SiccaLab
Test & Measurement, Simple

Automated test plans on the instruments you already use — no coding required.

Sequencer Guide

Version 1.0

www.siccalab.com

Contents

1. What is the Sequencer?

The Sequencer turns SiccaLab from a set of manually operated instrument panels into a test automation tool. You build an ordered test plan of steps — set a pin, read a voltage, send SPI bytes, wait, loop, prompt the operator, check a limit — run it against one or more connected boards, and get an overall PASS or FAIL verdict with an exportable report. No code is required.

This guide covers the Sequencer only. Installation, firmware flashing, board connection, and the eight instrument panels are covered in the SiccaLab User Guide.

1.1 Key concepts

- **A top-level tab.** The Sequencer sits alongside the Boards Panel and is not tied to any single board. One test plan can command several boards; steps reference boards by their display label.
- **Your panel configuration is the single source of truth.** The Sequencer never changes board configuration. You configure pins, buses, and channels in the panels exactly as for manual use, and the plan runs against that configuration. If a step needs something the current configuration does not provide, the run is blocked before it starts — with a precise message telling you what to change and in which panel.
- **Panels are locked during a run.** While a plan executes, no manual interaction with the board panels is possible until the run completes, is stopped, or errors.
- **One session file.** The test plan is saved in the same JSON file as your panel configuration — plan and configuration always travel together (section 11).
- **Fully included in the trial.** The Sequencer is available without restriction during the 14-day free trial.

2. The Sequencer Tab

[SCREENSHOT PLACEHOLDER — Sequencer tab: Variables · Steps grid · Step Editor, with a plan loaded]

The tab is divided into three zones, plus a run console and a board overview. The three zones are separated by draggable splitters, and the grid columns are individually resizable.

Zone	Location	Content
Variables	Left	Define test plan variables and watch their current values during a run
Steps grid	Center	The ordered test plan — one compact row per step
Step Editor	Right	All parameters of the selected step, with validation
Run Console	Bottom	Live, timestamped log of the run — collapsible, resizable by dragging its top edge
Board Overview	Right edge	Slide-in, read-only summary of each connected board's configured state — opened with the Boards toolbar button

2.1 Toolbar

- **New Plan** — clears the plan (variables and steps) in the editor. The file on disk is untouched until the next File → Save.
- **Run ▶ / Step / Stop ■** — start a full-speed run, start paused before step 1 for step-through debugging (section 9), or stop the run. Stop is always available here, on either tab, while a plan is running.
- **Stop on first failure** — when checked, the run halts at the first failed check and the remaining steps are marked SKIP. The setting is stored in the test plan.
- **Report…** — opens the Report Settings dialog (section 10.3).
- **Export Report** — appears after a run completes; exports the results as PDF or CSV.
- **Boards** — toggles the Board Overview panel: per board it shows the label, type, COM port, firmware version, and the configured state each step type depends on (active DIO pins and directions, Analog VREF, active PWM pins, SPI mode and slave CS pins, I2C mode and addresses, open UART channels).

2.2 Run status and verdict banner

A status area at the top of the main window stays visible whichever tab is active, so you can switch freely between the Sequencer and the Boards Panel during a run.

- **While a plan runs** it shows a RUNNING indicator, the session file name, the step progress (for example Step 4 / 12), and the elapsed time.
- **When the run ends** it shows the verdict — a large ✓ PASS (green) or ✗ FAIL (red) — with the number of checks passed, the run duration, and a Dismiss button.

It is hidden until the first run of the session.

3. Building a Test Plan

3.1 Adding steps

Authoring is board-first: pick the target board, then the action. Invalid combinations are prevented while you author, not discovered at run time.

