": 0.0,                  # V
      "vmin": None, "vmax": None,     # finite clamp limits (V), or None for the DAC-range endpoint
      "clamp_enabled": False,         # final clamp, independently enabled
      "clamp_protect": False,         # True forces enable and locks until FPGA reload
      "enabled": True,                # False: nearest-to-zero DAC level, then final clamp
      "fast": "auto",                 # "auto" (fast unless X/Y is used), True (fast), False (full)
      "lpf": {"enabled": False, "tau_s": 0.001},  # seconds; default is approximately 1 ms
      "comparator": {
        "enabled": False,
        "lower_threshold": 0.0, "upper_threshold": 0.0,  # wide virtual output volts
        "low_level": -0.5, "high_level": 0.5,            # physical DAC output volts
        "initial_high": False                           # used only on explicit comparator reset
      }
    },
    "2": {...}                        # as "1", with "mix": {"y": 1.0}
  }
}
```

Terms: `x`, `y`, `abs_x`, `abs_y`, `xy` (X·Y/1V), `x_div_y` ((X/Y)·1V), `x2` (X²/1V), `y2` (Y²/1V), `min`, `max`, `sign_x` (±1 V), `sign_y` (±1 V), `one` (1 V).

When a patch is merged, nested dicts are merged, except that a `mix` replaces the whole mix of that output. Unknown keys (e.g. a misspelled setting) and a non-boolean `enabled` are rejected with `ValueError`. `apply(config, replace=True)` starts from the default configuration instead of the current one, retaining protected clamp fields. Preset loading also retains those locked fields. Defaults are compiled with calibration, unlike raw hardware power-up passthrough.

Final clamp settings

The flat per-output keys are `clamp_enabled`, `clamp_protect`, `vmin`, `vmax`. Enable/protect default to `False`. A sparse bounds edit auto-enables clamping unless `clamp_enabled=False` is explicitly in the patch; protection always forces enable. A bound is a finite number or `None` (the corresponding DAC-range endpoint). Even when disabled, stored bounds must form an achievable interval.

The driver intersects the requested window with the **physical DAC14 grid** and rounds inward: the lowest achievable voltage at/above `vmin`, and the highest at/below `vmax`, including negative DAC gain. One grid step is about 138 µV on this board. No achievable level means `RangeError` before configuration writes.

Protection latches bounds and enable in hardware until **FPGA bitstream reload**. Attempts to disable/unprotect or change the realized bounds are rejected before writes. Ordinary core reset, processing reset, preset loading and `SAFE` retain the protected window. Mute selects the DAC level nearest calibrated 0 V, then the final guard clips it: **a window excluding zero gives a nonzero mute**. `SAFE` resets unprotected settings and mutes, preserving protected fields.

The final guard follows processing, saturation and mute; no core data mux can bypass it. With the FPGA configured and clocks running, new bounds apply to DAC capture on the first edge following commit, even during pipeline refill/valid gaps. This is a digital-stream bound, not an absolute connector-voltage guarantee. Analog settling/ringing, calibration/load error, power and reconfiguration are outside it; vendor DAC reset independently forces converter data to zero.

```
ma.set_clamp(1, enabled=True, protect=True, vmin=0.2, vmax=0.8)
ma.mute(1)                      # near +0.2 V; zero is outside the window
print(ma.status()["clamp"][1])
ma.load_preset("multiplier")    # keeps the protected OUT1 clamp
ma.safe()                       # keeps it too, then mutes both outputs
```

Only an intentional reload releases protection:

