ESP32 Four Channel Generator

User Manual

Draft compiled from the verified documentation set produced this project – for review, editing, and reorganization. Not a final release document.

Table of Contents

Part I: Getting Started

Chapter 1: Power On, Start, Stop, and Pause

What this covers: the very first power-up of a Generator that already has its microSD card installed, straight out of the box — plugging in power, turning it on, and using the three

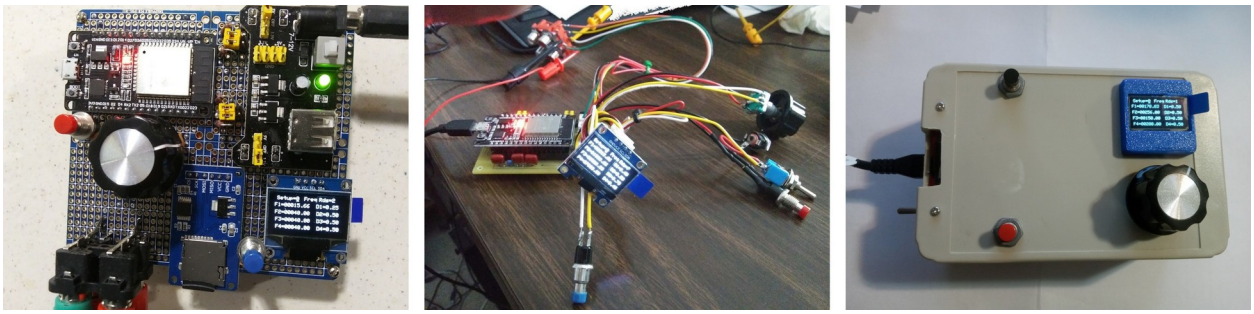
physical controls (the red button, the black cursor button, and the rotary knob) to start, pause, resume, and stop it. It also explains what the OLED display is telling you in each of those states.

Chapter 1 does not cover editing frequency/duty values or the Setup menus — just running what's already on the card. Turning the rotary knob (without pressing it) is how you'd edit values or step through Setup screens; that's covered in a Chapter 2 of this manual.

What the Generator looks like

The Generator in its enclosure, with the 3D-printed display frame.

The two ends. Left: the four channel outputs. Right: power inlet, toggle switch, and the microSD slot.



From left: the Generator built on a proto printed circuit board, before being placed in an enclosure, and the finished unit.

The three controls

Control	What it looks like	What it's for
Toggle switch	The only toggle switch on the case	Master power on/off for the whole unit
Red button	Round red pushbutton	**Start/Stop** — one button, toggles between the two
Black button	Round pushbutton, same shape as the red button — just black	**Pause/Resume** while running; moves the on-screen cursor while stopped
Rotary knob	The dial, with a built-in pushbutton	Turning it edits values/menus (not covered

Control	What it looks like	What it's for
		here); **pressing it in** is a full hardware reset (see the last section)

Step 1: Power on

1. Plug the power supply into the Generator.
2. Flip the toggle switch on the case to the **up** position.

The OLED display lights up and the Generator boots directly into **Stopped** mode — it does not start generating on its own.

Power supply and output voltage

The Generator runs on any DC supply between **5 volts and 12 volts**. There is nothing to set and no jumper to move for this — it simply works across that range.

What makes this worth knowing is that the supply voltage **determines the output voltage**. All four channels switch between 0 V and very slightly below whatever you feed the Generator:

```
Supply  5.0 V   ->   output swings 0 to about  4.9 V
Supply  7.5 V   ->   output swings 0 to about  7.4 V
Supply 12.0 V   ->   output swings 0 to about 11.8 V
```

The small shortfall is the drop across the output switching stage. It is roughly a fixed fraction of a volt rather than a percentage, so it matters proportionally less the higher you go.

About the internal jumpers. The board carries one jumper per channel — J1 to J4 on the schematic in Appendix D — selecting which rail feeds that channel's output stage. **Every unit ships with all four set to the 12 V position**, which is what makes the outputs follow the supply as described above. You should never need to open the case. To get 5 V outputs, use a 5 V wall module; do not move the jumpers.

The practical consequence: if you need a square wave of a particular amplitude, choose the power supply to match and the Generator does the rest. Wanting an output that swings 0 to 7.5 V is simply a matter of powering it from a 7.5 V supply, which gives about 0 to 7.4 V. There is no amplitude control on any Setup screen because the supply *is* the amplitude control.

Step 2: What the display shows when Stopped

On first boot (and any time it's stopped) the display looks like this:

```
Setup=0   Freq Rds=1
F1=00100.00  D1=0.50
F2=00007.83  D2=0.25
F3=01000.00  D3=1.00
F4=01000.00  D4=0.00
```

Completed Four Channel Generator

- **`Setup=0`** means you're on the normal run/dial screen, not in one of the numbered Setup menus.
- **`Rds=1`** is the record currently loaded — record 1, the factory default.
- **`F1–F4`** are the four channel frequencies in Hz, **`D1–D4`** are their duty cycles (0.00–1.00 = 0%–100%). While stopped, frequencies are always shown padded to 5 digits before the decimal point (e.g. `00100.00`), matching how they're entered — this padding disappears once you start the generator (see below).
- A small underscore `_` cursor mark shows where the black button/rotary knob would currently edit, if you turned or pressed them.

Nothing is being generated yet — all four channel outputs are off.

Step 3: Start the Generator

Press the red button once.

The four channel outputs turn on immediately using whatever record is currently loaded (record 1, shown above, on a fresh unit), and the display switches to the **Running** screen:

```
Running   ID=1
F1=100.00   D1=0.50
F2=7.83     D2=0.25
F3=1000.00  D3=1.00
F4=1000.00  D4=0.00
```

- **`Running`** in large text confirms the generator is actively outputting on all four channels.
- **`ID=1`** shows which record is active (matches `Rds=` from the stopped screen). In Mode 3 (Sweep) this field means something slightly different — see "How Mode 3 (Sweep) actually runs" in Chapter 2.
- The F/D lines are the same four channels and duty cycles as before, just displayed **without** the leading zeros this time (`100.00` instead of `00100.00`) — same values, just a different display format between the stopped and running screens.

Step 4: Pause / Resume while running

While running, press the black cursor button once to pause.

The outputs immediately turn off and the display shows:

```
Suspend   ID=1
F1=100.00   D1=0.50
F2=7.83     D2=0.25
F3=1000.00  D3=1.00
F4=1000.00  D4=0.00
```

This is a genuine pause, not a stop — the Generator remembers exactly where it was in any timed run/program and holds that position while paused.

Press the black cursor button again to resume. The display switches back to `Running`, the outputs turn back on, and any run/program timers pick up right where they left off (the paused time isn't counted against them).

Each press has a short built-in delay (about a half second) before it takes effect, so a single press is deliberate — don't worry about it double-toggling.

The red button does nothing while paused. The black cursor button is the *only* way out of Suspend — the red button is deliberately ignored the whole time you're paused, so an accidental bump of it won't cancel your run. You must press the black button again to resume before the red button will do anything.

Step 5: Stop the Generator

Press the red button again while Running.

The outputs turn off and the display returns to the **Stopped** screen shown in Step 2 (`Setup=0`, leading-zero-padded F/D values). From here you can press the red button again to restart, or use the rotary knob to browse or edit records.

Note: this only works from **Running**. If you're paused (Suspend), the red button won't do anything — see the note at the end of Step 4.

Quick summary

From this state	Press this	You get
Stopped	Red button	Running
Running	Red button	Stopped
Running	Black button	Suspend (paused)
Suspend	Black button	Running (resumed)
Suspend	Red button	**No effect** — ignored on purpose while paused

What pressing the rotary knob does

The rotary knob has a built-in pushbutton, separate from turning the dial. **Pressing it in performs a full hardware reset of the Generator** — the same as switching power off and back on. Everything restarts from scratch: outputs stop, and the unit reboots back to the Step 2 Stopped screen.

Two things worth knowing about it:

- **Screen saver wake-up:** if the Generator sits idle (stopped, untouched) for about 20 minutes, the display switches to a star-field screen saver. Any front panel control brings the display back — turn the knob, or press the black cursor button, or press the red Start/Stop button. Whichever you use, that press only wakes the display and does nothing else: the red button will not start a run, and the black button will not move the cursor. The 20 minutes is idle time, so anything you do on the panel starts it counting again and the screen saver will not interrupt you part way through setting a value. Firmware before Version 2.05 needed a reset to escape the screen saver; that is no longer necessary.
- **USB Comm mode:** connecting the Generator to a PC to send/receive protocol files also starts with a press of this same reset button, held together with the black cursor button. That procedure is covered in the separate USB file-transfer How-To documents — mentioned here just so a reset in that context isn't mistaken for a malfunction.

Because it's a real reset, avoid pressing it by accident while a run you care about is in progress — there's no confirmation prompt, it resets immediately.

Part II: Operating the Generator

Chapter 2: The Setup Screens: What Each One Does

What this covers: the `Setup=` screens (0 through 5) shown in the top-left corner of the display, what each one is for, how to dial a screen's own parameters in, and specifically how to change frequency and duty cycle values on Setup=1. This picks up where the "Power On, Start, Stop, and Pause" guide leaves off — that guide only covers Setup=0.

The Master Control Record's individual fields (memory group, end times, start/stop group IDs, sweep increment, file number) are referenced here by name where each screen edits them — see "The Master Control Record - Field Definitions" for the full field-by-field reference.

Moving between screens

Turning the rotary knob only changes which Setup screen you're on when the on-screen cursor (_) is sitting under the `Setup=` digit itself, top-left of the display. Press the black cursor button (while stopped) to walk the cursor over to that position, then turn the knob to step through Setup 0–5.

How dialing a value in works

Once the cursor is sitting on an editable position, turning the rotary knob changes just that position — the rest of the display's fields hold still. Pressing the black cursor button advances the cursor to the next position; it keeps cycling forward through every editable field on the current screen and then wraps back around to the `Setup=` digit.

Two different styles of "editable position" are used, depending on the field:

- **Digit-by-digit fields** (the four frequencies, the four duty cycles, the two run timers, the sweep frequency increment): each digit has its own cursor stop, and turning the knob one click adds or subtracts exactly that digit's place value — like an odometer. For example, with the cursor on the hundreds digit of F1, one click changes F1 by 100 at a time; move the cursor one step over to the tens digit and a click now changes it by 10.
- **Whole-number fields** (the active record number, Mode, the range/sweep Start and Stop group IDs, the file number): these have a single cursor stop, and each click of the knob counts the whole value up or down by 1.

Holding the black cursor button down, rather than tapping it, walks the cursor forward continuously — a real time-saver on screens with a lot of digits (Setup=1 especially) instead of pressing once per digit to cycle all the way around back to `Setup=`. Keep it held and you'll see the cursor visibly keep advancing on its own for as long as you hold it. This same continuous-hold state is also the key to saving a change to the SD card, covered next.

Setup=0 — Run the Generator

Covered in the "Power On, Start, Stop, and Pause" guide. This is the **only** screen the generator can actually run from — the red button starts/stops it here, nowhere else.

It's also the screen for **viewing the records in the currently active protocols file** — with the cursor on the record-number field (`Rds=`), turn the knob to dial through record numbers and see each one's F1–F4 and D1–D4 values. You cannot edit those digits from Setup=0, only browse them — editing is done on Setup=1 (below).

The Generator always boots from `protocols-0.txt`. That never changes. What it does *not* always do is start on record 1 — it starts from a selected record, whichever one is named in the control record's 2nd parameter. That value is **not** a live memory of "whatever record was showing when the Generator was last powered off" — it only gets updated when someone explicitly confirms an SD-card save (see Setup=1 through Setup=5, below) while that record happens to be the active one. So the starting record reflects the last record active *at the last confirmed save*, which may not be the same as the last record you were actually looking at before shutdown.

