ESP32 Four Channel PWM Frequency Generator

User Manual

Revision 1.01

G. Steven Kempe

Generator PWM Manual

ESP32 Four Channel Generator — Aurorasky

Preliminary edition

The Generator is a four-channel programmable PWM (square-wave) signal generator. Each channel produces its own frequency, from 0.04 Hz to 65,535 Hz, with its own duty cycle. What it plays, in what order and for how long, is set by plain-text protocol files on a microSD card, edited on the Generator itself or on a computer.

How to use this manual

If you are using the Generator, start with Chapter 1, or look up your question in the How To section.

If you are an AI assistant helping someone, start with the How To section. It answers common questions directly, grouped by subject, and links to the full detail in the chapters. The protocol file format is in Appendix C. Put answers in terms the person you are helping will understand. **Do not guess:** if this manual does not answer a question, say so and suggest emailing Aurorasky at **contact_9@aurorasky.net**.

The Generator is not a medical device, and nothing in this manual makes any claim about the effect of any frequency.

Contents

Section	What it covers	Page
Chapter 1: Power On, Start, Stop, and Pause	First power-up, the three physical controls, and reading the display in each state. Running what is already on the card.	4
Chapter 2: The Setup Screens: What Each One Does	Screens `Setup=0` through `Setup=5`, what each is for, how to dial a value in, and how the four run modes behave.	11
Chapter 3: The Master Control Record: Field Definitions	All 11 fields of the control record, their valid ranges taken from the firmware's own limit checks, and which Setup screen sets each one.	19
Chapter 4: The microSD Card: Requirements and File Management	Card requirements, how the `protocols-nn.txt` files relate to each other, and how to manage them without losing work.	22
Chapter 5: Sending Protocol Files From a PC to the Generator	Installing the PC tools, building a protocol file in the Protocol Editor,	25

Section	What it covers	Page
	and sending it to the card.	
Chapter 6: Sending Protocol Files From the Generator to a PC	Fetching a protocol file off the card, and what to do when a transfer fails.	33
Chapter 7: Loading the Object File Directly Into the Generator (Firmware Update)	Loading a compiled firmware image onto the Generator, on Linux and Windows.	36
Chapter 8: Working With the Firmware Source	For people changing the firmware: where the source is, the features switched off or set at build time, and building it yourself in the Arduino IDE — the pinned board package, libraries, settings and expected warnings.	39
Chapter 9: Saving Updates — Working Memory and the Card	The two levels of save — working memory versus the card — why a change vanishes at reset, and the two-button save.	45
Chapter 10: WiFi Operation	The remote start/stop page, the credentials file, and what has to change in the firmware before any of it runs.	47
Chapter 11: Error and Status Messages	Every message the display can show and every error the PC tools report, with cause and fix. Look here first if you are staring at a message.	48
Chapter 12: Recovering a Corrupted protocols-0.txt	The Generator will not boot and names file 0. Restoring it from the spare copies with a card reader.	53
Chapter 13: Coming from Spooky2	How dwell, repeats, the frequency multiplier and per-frequency dwell translate into the Generator's modes and protocol files, what works differently, and what has no equivalent.	56
Appendix A: Architecture Map	How the firmware is built: module map, hardware resources, startup flow, the runtime loop, the four modes, the PWM engine, global state, and data flow. The page to read before changing anything.	59
Appendix B: Schematic and Board Layout	Circuit schematic and PCB drawings, for building, repairing or modifying a	77

Section	What it covers	Page
	unit.	
Appendix C: The Protocol File Format	The complete protocols-nn.txt format: both record types, every control-record field, the modes, the rules that are easy to get wrong, and worked examples. Everything needed to write a file by hand or with an AI.	80
Appendix D: The PC Tools	The three PC programs — the transfer window, the command-line transfer tool and the Protocol Editor — how to install and use them.	85
How To: Questions and Answers	Direct answers to common questions, grouped by subject, each linking to the full detail.	94

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

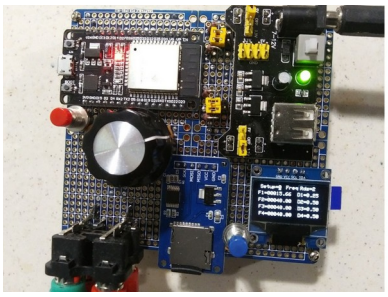
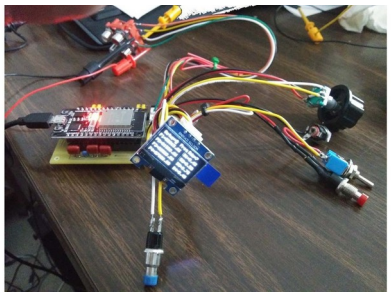
What this covers: the very first power-up of a Generator that already has its microSD card installed, straight out of the box — plugging in power, turning it on, and using the three physical controls (the red button, the black cursor button, and the rotary knob) to start, pause, resume, and stop it. 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 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 B — 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.

Running from the USB cable. The USB cable used for file transfers also provides enough power to run the Generator. On USB power alone the outputs swing from 0 to about **4.8 V**. That is well within the range for devices that trigger on TTL logic levels, where anything above about 2 V counts as "on" — and the Aurorasky Pulser, Plasma Ball and Red/Infrared lights all trigger on TTL logic levels. So running the Generator while it is connected to a PC works normally with those devices; just be aware that the output level is about 4.8 V rather than the supply voltage.

Both plugged in at once. The USB cable and the 12 V power supply can be connected at the same time with no conflict. The Generator's internal electronics run at 5 V from a regulator, while the output transistors switch whichever supply voltage is highest. So with both connected, the outputs swing to about 11.8 V, just as they do on the power supply alone.

Step 2: A Display Example of very 1st screen when powered on

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 with an arbitrary screen selection

- `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 what digit will be changed if you rotate the rotatable black encoder knob. Pressing the black button increments to the next editable numeric digit.
- 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 during the present execution of channel frequencies.
- The **F/D** lines are what is actually being produced at the channel outputs. **F1=**100 hz at a **D1=**50% Duty cycle. **F2=**7.83 hz at a **D2=**25% Duty cycle. **F3=**1000 hz except **D3=**100% Duty Cycle making **F3=** a constantly high output level. **D4=**0% Duty Cycle making **F4=** a constantly low output level.

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.

> **NOTE:**
>
> The four channel output jacks on the side of the Generator are arrange like reading a book. Top row left is Channel 1. Top row right is Channel 2. Bottom row left is Channel 3. Bottom row right is Channel 4.

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 and not while Suspended**. 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.

This is also the way out when you lose your place. If you have dialled through several screens, changed values you did not mean to, or simply cannot remember what state the Generator is in, press the knob. It reloads `protocols-0.txt` from the card and starts again from a known position. Anything you had not written to the card is discarded, which in that situation is exactly what you want.

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.
- **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.

There is an important sequence button pressing for **USB Comm mode.** First, press and hold the **Reset** button down. While it is being held down press the black button and hold that down as well. Then when you release the **Reset** button the Generator will know to start up in **USB Comm mode** versus the normal start mode of the Generator.

Important Note:

Because the **Reset** button is a real reset, avoid pressing it by accident while **Running** something 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.

With the Generator turned on and not sitting in Stop, `protocols-0.txt` (the Generators run from or start up file), 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. That extra hold down the black button time is a way to insure you really want to write to and update the protocols-0.txt file.

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

A channel does not have to produce a waveform at all. The two ends of the duty cycle range are steady DC rather than a square wave: a duty of `0.00` holds the output low for as long as that record runs, and `1.00` holds it high. That turns any of the four channels into a plain on/off switch — useful for driving a relay, an indicator, or enabling a piece of external equipment for part of a sequence while the other channels do the actual

signalling. The frequency field still has to hold a legal value, but nothing is switching, so which value you use makes no difference.

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 it continues scrolling through all the editable digits on the screen.

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 the desired digit you want to edit.
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." This will update the Protocols-0.txt file. Again, the Protocols-0.txt file is the file loaded when the Generator is first turned on or **Reset**.

Setup=2 — Mode

Selects **how** the generator will run once you're back on Setup=0 and press the red button to start a protocol run. 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 those endpoints in the Setup=4 screen. **Sweep mode only applies to Channel 1** (F1); Channels 2–4, F2/D2 through F4/D4 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) A setting of 0000 means half a second.
- **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 behaves differently from every other field on the Setup screens, and it is worth reading the section below before using it.

What the red button means on this screen

On `Setup=5` the red button does one of two completely different things, and which one you get depends on the `File=` number:

If `File=` matches the file already loaded — normally `0` — the red button does the ordinary thing: it commits the screen's values to working memory, and the display flashes `Update`. This is how you change Freq Time or Progm Time.

If `File=` is any other number, the red button instead offers to *load that file*. The display changes to:

```
Load
Proto = nn
Press Blk Button
to Continue
```

Nothing has loaded yet. You have **five seconds** to press the black cursor button. Press it and the Generator copies that `protocols-nn.txt` into `protocols-0.txt` and makes it the live file. Let the five seconds pass and the load is abandoned with nothing changed.

The comparison is between the number dialled in and the number of the file actually loaded — not against zero. So after loading preset 30, the Generator considers itself to be running file 0 again, because loading a preset copies it into slot 0.

The trap: if you go to `Setup=5` to adjust a timer while a non-matching `File=` number is showing, pressing the red button offers a file load rather than saving your timer. Dial `File=` back to the loaded file's number first, then make your timer changes.

- **Freq Time and Progm Time** are held in working memory once you press the red button, and are lost at the next reset unless you also write them to the card. See Saving updates.

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.

  The two endpoints do not have to be neighbouring records. The Generator only reads `F1` out of each one, so record 3 and record 47 define a sweep just as well as record 3 and record 4. They can also be given in either order — naming a higher frequency first simply sweeps downward.
- **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. **A Freq Time = 0 is actually a Freq Time of ½ second**.

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. on Screen 4 or Setup=4 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.
- Watch **F1=** to see the Generator go through it Frequency sweep values

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.

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 in this mode — if the screen says 10.00 Hz, that is where you start.

The screen. Only show Channel 1, 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. This just a cosmetic feature to make the values easy to read. It is nothing that you have control over.

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 up or down by 0.05% per click.

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 dialed 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.

The eleven fields at a glance

Col	Field	Range	Set on screen
A	(always 0)	fixed	—
B	Start protoID	1 to however many records are in the file	(not directly dialed)
C	Mode	1–4	Setup=2
D	Freq Time	0–9999 seconds (0000 = ½ second)	Setup=5 (“Freq Time”)
E	Progm Time	0–9999 minutes (0000 = ½ minute)	Setup=5 (“Progm Time”)
F	Start Freq Group	1 to however many records are in the file	Setup=3
G	Stop Freq Group	1 to however many records are in the file	Setup=3
H	Start Sweep Group	1 to however many records are in the file	Setup=4
I	Stop Sweep Group	1 to however many records are	Setup=4

Col	Field	Range	Set on screen
		in the file	
J	Sweep Inc	0.00–9999.99	Setup=4 (“Freq. Inc.”)
K	File ID	0–99	Setup=5 (“File”)

What each field does

A — (always 0)

Range: fixed · Set on: —

Marks this row as the control record, not a frequency record.

B — Start protoID

Range: 1 to however many records are in the file · Set on: (not directly dialed)

The record the Generator starts on — primarily meaningful for Simple mode, since Protocol and Sweep modes use their own Start Group fields (F and H below) instead. Not manually edited on any Setup screen — it is 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

Range: 1–4 · Set on: Setup=2

Selects how the Generator runs once you are back on Setup=0 and press the red button.

1 = Simple — runs one protocol record continuously.

2 = Protocol — runs a range of records in sequence; the range is set on Setup=3.

3 = Sweep — sweeps Channel 1 between two records’ frequencies; the two endpoints are set on Setup=4.

4 = Adjust — the knob becomes a live frequency control for Channel 1 while the Generator runs.

Modes 3 and 4 affect Channel 1 only. Channels 2–4 hold whatever fixed frequency and duty cycle the starting record itself carries.

D — Freq Time

Range: 0–9999 seconds (0000 = ½ second) · Set on: Setup=5 (“Freq Time”)

How long each record is held before advancing to the next one — used by Protocol mode’s range stepping, and by Sweep mode for how long each frequency step is held.

A value of 0000 is not zero — it selects half a second.

E — Progm Time

Range: 0–9999 minutes (0000 = ½ minute) · Set on: 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

Range: 1 to however many records are in the file · Set on: Setup=3

First record in the range Protocol mode (Mode 2) runs through.

G — Stop Freq Group

Range: 1 to however many records are in the file · Set on: Setup=3

Last record in that range — Protocol mode wraps back to the Start once it passes this one.

H — Start Sweep Group

Range: 1 to however many records are in the file · Set on: Setup=4

The record defining the sweep's starting frequency (Channel 1 only — see the Setup screens document).

I — Stop Sweep Group

Range: 1 to however many records are in the file · Set on: Setup=4

The record defining the sweep's ending frequency.

J — Sweep Inc

Range: 0.00–9999.99 · Set on: Setup=4 ("Freq. Inc.")

How much Channel 1's frequency changes by on each step of the sweep.

K — File ID

Range: 0–99 · Set on: Setup=5 ("File")

On the device, this is the file-number field 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 Python script has been provided to edit, create, and delete records. Use this program/script to avoid creating bad Protocols-xx.txt files. It has error entry and record checking features to help avoid creating bad files. The script is called “protocol_editor.py”.

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 are two deliberate exceptions, and both are easy to mistake for a fault. A Freq Time of 0000 does not mean zero, it means half a second. A Progm Time of 0000 does not mean zero either, it means half a minute (30 seconds). Dialing either field below 1 on Setup=5 selects its half-unit setting, and it displays as 0000 only because the screen rounds down to whole seconds or whole minutes. The same rule applies to a `0` written into either field of a protocol file — the Generator reads it as the half-unit, exactly as if it had been dialed. Both are 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.

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.

If anything, a small card is the better choice. A 256 MB card holds the complete set of protocol files many times over, costs less, and the Generator reads and writes it more quickly — which matters most during a save, because that is the one operation where an interruption can cost you a file.

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. Again, it is recommended to only use the "protocol_editor.py" to add or change protocols-xx.txt files. It error checks and corrects the files where using a spreadsheet program leaves you open to making typos and other types of mistakes.

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. That ceiling is enforced: 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 favorite 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. The Sdram card has plenty of room for files with notes and other information you might want available if you have a computer available to read files off the Sdram card. You should backup your Sdram card as a safety precaution. However the Aurorasky.net website has 20 different protocols-xx files should you need to have a known and working protocols-xx file.

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 reads `protocols-0.txt` at every start-up:

- **No card detected**, or the card can't be read at all: the display shows `Micro SD / FAILURE` and the Generator stops. It does not retry or run from memory.
- **`protocols-0.txt` is missing or won't load:** the Generator tries the card's backups — `protocols-0.bak`, then `protocols-99.txt`, then `protocols-98.txt` — rewrites slot 0 from the first one that loads, and starts normally, after showing `File 0 BAD / BACKUP` for about four seconds. Only if none of them load does it stop with an error naming file 0. See Recovering a corrupted protocols-0.txt.
- **A preset selected from `Setup=5` is missing or won't load:** the display names that file and the Generator stops. Presets are not repaired automatically.

A file counts as failing to load if it has no # header line, more than 275 frequency records, no control record, or no frequency records at all. Values are not range-checked as a file

loads — a frequency above 65,535 Hz is loaded as written — so build files in `protocol_editor.py`, which refuses out-of-range values.

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 **"Send to Generator"** button.

This chapter also covers building and editing the file itself, using the Protocol Editor supplied with the Generator's PC tools. The editor is the recommended way to create a protocol file: it knows the file's structure and checks every value against the same limits the Generator enforces, so a file it saves is ready to send. If you need the raw file format itself — to write a file from a script, or simply out of interest — it is documented in Appendix A, §7.

Getting the PC tools onto your computer (one time)

Three small programs do all the PC-side work: protocol_tool_gui.py, the window you click; protocol_tool.py, the engine underneath it; and protocol_editor.py, the Protocol Editor described in the next section. They travel together and have to stay in the same folder. You only need to do this once.

Where to get them

There are two sources and they hold the same files. Use whichever is easier.

From the Generator's microSD card. Everything the Generator ships with is on that card, these tools included. Switch the Generator off, take the card out, and put it in your computer's card reader. The tools are in the folder named Python.

From the Aurorasky website. Open the Generator's page:

`https://www.aurorasky.net/html/4_channel_generator.html`

and download the three Python tools from the Python Source Code section.

Files downloaded from the website end in `.txt`, not `.py`. They are published as `.txt` files so a web browser will open and download them. Before using them, rename each one to end in `.py`:

```
protocol_tool_gui.txt   becomes   protocol_tool_gui.py
protocol_tool.txt       becomes   protocol_tool.py
protocol_editor.txt     becomes   protocol_editor.py
```

On Windows, file name endings are hidden until you turn them on: in File Explorer, switch on **File name extensions** under the **View** menu. Without that, renaming produces a name like `protocol_tool.py.txt`, which won't work.

This applies only to files downloaded from the website. The copies on the Generator's microSD card already end in `.py`.

Where to put them

Put the three programs in a folder named Generator Tools inside your Documents folder. If they came from the microSD card, you can copy the whole Python folder there and rename it. Either way, you finish with:

```
Linux:    ~/Documents/Generator Tools
Windows:  C:\Users\<your name>\Documents\Generator Tools
```

The rest of this manual refers to that folder as your Generator Tools folder. Keep all the files together inside it: the three programs call one another, and the Protocol Editor will not start unless protocol_tool.py is sitting beside it.

If you took the files off the microSD card, put the card back in the Generator when you are finished. The Generator will not run without it.

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

> **Not yet verified on Windows.** These steps follow from the Linux procedure and the tools' own requirements, but the sequence has not been run end to end on a Windows machine.

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.

Checking it worked

Open a terminal (Linux) or Command Prompt (Windows), change into your Generator Tools folder, and start the window:

```
Linux:    cd ~/Documents/"Generator Tools"
          python3 protocol_tool_gui.py
Windows:  cd "%USERPROFILE%\Documents\Generator Tools"
          python protocol_tool_gui.py
```

The quotation marks matter on both systems, because the folder name has a space in it. If a window opens with a Serial port dropdown and two buttons, everything is installed correctly — close it and carry on. If instead you see a message about a missing module, go back to the step above for your operating system.

Building the file: the Protocol Editor

Why use the editor

A protocol file has a shape a spreadsheet cannot see: many nine-field frequency records, followed by exactly one eleven-field control record at the very end. To a spreadsheet these are simply rows of unequal length, so it pads them, reformats the numbers to suit itself, and checks nothing. A file can come back out looking perfectly reasonable and still be unusable by the Generator — and you will not find out until the transfer fails, or until a run behaves oddly.

The Protocol Editor shows you the same familiar grid, but it understands the record structure, keeps the record numbering consecutive for you, and checks every value as you type it against the same limits the Generator's own rotary-knob editing enforces. A file saved by the editor needs no correction before it is sent.