```
ma.load()                          # clears protection; uncalibrated passthrough
ma = ma.reconnect()                 # new handle; reconnect alone does not release a clamp
```

Automatic loopback calibration refuses any protected output. Out-of-window reference/raw holds and DAC calibration changes for a protected output (including reset to different constants) are rejected; input-offset measurement may remain available. See CALIBRATION.md.

LPF and Schmitt settings

Order per output: **unclipped main calculation → LPF → Schmitt comparator → final saturation → mute → final clamp**. Both processing stages default to bypass, independently. They keep processing every sample in bypass and while the output is muted.

- `lpf.enabled`: select the low-pass output instead of the aligned main signal.
- `lpf.tau_s`: exponential time constant, about 11.54 ns … 8796 s. Readback returns its quantized value. The default coefficient set is approximately 1 ms.
- `lpf.cutoff_hz`: alternative patch spelling for the digital −3 dB frequency. Supply **either** `tau_s` **or** `cutoff_hz` in an LPF patch, never both. A cutoff-only patch works even though stored configuration uses `tau_s`; readback is canonical `{enabled, tau_s}`. The two are related by the exact digital pole formula, not generally `1/(2π·tau_s)`.
- `comparator.enabled`: select its state-dependent output level.
- `lower_threshold`, `upper_threshold`: voltages of the selected main/filtered signal, with lower ≤ upper. At/above upper selects high, at/below lower selects low, and between them the old state persists. High wins at equal thresholds. These use the wide calibrated **virtual output** scale and may exceed the physical DAC range.
- `low_level`, `high_level`: voltages for the two states, within the calibrated DAC range and rounded to its **14-bit grid** (about 138 µV per step here). Their numerical order may be reversed. Mute and then the final clamp act afterwards.
- `initial_high`: boolean state selected by explicit comparator reset. Changing this setting alone does not change the current state.

Commits, presets, bypass and mute preserve history. Transitional samples after edits are processed without flushing; the LPF and comparator can retain their effects. `reset_processing()` explicitly seeds the LPF from the current main sample and/or resets the comparator bit. Threshold decisions resume on the next sample. A reset reply is a snapshot, not a filter-settling receipt.

The ranges each value can take are in the README, Specifications.

Python API (`MultAdd` on the board, `MultAddClient` on a PC)

The two classes share the high-level methods below. Low-level hooks and extra measurement controls are marked ”driver only”. Methods that change something return the **applied** values, after quantization to the hardware’s steps. Clamp limits read back as realized: rounded inward to DAC14 levels; bounds beyond the DAC range read back at its endpoints if the intersection is nonempty. With output gain 0 the mix reads back as requested, so restoring the gain restores the output.

Method	Description
`config()`	the applied configuration
`apply(patch, replace=False, expected_revision=None)`	apply a (partial) configuration atomically; reject a stale draft if a revision is supplied
`snapshot()`, `apply_snapshot(patch, replace=False, expected_revision=None)`	read a consistent state, or apply and return it in the same transaction; keys `config`, `revision`, `session`, `describe`, `status`, `calibration`, `ranges`
`set_input(name, source=, gain=, offset=)`	`name` = `"x"` or `"y"`
`set_mix(n, replace=False, **weights)`	e.g. `set_mix(1, xy=0.5, one=-0.1)` changes those weights and keeps the others; `replace=True` clears the other terms
`set_output(n, gain=, offset=, vmin=, vmax=, clamp_enabled=, clamp_protect=, enabled=, fast=, lpf=, comparator=)`	output stage of OUT1 or OUT2; processing settings are nested dicts

Method	Description
`set_clamp(n, enabled=, protect=, vmin=, vmax=)`	atomic sparse edit of the final guard; returns the applied output settings (flat `clamp_enabled`/`clamp_protect` keys)
`set_lpf(n, enabled=, tau_s=)`	patch OUTn's LPF; alternative `cutoff_hz=` in place of `tau_s=`; returns applied LPF settings
`set_comparator(n, enabled=, lower_threshold=, upper_threshold=, low_level=, high_level=, initial_high=)`	patch OUTn's comparator; returns applied settings
`reset_processing(*n, lpf=True, comparator=True)`	explicit reset of selected stages; both outputs if none supplied; returns a snapshot
`set_test_levels(t1=, t2=)`, `set_y_floor(volts)`	test levels, X/Y floor
`describe()`	`{1: "OUT1 = ...", 2: "OUT2 = ..."}`, the realized equations
`status()`	per output: `fast`, output gain code, `lpf` coefficient/response details, `comparator` state/grid and actual `clamp` bounds/protection; `scale`, `latency`, design, hardware/driver/rpfpga versions, build ID and date, calibration method, `calibrates`, session/revision; `rpfpga.version_text(ma)` prints the versions on one line
`mute(*n)`, `unmute(*n)`	all outputs if none given
`safe()`	calibrated default configuration with both outputs muted, preserving all protected clamp fields
`presets()`, `load_preset(name)`, `save_preset(name)`, `delete_preset(name)`	built-in and saved presets (README)
`meters(window_s=0.01, fresh=True, timeout_s=5.0)`	per signal (`in1 in2 x y out1 out2`): `mean`, `rms`, `ac_rms`, `min`, `max` in V; plus `samples`, actual `window_s`, `session`, `revision`
`flags(clear=False)`, `clear_flags()`	sticky clip flags and `counts` (see below)
`ranges()`, `set_input_range(ch, "lv"/"hv")`	the board's input jumper settings (common to all designs)
`calibration()`, `measure_input_offsets()`, `hold_reference()`, `release_reference()`, `calibrate(dmm=None)`, `reset_calibration()`, `set_output_response()`, `set_output_delay()`, `set_pins_delay()`	the board calibration (common to all designs); mult_add can measure it (../../common/CALIBRATION.md)
`measure_input_offsets(window_s=0.1)`, `calibrate(dmm=None, levels=9, window_s=0.02)`	extra measurement controls: driver only; client/TCP use defaults
`adc_codes(window_s)`, `hold_dac_codes({1: code, 2: code})`	raw calibration hooks, driver only; protected holds preflight before any output write
`load()`, `reconnect()`	reload invalidates all handles, including the caller; reconnect explicitly to obtain a new handle. Client `load(design)` can switch designs.
`registers()`, `latency()`, `read(offset)`, `write(offset, value)`	raw registers; latency returns 58/77 register stages and 456/608 edge-to-edge ns (driver only; REGISTERS.md)
`build_id()`, `expected_build_id()`	build in the FPGA and in `mult_add.json` (driver-only methods); only the actual `build_id` is in `status()`, not the expected manifest value

Meters. The window is the power of two of samples nearest to `window_s`, from 2^{10} to 2^{27} samples (8.2 µs to 1.07 s): 0.01 → 8.4 ms, 0.02 → 16.8 ms, 0.1 → 134 ms, 1 → 1.07 s. By default a reading waits for a window that starts after the call, so it takes one to two windows. `fresh=False` can use the last completed hardware window when the requested length is already selected; changing the length still waits for a fresh window. Cached samples can therefore predate the call or a recent edit; the returned session/revision identify the acquisition context, not a promise that every cached sample used that revision. `timeout_s` must be positive and bounds the acquisition wait. `in1`/`in2` are the input voltages, `x`/`y` the conditioned inputs, `out1`/`out2` the output voltages that are sent to the converters (after processing, saturation, mute and final clamp). They are digital readback, not independent measurements of the connectors. The LPF response time is independent of the meter window: a fresh window need not contain a settled filtered signal. An edit cancels a waiting reading, and a second simultaneous meter/calibration acquisition returns `Busy`. Use another connection to edit while a meter call waits.

Processing status. `status()["lpf"][1]` (or `[2]`) includes `mantissa`, `exponent`, `feedback_coefficient`, `alpha`, `lookahead_samples`, `fir_q24`, realized `tau_s`/`cutoff_hz`, requested values, `tau_error_s` and `rel-`

`ative_error`. These characterize the quantized digital pole; Q24 FIR rounding leaves small residual response errors (THEORY.md). `status()["comparator"][n]` includes `high` (the running state), `reverse`, `level_step_v` and `threshold_step_v`. Normal Python `status()` uses integer output keys; JSON-normalized `snapshot()` uses string keys throughout.

Clamp status. `status()["clamp"][n]` and TCP `OUTn CLAMP?` report actual `enabled`, `protected`, finite realized `vmin`/`vmax`, signed stream-code `code_min`/`code_max`, signed `dac14_min`/`dac14_max`, and `step_v`. Stream-code bounds are multiples of 4 in −32768…32764; DAC14 bounds are −8192…8191. Voltage bounds remain ordered even with negative DAC gain. Configuration can retain `None` for an unspecified endpoint; status supplies the actual endpoint. The GUI's Outputs tab shows enable/protect and active readback, and disables locked clamp fields until reload/reconnect.

Flags (all sticky until cleared): `x_clip`, `y_clip` (X or Y beyond their range), `div_clip` (|X/Y| reached 8), `div_floor` (|Y| below the floor), `out1_saturated`, `out2_saturated` (the output stage reached its range), `out1_clamped`, `out2_clamped` (a clamp limit acted). `counts` holds the number of affected samples: `out1_limited`, `out2_limited`, `x_clip`, `y_clip`, `div_clip`. The X/Y flags are raised only while an output in full mode uses X/Y.

Errors raise exceptions. Configuration validation and revision rejection occur before hardware writes; runtime/transport failures may occur after a write has taken effect and do not imply rollback:

- `ValueError`: a bad name, type or value.
- `mult_add.RangeError` (a `ValueError`): the hardware cannot represent the request, for example a gain too large or min/max with one input on LV and the other on HV. The message names the setting and, for weights, the nearest achievable value.
- `RuntimeError`: the overlay is not loaded, or a calibration problem (for example a calibration step whose constants cannot realize the current configuration: "The new calibration was not stored …").
- Stable concurrency errors on both driver and client are `Busy`, `Cancelled`, `Stale` and `Conflict` from `rpfpga.errors` (also `BusyError`, etc.). They are subclasses of `RemoteError`/`mult_add.MultAddError`. Other remote errors use that base class and preserve the message.

An acquisition cancelled by an accepted edit does not undo that edit. Every reload, including the same design, invalidates old handles. A client transport failure closes the connection; it does not replay commands. Reconnect explicitly and inspect the current state before continuing. See the shared concurrency reference.

For a staged edit based on current settings:

```
base = ma.snapshot()
ma.apply_snapshot({"outputs": {"1": {
    "lpf": {"enabled": True, "tau_s": 0.01},
    "comparator": {"enabled": True, "lower_threshold": 0.15,
                   "upper_threshold": 0.25, "low_level": 0.0, "high_level": 0.8}
}}}, expected_revision=base["revision"])
```

A changed revision raises `Conflict` before writing. This is also how staged GUI Apply protects a draft; Discard is explicit.

```
# On the board (Jupyter, or sudo python):
import rpfpga                       # must come first
from pynq.overlays.mult_add import DRIVER
ma = DRIVER()                       # attaches if mult_add was loaded in this boot, else loads it

# On a PC (common/python and mult_add/control on PYTHONPATH; the board must run mult_add):
from mult_add import MultAddClient
ma = MultAddClient("192.168.0.149")

ma.load_preset("multiplier")
ma.set_output(1, gain=2, vmin=-0.5, vmax=0.5)
print(ma.describe()[1])             # realized equation and inward-rounded clamp window
print(ma.meters(0.1)["out1"]["mean"])
```

`DRIVER()` attaches to the running design without reprogramming if mult_add was loaded since the last boot (by the service, Jupyter or the command line), so their settings stay in place; otherwise it loads the design first. `rpfpga.client.connect(host)` returns a client of whichever design is running.

Time per call over TCP

Configuration, query and acquisition methods use **one request/reply round trip** on an established client; connection/reconnect discovery is separate. In particular, `set_mix(..., replace=False)` now uses an atomic server-side `MIX<n> PATCH`; it does not read and merge on the PC. Mutation helpers return their own applied readback or snapshot in that reply. `flags(clear=True)` reads and clears in one request. Acquisitions still include their hardware wait.

Measured over TCP on 2026-10-04 (lab network): a `MIX1` command about 36 ms, `DESCRIBE?` about 9 ms, `METERS? 0.001` about 22 ms. Commands that change the configuration take longer than queries. On the board, the measured short driver calls took under 1 ms; this does not include acquisitions, loading or calibration. These timings predate the current transaction/session implementation. Opening a typed client now obtains `SESSION?`; generic `connect()` also discovers the active design. The concurrent control response target of 100 ms has not been measured on the board.

To change several settings at once, send one `apply()` (TCP `CONFIG`) with all of them: one round trip and one atomic change.

Command line

The common command line (`rpfpga.cli`) with mult_add's options, on the board (as root, without the service) or from a PC through the service:

```
ssh xilinx@192.168.0.149 sudo -n /usr/local/share/pynq-venv/bin/python \
    -m rpfpga.cli mult_add --preset multiplier --out1 vmax=0.5 --meters 0.1
PYTHONPATH=common/python:mult_add/control python3 -m rpfpga.cli --host 192.168.0.149 --mix2
 ↪  abs_x=1,one=-0.2
```

Common options (every design):

Option	Meaning
`design`	the design name, as the first argument; required on the board, optional with `--host` (the running design is used)
`--host HOST`, `--port PORT`	control a board through its TCP service (port 5025 by default)
`--data-dir DIR`	board data directory (on the board; default `/home/xilinx/rpfpga/mult_add`)
`--load`	reprogram the FPGA first (power-up configuration)
`--range IN=lv|hv`	record an input jumper setting, e.g. `--range in1=hv`; repeatable
`--reset`	start from calibrated defaults, retaining protected clamp fields
`--preset NAME`	load a built-in or saved preset
`--safe`	finally mute with protected clamps retained; may be nonzero
`--calibration`	also print the board calibration
`--json`	print the configuration and readings as JSON

mult_add options:

Option	Meaning
`--x SPEC`, `--y SPEC`	`source=in1|in2|test,gain=G,offset=V`
`--mix1 SPEC`, `--mix2 SPEC`	replace a mix: `xy=0.5,one=-0.1`
`--out1 SPEC`, `--out2 SPEC`	`gain=,offset=,vmin=,vmax=,clamp_enabled=,clamp_protect=,enabled=,fast=` (values: numbers, `none`, `true`/`false`, `auto`)
`--clamp1 SPEC`, `--clamp2 SPEC`	`enabled=true,protect=true,vmin=0.2,vmax=0.8`; aliases for the flat output clamp fields
`--lpf1 SPEC`, `--lpf2 SPEC`	`enabled=true,tau_s=0.01`, or use `cutoff_hz=` instead of `tau_s=`
`--cmp1 SPEC`, `--cmp2 SPEC`	`enabled=,lower_threshold=,upper_threshold=,low_level=,high_level=,initial_high=`; booleans `true`/`false`, other values in V
`--reset-processing1 STAGE`, `--reset-processing2 STAGE`	explicitly reset `lpf`, `comparator` or `both` after applying settings
`--test t1=V,t2=V`, `--y-floor V`	test levels, X/Y floor
`--mute`, `--unmute`	both outputs
`--meters [SECONDS]`	print meter readings (default window 0.01 s)
`--flags`, `--clear-flags`	print or clear the clip flags

The actions run in this order: `--load`, `--range`, `--reset`, `--preset`, then the mult_add settings (`--x`, `--y`, `--mix*`, `--out*`, `--clamp*`, `--lpf*`, `--cmp*`, `--test`, `--y-floor` as one patch), `--reset-processing1/2`, `--mute`, `--unmute`, `--clear-flags`, and finally `--safe`. Then the meters and flags are read, and the equations are always printed.

Example (filtered passthrough):

```
PYTHONPATH=common/python:mult_add/control python3 -m rpfpga.cli --host 192.168.0.149 \
    --preset passthrough --lpf1 enabled=true,tau_s=0.01 --meters 0.1
```

Positive-only protected mute (deliberately locks until reload):

```
PYTHONPATH=common/python:mult_add/control python3 -m rpfpga.cli --host 192.168.0.149 \
    --clamp1 enabled=true,protect=true,vmin=0.2,vmax=0.8 --mute
```

OUT1 then selects its lower allowed voltage, near +0.2 V. `--load` is a separate deliberate reload action; it clears protection and starts raw passthrough.

TCP protocol

The common service (`rp-overlay.service`, port 5025) serves mult_add when it is the running design (`LOAD mult_add`, or `make install-service` from this directory). One command per line, keywords case-insensitive; each reply is one line, `OK <value>` or `ERR <message>`, with JSON for structured values.

The **core commands** are described in ../../common/README.md: `*IDN?`, `HELP`, `DESIGNS?`, `DESIGN?`, `LOAD`, `CONFIG?`, `CONFIG <json>`, `CONFIG REPLACE <json>`, `SNAPSHOT?`, `SESSION?`, `REQUEST <json>`, `DESCRIBE?`, `STATUS?`, `PRESETS?`, `PRESET`, `SAFE`, and the board's input ranges and calibration, `RANGE?`, `RANGE <IN1|IN2> <LV|HV>`, `CAL?`, `CAL RESET`, `CAL OFFSETS`, `CAL HOLD`, `CAL RELEASE`, `CAL LOOPBACK [<dmm json>]`, `CAL RESPONSE`, `CAL DELAY`, `CAL PINS` (../../common/CALIBRATION.md). mult_add adds:

Command	Reply	Notes
`MIX<n>?`	mix JSON	n = 1, 2
`MIX<n> <term> <w>`	applied mix	changes one weight, keeps the others
`MIX<n> SET <t>=<w>,...`	applied mix	replaces the mix; `MIX<n> SET` alone empties it
`MIX<n> PATCH <json>`	applied mix	atomically changes only supplied weights, e.g. `MIX1 PATCH {"xy":0.5,"one":-0.1}`
`INX?` / `INY?`	input JSON	
`INX SRC in1|in2|test`, `INX GAIN <g>`, `INX OFFS <V>`	input JSON	same for `INY`
`OUT<n>?`	output JSON	
`OUT<n> GAIN <g>` / `OFFS <V>` / `MIN <V|NONE>` / `MAX <V|NONE>` / `EN <ON|OFF>` / `FAST <ON|OFF|AUTO>`	output JSON	
`OUT<n> CLAMP?`	actual clamp status JSON	enable/protection, finite voltage bounds, stream/DAC14 bounds and step
`OUT<n> CLAMP EN <ON|OFF>`, `OUT<n> CLAMP PROTECT <ON|OFF>`	applied output JSON	protect forces enable; OFF cannot release a protected clamp
`OUT<n> CLAMP MIN <V|NONE>`, `OUT<n> CLAMP MAX <V|NONE>`	applied output JSON	sparse bounds edits auto-enable; NONE selects converter-range endpoint
`OUT<n> CLAMP SET <json>`	applied output JSON	atomic settings `{enabled,protect,vmin,vmax}`; change both bounds/protect together
`OUT<n> LPF?`	LPF JSON	`{enabled,tau_s}`
`OUT<n> LPF EN <ON|OFF>`, `OUT<n> LPF TAU <seconds>`, `OUT<n> LPF FREQ <Hz>`	applied LPF JSON	TAU/FREQ are alternative settings
`OUT<n> LPF RESET`	snapshot JSON	reset this filter from current main
`OUT<n> CMP?`	comparator JSON	settings; live high/low state is in `STATUS?`
`OUT<n> CMP EN <ON|OFF>`	applied comparator JSON	independent of LPF enable
`OUT<n> CMP LOWER <V>` / `UPPER <V>` / `LOW <V>` / `HIGH <V>`	applied comparator JSON	lower/upper thresholds; low/high output levels
`OUT<n> CMP INITIAL <LOW|HIGH>`	applied comparator JSON	state to use on next explicit comparator reset
`OUT<n> CMP RESET`	snapshot JSON	reset comparator only

Command	Reply	Notes
`PROCESS RESET <1|2|ALL> <LPF|CMP|BOTH>`	snapshot JSON	explicit output/stage selection
`TEST?`, `TEST <t1> <t2>`	test levels JSON	both levels, in V
`YFLOOR?`, `YFLOOR <V>`	applied floor	
`METERS? [seconds [FRESH|CACHED [timeout_s]]]`	meters JSON	defaults: 0.01 s, FRESH, 5 s timeout; changed window length always waits fresh
`FLAGS?`, `FLAGS? CLEAR`, `FLAGS CLEAR`	flags JSON, pre-clear flags JSON, `OK`	query-and-clear is one transaction
`MUTE [n]`, `UNMUTE [n]`	equations JSON	both outputs if no n

To change a threshold band without invalid intermediate lower/upper ordering, send both in one patch:

```
CONFIG {"outputs":{"1":{"comparator":{"enabled":true,"lower_threshold":0.15,"upper_threshold":0.25
↪  ,"low_level":0,"high_level":0.8}}}}
OUT1 LPF TAU 0.01
OUT1 LPF EN ON
OUT1 CMP?
METERS? 0.1 FRESH 5
```

Each setter's reply is its applied readback from the same transaction. A `REQUEST` envelope adds a session/revision guard; Python clients use it automatically. Stable concurrency errors are `ERR BUSY ...`, `ERR CANCELLED ...`, `ERR STALE ...`, `ERR CONFLICT ...`. An explicit reset is optional and separate from setting or enabling a stage.

Set a whole clamp window atomically rather than pass through intermediate bounds:

```
OUT1 CLAMP SET {"enabled":true,"protect":true,"vmin":0.2,"vmax":0.8}
OUT1 CLAMP?
MUTE 1
SAFE
```

Mute and SAFE retain that protected positive window. Releasing it requires `LOAD mult_add`; Python clients must explicitly reconnect afterwards. `CAL HOLD` persists until `CAL RELEASE` or a superseding edit, and refuses an out-of-window reference before writes. `CAL LOOPBACK` refuses any protected clamp.

An example session with `nc 192.168.0.149 5025`, recorded on the board on 2026-10-04 with the **nominal** calibration and MADD 1.0 (historical replies, before the added processing fields; meter output shortened). The numbers in replies depend on the calibration: with this board's calibration they read, for example, `"xy":1.0000032`, `"offset":-1.7e-05`, `"vmax":0.499989` and `"one":-0.20000029`. Those historical endpoint examples use the pre-3.0 clamp grid; current clamp endpoints use inward DAC14 rounding.

```
PRESET multiplier
OK {"1":"OUT1 = X·Y/1V","2":"OUT2 = Y"}
OUT1 MAX 0.5
OK {"mix":{"xy":1.0},"gain":1.0,"offset":0.0,"vmin":null,"vmax":0.5,"enabled":true,"fast":"auto"}
MIX2 SET abs_x=1,one=-0.2
OK {"abs_x":1.0,"one":-0.1999969482421875}
OUT1 FAST ON
OK {"mix":{"xy":1.0},"gain":1.0,"offset":0.0,"vmin":null,"vmax":0.5,"enabled":true,"fast":true}
MIX1 x_div_y 0.1
ERR outputs.1: X/Y is not available in fast mode
DESCRIBE?
OK {"1":"OUT1 = X·Y/1V (clamped ≤ 0.5 V)","2":"OUT2 = |X| - 0.2 V"}
METERS? 0.1
OK {"window_s":0.134217728,"samples":16777216,"in1":{"mean":-0.01222,"rms":0.13276,"ac_rms":0.1321
↪  9,"min":-0.2026,"max":0.1786},...}
```

Replies contain the **applied** values. For example, −0.2 V realized on the hardware's grid reads back as −0.1999969 (nominal calibration). Validation errors occur before writes; a general `ERR` or transport failure does not establish that no hardware change took place. Replies are UTF-8.

Security

The service has **no authentication**. Keep the board on a trusted network, or install the service bound to 127.0.0.1 (`SERVER_BIND=127.0.0.1`) and tunnel over SSH (../../common/README.md):