- The step picker groups all 28 step types into 12 groups: Timing, Digital I/O, Analog, PWM, Square Wave, SPI, I2C, UART, DAQ, Flow, Logic, and Variables (section 6).
- Board is always a dropdown of connected boards by display label — never free text. Boards that do not support the selected action are grayed out, and actions the selected board does not support are grayed out too (for example, UART steps on an Uno or Nano).
- Pin and channel fields are dropdowns built from the board's definition. Only entries currently enabled in the corresponding panel are selectable; everything else is grayed out with an explanatory tooltip. Your pin labels from the panels are shown next to the pin names.
- Board-independent steps (the Timing, Flow, Logic, and Variables groups) have no Board field at all.
- SPI and I2C steps have no bus pin fields — the bus is configured at panel level, and sequences use it as configured.

3.2 Editing steps

Steps are edited in a hybrid model: the grid shows one compact row per step, and selecting a row opens all of its parameters in the Step Editor. There are no dialogs to open and close.

- **Name** — free text shown in the grid, the console, and reports; defaults to an auto-generated summary such as D13 = HIGH.
- **Enabled** — uncheck to skip a step without deleting it; skipped steps show SKIP in the Result column.
- **Reorder** steps with the ▲ / ▼ buttons; **copy, cut, and paste** with Ctrl+C / X / V or the right-click menu (within the current session).
- Steps inside a loop are **indented** in the grid, one level per nesting depth, so loop boundaries are immediately visible.
- **Incomplete steps are flagged as you author.** An empty mandatory field shows a red border and a hint in the Step Editor, and the grid row shows a ⚠ indicator whose tooltip lists what is missing. Expressions are syntax-checked at authoring time — a formula that does not parse, or references an unknown variable, marks the step red and blocks Run.

3.3 Values: literals or variables

Every value input is dual-mode: type a literal, or pick a variable from a dropdown filtered to the parameter's type — an int field only offers int variables, a HIGH/LOW field only offers bool variables. Literals are validated as you type: digits only for int fields, one decimal separator for float fields, and the hex character set for byte fields.

Pins and channels are the exception: they are always chosen from the panel-aware dropdowns, never from variables.

4. Variables

Variables are defined at test plan level and scoped to a single run: at the start of every run, each variable is reset to its initial value. Values never persist across runs.

The Variables panel is a read-only overview (Name, Type, Initial, Current); adding and editing happen in a compact dialog with type-correct input fields. The Current column updates live while a plan runs.

Type	Typical use
bool	Pin states, pass/fail flags
int	Counts, durations, byte values
float	Analog voltages, calculated values
string	UART/SPI/I2C byte sequences, labels

- Names are case-insensitive — voltage and VOLTAGE are the same variable — and follow identifier rules: letters, digits, and underscore; no spaces or special characters.
- Renaming a variable updates every step that references it directly (Store-in fields and picked values). Expressions are not rewritten automatically.
- Variables cannot be added or deleted while a plan is running.

4.1 Built-in variables

A read-only Built-in section of the Variables panel lists the variables the engine provides on every run, so they are easy to discover and reference:

Variable	Type	Description
{name}_index	int	Iteration counter of the loop named {name}, starting at 1 (for example outer_index). One per named loop.
step_number	int	Number of the currently executing step
run_timestamp	string	ISO timestamp of the run start
last_read_value	varies	Result of the most recent Read / Receive / DAQ Analyze step

5. Expressions & Functions

Expressions are used in Check steps, While-loop conditions, Formula limits, Math Operation and String Operation steps, and — via {variable} placeholders — in Log and Operator Prompt messages. The standard operators are available: + − * / % ^ ( ) and the comparisons == != > < >= <=.

Every expression field has an **fx** button that opens a searchable list of all functions below, grouped by category, each with its signature, a one-line description, and an example. Clicking a function inserts it at the cursor.