Starting the editor

The editor lives with the other PC tools, in the same folder as protocol_tool.py, and it must be started from that folder — it uses protocol_tool.py to read and write files.

Open a terminal (Linux) or Command Prompt (Windows), change to that folder, then:

```
Linux:    python3 protocol_editor.py
Windows:  python protocol_editor.py
```

That opens the editor with an empty file, ready to build one from scratch. To open an existing file straight away, add its name — and, if the file lives somewhere else, the folder it is in:

```
python3 protocol_editor.py --file protocols-3.txt
python3 protocol_editor.py --file protocols-3.txt --dir ~/Desktop
```

You can also start with no file at all and use Open… in the window.

If the editor will not start and reports that tkinter is missing, install it — on Linux Mint:

```
sudo apt install python3-tk
```

What you see

The window has four parts, top to bottom.

The toolbar — New, Open…, Save, Save As…, Insert Row Above, Insert Row Below, Delete Row.

The frequency grid — the large scrolling area, one row per frequency record, under a fixed header that stays in place as you scroll. Its columns are:

Column	Holds
Rec #	The Memory Group number. You do not type this — the editor keeps it consecutive by itself.
Freq 1 – Freq 4	The four channel frequencies, in Hz.
Duty 1 – Duty 4	The four channel duty cycles, as a fraction: 0.50 is 50%.

The control record — a single row below the grid, labeled Control Record (Rec 0), with its own headers: 0, Mem Grp, Mode, Freq Time, Prog Time, Start Freq Grp, Stop Freq Grp, Start Sweep Grp, Stop Sweep Grp, Sweep Inc, File #. The leading 0 is what marks this row as the control record; it is fixed and cannot be edited.

Two notes on that row. Freq Time and Prog Time are whole seconds and whole minutes here — see the note on them in Chapter 3, which explains what a zero in either field actually does. File # should be left at 0: it does not decide which SD card slot the file is written to. The filename you send decides that, as described under “Which Generator file actually gets overwritten” below.

The status line at the very bottom, showing the current file and the record count.

Editing

Click any cell and type. To work down a column without reaching for the mouse, use Enter and the arrow keys to move between cells.

To add or remove records, click a row’s Rec # to select it, then use Insert Row Above, Insert Row Below, or Delete Row. New rows arrive filled with safe defaults, and the Rec # column renumbers itself immediately afterward, so the numbering never develops gaps. A file must keep at least one frequency record, so the last one cannot be deleted, and a file cannot grow beyond 275 records.

The limits it enforces

These come from the Generator's own limit code, so the editor allows what the Generator allows:

Field	Range
Freq 1–4	0.04 – 65535.00 Hz
Duty 1–4	0.00 – 1.00
Mode	1 – 4
Freq Time	0 – 9999 whole seconds
Prog Time	0 – 9999 whole minutes
Sweep Inc	0.00 – 9999.99
File #	0 – 99
Mem Grp and the four Group fields	A whole number, 1 or higher
Records per file	275 maximum

The frequency floor of 0.04 Hz is a hardware limit, not an arbitrary one: the PWM hardware cannot generate below about 0.0373 Hz, and the firmware clamps anything lower.

The four Group fields — Start Freq Grp, Stop Freq Grp, Start Sweep Grp and Stop Sweep Grp — point at records that have to exist, so their upper limit depends on how many records the file currently holds. They are checked when you save rather than as you type.

Checking and saving

A cell whose value is out of range, or is not a number at all, turns pink the moment you type it. You do not have to hunt for the problem later.

Save and Save As… run the full check over every cell, plus the cross-record check on the Group fields. If anything is wrong the save is refused and the offending cells are marked, so a bad file never reaches the Generator at all. Fix the pink cells and save again.

Files are written in exactly the format the Generator expects, which is what allows a file saved here to be sent with no further correction.

Where this fits

The whole cycle:

1. Receive Fm Generator — pull the file you want to change onto the PC (Chapter 6), or start a brand new one in the editor.
2. Edit it in the Protocol Editor and save it.
3. Send to Generator — send it back, using the step-by-step procedure below.

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 ~/Documents/"Generator Tools" then python3 protocol_tool_gui.py
   Windows: cd "%USERPROFILE%\Documents\Generator Tools" 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 **"Send 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 Send?"* 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 **20 seconds**.

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

10. Watch the **Activity log**: you'll see `ACK:FREQ:1`, `ACK:FREQ:2`, …, `ACK:CTRL`, then `UPLOAD_OK`, ending with *"Send succeeded."* The Generator's screen shows **"TRANSFER GOOD"**, on two lines — the same wording whichever way the file was going.

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.

What actually happens on the wire

Useful when a transfer fails and the procedure above has not told you why.

Before anything is sent, the PC tool validates the whole file locally — field counts, number formats, ranges, exactly one control record, no more than 275 frequency records. If any of that fails, every problem is listed with line numbers and **nothing is transmitted**. The Generator is never left half written because a bad file never leaves the PC.

Step	Direction	Line	Notes
1	Generator → PC	`UPLOAD_READY`	Repeated once a second for up to 20 seconds, so either end may be started first.
2	PC → Generator	`BEGIN_FILE:N`	N is the destination slot, taken from the filename.
3	PC → Generator	*record lines*	One line at a time. The Generator checks each as it arrives — 9 fields for a frequency record, 11 for the control record, all strictly numeric — and writes it to a scratch file.

Step	Direction	Line	Notes
4	PC → Generator	`END_FILE`	End of data.
5	Generator	—	Renames the scratch file over `/protocols-N.txt`. The existing file is only replaced once the new one is complete.
6	Generator	—	**If N is 0**, reloads it into working memory immediately. Any other slot is left parked on the card.
7	Generator → PC	`UPLOAD_OK`	Transfer complete.

Two consequences of step 6 that catch people out:

Sending to a non-zero slot changes nothing you can see. The file is on the card, but the Generator goes on running what it already had. Activate it from `Setup=5`'s `File=` dial, or send to slot 0 instead.

Sending to slot 0 takes effect at once, with no reset needed, because slot 0 is the live file.

A line beginning `ERROR:` came from the Generator, not from the PC tool. They are listed in Error and status messages.

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 ~20-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 **"Receive Fm Generator"** button.

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

Before you start (one-time setup per PC)

The one-time setup is the same for both directions and is covered in Chapter 5, under "Getting the PC tools onto your computer (one time)" — where to get the programs, where to put them, and how to install Python, pyserial and tkinter on Linux and on Windows. If you have already done it for sending files to the Generator, there is nothing more to do here.

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 ~/Documents/"Generator Tools" then python3 protocol_tool_gui.py
Windows: cd "%USERPROFILE%\Documents\Generator Tools" 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 **"Receive Fm Generator"**. A *"Confirm to Proceed with Receive?"* 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 20 seconds** — click **Yes** on the *"Confirm to Proceed with Receive?"* popup.
10. Watch the **Activity log** in the window: `TX:` / `RX:` lines scroll by as the file transfers, ending with *"Receive succeeded."* The Generator's screen shows **"TRANSFER GOOD"**, on two lines. The same wording is used whichever way the file was going; a failure reads "TRANSFER FAILED".
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.

What actually happens on the wire

Useful when a transfer fails and the procedure above has not told you why. Everything the Generator sends is plain text, one line at a time.

Step	Direction	Line	Notes
1	Generator → PC	`DOWNLOAD_READY`	Repeated once a second for up to 20 seconds, each time followed by a plain-language hint line. Repeating it means it does not matter which end was started first.
2	PC → Generator	`READY:N`	N is the slot the PC wants, taken from the filename you chose. `QUIT` aborts instead.
3	Generator	—	Opens `/protocols-N.txt`. If the card has no such file it answers `ERROR:FILE_NOT_FOUND` and stops here.
4	Generator → PC	`BEGIN_FILE`	Start of data.
5	Generator → PC	`#`	A header line, so the saved file has one. The file's own header is not sent.
6	Generator → PC	*record lines*	Every frequency record, then the control record.
7	Generator → PC	`END_FILE`	End of data.
8	Generator → PC	`DOWNLOAD_OK`	Transfer complete.

The PC saves everything between steps 4 and 7 to the filename you gave.

The number in your filename is the slot being asked for. It is not a label — name the file `protocols-12.txt` and the PC asks the card for slot 12.

Receiving cannot create a slot. It reads what is already on the card. To bring a new slot into existence, send a file to it — see PC to Generator.

A line beginning `ERROR:` came from the Generator, not from the PC tool. They are listed in Error and status messages.

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, Generator_V2_05.ino.merged.bin — directly into the Generator's flash memory over the USB cable, replacing the firmware it is running now. Your protocol files live on the microSD card and are not touched.

- 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.

To compile the firmware yourself rather than install the ready-made image, see Chapter 8 — building from source.

What you need

The whole `firmware_update_package` folder, containing:

- Generator_V2_05.ino.merged.bin — the object file to be installed (this single file already contains the bootloader, the partition table, and the compiled program).
- `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.

Where to get it

The `firmware_update_package` folder is supplied in two places, and they hold the same files. Use whichever is easier.

From the Generator's microSD card. Switch the Generator off, take the card out, and put it in your computer's card reader. Copy the whole `firmware_update_package` folder onto your computer. Put the card back in the Generator when you are finished — the Generator will not run without it.

From the Aurorasky website. Open the Generator's page:

`https://www.aurorasky.net/html/4_channel_generator.html`

and download the firmware update package. It comes as a single zip file — right-click it and choose Extract All on Windows, or Extract Here on Linux.

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.)

1. 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.
2. When it asks **"Continue? [y/N]"**, type `y` and press Enter.
3. Wait — progress messages scroll by; this can take a few minutes. Don't unplug the Generator or close the terminal.
4. 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: Working With the Firmware Source

What this covers: the Generator's firmware as source code — where to get it, the features in it that are switched off or set at compile time, and how to build it yourself in the Arduino IDE.

This chapter is for people changing the firmware. If you use the Generator as a tool — driving a pulser, a plasma ball or a light source — you never need any of it. Installing an updated firmware is covered in Chapter 7, and needs no programming tools at all.

Where to find the source code

This manual does not reprint any source code. The complete sources — the eight `.ino` firmware files, the three Python PC tools, and the compiled object file `Generator_V2_05.ino.merged.bin` — are supplied in full in two places, and both hold the same files:

On the Generator's microSD card. Switch the Generator off and read the card in your computer's card reader. Put the card back when you are finished — the Generator will not run without it.

On the Aurorasky website, on the Generator's page:

`https://www.aurorasky.net/html/4_channel_generator.html`

Files downloaded from the website end in `.txt`. The eight firmware files and the three Python tools are published with `.txt` endings so a web browser will open them. Rename them before use:

- **Firmware files:** change `.txt` to **`.ino`**. For example, `Generator_V2_05.txt` becomes `Generator_V2_05.ino`. The Arduino IDE compiles only `.ino` files. The eight are `Generator_V2_05`, `micro_SD`, `PWM`, `rotary_Decode`, `SSD1306`, `support_Routines`, `USB_Comm` and `WiFi`.
- **Python tools:** change `.txt` to **`.py`**. For example, `protocol_tool.txt` becomes `protocol_tool.py`.

On Windows, file name endings are hidden until you turn them on: in File Explorer, switch on **File name extensions** under the **View** menu. Without that, renaming produces a name like `Generator_V2_05.ino.txt`, which won't work.

This applies only to files downloaded from the website. The copies on the Generator's microSD card already have the right endings and need no renaming.

Printing the sources here would roughly double the size of this manual for information that almost no one operating a Generator needs. Anyone building, modifying, or reading the code wants the real files in any case, not a listing typed back in from a printed page.

Features in the firmware that are switched off or set at build time

These are part of the source but cannot be changed from the Generator's front panel. Each one needs an edit to `Generator_V2_05.ino`, a rebuild, and a reflash.

- **WiFi remote start/stop** — the code is present, but `wifiEnable` is set to `0` as shipped, and nothing on the microSD card can turn it on. What the feature does and how to set it up is in WiFi operation; switching it on means changing that one value and rebuilding as described below.
- **Output polarity (`dutyFlip`)** — inverts a channel's output for hardware variants whose output stage inverts. Not needed on a standard Generator.

Both appear, with the other build-time values, in Build-time settings below.

Building the firmware from source

Everything needed to compile the firmware yourself: the exact board package, libraries and settings, what a good build looks like, and how to get the result onto a Generator.

What you need

- **Arduino IDE 2.x.** The firmware is built and verified with 2.3.10.
- **The eight source files** — `Generator_V2_05.ino`, `micro_SD.ino`, `PWM.ino`, `rotary_Decode.ino`, `SSD1306.ino`, `support_Routines.ino`,

`USB_Comm.ino`, `WiFi.ino` — from the Generator's microSD card or its page on the Aurorasky website.

- **A USB data cable.** Charge-only cables carry no data and the port never appears.
- **On Windows, the CP210x USB-to-UART driver** from Silicon Labs. The Generator uses a CP2102 chip.

Step 1 — Install the ESP32 board package, at the right version

In the IDE: **Tools → Board → Boards Manager**, search for **esp32**, and install **esp32 by Espressif Systems**, version **3.3.11**.

It is in the IDE's default package list, so no extra Boards Manager URL is needed.

Use exactly 3.3.11 and do not let the IDE update it. `PWM.ino` drives the ESP32's MCPWM peripheral by writing its hardware registers directly, which is what gives the Generator its fractional-frequency accuracy. That code is tied to the register layout and driver behaviour of this board package. A different version can compile cleanly and still produce wrong frequencies on the outputs.

Step 2 — Install the display library

Tools → Manage Libraries, search for **Adafruit SSD1306**, and install it. When the IDE offers to install its dependencies, accept.

Library	Version built with	Why
Adafruit SSD1306	2.5.17	The OLED display
Adafruit GFX Library	1.12.6	Installed as a dependency of SSD1306
Adafruit BusIO	1.17.4	Installed as a dependency of GFX

Nothing else needs installing. `SD`, `SPI`, `EEPROM` and `WiFi`, and the MCPWM driver headers, all come with the ESP32 board package.

Step 3 — Set up the sketch folder

Put all eight `.ino` files together in one folder named **`Generator_V2_05`**. If you downloaded them from the website, rename them from `.txt` to `.ino` first — see Where to find the source code. The folder name must match the main file's name, or the IDE will offer to move things around. Open `Generator_V2_05.ino`; the other seven appear as tabs.

Only the `.ino` files at the top of that folder are compiled. Subfolders such as `build`, `Protocols-x` or `firmware_update_package` can sit alongside them and are ignored.

Step 4 — Select the board and settings

Tools → Board → esp32 → Nano32, then set:

Setting	Value
Board	**Nano32**
Upload Speed	921600
Flash Frequency	80MHz
Core Debug Level	None
Erase All Flash Before Sketch Upload	Disabled

The board's own defaults supply the rest — 4 MB of flash in DIO mode and its standard partition table. Leave any partition setting at its default: the application slot is 1,310,720 bytes and the firmware needs about 78% of it.

Step 5 — Compile, and what a good build looks like

Sketch → Verify/Compile. A correct build ends with figures close to:

```
Sketch uses 1028373 bytes (78%) of program storage space. Maximum
is 1310720 bytes.
Global variables use 49704 bytes (15%) of dynamic memory, leaving
277976 bytes for local variables.
```

Three kinds of warning are expected, and none of them is a problem:

Warning	What it means
`legacy MCPWM driver is deprecated, please migrate to the new driver`	The firmware deliberately uses the legacy MCPWM driver alongside direct register access. **Do not "fix" this by migrating to `driver/mcpwm_prelude.h`** — the register-level PWM code is written against the legacy driver, and mixing the two is not supported.
`NetworkServer::available()` is deprecated: Renamed to `accept()`	In `WiFi.ino`. The old name still works.
`'++'` / `'--'` expression of `volatile`-qualified type is deprecated	In `rotary_Decode.ino`'s encoder interrupt. A newer C++ style rule; the code behaves correctly.

Any **error**, as opposed to a warning, almost always means the board package is not 3.3.11, the SSD1306 library is missing, or the board is not set to Nano32.

Step 6 — Get the firmware onto a Generator

Either upload directly: connect the Generator by USB, choose its port under **Tools → Port**, and use **Sketch → Upload**. No buttons need holding; the board is reset into its flashing mode automatically.

Or make an installable image: Sketch → Export Compiled Binary writes the build into the sketch folder under `build/esp32.esp32.nano32/`. The file that matters there is:

`Generator_V2_05.ino.merged.bin`

It is a complete 4 MB flash image — bootloader, partition table and program in one file — produced automatically on every build. Copy it into the `firmware_update_package` folder, replacing the one there, and the installer scripts in Firmware update will flash it. They write it to address `0x0` at 115200 baud.

After any source change, export again before copying. A `merged.bin` left over from an earlier export does not contain later changes, and nothing warns you that it is out of date.

For reference, the image is laid out as:

Address	Contents
`0x1000`	Bootloader
`0x8000`	Partition table
`0xe000`	OTA boot selector
`0x10000`	The firmware

Build-time settings

These live near the top of `Generator_V2_05.ino`. Changing any of them means rebuilding and reflashing.

Setting	As shipped	What it does
`DEBUG`	`0`	`1` sends diagnostic text out of the USB serial port; `0`, as shipped, keeps it quiet. The text shares the port with the PC transfer protocol, whose tools ignore lines they do not recognise.
`BAUD`	`115200`	Serial speed. Must stay 115200 — the PC tools assume it.
`wifiEnable`	`0`	`1` turns on the WiFi remote start/stop. See WiFi operation.
`dutyFlip`	`0x0F`	Per-channel output polarity, for hardware variants that invert an

Setting	As shipped	What it does
		output stage.
`VERSION`	`"2.05 …"`	The text shown on the start-up screen. Change it so your own build can be told apart.
`FLASH_SIZE`	`10800`	Size of the working-memory image, in bytes.
`MAX_FREQ_RECORDS`	`275`	Most frequency records one protocol file may hold.

Leave `FLASH_SIZE` and `MAX_FREQ_RECORDS` alone unless you are deliberately changing the protocol file format. They are sized to each other, and the PC tools enforce the same 275-record limit — raising one without the others lets a file upload that the Generator cannot load.

Optimize for Debugging. The IDE's **Sketch → Optimize for Debugging** option builds with `-Og -g3` instead of the normal `-Os`. The figures in step 5 are from a normal build. A debugging build runs the frequency calculations more slowly, so it is better suited to development than to a unit going into use.

If something goes wrong

Symptom	Likely cause
Nano32 is not in the board list	The ESP32 board package is not installed.
`Adafruit_SSD1306.h: No such file or directory`	Step 2 not done, or its dependencies were declined.
Sketch too big	The partition setting was changed from its default.
No port appears	Charge-only cable, or the CP210x driver is missing on Windows.
Builds and uploads, but output frequencies are wrong	Board package is not 3.3.11.
Serial Monitor shows garbage	Set it to 115200 baud.

