User Manual



PROG-STM/ESP Programmer

STANDALONE

STM32 OVER SWD

ESP32 / ESP8266

Flash and verify firmware without a PC, straight from the microSD card

For anything not covered here, see section 10, Troubleshooting, at the end of this manual.

Contents

Introduction 01

Menu and navigation 02

File structure and formats on the SD card 03

Programming and verifying a firmware 04

Option bytes and OTP through .OPT files 05

Updating the programmer's own firmware 06

USB mode 07

STM32 family compatibility 08

ESP32/ESP8266 compatibility and boot pins 09

Troubleshooting: on-screen messages 10

SECTION 01

Introduction

The NORTHED PROG-STM/ESP programmer is a standalone device (no PC required) for flashing and verifying firmware on STM32 microcontrollers (F0-F7, G0/G4, L0/L1/L4/L5, H5/H7 families) and on ESP32/ESP8266 modules. It reads the firmware files from a microSD card and transfers them to the target board over SWD for STM32, or over the ROM bootloader serial port for ESP devices.

Parts of the device: OLED display, OK button (confirm/enter), SELECT button (navigate/back), target board connector, microSD slot and USB connector.

On power-up the device briefly shows the NORTHED logo and then goes straight to the main menu.





SECTION 02

Menu and navigation

There are only two buttons:

OK	Confirms or enters the highlighted option.
SEL	Moves down to the next option in the list; at the last one it wraps around to the first (there is no "up", only cyclic advance).
1 s	SELECT held for 1 second: goes back one level (BACK). If kept held, it keeps going up one level per second until it reaches the root menu.

The highlighted option is shown on screen as `-[ TEXT ]-` . Long lists (families, files) scroll by themselves so the cursor is always visible. Only the folders and files that actually exist on the SD card are listed.

MENU MAP

SELECT

```
STM → family → .HEX file
ESP → variant → .BIN file
→ PROGRAM / VERIFY / BACK

FIRMWARE.HEX (root)
→ programmer self-update
```

OPTIONS

```
SPANISH / ENGLISH
SPEED → STM32 / ESP
USB MODE
```

Inside **STM** the possible family folders are `F0 F1 F2 F3 F4 F7 | G G0 G4 | L L0 L1 L4 L5 | H H5 H7` (section 8). Inside **ESP** the variants are `ESP32, ESP32-D, ESP32-S, ESP32-C, ESP32-H, ESP8266` (section 9).

OPTIONS offers:

- **SPANISH / ENGLISH**: switches the menu language instantly, with no restart.
- **SPEED → STM32**: `FAST 3.9MHZ` (default) or `SLOW 976KHZ` for the SWD clock. If a particular target does not connect reliably at FAST, switch to SLOW.
- **SPEED → ESP**: `FAST 460800` (default, after syncing at 115200) or `SLOW 115200` for the UART to the ESP.
- **USB MODE**: see section 7.

> These speed and language settings are held in RAM only: when the programmer is restarted they return to their default values (SPANISH/FAST).

SECTION 03

File structure and formats on the SD card

The microSD card must be formatted as **FAT16 or FAT32** and must always follow this fixed folder structure:

```
/STM/F0/ F1/ F2/ F3/ F4/ F7/ G/ G0/ G4/ L/ L0/ L1/ L4/ L5/ H/ H5/ H7/   -> .HEX files (Intel HEX)
/ESP/ESP32/ ESP32-D/ ESP32-S/ ESP32-C/ ESP32-H/ ESP8266/                -> .BIN files
/FIRMWARE.HEX   (optional, in the root) -> update of the programmer itself (section 6)
```

You only need to create the folders for the families you are going to use; the menu lists only the ones present on the card.

File names

The programmer's file system only recognises **short 8.3 names** (DOS format, case-insensitive). A file copied from Windows with a long name still works, because Windows automatically generates an 8.3 alias for it, but that alias is the name shown on the programmer's display; to avoid confusion it is best to name the files in short format from the start (up to 8 characters plus extension).

STM32 firmware · `STM/<family>/*.HEX`

Standard Intel HEX format, exactly as exported by the development environment (enable "Convert to Intel Hex" in the project's output settings if it is not generated automatically).

ESP firmware · `ESP/<variant>/*.BIN`

It must be a **single, already merged image** starting at offset 0 (bootloader + partition table + application, with the gaps filled), as produced by the ESP-IDF project itself when built with merged output, or by a binary merging tool. The programmer writes the image as-is, without touching its flash mode/size header, so the `.bin` must already be generated with the correct flash settings for the module.

Auxiliary files

`NAME.OPT` (optional): a text file with the same base name as a `.HEX` file and in the same folder, used to apply option bytes / OTP while flashing. See section 5.

`TARGET.CFG` (optional, next to a family's `.HEX` file): a text file that may contain the word `RESET` so that the programmer connects to the target while holding the reset line active for the whole session (useful when the core must not run the old firmware before flashing).

FIRMWARE.HEX in the root

Reserved name. It is not a target firmware: when you enter SELECT and the programmer finds it, it understands that you are asking it to update its own firmware (section 6). Target firmware files always go inside `STM/<family>/` or `ESP/<variant>/`, never loose in the root.