5.1 Numeric functions

Function	Returns	Description
abs(x)	same type	Absolute value
round(x)	int	Round to nearest integer
floor(x)	int	Round down
ceil(x)	int	Round up
min(a, b)	same type	Minimum of two values
max(a, b)	same type	Maximum of two values
clamp(val, min, max)	float	Clamp value between min and max
to_float(x)	float	Convert int or string to float
to_int(x)	int	Convert float or string to int (truncates)

5.2 Boolean functions

Function / operator	Returns	Description
not(x)	bool	Logical NOT
and(a, b)	bool	Logical AND
or(a, b)	bool	Logical OR
== / !=	bool	Equality / inequality comparison
> < >= <=	bool	Numeric comparisons
to_bool(x)	bool	Convert int or string (0 / false / "false" = false)

5.3 String functions

Function	Returns	Description
length(s)	int	Number of characters
contains(s, sub)	bool	True if s contains sub
substring(s, start, len)	string	Extract a portion of a string
to_upper(s) / to_lower(s)	string	Convert case

Function	Returns	Description
trim(s)	string	Remove leading/trailing whitespace
concat(a, b)	string	Concatenate two strings
starts_with(s, prefix)	bool	True if s starts with prefix
ends_with(s, suffix)	bool	True if s ends with suffix
replace(s, old, new)	string	Replace all occurrences of old with new
byte_at(s, index)	int	Parse a hex byte string, return the byte at index
byte_count(s)	int	Count bytes in a hex byte string ("0x01,0x02" = 2)
to_hex(val)	string	Convert an integer to a hex string (255 → "0xFF")
to_int(s)	int	Parse a string as an integer or hex value

byte_at() and byte_count() are designed for SPI/I2C/UART response strings in the format "0x01,0x02,0x03".

5.4 DAQ analysis functions

DAQ functions operate on the buffer captured by the most recent DAQ Acquire step. The channel argument is the acquisition slot (1 or 2), not the analog channel number.

Function	Returns	Description
daq_max(channel)	float	Maximum voltage in the buffer
daq_min(channel)	float	Minimum voltage in the buffer
daq_mean(channel)	float	Average voltage
daq_rms(channel)	float	RMS value
daq_peak_to_peak(channel)	float	Max − Min
daq_frequency(channel)	float	Dominant frequency via zero-crossing detection
daq_sample_count(channel)	int	Number of samples captured

daq_frequency uses zero-crossing detection — results on non-periodic or noisy signals are undefined.

5.5 Examples

- voltage > 2.5 — check an analog reading against a threshold
- clamp(sensor * 3.3 / 1023, 0, 3.3) — scale a raw 10-bit reading to a voltage
- contains(last_read_value, "0x4F") — check whether a UART response contains a byte
- byte_at(last_read_value, 0) == 0x9F — check the first byte of an SPI response
- outer_index <= 10 — While-loop condition using a loop counter
- concat("Cycle ", to_hex(status_byte)) — build a dynamic log message
- daq_peak_to_peak(1) < 0.1 — check that a captured signal is stable (low ripple)

6. Step Reference

The Sequencer provides 28 step types in 12 groups. Every step has a Name, an Enabled checkbox, and a Result; hardware steps also have a Board. Value-producing steps additionally have a Limit (section 7).

Parameters marked "supports variable" accept a variable or an expression in place of a literal value.

6.1 Timing

Wait. Pauses execution for a fixed duration. Board-independent.

Parameter	Type	Description
Duration	int (ms)	Wait time in milliseconds. Supports variable.

6.2 Digital I/O

Set Digital. Writes HIGH or LOW to a digital pin. The pin must be configured as an output in the Digital I/O panel.

Parameter	Type	Description
Pin	int	Digital pin (panel-aware dropdown)
Value	bool	HIGH (true) or LOW (false)

Read Digital. Reads the current state of a digital pin. The result is stored in last_read_value and, optionally, in a named variable.

Parameter	Type	Description
Pin	int	Digital pin (panel-aware dropdown)
Store in	variable (bool)	Optional: variable to store the result

6.3 Analog

Read Analog. Reads the voltage of an analog channel, converted using the VREF configured in the Analog panel. The result is stored in last_read_value and, optionally, in a named variable.

Parameter	Type	Description
Channel	int	Analog channel (panel-aware dropdown)
Store in	variable (float)	Optional: variable to store the voltage

6.4 PWM

Set PWM. Sets the PWM duty cycle on a pin.