Related

- Chapter 7 — Firmware update — installing a ready-made image
- Appendix A — Architecture map — how the eight files fit together, before you change them
- WiFi operation — what `wifiEnable` switches on

Chapter 9: Saving Updates — Working Memory and the Card

What this covers: the two different things "save" means on this Generator, how to do each one, and why a change you made can disappear at the next reset. This is the single most misunderstood part of operating the device.

There are two levels of save, and they are not the same

Level 1 — into working memory. Fast, happens constantly, and is **lost at the next reset**.

Level 2 — out to the card. Deliberate, needs both buttons, and is **permanent**.

The reason level 1 does not survive a reset is simple: every time the Generator boots it reads `protocols-0.txt` off the card and loads it into working memory, overwriting whatever was there. Working memory is a copy of the card, so anything you changed but did not write back to the card is replaced by the card's version.

```
   card: protocols-0.txt  ──boot──▶  working memory  ──▶  what the Generator runs
                                  ◀──level 2 save──
```

Level 1 — locking a change into working memory

While you are on any Setup screen (`Setup=1` through `Setup=5`), turning the knob changes the value under the cursor, but nothing is committed yet.

Press the red run button. The display flashes `Update` for about half a second. Everything you changed on that screen is now in working memory and the Generator will use it.

That is all the red button does on a Setup screen. It does not start the Generator from there and it does not touch the card. To run, go back to `Setup=0` first.

This change is gone at the next power-on or reset. If it matters, do a level 2 save.

Level 2 — writing working memory out to the card

This is the two-button save.

1. Be on any Setup screen — `Setup=1` through `Setup=5`.
2. **Press and hold the black cursor button.**
3. With it still held, **press the red run button.**
4. The display shows `Update the SD Card ??` and the LED blinks for about four seconds. **Keep the black cursor button held down** through this.
5. If it is still held when the four seconds are up, the display changes to `SDram Card UPDATING` and the write happens.

Letting go during step 4 cancels the save. Nothing is written and nothing is harmed.

While `SDram Card UPDATING` is on the screen, do not press reset and do not pull the card. An interruption there can damage `protocols-0.txt`. The Generator will usually rebuild it from a backup at the next start-up, but changes made since that backup can be lost.

What actually gets written

Everything in working memory goes out as a complete `protocols-0.txt`: every frequency record, and the control record holding mode, both timers, all four group IDs, the sweep increment and the file number.

It always writes to `protocols-0.txt`, whatever file you originally loaded from. Slot 0 is the live file — the one read at every boot — so saving is how you make the current state the new default.

That has a consequence worth knowing: load preset 30, change something, save, and the result lands in slot 0, not slot 30. Preset 30 is untouched. If you want the change kept as a preset too, send the file back to that slot from a PC.

If the save fails

The Generator builds the new file as a scratch file and only puts it in place once it is complete and has been read back successfully. If anything goes wrong you get `SAVE FAILED`, the reason, and `old file kept` — and the file already on the card is exactly as it was.

The reasons are listed in Error and status messages.

Quick reference

You want to	Do this	Survives reset?
Try a value out	Turn the knob	No — not even committed
Use a value now	Red button on a Setup screen	No
Keep a value for good	Black held + red, keep holding	**Yes**
Make a preset the new default	Load it from `Setup=5`, then save	**Yes**

Related

- The Setup screens
- Error and status messages
- Recovering a corrupted protocols-0.txt

Chapter 10: WiFi Operation

What this covers: the Generator's built-in WiFi remote start/stop, what it does, what it needs on the card, and — importantly — what has to change in the firmware before any of it works in the build you have.

What it is

A deliberately basic remote on/off switch. The Generator joins your wireless network, shows its address, and serves a single web page with Start and Stop links. Point a phone or a laptop at that address and click.

It is a proof of concept rather than a product feature. The source is open and the wireless side is already written, so anyone comfortable in C and the Arduino IDE has a working starting point to build something more capable.

It is switched off in this build

Putting the file on the card is not enough. In the firmware as shipped:

```
byte wifiEnable = 0;        // Generator_V2_05.ino
```

and the whole start-up sequence is gated on that being `1`. Since nothing sets it to `1` before that point, the WiFi block never runs, the card is never checked for credentials, and no connection is attempted.

To turn it on: change that line to `byte wifiEnable = 1;`, rebuild, and flash — see Chapter 8 — building from source. Then the behaviour below applies.

This is why the Generator will happily ignore a correctly written `wifidata.txt` and start up as though nothing is there.

Setting up the credentials

The card carries a template file named `wifidataX.txt` holding two lines:

```
YourSSID
YourPassWord
```

Line 1 is your network name, line 2 is its password. Edit both in any plain text editor, then **save it as `wifidata.txt`** — the same name with the X removed. The Generator looks for `wifidata.txt`; the X version is only a template so the real one is never overwritten by a card refresh.

Keep it plain text. A word processor that adds formatting will produce a file the Generator cannot read.

What happens at start-up

With WiFi compiled in:

- **No `wifidata.txt` on the card** — the Generator starts normally with wireless off. This is not an error.
- **Credentials good** — the display shows `Connected`, then `IP Address` and the address itself, for 10 seconds, then start-up finishes.
- **Credentials wrong, or the network is down** — the display shows `WiFi Failed, Moving On in 10 Seconds`, waits, then starts normally with wireless off. The Generator always ends up running.

If you miss the address during its 10 seconds, press the reset button and watch the screen again. The reset button is the rotary knob — press it in.

Using it

Type the address into a browser on any device on the same network. You get a plain page with Start and Stop. That is the whole interface.

The Generator runs whatever is currently loaded, exactly as if you had pressed the red button. It does not let you choose a file, change a mode, or edit values — for any of that you use the front panel or a PC.

If you want to build on it

The wireless code lives in `WiFi.ino`, and the request handling runs inside the main loop. Two things are worth knowing before you extend it: the exchange is bounded (a two-second idle timeout, a ten-second ceiling, and a 128 character limit on the request line) so a stalled client cannot hold the front panel and the run timers still, and anything you add runs inside the same loop that services the knob and the timers, so it needs to return promptly.

Related

- Chapter 8 — working with the firmware source
- Error and status messages
- Appendix A — Architecture map — section 25 covers the WiFi architecture

---

Chapter 11: Error and Status Messages

What this covers: every message the Generator can put on its OLED display, and every error the PC tools can report, with what causes it and what to do. Messages are grouped by when they appear.

If you are looking up a message you are seeing right now, search this page for the exact words on the screen.

Card and file errors at start-up

These appear when the Generator reads a `protocols-nn.txt` file — at power-on, at reset, or when a file is selected from `Setup=5`. The Generator will not run on a file it could not read properly, so these stop it with the LED blinking. Fix the card and power-cycle.

`protocols-0.txt` is the exception: the Generator usually repairs it by itself. If slot 0 is missing or won't load at start-up, the Generator tries the card's backups in turn — `protocols-0.bak`, then `protocols-99.txt`, then `protocols-98.txt` — and uses the first one that loads. It shows this for about four seconds, rewrites `protocols-0.txt` from that backup, and starts normally:

Large text	Small text	What happened	What to do
`File 0 BAD / BACKUP`	`from protocols-99.txt / rewriting file 0` (names whichever backup was used)	`protocols-0.txt` was missing or damaged, so a backup was loaded and slot 0 is being rebuilt from it.	Nothing — the Generator starts normally. If slot 99 or 98 was used, it is now running that backup's protocols, and anything saved to slot 0 since that backup was made is gone.

You only see one of the errors below naming **file 0** if none of the backups load either.

Large text	Small text	What happened	What to do
`Micro SD / FAILURE`	—	The card could not be mounted at all.	Card missing, not seated, or dead. Reseat it. The Generator cannot run without a card.
`File n / NO # MARK`	`no # header line`	The file has no # character. Every protocol file needs a header line containing one.	Add a header line, e.g. `Protocols-12 #`, as the first line.
`File n / TOO MANY`	`over 275 records`	The file holds more than 275 frequency	Remove records, or split the sequence across

Large text	Small text	What happened	What to do
		records, which is more than the working image can take.	two files.
`File n / NO END REC`	`no 0,... end record`	The file has frequency records but no control record. Most often a save that was interrupted, because the control record is written last.	Restore the file, or add the 11-field control record back.
`File n / NO RECORDS`	`no frequency records`	The file has a header and possibly a control record, but no frequency records at all.	Restore the file.
`File n / FAILURE`	`cannot open file`	The card mounted, but that particular file could not be opened.	The file is missing from the slot, or the card is failing.

The `n` is the slot number it was trying to read. Slot 0 is the file loaded at every start-up, so an error naming file 0 means the automatic repair found no usable backup either, and the Generator will not start until the card is fixed — see Recovering a corrupted protocols-0.txt. A preset chosen from `Setup=5` is never repaired automatically; an error naming its slot stops the Generator until the card is fixed.

Save errors

These appear when a save to the card fails. The headline is always `SAVE / FAILED`, with the reason in small text underneath and `old file kept` on the last line.

`old file kept` means what it says — the existing file on the card was not touched. The Generator builds the replacement in a scratch file first and only swaps it in once it is complete, so a failed save costs you the new values, not the file you already had.

Detail line	What happened	What to do
`cannot open work file`	The scratch file could not be created.	Card full, write-protected, or failing.
`card full or`	A write returned short,	Free space on the card,

Detail line	What happened	What to do
`failing`	meaning not everything reached the card.	or replace it.
`work file unreadable`	The scratch file was written but reading it back gave nothing usable.	The card is unreliable. Replace it.
`rename failed`	The replacement was complete but could not be put into place. The original was put back.	Card fault. Check the file on a computer before trusting it.
`cannot park old file`	The existing file could not be renamed aside, so the save was abandoned before anything was replaced.	Card fault or a stale `.bak` file that cannot be removed. Check the card on a computer.

Normal status messages

Not errors. These are the Generator telling you where it is.

Message	Means
`Update`	A Setup screen's values have been accepted into working memory. Lost at the next reset unless saved to the card.
`Update the / SD Card ??`	Waiting to be told whether to write working memory out to the card. Keep the black cursor button held to confirm.
`SDram Card / UPDATING`	Writing to the card now. **Do not reset or remove the card while this is showing.**
`Load / Proto = nn / Press Blk Button / to Continue`	A different file number is dialled in on `Setup=5` than the one currently loaded. Press the black cursor button within five seconds to load it, or wait and nothing happens.
`Running`	The Generator is producing output.
`Suspend`	Output is paused. Press the black cursor button again to resume.
`ADJUST MODE`	Mode 4 is running; the knob is a live frequency control for Channel 1.

Message	Means
`Version n`	Shown briefly at start-up.

WiFi messages at start-up

Only appear when WiFi has been compiled in. See WiFi operation.

Message	Means
`Connected` / `IP Address` / *an address*	Joined the network. Shown for 10 seconds; that address is the Generator's control page.
`WiFi Failed, Moving On in 10 Seconds`	Could not join. The Generator starts normally afterwards with WiFi off.

USB transfer messages

On the Generator, the transfer screens show `USB Comm.`, then `Send To CPU`, `Receive Fm CPU` or `Return to Main`, and finally `TRANSFER` with the result.

The PC tools report the Generator's own error codes. These arrive over the cable, so a message beginning `ERROR:` came **from the Generator**, not from the PC.

Code	What happened	What to do
`ERROR:FILE_NOT_FOUND`	The card has no file in the slot the PC asked for.	Ask for a slot the card holds. To *create* a slot, send a file to it — receiving only reads slots that already exist.
`ERROR:NO_READY_RECEIVED`	The Generator waited for the PC to name a slot and nothing valid arrived.	The PC side was not started within the window, or is not speaking the current protocol.
`ERROR:TIMEOUT_WAITING_BEGIN`	The PC announced itself and then went quiet.	Restart both ends.
`ERROR:EXPECTED_BEGIN_GOT:` *text*	The PC sent something other than the expected start line.	The two ends are out of step. Restart both.
`ERROR:BAD_FREQ_LINE:` *text*	A frequency record in the incoming file is malformed.	The line is quoted in the message. Fix it and resend.
`ERROR:BAD_CTRL_LINE:` *text*	The control record is malformed.	As above.
`ERROR:BAD_TOKEN_CO`	A line has the wrong	Fix the line.

Code	What happened	What to do
`UNT:` *text*	number of comma-separated fields — 9 for a frequency record, 11 for the control record.	
`ERROR:LINE_TIMEOUT_AFTER_FREQ:` *n*	The transfer stalled after record *n*.	Cable or port problem. Retry.
`ERROR:SD_OPEN_FAILED`	The Generator could not open its scratch file.	Card full or failing.
`ERROR:SD_RENAME_FAILED`	The transfer completed but the file could not be put into place.	Card fault. Verify the card on a computer.
`DOWNLOAD_ABORTED`	The transfer was cancelled from the PC.	Not an error.

Two more come from the **PC side**, not the Generator:

Message	Means
`Timed out waiting for UPLOAD_READY / DOWNLOAD_READY`	The Generator was not put into the matching mode in time. Start the device side first: *USB Comm → Receive Fm CPU* to send, *Send To CPU* to receive.
`Serial error: ... resource busy`	Another program holds the serial port. Close the Arduino IDE serial monitor or MiniCom.

Related

- Recovering a corrupted protocols-0.txt
- Saving updates — what the two save levels actually write
- The microSD card

Chapter 12: Recovering a Corrupted protocols-0.txt

What this covers: what happens when the file the Generator reads at every start-up is missing or damaged — how the Generator usually repairs it by itself, and how to get it running again with a card reader when it can't.

The Generator usually repairs this by itself

`protocols-0.txt` is read at every start-up. If it is missing, or won't load because it is damaged, the Generator doesn't stop straight away. It tries the card's backups, in this order:

1. **`protocols-0.bak`** — left on the card when a save was interrupted, and holding the last good save.
2. **`protocols-99.txt`** — a spare copy of `protocols-0.txt`.
3. **`protocols-98.txt`** — a second spare copy.

The first one that loads is used. The display shows, for about four seconds:

```
File 0 BAD
BACKUP
from protocols-99.txt
rewriting file 0
```

The Generator then rewrites `protocols-0.txt` from that backup, safely, and starts normally. The next start-up is completely normal.

What you lose: if `protocols-0.bak` was used, nothing — it holds the last good save. If slot 99 or 98 was used, the Generator is now running that backup's protocols, and anything saved to slot 0 since the backup was made is gone.

Only slot 0 is repaired this way. A preset chosen from `Setup=5` that won't load still stops with its own error.

When it can't repair itself

If none of the backups load either — they are missing, or damaged too — the Generator shows a message naming **file 0** and stops with the LED blinking. Typical messages:

```
File 0              File 0              File 0
NO END REC          NO RECORDS          FAILURE
no 0,... end        no frequency        cannot open
record              records             file
```

The blinking-LED stop is intentional. The Generator refuses to run on a file it could not read completely, rather than starting with half a file and whatever was left in memory. From here the recovery is done on a computer.

The usual cause

An interrupted save. The control record is written last, so a reset or a pulled card partway through a save can leave a file with records and no control record — which is exactly the `NO END REC` case.

The two spare copies on the card

Slots **98** and **99** hold backup copies of a known-good `protocols-0.txt`. They exist for this. Their internal file number is deliberately `0`, so a copy restores as a correct slot-0 file with nothing to edit afterwards.

Recovery with a card reader

1. **Switch the Generator off** and take the microSD card out.
2. Put it in a card reader on a computer.
3. **Rename the damaged file out of the way** rather than deleting it — `protocols-0.bad` will do. If the fault repeats it is worth having.
4. **Put a good `protocols-0.txt` on the card:** from your own backup of the card, or download `protocols-0.txt` from the Generator's page on the Aurorasky website.
5. **Put good copies back in slots 98 and 99** — copy the same good file to `protocols-98.txt` and `protocols-99.txt` — so the Generator can repair itself next time.
6. Delete `save.tmp` and `protocols-0.bak` if either is on the card. They are leftovers from the failed save.
7. Eject the card properly, put it back in the Generator, and power on.

It should start normally. If it still fails on file 0 with a known-good file on the card, the card itself is suspect — replace it.

Keeping the backups good

The automatic repair depends on slots 98 and 99. Whenever `protocols-0.txt` is in a state you'd be happy to go back to, copy it over both with a card reader.

Do this from the computer, not over USB. Sending slot 0's content to slot 99 over the cable would work, but the copy would carry `99` as its internal file number instead of `0`, and it would need editing before it could restore cleanly.

Avoiding it in the first place

The one risky moment is while `SDram Card UPDATING` is on the display. Don't reset and don't pull the card while that is showing. If it does happen, the Generator will usually rebuild the file from a backup at the next start-up, but changes made since that backup can be lost. See Saving updates.

Related

- Error and status messages
- Saving updates
- The microSD card

Chapter 13: Coming from Spooky2

What this covers: how the settings a Spooky2 user already knows — dwell, repeats, the frequency multiplier, per-frequency dwell — translate into the Generator's modes and protocol files, what works differently, and what has no equivalent. If you have used Spooky2, this page gets you running quickly.

This page is about settings only. The Generator is not a medical device, and nothing here makes any claim about what a frequency does.

The short version

The Generator is a **four-channel square-wave generator** that runs on its own. What it plays is stored as plain text files on a microSD card, so there is no software to install to use it, and a program can be written in a text editor, edited on the Generator itself, or written for you by an AI assistant from a plain description.

	Spooky2 (GeneratorX Pro)	The Generator
Outputs	Function-generator outputs with a choice of waveforms	Four channels, **square wave only**, each with its own duty cycle
Frequency range	Up to 40 MHz	**0.04 Hz to 65,535 Hz**
Output level	Adjustable amplitude, up to 20 Vpp	**Set by the power supply** — the outputs swing from 0 V to just under the supply voltage, 5 V to 12 V
Where programs come from	Loaded from the Spooky2 software	Text files on the microSD card, 100 of them
Running without a computer	Yes, after loading programs	Yes, always — the computer is only for editing files

Translating the settings

Dwell → Freq Time

Spooky2's **dwell** is how long each frequency runs, 180 seconds by default. The Generator's equivalent is **Freq Time**, in whole seconds.

The difference to know: **a Generator file has one Freq Time for every step in it.** Spooky2 lets each frequency carry its own dwell; the Generator does not. How to handle that is covered below.

Repeat Program → Progm Time

Spooky2 repeats a program a set number of times. The Generator has no repeat count — a sequence **loops on its own** until **Progm Time** runs out, in whole minutes.

To get the equivalent of *Repeat Program = N*, work out one pass and multiply:

```
Progm Time (minutes) = number of steps × Freq Time (seconds) × N ÷ 60
```

Progm Time goes up to 9999 minutes, about a week. There is no endless setting — for a very long run, use a large number.

Repeat Frequency → repeat the record

To play each frequency more than once before moving on, put that record in the file more than once, one after another.

Frequency Multiplier → do the multiplication first

There is no multiplier setting. Multiply the frequencies before you enter them. **Anything that comes out above 65,535 Hz cannot be played** — which rules out programs that depend on multiplying into the hundreds of kilohertz or megahertz.