```
ssh -N -L 5025:localhost:5025 xilinx@192.168.0.149 &
PYTHONPATH=common/python:mult_add/control python3 -m rpfpga.cli --host localhost
```

Register map

`mult_add_0` is an AXI4-Lite slave at **0x43C00000**, size 4 KiB, 32-bit registers, little endian. The common sysid block at 0x40000000 identifies the design (”MADD”), version **3.0** and build ID; see `../../common/README.md`. Only full 32-bit accesses are needed; byte strobes are honored. Unlisted addresses read 0 and ignore writes. The register file is `src/ma_regs.v`; the arithmetic it controls is described in THEORY.md.

Normally you use the driver (`control/mult_add/driver.py`), which converts volts to codes, applies calibration and commits atomically. This page is for raw access, debugging and other languages.

Commit semantics

Registers in the **Configuration (shadowed)** tables are configuration:

- Writes go to a shadow copy.
- Reads return the shadow copy.
- The datapath uses the **active** copy, which changes only when CTRL bit 0 is written. Updates occur on the same clock edge, except protected clamp fields remain locked. Unprotected clamp bounds/control normalize on commit (below).

CTRL bit 0 reads back as 1 after any shadow write, until the next commit. It does not compare values, so rewriting a value that is already active also sets it. Ignored writes to protected clamp banks do not dirty them. **Bitstream reload** initializes both copies to the power-up values below. Ordinary `resetn` resets other settings, but preserves both active and shadow fields of each protected clamp bank; it is not equivalent to bitstream reload.

Identification and control

Offset	Name	Access	Reset	Description
0x000	ID	R	0x4D414444	”MADD”
0x004	CTRL	R/W	0	W pulse bits: [0] commit, [1] clear all flags/counters, [2] reset LPF1, [3] reset CMP1, [4] reset LPF2, [5] reset CMP2. R: bit 0 = uncommitted shadow writes
0x038	LATENCY	R	0x004D003A	[15:0] fast-path register stages (58), [31:16] full-path register stages (77); edge intervals 57/76 = 456/608 ns
0x03C	VERSION	R	0x00030000	[31:16] major, [15:0] minor

Configuration (shadowed)

Offset	Name	Reset	Format
0x008	MODE	0xF4	[1:0] X source (0 IN1, 1 IN2, 2/3 T1); [3:2] Y source (0 IN1, 1 IN2, 2/3 T2); [4] OUT1 enable; [5] OUT2 enable; [6] OUT1 fast; [7] OUT2 fast
0x00C	TEST	0	[15:0] T1, [31:16] T2: signed codes used in place of an ADC sample. The driver writes round(t·K), with K the calibrated LV codes per volt of IN1 (for T1) or IN2 (for T2), so a test level acts like a calibrated LV signal on that input (nominally K = 32768; about 29,050 and 28,831 on this board)
0x010	IN_X	0x01000000	[31:16] gain gx (signed Q8.8), [15:0] offset ox (signed code)
0x014	IN_Y	0x01000000	as IN_X for Y
0x018	FLOOR	256	[15:0] minimum |Y| for X/Y, 1…32768 codes (0 behaves as 1)
0x040 + 4i	W1[i]	W1[0] = 0x4000	OUT1 weight of term i (i = 0…12; 13–15 reserved): [17:0] signed Q4.14; reads sign-extended
0x080 + 4i	W2[i]	W2[1] = 0x4000	OUT2 weights, as W1
0x0C0	OUT1_CTRL	0x01000000	[31:16] output gain (signed Q8.8), [15:0] offset (signed code)
0x0C4	OUT2_CTRL	0x01000000	as OUT1_CTRL
0x0C8	OUT1_LIMITS	0x7FFC8000	[31:16] upper, [15:0] lower: signed stream-code bounds of the final guard, normalized inward to multiples of 4 on commit

Offset	Name	Reset	Format
0x0CC	OUT2_LIMITS	0x7FFC8000	as OUT1_LIMITS
0x0D0	MUTE	0	[15:0] OUT1, [31:16] OUT2: candidate code while output disabled; still passes through final clamp
0x0D4	OUT1_CLAMP_CTRL	0	bit 0 enable, bit 1 protect; reserved bits read zero; protection forces enable on commit
0x0D8	OUT2_CLAMP_CTRL	0	as OUT1_CLAMP_CTRL

Term indices: 0 X, 1 Y, 2 |X|, 3 |Y|, 4 X·Y, 5 X/Y, 6 X^2, 7 Y^2, 8 min, 9 max, 10 sign X, 11 sign Y, 12 constant 1.

The power-up values give OUT1 = X = IN1 and OUT2 = Y = IN2 in fast mode, with unity gains and no clamping, code for code and therefore without the board calibration. This is the same passthrough as analog_echo at unity gain.

Final clamp commit and protection

`ma_final_clamp` follows rounding/saturation and the mute selection, with no downstream core data mux. When enabled it takes signed `candidate[15:2]` (DAC14 −8192…8191), clips to the active bounds, and emits that code followed by two zero bits (−32768…32764 in multiples of 4). It forces output valid so even a held sample or empty pipeline is corrected at the next DAC capture. Disabled, it passes candidate data/valid unchanged.

At commit, an unprotected bank rounds raw lower up and raw upper down to this grid. An empty/reversed interval, including lower-ceiling overflow to +32768, **retains the last valid active bounds**. The enable/protect request still applies to those bounds and unrelated configuration still commits. Shadow limits/control read back the normalized active values after commit. Software voltage requests instead reject an interval containing no achievable level.

Protect bit 1 forces bit 0 on and locks bounds, enable and protection in **both shadow and active copies** until bitstream reload. All writes to that bank's limits/control are ignored thereafter, for every byte-strobe combination. Before protection, limits honor all four byte strobes; control bits use strobe 0 only. Processing reset pulses, flag clears and ordinary core `resetn` cannot release protection. Other configuration remains writable.

The live guard updates after commit edge C; DAC capture at C+1 uses the new bounds (8 ns later), independently of the 58/77-stage pipeline. This contract requires a configured FPGA and running clocks. The core guard remains active during ordinary reset, but the vendor DAC reset independently forces converter data to zero. Analog settling/ringing, calibration/load error, power and reconfiguration are not bounded by these registers (THEORY.md).

Post-processing configuration (shadowed)

Each output has an 11-word block: **OUT1 base 0x300**, **OUT2 base 0x340**. All are shadowed and commit with the existing configuration. Unlisted words in each 0x40-byte block are reserved. The signal path is wide main → selected LPF/bypass → selected comparator/bypass → rounding/saturation → mute → final clamp.

Block offset	Name	Reset	Format
+0x00	POST_CTRL	0	[0] LPF enable; [1] comparator enable; [2] comparator reset `initial_high`; [3] reverse threshold comparisons (negative DAC volts/code); other bits reserved
+0x04	B_MANTISSA	8795786	[23:0] normalized unsigned feedback mantissa M, 2^23 … 2^24−1
+0x08	B_EXP	37	[5:0] exponent P, 24 … 60; b = M·2^−P = 1−(1−alpha)^8
+0x0C	C1	8388574	[23:0] unsigned Q24 FIR factor, round(2^24·a/(1+a))
+0x10	LOWER_LO	0	bits [31:0] of lower threshold payload
+0x14	LOWER_HI	0	bits [63:32] of lower threshold payload
+0x18	UPPER_LO	0	bits [31:0] of upper threshold payload
+0x1C	UPPER_HI	0	bits [63:32] of upper threshold payload
+0x20	LEVELS	0x4000C000	[15:0] low-state code (−16384), [31:16] high-state code (+16384); signed 16-bit codes, driver uses multiples of 4
+0x24	C2	8388541	[23:0] unsigned Q24 FIR factor, round(2^24·a^2/(1+a^2))
+0x28	C4	8388474	[23:0] unsigned Q24 FIR factor, round(2^24·a^4/(1+a^4))

`a = (1−b)^(1/8)` must be derived from the **quantized b** before computing C1/C2/C4. Require `b <= 1−0.5^8`; use the driver's coefficient conversion to obtain one consistent set. The reset set corresponds to approximately 1 ms. This is an eight-sample lookahead running at the full sample rate, not a decimated filter or a direct-alpha register.

The internal feedback state is signed 89-bit code-Q55; it is not exposed as a register. Quantized FIR factors leave small pole-cancellation residues; see THEORY.md.

Threshold payloads are **signed 49-bit code-Q15, sign-extended to 64 bits** and split low word first. Hardware compares bits [48:0]; the driver validates the range. With `V = G·code+B`, encode a threshold as `round((V−B)/G · 2^15)`, without saturation to 16-bit DAC codes. Threshold names are in physical voltage order; POST_CTRL[3] reverses the code comparison when G is negative. Upper comparison has priority at equality. Level codes are MSB-aligned DAC14 stream values (−32768 … 32764 in steps of 4); their nominal reset voltages are −0.5/+0.5 V, not calibrated board measurements. Mute and final clamp remain downstream of the selected level.

State/reset semantics: both processors run when bypassed and muted. Commit does not clear either state. CTRL[2:5] are one-cycle pulses, honored with byte strobe 0, and do not require a commit; they may accompany one. LPF reset seeds history with the current main sample; comparator reset selects POST_CTRL[2] for its reset sample, then thresholds apply on the next sample. No transition samples are flushed or suppressed after a commit/reset.

Status (not shadowed)

Offset	Name	Access	Description
0x0DC	CLAMP_STATUS	R	active OUT1 enable/protect in bits [1:0], OUT2 in [3:2]; other bits zero
0x0E0	OUT1_ACTIVE_LIMITS	R	actual accepted bounds; upper [31:16], lower [15:0], signed multiples of 4
0x0E4	OUT2_ACTIVE_LIMITS	R	as OUT1_ACTIVE_LIMITS
0x100	FLAGS	R/W1C	Sticky: [0] X input clipped, [1] Y input clipped, [2] |X/Y| reached 8, [3] |Y| < floor, [4] OUT1 saturated, [5] OUT2 saturated, [6] OUT1 clamped, [7] OUT2 clamped. Writing 1 clears a bit. Bits 2–3 are raised only while a full-mode output uses X/Y
0x104	COUNT_OUT1	R	samples on OUT1 that saturated or were clamped
0x108	COUNT_OUT2	R	the same for OUT2
0x10C	COUNT_X	R	samples where the X input stage saturated
0x110	COUNT_Y	R	samples where the Y input stage saturated
0x114	COUNT_DIV	R	samples where X/Y saturated at ±8 (while used)
0x380	PROCESSING_STATUS	R	[0] OUT1 comparator high; [1] OUT2 comparator high; remaining bits zero. Live background state, including while bypassed/muted.

Counters saturate at 0xFFFFFFFF. CTRL bit 1 clears all flags and counters.

Meters (not shadowed)

Offset	Name	Access	Description
0x200	METER_CTRL	R/W	[4:0] k: window = 2^k samples, clamped to 10…27; [8] freeze: hold the snapshot (accumulation continues)
0x204	METER_SEQ	R	increments at each latched window
0x208	METER_COUNT	R	number of samples in the latched window
0x220 + 0x20·s	block s	R	s = 0 IN1, 1 IN2, 2 X, 3 Y, 4 OUT1, 5 OUT2

Each block contains:

Block offset	Content
+0x00	Σx, bits [31:0]
+0x04	Σx, bits [47:32], sign-extended to 32 bits
+0x08	Σx^2, bits [31:0] (unsigned)
+0x0C	Σx^2, bits [63:32]
+0x10	[31:16] max, [15:0] min (signed codes)

IN1/IN2 are raw ADC codes. X/Y are input-stage outputs. OUT1/OUT2 are the codes sent to the DAC (after LPF/comparator, saturation, mute and final clamp). The driver converts all of them to volts with the board cali-

bration.

To read consistently:

1. Write k (and freeze = 0).
2. Wait until METER_SEQ has advanced twice. This guarantees a window that started after the change.
3. Set freeze.
4. Read the blocks.
5. Clear freeze.

The driver performs the start, polls and frozen read in short checked transactions, releasing the board lock between polls. A concurrent mutation cancels the acquisition rather than returning a mixed-configuration reading. `fresh=False` may reuse the last window only when k is already the requested value; changing k still waits for a fresh window. See PROTOCOL.md.

Example: raw access from Python on the board

Use the driver for loading. An access while the FPGA does not hold mult_add freezes the board (see "Safe loading" in the common README). After that, the driver's `read`/`write` give raw access:

```
import rpfpga                                    # first: prepares PYNQ's environment
from pynq.overlays.mult_add import DRIVER
ma = DRIVER()                             # safe load / attach, sysid and build checked
with ma.transaction():                   # keep initialization, raw edit and readback together
    assert ma.read(0x000) == 0x4D414444
    ma.apply({}, replace=True)      # known calibrated default mix/processing; protected clamps
 ↪  retained
    ma.write(0x040, 0)              # OUT1 raw X weight = 0
    ma.write(0x050, 0x4000)          # OUT1 raw X·Y weight = 1.0 (Q4.14)
    ma.write(0x004, 1)              # commit with the initialized input/output scale registers
    print(ma.config())             # derive the actual calibrated equation

# To request OUT1 = X·Y/1V in real volts, use the calibrated API instead:
ma.apply({"outputs": {"1": {"mix": {"xy": 1.0}}}}, replace=True)
```

Raw writes bypass the driver's state file. The driver notices and derives the configuration from the registers the next time it reads them. A unit raw weight does not by itself specify a unit physical coefficient: input scales, output gain/offset and calibration also matter. The transaction serializes software access; it does not roll back raw writes after a runtime failure.

Verification record

What was tested on mult_add, when, with which tools, and what came out. Entries are newest first. Each entry describes the state of the design and its software at that date; later entries can supersede earlier conclusions, and failures and incidents are kept as they happened. Quantities that were never measured are listed in the README. Build resources and timing are at the end.

Tools throughout: Vivado 2025.1, Python 3.12 on the build machine, PYNQ 3.0.1 on the board at 192.168.0.149.

2026-10-05: MADD 3.0 protected final DAC14 clamp

Build 0xcc5f666c, Vivado **2025.1**, target `xc7z010clg400-1`, board definition `redpitaya.com:redpitaya-125-14:part0:1.0`.

The new `ma_final_clamp` is after processing, saturation and mute, with no downstream core data mux. Enabled candidates are converted to signed DAC14 and clipped to inward-rounded bounds. Protection forces enable and retains both active and shadow bounds/control across ordinary `resetn`; only FPGA bitstream reload releases it. New bounds reach DAC capture on the first edge after commit. The vendor DAC reset independently forces converter data to zero: this is a configured, clock-running digital-stream contract, not an absolute physical connector-voltage guarantee.

Full make build passed. Implementation evidence is in `mult_add.runs/impl_1/runme.log`, the routed timing summary and routed DRC report:

Quantity	Result
Setup	WNS **+0.435 ns**, TNS **0**
Hold	WHS **+0.012 ns**, THS **0**
LUTs	**11,315**
Flip-flops	**14,710**
DSP48E1	**65/80**
Block RAM	**0**
Routed DRC	**0 errors; 4 DPOP-2 + 1 RTSTAT-10**

DAC outputs still have no output-delay constraints. These build figures do not measure analog delay, settling or overshoot.

`make sim`: all seven benches passed, including common sysid:

Bench	Receipt / comparisons
`tb_sysid`	common sysid passed
`tb_divider`	divider passed
`tb_core`	**114,290** checked output cycles
`tb_system`	ADC/mult_add/DAC system passed, including core retention versus vendor DAC reset
`tb_post_lpf`	post LPF passed
`tb_post`	post processor passed
`tb_clamp`	**10,827 guarded cycles, 52 commits**; reset retention and raw AXI writes

The clamp bench covers both outputs, positive/negative windows, grid endpoints, empty/reversed raw intervals (retain last valid bounds while control applies), normalized readback, partial strobes and ignored protected writes, mute, processing/reset changes and valid gaps. Latency remains **58/77 register stages**, **57/76 sampling-edge intervals = 456/608 ns** (fast/full), excluding LPF response and converter/analog delays.

`verify_build.py --design mult_add` passed all eight stages: recipe-only build, unchanged rebuild, missing-export recovery, vendor HDL edit, design RTL edit, failed-build upload blocking, cleanup and saved-project regeneration. Recipe creation and regeneration both produced **0xcc5f666c**. Evidence: `mult_add/.build/verification-57mxbwsz/verification_logs/`, files `01-recipe-build.log` through `08-saved-project-regeneration.log`. The errors in stage 06 are the deliberately injected failed-build case.

Python: all 273 tests passed, none skipped, using Python 3.12 and ipywidgets 7.8.5 through the repository's `make test`: 134 common, 18 analog_echo and 121 mult_add tests. Coverage includes inward DAC14 rounding for either calibration polarity, immutable raw-write protection, ordinary-reset retention, preset/mute behavior, actual versus uncommitted clamp readback, legacy configuration migration, GUI locking and staged edits, and TCP/CLI controls. An accepted protection edit during calibration cancels the losing calibration without restoring over the protected window; a rejected protected edit leaves an active meter acquisition running. `git diff --check` passed.

No deployment or physical measurement was performed for MADD 3.0. No new `make test-remote`, `make test-hw`, DMM or scope result is claimed. Protected clamp behavior at the connectors, LPF/Schmitt analog response and the 100 ms concurrent-control target remain unmeasured.

Documentation corrections (2026-10-05): earlier clamp wording overstated a physical connector guarantee; only the digital contract above applies. Earlier loopback coverage was all terms on OUT1 with selected OUT2 checks, not all terms on both outputs, and internal test levels are calibrated but quantized. Without a new DMM reference, loopback retains stored ADC gains; they are nominal only when not previously calibrated/reset. These clarifications do not add or change historical measurements. MADD 2.0's path, counts and timing remain recorded below.

2026-10-05: MADD 2.0 output processing and shared concurrent control

Build 0xf51fd230, Vivado **2025.1**, target `xc7z010clg400-1`, board definition `redpitaya.com:redpitaya-125-14:part0:1.0`.

Implemented signal path: unclipped signed 49-bit code-Q15 main calculation → full-rate low-pass filter → Schmitt comparator → final saturation/clamp/mute. Both stages default to independent bypass and run in the background, including while muted. Commits and presets preserve state and process transitional samples without flushing. Explicit resets seed the LPF from current main and the comparator from `initial_high`, with threshold decisions on the next sample.

Full build passed. Implementation run `mult_add.runs/impl_1/runme.log` completed bitstream generation and stamped build ID 0xf51fd230. Routed reports:

Quantity	Result
Setup	WNS **+0.378 ns**, TNS **0**
Hold	WHS **+0.013 ns**, THS **0**
LUTs	**11,356 (64.52%)**
Flip-flops	**14,830 (42.13%)**
DSP48E1	**65/80 (81.25%)**
Block RAM	**0**
Routed DRC	**4 DPOP-2 + 1 RTSTAT-10**, no errors

The existing external-interface limitation remains: DAC outputs have no output-delay constraints. Timing closure is not an analog delay measurement.

XSim: all six benches passed, with their `PASS:` receipts:

Bench	Receipt / comparisons
`tb_sysid`	common sysid passed
`tb_divider`	250,020 results
`tb_core`	113,470 checked output cycles
`tb_system`	ADC/mult_add/DAC system passed
`tb_post_lpf`	8,715 model comparisons
`tb_post`	2,299 output comparisons

Final core latency is **58/77 register stages** (fast/full), or **57/76 sampling-edge intervals = 456/608 ns**. These figures exclude converter and analog delays and LPF response time. This convention supersedes the latency interpretation in older entries; those historical records are retained below.