Parameter	Type	Description
Pin	int	PWM-capable pin (panel-aware dropdown)

Parameter	Type	Description
Duty	int	Duty cycle 0–100 %. Supports variable.

6.5 Square Wave

Set Square Wave. Starts or stops a square wave on a pin — the step equivalent of the Square Wave panel. Non-blocking: the wave keeps running while subsequent steps execute; a second Set Square Wave step with State = Stop ends it.

Parameter	Type	Description
State	Start / Stop	Start or stop the output
Pin	int	Square-wave-capable pin (dropdown)
Frequency	int (Hz)	Output frequency (Start only), validated against the board's capability. Supports variable.

6.6 SPI

The bus itself — speed, mode, bit order, Master/Slave — is configured in the SPI panel. Pre-run validation blocks the run if the target board is not in the mode a step requires.

Send (Master). Sends bytes to a slave device and captures the received bytes. Requires SPI Master mode.

Parameter	Type	Description
CS Pin	int	Chip Select pin — CS pins of slave devices configured in the SPI panel are listed first
CS Active	enum	Active-Low or Active-High
Bytes	string (hex)	Bytes to send, plain hex (e.g. 9F,00,01). Supports variable.
Store RX in	variable (string)	Optional: variable to store the received bytes

Slave Read Received. Reads the bytes an external master has clocked into this board. Requires SPI Slave mode. Result in last_read_value and, optionally, a named variable.

Parameter	Type	Description
Store in	variable (string)	Optional: variable to store the received bytes

Clear Buffer. Discards the SPI slave receive buffer. No parameters.

6.7 I2C

The bus is configured in the I2C panel; sequences use it as configured. Addresses use plain hex (e.g. 27) and are validated against the 7-bit range 0x03–0x77.

Write (Master). Writes bytes to an I2C slave device. Requires I2C Master mode.

Parameter	Type	Description
Address	int (hex)	7-bit slave address, plain hex
Bytes	string (hex)	Bytes to write, plain hex. Supports variable.

Read (Master). Reads bytes from an I2C slave device. Requires I2C Master mode. Result in last_read_value and, optionally, a named variable.

Parameter	Type	Description
Address	int (hex)	7-bit slave address, plain hex
Count	int	Number of bytes to read (1–32)
Store in	variable (string)	Optional: variable to store the received bytes

Slave Read Received. Reads the bytes an external master has written to this board. Requires I2C Slave mode.

Parameter	Type	Description
Store in	variable (string)	Optional: variable to store the received bytes

Slave Load TX. Preloads the response bytes this board will send when an external master reads from it. Requires I2C Slave mode. The 32-byte AVR buffer limit applies.

Parameter	Type	Description
Bytes	string (hex)	Bytes to preload, plain hex. Supports variable.

Clear Buffer. Discards the I2C slave receive buffer. No parameters.

6.8 UART

UART steps require a board with spare hardware UART channels (Mega 2560: channels 1–3, Micro: channel 1) and the channel must already be open in that board's UART panel. Byte fields offer the same ASCII / HEX toggle as the panel.

Write. Sends bytes over a UART channel.

Parameter	Type	Description
Channel	int	UART channel 1, 2, or 3 (board-dependent)
Bytes	string	ASCII text or hex bytes to send. Supports variable.

Read. Reads the current receive buffer of a UART channel. Result in last_read_value and, optionally, a named variable.

Parameter	Type	Description
Channel	int	UART channel 1, 2, or 3 (board-dependent)
Store in	variable (string)	Optional: variable to store the received bytes

Clear Buffer. Discards the receive buffer of a UART channel.

Parameter	Type	Description
Channel	int	UART channel 1, 2, or 3 (board-dependent)

6.9 DAQ

Acquire. Captures one or two analog channels for a fixed duration. Execution blocks until the duration elapses — the run console shows live progress (for example, Acquiring A0… 3.2 s / 5.0 s) — then the acquisition stops automatically. The captured buffer is held in memory for DAQ Analyze steps and the daq_* functions. Unlike SPI/I2C, the channels are chosen in the step itself, because they are test-specific.