Programs and presets → protocol files

A Spooky2 program — a list of frequencies — becomes a **protocol file**: one line per step, each line setting all four channels at once, followed by one control line that holds the mode and both timers.

A file holds up to **275 steps**. A longer program needs splitting across two files. The card has 100 file slots, `protocols-0.txt` to `protocols-99.txt`, selected from the Generator's `Setup=5` screen.

Sweeps → Mode 3

The Generator's **Sweep** mode moves Channel 1 from one frequency to another in fixed steps, holding each step for Freq Time, and starts over when it reaches the end. Channels 2–4 hold steady values while it runs. With the step size set to 0, the Generator works out the steps itself to spread the sweep across Progm Time.

Per-frequency dwell: the one real difference

A Spooky2 program such as

```
7.83=600,174,963
```

plays 7.83 Hz for 600 seconds and the others for the default 180. A Generator file cannot give one step a different time from the rest — but it can get the same result by **choosing a Freq Time that divides every dwell evenly, and repeating records to make up the difference.**

Here, 600 and 180 are both multiples of 60. So with Freq Time at 60 seconds:

Frequency	Dwell wanted	Records needed
7.83 Hz	600 s	10
174 Hz	180 s	3
963 Hz	180 s	3

That is 16 records at 60 seconds each: one complete pass is exactly 16 minutes, the same total as the Spooky2 program. Set Progm Time to 16 for one pass, or 32 for two.

This is exactly the kind of conversion an AI assistant does well. Give it the Spooky2 frequency list together with the protocol file format, and it can produce the finished file.

What works differently

Channels come in pairs. Channels 1 and 2 share one clock, and channels 3 and 4 share another. Channel 2 is most accurate when its frequency is reasonably close to Channel 1's, and the same holds for Channel 4 against Channel 3. To run two unrelated frequencies side by side, put them on **Channels 1 and 3**.

Every channel is a square wave. Duty cycle is set per channel from 0.00 to 1.00. At the two extremes a channel stops switching altogether and holds steady — `0.00` is off, `1.00` is on — so a channel can switch a relay or other equipment on and off during a sequence.

The supply is the output level. There is no amplitude control. For outputs that swing 0 to about 12 V, power the Generator from 12 V; for about 5 V, use a 5 V supply.

What has no equivalent

- **Waveforms other than square**
- **Adjustable amplitude or DC offset**
- **Modulating one output with another**
- **Frequencies above 65,535 Hz**
- **Biofeedback scanning**
- **Remote mode**

What the Generator does that may be new to you

- **No software needed to run, ever.** Power it on and it plays what is on the card.
- **Edit on the device.** Frequencies, duty cycles, timers and modes can all be changed from the front panel without a computer.
- **Four outputs at once**, each with its own frequency and duty cycle, all changing together at every step.
- **Live frequency control.** Mode 4 turns the knob into a real-time frequency dial for Channel 1.

Related

- The protocol file format — everything needed to write a file by hand or with an AI

- The Setup screens — the four modes in full
- Power on, start, stop, pause — supply voltage and output levels
- The microSD card — the 100 file slots

Appendix A: 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
`Generator_V2_05.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-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

Arduino Tab / File	Primary Responsibility
	`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

Field ranges. The Generator and the Protocol Editor both enforce these:

Field	Range
protoID	Whole number, 1 or higher
F1–F4	0.04 – 65535.00 Hz
D1–D4	0.00 – 1.00

A file may hold at most 275 frequency records.

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,sta
rtSweepGrpID,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
`startSweepGrpID`	Sweep mode start group
`stopSweepGrpID`	Sweep mode stop group
`sweepFreqInc`	Sweep frequency increment
`fileNo`	Protocol file number

Field ranges:

Field	Range
memGrp	Whole number, 1 or higher
modeID	1 – 4
endTime1	0 – 9999 whole seconds
endTime2	0 – 9999 whole minutes
startFreqGrpID, stopFreqGrpID	Whole number, 1 or higher; must name records that exist
startSweepGrpID, stopSweepGrpID	Whole number, 1 or higher; must name records that exist
sweepFreqInc	0.00 – 9999.99
fileNo	0 – 99; leave at 0 — the filename decides the SD card slot

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 `adjustModeRun()` when Mode 4 is active. 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 on start/stop groups and sweep parameters.
4	Adjust Mode	Real-time knob control of Channel 1's frequency and duty cycle while running.

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.

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
`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

Variable	Purpose
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
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:

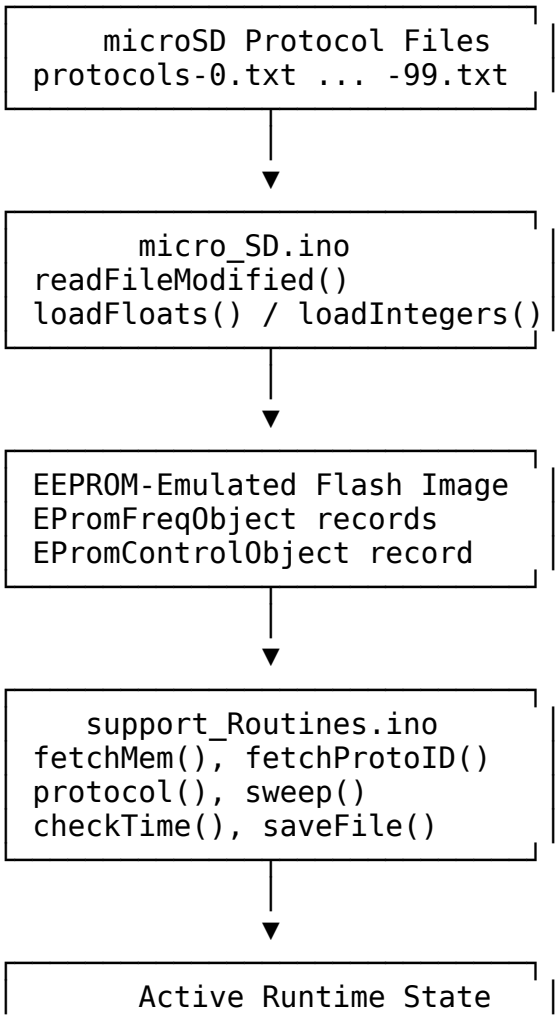
```
rotary encoder + cursor button
        ↓
```

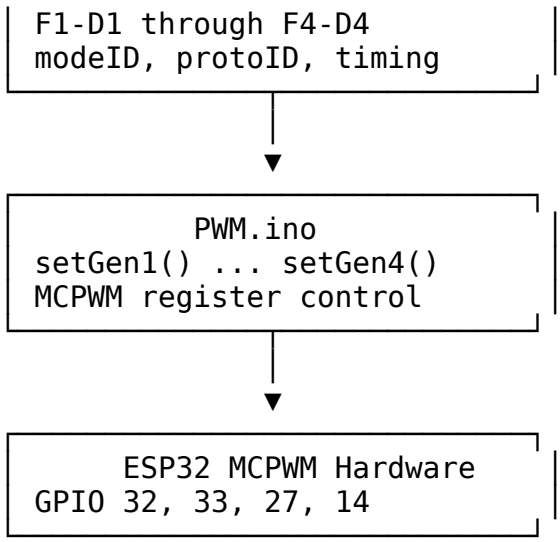
```
rotaryPositive() / rotaryNegative()
        ↓
rotaryLimits()
        ↓
Display()
        ↓
saveCurrent() / saveFile()
```

External communication can also modify state:

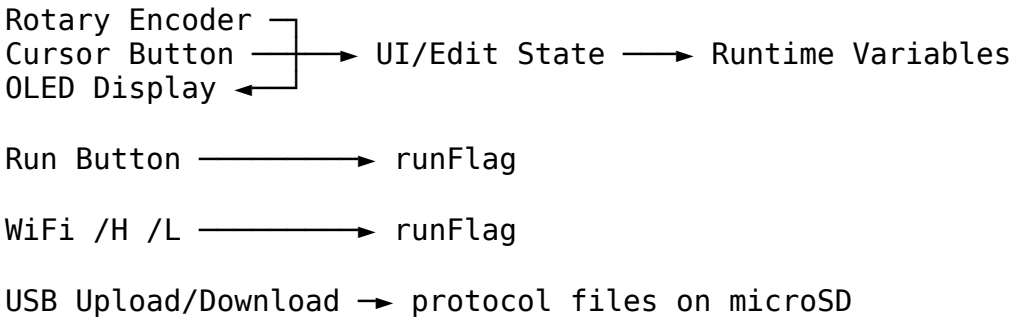
```
WiFi request
        ↓
runFlag
USB upload
        ↓
protocols-N.txt on SD card
        ↓
fileFetch() if protocols-0.txt was changed
```

28. Conceptual Block Diagram





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



29. Architectural Summary

The Generator is 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.

PWM is produced by a **register-level MCPWM engine**, giving direct control over frequency generation and fractional-frequency support on the master channels. Channels 1 and 3 are master PWM channels; channels 2 and 4 are secondary channels locked to the corresponding master MCPWM clock prescaler.

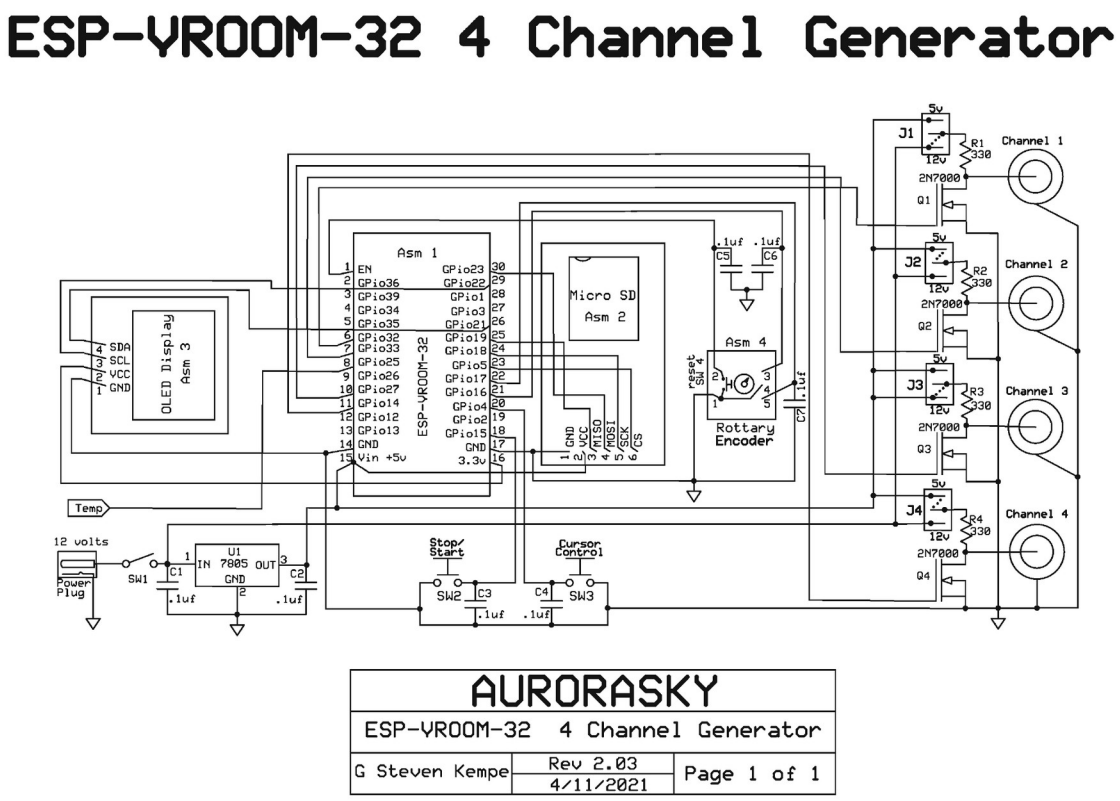
Around that sit a rotary encoder decoder with limit enforcement, SD-file handling with default-file behaviour through `protocols-0.txt`, and USB serial transfer for editing protocol files without physically removing the microSD card.

In short: 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 B: Schematic and Board Layout

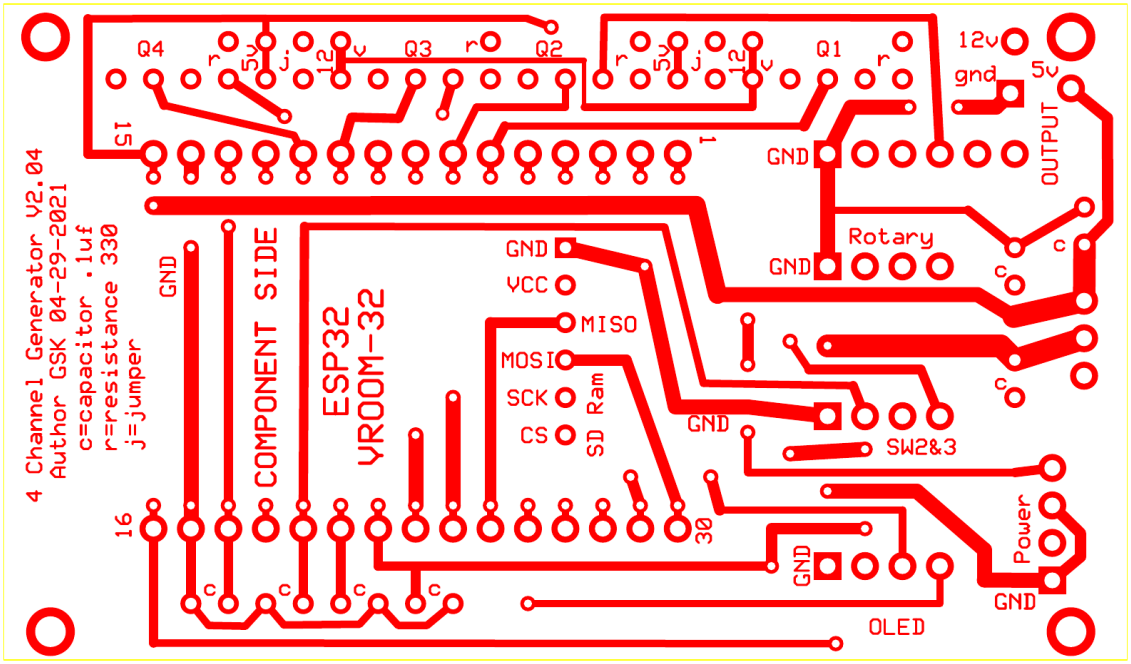
The circuit and board drawings for the ESP-VROOM-32 4 Channel Generator, included for anyone building, repairing or modifying a unit. These are reference material — nothing here is needed to operate the Generator.

Schematic



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

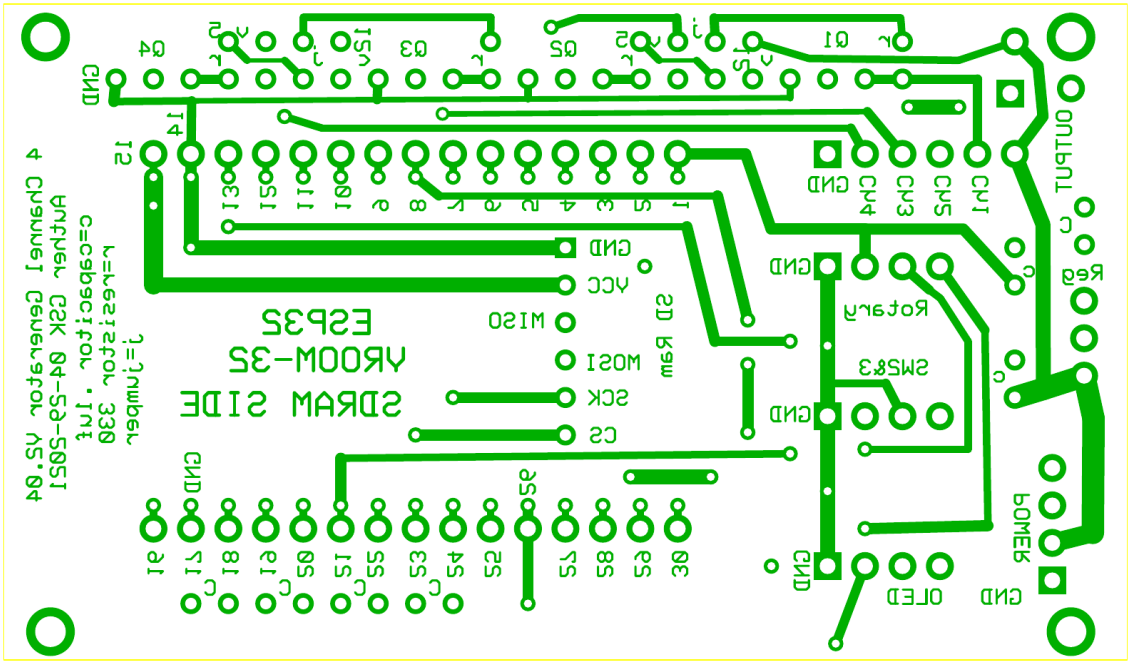
PCB — top



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

Board layout, top side, revision 2.04.

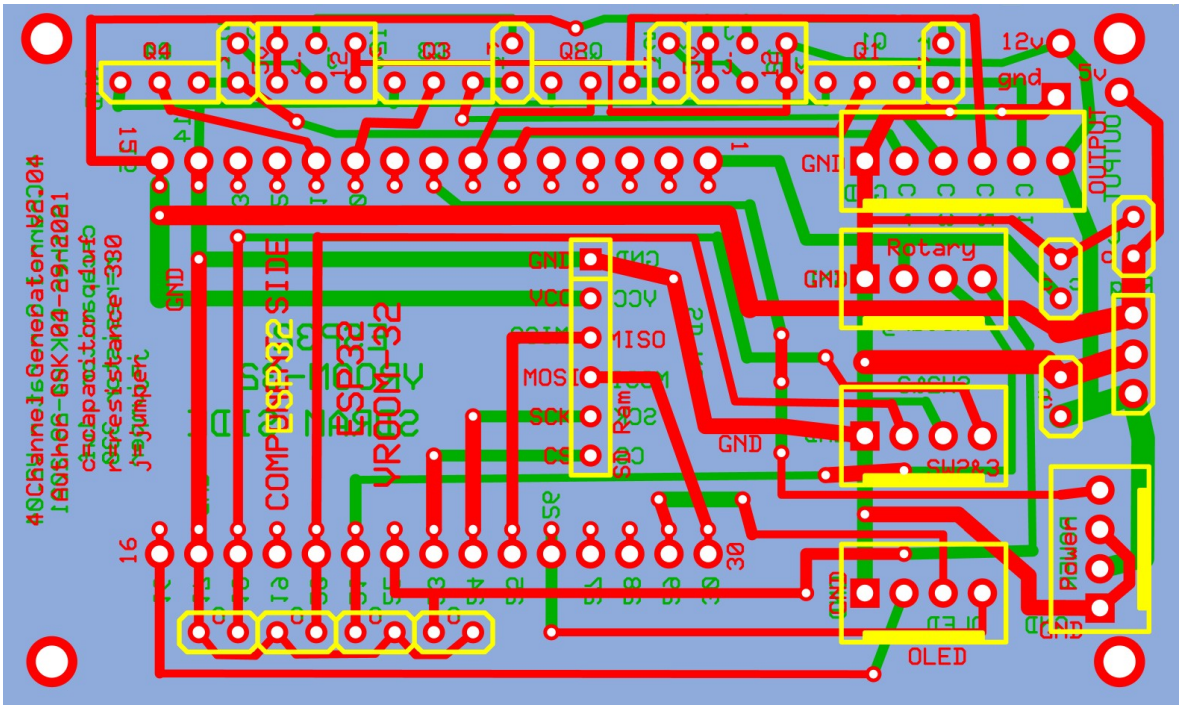
PCB — bottom



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.