Once it's running, `protocols-0.txt` can be **replaced with the contents of any of the other 99 possible `protocols-nn.txt` files** on the microSD card (file numbers 1–99), via the File field on Setup=5. The card itself never needs to be removed for this if you've planned ahead and pre-loaded several protocol files onto it — it only needs to come out if you want to swap in a different *set* of protocol files entirely (a freshly prepared card with its own correctly formatted files).

Setup=1 through Setup=5 — Update Mode

Any screen other than Setup=0 cannot start or stop the generator. The red button doesn't act as Start/Stop here.

While on any of these screens, the rotary knob and black cursor button are used to browse to and edit values — either a record's frequencies/duty cycles, or the Master Control Record's parameters. Understanding what happens when you press the red button here means understanding two separate places your data can live:

- **EEPROM** — the Generator's own working memory. This is what it actually runs from, moment to moment, whether editing or running.
- **The `protocols-0.txt` file on the microSD card** — the permanent, power-cycle-safe copy. Every boot reloads EEPROM from this file, which is also why any change that's only in EEPROM disappears the next time the Generator is powered up.

Pressing the red button alone commits whatever's currently on the screen into EEPROM. The change takes effect immediately — go back to Setup=0 and start the Generator and it'll use the new values — but it is **not** written to the SD card yet.

Saving that change permanently to `protocols-0.txt` takes a second, separate step — and the order matters:

1. **Press the red button by itself first.** This is the step above — it commits your edit into EEPROM. Skip this and the file-save step below will save whatever was in EEPROM *before* your edit, not your edit itself — the Generator only picks up screen changes into EEPROM at this step, and the save-to-file step doesn't re-check the screen, only EEPROM.
2. **Press and hold the black cursor button.** You'll see the cursor start auto-advancing across the screen, confirming it's registering as held down.
3. **While still holding black, press the red button a second time.** This brings up **"Update the SD Card??"**
4. **Keep holding black** through that prompt until the display reads **"SDram Card UPDATING."** Only then has EEPROM actually been written out to `protocols-0.txt`. Let go of black too early and nothing gets saved to the card — your edit still works from EEPROM for the rest of this session, but reverts to the file's old value on the next reboot.

Pressing red alone, or pressing red without black already held down first, only ever updates EEPROM — it will never by itself write to the SD card.

Setup=1 — Edit Frequency & Duty Cycle values

This is the screen used to actually change F1–F4 and D1–D4 for a record — the display looks just like Setup=0 (`Set=` in place of `Rds=`), but here the cursor button doesn't stop at the record number.

To change a record's values:

1. Dial to **Setup=1**.
2. Press the cursor button once — the cursor lands on the record-number field. Turn the knob to pick which record you want to edit (or press the cursor button again immediately to stay on the record already showing).
3. Press the cursor button again to step onto the **first digit of F1**. Each further press moves one digit to the right across F1, then onto D1's two digits, then F2 and D2, then F3 and D3, then F4 and D4 — press through however many digits you need to reach.
4. With the cursor on a digit, turn the knob to change that digit — clockwise for higher, counter clockwise for lower.
5. Once your edits are in, press the red button — this immediately loads your changes into EEPROM, so the Generator will run with the new values right away. To also make the change permanent (survive a power cycle), follow up with the hold-black-then-red-again sequence described just above, in "Setup=1 through Setup=5 — Update Mode."

Setup=2 — Mode

Selects **how** the generator will run once you're back on Setup=0 and press the red button. Turn the knob with the cursor on the Mode digit to choose:

- **Mode 1 — Simple:** runs a single protocol record continuously — whatever record is currently selected, and nothing else.
- **Mode 2 — Protocol (range mode):** consecutively and automatically runs through a range of protocol records, one after another — the range itself is set up on Setup=3.

- **Mode 3 — Sweep:** sweeps from a starting frequency to an ending frequency, needing only two protocol records to define those two endpoints — set up on Setup=4. **Sweep mode only applies to Channel 1** (F1); channels 2–4 just run at whatever fixed frequency/duty is in the starting record for the sweep.
- **Mode 4 — Adjust:** turns the knob into a live frequency control. Pick a record on Setup=0, press the red button, and Channel 1's frequency then follows the knob while the Generator runs — see "How Mode 4 (Adjust Mode) works" below. As with Sweep, **only Channel 1 is affected**; channels 2–4 hold the record's own values.

Setup=3 — Protocol Range (for Mode 2)

Defines which range of protocol records Mode 2 (Protocol) runs through — a Starting group ID and a Stopping group ID. Once running, the generator advances one record at a time through that range, then wraps back to the start.

Setup=4 — Sweep Range (for Mode 3)

Defines the two protocol records Mode 3 (Sweep) sweeps between — a Starting group ID and a Stopping group ID — plus the **Frequency Increment/Decrement** value: how much Channel 1's frequency changes by on each step of the sweep.

Setup=5 — Timers and File Selection

Three things live on this screen:

- **Freq Time** — how long each frequency record is held before the generator advances to the next one (used by Mode 2's range and Mode 3's sweep).
- **Progm Time** — how long the overall program runs before stopping. A setting of 0000 means half a minute (30 seconds), not zero.
- **File** — the protocol file number (0–99) to load. This field uses its own confirmation step, separate from the general Setup=1–5 save described above: dial in a file number, then press the red button. That **prepares** the load and shows a "Load Proto = nn" prompt — it hasn't actually loaded anything yet. Press the black cursor button within the next **5 seconds** to confirm, and the Generator copies that `protocols-nn.txt` file's contents into `protocols-0.txt`, making it the new active/operational file. Miss the 5-second window and the load is simply abandoned — nothing changes. This is the mechanism referred to under Setup=0 above for switching which protocol set is running.

How Mode 3 (Sweep) actually runs

Sweep mode is the one mode whose behaviour is not obvious from the setup screens alone, so it is worth walking through. A sweep needs four values, spread across two screens:

- **Start Sweep Group** (Setup=4) — the record whose F1 is the frequency the sweep **starts** from. This record also supplies everything else the Generator puts out: D1, and all of F2/D2, F3/D3 and F4/D4, which stay fixed at those values for the entire sweep.
- **Stop Sweep Group** (Setup=4) — the record whose F1 is the frequency the sweep **stops** at. Nothing else in this record is used — only its F1 value matters.

- **Freq. Inc.** (Setup=4) — how far Channel 1's frequency moves at each step.
- **Freq Time** (Setup=5) — how long each step is held before moving to the next one.

A worked example. In the supplied `protocols-1.txt`, record 3 holds 5.00 Hz and record 4 holds 10.00 Hz. Set:

```
Start Sweep Group = 3       (start at   5.00 Hz)
Stop Sweep Group  = 4       (stop at   10.00 Hz)
Freq. Inc.        = 1.00    (move 1 Hz each step)
Freq Time         = 1       (hold each step 1 second)
```

Press the red button and Channel 1 runs:

`5.00 → 6.00 → 7.00 → 8.00 → 9.00 → 10.00`

one second per step. On reaching 10.00 Hz the sweep starts over from 5.00 Hz and keeps repeating until Progm Time runs out and the Generator stops. Changing Freq. Inc. to 0.50 halves the step size and doubles the number of steps — 5.00, 5.50, 6.00 … 10.00 — each still held for Freq Time.

Sweeping downward. Nothing special is needed. Put the higher frequency in the Start Sweep Group and the lower one in the Stop Sweep Group, and the Generator works out the direction by itself. Freq. Inc. is always entered as a positive number.

What ID= shows during a sweep. This is worth understanding, because it does **not** work the way it does in Mode 1 or Mode 2. In those modes `ID=` is the record currently being output, and in Mode 2 you watch it count up through the range. A sweep, by contrast, **never changes records** — it stays on the Start Sweep Group the whole time and simply recalculates F1 for itself, one step at a time. So:

- `ID=` reads the **Start Sweep Group** number for the whole sweep. In the example above it shows `ID=3` and stays there.
- On the single step where the sweep lands exactly on the limit frequency, `ID=` changes to the **Stop Sweep Group** number — `ID=4` in the example, on the 10.00 Hz step. That is your confirmation that the sweep reached its endpoint.
- It then returns to `ID=3` as the next pass begins.

If you have run firmware older than Version 2.02c you may remember `ID=` counting **down** during a sweep — 6, 5, 4 and so on. That was an internal step counter being put on the screen by mistake; it never referred to a record at all. It has been corrected.

Letting the Generator choose the step size. If Freq. Inc. is left at `0.00`, the Generator works out a step size for you from the two timers on Setup=5: it divides Progm Time by Freq Time to see how many steps will fit in the run, then spreads the whole start-to-stop frequency span across them. The result is one slow sweep that takes about the entire program run to complete, instead of a quick sweep repeating many times.

Landing on the endpoint. The sweep always finishes exactly on the stop frequency, and Freq. Inc. is the step it actually uses. Where the span does not divide evenly by it, the Generator takes as many full-size steps as will fit and then makes the LAST step a short one, so the sweep still

lands on the limit rather than overshooting it or stopping short. Sweeping 5.00 to 10.00 Hz — a span of 5 — with Freq. Inc. set to 0.75 therefore runs 5.00, 5.75, 6.50, 7.25, 8.00, 8.75, 9.50 and then 10.00, that final step being 0.50 rather than 0.75. Firmware before Version 2.05 behaved differently: it used Freq. Inc. only to work out how many steps would fit and then divided the whole span evenly between them, so the step actually used was rarely the number entered — a sweep of 2 to 1000 Hz asking for 50 stepped by 52.53.

If Freq. Inc. is larger than the span. Say your two records are 5 Hz apart but Freq. Inc. is set to 10.00. There is no room for even one intermediate step, so the Generator does the next most sensible thing: Channel 1 alternates straight between the start frequency and the stop frequency, holding each for Freq Time. Nothing breaks — but you get a two-point alternation rather than a sweep, so if that is not what you wanted, bring Freq. Inc. back below the difference between your two frequencies. Setting both Sweep Groups to the same record behaves the same way, holding one steady frequency.

How Mode 4 (Adjust Mode) works

Mode 4 does something none of the other modes do: it hands the frequency control to you. Instead of the Generator stepping through records or sweeping on a timer, you move Channel 1 by hand while it is running and see the effect straight away. It is the mode to use when you are looking for a frequency rather than replaying one you already know.

Getting there. Three steps: on Setup=2 dial Mode to 4; go back to Setup=0 and dial to the record you want to start from; press the red button.

The starting frequency and duty cycle are simply whatever F1 and D1 are showing on Setup=0 at the moment you press the red button. There is no separate record to set up for this mode — if the screen says 10.00 Hz, that is where you start.

The screen. Only Channel 1 is shown, because only Channel 1 is being adjusted:

```
ADJUST MODE      Rds=3
 F=10.00
 D=0.50
```

Whichever value is currently under the knob's control is drawn in inverse video — dark text on a light bar. Above, that is the F line. `Rds=` shows the record you started from; it does not change while you dial.

Both values are indented one space, which keeps the F and D letters clear of the edge of the highlight bar and makes them easier to pick out at a glance. Above 9999.99 Hz that space is dropped automatically, because at this text size a line holds only ten characters and `F=12345.67` already uses all ten.

The controls. The knob moves whichever value is highlighted. The **black** button flips the highlight between F and D. The **red** button stops the Generator and returns you to the normal Setup=0 screen.