Digital response verification: the LPF uses eight-sample lookahead, normalized 24-bit feedback mantissa/exponent, three Q24 FIR factors computed from the quantized feedback coefficient, and signed 89-bit code-Q55 state. Across **124 tested coefficient settings**, including alpha = 2^−40, numerically evaluated response residue was **≤ 0.605 ppm**. This is numerical digital verification, not an analog measurement or an exhaustive bound over every coefficient setting.

Build workflow verification: `verify_build.py --design mult_add` passed all eight stages: recipe-only build, unchanged rebuild, missing-export recovery, vendor HDL edit, design RTL edit, failed-build upload blocking, cleanup and saved-project regeneration. Regeneration reproduced **0xf51fd230**. Evidence is under `mult_add/.build/verification-e2vaj249/verification_logs/` (repository-relative), including `01-recipe-build.log` through `08-saved-project-regeneration.log`; the copied overlay manifest also records MADD 2.0 / 0xf51fd230.

The software adds LPF/comparator config, Python/TCP/CLI/GUI controls, atomic mix patches and replies, session/revision snapshots, guarded staged edits, interruptible metering/calibration and explicit reconnect after every reload.

Python: all 235 tests passed, none skipped, using Python 3.12 and ipywidgets 7.8.5, through the repository's `make test`: 134 common, 18 analog_echo and 83 mult_add tests. This includes model-based processing tests, real localhost TCP requests, process/thread coordination, GUI workers and drafts, cancellation during calibration, and explicit reconnect after loading. The expanded loopback script's 59 checks passed on the calibrated model; its eight new checks cover LPF DC gain, wide-domain Schmitt thresholds/history, reset and independent inverted output levels. They do not establish the physical frequency response. `git diff --check` passed.

Not yet performed: physical-board deployment/test of this build, analog LPF/Schmitt measurements, recipes 19–21 on hardware, and measurement of the 100 ms concurrent-control responsiveness target. The earlier 1.0 board results below do not verify the new processing path.

2026-10-04: consistency and correctness review

Four independent reviews (Python code, hardware and build scripts, documentation against code, physics of the docs) found, among smaller items:

- **Delay one clock longer than documented:** 25 and 44 register stages are 200 and 352 ns from the ADC stream to the DAC stream (documented as 192/344 ns, a half-clock artifact of the testbench). `latency()` and the docs are corrected.
- **Output stage alignment (RTL):** the enable switched one sample and the clamp limits four samples ahead of the gain and offset after a commit. All now switch on the same output sample. `make sim` passed all four testbenches (including `tb_sysid`).
- **min/max and X/Y pairing:** failed at ordinary gains such as 0.3 or 1.1 with this board's calibration. The search now uses the full range of usable codes; it matches to about 1e-5 for gains within a factor of about 10 of each other.
- **Software bugs fixed:** output gain 0 deleted the mix; a calibration that could not realize the configuration was stored anyway; the TCP client was not thread-safe (the GUI meter thread); the calibration summary showed the ADC offset with the wrong sign; unknown configuration keys were accepted; a fresh load on a calibrated board reported nominal values; a failed `LOAD` kept describing the old design. Each has a regression test.
- **Build:** a design ID or version changed in `design.tcl`, or a new RTL file, now reaches an existing project (`rp::sync_project`).

Results (build 0x05192512):

- `make sim`: all testbenches passed. Build: timing met, WNS +0.174 ns, WHS +0.018 ns (the delayed limits cost some slack); 33.1% of LUTs (5,826), 21.8% of registers (7,688), 41 DSP48E1; DRC as before (two DPOP-2, one RTSTAT-10).
- `make test-remote` (both designs), `make install-service` and `make test-hw`: all 51 loopback checks passed, worst error 1.5 mV.
- `make test`: 69 framework, 18 analog_echo and 30 mult_add tests passed (GUI tests too, with ipywidgets); on the board the analog_echo, mult_add and framework suites (18, 30, 36 tests) passed.

2026-10-04: calibrated test levels and X/Y clip (board)

- After the test levels became calibrated (Tn on INn's LV scale), `make test-hw` passed all 51 checks, but the X/Y clip check read 0.794 V instead of 0.800 V (worst error 6.0 mV): the clip is a code ratio, and T1/T2 now carry IN1's and IN2's slightly different ADC gains (0.75%).
- Fixed by matching the X and Y scales whenever X/Y is used, as for min/max. A new model test checks the clip at ±8 with this board's calibration.
- `make test-hw` then passed all 51 checks at calibrated tolerances; worst error 1.6 mV over the 49 voltage checks.

2026-10-04: recipes checked on the model (offline)

- Every recipe in RECIPES.md (18 recipes, 83 checks) was applied through the driver to the bit-exact board model (`tests/fake_board.py`), using this board's measured converter errors and the matching calibration (DAC +13.5%/+14.3%, LV ADC −11.3%/−12.0%, ADC offsets −444.6 and −211.8 codes). Each stated output voltage matched within 0.5 mV, except recipe 8 (the quadratic fit itself is accurate to 3 mV) and the beam-off reading of recipe 4 (0.7 mV, from the ADC step divided by the floor).
- The lock-in and beat formulas of recipes 1 and 2 were checked numerically (numpy), not on the model.
- This is a check of the settings and arithmetic only. Recipes 4–8 have not been tried on hardware.

2026-10-04: calibration moved into the common framework

- The calibration and input ranges belong to the board now (`/home/xilinx/rpfpga/calibration.json`, `board.json`). The procedure is in `rpfpga.Overlay`; mult_add provides the `adc_codes` (meters) and `hold_dac_codes` hooks. `make install-service` moved the existing calibration.
- **Tests:** 60 framework tests (now including the fit, the board files, the core `RANGE`/`CAL` commands and the migration), 17 analog_echo and 27 mult_add tests passed offline. On the board, the design tests and the framework overlay, server and calibration tests passed.
- **`make test-hw`** passed on the migrated calibration.
- **`CAL LOOPBACK` through the common code,** with the same DMM readings, gave constants within 0.1% (gain) and 0.5 mV (offset) of the first run, 13 minutes earlier. `make test-hw` then passed all 51 checks, worst error 1.5 mV.
- **Independent check through analog_echo:** with the same calibration, a DMM read +0.50046 V and −0.30030 V for requested +0.5 V and −0.3 V (see the analog_echo record).

2026-10-04: moved onto the common framework

- **`make sim`:** `tb_sysid` (common), `tb_divider`, `tb_core` and `tb_system` passed.
- **`make test` (repository root):** 53 framework tests, 15 analog_echo tests and 27 mult_add tests passed offline (the GUI tests are skipped without ipywidgets). On the board, the overlay and server tests (11 + 8) and both designs' tests passed, including the GUIs.
  - Restoring the old `is_loaded()` logic makes 4 of the 8 loading-safety tests fail.
- **Build:** see Build resources and timing below; `check_stamp` found build ID 0xc821a10d in the `.bit`.
- **`python3 ../common/tests/verify_build.py --design mult_add`:** every stage passed.
  - Recipe-only creation took 388 s; an unchanged rebuild took 22 s.
  - Also covered: missing-export recovery, a vendor HDL edit, a design RTL edit, a failed build blocking upload, cleanup, and regeneration from the saved project.