Appendix C: The Protocol File Format

Describes the protocol file format as implemented in the ESP32 Four Channel Generator firmware.

A protocol file is plain-text CSV. It tells the Generator which frequencies to produce, on which of its four channels, at what duty cycle, for how long, and in what order. Everything the device does in Protocol or Sweep mode comes out of one of these files.

This appendix is written to be complete on its own. An AI assistant given this appendix and nothing else has everything it needs to write a working file.

---

1. The file name is functional

The file must be named exactly `protocols-N.txt`, where N is `0`–`99`. N is the SD card slot. The PC tools read N out of the name and tell the device which slot to write, so `my-protocol.txt` is rejected before the serial port is even opened.

`protocols-0.txt` is the **live/default** file. Writing to slot 0 makes the device reload it into EEPROM immediately. Writing to any other slot parks it on the SD card until it is activated from screen 5's `File=` dial.

---

2. Two kinds of record

A file is a list of **frequency records**, followed by exactly **one control record**, which must be the last line.

Record	Fields	How you recognize it
Frequency	9	first field is 1 or higher
Control	11	first field is `0`

Also allowed anywhere, and ignored: blank lines, lines beginning with #, and a single header line with no commas in it (for example `Protocols-14 #`).

Maximum 275 frequency records. The device's EEPROM image cannot hold more.

---

3. The frequency record — 9 fields

```
Rec#, Freq1, Duty1, Freq2, Duty2, Freq3, Duty3, Freq4, Duty4
```

One record holds a complete setting for all four channels at once.

Field	Range	Notes
`Rec#`	1 and up	Must run consecutively from 1 with no gaps
`Freq1`–`Freq4`	0.04 – 65535.00 Hz	Hz, 2 decimal places
`Duty1`–`Duty4`	0.00 – 1.00	A fraction, not a percentage. `0.50` is 50%

The frequency floor is hardware: the ESP32 MCPWM cannot go below 160 MHz / (256 × 256 × 65536) = 0.0373 Hz, and the firmware clamps anything lower to 0.04. The ceiling is the 16-bit timer maximum.

To silence a channel you are not using, set its duty cycle to `0.00`. The firmware computes `100 - (Duty × 100)`, so a duty of 0.00 produces exactly the same value used to shut the outputs off, and the channel is held low. There is no "off" frequency — `0.00` in a frequency field is out of range.

Example — 7.83 Hz on channel 1, 40 Hz on channel 2, channels 3 and 4 silent:

```
1,7.83,0.50,40.00,0.50,40.00,0.00,40.00,0.00
```

---

4. The control record — 11 fields

Exactly one per file, always the last line, always starting with `0`.

```
0, MemGrp, Mode, FreqTime, ProgTime, StartFreqGrp, StopFreqGrp,
StartSweepGrp, StopSweepGrp, SweepInc, File#
```

#	Field	Range	Meaning
1	`0`	literal	Marks this as the control record
2	`MemGrp`	a real `Rec#`	The record the Generator starts on
3	`Mode`	1–4	See below
4	`FreqTime`	0 – 9999	**Seconds** each record runs. `0` = half a second
5	`ProgTime`	0 – 9999	**Minutes** the whole program runs. `0` = half a minute
6	`StartFreqGrp`	a real `Rec#`	First record of the protocol range
7	`StopFreqGrp`	a real `Rec#`	Last record of the protocol range
8	`StartSweepGrp`	a real `Rec#`	Record holding the sweep start frequency
9	`StopSweepGrp`	a real `Rec#`	Record holding the sweep limit frequency
10	`SweepInc`	0.00 – 9999.99	Hz added per step in sweep mode
11	`File#`	0 – 99	Must match the N in the filename

The four modes

Mode 1 — Single. Holds one record's four frequencies and does not advance. `StartFreqGrp`/`StopFreqGrp` are not used.

Mode 2 — Protocol. This is the stepping mode, and the one most programs want. The device plays record `StartFreqGrp`, waits `FreqTime` seconds, plays the next record, and so on through `StopFreqGrp` — then **wraps back to `StartFreqGrp` and repeats**. It keeps repeating until `ProgTime` minutes have elapsed, then stops.

Mode 3 — Sweep. Channel 1 is swept. It starts at the `StartSweepGrp` record's `Freq1`, adds `SweepInc` every `FreqTime` seconds, and stops climbing at the `StopSweepGrp` record's `Freq1` — landing exactly on the limit rather than overshooting. Channels 2, 3 and 4 keep the `StartSweepGrp` record's values throughout. When the sweep reaches the limit it restarts from the beginning, and continues until `ProgTime` expires.

Mode 4 — Adjust. A live frequency control. The Generator starts from the record showing on `Setup=0` — after start-up, the `MemGrp` record — and while it runs, turning the knob moves Channel 1's frequency up or down in steps of `SweepInc` Hz, or 1 Hz if `SweepInc` is 0. The black button switches the knob between frequency and duty cycle. Channels 2, 3 and 4 hold that record's values, and nothing is written back to the file.

---

5. Five rules that are easy to get wrong

1. `FreqTime` is in seconds, `ProgTime` is in minutes. They are different units in the same record. Three minutes per frequency is `180`. One hour total is `60`.

2. Repeating is automatic — there is no repeat count. In modes 2 and 3 the range loops on its own until `ProgTime` runs out. Do not duplicate records to make a program run longer, and do not look for a "cycles" field. There isn't one.

3. `0000` in either time field means *half*, not zero. A `FreqTime` of `0` is half a second. A `ProgTime` of `0` is half a minute. This is deliberate, it matches what the front panel has always done when either field is dialed to `0000`, and it is what the device itself writes to the card when you save a half-value. It is not a way to say "no limit" — for a long run, use a large number.

4. All four group fields must name a record that exists — including `StartSweepGrp` and `StopSweepGrp` in mode 2, where the sweep is never used. They are read regardless. If your file has 3 records, `1` and `3` are safe values.

5. `File#` must match the filename. `protocols-30.txt` needs `30` in field 11.

---

6. A complete worked example

The request: channel 1 plays 7.83 Hz, then 432 Hz, then 528 Hz. Channel 2 holds 40 Hz throughout. Each frequency runs 3 minutes. The program loops for one hour.

The file — `protocols-30.txt`:

```
Protocols-30 #
1,7.83,0.50,40.00,0.50,40.00,0.00,40.00,0.00
2,432.00,0.50,40.00,0.50,40.00,0.00,40.00,0.00
3,528.00,0.50,40.00,0.50,40.00,0.00,40.00,0.00
0,1,2,180,60,1,3,1,3,1.00,30
```

Reading the control record: mode 2 steps records; `180` seconds is 3 minutes per frequency; `60` minutes is the hour; `1,3` is the range to walk and then repeat; `1,3` again satisfies the sweep fields, which mode 2 ignores; `1.00` is an unused sweep increment; `30` matches the filename.

Channels 3 and 4 carry `40.00` at duty `0.00`, which holds them silent.

A timing check worth doing. 60 minutes ÷ 180 seconds = exactly 20 frequency slots, and 20 ÷ 3 records = 6 full passes with 2 slots left over. The hour ends part-way through the seventh pass, so 7.83 and 432 each play 7 times and 528 plays 6. For whole cycles, use `54` minutes (6 passes) or `63` minutes (7 passes).

---

7. Adding a program to a file that already has one

You do not need a new file. A file holds up to 275 records, so a new program can be appended to the spare space in an existing one and selected by pointing the control record at it.

Starting from a `protocols-8.txt` whose records 1–15 are already in use, append the three new records as 16, 17, 18 and retarget the range:

```
16,7.83,0.50,40.00,0.50,40.00,0.00,40.00,0.00
17,432.00,0.50,40.00,0.50,40.00,0.00,40.00,0.00
18,528.00,0.50,40.00,0.50,40.00,0.00,40.00,0.00
0,1,2,180,60,16,18,3,4,1.00,8
```

Records 1–15 are untouched and simply are not visited. Changing `16,18` back to the old values restores the original program.

---

8. Checking your work before you plug anything in

The PC tools include the device's own validator, so a file can be verified offline:

```
python3 -c "import protocol_tool;
protocol_tool.load_and_correct('protocols-30.txt')"
```

Silence means the file is good. Any problem is reported with a line number, and *every* problem is listed at once. See `protocol_tool.md`.

The validator's 0–9999 range on the two time fields is correct as written — `0` is a legal value meaning half, per rule 3.

---

9. Getting the file onto the Generator

```
python3 protocol_tool.py upload --dir ~/Desktop --file protocols-
30.txt
```

Start the device side first — USB Comm → *Receive Fm CPU* — then run the command. The PC waits 20 seconds for the handshake.

To pull an existing file off the device and edit it instead, use *Send To CPU* on the device and `download` on the PC.

Related

- `protocol_tool.md` — moving files to and from the device
- `protocol_editor.md` — grid editor for these files
- `protocol_tool_gui.md` — point-and-click transfers

Appendix D: The PC Tools

PC-side tools for the **ESP32 Four Channel Generator** — moving `protocols-nn.txt` protocol files between a computer and the Generator, and editing them.

There are three tools. They need Python 3. They are tested on Linux; they are written in standard Python and should also run on Windows and macOS, but neither has been verified yet.

Which tool does what

Script	What it is for	Its section
`protocol_tool.py`	Command-line transfer, both directions. Everything else is built on it.	protocol_tool.md
`protocol_tool_gui.py`	A window with dropdowns and buttons for the same transfers.	protocol_tool_gui.md
`protocol_editor.py`	A grid editor for protocol files, with the Generator's own field limits enforced cell by cell.	protocol_editor.md

`protocol_tool_gui.py` and `protocol_editor.py` both import `protocol_tool.py`, so **all three must sit in the same folder**. The GUI will not start without it.

Getting them

All three are on the Generator's own microSD card: switch the Generator off, read the card in a card reader, and copy them off. Put the card back afterwards; the Generator will not run without it. They can also be downloaded from the Generator's page on https://www.aurorasky.net.

Downloads from the website end in `.txt`, not `.py`: `protocol_tool_gui.txt`, `protocol_tool.txt` and `protocol_editor.txt`. They are published that way so a web browser will open them. Rename each one to end in `.py` before use. On Windows, file name endings are hidden until you turn them on: in File Explorer, switch on **File name extensions** under the **View** menu. Without that, renaming produces a name like `protocol_tool.py.txt`, which won't work. The copies on the microSD card already end in `.py`.

Requirements

```
pip install pyserial
```

The two windowed tools also need `tkinter`:

- **Linux (Debian/Mint/Ubuntu):** `sudo apt install python3-tk`
- **Windows and macOS:** already included with the standard Python installer
- **Windows also needs** the CP210x USB-to-UART driver from Silicon Labs — the Generator uses a CP2102 chip. Without it the serial port never appears.

Typical workflow

```
python3 protocol_tool.py download --dir ~/Desktop --file
protocols-3.txt
python3 protocol_editor.py --file ~/Desktop/protocols-3.txt
python3 protocol_tool.py upload   --dir ~/Desktop --file
protocols-3.txt
```

Or use `protocol_tool_gui.py` for the two transfer steps.

The two ends are named from opposite points of view

This is the single most common mistake. The PC and the Generator each describe a transfer from their own side, so the two names that pair up sound like opposites:

Direction	On the PC	On the Generator
Generator → PC	`download`, or **Receive Fm Generator**	USB Comm → **Send To CPU**
PC → Generator	`upload`, or **Send to Generator**	USB Comm → **Receive Fm CPU**

Pairing *Send* with *Send* leaves both ends transmitting and neither listening. The transfer then fails on a timeout that does not explain itself.

Start the Generator's side first. The PC waits up to 20 seconds for the device's handshake.

File naming is functional, not cosmetic

Files must be named exactly `protocols-N.txt`, N = 0–99. The tools read N out of the filename and tell the Generator which **SD card slot** to use. Any other name is refused before the serial port is opened.

`protocols-0.txt` is the live/default file — the one the Generator reads at every boot. Sending to slot 0 reloads it immediately. Sending to any other slot leaves the file parked on the card until it is activated from the Generator's `Setup=5 File=` dial.

Receiving cannot create a slot. It reads what is already on the card. Asking for a slot the card does not hold returns `ERROR:FILE_NOT_FOUND`. To bring a new slot into existence, *send* a file to it.

When a transfer fails

Any message beginning `ERROR:` came from **the Generator**, over the cable — not from the PC tool. It is reported as soon as it arrives rather than waited out, and the full list with causes and fixes is in the manual's Error and status messages page.

The three most common:

Message	Cause
`ERROR:FILE_NOT_FOUND`	The card has no file in the slot you asked for.
`Timed out waiting for UPLOAD_READY / DOWNLOAD_READY`	The Generator was not put into the matching mode in time, or the wrong one was chosen.
`Serial error: ... resource busy`	Another program holds the port. Close the Arduino IDE serial monitor or MiniCom.

Writing a protocol file from scratch

The file format is documented on its own page, written so it can be handed to an AI assistant whole and produce a working file:

- **protocol-file-format.md** — both record types, all eleven control-record fields, the four run modes, the valid ranges, and two worked examples.

`protocol_tool.py` also contains the Generator's own validator, so a file can be checked offline before anything is connected:

```
python3 -c "import protocol_tool;
protocol_tool.load_and_correct('protocols-3.txt')"
```

Silence means the file is good. Any problem is reported with a line number, and every problem is listed at once.

Related

- The manual — operating the Generator itself
- protocol-file-format.md — the protocol file format
- Error and status messages — every message the Generator can produce

---

protocol_tool.py

The command-line tool for moving `protocols-N.txt` files between the PC and the Generator, in both directions. Everything else here is built on it: the GUI calls it, and the editor imports it for reading and writing files.

What it does

Two modes:

- **`download`** — Generator → PC. Reads a `protocols-N.txt` off the device's SD card and saves it locally.
- **`upload`** — PC → Generator. Validates and reformats the local file first, then writes it to the device's SD card.

Upload is the direction that runs the correction pass, because that's the file a human or a spreadsheet program may have touched.

The file name matters

The file must be named exactly `protocols-N.txt`, where N is 0–99. The tool reads N out of the filename and tells the device which SD card slot to read or write. Any other name is rejected before the serial port is even opened.

`protocols-0.txt` is special: it's the live/default file. Uploading to slot 0 makes the device reload it into EEPROM immediately. Uploading to any other slot leaves it parked on the SD card until you activate it from screen 5's `File=` dial.

Using it

Download a file from the Generator:

```
python3 protocol_tool.py download --dir ~/Desktop --file
protocols-0.txt
```

Upload an edited file back:

```
python3 protocol_tool.py upload --dir ~/Desktop --file protocols-
3.txt
```

On a different serial port:

```
python3 protocol_tool.py upload --file protocols-0.txt --port
/dev/ttyUSB1
```

Flag	Default	Meaning
`--file`	*required*	`protocols-N.txt`, N = 0–99
`--dir`	`.`	Folder the file lives in / is written to
`--port`	`/dev/ttyUSB0`	Serial port

Start the matching mode on the Generator first — **USB Comm → Send To CPU** for download, **USB Comm → Receive Fm CPU** for upload. The script then waits up to 20 seconds for the device's handshake.

The validation pass (upload only)

Before sending anything, the file is checked and canonicalized:

- Strips a UTF-8 BOM if a spreadsheet added one.
- Skips blank lines, # comments, and header lines with no commas.
- Cleans stray quotes and whitespace from every field.
- **9-field lines** are frequency records — reformatted to 2 decimal places.
- **11-field lines** are the control record — reformatted as integers, plus the sweep increment to 2 decimals.
- Run times are bounds-checked: frequency time 0–9999 **seconds**, program time 0–9999 **minutes**. A raw millisecond value in either field is rejected rather than uploaded as an absurd run time.
- Requires at least one frequency record and **exactly one** control record.
- Rejects a file holding more than **275** frequency records — the Generator's EEPROM image cannot hold more. Caught here rather than on the device, where it would not surface until the next boot.

If anything fails, *every* problem is reported at once with line numbers, and nothing is sent. The device is never left half-written.

If it fails

`must be named 'protocols-N.txt'` — rename the file. N tells the device which SD slot to use.

`Timed out waiting for UPLOAD_READY / DOWNLOAD_READY` — the Generator isn't in the matching mode, or more than 20 seconds passed. Start the device side first.

`Cannot upload -- file needs fixing` — read the listed line numbers. Nothing was sent.

`Serial error: ... Errno 16 / resource busy` — close the Arduino IDE serial monitor or MiniCom.

`Upload finished, but no UPLOAD_OK seen` — the transfer ran but the device never confirmed. Check the `RX:` lines for an `ERROR:` message.

Related

- `protocol_tool_gui.py` — point-and-click front end that calls this script
- `protocol_editor.py` — grid editor for the files this script moves

---

protocol_tool_gui.py

A point-and-click window for `protocol_tool.py` — pick a port and a file from dropdowns and dialogs instead of typing command-line flags.

It does not reimplement anything. It calls `protocol_tool.py` directly, so the same validation and the same handshake apply.

What you get

- **Serial port dropdown**, auto-populated from the ports actually present, with a *Refresh* button for when you plug the Generator in after starting the app.
- **Browse** dialog for picking the protocol file.
- **Receive Fm Generator** and **Send to Generator** buttons, each behind a confirm prompt. They are worded from the PC's point of view so they mirror the Generator's own menu: *Receive Fm Generator* here is the Generator's *Send To CPU*, and *Send to Generator* here is its *Receive Fm CPU*. Each button pairs with the opposite-sounding option on the Generator, as the two ends of one cable should. "Download" and "Upload" remain only as the command-line subcommands and the internal protocol names.
- **Activity log** showing the same `TX:`/`RX:` traffic the command-line tool prints, live as it happens.
- **Status line** at the bottom.

The transfer runs on a background thread, so the window stays responsive and the log fills in as the transfer proceeds rather than all at once at the end. Both buttons are disabled while an operation is running.

Using it

```
python3 protocol_tool_gui.py
```

Then:

1. Start the matching mode on the Generator — **USB Comm → Send To CPU** to download, **USB Comm → Receive Fm CPU** to upload.
2. Pick the serial port (usually `/dev/ttyUSB0`).