Parameter	Type	Description
Channel 1	int	Primary analog channel
Channel 2	int / None	Optional second channel (must differ from Channel 1)
Frequency	int (Hz)	Sampling rate, 1–100 Hz. Supports variable.
Duration	int (s)	Acquisition length, 1–300 s. Supports variable.

DAQ is slow logging over USB serial — 1–100 Hz for 1–300 s — not an oscilloscope. After the run, each Acquire row shows a View Graph button plotting that capture's voltage over time; every Acquire step keeps its own buffer until the next run.

Analyze. Computes a measurement from the most recent captured buffer — pure math, no board communication. Result in last_read_value and, optionally, a named variable.

Parameter	Type	Description
Function	enum	One of the DAQ analysis functions (section 5.4)
Channel	1 or 2	Acquisition slot from the last DAQ Acquire step
Store in	variable (float/int)	Optional: variable to store the result

6.10 Flow

Loop Start / Loop End. Repeats the enclosed steps. Loops can be nested to any depth; each nesting level is indented in the grid, and Loop End always closes the innermost open loop. Every loop has a name (default loop1, loop2, …), and its iteration counter is available as the built-in variable {name}_index — for example outer_index — so counters stay unambiguous when loops nest.

Parameter	Type	Description
Loop name	string	Identifier for this loop; its counter is {name}_index
Mode	enum	Fixed count or While condition
Count	int	Number of iterations (Fixed count mode, ≥ 1). Supports variable.
Condition	expression	Loop continues while true (While mode)

Parameter	Type	Description
	(bool)	

6.11 Logic

Check. Evaluates an expression and records PASS or FAIL. Board-independent — a pure evaluation of variables and functions, for conditions that are not tied to a single step's output.

Parameter	Type	Description
Expression	expression (bool)	Must evaluate to true for PASS

Log. Writes a message to the run console. Does not affect the verdict.

Parameter	Type	Description
Message	string template	Text with {variable} placeholders resolved at run time (e.g. "V = {voltage}")
Include in report	bool	If checked, the line appears in the PDF report's narrative section

Comment. A no-operation step used purely to structure long plans in the grid. Skipped instantly at execution and never contributes to the verdict.

Parameter	Type	Description
Text	string	Comment text shown in the grid

Operator Prompt. Pauses the run and shows a message dialog to the operator — for example, "Connect probe to TP3". The run continues when the operator clicks OK, or this step fails when the operator clicks Fail. Prompt time counts toward the run duration.

Parameter	Type	Description
Message	string template	Instruction text with {variable} placeholders resolved at run time

6.12 Variables

Assign. Sets a variable to a plain value — no operators or functions. This is the way to set booleans and to copy values between variables.

Parameter	Type	Description
Variable	variable	Target variable
Value	literal / variable	A type-checked literal, or another variable of the same type

Math Operation. Evaluates a numeric expression and stores the result in a variable. Has an optional Limit to judge the computed result inline.

Parameter	Type	Description
Variable	variable	Target variable
Expression	expression	Numeric expression; the result type must match the variable type

String Operation. Evaluates a string expression — including the byte helpers for SPI/I2C/UART responses — and stores the result in a variable. Has an optional Limit.

Parameter	Type	Description
Variable	variable	Target variable
Expression	expression	String expression; the result type must match the variable type

7. Limits — Pass/Fail on Steps

Value-producing steps — Read Digital, Read Analog, SPI Send (received bytes), the SPI/I2C/UART reads, DAQ Analyze, Math Operation, and String Operation — carry a Limit: a pass/fail check evaluated against the step's output, which is available in the limit as the keyword **value**.

- **None** — the step just executes and shows its measured value.
- **Simple** — a quick, typed input: an expected True/False for bool outputs; an operator and value, or a Min–Max range, for numeric outputs; Equals or Contains for string outputs.
- **Formula** — a free expression over value, your variables, and all functions — for example value >= 2 AND value <= 3, or contains(value, "4F").

