

RCSIM Tier 1 — Arduino/RP2350 Digital to PPM: User Manual

Illustrated user guide: configuration cards, deployment actions and step-by-step procedures. The screenshots show the English interface.

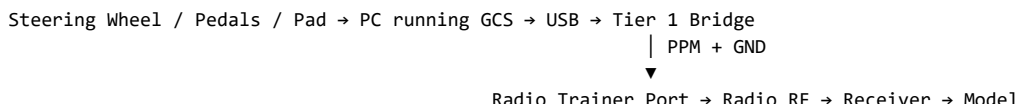
Contents

1. What Tier 1 Does	3
2. Selecting Firmware Before Wiring	3
2.1. Arduino Nano / Uno (ATmega328P)	3
2.2. Standalone RP2350 Tier 1	3
2.3. MCS RP2350 Co-Processor Variant	4
3. Bench Setup and Hardware Wiring	4
3.1. Power and Logic Levels	4
4. Flashing Arduino via Arduino IDE	5
5. Flashing XIAO RP2350	6
5.1. Flashing Precompiled UF2	6
5.2. Building via PlatformIO	6
6. GCS Configuration for Bridge Mode	6
7. Channels, Steering Wheel, and Pedals Setup	7
7.1. Channel Definitions	7
7.2. Control Mapping	8
7.3. Deadzones and Expo Curves	9
8. Transmitter and Receiver Trainer Setup	10
8.1. General Procedure	10
8.2. RadioMaster MT12 Implementation Example	10
9. CSV vs MCS Binary Protocols and Polarity	11
10. Failsafe, Disconnect, and Safe Return	11
10.1. Mandatory Bench Test Sequence	11
10.2. Ending a Session	12
11. Firmware Updates and Recovery	12
12. Quick Diagnostics	12
13. Source Code Verification Trails	13

Edition: 14 October 2026. This manual is grounded in the project codebase and physical hardware photographs. It does not replace physical testing or bench verification of your specific hardware assembly. Photographs show verified project assemblies; schematics, pin definitions, and electrical measurements take precedence over wire colors in the images.

1. What Tier 1 Does

Tier 1 translates digital control commands from the PC into a Pulse Position Modulation (PPM) stream delivered to an RC transmitter's trainer port. A steering wheel, pedal set, or gamepad provides inputs to GCS; an Arduino or RP2350 bridge converts serial channel data into PPM pulses, which the radio transmits over the air to its paired vehicle receiver.



Do not connect the combined PPM signal directly to a standard servo or ESC input. A standard servo requires a single-channel PWM signal, whereas PPM carries a multiplexed train of all channels. PPM does not transmit FPV video, GPS, IMU, or LiDAR data back to GCS. Video and telemetry operate over completely independent channels.

Tier 0 synthesizes PPM directly via the PC sound card DAC without a microcontroller bridge. Tier 2 is the ESP32-CAM platform; Tier 3 covers direct RF systems (ELRS/mLRS, CRSF/MAVLink, EdgeTX USB-VCP, ER5C V2 mods); Tier 4 is the autonomous Raspberry Pi 5 platform. Auxiliary iBUS/SBUS/CRSF modes supported by some co-processor builds do not alter the definition of Tier 1 Digital to PPM.

2. Selecting Firmware Before Wiring

2.1. Arduino Nano / Uno (ATmega328P)

Source sketch: `RCSIM_PC/pc_app/ESP32Arduino/RCSIM_Arduino_ESP32/Arduino Tier 1/Arduino.ino`.

- USB Serial: **115200 bps**, ASCII comma-separated channel values (CSV).
- PPM Output: **D10** and shared **GND**.
- **8 channels**; the sketch uses the **PPMEncoder** library and AVR Timer1.
- Initialized with positive polarity. Verify the exact frame timing and polarity with the installed library version.
- Watchdog: Loss of valid frames for more than **500 ms** or an ESTOP command stops the generator and switches D10 to INPUT (high impedance). Ensure the radio failsafe behavior is properly configured for trainer signal loss.

2.2. Standalone RP2350 Tier 1

Project directory: `RCSIM_PC/pc_app/ESP32Arduino/RCSIM_Arduino_ESP32/RP2350 Tier 1/`. Contains `main.cpp`, `platformio.ini`, and precompiled firmware.uf2 for the **Seeed Studio XIAO RP2350**.

- USB CDC: **115200 bps**, CSV, **8 channels**.
- PPM Output: **D8 / GPIO 2** (not D7).
- Timing: **22.5 ms** frame, **300 µs** active-low sync pulses, idling HIGH in default push-pull mode.
- Watchdog: Exceeding **500 ms** without valid data or triggering ESTOP halts frame generation, holding GPIO 2 in a HIGH idle state. An alternative open-drain configuration requires rebuilding the source.
- Note: A "Signal Killed" diagnostic message indicates pulse suppression, not physical power cut to the model.