Step sizes. Frequency moves by the **Freq. Inc.** value from Setup=4 — the same field Sweep mode uses, so setting it to 0.50 gives you half-hertz clicks here too. If Freq. Inc. happens to be

0.00, Adjust Mode falls back to 1.00 Hz per click rather than leaving you with a knob that does nothing. Duty cycle moves by 0.05 per click, which is fixed and not adjustable from any screen.

Nothing is saved. Adjustments in this mode are deliberately temporary. The protocol record is never written to, so however far you dial, pressing the red button leaves the record exactly as it was. Experiment freely — you cannot damage a protocol you spent time building. The flip side is that if you find a value worth keeping, write it down: it is gone the moment you stop. To make it permanent, dial it into the record on Setup=1 afterwards.

There is no Pause in this mode. The black button is the F/D toggle here, so it cannot also serve as Pause. The red button stops, as it does everywhere else.

What to expect from the knob. Every click reprograms the PWM hardware, and that takes a moment. Turn the knob at a normal pace and the output keeps up. Spin it quickly and the output falls behind, then catches up a moment after you stop. No clicks are ever lost — the count stays correct no matter how fast you turn — and once the knob settles the output always matches the number on the screen. This is a limitation of how the ESP32's PWM hardware accepts frequency changes while it is running, not something that can be dialled out.

See "The Master Control Record - Field Definitions" for the full field-by-field reference for everything stored in the control record.

Chapter 3: The Master Control Record: Field Definitions

What this covers: the 11 fields of the Master Control Record — the special record (always `protoID` 0, always the last line of a `protocols-nn.txt` file) that stores everything about *how* the Generator runs, as opposed to a normal frequency record which just stores one set of F1–F4/D1–D4 values. Column letters below match the table already in "Sending Protocols Files From a PC to the Generator" for that file's CSV layout; this document adds the valid range for each field and which Setup screen (if any) is used to dial it in on the device itself.

Ranges are taken directly from the firmware's own `rotaryLimits()` function (`rotary_Decode.ino`), so they match exactly what the Generator itself will clamp a value to — not just what seems reasonable.

Col	Field	Range	Set on screen	What it does
A	(always 0)	fixed	—	Marks this row as the control record, not a frequency record.
B	Start protoID	1 to however	(not directly	The record the

Col	Field	Range	Set on screen	What it does
		many records are in the file	dialed)	Generator starts on — primarily meaningful for **Simple mode**, since Protocol and Sweep modes use their own Start Group fields (F/H below) instead. **Not** manually edited on any Setup screen — it's set automatically to whatever record was active at the moment of the *last confirmed SD-card save* (see the Setup screens document for how EEPROM-vs-file saves work).
C	Mode	1–3	Setup=2	1 = Simple (run one record continuously), 2 = Protocol (run a range of records in sequence), 3 = Sweep (sweep Channel 1 between two records' frequencies).
D	Freq Time	0–9999 seconds	Setup=5 ("Freq Time")	How long each record is held before advancing to the next one — used by Protocol mode's range

Col	Field	Range	Set on screen	What it does
				stepping.
E	Progm Time	0–9999 minutes (0000 = ½ minute)	Setup=5 (“Progm Time”)	How long the overall program runs before stopping automatically. A value of 0000 is not zero — it selects half a minute (30 seconds).
F	Start Freq Group	1 to however many records are in the file	Setup=3	First record in the range Protocol mode (Mode 2) runs through.
G	Stop Freq Group	1 to however many records are in the file	Setup=3	Last record in that range — Protocol mode wraps back to the Start once it passes this one.
H	Start Sweep Group	1 to however many records are in the file	Setup=4	The record defining the sweep’s starting frequency (Channel 1 only — see the Setup screens document).
I	Stop Sweep Group	1 to however many records are in the file	Setup=4	The record defining the sweep’s ending frequency.
J	Sweep Inc	0.00–9999.99	Setup=4 (“Freq. Inc.”)	How much Channel 1’s frequency changes by on each step of the sweep.
K	File ID	0–99	Setup=5 (“File”)	On the device, this is the file-number field

Col	Field	Range	Set on screen	What it does
				used to load a different `protocols-nn.txt` into `protocols-0.txt` (see the Setup screens document for the 5-second confirm). In an uploaded file from a PC, this column is meaningless — the filename alone decides the SD card slot, so leave it at 0 there.

A note on the range fields (F, G, H, I)

Group ID fields (Start/Stop Freq Group, Start/Stop Sweep Group, and Start protoID) are only ever valid pointing at a record that actually exists in the current file — the Generator clamps them to between 1 and the highest record number present. If a file is edited down to fewer records than one of these fields pointed at, the firmware resets the offending field rather than pointing at a record that no longer exists.

A note on Freq Time and Progm Time

Internally these are tracked in milliseconds and always rounded down to a whole second (Freq Time) or whole minute (Progm Time) — any finer value gets truncated. The CSV file and the Setup=5 screen both show them already converted to those whole units. There is one deliberate exception, and it is easy to mistake for a fault: a Progm Time of 0000 does not mean zero, it means half a minute (30 seconds). Dialling Progm Time below 1 on Setup=5 selects that half-minute setting, and it displays as 0000 only because the screen rounds down to whole minutes. This is by design.

Part III: The microSD Card and Protocol Files

Chapter 4: The microSD Card: Requirements and File Management

What this covers: what the microSD card in the Generator needs to look like, how the `protocols-nn.txt` files on it relate to each other, and how to manage them without losing work. This replaces two older documents in this folder — "Generator Micro Sdram FILES and FORMAT" and "CREATING A NEW SDRAM" — both of which describe an earlier version of the Generator (a single `protocols.txt` file, a 9-file limit, a 50000 Hz frequency cap, and raw-millisecond values in the file itself). Everything below is checked against the current firmware instead.

Card requirements

The Generator reads the card using the standard Arduino `SD` library over SPI — this only supports cards formatted **FAT16 or FAT32**, not exFAT. Cards 32 GB and under almost always ship formatted FAT32 already; cards larger than that (SDXC) typically ship formatted exFAT by default and would need to be reformatted to FAT32 before the Generator can read them. Staying at or under 32 GB is the simplest way to avoid that step entirely — and since the protocol files themselves are tiny (a few hundred lines at most per file), there's no capacity reason to reach for a larger card anyway.

Every `protocols-nn.txt` file **must sit in the root directory of the card** — not inside a folder. The Generator only ever looks for `/protocols-0.txt`, `/protocols-1.txt`, and so on, with nothing in front of that leading slash.

The files themselves

- **`protocols-0.txt` is mandatory.** This is the one file the Generator cannot run without — it's read unconditionally on every power-up, and it's the only file that's ever actively running. Every other file number is optional.
- **`protocols-1.txt` through `protocols-99.txt` are optional presets.** None of them run directly — the Generator's only route to "running" a different protocol set is to load it into `protocols-0.txt` first, via the File field on Setup=5 (covered in "The Setup Screens" document). Loading a preset **overwrites** whatever was in `protocols-0.txt` at the time, so anything unsaved there is lost the moment a different preset is confirmed in.
- **Keep a full set of preset files on the card**, even if some are duplicates or simple placeholders — switching between them from the Generator itself, with no PC involved, only works if the file is already sitting on the card under the right name. There's no way to create a brand-new `protocols-nn.txt` file from the device itself; new files can only be added by editing them on a PC (with `protocol_editor.py` or the spreadsheet workflow) and copying them onto the card.

How many records does this file have?

Nothing on the screen tells you directly, and a `protocols-nn.txt` file can hold anywhere from a single record to 275 of them. From Version 2.05 that ceiling is enforced rather than advisory: a file holding more is refused instead of part-loaded, the display reads TOO MANY, and the PC upload tool rejects it before it ever reaches the card. There is a quick way to find out.

On Setup=0, dial `Rds=` down to 1 — then give the knob **one more click down**. Rather than stopping, it wraps around to the highest record number in the file. That number is the record count. It wraps the other way too: dial past the top and you come back to 1.

This is the fastest way to confirm you have loaded the file you meant to. If you were expecting a forty-record set and it wraps to 8, you have the wrong file — better to find that out now than part way through a session.

One file can hold several sequences

It is natural to assume one protocol per file, using the 0–99 filenames to keep them apart. That works, but it is not the only way, and often not the best one.

Because Mode 2 runs a **range** — a Starting and a Stopping group, set on Setup=3 — one file can hold many independent sequences side by side. Suppose a favourite sequence needs only eight records. A single file with room for hundreds could be laid out like this:

```
Records  1 - 8    an eight-step sequence
Records  9 - 20   a longer session
Records 21 - 24   a short four-frequency set
Records 25 - 60   spare slots for later
```

To run the second one, set Setup=3's range to 9 and 20. To run the first, set it to 1 and 8. Same file, no card swapping, no PC. Mode 3 works the same way: the Start and Stop Sweep Groups on Setup=4 can point at any two records anywhere in the file.

So the ninety-nine available filenames are not a limit on how many sequences you can keep — they are a convenience. Many users will find everything they need fits comfortably in one well-organised file.

One practical consequence: keep a written note of which record ranges hold what. The Generator has no way to label a range, and a list of bare frequencies gives no hint where one sequence ends and the next begins. A comment column in the spreadsheet you build the file from, or a note kept with the card, saves a lot of counting later.

You can't add new records from the device — plan for that

The number of frequency records available to browse or edit on the Generator (Setup=0's `Rds=`, Setup=1's `Set=`) is fixed at whatever count was in the file when it was loaded — there is no on-device "insert a new record" function. If you want the ability to add protocols later using only the Generator's own controls, **build extra placeholder records into the file ahead of time** on a PC (all-zero values work fine as a placeholder) before it ever goes on the card. Then

"adding" a protocol from the device is really just dialing real values into one of those spare slots on Setup=1, rather than trying to create a slot that doesn't exist.

What happens if a file is missing or the card fails

The Generator checks for the card and for `protocols-0.txt` specifically at every boot, and neither failure is recoverable without fixing the card and power-cycling:

- **No card detected** (or the card can't be read at all): the display shows `Micro SD` / `FAILURE` and the Generator locks up — it does not retry or fall back to running from memory.
- **`protocols-0.txt` (or whichever file number was requested) doesn't exist on the card**: the display shows `File <n>` / `FAILURE` and locks up the same way.

Both are simple "file failed to open" checks, not deep validation of the file's contents — a `protocols-0.txt` that exists but has scrambled or missing fields won't necessarily trigger this screen; it'll more likely just run with wrong values. Malformed files are best caught before they ever reach the card, by using `protocol_editor.py` rather than hand-editing.

Related documents: "The Setup Screens - What Each One Does" (Setup=5's File field and the load-confirmation sequence), "The Master Control Record - Field Definitions" (the File ID field), and "Sending Protocols Files From a PC to the Generator" (the file format and naming rules in detail).

Chapter 5: Sending Protocol Files From a PC to the Generator

What this does: sends a `protocols-nn.txt` file from your PC to the Generator over the USB cable, and writes it onto its microSD card — no need to remove the card. On the Generator's own screen this is the **"Receive Fm CPU"** menu option. In the PC-side program it's the **"Upload to Generator"** button.

This document also covers preparing the file in a spreadsheet (Excel or LibreOffice Calc) and what each column means. A separate, more detailed file-format reference document may follow later — this section covers what you need to safely build/edit a file for upload.

Preparing the file in a spreadsheet

1. Open Excel or LibreOffice Calc.
2. Lay the data out as plain columns of numbers, comma-separated when saved (see the layout below) — one row per frequency record, plus exactly one more row at the very end for the control record.