A ? button next to the Limit section explains the available modes, operators, and formula possibilities for the step's output type, with examples.

A step with a Limit contributes to the overall verdict exactly like a Check step (section 8.3). For checks that span several values or are not tied to one step's output, use the standalone Check step.

[SCREENSHOT PLACEHOLDER — Step Editor: Read Analog step with a numeric Range limit]

8. Running a Test Plan

8.1 Pre-run validation

Validation is all-or-nothing: if any check fails, the run does not start, hardware is not touched, and the problems are shown inline on the offending step rows. There are no partial runs. Before every run, the Sequencer verifies that:

- All boards referenced by enabled steps are connected, and board labels are non-blank and unique.
- Pins and channels are valid for their target board.
- The current board configuration satisfies what each step needs — for example, Set Digital on D5 requires D5 configured as an output, SPI Send requires SPI Master mode, UART steps require the channel open. On a mismatch the run is blocked with a precise message, such as "Board X: set D5 to OUTPUT in the DIO panel".
- All expressions parse and reference known variables.
- Loop counts are ≥ 1 and every Loop Start has a matching Loop End.

A plan referencing boards that are not connected can still be opened and edited — the offending steps are flagged, and Run stays blocked until they are resolved.

8.2 During the run

- All board panels, including DAQ, are locked with the familiar busy overlay. The lockout works both ways: while DAQ or another panel process is running, the Sequencer is locked, and a run is refused with a message naming what is still running.
- Automatic panel activity on the target boards (auto-reads, slave and UART polling) is paused at run start and resumed when the run ends.
- The currently executing step is highlighted; every completed step shows its result and execution time on its row, and the run console streams each action as it happens.
- Disabled steps are skipped and marked SKIP.
- If a board disconnects unexpectedly, the run stops immediately, all panels are unlocked, and an error is shown.

A save is never forced before Run — an unsaved or modified session is runnable.

8.3 Pass/fail semantics

- Only steps with a Limit (Simple or Formula) and standalone Check steps contribute to the verdict.
- The overall result is PASS only if every contributing check passed.
- With **Stop on first failure** checked, execution halts at the first failed check and the remaining steps are marked SKIP. Unchecked, execution continues — the failing step is marked FAIL and the overall result is FAIL at the end.
- The run duration and timestamp are recorded with every run.

9. Debugging

Three tools help you develop a plan against real hardware without running it blind.

- **Breakpoints** — right-click a step and choose Toggle Breakpoint; a red dot appears in the # column. A normal Run executes at full speed and pauses before a breakpointed step.
- **Step-through** — the Step button starts the plan paused before step 1. While paused (at a breakpoint or in step mode), the next step is highlighted, current variable values are inspectable, the boards stay locked, and the toolbar offers Step (execute the next step, then pause again), Continue (resume full-speed to the next breakpoint or the end), and Stop.
- **Run this Step only** — right-click a step to execute just that step against hardware, with its own validation; the result appears on the row. It runs the step as-is against the current variable store and cannot manufacture missing prerequisites — a Check referencing a variable that no step has set, or a DAQ Analyze with no prior capture, reports a clear error.

Pausing always happens between steps, never mid-step: a DAQ capture or a serial exchange always completes, leaving the hardware in a known state.

10. Results & Reports

10.1 In the Sequencer

- The **Result** column shows PASS ✓ / FAIL ✗ / SKIP / the measured value per step, together with each step's elapsed time. Results are cleared when the next run starts.
- The **run console** streams the run — a timestamped header, each step's action and result, every Log message — and ends with a summary: total checks, passed, failed, verdict, and duration. Its text is selectable and copyable.
- Each **DAQ Acquire** row gains a View Graph button after the run, plotting that acquisition's captured voltages over time.

10.2 Exporting a report

After a run completes, Export Report lets you pick PDF or CSV and a save location. Reports are never written without your say-so unless you enable auto-generation (section 10.3).