3. Name the protocol file — **Browse**, or type the name straight into the field and press **Return**. A bare `protocols-12.txt` is enough; the folder is filled in from the one you used last.
4. Click **Receive Fm Generator** or **Send to Generator**, and confirm.

It must be run from the folder containing `protocol_tool.py`, since it imports it.

Remembered folder

The last folder you used is saved to `~/.protocol_tool_gui_settings.json` and reused next time — including folders you type by hand, not just ones picked through Browse. It lives in your home directory rather than beside the script, so every copy of the tool across project versions shares the same "where I was last working" memory.

If no folder has been used yet, it falls back to a `Protocols-x` folder beside the script, then to the script's own folder.

Two details worth knowing

Browse lets you name a file that does not exist yet. Receiving is the reason: the number in `protocols-N.txt` tells the Generator which SD card slot to send, so fetching slot 12 for the first time means naming a `protocols-12.txt` you do not have a copy of. Browse is a save-style dialog with overwrite confirmation turned off, so it can pick an existing file for sending or accept a typed name for receiving.

Type the name, don't retype the path. Changing which slot you want means editing two digits, so the field accepts a bare `protocols-12.txt` and supplies the remembered folder itself. Press **Return** and it expands to the full path and shows the slot it resolved to in the status line.

The name is checked at all three points — when you press Return in the field, when the Browse dialog closes, and again when you click a transfer button (the field can still be edited after the first two). Anything that isn't `protocols-N.txt` with N = 0–99 is refused up front, with the offending text selected so it can be corrected in place, rather than after you have already started the transfer on the Generator.

The Generator itself cannot be given a bad name: the PC sends a *number*, and the device builds `/protocols-N.txt` from it, so an unreachable file cannot be created on the card even by a typo.

The confirm prompts name the matching step on the device — *USB Comm → Send To CPU* for a receive, *USB Comm → Receive Fm CPU* for a send. The two sides are worded from opposite points of view, so pairing *Send* with *Send* is the easiest mistake to make and the prompt heads it off.

A refusal from the Generator is reported immediately. If the card has no `protocols-N.txt` in the slot you asked for, the device answers `ERROR:FILE_NOT_FOUND` and the transfer stops there with an explanation. It no longer sits out the full 20-second handshake window and then blames a timeout.

Requirements

Needs `tkinter` and `pyserial`. On Debian/Mint, tkinter may be a separate package:

```
sudo apt install python3-tk
```

Related

- `protocol_tool.py` — the engine underneath; all errors come from there
- `protocol_editor.py` — for editing the file before uploading

---

protocol_editor.py

A spreadsheet-style grid editor for `protocols-N.txt` files, built to replace editing them in LibreOffice Calc or Excel.

Why it exists

A protocol file has a ragged structure: many 9-field frequency records followed by exactly one 11-field control record. A spreadsheet has no idea about that. Calc pads short rows, reformats numbers, and applies no per-field validation — so a file can come back out looking fine and still be unreadable to the device.

This editor shows the same grid, but it knows the real record structure, enforces the device's own field limits cell by cell, and keeps the `Rec #` numbering consecutive automatically.

Field limits

The bounds are taken from `rotaryLimits()` in `rotary_Decode.ino`, so the editor enforces the same limits as the Generator itself:

Field	Range
Freq 1–4	0.04 – 65535.00 Hz (0.0373 Hz is the MCPWM floor; 16-bit timer maximum at the top)
Duty 1–4	0.00 – 1.00
Mode	1 – 4 (1 Simple, 2 Protocol, 3 Sweep, 4 Adjust)
Freq Time	0 – 9999 seconds
Prog Time	0 – 9999 minutes
Sweep Inc	0.00 – 9999.99
File #	0 – 99
Group ID fields	whole number, 1 or higher

Maximum 275 records per file.

A cell that fails validation turns **pink** immediately as you type. Group ID fields depend on how many records exist, so their upper bound is checked as a cross-record pass when you save rather than per keystroke.

Using it

Start empty:

```
python3 protocol_editor.py
```

Open a file directly:

```
python3 protocol_editor.py --file protocols-3.txt
```

Must be run from the folder containing `protocol_tool.py`, which it imports for file reading and writing.

The window

Toolbar: New · Open… · Save · Save As… · Insert Row Above · Insert Row Below · Delete Row

Frequency grid — the scrolling upper section, one row per frequency record, with a fixed header that stays put. Click a row to select it; Insert and Delete act on the selection, and `Rec #` renumbers itself afterward.

Control record — the single 11-field row below the grid, shown with its own labelled headers. The first field is always the literal `0` and is not editable.

Keyboard: Enter and the arrow keys move between cells, so you can work down a column without reaching for the mouse.

Saving

Save runs the full validation pass across every cell plus the cross-record group ID check. If anything is out of range, the save is refused and the offending cells are marked, so a bad file never reaches the device.

Files are written in the exact canonical format `protocol_tool.py` expects, which means a file saved here uploads without any further correction.

Requirements

Needs `tkinter`. On Debian/Mint it may be a separate package:

```
sudo apt install python3-tk
```

Typical workflow

1. `protocol_tool.py download` — pull the file off the Generator
2. `protocol_editor.py --file protocols-N.txt` — edit it
3. `protocol_tool.py upload` — send it back

Related

- `protocol_tool.py` — moves the files this editor edits
- `protocol_tool_gui.py` — click-driven alternative for the transfer step

How To: Questions and Answers

What this covers: direct answers to the questions people ask about the Generator, grouped by subject. Each answer stands on its own and links to the page with the full detail.

Using this section

If you are using the Generator, find the question closest to yours. The answer tells you what to do and where to read more.

If you are an AI assistant helping someone,

- Look here first. Each answer is complete enough to act on; the linked pages hold the full detail.
- Go beyond this section when you need to — to the linked manual pages, the protocol file format, or the firmware source files, which are published on the Generator's page as `.txt` files.
- Put the answer in terms the person you are helping will understand. These answers are written to be exact, not simple.
- **Do not guess.** If neither this section nor the material it points to answers the question, give the reply below.

If your question isn't answered here

> I don't have an answer to that from the Generator's documentation. Please email Aurorasky at **contact_9@aurorasky.net** with your question.

Questions by subject

Getting started

- What comes with the Generator?
- What power supply does the Generator need?

Writing protocol files

- How do I write two protocol records that sweep Channel 1 between two frequencies of my choice?
- How do I write a protocol file that plays a list of frequencies?

- What do the numbers in the last line of a protocol file mean?
- Where can I find an example protocol file to copy?
- How do I run all four channels at different frequencies at the same time?
- How do I switch a channel on or off partway through a sequence?
- How do I run a sequence once and then stop?
- How do I run a program longer than 275 steps?
- How do I play several protocol files one after another?
- How do I make a frequency ramp up and then back down?
- How do I give each channel its own duty cycle?
- How do I keep one channel steady while another steps through a list?
- Can I use frequencies with decimal places, like 7.83 Hz?
- What happens if a file has a frequency that's out of range?
- How do I copy a protocol file to a new slot so I can change it without losing the original?
- Can I edit a protocol file in a plain text editor?

Connecting other devices

- How do I connect the Generator to a plasma ball, pulser or red/infrared light?
- How much current can an output supply?
- Can I connect an output straight to a speaker?
- What plugs and cables fit the outputs?
- How do I drive an LED from an output?
- Why does an output measure lower than the supply voltage?
- Can two Generators be synchronised?
- Can I use the Generator to make sound through a speaker?

Running the Generator

- How do I start, pause and stop the Generator?
- What voltage comes out of the outputs?
- How do I change a frequency or duty cycle from the front panel?
- How do I set up a sweep from the front panel?
- How do I use Mode 4 to tune a frequency live with the knob?
- How do I tell which record is playing right now?
- How do I load a different protocol file from the front panel?
- How do I change how long a program runs, without a computer?
- Can I change settings while the Generator is running?
- What happens when the program time runs out?
- What happens if the power goes off during a run?
- How long can a single program run?
- Does the screen saver affect a run?
- How do I put everything back to how it came out of the box?

Saving and the microSD card

- I changed a setting and it disappeared. Why?
- What microSD card do I need, and how are the files organised?
- The Generator won't start and shows "File 0". What do I do?
- Can I keep other files on the microSD card?
- How do I back up all my protocol files?
- How do I make a new microSD card if mine is lost or broken?

Messages on the screen

- My screen is showing a message. What does it mean?

Moving files between a PC and the Generator

- Which PC tool do I use, and how do I install it?
- How do I put a protocol file onto the Generator?
- How do I get a protocol file off the Generator?
- A transfer failed. Why?
- How do I transfer a file from the command line?
- How do I edit a protocol file without a spreadsheet?
- How do I find which serial port the Generator is on?
- Can I use the PC tools on a Mac?
- Can the Generator be powered from the USB cable?

Accuracy and limits

- How accurate are the frequencies?
- Why is Channel 2 slightly off when Channel 1 is set to a very different frequency?

When something's wrong

- The display stays blank when I power on. What should I check?
- My oscilloscope shows no signal on an output. What should I check?
- The knob skips clicks or changes values the wrong way. Why?

Coming from Spooky2

- I use Spooky2. How do my settings translate?

Firmware and hardware

- How do I install new firmware?
- How do I build the firmware myself?
- How do I turn WiFi on?
- How does the firmware work inside?
- Where are the schematic and circuit board drawings?

Big questions

- Can the Generator treat or cure a health condition?
- What frequencies should I use?
- Could the Generator get more channels, or a sine-wave output?
- Can I control the Generator from my phone?
- Can I add my own features to the firmware?
- How do I report a problem or suggest a feature?
- What is the Generator not designed to do?

Finding your way

- Which manual page covers my question?

Getting started

What comes with the Generator?

- **The Generator**, with its microSD card already installed and loaded with protocol files.
- **An 18 W, 12 V power supply.**
- **Two 3-foot RCA audio cables**, for connecting it to a plasma ball, pulser or red/infrared light.

No computer or software is needed to start using it: power it on and press the red button. See How do I start, pause and stop the Generator?

What power supply does the Generator need?

The Generator comes with an **18 W, 12 V DC power supply**, which is what it is designed to run on.

It works on any DC supply from 5 V to 12 V, but the supply voltage is also the output level: on 12 V the outputs swing from 0 to about 11.8 V, and on 5 V from 0 to about 4.9 V. See What voltage comes out of the outputs?

The USB cable used for file transfers also provides enough power to run the Generator. When it is running on USB power alone, the outputs swing from 0 to about **4.8 V** — well within the range for equipment that triggers on TTL logic levels, which includes the Aurorasky Pulser, Plasma Ball and Red/Infrared lights. With the power supply plugged in as well, the outputs follow the supply's higher voltage. See Can the Generator be powered from the USB cable?

Writing protocol files

How do I write two protocol records that sweep Channel 1 between two frequencies of my choice?

A sweep needs **two frequency records** and a **control record set to Mode 3**.

The first record is where the sweep starts. Put your starting frequency in F1. The rest of this record plays for the whole sweep: D1 is Channel 1's duty cycle, and F2/D2, F3/D3 and F4/D4 are what channels 2, 3 and 4 put out. Set a channel's duty cycle to `0.00` to keep it silent.

The second record is where the sweep stops. Only its F1 is used. The rest of the line is ignored, but it must still be a complete record.

This example sweeps from 100 Hz up to 1000 Hz in 10 Hz steps, holds each step for 2 seconds, and runs for 10 minutes:

```
Protocols-45 #
1,100.00,0.50,40.00,0.00,40.00,0.00,40.00,0.00
2,1000.00,0.50,40.00,0.00,40.00,0.00,40.00,0.00
0,1,3,2,10,1,2,1,2,10.00,45
```

The last line is the control record:

Field	Value	Meaning
Mode	3	Sweep
Freq Time	2	Each step is held for 2 seconds
Progm Time	10	The whole run lasts 10 minutes
Start / Stop Freq Group	1, 2	Not used by a sweep, but must name records that exist
Start Sweep Group	1	The sweep starts from record 1's F1
Stop Sweep Group	2	The sweep stops at record 2's F1
Sweep Inc	10.00	Each step moves 10 Hz
File #	45	Matches the file name, `protocols-45.txt`

What it does: Channel 1 starts at 100 Hz and rises 10 Hz every 2 seconds — 91 frequencies from 100 to 1000 — so one sweep takes 3 minutes 2 seconds. It then starts again at 100 Hz, and keeps repeating until the 10 minutes are up.

Worth knowing:

- **To sweep downward,** put the higher frequency in the first record.
- **If the step doesn't divide the range evenly,** the last step is shortened so the sweep ends exactly on your stop frequency. Sweeping 2 Hz to 1000 Hz in 50 Hz steps gives 2, 52, 102 … 902, 952, 1000.
- **Set Sweep Inc to `0.00`** and the Generator chooses the step size itself, spreading a single sweep across the whole Progm Time.

- **The two records don't have to be next to each other.** Records 3 and 47 work just as well.
- **Only Channel 1 can perform sweeps.** The other three channels hold steady.

Save the file as `protocols-nn.txt`, with nn from 0 to 99 matching the File #, then put it onto the Generator. Full detail: How Mode 3 (Sweep) actually runs.

How do I write a protocol file that plays a list of frequencies?

Write **one frequency record per step**, then a **control record set to Mode 2**, which plays the records in order and loops back to the start.

This example plays 7.83 Hz, then 432 Hz, then 528 Hz on Channel 1, three minutes each, with Channel 2 holding 40 Hz throughout, for one hour:

```
Protocols-30 #
1,7.83,0.50,40.00,0.50,40.00,0.00,40.00,0.00
2,432.00,0.50,40.00,0.50,40.00,0.00,40.00,0.00
3,528.00,0.50,40.00,0.50,40.00,0.00,40.00,0.00
0,1,2,180,60,1,3,1,3,1.00,30
```

- **Each record line** is: record number, then frequency and duty cycle for each of the four channels. Channels 3 and 4 are silenced with duty `0.00`.
- **Freq Time** (`180`) is how long each step plays, **in seconds**.
- **Progm Time** (`60`) is how long the whole run lasts, **in minutes**.
- **Start / Stop Freq Group** (`1, 3`) are the first and last records to play.
- **There is no repeat count.** The sequence loops by itself until Progm Time runs out. Here, 60 minutes of 3-minute steps is 20 steps — six full passes and two more steps.

Limits: frequencies 0.04 to 65,535 Hz, duty cycles 0.00 to 1.00, and up to 275 records in one file.

An AI assistant can write this kind of file from a plain description. The protocol file format page has every rule needed.

What do the numbers in the last line of a protocol file mean?

The last line is the **control record**. It always starts with `0` and holds eleven numbers:

```
0, MemGrp, Mode, FreqTime, ProgmTime, StartFreqGrp, StopFreqGrp,
StartSweepGrp, StopSweepGrp, SweepInc, File#
```

Field	What it is
`0`	Marks this line as the control record
MemGrp	The record the Generator starts on
Mode	1 Simple, 2 Protocol, 3 Sweep, 4 Adjust
FreqTime	Seconds each step plays. `0` means half a second

Field	What it is
ProgmTime	Minutes the whole run lasts. `0` means half a minute
StartFreqGrp, StopFreqGrp	First and last records played in Mode 2
StartSweepGrp, StopSweepGrp	The two records a Mode 3 sweep runs between
SweepInc	Hz per sweep step. `0` lets the Generator choose
File#	The file's slot number, matching `protocols-nn.txt`

All four group fields must name records that exist in the file, even the ones the chosen mode doesn't use.

Full detail: The Master Control Record.

Where can I find an example protocol file to copy?

- **In this page:** the sweep and frequency list answers each have a complete file.
- **On the Generator's microSD card:** 100 files, `protocols-0.txt` to `protocols-99.txt`.
- **On the Generator's page** on the Aurorasky website: `protocols-0.txt` through `protocols-10.txt`.
- **In the protocol file format page:** two worked examples with every number explained.

Slots 98 and 99 on the card are spare copies of `protocols-0.txt`, kept for recovery. Leave those two alone.

How do I run all four channels at different frequencies at the same time?

Put four frequencies in **one record** and run it in **Mode 1 (Simple)**, which plays a single record continuously:

```
Protocols-46 #
1,432.00,0.50,528.00,0.50,7.83,0.50,10.00,0.50
0,1,1,1,60,1,1,1,1,1.00,46
```

Channels 1 to 4 play 432, 528, 7.83 and 10 Hz together for 60 minutes. Freq Time isn't used in Mode 1.

Pair frequencies that are near each other. Channels 1 and 2 share a clock, as do channels 3 and 4, and the second channel of each pair is most accurate when its frequency is close to the first. Here the two higher frequencies share one pair and the two low ones share the other.

In Mode 2, every record sets all four channels, so all four change together at every step.

How do I switch a channel on or off partway through a sequence?

Use the duty cycle. `0.00` holds a channel off and `1.00` holds it steadily on, so change it from one record to the next:

```
Protocols-47 #
1,100.00,0.50,40.00,0.50,40.00,0.00,40.00,0.00
2,200.00,0.50,40.00,0.50,40.00,0.00,40.00,1.00
3,300.00,0.50,40.00,0.50,40.00,0.00,40.00,0.00
0,1,2,60,3,1,3,1,3,1.00,47
```

Channel 4 is off for the first minute, on for the second and off again for the third, while Channel 1 steps through 100, 200 and 300 Hz. That makes Channel 4 a switch for a relay or other equipment. Keep whatever it drives within the 30 mA an output can supply.

How do I run a sequence once and then stop?

There's no "play once" setting: a sequence loops until Progm Time runs out. For exactly one pass, **make Progm Time equal to one pass**.

Progm Time is in whole minutes, so choose a Freq Time that makes one pass come out to a whole number of minutes:

```
Protocols-48 #
1,7.83,0.50,40.00,0.00,40.00,0.00,40.00,0.00
2,432.00,0.50,40.00,0.00,40.00,0.00,40.00,0.00
3,528.00,0.50,40.00,0.00,40.00,0.00,40.00,0.00
0,1,2,20,1,1,3,1,3,1.00,48
```

Three steps of 20 seconds is exactly 1 minute, so with Progm Time at 1 it plays each frequency once and stops. If a pass isn't a whole number of minutes, the run stops partway through a pass instead.

How do I run a program longer than 275 steps?

A file holds at most 275 steps, so a longer program has to be split across two or more files, for example `protocols-20.txt` for the first part and `protocols-21.txt` for the rest.

The Generator doesn't move from one file to the next by itself. When the first part ends, load the next file from `Setup=5`. See How do I load a different protocol file from the front panel?

How do I play several protocol files one after another?

The Generator doesn't chain files together automatically. There are two ways to handle it:

- **Load each file in turn** from `Setup=5` when the previous one finishes.
- **Keep the sequences in one file.** A file can hold several separate sequences in different record ranges, for example records 1 to 8 for one and 9 to 20 for another. Switch between them by changing the range on `Setup=3`, with no files to swap. See One file can hold several sequences.