3. **Save as CSV** (Excel: “CSV (Comma delimited)”; Calc: “Text CSV”, comma as field separator, UTF-8 encoding is fine).
4. **Rename the saved file so it ends in `.txt`, not `.csv`**, and make sure the name is exactly `protocols-N.txt` where `N` is a number from 0–99 — nothing before “protocols-” and nothing between the number and “.txt”. Names like `my_protocols-5.txt` or `protocols-5-final.txt` will be rejected; the Generator’s tool uses this filename, on your PC, to know which slot on the SD card to write.

You don’t need to worry much about exact number formatting in the spreadsheet — before anything is sent, the tool automatically: - strips stray quote marks or extra spaces some spreadsheet exports add, - strips a UTF-8 “byte order mark” if your spreadsheet program added one, - reformats frequency/duty-cycle numbers to the 2-decimal format the Generator expects (e.g. `150` becomes `150.00`), - checks every value is in range and every line has the right number of columns.

If anything doesn’t check out, **nothing is sent** — you’ll get a message naming the exact line and what’s wrong with it, so you can fix it in the spreadsheet and try again.

File layout at a glance

- One **frequency record** line per Memory Group — as many as you want (real files commonly have 200+).
- Followed by **exactly one control record** line — always the *last* line in the file.
- An optional first line starting with # (or any line with no commas) is treated as a title and ignored — handy for a spreadsheet header row.

Frequency record — 9 columns, one row per Memory Group

Col	Field	Meaning
A	protoID	Memory Group number for this row (whole number). Other rows and the control record refer back to this number to define ranges.
B	F1	Channel 1 frequency, Hz
C	D1	Channel 1 duty cycle, 0.00–1.00
D	F2	Channel 2 frequency, Hz
E	D2	Channel 2 duty cycle, 0.00–1.00
F	F3	Channel 3 frequency, Hz
G	D3	Channel 3 duty cycle, 0.00–1.00
H	F4	Channel 4 frequency, Hz
I	D4	Channel 4 duty cycle, 0.00–

Col	Field	Meaning
		1.00

Control record — 11 columns, exactly one row, always last

Col	Field	Meaning
A	*(always 0)*	Fixed at 0 — this is what marks the row as the control record rather than a frequency record
B	Start protoID	Which Memory Group to run when in Simple Mode
C	Mode	1 = Simple, 2 = Protocol, 3 = Sweep
D	Freq Time	Frequency run time, whole **seconds** (0–9999) — matches "Freq Time" on the Generator's screen 5
E	Progm Time	Program run time, whole **minutes** (0–9999) — matches "Progm Time" on screen 5
F	Start Freq Group	Start of the Frequency Range (protoID) used in Protocol mode
G	Stop Freq Group	Stop of the Frequency Range
H	Start Sweep Group	Start of the Sweep Range used in Sweep mode
I	Stop Sweep Group	Stop of the Sweep Range
J	Sweep Inc	Sweep frequency increment, 2 decimals
K	File ID	**Leave this at 0.** It does not control which SD card slot gets written — see below for what actually does.

Which Generator file actually gets overwritten

The filename you upload — `protocols-N.txt` — is what determines the SD card slot, **not** column K above:

- **Uploading `protocols-0.txt`** takes effect **immediately** — the Generator reloads and starts running the new data right away, no reset needed.

- **Uploading any other number** (`protocols-1.txt` through `protocols-99.txt`) writes it to the SD card but does **not** change what the Generator is currently running. It stays inactive until you separately select it on the Generator itself: on **screen 5**, dial **File = N** to the number you uploaded, press the red button, then confirm with the black button within 5 seconds.

Step-by-step procedure

1. **Make sure the USB cable is connected** between the Generator and PC — it both powers the Generator and carries the data.

2. Open a terminal (Linux) or Command Prompt (Windows) and start the tool:

   - Linux: `cd ~/Arduino/ESP32_Four_Channel_Generator_V2_02c` then `python3 protocol_tool_gui.py`
   - Windows: `cd path\to\ESP32_Four_Channel_Generator_V2_02c` then `python protocol_tool_gui.py`

3. In the window:

   - **Serial port:** pick your port from the dropdown (Refresh if needed).
   - **Protocol file:** Browse to (or type) the `protocols-N.txt` file you prepared in the spreadsheet.

4. Click **"Upload to Generator."** The tool checks your file's formatting *immediately, before touching the Generator at all* — if there's a problem, you'll see it right away and can fix it without doing anything on the Generator side. If the file checks out, a *"Confirm to Proceed with Upload?"* popup appears — **leave it open, don't click Yes yet.**

5. On the Generator: press the **reset button** (the rotary encoder's built-in pushbutton), and press-and-hold the **black cursor button** either just before or just after reset — it needs to be held through the brief moment, a second or two after reset, when the startup code checks for it. Either order works.

6. Keep holding the black button: the OLED shows **"USB Comm."** and cycles through *Send To CPU*, *Receive Fm CPU*, *Return to Main* roughly every quarter second, with an arrow marking the current one.

7. **Release** the black button the instant **"-> Receive Fm CPU"** is shown.

8. Press the **red button** once to confirm. The screen briefly shows *"Turn off / MiniCom / Start Py"* (safe to ignore), then the Generator starts waiting for the PC — for uploads, this wait window is about **30 seconds**.

9. Back on the PC — within that window — click **Yes** on the *"Confirm to Proceed with Upload?"* popup.

10. Watch the **Activity Log**: you'll see `ACK:FREQ:1`, `ACK:FREQ:2`, ..., `ACK:CTRL`, then `UPLOAD_OK`, ending with *"Upload succeeded."* The Generator's screen shows **"SEND GOOD"** (same fixed wording used for both directions).

11. If you uploaded `protocols-0.txt`, the Generator is already running the new data — nothing else to do. If you uploaded any other number, activate it via screen 5 as described above whenever you're ready.

If something goes wrong

- **"Cannot upload — file needs fixing: …"** — the tool caught a problem in your spreadsheet-saved file before it ever touched the Generator. The message names the exact line and reason (wrong number of columns, a non-numeric value, a run time outside 0–9999, etc.). Fix it in the spreadsheet, re-save, try again.
- **"…must be named 'protocols-N.txt'"** — the filename doesn't match the required pattern. Rename it exactly.
- **"Timed out waiting for UPLOAD_READY"** — you didn't click Yes within the ~30-second window, or the black button wasn't held long enough at boot. Start again from step 5.
- **`ERROR:BAD_FREQ_LINE` / `ERROR:BAD_CTRL_LINE` / `ERROR:BAD_TOKEN_COUNT` from the Generator** — the same kind of problem, but caught on the Generator's side. This shouldn't normally happen since the PC-side check catches these first, but if it does, the reported line is what needs fixing.
- **Serial error / "port busy"** — another program has the port open (Arduino IDE Serial Monitor, a terminal, minicom, etc.). Close it and try again.

Chapter 6: Sending Protocol Files From the Generator to a PC

What this does: pulls the content of one specific `protocols-nn.txt` file straight off the Generator's microSD card and saves it on your PC — over the USB cable, without ever removing the microSD card. On the Generator's own screen this is the **"Send To CPU"** menu option. In the PC-side program it's the **"Download from Generator"** button.

This document covers the point-and-click tool, `protocol_tool_gui.py`.

Before you start (one-time setup per PC)

Linux

1. Python 3 is normally already installed — check with `python3 --version`.
2. Install pyserial (the library that talks to the USB port): `sudo apt install python3-serial` (or `pip3 install pyserial` if you're not on a Debian/Ubuntu-based system).
3. `tkinter` (needed for the window itself) is included by default on most distros. If the program complains it's missing: `sudo apt install python3-tk`
4. Make sure your account can use the USB serial port — run `groups` and check `dialout` is listed. If it isn't: `sudo usermod -aG dialout $USER`, then log out and back in.

Windows *(setup steps below, not yet verified on real hardware — please confirm/correct after trying)*

1. Install Python 3 from **python.org**. On the first install screen, check **"Add python.exe to PATH"**. Leave the default components selected — the official installer includes `tkinter`.
2. Open Command Prompt and install pyserial: `pip install pyserial`
3. If Windows doesn't automatically detect the Generator's USB port, install the **CP210x USB-to-UART driver** (the Generator uses a CP2102 chip) from Silicon Labs' website.

Every time: find your serial port name

Linux

Plug in the USB cable, then run `ls /dev/ttyUSB*`. The Generator normally shows up as `/dev/ttyUSB0`.

Windows

Open **Device Manager → Ports (COM & LPT)**. The Generator shows up as something like *"Silicon Labs CP210x USB to UART Bridge (COM3)"* — note the COM number.

Step-by-step procedure

(Same on both operating systems — the window looks and works identically.)

1. **Make sure the USB cable is connected** between the Generator and the PC. This both powers the Generator and carries the data. If the Generator is instead running only from a separate power supply with no USB cable to a PC, downloading cannot work at all — there's no communication path.

2. Open a terminal (Linux) or Command Prompt (Windows) and start the tool:

   - Linux: `cd ~/Arduino/ESP32_Four_Channel_Generator_V2_02c` then `python3 protocol_tool_gui.py`
   - Windows: `cd path\to\ESP32_Four_Channel_Generator_V2_02c` then `python protocol_tool_gui.py`

3. In the window:

   - **Serial port:** pick the port you found above (click Refresh if it's not listed).
   - **Protocol file:** type or Browse to where you want the download saved, using the exact name `protocols-N.txt` — the number `N` (0–99) tells the Generator *which* file on its SD card to send. For example, type `protocols-7.txt` to pull file #7. **Note:** the Browse button only shows files that already exist. If this is the first time you're downloading that particular number, just type the filename directly into the box — any folder you like, the file will be created there.

4. Click **"Download from Generator."** A *"Confirm to Proceed with Download?"* popup appears — **leave it open, don't click Yes yet.**

5. On the Generator itself: press the **reset button** (the rotary encoder's built-in pushbutton), and press-and-hold the **black cursor button** either just before or just after pressing reset. What matters is that the black button is being held down during the brief moment — a second or two after reset, while the hardware is still initializing — when the Generator's startup code checks for it. Either order works, as long as it's held through that window; otherwise the Generator just boots normally instead of entering USB Comm mode.

6. Keep holding the black button: the OLED shows **"USB Comm."** and automatically cycles through three options roughly every quarter second — *Send To CPU*, *Receive Fm CPU*, *Return to Main* — with an arrow marking the current one.

7. **Release** the black button the instant **"-> Send To CPU"** is shown.

8. Press the **red button** once to confirm. The screen briefly shows *"Turn off / MiniCom / Start Py"* (a leftover reminder from early testing — safe to ignore if you're not running a separate serial terminal program), then the Generator starts waiting for the PC.

9. Back on the PC — **within about 15 seconds** — click **Yes** on the *"Confirm to Proceed with Download?"* popup.

10. Watch the **Activity Log** in the window: `TX:`/`RX:` lines scroll by as the file transfers, ending with *"Download succeeded."* The Generator's screen shows **"SEND GOOD"** — this exact wording is used for both directions (uploads and downloads alike); it just means the transfer succeeded.

11. Done — the requested `protocols-nn.txt` content is now saved on your PC, ready to open in a spreadsheet program. See the companion document, *"Sending Protocols Files From a PC to the Generator,"* for the file format and how to edit it.

If something goes wrong

- **"Timed out waiting for DOWNLOAD_READY"** — you didn't click Yes within the Generator's ~15-second wait window, or the black button wasn't held long enough at boot. Start again from step 5.
- **Serial error / "port busy" / "resource busy"** — another program (Arduino IDE's Serial Monitor, a terminal program, minicom, etc.) already has the port open. Close it and try again.
- **"ERROR:FILE_NOT_FOUND"** — the file number you typed doesn't exist on the Generator's SD card. Double-check the number.
- **Nothing in the port dropdown** — check the USB cable is fully plugged in on both ends, then click Refresh.