2.3. MCS RP2350 Co-Processor Variant

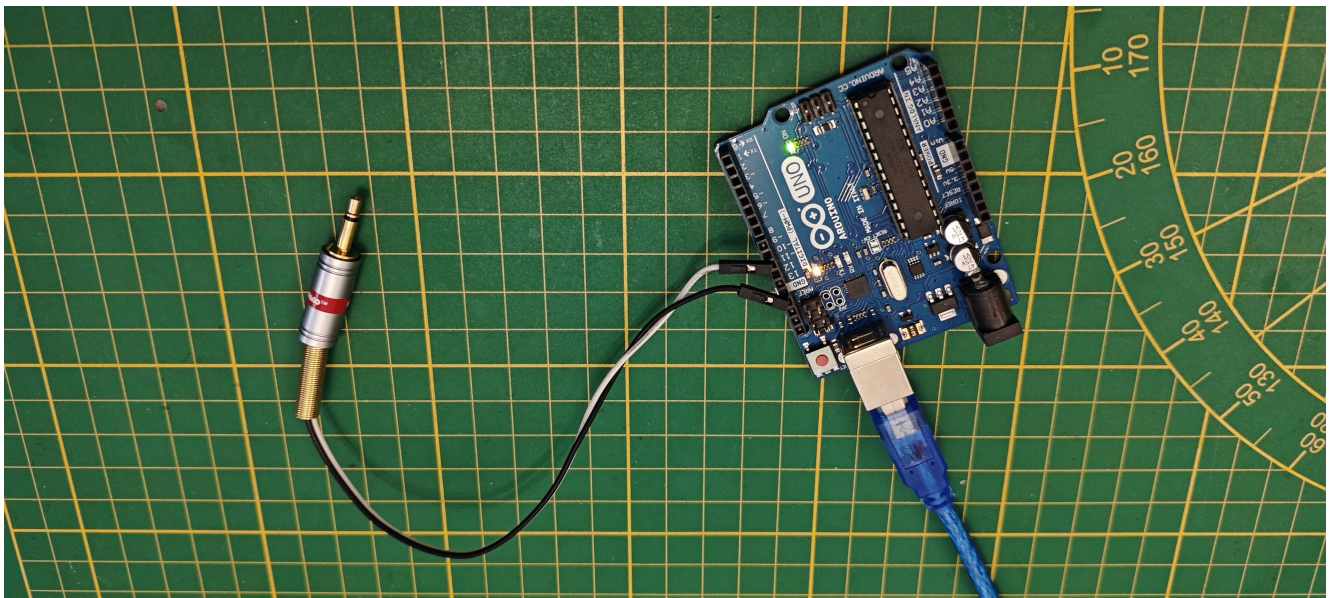
Directory: `RCSIM_MCS/firmware/xiao_rp2350/`. The source `src/main.cpp`, `platformio.ini`, and its included `firmware.uf2` are distinct from the standalone Tier 1 project.

- Supports both GCS ASCII CSV and the binary MCS protocol. Valid CSV automatically selects PPM output; the binary MCS stream selects between PPM, iBUS, SBUS, or CRSF.
- **D7 / GPIO 1** drives the mini-jack output designed for an inverting MOSFET buffer; **D8 / GPIO 2** serves as the serial TX line in iBUS/SBUS/CRSF modes.
- In PPM mode, the current firmware outputs pulses on both D7 and D8. At the raw GPIO level, both feature active-low pulses; however, passing D7 through the external MOSFET inverts the polarity seen by the radio. Do not swap D7 and D8 without reviewing your circuit.
- PPM Timing: **8 channels**, **22.5 ms** frame, **300 µs** pulse width, **500 ms** timeout. In default builds, failsafe stops pulsing and holds GPIO lines HIGH.
- Other protocols have different failsafe semantics: do not apply PPM instructions to SBUS failsafe flags.

Pin Summary: Arduino = D10; Standalone RP2350 = D8 / GPIO 2; MCS Co-processor = D7 / GPIO 1 (for MOSFET mini-jack) with D8 protocol-dependent. Label your board indicating which UF2 or sketch was flashed. Do not swap UF2 binaries between projects.

3. Bench Setup and Hardware Wiring

Prerequisites: A compatible microcontroller board, USB data cable, prepared firmware, GCS, sim controller, RC radio with DSC trainer port, appropriate signal patch lead, and bound receiver. During initial testing, rely on the transmitter's channel monitor; an oscilloscope or logic analyzer helps distinguish signal absence from menu misconfigurations.