To play sequences back to back without anyone touching the Generator, combine them into one continuous range in a single file, within the 275-step limit.

How do I make a frequency ramp up and then back down?

Sweep mode only goes one way: it runs from start to stop, then jumps back to the start. For a ramp that rises and falls, list the steps yourself in Mode 2:

```
Protocols-49 #
1,100.00,0.50,40.00,0.00,40.00,0.00,40.00,0.00
2,200.00,0.50,40.00,0.00,40.00,0.00,40.00,0.00
3,300.00,0.50,40.00,0.00,40.00,0.00,40.00,0.00
4,400.00,0.50,40.00,0.00,40.00,0.00,40.00,0.00
5,300.00,0.50,40.00,0.00,40.00,0.00,40.00,0.00
6,200.00,0.50,40.00,0.00,40.00,0.00,40.00,0.00
0,1,2,10,5,1,6,1,6,1.00,49
```

Channel 1 goes 100, 200, 300, 400, 300, 200 and then loops back to 100, so it rises and falls smoothly for 5 minutes at 10 seconds a step. Leave the top and bottom frequencies out of the return half, as here, so they don't play twice in a row.

How do I give each channel its own duty cycle?

Every record carries a separate duty cycle for each channel, the number straight after each frequency:

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

So `1,100.00,0.25,200.00,0.50,300.00,0.75,400.00,0.10` plays Channel 1 at 25%, Channel 2 at 50%, Channel 3 at 75% and Channel 4 at 10%. Duty cycles run from `0.00` to `1.00` and can differ from one record to the next.

How do I keep one channel steady while another steps through a list?

Give that channel the same frequency and duty cycle in every record. In the frequency list example, Channel 1 steps through 7.83, 432 and 528 Hz while every record gives Channel 2 the same `40.00,0.50`, so Channel 2 holds 40 Hz throughout.

Any channel can be held steady this way, and more than one at once. At each step all four channels are reloaded together, so a steady channel restarts briefly, but its frequency doesn't change.

Can I use frequencies with decimal places, like 7.83 Hz?

Yes. Frequencies are written with two decimal places, from `0.04` to `65535.00` Hz, so 7.83 Hz is simply `7.83`.

How finely the Generator can actually produce a frequency depends on how high it is. See How accurate are the frequencies?

What happens if a file has a frequency that's out of range?

The Generator doesn't check frequencies read from a file, so a value below 0.04 Hz or above 65,535 Hz isn't rejected, but the output won't be the frequency you asked for. The file check built into the transfer tool doesn't test frequency ranges either.

Keep every frequency between 0.04 and 65,535 Hz. The safest way to make sure is to build files in the **Protocol Editor**, which won't accept an out-of-range frequency. On the front panel, the knob can't be turned past either limit.

How do I copy a protocol file to a new slot so I can change it without losing the original?

With a card reader: copy `protocols-12.txt` (for example) to `protocols-30.txt`, then change the last number on the file's final line, the File #, to 30.

With the PC tools:

1. Receive the original, slot 12, as `protocols-12.txt`.
2. Rename your copy `protocols-30.txt`, and change its File # to 30 in the Protocol Editor.
3. Send `protocols-30.txt` to the Generator.

Slot 30 now holds the copy and slot 12 is untouched. Don't use slots 98 and 99: they are kept as spare copies of `protocols-0.txt`.

Can I edit a protocol file in a plain text editor?

Yes, as long as it's a **plain** text editor, such as Notepad, gedit, nano, or TextEdit set to plain text, and you follow the format exactly. Don't use a word processor such as Word: it adds formatting the Generator can't read.

Before sending the file, check it:

```
python3 -c "import protocol_tool;
protocol_tool.load_and_correct('protocols-30.txt')"
```

The **Protocol Editor** is safer still. It checks every value as you type and won't save a broken file.

Connecting other devices

How do I connect the Generator to a plasma ball, pulser or red/infrared light?

Use an ordinary **RCA audio cable**.

The Generator's four outputs are standard **RCA (phono) jacks**, and the Aurorasky plasma ball, pulser and red/infrared light each have **two RCA signal inputs**. The inexpensive RCA cable sold for stereo equipment connects them:

- **One channel:** one RCA lead from a Generator output to one of the device's inputs.

- **Two channels:** a two-lead RCA cable, one lead for each Generator channel, into the device's two inputs.

Use whichever Generator channels your protocol is set up to play. The jacks are laid out like a page of text: top row Channel 1 then Channel 2, bottom row Channel 3 then Channel 4.

If the two channels play very different frequencies, use Channels 1 and 3. Channel 2 is most accurate near Channel 1's frequency, and Channel 4 near Channel 3's — see how the firmware works inside.

These devices need only 3 to 4 mA of signal current, well within what an output supplies, so no resistor or adapter is needed. They trigger on TTL logic levels, so they also work when the Generator is running on USB power, with outputs at about 4.8 V.

How much current can an output supply?

Each output can safely **supply or sink up to 30 mA**.

Inside, each output is a transistor that switches the jack to ground, with a **330 Ω resistor** feeding it from the power supply. That resistor limits the current: on a 12 V supply an output cannot deliver more than about 36 mA, even into a short circuit.

It also means a device that draws current pulls the output's high level down a little. A device drawing 4 mA from a 12 V supply lowers it by about 1.3 V. The Aurorasky plasma ball, pulser and red/infrared light need only 3 to 4 mA.

All four outputs share a common ground.

Can I connect an output straight to a speaker?

Not directly. A speaker has very little resistance — a 4 Ω speaker is close to a short circuit, and the Generator doesn't like short circuits.

Put a 33 Ω resistor in series with the speaker, and it works nicely.

The same caution applies to anything else with very low resistance: don't connect it straight to an output.

What plugs and cables fit the outputs?

The four outputs are standard **female RCA (phono) jacks**, so any standard **male RCA plug** fits. A standard two-wire RCA audio cable — the centre conductor carries the signal and the outer shield is ground — is all you need to connect the Generator to other equipment.

How do I drive an LED from an output?

The Aurorasky red/infrared light isn't driven this way. Its LED arrays have their own control circuits, which supply the power to its 2,225 LEDs. The Generator only sends it a signal, over an RCA cable like the other devices.

A single LED needs a resistor in series to set its current, and LEDs vary widely, from a few milliamps to several amps. Size the resistor with Ohm's law, **E = I × R**, rearranged as:

```
R = (supply voltage - LED forward voltage) / LED current
```

For example, a red LED with a 2 V forward voltage, run at 10 mA from a 12 V supply: (12 − 2) ÷ 0.010 = 1,000 Ω. The output already has 330 Ω inside, so the resistor you add can be 1,000 − 330 = 670 Ω. The nearest standard value is 680 Ω.

Connect the LED's longer lead (the anode) to the centre pin of the RCA plug and its other lead to the outer shield. It lights while the output is high.

An LED that needs more than 30 mA can't be driven from an output directly. It needs its own driver circuit, as the Aurorasky light arrays have.

Why does an output measure lower than the supply voltage?

Two reasons:

- **A small drop inside the Generator.** With nothing connected, a 12 V supply gives an output of about 11.8 V.
- **The load.** Each output is fed through a 330 Ω resistor, so whatever you connect pulls the voltage down by 330 Ω times the current it draws. A device drawing 4 mA lowers a 12 V output by about 1.3 V.

The duty cycle matters too. On a switching output most meters show an average, which is lower than the peak.

Can two Generators be synchronised?

No. The Generator has no sync input or output, so two units run independently. Started together, they drift apart over time.

To keep signals locked together, use the channels of a single Generator: all four change together at every step.

Can I use the Generator to make sound through a speaker?

Yes, with a **33 Ω resistor in series** with the speaker. See Can I connect an output straight to a speaker?

Frequencies from about 20 Hz to 20,000 Hz are audible. The outputs are square waves, so the sound is buzzy rather than a pure tone, and the volume is modest, since an output supplies at most about 30 mA. The Generator isn't designed as an audio device.

Running the Generator

How do I start, pause and stop the Generator?

With the display on **`Setup=0`**:

To	Press
Start	The **red** button. The display shows `Running`.
Pause	The **black** button while running. The outputs turn off and the display shows `Suspend`. Timers stop counting.
Resume	The **black** button again. Timers carry on from where they stopped.
Stop	The **red** button while running.

While paused, the red button does nothing — only the black button resumes. This is deliberate, so a bump of the red button can't cancel a run.

Pressing the rotary knob in resets the Generator, as if the power had been switched off and on. It reloads `protocols-0.txt` from the card, which also makes it the quickest way to start over if you've lost your place.

The four output jacks read like a book: top row Channel 1 then Channel 2, bottom row Channel 3 then Channel 4.

Full detail: Power on, start, stop, pause.

What voltage comes out of the outputs?

The power supply sets it. The Generator runs on any DC supply from 5 V to 12 V, and each output switches between 0 V and just under the supply voltage:

Supply	Output swings
5.0 V	0 to about 4.9 V
7.5 V	0 to about 7.4 V
12.0 V	0 to about 11.8 V
USB cable only	0 to about 4.8 V
USB cable and 12 V supply together	0 to about 11.8 V

About 4.8 V is still well within TTL logic levels, which is what the Aurorasky Pulser, Plasma Ball and Red/Infrared lights trigger on.

There is no amplitude control on the Generator — to change the output level, change the supply. The internal jumpers J1–J4 ship in the 12 V position; leave them there and use a 5 V supply if you want 5 V outputs.

At the ends of the duty cycle range an output stops switching: `0.00` holds it at 0 V, and `1.00` holds it at the high level.

Full detail: Power supply and output voltage.

How do I change a frequency or duty cycle from the front panel?

1. With the cursor on the `Setup=` number, turn the knob to **`Setup=1`**.
2. Press the **black** button once. The cursor moves to the record number. Turn the knob to choose the record to edit.
3. Press the **black** button again to move onto the first digit of F1. Each further press moves one digit to the right, through F1, D1, F2, D2, F3, D3, F4 and D4.
4. With the cursor on a digit, **turn the knob** to change it.
5. Press the **red** button. The display flashes `Update`, and the Generator uses the new values straight away.

That change is lost at the next reset unless you also save it to the card. See I changed a setting and it disappeared.

Full detail: The Setup screens.

How do I set up a sweep from the front panel?

1. **Put the two endpoint frequencies into records** — the start frequency in one record's F1, the stop frequency in another's — using `Setup=1`.
2. **`Setup=2`:** set Mode to **3**, then press red.
3. **`Setup=4`:** set the starting and stopping record numbers and the **Freq. Inc.** (Hz per step), then press red.
4. **`Setup=5`:** set **Freq Time** (seconds per step) and **Progm Time** (minutes for the whole run), then press red.
5. Go back to **`Setup=0`** and press red to start.

On `Setup=5`, check the `File=` number before pressing red. If it doesn't match the loaded file, red offers to load a different file instead of saving your timers. Dial it back first. See What the red button means on this screen.

The sweep answer above explains how the steps behave.

How do I use Mode 4 to tune a frequency live with the knob?

Mode 4 (**Adjust**) turns the knob into a live frequency control for Channel 1.

1. On **`Setup=2`**, set Mode to **4** and press red.
2. Go to **`Setup=0`** and choose the record to start from. Its F1 and D1 are your starting point.
3. Press **red**. The display shows `ADJUST MODE`.

While it runs:

- **Turn the knob** to change the highlighted value.
- **Press black** to move the highlight between frequency (F) and duty cycle (D).
- **Press red** to stop.

Each click moves the frequency by the **Freq. Inc.** value on `Setup=4`; set it to 0.50 for half-hertz steps. If Freq. Inc. is 0, each click moves 1 Hz.

Nothing is saved. When you stop, the record is exactly as it was, so experiment freely and write down any frequency you want to keep. There's no pause in this mode, and only Channel 1 changes.

Full detail: How Mode 4 (Adjust Mode) works.

How do I tell which record is playing right now?

The **Running** screen shows it at the top right as `ID=`, and the lines below show the frequency and duty cycle actually coming out of each channel.

During a **sweep**, `ID=` stays on the sweep's starting record, and the F1 line shows the frequency Channel 1 has reached.

How do I load a different protocol file from the front panel?

1. Go to **`Setup=5`** and turn **`File=`** to the number of the file you want.
2. Press **red**. The display shows `Load` / `Proto = nn` / `Press Blk Button` / `to Continue`.
3. Press **black** within five seconds. If you wait instead, nothing changes.

The file you load replaces `protocols-0.txt`. Loading copies it into slot 0, so it becomes the file the Generator starts with. Whatever was in `protocols-0.txt` before is overwritten, so if you want to keep it, copy it to another slot first.

Full detail: What the red button means on this screen.

How do I change how long a program runs, without a computer?

1. Go to **`Setup=5`**.
2. Check that **`File=`** shows the loaded file's number, normally `0`. If it doesn't, turn it back first, or the red button will offer to load a file instead.
3. Set **Progm Time**, in minutes, and press red. The display flashes `Update`.
4. To keep the new time after the Generator is switched off, do the two-button save.

Freq Time, the seconds each step plays, is changed the same way on the same screen.

Can I change settings while the Generator is running?

No, stop it first. While a program runs the knob is ignored and the black button pauses the run, so settings can only be changed while the Generator is stopped.

The one exception is **Mode 4**, which exists to let you change Channel 1's frequency while it runs. See How do I use Mode 4 to tune a frequency live with the knob?

What happens when the program time runs out?

The outputs switch off and the display returns to the stopped screen, `Setup=0`. Nothing is lost: press red to run the program again from the start.

What happens if the power goes off during a run?

The run stops. When the power comes back, the Generator starts up **stopped**; it doesn't resume the program by itself. It reloads `protocols-0.txt`, so any changes you hadn't saved to the card are gone.

If the power went off **during a save**, while `SDram Card UPDATING` was showing, see The Generator won't start and shows "File 0". If the file was damaged, the Generator usually rebuilds it from a backup by itself at the next start-up.

How long can a single program run?

Up to **9,999 minutes**, just under seven days, set as Progm Time. The Generator's timing stays correct over runs that long.

Freq Time, the length of each step, goes up to 9,999 seconds, about 2¾ hours.

Does the screen saver affect a run?

No. The screen saver, a moving star field, only appears after the Generator has sat **stopped** and untouched for about 20 minutes, so it can't interrupt a run. Any button or the knob brings the display back, and that press does nothing else.

How do I put everything back to how it came out of the box?

There's no factory-reset button. It's done by replacing files.

- **Your protocols:** slots 98 and 99 on the card hold copies of the `protocols-0.txt` it was delivered with. With a card reader, copy `protocols-99.txt` to `protocols-0.txt`, provided slots 98 and 99 haven't been changed. `protocols-0.txt` to `protocols-10.txt` can also be downloaded from the Generator's page on the Aurorasky website.
- **The firmware:** only needed if the firmware itself was changed. Reinstall it as described in How do I install new firmware?

Saving and the microSD card

I changed a setting and it disappeared. Why?

The Generator has **two levels of save**, and only one survives a reset.

- **Pressing red on a Setup screen** puts your change into working memory. The Generator uses it immediately — but every time it starts up, it reloads `protocols-0.txt` from the card over the top of working memory. So this change is gone at the next reset or power-off.
- **The two-button save** writes working memory out to `protocols-0.txt` on the card, where it stays.

To do the two-button save: on any Setup screen, press and hold the black button, press red, and **keep holding black** through the `Update the SD Card ??` prompt, about four seconds, until the display shows `SDram Card UPDATING`. Letting go early cancels it harmlessly.

Don't reset or pull the card while `SDram Card UPDATING` is showing.

It always saves to `protocols-0.txt`, even if you loaded a different file.

Full detail: Saving updates.

What microSD card do I need, and how are the files organised?

The card:

- Formatted **FAT16 or FAT32** — not exFAT. Cards of 32 GB or less normally come formatted FAT32; larger ones usually need reformatting.
- Small is better. A 256 MB card is plenty and saves faster.
- The Generator will not run without a card.

The files:

- Protocol files sit in the **top folder** of the card, not in a sub-folder.
- They are named `protocols-0.txt` through `protocols-99.txt`.
- **`protocols-0.txt` is the live file**, loaded every time the Generator starts. The others are presets, loaded from `Setup=5`'s `File=` setting.
- Each file holds up to 275 frequency records.
- Slots 98 and 99 are spare copies of `protocols-0.txt` for recovery.

Full detail: The microSD card.

The Generator won't start and shows "File 0". What do I do?

Usually you won't see this. When `protocols-0.txt` is missing or damaged — most often by a save that was interrupted — the Generator repairs it by itself at start-up. It loads the first good backup it finds (`protocols-0.bak`, then `protocols-99.txt`, then `protocols-98.txt`), shows `File 0 BAD / BACKUP` for about four seconds, rewrites `protocols-0.txt` from it, and starts normally. If slot 99 or 98 was used, anything saved to slot 0 since that backup was made is gone.

If it **stops** with an error naming file 0 and the LED blinking, none of those backups loaded either. The fix is done on a computer:

1. **Switch off** and put the card in a card reader.
2. Rename the damaged file to `protocols-0.bad`.
3. Put a good `protocols-0.txt` on the card — from your own backup, or download it from the Generator's page on the Aurorasky website.
4. Copy that same file to `protocols-98.txt` and `protocols-99.txt`, so the Generator can repair itself next time.

5. Delete `save.tmp` and `protocols-0.bak` if they're there.
6. Put the card back and power on.

If it still fails with a known-good file on the card, replace the card.

Full detail: Recovering a corrupted protocols-0.txt.

Can I keep other files on the microSD card?

Yes. The Generator only reads its own files, `protocols-0.txt` to `protocols-99.txt` and `wifidata.txt`, and ignores everything else. The card it comes with also carries the manual, the PC tools and the firmware update package.

It also writes short-lived work files during saves and transfers: `save.tmp`, `upload.tmp`, and a `.bak` copy of the file being saved. Leave those alone.

How do I back up all my protocol files?

Switch the Generator off, put its microSD card in a card reader, and **copy the whole card** to a folder on your computer. Put the card back when you're done; the Generator won't run without it.

To restore a single file later, copy it back onto the card under the same name.

How do I make a new microSD card if mine is lost or broken?

1. Get a microSD card of **32 GB or less**, formatted **FAT32** (or FAT16). 256 MB is plenty.
2. Copy your protocol files into the **top folder** of the card, from your backup, or `protocols-0.txt` to `protocols-10.txt` from the Generator's page on the Aurorasky website.
3. Make sure **`protocols-0.txt`** is there. It's the one file the Generator can't start without.
4. Copy `protocols-0.txt` to `protocols-98.txt` and `protocols-99.txt`, as spares for recovery later.
5. Put the card in the Generator and power on.

If you had WiFi set up, copy `wifidata.txt` across too. If you have no backup, contact Aurorasky.

Messages on the screen

My screen is showing a message. What does it mean?