Part IV: Firmware

Chapter 7: Loading the Object File Directly Into the Generator (Firmware Update)

What this does: installs a new compiled program — the **object file**, `ESP32_Four_Channel_Generator_V2_02b.ino.merged.bin` — directly into the Generator's flash memory over the USB cable. This **replaces the program itself**, not a data file. It's a different kind of update from the other two How-To documents:

- The other two documents move **protocols-nn.txt data files** (frequency settings) back and forth over USB — they never touch the program itself.
- This document installs a **new program** onto the Generator — it does **not** touch the microSD card or any `protocols-nn.txt` files, which are left exactly as they were.

No Arduino IDE, no Python, and normally no extra software installation is needed — everything required is bundled in the `firmware_update_package` folder.

What you need

The whole `firmware_update_package` folder, containing:

- `ESP32_Four_Channel_Generator_V2_02b.ino.merged.bin` — the object file to be installed (this single file already contains the bootloader, the partition table, and the compiled program together).
- `install_update.sh` — the Linux installer.
- `install_update.bat` and `install_update.ps1` — the Windows installer (double-click the `.bat`; it launches the `.ps1` for you).
- `bin_linux_amd64/esptool` and `bin_windows_amd64/esptool.exe` — the flashing tool, bundled for each operating system so nothing else needs to be installed for this step.

Keep all of this together in one folder exactly as provided — the installer scripts look for the `.bin` file and the bundled `esptool` right next to themselves.

You'll also need a USB cable connecting the Generator to the PC.

Important cautions

- **No button-holding is needed for this.** Unlike the Send/Receive USB Comm menu used in the other two How-Tos, the Generator doesn't need to be put into any special mode by hand — both installer scripts reset the board into its flashing mode automatically over the USB connection.

- **Don't unplug the Generator or close the window while it's flashing** — it can take a couple of minutes. An interrupted flash can leave the Generator unable to run correctly until it's flashed again — it isn't permanently damaged (the ESP32's boot-loading circuitry lives in unerasable silicon), but don't count on the device being usable again until a full, successful flash completes.
- A firmware update does not touch or erase the microSD card.

Step-by-step: Linux

1. Make sure the `firmware_update_package` folder is on your PC with everything inside it intact.
2. Plug the Generator into the PC with a USB cable.
3. Open a terminal and `cd` into the `firmware_update_package` folder.
4. Run:

   ```
   ./install_update.sh
   ```

   (If you get "permission denied," run `chmod +x install_update.sh` once, then try again.)
5. The script automatically finds the Generator's USB port. If more than one serial device is plugged in, it lists them and asks you to pick the right one.
6. When it asks **"Continue? [y/N]"**, type `y` and press Enter.
7. Wait — progress messages scroll by; this can take a few minutes. Don't unplug the Generator or close the terminal.
8. On success you'll see **"SUCCESS – the update has been installed."** The Generator restarts on its own — nothing further to do.

Step-by-step: Windows

1. Make sure the `firmware_update_package` folder is on your PC with everything inside it intact.
2. Plug the Generator into the PC with a USB cable.
3. In the `firmware_update_package` folder, **double-click `install_update.bat`.**
   - Windows may show a blue "Windows protected your PC" SmartScreen warning the first time you run it. Click **"More info"**, then **"Run anyway."** This appears because the script isn't digitally signed, not because anything is wrong with it.
4. A console window opens and automatically finds the Generator's COM port. If it reports no device found, see Troubleshooting below.
5. When it asks **"Continue? [y/N]"**, type `y` and press Enter.

6. Wait — progress messages scroll by; this can take a few minutes. Don't unplug the Generator or close the window.
7. On success you'll see **"SUCCESS – the update has been installed."** Press Enter to close the window. The Generator restarts on its own — nothing further to do.

Troubleshooting (both operating systems)

- **"No Generator was found connected to this computer"** — check the USB cable is fully seated at both ends; try a different cable (some USB cables are charge-only and carry no data). On Windows, open **Device Manager → Ports (COM & LPT)** and look for an unrecognized device or a yellow warning icon — if present, install the matching USB driver (CP210x or CH340, depending on the board) and try again.
- **The script says esptool "isn't installed" / the bundled copy won't run** — unlikely on a normal 64-bit Windows or Linux PC. The message printed by the script explains the fallback: install `esptool` via `pip install esptool`, or your Linux distro's package manager.
- **"FAILED – the update did not install correctly"** — safe to just run the installer again from the start; nothing is left in a state that blocks a retry. If it keeps failing, try a different/shorter USB cable, or unplug and replug the Generator before trying again.
- **More than one device listed and you're not sure which is the Generator** — unplug the Generator, run the installer again to see which entry disappears from the list, then plug it back in and pick that one.

Part V: Open Items

Chapter 8: Status of Other Firmware Features

Two features exist in the firmware but are intentionally inactive, so no operating instructions are included for them in this manual:

- **WiFi remote start/stop** – the code is present (a simple web page with Start/Stop links), but `wifiEnable` is hard-coded off in this build for faster development iteration. No `wifidata.txt` file on the card will change that until the firmware is rebuilt with it turned back on.
- **Adjust Mode (Mode=4)** – was an unfinished feature, and was made deliberately unreachable in Versions 2.02c and 2.03 by clamping Mode to a maximum of 3. It has been reworked and is fully working as of Version 2.04: the Mode limit is now 4, Setup=2 lists it, and it gives real-time knob control of Channel 1. See "How Mode 4 (Adjust Mode) works" in Chapter 2.

Also present in the firmware but out of scope for this manual: `dutyFlip`, a compile-time setting (not adjustable from the device itself) for inverting a channel's output polarity for certain hardware variants – relevant to whoever builds the firmware from source, not to normal day-to-day operation.

Appendix A: Python Tool Source Code

The PC-side tools used to prepare, upload, and download protocol files. `protocol_tool_bkp.py` (an older backup copy of `protocol_tool.py`) is intentionally excluded here as it is not part of the active toolset – flag if that should be reconsidered.

A.1 protocol_editor.py

A.4 eeprom_download.py

A.5 eeprom_upload.py

A.6 eeprom_upload2.py

Appendix B: Generator Firmware Source Code

Appendix C: Architecture Map

A developer-oriented map of the firmware's module structure and internal function responsibilities – not needed for day-to-day operation, included here for anyone maintaining or extending the source.

> Documentation-ready architecture overview for the ESP32 Four Channel Generator project.

1. High-Level System Overview

The ESP32 Four Channel Generator is a **four-channel programmable PWM/square-wave generator** built around the ESP32 MCPWM hardware. It loads frequency, duty-cycle, timing, mode, sweep, and file-selection information from protocol files stored on a microSD card.

Runtime control is handled through a physical run/stop interrupt button, rotary encoder, cursor button, OLED display, optional WiFi start/stop control, and optional USB protocol-file transfer.

The system consists of five major subsystems:

1. **Core control firmware** — initializes hardware, manages run/stop state, coordinates setup screens, operating modes, timing, file loading, and saving.
2. **PWM generation engine** — uses ESP32 MCPWM register-level control to generate four PWM outputs.

3. **Protocol storage system** — reads `protocols-N.txt` files from microSD and loads them into EEPROM-emulated flash memory.
4. **User interface** — SSD1306 OLED display, rotary encoder, cursor/select button, and run/stop button.
5. **External communication** — optional WiFi web start/stop and USB serial protocol-file transfer.

2. Arduino IDE Tab / Module Map

The project is organized as an eight-tab Arduino IDE sketch. These tabs compile together as one firmware image, but each tab groups related functionality.

Arduino Tab / File	Primary Responsibility
`ESP32_Four_Channel_Generator_V2_02c.ino`	Main program file. Contains version notes, includes, pin definitions, display object, global variables, data structures, interrupt handler, WiFi server object, `setup()`, and `loop()`.
`PWM.ino`	Register-level ESP32 MCPWM driver. Calculates prescalers and timer periods, configures four PWM generators, and provides `setGen1()` through `setGen4()`, `startGen()`, `stopGen()`, `startAll()`, and `stopAll()`.
`support_Routines.ino`	Main operating logic. Handles frequency start/stop, SD-card file fetch coordination, EEPROM fetch/save, protocol mode, sweep mode, timing, suspend/resume, saving, and runtime updates.
`micro_SD.ino`	Low-level SD-card support. Provides directory/file utilities, reads protocol files, parses CSV fields, and stores parsed records into EEPROM-emulated flash.
`SSD1306.ino`	OLED display and cursor UI. Handles normal display, running display, suspended display, update display, screen saver, common frequency/duty display, and cursor movement.
`rotary_Decode.ino`	Rotary encoder decoding and parameter adjustment. Uses a quadrature transition table, updates selected parameters based on cursor position, and enforces limits.
`USB_Comm.ino`	USB serial communication for protocol-file upload/download. Allows a computer-side program to send or receive `protocols-`

Arduino Tab / File	Primary Responsibility
	`N.txt` files through the ESP32 USB serial port.
`WiFi.ino`	Optional WiFi web server. Provides simple browser-accessible start/stop control by setting or clearing `runFlag`.

3. Hardware Resource Map

Function	ESP32 Resource / Pin
PWM Channel 1 / Generator 1	GPIO 32, MCPWM0 Timer 0 Operator 0
PWM Channel 2 / Generator 2	GPIO 33, MCPWM0 Timer 1 Operator 1
PWM Channel 3 / Generator 3	GPIO 27, MCPWM1 Timer 0 Operator 0
PWM Channel 4 / Generator 4	GPIO 14, MCPWM1 Timer 1 Operator 1
Run/Stop interrupt button	GPIO 15
Cursor / select button	GPIO 4
Rotary encoder A	GPIO 16
Rotary encoder B	GPIO 17
Status / blink LED	GPIO 2
OLED display	SSD1306 I2C display at address `0x3C`
microSD card	Arduino `SD` library over SPI
WiFi server	ESP32 WiFi server on port `80`
USB communication	ESP32 serial port at `115200` baud

4. Main Program Structure

The main program file is the central coordinator. It defines the global state used by all modules.

The two most important functions are:

```
void setup();
void loop();
```

`setup()` performs one-time initialization. `loop()` continuously manages user input, display updates, run/stop control, timing, protocol changes, sweep execution, and optional WiFi requests.

Because this is an Arduino multi-tab sketch, the functions in the other tabs are available to the main program as if they were part of one large source file.

5. Startup Flow

The startup sequence begins in `setup()`.

1. **Initialize UI state** — sets the initial cursor position using `curX` and `curY`.
2. **Configure GPIO** — configures the blink LED, run/stop interrupt pin, cursor button, and rotary encoder pins.
3. **Attach interrupt** — attaches GPIO `15` to `handleInterrupt()`, which toggles `runFlag` with debounce protection.
4. **Initialize serial** — starts serial communication at `115200` baud.
5. **Enable MCPWM peripherals** — resets and clock-enables MCPWM0 and MCPWM1, configures timer routing, and attaches PWM GPIOs.
6. **Initialize EEPROM-emulated flash** — calls `EEPROM.begin(FLASH_SIZE)`.
7. **Initialize OLED** — starts the SSD1306 display at I2C address `0x3C`.
8. **Load default SD protocol file** — calls `fileFetch()`, with `fileNo` forced to `0`, causing `protocols-0.txt` to load.
9. **Optional USB communication mode** — if the cursor button is held during startup, enters `usbcomm()`.
10. **Optional WiFi startup** — if `wifiEnable == 1`, connects to WiFi and starts a web server on port `80`.