Project example: Arduino Uno, USB and trainer port cable

3.1. Power and Logic Levels

1. Disconnect the vehicle drivetrain or elevate the drive wheels. Remove propellers on aerial vehicles during setup.

2. Power the bridge strictly via USB per board specifications. Never bridge the radio's positive battery supply to USB 5V.
3. Connect the bridge ground (GND) to the trainer port ground. Missing a common ground prevents decoding or causes erratic jitter.
4. Route the signal pin through a level-matching circuit suitable for your radio. Classic Arduino 5V D10 and XIAO 3.3V GPIO pins have distinct electrical characteristics.
5. Check your radio's input voltage tolerance. A "5V-tolerant" comment in code comments does not mean the XIAO outputs 5V, nor does it allow unrestricted 5V injection.
6. Verify the jack connector pinout with a multimeter and radio manual. There is no universal "Tip is always signal" standard across all manufacturers.

Arduino Uno/Nano:	D10 — Signal Buffer / Matching —	Radio PPM Trainer Input
	GND —————	Radio Trainer Ground
RP2350 Standalone:	D8 / GPIO 2 — Matching Circuit —	Radio PPM Trainer Input
	GND —————	Radio Trainer Ground
RP2350 MCS Firmware:	D7 / GPIO 1 — Inverting MOSFET —	Mini-Jack PPM Input
	GND —————	Radio Trainer Ground

For the MCS variant, do not bypass the MOSFET buffer simply because pulses are visible on GPIO 1; bypassing it inverts the polarity perceived by the radio.

4. Flashing Arduino via Arduino IDE

The following procedure applies to Uno/Nano boards with the ATmega328P running the Tier 1 Arduino sketch. Do not target ESP32 or RP2350 boards with this sketch.

1. Launch Arduino IDE and open `Arduino.ino` from `RCSIM_PC/pc_app/ESP32Arduino/RCSIM_Arduino_ESP32/Arduino Tier 1/`.
2. Allow the IDE to create a matching folder if prompted. Do not mistakenly open `RCSIM_MCS/firmware/co_processor/Arduino.ino`.
3. In the Library Manager, install **PPMEncoder** (`#include "PPMEncoder.h"`). Ensure it provides the `ppmEncoder.begin` and `setChannel` methods.
4. Select your board (Arduino Uno or Nano), the correct processor (e.g. ATmega328P Old Bootloader for clones), and the active COM port.
5. Click **Verify**. If `PPMEncoder.h` is missing, return to the library manager.
6. Close GCS or any serial monitor occupying the port, then click **Upload**.
7. Confirm the COM port persists after flashing. You may open the Serial Monitor at 115200 bps briefly to read the ready message, then close it before launching GCS.
8. Proceed to trainer testing. An IDE upload success message verifies flash write, not correct PPM voltage or radio failsafe.

Troubleshooting Uploads: Missing COM → check USB cable and CH340/FTDI drivers; Port busy → close GCS/monitors; Nano sync error → verify processor and bootloader settings. Never alter D10 wiring to fix USB upload errors.

5. Flashing XIAO RP2350

5.1. Flashing Precompiled UF2

1. Select your target firmware variant and locate its exact `firmware.uf2`. Note its folder origin; do not select arbitrary UF2 files from system search.
2. Disconnect GCS, close serial monitors, and unplug the signal cable from the radio during programming.
3. Connect the XIAO RP2350 via a USB data cable. Hold **BOOT**, press and release **RESET**, then release **BOOT**.
4. The board will mount as a mass-storage drive named RPI-RP2 (or similar).
5. Drag and drop the selected `firmware.uf2` onto the drive. The drive will automatically unmount and the board will reboot into USB CDC mode.
6. Note the new COM port. Reconnect the signal wire: D8 for standalone Tier 1, or D7/MOSFET for MCS.
7. Perform bench verification. Binary presence in git does not guarantee compatibility with arbitrary subsequent source changes.

5.2. Building via PlatformIO

For standalone Tier 1, open `RP2350_Tier_1` as a PlatformIO project. The configuration references sources in the project root; do not move `main.cpp` into a `src/` directory. Select the `seeed_xiao_rp2350` environment and run **Build**. Use the generated UF2 for the bootloader flash procedure.

For the MCS co-processor, open `RCSIM_MCS/firmware/xiao_rp2350`. The source resides in `src/main.cpp`, requires the Adafruit NeoPixel library, and utilizes `picotool` for uploading. Build, enter BOOT mode, and upload. Do not merge both `platformio.ini` files.

Both projects target the **Seeed Studio XIAO RP2350** using the Earle Philhower core. If tools fail to detect the board in bootloader mode, check the cable, BOOT button sequence, and USB device permissions.