- **CSV** — one row per executed step: #, Name, Board, Action, Parameters, Limit, Result, Measured value, Elapsed, Timestamp. Opens directly in Excel.
- **PDF** — the shareable, auditable report: a header with the session file name, run date and time, and the boards used (label, type, COM port, firmware); a prominent verdict block with the check counts and duration; the results table; the Log messages marked Include in report, in order; and an embedded voltage-over-time graph for every DAQ Acquire step.

[SCREENSHOT PLACEHOLDER — PDF report: verdict block and results table]

10.3 Report Settings

The Report… toolbar button opens the Report Settings dialog:

- **Auto-generate** — off (default): reports are exported manually only. On: a report is written automatically at the end of every run.
- **Format** — PDF, CSV, or both.
- **Output folder** — where auto-generated reports are written.
- **Filename template** — built from insert-tokens (session name, date, time, date & time, verdict) with preset date and time formats and a live preview of the resolved filename.

These settings are application-level: they persist across restarts and apply to every plan. The manual Export Report button always remains available.

11. Session Files

SiccaLab uses one file for everything: the session file (.json) holds the panel configuration and the test plan together, so a plan always travels with the configuration it was authored against. File → Open, Save, and Save As act on the whole session.

- **File → New** clears everything — board configuration and plan.
- **New Plan** (Sequencer toolbar) clears only the plan in the editor; the file on disk is untouched until the next File → Save.
- An unsaved-changes indicator (*) is shown on the session file name, whichever part changed. Closing the application or opening another file with unsaved changes prompts first.

Session files are plain JSON — they can be shared between setups, version-controlled, or kept alongside lab notebooks, exactly like configuration files before the Sequencer.

12. Tips & Best Practices

Configure the panels first, then author

The pin and channel dropdowns only offer what is active in the panels, and validation checks the plan against the panel configuration. Set up your boards as if for manual use, save the session, then build the plan on top.

Name your steps

Step names carry into the grid, the run console, and the reports. A well-named plan reads like a test protocol — "Verify rail voltage" beats "A0 read".

Structure long plans with Comment steps

Comments cost nothing at execution and make a fifty-step plan navigable.

Put the pass/fail where the measurement is

Prefer a Limit on the measuring step itself; use standalone Check steps for conditions that combine several values.

Use Log with Include in report

Logged lines marked for the report become the narrative between the numbers — the part a colleague actually reads.

Walk a new plan with Step first

Step through the first run, watch the variables update, then let it run at full speed.

13. Notes & Limitations

Behaviours that are intentional but worth knowing.

Topic	Detail
One activity at a time	A running plan locks all board panels, and a running DAQ or panel process locks the Sequencer. Stop one before starting the other.
The Sequencer never configures boards	Pin directions, SPI/I2C mode, speed and addresses, and UART channels are set in the panels. Validation blocks the run and names the exact change needed.
Execution is sequential	Steps run one at a time, in order — including across boards. There is no parallel execution and no conditional branching (if/else) in this version.
Pausing is between steps	Breakpoints and step-through pause between steps only; a step in progress always completes.
DAQ Acquire blocks the run	The run waits for the acquisition to finish (live progress is shown). The envelope is 1–100 Hz for 1–300 s — slow logging, not an oscilloscope.
Copy/paste scope	Step copy, cut, and paste work within the current session only.
Results are per-run	Grid results and DAQ capture buffers are cleared when the next run starts. Export a report to keep them.

14. Support & Contact

Reach out for product questions, bug reports, feature requests, or licensing matters.

Website
www.siccalab.com

Email
contact@siccalab.com

When reporting a Sequencer issue, please include:

- Your board type(s) and firmware version (visible in the BOARDS sidebar).
- The SiccaLab application version (Help → About).
- The session file (.json), if the issue involves a specific test plan.
- The step type and the exact action that triggered the issue, plus a screenshot if anything visual is involved.

Thank you for using SiccaLab.