6. Default Protocol File Role

The file `protocols-0.txt` is the default startup file.

Example default file:

```
#
1,178.83,0.50,256.00,0.50,150.00,0.50,200.00,0.50
2,300.00,0.50,100.00,0.50,150.00,0.50,198.00,0.50
0,1,2,100,9,1,2,1,2, 1.00, 0
```

This file contains:

- a header marker line containing `#`,
- one or more frequency/duty records,
- and one final control record beginning with `0`.

The default startup file is loaded by `fileFetch()` during `setup()`.

7. Protocol File Format

The Generator uses files named:

```
protocols-0.txt
protocols-1.txt
protocols-2.txt
...
protocols-99.txt
```

7.1 Frequency Record Format

A normal frequency record begins with a nonzero protocol ID.

```
protoID,F1,D1,F2,D2,F3,D3,F4,D4
```

Example:

```
1,178.83,0.50,256.00,0.50,150.00,0.50,200.00,0.50
```

Field	Meaning
`protoID`	Frequency group / protocol row number
`F1`	Channel 1 frequency
`D1`	Channel 1 duty cycle
`F2`	Channel 2 frequency
`D2`	Channel 2 duty cycle
`F3`	Channel 3 frequency
`D3`	Channel 3 duty cycle
`F4`	Channel 4 frequency
`D4`	Channel 4 duty cycle

Duty values are stored as fractional values, where `0.50 = 50%` and `1.00 = 100%`.

7.2 Control Record Format

The final control record begins with `0`.

```
0,memGrp,modeID,endTime1,endTime2,startFreqGrpID,stopFreqGrpID,startSweepGrpID,stopSweepGrpID,sweepFreqInc,fileNo
```

Example:

```
0,1,2,100,9,1,2,1,2, 1.00, 0
```

Field	Meaning
`0`	Marks this as the control record
`memGrp`	Startup memory group / starting frequency group
`modeID`	Operating mode
`endTime1`	Frequency-set runtime, stored in file as whole seconds
`endTime2`	Program runtime, stored in file as whole minutes
`startFreqGrpID`	Protocol mode start group
`stopFreqGrpID`	Protocol mode stop group

Field	Meaning
`startSweepGrpID`	Sweep mode start group
`stopSweepGrpID`	Sweep mode stop group
`sweepFreqInc`	Sweep frequency increment
`fileNo`	Protocol file number

Internally, the firmware converts timing fields: `endTime1` from seconds to milliseconds and `endTime2` from minutes to milliseconds.

8. SD Card to EEPROM Working Image

The microSD file is not used directly during every runtime operation. Instead, it is loaded into EEPROM-emulated flash memory, and the firmware works from that loaded image.

```
microSD protocols-N.txt
        ↓
readFileModified()
        ↓
loadFloats() / loadIntegers()
        ↓
EEPROM-emulated flash memory
        ↓
fetchMem()
        ↓
global runtime variables
        ↓
fetchProtoID()
        ↓
active F1/D1/F2/D2/F3/D3/F4/D4 settings
```

Frequency records are stored using `EPromFreqObject`. Control records are stored using `EPromControlObject`.

9. Runtime Loop Architecture

The main `loop()` controls the firmware state machine.

```
loop()
│
├── if WiFi enabled:
│       wifiRoutine()
│
├── if runFlag == 1:
│       running / update behavior
│
└── else:
        stopped / edit behavior
```

`runFlag` is the main run/stop control flag. It can be changed by the physical run/stop button, WiFi commands, timeout logic, update-mode logic, or USB communication cleanup.

10. Stopped Mode

When `runFlag == 0`, the Generator is in edit/idle mode.

The firmware:

1. Turns off the output frequencies using `setFrequency(0)`.
2. Clears runtime clocks using `setClocks(0)`.
3. Displays the setup/edit screen using `Display()`.
4. Reads rotary encoder input using `rotary()`.
5. Reads cursor movement using `cursorMove()`.
6. If `protoID` changes, loads the new frequency group using `fetchProtoID(protoID)`.
7. Updates the OLED display when values change.
8. Starts the screen saver after the configured idle time.

11. Running Mode

When `runFlag == 1`, the Generator enters active operation.

On the first pass through running mode, controlled by `singleShot`, it:

1. Selects the proper starting protocol group depending on mode.
2. Calculates sweep parameters if needed.
3. Loads the active frequency group using `fetchProtoID()`.
4. Starts the PWM outputs using `setFrequency(1)`.
5. Displays the running screen.
6. Saves the current state using `saveCurrent()`.
7. Starts runtime clocks using `setClocks(1)`.

After that, it calls `checkTime()` for normal protocol/sweep timing, or `adjustMode()` — renamed `adjustModeRun()` as of Version 2.04 — when Mode 4 is active. Mode 4 was intentionally unreachable in Versions 2.02c and 2.03; it is reachable again in 2.04. Unlike the other modes it does not return to the main loop between events: adjustModeRun() runs its own loop until the red button's interrupt clears `runFlag`.

12. Operating Modes

Mode	Name	Description
1	Simple	Runs a selected frequency group.
2	Protocol	Steps through a range of frequency groups over time.
3	Sweep	Sweeps frequency based

Mode	Name	Description
		on start/stop groups and sweep parameters.
4	Adjust Mode	Real-time knob control of Channel 1's frequency and duty cycle while running. Unreachable in 2.02c and 2.03; implemented and working in Version 2.04.

13. Protocol Mode

Protocol mode is selected when `modeID == 2`.

In protocol mode:

1. The starting group is `startFreqGrpID`.
2. The stopping group is `stopFreqGrpID`.
3. At each frequency timeout, `protocol()` increments `protoID`.
4. If `protoID` exceeds `stopFreqGrpID`, it wraps back to `startFreqGrpID`.
5. The new group is loaded using `fetchProtoID(protoID)`.
6. The PWM outputs are updated using `setFrequency(1)`.

14. Sweep Mode

Sweep mode is selected when `modeID == 3`.

The sweep system uses:

```
startSweepGrpID
stopSweepGrpID
sweepFreqInc
sweepAccum
sweepAccumInc
sweepNumCnt
sweepShowID
```

The basic sweep flow is:

```
calcSweepParams()
        ↓
fetch start sweep group
        ↓
fetch stop sweep group
        ↓
calculate sweep increment and count
        ↓
sweep()
        ↓
```

```
update F1 using sweep accumulator
        ↓
setFrequency(1)
        ↓
repeat until sweep count expires
```

In the current sweep implementation, `F1` is the frequency directly modified by the sweep accumulator. F2 through F4, and all four duty cycles, are re-fetched from the start sweep group on every step and therefore hold steady for the whole sweep.

`sweepNumCnt` is the number of steps remaining in the current pass. `calcSweepParams()` sets it to `abs(sweepF2 - sweepF1) / sweepFreqInc` plus one, and derives `sweepAccumInc` as `(sweepF2 - sweepF1) / sweepNumCnt` — signed, so a descending sweep needs no special handling, and the final step lands exactly on the stop frequency. When `sweepFreqInc` is zero the step count comes from the ratio of `endTime2` to `endTime1` instead, giving a single pass spread across the program run.

`sweepShowID` exists purely for the display. Because a sweep never leaves the start group, `protoID` is not a meaningful thing to show while sweeping. `calcSweepParams()` initialises `sweepShowID` to `startSweepGrpID`; `sweep()` latches `stopSweepGrpID` into it for the single step on which `sweepNumCnt` was 1 on entry — the step whose F1 equals the limit frequency. The three `ID=` display sites in `SSD1306.ino` — Running, Suspend and Update — print `sweepShowID` when `modeID == 3` and `protoID` otherwise.

Prior to Version 2.02c these three sites printed `sweepNumCnt` directly, so the `ID=` field counted down through the remaining sweep steps rather than naming a record. Assignment of `sweepShowID` happens after the wrap-around call to `calcSweepParams()`, so the restart does not overwrite the endpoint indication before the display runs.

15. PWM Architecture

The PWM engine is implemented in `PWM.ino` using ESP32 MCPWM hardware.

```
MCPWM0
├── Gen 1: Timer 0, Operator 0, GPIO 32 — master
└── Gen 2: Timer 1, Operator 1, GPIO 33 — secondary / locked clock

MCPWM1
├── Gen 3: Timer 0, Operator 0, GPIO 27 — master
└── Gen 4: Timer 1, Operator 1, GPIO 14 — secondary / locked clock
```

Channels 1 and 3 are master channels. They calculate and set the MCPWM clock prescaler, timer prescaler, timer period, and compare value. The master optimizer is `findBestPrescalers()`.

Channels 2 and 4 are secondary channels. They inherit the MCPWM unit clock prescaler from their corresponding master channel and use `findBestWithLockedClock()` to calculate the best timer prescaler and period.

This design improves synchronization within each channel pair, but channels 2 and 4 are most accurate when their requested frequencies are reasonably close to their corresponding master channels.

16. PWM Output Update Path

```
F1/D1/F2/D2/F3/D3/F4/D4
        ↓
setFrequency(1)
        ↓
stopAll()
        ↓
setGen1(F1, converted D1)
setGen2(F2, converted D2)
setGen3(F3, converted D3)
setGen4(F4, converted D4)
        ↓
MCPWM hardware runs in background
```

When turning outputs on, duty is intentionally inverted before being sent to the MCPWM engine:

```
setGen1(F1, 100 - (D1 * 100));
setGen2(F2, 100 - (D2 * 100));
setGen3(F3, 100 - (D3 * 100));
setGen4(F4, 100 - (D4 * 100));
```

17. User Interface Architecture

The user interface is built around the SSD1306 OLED, rotary encoder, cursor/select button, and run/stop button.

The primary display function is `Display()`, which changes what is shown based on `setupOpt`.

`setupOpt`	Screen / Purpose
`0`	Normal frequency read/display mode
`1`	Frequency and duty setup mode
`2`	Mode selection screen
`3`	Protocol range setup
`4`	Sweep range and sweep increment setup
`5`	Timing and protocol-file selection setup

Additional display functions include `displayRunning()`, `displaySuspended()`, `displayUpdate()`, `displayCommon()`, and `displayScreenSaver()`.

18. Cursor and Rotary Encoder System

The cursor position is stored in `curX` and `curY`. The cursor button advances through editable fields using `cursorMove()`.

The rotary encoder is decoded in `rotary_Decode.ino` using a quadrature transition lookup table. After a valid step:

- clockwise movement calls `rotaryPositive()`,
- counterclockwise movement calls `rotaryNegative()`,
- and limits are applied using `rotaryLimits()`.

Editable parameters include `setupOpt`, `protoID`, `modeID`, `F1` through `F4`, `D1` through `D4`, `endTime1`, `endTime2`, protocol range values, sweep range values, `sweepFreqInc`, and `fileNo`.

19. Limit Enforcement

The function `rotaryLimits()` enforces operating limits after rotary changes.

Parameter	Limit Behavior
`setupOpt`	Clamped to available setup screens
`protoID`	Wrapped between `1` and `protoIDlast`
`modeID`	Limited to `1` through `4`
`F1`–`F4`	Limited approximately from `0.04` to `65535.00`
`D1`–`D4`	Limited from `0.00` to `1.00`
`endTime1`	Stored internally in milliseconds and aligned to whole seconds above 1 second
`endTime2`	Stored internally in milliseconds and aligned to whole minutes above 1 minute
`sweepFreqInc`	Wrapped between `0.00` and `9999.99`
`fileNo`	Wrapped across `0` through `99`

20. Runtime Timing

The timing system uses `millis()` and these primary variables:

```
endTime1
endTime2
runTime1
runTime2
runTime
```

`endTime1` is the runtime for an individual frequency set. `endTime2` is the total program runtime.