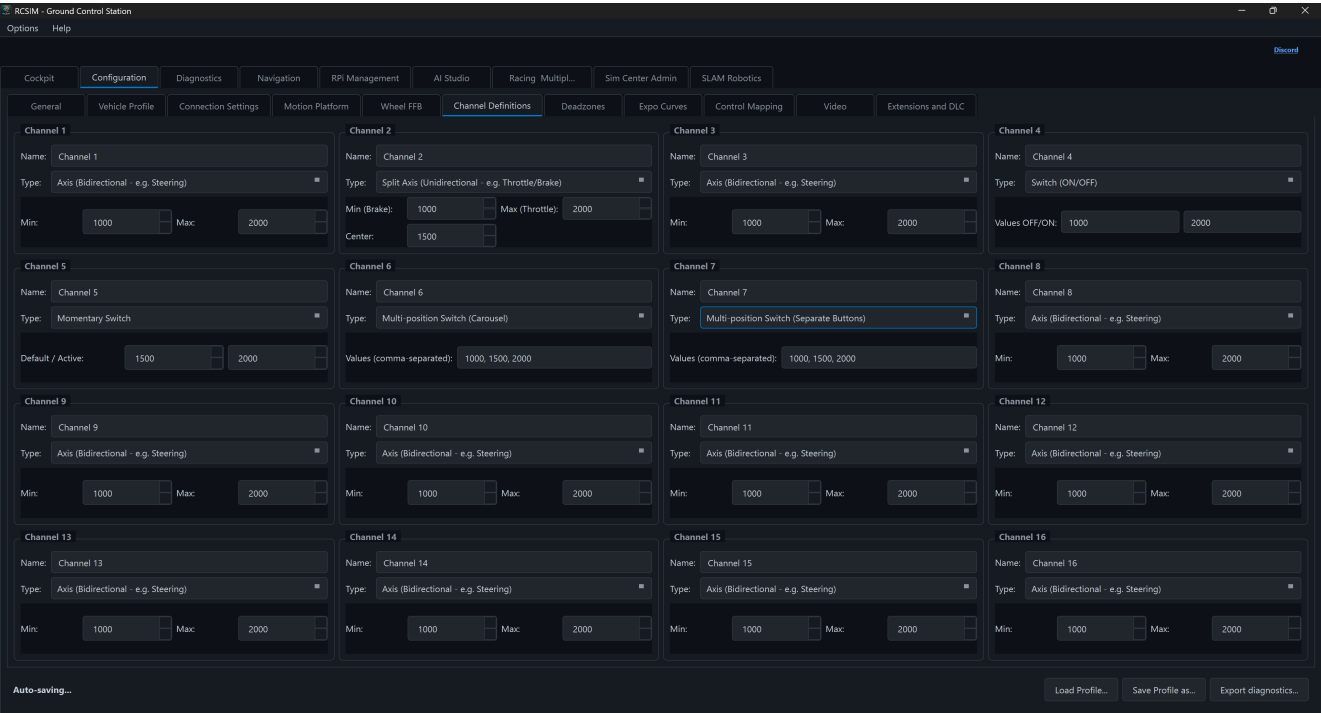
6. GCS Configuration for Bridge Mode

1. Connect your steering wheel/pedals and the microcontroller bridge. Launch GCS and open **Configuration** → **Connection Settings**.
2. Under **GCS Mode**, select **Arduino (Port Trenera)** for Arduino or **RP2350 (Port Trenera)** for XIAO. Opening the Tier 1 card without updating GCS Mode will not activate the connection.
3. On the **Tier 1 (Serial / USB)** card, click **Refresh Ports**, select the identified **Port**, and set **Baud = 115200**.
4. Allow auto-save to complete. **Save Profile as...** saves a local copy of your bench profile; it does not flash firmware.
5. In **Cockpit**, select **Manual (Driver)** mode, confirm **Connection Mode**, and click **Connect**. A successful COM open does not confirm radio PPM reception.
6. Verify channel motion in GCS first. With the drivetrain secured, click **ARM SYSTEM** to enable the control stream and clear the bridge ESTOP state.
7. Inspect the transmitter's channel monitor. When disarmed, GCS blocks the active control stream; lack of radio movement does not indicate a severed wire.

DTR/RTS Handling: The RP2350 strategy in GCS asserts DTR/RTS and pauses **~0.3 s** for USB CDC enumeration. Arduino opens standard serial, which triggers an AVR hardware reset. Do not apply Nomad or MCS DTR/RTS advice to this path. After reconnecting, allow the board to initialize before testing.