  - The build ID of the regenerated project equals that of the first build (0xc821a10d).
  - A failed run used to leave the other launched runs behind, holding the workflow lock; `build.tcl` now waits for them.
- **`make test-remote`:** the board was programmed through the driver; sysid (MADD 1.0), the build ID and all power-up registers were verified, and the `passthrough` preset was applied.
- **`make install-service`:** `rp-overlay.service` is enabled and running, and the old `mult-add`/`analog-echo` units were removed. The data in `/home/xilinx/mult_add/` moved to `/home/xilinx/rpfpga/mult_add/`.
  - Over TCP, `LOAD analog_echo` and `LOAD mult_add` switch designs (about 1.2–1.8 s once the files are cached, 5.8 s on the first mult_add load). `connect()` returns the client class matching the loaded design.
  - Round trips: `MIX1` about 36 ms, `DESCRIBE?` 9 ms, `METERS? 0.001` 22 ms. The live state lives on tmpfs and is persisted in the background, because SD-card writes stalled for 0.25–2.4 s.
  - The common CLI worked over TCP.
- **Jupyter:** `control/mult_add.ipynb` executed headlessly (nbconvert) on the board.
- **Reboot with the service enabled:**
  - First attempt: the board stayed responsive and the FPGA was never touched unconfigured, but the service refused its own download and did not restore the design. Two PYNQ 3.0.1 behaviors removed PYNQ's load record: the stale-cache path of `download()`, and `clear_pl_statefile.service` running after Jupyter. Both are fixed: the driver discards PYNQ's cache flag before downloading, and the service is ordered after `clear_pl_statefile.service` (`../../common/README.md`, "Safe loading"). The test fake now models both behaviors.
  - Second attempt: mult_add was loaded with a valid load record and the `quadratic` configuration was restored. The service answered about 1¾ minutes after the reboot command.
- **Loopback (OUT1 → IN1, OUT2 → IN2, LV):**
  - `make test-hw` without calibration: 51 checks passed at the uncalibrated tolerances. The OUT1 loop was off by about −20 mV.
  - `make test-hw HW_CALIBRATE=1`: the loop gain errors were +0.63% (OUT1) and +0.52% (OUT2), and the offsets −20.4 mV and −5.2 mV. The largest fit residuals were 0.46 mV and 0.61 mV (nominal ADC gain, no DMM, no input offset step).
  - Afterwards, all 51 checks passed at the calibrated tolerances. The worst error over the 49 voltage checks was 1.7 mV. The user configuration was restored after each run.
- **Full calibration (all three steps, LV):**
  - Step 1, inputs shorted: ADC offsets of −444.6 codes (IN1) and −211.8 codes (IN2).
  - Step 2: two independently calibrated 6.5-digit DMMs read 0.5601 V and 0.57299 V at the reference code 16384 (nominal 0.5 V), and agreed to within 0.1 mV.
  - Step 3, loopback with these readings:
    - the DAC gains are +13.5% (OUT1) and +14.3% (OUT2), with offsets −7.5 mV and +1.7 mV;
    - the LV ADC gains are 29,050 and 28,831 codes/V (−11.3% and −12.0%), with input offsets −15.3 mV and −7.3 mV;
    - the fit residuals are at most 0.62 mV.
  - The gain errors are far beyond the 1–3% the docs used to assume. They cancel in loopback, which is why the earlier nominal-gain calibration looked accurate.
  - Afterwards, `make test-hw` passed all 51 checks at the calibrated tolerances. The worst error over the 49 voltage checks was 0.84 mV.
- **Not yet done:**
  - the HV range: its gain is still nominal (offsets: step 1 with the jumpers on HV);
  - an independent DMM check of an output at a level other than the reference: done through analog_echo (see the entry above).

2026-10-04: version 1.0, stand-alone

- **`make sim`:** all three testbenches passed.
  - `tb_divider`: 250,020 quotients against an exact reference.
  - `tb_core`: 104,459 output samples checked one by one, plus registers, flags, counters and meters.
  - `tb_system`: actual ADC/DAC VHDL.
  - A mutation batch in `tb_core` caught every non-equivalent RTL fault: divider rounding, mixer rounding, clamp order, commit of the mute register, sign(0), meter freeze, and divider-flag gating.
- **`make test`:** 47 offline tests passed (GUI skipped). On the board, all 28 control tests, including the GUI, passed with ipywidgets 7.7.1.
  - The model-board tests show every preset matching its formula within 5 ADC steps.
  - Calibration against hidden errors (±5% gain, ±12 mV offset) reduces the worst output error from 52 mV to under 0.1 mV.
- **Build:** see Build resources and timing.
- **`python3 tests/verify_build.py`:** every stage passed.
  - Recipe-only creation took 363 s; an unchanged rebuild took 18 s.
  - Also covered: missing-export recovery, a vendor HDL edit, a mixer RTL edit, a failed build blocking upload, and cleanup.
- **`make test-remote`:**
  - Programmed the board, then verified ID, version, latency and all power-up registers.
  - The `passthrough` preset was applied and read back.
  - The uploaded bitstream hash matched the local export.
- **`make install-service`:**
  - `mult-add.service` enabled and running, and `analog-echo.service` disabled.
  - The PC client, the CLI and raw `nc` sessions all worked against it.
  - A command takes about 10 ms per round trip; a 1 ms meter reading takes 16 ms.
- **Jupyter:** the GUI notebook executed headlessly (nbconvert) on the board.
- **Incident, fixed afterwards:**
  - The service froze the board at every boot. PYNQ's "loaded" record survives reboots and is cleared only after Jupyter starts, so the driver skipped the download and read registers with an empty FPGA and the level shifters disabled.
  - The driver now requires its own per-boot download record and checks the FPGA manager state and the level shifters before any access.
  - The board readings before and after a load (state "unknown" → "operating", LVL_SHFTR_EN 0x0 → 0xF) confirmed the mechanism.
  - Offline tests reproduce the hang with the old logic.
  - The very first freeze, during Jupyter GUI use, is not explained by the logs.
- **Build ID (added afterwards):**
  - The bitstream carries a source hash in USR_ACCESS, read at BUILD_ID (0x030) and compared with the uploaded `mult_add.json`.
  - On the board, BUILD_ID read 0x868404bc, matching the `.json`.
  - A second client attached without reprogramming; the configuration was kept.
  - The recipe-only verification passed, and its regenerated build produced the same build ID.
- **Not yet verified on hardware:**
  - `make test-hw` and loopback calibration ran without loopback cables connected; the inputs did not follow the outputs. Every analog check therefore failed. The digital output meters were exact (±0.5000 V as commanded), and flag checks passed.
  - Physical waveforms have not been observed with a scope.

  To finish, connect OUT1→IN1 and OUT2→IN2 and run `make test-hw HW_CALIBRATE=1`. (Done in the entry "moved onto the common framework" above.)

Build resources and timing

The current MADD 3.0 figures are in the protected-clamp entry. MADD 2.0 figures remain in its 2026-10-05 entry. The following figures are retained for the earlier build.

From the 2026-10-04 build of `xc7z010clg400-1` (build 0x05192512, after the consistency review, including the sysid block):

- 33.1% of LUTs (5,826), 21.8% of registers (7,688), 41 of 80 DSP48E1, no block RAM.
- All timing constraints met: WNS +0.174 ns, WHS +0.018 ns.
- Routed DRC shows two advisory DPOP-2 warnings (the output-stage multiplier cascade could use one more pipeline register) and one RTSTAT-10 ”no routable loads” warning, the same as in analog_echo.

The same external-interface caveat as analog_echo applies: the DAC outputs have no output-delay constraints.