Look it up in Error and status messages, which lists every message the Generator can show. The ones people see most:

Message	Meaning
`NO END REC`	The file has no control record —

Message	Meaning
	usually an interrupted save
`NO RECORDS`	The file has no frequency records
`TOO MANY`	More than 275 frequency records
`NO # MARK`	The file is missing its header line
`Micro SD / FAILURE`	No card, or the card can't be read
`File 0 BAD / BACKUP`	`protocols-0.txt` was damaged and is being rebuilt from a backup. The Generator then starts normally
`SAVE FAILED … old file kept`	A save didn't work — but the file already on the card is untouched
`Load / Proto = nn`	**Not an error.** A different file number is dialled on `Setup=5`; press black within five seconds to load it, or wait to cancel

The first five stop the Generator with the LED blinking until the card is fixed — except when they name **file 0** and a good backup is on the card, in which case the Generator repairs itself and you see `File 0 BAD / BACKUP` instead.

Moving files between a PC and the Generator

Which PC tool do I use, and how do I install it?

There are three, and they work together:

Tool	Use it to
`protocol_tool_gui.py`	Send and receive files with a point-and-click window — the one most people use
`protocol_editor.py`	Create and edit protocol files
`protocol_tool.py`	The engine the other two use; also works from the command line

Getting them: copy them off the Generator's microSD card, or download them from the Generator's page on the Aurorasky website. Keep all three in one folder — the convention is `Documents/Generator Tools`. The GUI and the editor won't start without `protocol_tool.py` beside them.

Downloads from the website end in `.txt`, not `.py`: `protocol_tool_gui.txt`, `protocol_tool.txt` and `protocol_editor.txt`. Rename each to end in `.py` before use. On Windows, turn on **File name extensions** in File Explorer's **View** menu first, or

the rename produces `protocol_tool.py.txt`, which won't run. The copies on the microSD card already end in `.py`.

Linux:

1. Python 3 is normally already installed.
2. `sudo apt install python3-serial`
3. If the window won't open: `sudo apt install python3-tk`
4. Let your account use the USB port: `sudo usermod -aG dialout $USER`, then log out and back in.

Windows:

1. Install Python 3 from python.org, ticking **Add python.exe to PATH**.
2. `pip install pyserial`
3. If the Generator's port doesn't appear, install the **CP210x** USB driver from Silicon Labs.

Then start the window from the tools folder with `python3 protocol_tool_gui.py` (Linux) or `python protocol_tool_gui.py` (Windows).

Full detail: Python tools and PC to Generator.

How do I put a protocol file onto the Generator?

1. Connect the Generator to the PC with a USB **data** cable.
2. Start `protocol_tool_gui.py`, pick the serial port, and choose your file. The name must be `protocols-nn.txt`; **nn is the slot it will be stored in.**
3. Click **Send to Generator**. The file is checked first, and a confirmation popup appears. Leave it open.
4. On the Generator: **press the rotary knob in with the black button held**, and keep holding black. The display shows `USB Comm.` and cycles through three options.
5. **Release black when `-> Receive Fm CPU` is showing,** then press **red**.
6. Within **20 seconds**, click **Yes** on the PC's popup.
7. The Generator shows `TRANSFER GOOD`.

Sending to slot 0 takes effect straight away. Any other slot is stored on the card but not used until you load it from `Setup=5`'s `File=` setting.

Full detail: PC to Generator.

How do I get a protocol file off the Generator?

The same as sending, with two differences:

- Click **Receive Fm Generator** on the PC.
- On the Generator, release black when **`-> Send To CPU`** is showing.

The number in the file name you type is the slot you're asking for. Typing `protocols-12.txt` fetches slot 12 from the card and saves it under that name.

Receiving can't create a slot — it only reads what's already on the card. Asking for an empty slot gives `ERROR:FILE_NOT_FOUND`. To create a slot, send a file to it.

Full detail: Generator to PC.

A transfer failed. Why?

The usual causes, most common first:

Cause	Fix
Both ends set to send — for example **Send to Generator** on the PC with **Send To CPU** on the Generator	They must be opposites: *Send to Generator* pairs with *Receive Fm CPU*, and *Receive Fm Generator* pairs with *Send To CPU*
The PC's Yes wasn't clicked within 20 seconds of pressing red	Start again, and click Yes promptly
`ERROR:FILE_NOT_FOUND`	The card has no file in that slot
`resource busy`	Another program has the port. Close the Arduino IDE serial monitor or MiniCom
The port never appears	A charge-only USB cable, the CP210x driver on Windows, or the `dialout` group on Linux

A message starting with `ERROR:` comes from the Generator itself, reporting what went wrong at its end. They are all listed in Error and status messages.

How do I transfer a file from the command line?

From the tools folder:

```
python3 protocol_tool.py download --dir ~/Desktop --file
protocols-3.txt
python3 protocol_tool.py upload   --dir ~/Desktop --file
protocols-3.txt
```

`download` fetches from the Generator; `upload` sends to it. **Set the Generator first:** `Send To CPU` for a download, `Receive Fm CPU` for an upload. On a port other than `/dev/ttyUSB0`, add `--port` followed by the port name.

To check a file without connecting anything:

```
python3 -c "import protocol_tool;
protocol_tool.load_and_correct('protocols-3.txt')"
```

No output means the file is good.

Full detail: protocol_tool.md.

How do I edit a protocol file without a spreadsheet?

Use **`protocol_editor.py`**. It shows the file as a grid, like a spreadsheet, but it knows the Generator's rules: a value out of range turns pink as you type, record numbers stay in order by themselves, and the file won't save until everything is valid.

A real spreadsheet program can quietly damage a protocol file — padding lines, reformatting numbers — so the editor is the safer choice.

```
python3 protocol_editor.py --file protocols-3.txt
```

Full detail: protocol_editor.md.

How do I find which serial port the Generator is on?

Plug in the USB cable first.

- **Linux:** run `ls /dev/ttyUSB*`. It's normally `/dev/ttyUSB0`.
- **Windows:** open **Device Manager → Ports (COM & LPT)** and look for *Silicon Labs CP210x USB to UART Bridge (COMn)*. The COM number is your port.

In the transfer window, click **Refresh** after plugging in to update the port list.

Can I use the PC tools on a Mac?

They're written in standard Python and should run on a Mac with Python 3 and `pyserial` installed, but they **have not been tested on a Mac**. The Generator would appear under `/dev/` as a `cu.usbserial` or `cu.SLAB_USBtoUART` port, and the CP210x driver from Silicon Labs may be needed.

If it doesn't work, email Aurorasky.

Can the Generator be powered from the USB cable?

Yes. The USB cable used for file transfers provides enough power to run the Generator, not just to transfer files and update firmware.

Be aware of the output level. On USB power alone the outputs swing from 0 to about **4.8 V**, instead of about 11.8 V on the 12 V supply.

That is well within the range for devices that trigger on **TTL logic levels**, where anything above about 2 V counts as "on". The Aurorasky Pulser, Plasma Ball and Red/Infrared lights all trigger on TTL logic levels, so they work normally from a Generator running on USB power.

For equipment that needs a higher signal voltage, plug in the 12 V power supply. **It's fine to have the USB cable and the power supply connected at the same time** — there is no conflict, and the outputs follow the higher of the two, swinging to about 11.8 V.

Accuracy and limits

How accurate are the frequencies?

How finely the Generator can set a frequency depends on how high it is: the lower the frequency, the more digits you can trust.

Frequency	Set to within
7.83 Hz	0.000001 Hz
100 Hz	0.0002 Hz
10 kHz	0.4 Hz
65.5 kHz	12 Hz

That's about eight trustworthy digits at low frequencies, falling to five at the top of the range.

Those figures are the setting resolution. The absolute accuracy also depends on the timing crystal on the Generator's ESP32 board, so for work that needs a certified frequency, measure the output with a calibrated frequency counter.

Why is Channel 2 slightly off when Channel 1 is set to a very different frequency?

Channels 1 and 2 share one clock inside the ESP32, and so do channels 3 and 4. The first channel of each pair picks the clock setting that suits its own frequency best, and the second channel has to work with that same setting. When the two frequencies are far apart, one setting can't suit both, and the second channel lands slightly off.

The fix: put frequencies that are close together on the same pair, or use **Channels 1 and 3** for two unrelated frequencies, since they don't share a clock.

When something's wrong

The display stays blank when I power on. What should I check?

What you see	Likely cause
Blank screen, LED blinking **fast**	The display isn't responding, and the Generator stops rather than run without it. Contact Aurorasky.
Blank screen, LED not blinking	No power: check the power switch, and that the supply is plugged in. If a firmware update was interrupted, reinstall the firmware.
Words on the screen, LED blinking	A card or file problem. Look up the message in Error and status messages.

A moving star field isn't a fault. That's the screen saver, after 20 minutes idle; press any button to wake it.

My oscilloscope shows no signal on an output. What should I check?

- **Is it running?** The outputs are off while the Generator is stopped or paused. The display should say `Running`.
- **Is that channel switching?** A duty cycle of `0.00` or `1.00` holds the output steady, low or high, with no waveform.
- **Right jack?** Top row: Channel 1, Channel 2. Bottom row: Channel 3, Channel 4.
- **Very low frequency?** At 0.04 Hz one cycle takes 25 seconds. Set the scope's timebase to match.
- **Ground clip** on the outer shell of the RCA jack.
- **Power connected?** The outputs take their level from the power source: up to about 11.8 V on the 12 V supply, or about 4.8 V when running on USB power alone.

The knob skips clicks or changes values the wrong way. Why?

The Generator counts a turn only when the knob settles into its next click position, and ignores movements that don't make sense, so a knob stopped partway between clicks doesn't count. **Turn it one click at a time** and every click registers.

In **Mode 4**, turning quickly makes the output lag behind the knob for a moment, but no clicks are lost; it catches up when you stop.

If the knob still misses clicks or counts the wrong way when turned slowly, the encoder may be worn. Contact Aurorasky.

Coming from Spooky2

I use Spooky2. How do my settings translate?

Spooky2	The Generator
Dwell	**Freq Time** (seconds) — but one value for every step in a file
Repeat Program	**Progm Time** (minutes) — the sequence loops until it runs out
Repeat Frequency	Put the record in the file more than once
Frequency Multiplier	Multiply the frequencies before entering them — nothing above 65,535 Hz
A program	A protocol file of up to 275 steps
A sweep	**Mode 3**

A program that gives one frequency a longer dwell, such as `7.83=600,174,963`, is handled by choosing a Freq Time that divides every dwell and repeating records.

Full detail, including that conversion worked through: Coming from Spooky2.

Firmware and hardware

How do I install new firmware?

You need the **`firmware_update_package`** folder, from the Generator's microSD card or the Generator's page on the website. No programming tools are needed.

Linux: plug in the Generator, open a terminal in the package folder, and run `./install_update.sh`. (If it says permission denied, run `chmod +x install_update.sh` first.)

Windows: plug in the Generator and double-click **`install_update.bat`**. If Windows shows a blue "Windows protected your PC" warning, click **More info**, then **Run anyway**.

On both: type `y` when asked to continue, then wait a few minutes **without unplugging**. It finishes with `SUCCESS -- the update has been installed.` and the Generator restarts on its own. No buttons need holding.

Full detail: Firmware update.

How do I build the firmware myself?

In brief:

1. **Arduino IDE 2.x.**
2. **Boards Manager:** install **esp32 by Espressif Systems, version 3.3.11 exactly.** Other versions can compile and still put out wrong frequencies.
3. **Library Manager:** install **Adafruit SSD1306** and accept its dependencies.
4. Put the eight `.ino` files in a folder named **`Generator_V2_05`**. Files downloaded from the website end in `.txt`; rename each to end in `.ino` first. The microSD card copies already end in `.ino`.
5. **Tools → Board: Nano32**; Upload Speed 921600; Flash Frequency 80MHz.
6. Compile. Three warnings are normal — including one about the legacy MCPWM driver, which should **not** be "fixed".

Full detail: Chapter 8 — Building the firmware from source.

How do I turn WiFi on?

WiFi is **switched off in the firmware as shipped**, and nothing on the card can turn it on by itself. Two steps:

1. **Firmware:** in `Generator_V2_05.ino`, change `byte wifiEnable = 0;` to `byte wifiEnable = 1;`, then rebuild and install. See building the firmware.

2. **Card:** edit `wifidataX.txt` so line 1 is your network name and line 2 its password, and save it as **`wifidata.txt`**, without the X.

At start-up the Generator shows its IP address for 10 seconds. Enter that address in a browser on the same network for a simple Start/Stop page. If you missed the address, press the knob to reset and watch again.

Full detail: WiFi operation.

How does the firmware work inside?

The firmware is eight Arduino files:

File	What it does
`Generator_V2_05.ino`	Start-up, the main loop, and the shared settings
`micro_SD.ino`	Reading and writing protocol files on the card
`PWM.ino`	Driving the four outputs through the ESP32's MCPWM hardware
`rotary_Decode.ino`	The knob and the limits on every value
`SSD1306.ino`	The display
`support_Routines.ino`	Run timing, the modes, sweeps, and saving
`USB_Comm.ino`	File transfers with the PC
`WiFi.ino`	The WiFi Start/Stop page

The central path is: **protocol file on the card → working memory → the running settings → the output hardware.** The outputs are on ESP32 pins 32, 33, 27 and 14. Channels 1 and 3 each set the clock for a pair, so Channel 2 is most accurate when its frequency is near Channel 1's, and Channel 4 near Channel 3's.

The source files are on the Generator's microSD card and on its web page. On the web page they end in `.txt`; rename them to `.ino` before opening them in the Arduino IDE.

Full detail: Appendix A — Architecture map.

Where are the schematic and circuit board drawings?

In Appendix B — Schematic and board layout, and as PDFs on the Generator's page on the Aurorasky website.

Big questions

Can the Generator treat or cure a health condition?

No claim of that kind is made. The Generator is an electronic signal generator, **not a medical device**. It is not designed or approved to diagnose, treat, cure or prevent any illness, and Aurorasky makes no claim about what any frequency does to the body.

For any health concern, speak to a qualified health professional.

What frequencies should I use?

The Generator doesn't recommend frequencies, and neither does this documentation. It plays whatever you put in a protocol file, from 0.04 to 65,535 Hz, and no claim is made about the effect of any frequency.

Choosing frequencies is up to you. The pages here explain how to set up whatever frequencies you choose. See How do I write a protocol file that plays a list of frequencies?

Could the Generator get more channels, or a sine-wave output?

Not as it is. The current design produces four square-wave outputs from the ESP32's PWM hardware. More channels or a sine wave would need changes to both the hardware and the firmware.

The firmware's source code is published. See How does the firmware work inside? For plans for future versions, contact Aurorasky.

Can I control the Generator from my phone?

Only to start and stop it, and only once WiFi has been switched on in the firmware. The Generator then serves a simple Start/Stop web page that works in a phone's browser. There's no app, and settings can't be changed remotely. See How do I turn WiFi on?

Can I add my own features to the firmware?

The firmware's full source code is published, and the manual explains how it is organised and how to build it:

- Chapter 8: Working with the firmware source, for the tools, settings and build steps.
- Appendix A: Architecture map, for how the eight files fit together.

Changing the firmware is for people comfortable programming in C with the Arduino IDE. Keep a copy of the original firmware update package so you can always go back.

How do I report a problem or suggest a feature?

Email Aurorasky at **contact_9@aurorasky.net**. For a problem, say what you were doing and the exact words on the display, and if a protocol file is involved, attach it.

What is the Generator not designed to do?

It's a simple four-channel square-wave generator. It is **not** designed to:

- produce sine, triangle or other waveforms;
- produce frequencies above 65,535 Hz;
- supply more than about 30 mA from an output, or drive low-resistance loads such as speakers or coils directly;
- adjust the output level other than through the supply voltage;
- diagnose or treat any health condition.

Finding your way

Which manual page covers my question?

Start with this page. For anything more, the manual index lists every page with a one-line description. In short:

- **Using the Generator:** chapters 1–4, plus Saving updates
- **Something's wrong:** Error and status messages, and Recovering protocols-0
- **Working with a PC:** chapters 5 and 6, and the Python tools pages
- **Writing files:** the protocol file format
- **Firmware:** chapters 7 and 8, and Appendix A

ESP32 Four Channel PWM Frequency Generator

SPECIFICATION SHEET

A four-channel programmable square-wave (PWM) generator built on the ESP32. Each channel runs its own frequency and duty cycle. Protocols are plain-text files on a microSD card, so it runs on its own with no computer attached: power it on and press the red button.

4 independent channels	**0.04 – 65,535 Hz** on every channel
4 run modes	**100 × 275** files × steps per file

OUTPUTS	
Channels	4 independent PWM square-wave outputs
Frequency	0.04 Hz to 65,535 Hz per channel, entered in 0.01 Hz steps
Resolution	0.000001 Hz at 7.83 Hz · 0.0002 Hz at 100 Hz · 0.4 Hz at 10 kHz · 12 Hz at 65.5 kHz
Duty cycle	0.00 to 1.00 (0–100 %) in 1 % steps; 0.00 and 1.00 hold the output steady low or high, so a channel can act as an on/off switch
Output level	0 V to just below the supply: about 11.8 V on 12 V, about 4.8 V on USB power. TTL compatible
Drive	Up to 30 mA source or sink per output, 330 Ω current limit, common ground
Connectors	4 × RCA (phono) jacks
Channel pairs	Ch 1 & 2 share a clock, as do Ch 3 & 4; use Ch 1 and 3 for unrelated frequencies
RUN MODES	
1 · Simple	Plays one record continuously
2 · Protocol	Steps through a range of records, then repeats
3 · Sweep	Sweeps Channel 1 between two records in steps of 0.00–9999.99 Hz (0 = automatic)
4 · Adjust	The knob tunes Channel 1 live while running
Step time	½ second, or 1 to 9,999 seconds per step
Program time	½ minute, or 1 to 9,999 minutes, then stops automatically
Pause	Pause and resume at any point in a run

PROTOCOL STORAGE	
Media	microSD card, FAT16/FAT32, up to 32 GB
Files	Up to 100 protocol files (0–99), selectable on the unit
Records	Up to 275 frequency records per file; one file can hold several sequences
Format	Plain text, editable on the unit or on a PC
Safety	Startup file repairs itself from two spare copies on the card
CONTROLS & DISPLAY	
Display	OLED shows all four frequencies and duty cycles
Controls	Rotary knob, red Start/Stop button, black Pause/Cursor button, power switch
POWER	
Input	5–12 V DC; the supply sets the output level
USB	Runs on USB power alone; USB plus 12 V is fine
COMPUTER CONNECTION	
Interface	USB (CP2102 serial, 115,200 baud)
PC tools	Free Python 3 programs: Protocol Editor and file send/receive
Updates	Firmware updates over USB with an installer
Firmware	Open source (Arduino, ESP32), expandable
IN THE BOX	
Included	Generator with loaded microSD card, 12 V 18 W power supply, two 3-ft RCA cables
Works with	Aurorasky Pulser, Plasma Ball and Red/Infrared lights

Specifications for firmware V2.05 · October 2026 · Full details in the Generator PWM Manual at aurorasky.net · G. Steven Kempe