7. Channels, Steering Wheel, and Pedals Setup

7.1. Channel Definitions



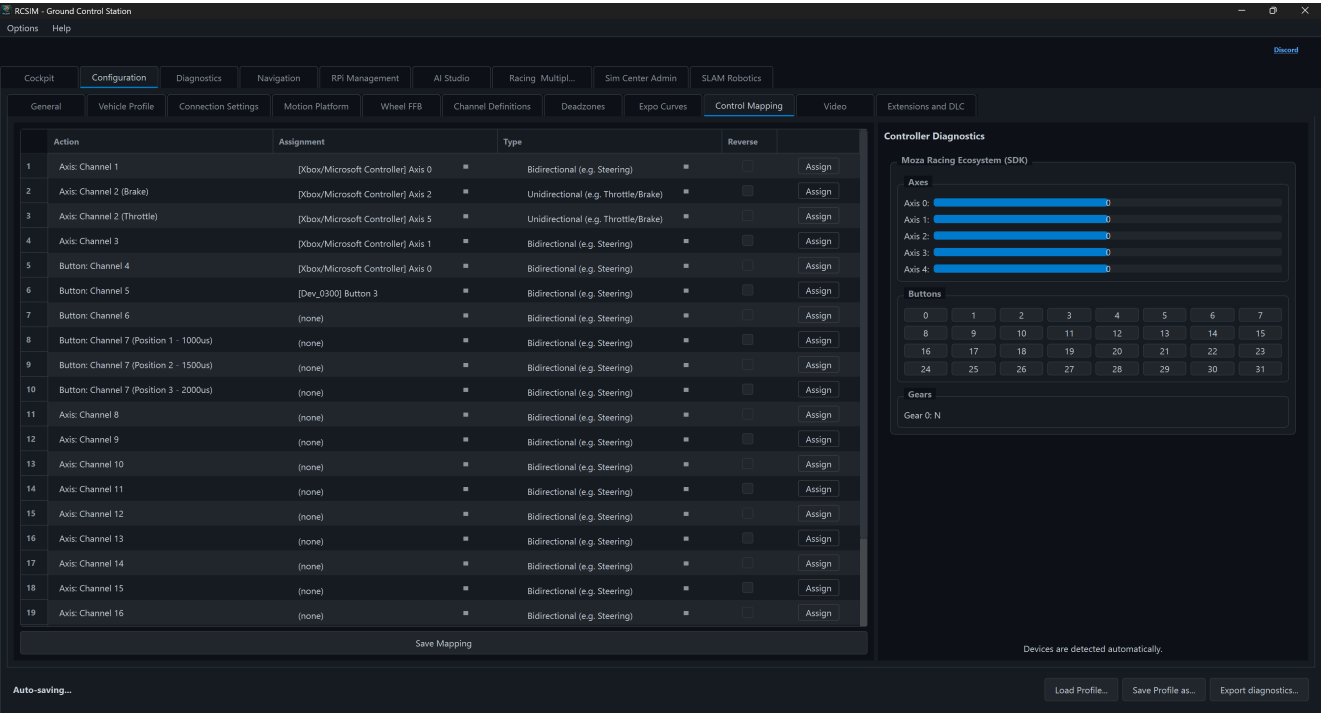
GCS — channel definitions

Open **Configuration** → **Channel Definitions**. Name CH1 as Steering and set it to **Axis (Bidirectional)**. For a surface RC car using throttle/brake/reverse on a single channel, configure CH2 as **Split Axis (Unidirectional)** to allow separate pedal assignments.

Set Min, Max, and Center to match your ESC, servo, and radio setup. Begin with constrained endpoints if physical travel limits are unknown.

Tier 1 PPM bridges transmit the **first 8 channels**. Do not assign primary driving functions solely to CH9–CH16. GCS truncates the list to 8 channels for RP2350; Arduino firmware similarly parses 8 values. The parser acceptance window of **800–2200 µs** is an envelope limit; standard operational range is **1000–2000 µs** with 1500 µs neutral.