`setClocks(1)` sets active timeout values:

```
runTime1 = endTime1 + millis();
runTime2 = endTime2 + millis();
```

`checkTime()` monitors whether the full program time has expired, the current frequency-set time has expired, the cursor button has requested suspend/resume, protocol mode should advance, or sweep mode should advance.

21. Suspend / Resume Behavior

During running mode, pressing the cursor button suspends the Generator.

The suspend logic:

1. Displays the suspended screen.
2. Saves the remaining program time and frequency-set time.
3. Turns off the PWM outputs using `setFrequency(0)`.
4. Waits through the suspend/resume button sequence.
5. Restores the running display.
6. Restarts the PWM outputs using `setFrequency(1)`.
7. Restores the remaining runtime values.

22. File Selection and Default File Update Behavior

The Generator uses `fileNo` to select a protocol file. The file name is generated with:

```
sprintf(fileName, "/protocols-%d.txt", fileNo);
```

This allows file numbers beyond a single digit.

The UI allows protocol file selection on setup screen 5. When a nonzero preset file is loaded and confirmed, the code contains logic to make that preset become the new default by writing the loaded data back to `protocols-0.txt`.

This means `protocols-0.txt` acts as the live/default startup file, while other `protocols-N.txt` files act as selectable presets.

23. Saving to microSD

The function `saveFile()` writes the current EEPROM working image back to the selected SD file.

The save process:

1. Saves the current `protoID`.
2. Builds the target file name from `fileNoHold`.
3. Removes the existing file first.
4. Writes a header line.
5. Iterates through all known protocol records.

6. Writes each frequency/duty record.
7. Writes the final control record.
8. Restores the previous active protocol ID.
9. Blinks the LED as feedback.
10. Displays the running screen.

24. USB Communication Architecture

USB communication is handled by `usbcomm()`. This mode is entered during startup if the cursor button is held.

The USB menu allows:

1. Send protocol file to CPU.
2. Receive protocol file from CPU.
3. Return to main program.

`downloadProgramOverUsb()` sends a requested `protocols-N.txt` file from the SD card to the computer over serial.

`uploadProgramOverUsb()` receives a protocol file from the computer, validates it, writes it to `/upload.tmp`, then replaces the target `protocols-N.txt` file. If `N == 0`, it immediately reloads `protocols-0.txt` using `fileFetch()`.

25. WiFi Architecture

WiFi support is optional and controlled by `wifiEnable`.

If enabled, the firmware connects to WiFi during `setup()` and starts a server on port `80`.

The function `wifiRoutine()` checks for incoming clients and serves a simple HTML page with two links:

```
/H = Start the Generator
/L = Stop the Generator
```

The request handlers are simple:

```
GET /H → runFlag = 1
GET /L → runFlag = 0
```

26. Major Global State Variables

Run / Mode / UI State

Variable	Purpose
`runFlag`	Main run/stop flag
`singleShot`	Allows one-time execution when entering run or stop states
`setupOpt`	Current setup/display screen

Variable	Purpose
`modeID`	Current operating mode
`protoID`	Current frequency group ID
`protoIDlast`	Last available protocol group loaded from file
`curX`, `curY`	Current OLED cursor position
`change`, `change2`	Display/update flags after value or cursor changes
`fileNo`	Currently requested protocol file number
`fileNoHold`	Saved/active protocol file reference used during load/save

Frequency / Duty State

Variable	Purpose
`F1`, `F2`, `F3`, `F4`	Active channel frequencies
`D1`, `D2`, `D3`, `D4`	Active channel duty-cycle values
`gen1_Frequency` etc.	Saved runtime PWM frequency settings
`gen1_DutyCycle` etc.	Saved runtime PWM duty settings
`genRunning[5]`	PWM running state array for channels 1–4

Timing State

Variable	Purpose
`endTime1`	Frequency-set runtime
`endTime2`	Program runtime
`runTime1`	Next frequency-set timeout
`runTime2`	Program timeout
`saveRunTime`	Saved remaining frequency time during suspend
`savePgmTime`	Saved remaining program time during suspend
`saverTime`	Screen saver timeout
`saverStartTime`	Idle time before screen saver starts

Protocol / Sweep State

Variable	Purpose
`memGrp`	Startup group from control record
`startFreqGrpID`	Protocol start group
`stopFreqGrpID`	Protocol stop group
`startSweepGrpID`	Sweep start group

Variable	Purpose
`stopSweepGrpID`	Sweep stop group
`sweepFreqInc`	Requested sweep increment
`sweepAccum`	Current sweep accumulator
`sweepAccumInc`	Calculated sweep step
`sweepNumCnt`	Remaining sweep count
`sweepShowID`	Group ID shown as ID= while sweeping

27. Overall Data Flow

```
microSD protocol file
        ↓
readFileModified()
        ↓
EEPROM-emulated flash image
        ↓
fetchMem()
        ↓
global control variables
        ↓
fetchProtoID()
        ↓
F1/D1/F2/D2/F3/D3/F4/D4
        ↓
setFrequency()
        ↓
setGen1() / setGen2() / setGen3() / setGen4()
        ↓
ESP32 MCPWM hardware
        ↓
four output channels
```

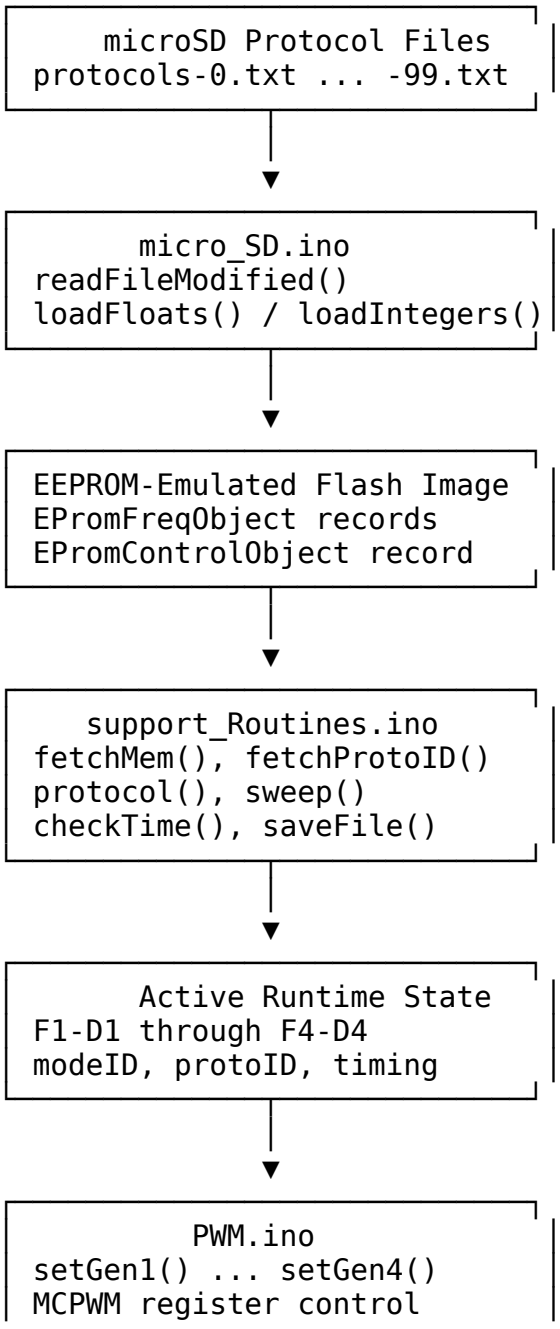
User input can modify global variables before they are saved or applied:

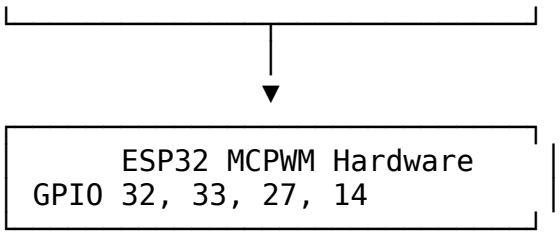
```
rotary encoder + cursor button
        ↓
rotaryPositive() / rotaryNegative()
        ↓
rotaryLimits()
        ↓
Display()
        ↓
saveCurrent() / saveFile()
```

External communication can also modify state:

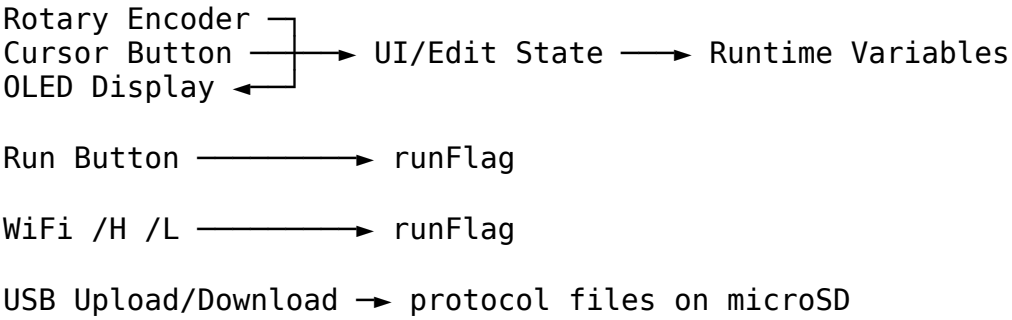
```
WiFi request
        ↓
runFlag

USB upload
        ↓
protocols-N.txt on SD card
        ↓
fileFetch() if protocols-0.txt was changed
```

28. Conceptual Block Diagram





The user interface and communication modules interact with the central runtime state:



29. Version 2.05 Architectural Summary

Version 2.05 is best described as a **microSD-programmable, four-channel ESP32 MCPWM generator** with a local OLED/rotary user interface and optional external control paths.

Its central design pattern is:

```
Protocol file → EEPROM working image → runtime variables → MCPWM hardware
```

The project is modularized by function rather than by C++ classes or libraries. This fits the Arduino IDE tab model and keeps the original large program manageable.

The most significant recent architectural change is the move to a **register-level MCPWM PWM engine**, which gives more direct control over frequency generation and enables fractional-frequency support on master channels. Channels 1 and 3 serve as master PWM channels, while channels 2 and 4 are secondary channels locked to the corresponding master MCPWM clock prescaler.