7.2. Control Mapping

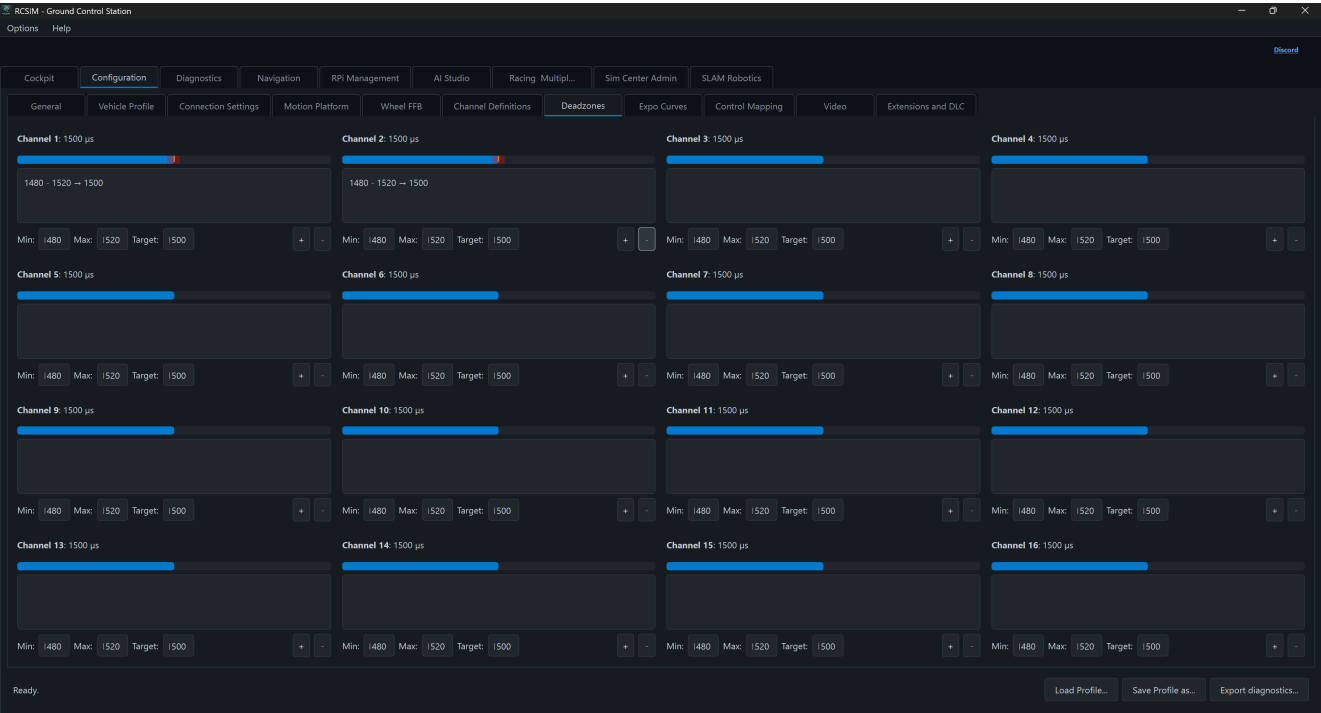


GCS — control mapping

1. Open **Control Mapping**, click **Assign** beside Steering, and rotate only the steering wheel.
2. Assign **Brake** and **Throttle** separately for the split throttle channel. Release all other pedals during assignment.
3. Confirm **Type**: Steering is bidirectional, pedals are unidirectional.
4. Toggle **Reverse** if axis direction is inverted. Confirm full range and released pedal state.
5. Map auxiliary switches within CH1–CH8, then click **Save Mapping**.
6. In Controller Diagnostics, actuate each axis individually. Pedals must not conflict or pull in opposite directions.

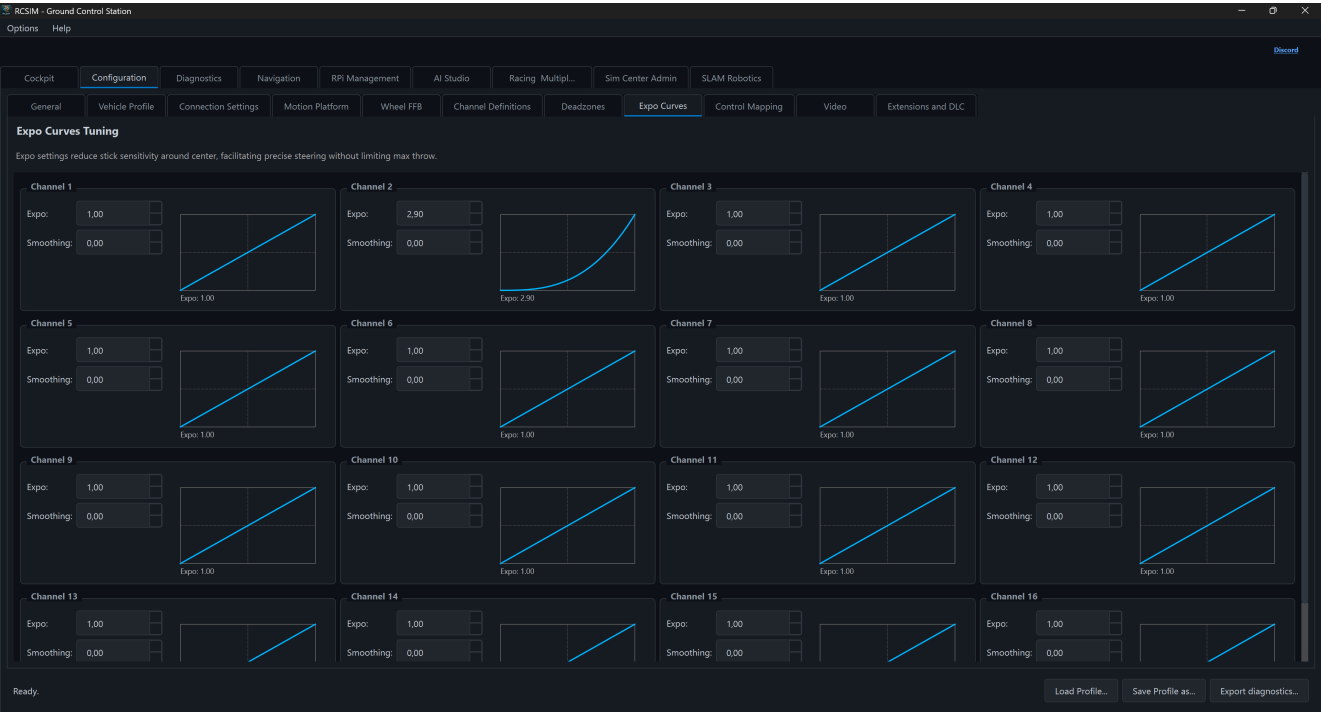
Expected Result: Steering centers properly, released throttle demands zero drive, and brake actuates in the intended direction. Verify identical behavior in the radio monitor.

7.3. Deadzones and Expo Curves



GCS — deadzones

In **Deadzones**, monitor the resting channel values. A Min–Max → Target rule (e.g. **1480–1520 → 1500**) eliminates baseline sensor jitter for channels with a true 1500 neutral. Do not apply this to aircraft throttle (minimum at 1000).



GCS — expo curves

In **Expo Curves**, **1.00** represents linear response; higher values soften initial center sensitivity. Keep **Smoothing = 0.00** initially. Heavy smoothing introduces control lag.

Calibrating input axes in GCS is distinct from ESC throttle range calibration. Execute ESC calibration following manufacturer instructions only after receiver range and neutral points are established.

8. Transmitter and Receiver Trainer Setup

8.1. General Procedure

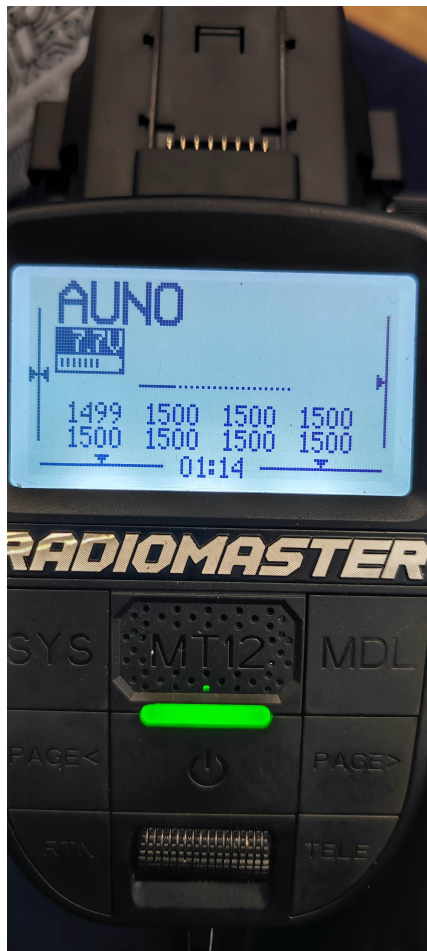
1. Select your target model in the radio. Back up settings before modification.
 2. Configure trainer input as **PPM/CPPM**, specifying the correct jack port, polarity, and channel count.
 3. In EdgeTX, set trainer mode to **Master / Jack**. Do not select Master/CRSF unless using a wireless RF link.
 4. In the Trainer menu, map inputs to channels. Ensure Steering and Throttle map to the correct physical vehicle channels.
 5. Assign trainer activation to a physical toggle switch via **Special Functions**. Verify that releasing the switch instantly restores local control.
 6. First observe inputs on the channel monitor without model power. Then test the bound receiver and ESC.
- Receiver binding is handled natively by the transmitter; the USB bridge plays no role in RF pairing.

8.2. RadioMaster MT12 Implementation Example



MT12 — trainer special function example

The image illustrates **SPECIAL FUNCTIONS** in EdgeTX with **Trainer** enabled via an assigned switch.



MT12 — 8 channels preview

During verification, check that actuating one GCS axis modulates exactly one intended channel on the display. Near-center values shown in photographs are not pre-calibrated ESC setpoints. If the radio displays "No trainer signal" despite active USB, verify PPM signal continuity, pinout, and polarity.

9. CSV vs MCS Binary Protocols and Polarity

GCS transmits ASCII CSV strings with values in microseconds terminated with a newline (1500,1500,...\n). ARM and ESTOP commands are sent as text strings.

Standalone Arduino and RP2350 firmware only accept CSV. The MCS co-processor additionally accepts a **19-byte binary frame** starting with header **0xAA**, protocol selector, and 8 packed channel values. Directing the wrong format will establish a serial connection but fail to actuate channels.

Only one host application may control the bridge at a time. Never open the serial port concurrently from GCS and MCS.

Measure polarity at the transmitter input pin, taking all intermediate buffer transistors into account. A library setting, GPIO pulse, and inverted MOSFET buffer represent distinct measurement nodes.