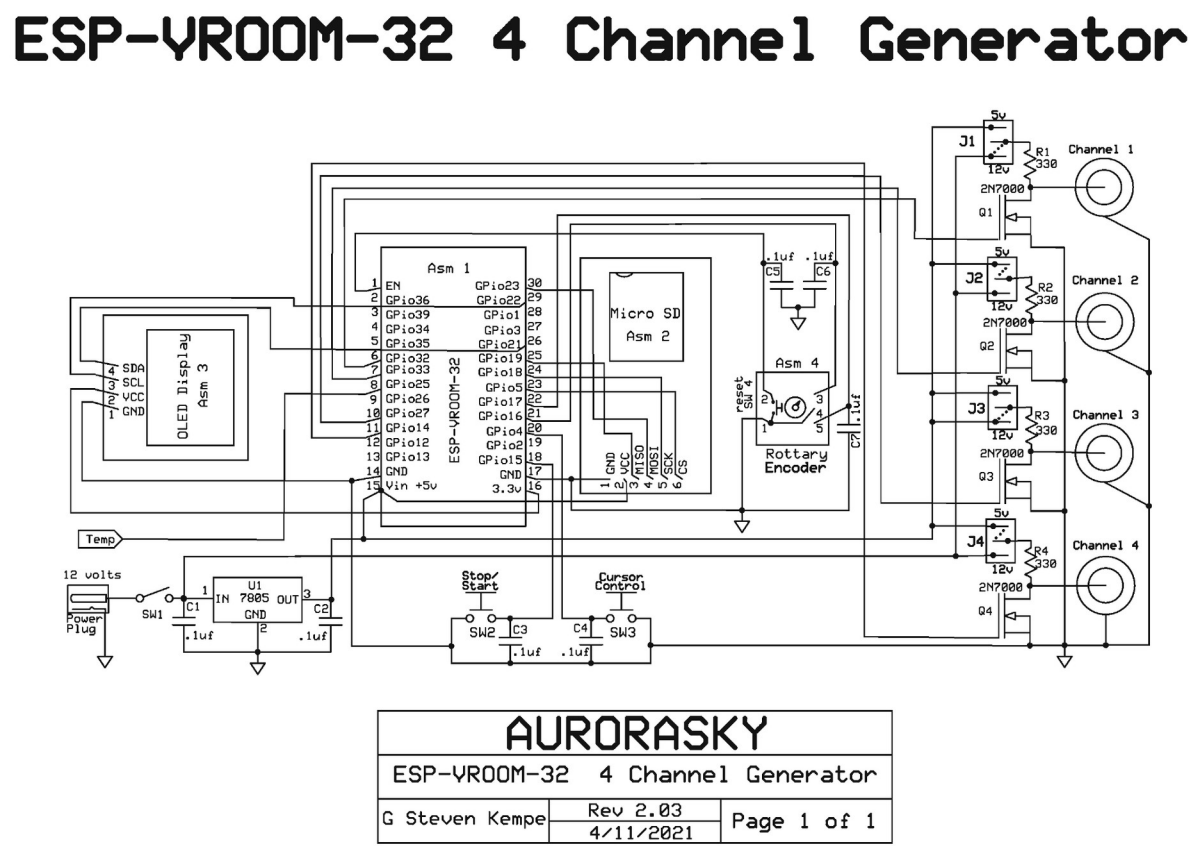
The current project also includes a more robust rotary encoder decoder, improved SD-file handling, default-file behavior through `protocols-0.txt`, and USB serial transfer support for editing protocol files without physically removing the microSD card.

In short, the Generator is no longer just a fixed four-channel signal generator. It is now a **field-programmable protocol-driven waveform controller** with local editing, persistent SD-card storage, optional WiFi start/stop, and USB-based protocol-file maintenance.

Appendix D: Schematic and Board Layout

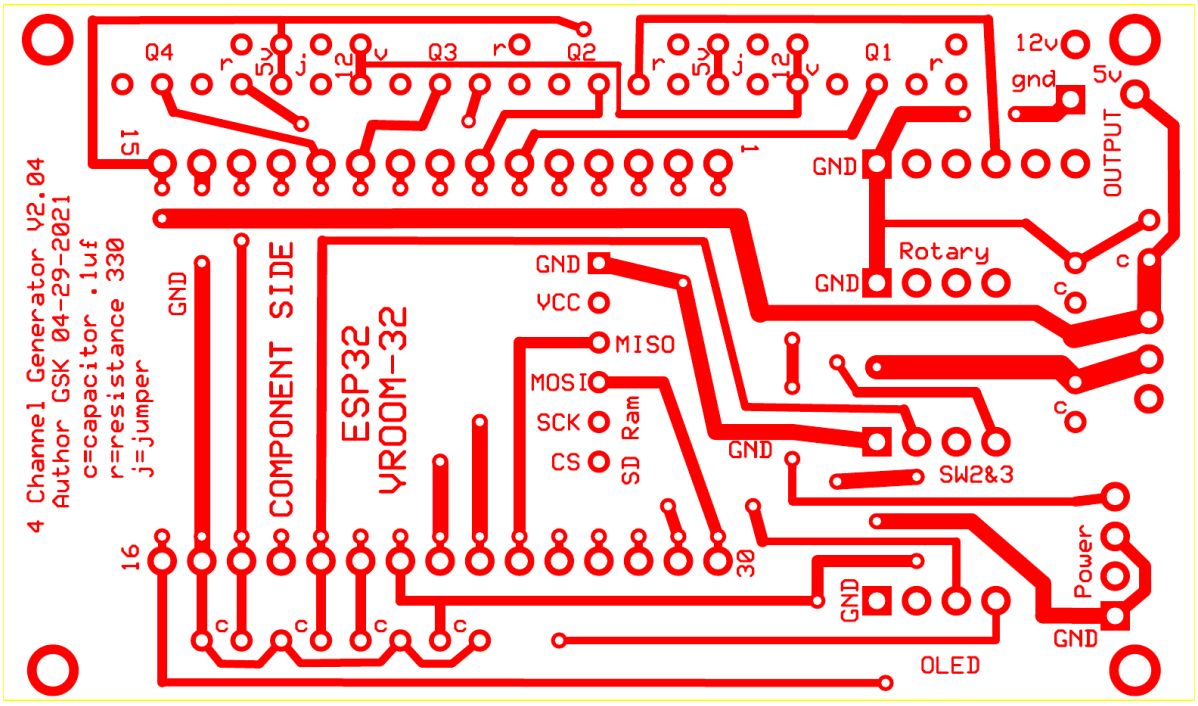
The circuit and board drawings for the ESP-VROOM-32 4 Channel Generator, included for anyone building, repairing or modifying a unit. These are reference material — nothing here is needed to operate the Generator.

Schematic



ESP-VROOM-32 4 Channel Generator — schematic, Rev 2.03.

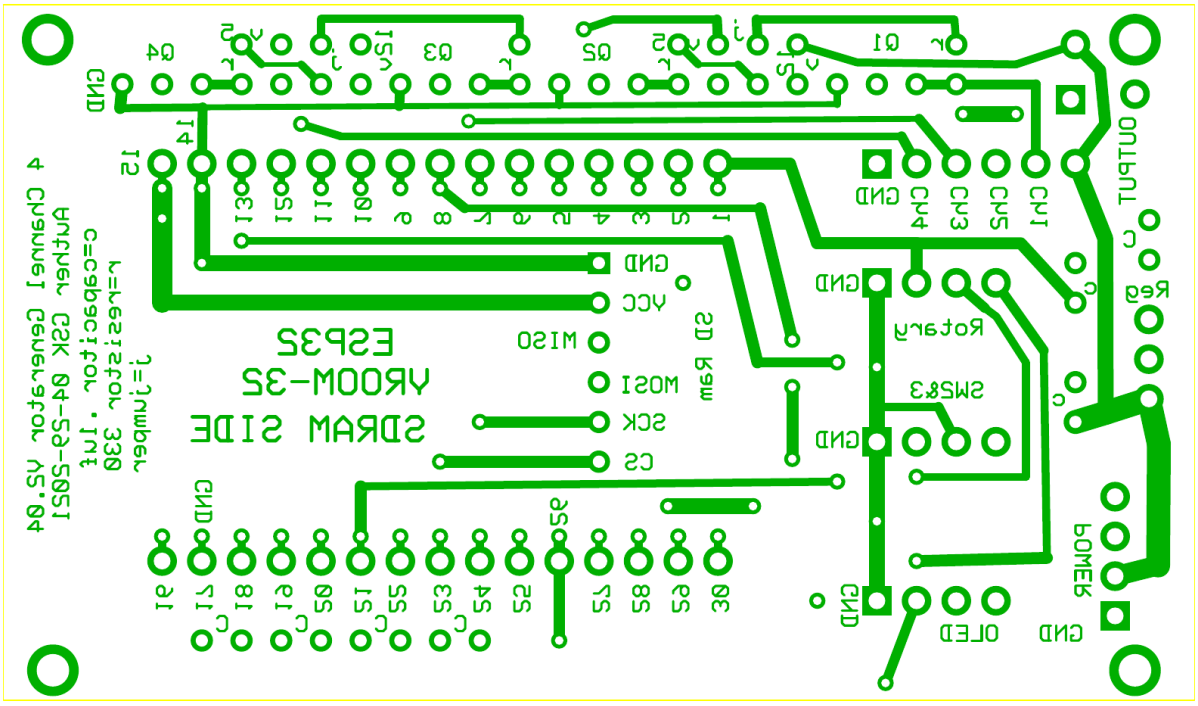
PCB — top



D:\ExpressPCB\ESP VROOM 32 4 Channel Generator PCB 2_04.pcb (Top layer)

Board layout, top side, revision 2.04.

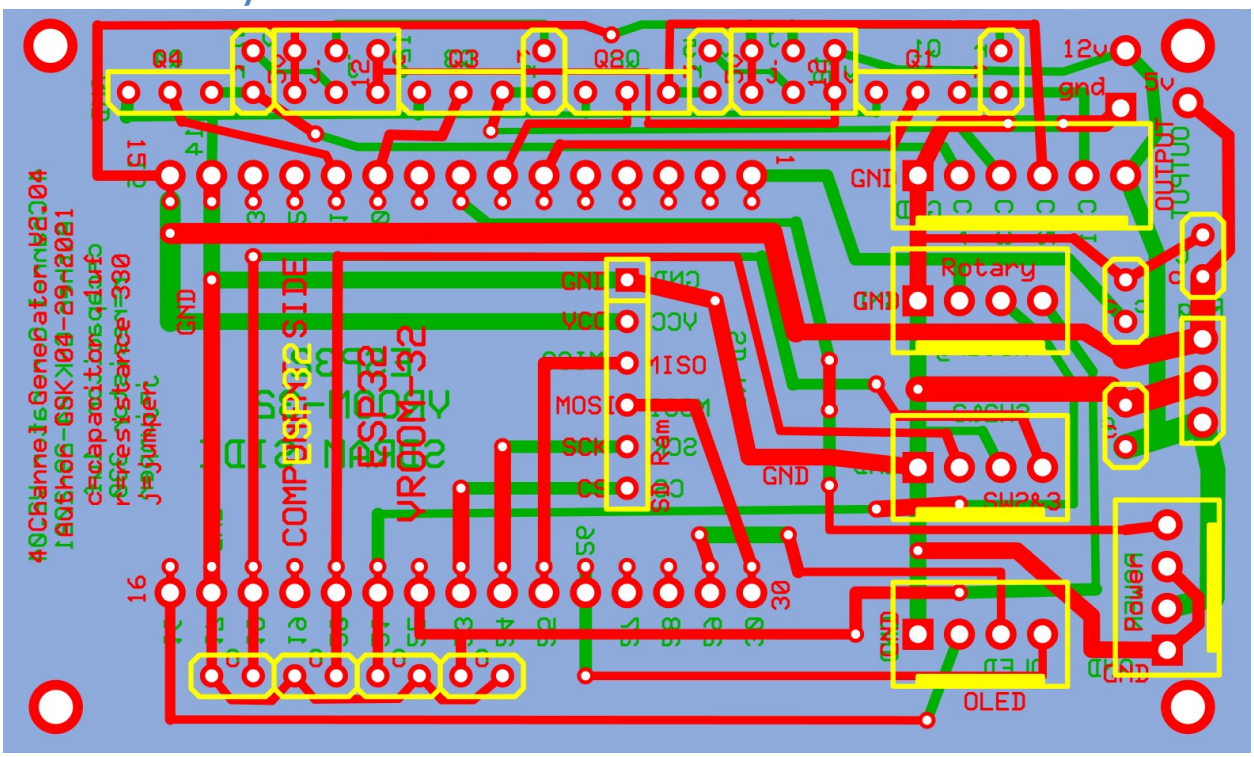
PCB — bottom



D:\ExpressPCB\ESP VROOM 32 4 Channel Generator PCB 2_04.pcb (Bottom layer)

Board layout, bottom side, revision 2.04.

PCB — assembly view



Board assembly view, revision 2.04.