10. Failsafe, Disconnect, and Safe Return

10.1. Mandatory Bench Test Sequence

1. Elevate the vehicle on a stand so wheels rotate freely.

2. Arm the system in GCS, test steering and throttle, and note neutral receiver values.
3. Disarm in GCS. Confirm PPM pulse suppression or the expected neutral receiver output.
4. Re-arm and unplug the USB cable or stop the control stream. The bridge firmware watchdog trips after **500 ms** without valid data.
5. Observe the radio's response to trainer loss, followed by receiver output. The radio may revert to internal sticks or enter a programmed failsafe; loss of PPM does not automatically guarantee throttle cut unless programmed.
6. Test radio RF link loss independently by powering off the transmitter.
7. Upon restoring USB or re-arming, ensure throttle is neutral and verify the vehicle remains stationary.

Arduino: ESTOP latches until an ARM command is received. **Standalone RP2350:** ESTOP suppresses CSV until re-armed; pulse generation halts on watchdog timeout. **MCS RP2350:** PPM halts pulses on failure, but subsequent behavior depends on whether CSV or binary mode is active.

10.2. Ending a Session

Stop the vehicle, ensure neutral throttle, disarm in GCS, and confirm receiver status. Disconnect GCS, disconnect the battery from the model, and then unplug the USB bridge. Never rely on closing a window or unplugging USB as an operational brake.

11. Firmware Updates and Recovery

Before updating, back up your GCS profile, radio model setup, and record the current firmware binary version. Disconnect vehicle power and the PPM signal wire. Close GCS and serial monitors.

For Arduino, reflash via Arduino IDE with the matching board and library. For XIAO, enter bootloader mode using BOOT/RESET buttons and copy the target UF2. Verify the assigned COM port.

If the bridge fails to respond after flashing, verify whether standalone or MCS firmware was installed, check GCS strategy selection, and ensure the port is free. Re-flash a known working binary if necessary.

Re-verify neutral throttle, failsafe, USB disconnect, and RF loss after every firmware, library, or hardware modification.

12. Quick Diagnostics

- **No COM Port Detected:** Check USB cable (data vs charge-only), board power, bootloader state, and serial drivers. UF2 mass storage and CDC serial are distinct operating states.
- **Port Busy Error:** Close Arduino Serial Monitor, terminal emulators, secondary GCS instances, or background MCS services.
- **GCS Connected but Radio Shows No Trainer:** Check ARM SYSTEM in GCS, firmware variant, GPIO pin, ground wire, logic level, and radio trainer port selection.
- **Radio Displays Channels but Model Does Not Move:** Verify trainer switch/mixers in EdgeTX, model profile, receiver binding, and ESC arming.
- **Throttle Does Not Return to Neutral:** Check axis assignment, Invert, Center point, and ESC neutral calibration. Do not mask this using Expo alone.
- **Channel Jitter:** Verify common ground connection, inspect PPM waveform on an oscilloscope, and check for loose jack contacts.

- **Model Accelerates Upon Reconnection:** Halt testing immediately; configure proper trainer fallback and receiver failsafe settings before proceeding.
- **No Telemetry / Video / FFB via Bridge:** PPM is strictly a unidirectional control protocol; video and telemetry require independent dedicated links.

13. Source Code Verification Trails

- RCSIM_PC/pc_app/ESP32Arduino/RCSIM_Arduino_ESP32/Arduino Tier 1/Arduino.ino:16–22,36–43,65–76,91–113,154–172 — pins, library, watchdog, ARM/ESTOP.
- RCSIM_PC/pc_app/ESP32Arduino/RCSIM_Arduino_ESP32/RP2350 Tier 1/main.cpp:20–34,101–126,213–238,263–276 — PPM waveform generation, CSV parsing, pulse suppression.
- RCSIM_PC/pc_app/ESP32Arduino/RCSIM_Arduino_ESP32/RP2350 Tier 1/platformio.ini:1–10 — standalone project environment and board definition.
- RCSIM_MCS/firmware/xiao_rp2350/src/main.cpp:20–42,173–196,243–275,382–449,455–510,558–587 — dual protocols, pins, PPM buffer control.
- RCSIM_MCS/firmware/xiao_rp2350/platformio.ini:1–28 — MCS build configuration, picotool, NeoPixel.
- RCSIM_MCS/src/output/bridge.py:15–56 — MCS binary bridge packet structure.
- RCSIM_PC/pc_app/core/comm/serial_strategy.py:48–51,84–91,108–131 — Arduino serial opening, CSV formatting, commands.
- RCSIM_PC/pc_app/core/comm/rp2350_strategy.py:47–55,101–113,124–147 — RP2350 DTR/RTS assertion, 8-channel limitation, command structure.
- RCSIM_PC/pc_app/core/comm_manager.py:232–258,483–498 — arming logic and stream gating.
- RCSIM_PC/pc_app/gui/tabs/connection_ui.py:94–101,195–205 — UI bridge selection and baud rate settings.