SECTION 04

Programming and verifying a firmware

When you pick a file inside a family/variant folder, the **PROGRAM / VERIFY / BACK** submenu appears (BACK returns to the file list of that same folder, not to the root):

- **PROGRAM**: the full flow — connect, erase, write and verify. If an `.OPT` file with the same name exists, its option bytes/OTP are applied at the end as well (section 5).
- **VERIFY**: connects and **only compares**, without erasing or writing anything on the target. On STM32 it compares word by word over SWD; on ESP it reads the MD5 hash calculated by the chip itself and compares it with the file's. Result: `OK - MATCH` or `MISMATCH`.
- **ESP8266 does not support VERIFY** (its ROM does not calculate MD5): it shows `NO VERIFY / ROM WITHOUT MD5` without attempting anything.

Progress screen

Detected chip model (top, centred), current phase, progress bar and percentage. The "exact model" shown depends on what the chip itself allows to be read: on STM32 it may stop at the line/density (e.g. `STM32F10X 64KB`) when the chip cannot distinguish further; on ESP it is usually exact (`ESP32-C3`, etc.).

`CONNECTING` `UNLOCKING` `ERASING` `FLASHING` `VERIFY` `READING FILE` `VERIFY MD5`

Match check: after connecting, the programmer identifies the real chip. If it does not match the family/variant of the chosen folder, it stops with `WRONG CHIP` and the name of the detected chip, without writing anything. If the chip is not in the table (a new model), it does not block: the folder's driver is assumed.

When it finishes

SUCCESS	ON ERROR	LED
DONE screen with the timing breakdown, for 2 s or until OK is pressed, then an automatic return to PROGRAM/VERIFY/BACK for the same file — designed for flashing several boards in a row without navigating the menu again.	The failure screen stays visible until OK is pressed (or SELECT held), showing the corresponding diagnosis (section 10), and it also returns to PROGRAM/VERIFY/BACK for the same file.	When the operation ends, the LED stays **solid** if it went well and **blinking** if it failed; it switches off automatically when the next programming or verification starts.

> If the chosen family or variant does not have flashing support yet (see the compatibility tables, sections 8 and 9), picking the file shows `NO DRIVER` straight away without attempting anything; browsing those folders works normally.

SECTION 05

Option bytes and OTP through .OPT files

Besides writing the main flash, the programmer can apply **option bytes** (the chip's configuration bits: read/write protection, watchdog behaviour, BOR level, boot pins…) and write to the **OTP** (one-time programmable memory). This happens only when you press **PROGRAM** (never on VERIFY) and **after** the flash has been written and verified successfully.

How to enable it

Create a text file `NAME.OPT` with the **same base name** as the corresponding `NAME.HEX`, in the same family folder. The programmer detects it automatically.

Ready-to-use templates

Every family folder inside `STM/` includes a `TEMPLATE.OPT` file with the valid keys for that family, its factory values active (so it changes nothing as supplied) and the rest commented out with an explanation of each bit. Copy that template next to the `.HEX` file, rename it to match, and uncomment/edit only what you need. There is also a general `EXAMPLE.OPT` in the root of the SD card for reference.

Format: one key per line, `=` for the value, `#` or `;` for comments, case-insensitive, up to 3 KB.

KEY	FAMILIES WHERE IT APPLIES	MEANING
`RDP=0 / 1 / 2`	all except H5	Read protection level. **2 requires PERMANENT=YES** (irreversible). On F1 only 0/1 exist
`WRP=NONE / WRP=ALL`	F0/F1/F3, F2/F4/F7, G0/G4/L4/L5, H7	Write protection of the whole flash
`IWDG_SW=0/1, NRST_STOP=0/1, NRST_STDBY=0/1`	all except L0/L1	Classic user bits (1 = factory value)
`NBOOT0, NBOOT1, NSWBOOT0, NBOOT_SEL (=0/1)`	depending on family	Boot configuration bits. Asking for one the family does not have gives `BIT NOT IN FAMILY`
`BOR=n`	F2/F4/F7, G4/L4/L5, H5, H7	Brown-out detection level (numeric value depending on the family)
`USER=0xXX`	F0/F1/F3	The complete USER byte (takes priority over the individual bit keys)
`PRODUCT_STATE=0xXX`	H5	The H5 has no RDP: this is its equivalent. **Requires PERMANENT=YES**
`RAW=0xVALUE/0xMASK`	all	Individual bits of the main option register, for cases not covered by the keys above
`OTP 0xADDRESS=00112233...`	F2/F4, F7, G0/G4/L4, L5, H5, H7	Writes up to 4 lines of 32 bytes into the OTP area. The area must be **blank (0xFF)**; if it is not, the error is `OTP NOT BLANK`. **Requires PERMANENT=YES**
`PERMANENT=YES`	—	Confirms that an **irreversible** change is accepted (RDP=2, OTP, PRODUCT_STATE). Without this line the file is rejected as invalid

On-screen confirmation: if the `.OPT` file asks for something sensitive (RDP≥1, OTP, PRODUCT_STATE or RAW), the file name and a summary are shown before the target is touched, with `OK: YES SEL: NO`; pressing SELECT cancels (`CANCELLED`) without applying anything. With harmless settings (user bits, WRP) it does not ask, so that many boards can be flashed in a row without interruptions.

Errors: a badly written `.OPT` file, or one with a key that is not valid for that family, is rejected **before** the target is touched, showing `OPT FAIL` and the reason (`KEY LINE n`, `RDP LINE n`, `OTP HEX n`, `PERMANENT MISSING`, `OTP NOT BLANK`, `BIT NOT IN FAMILY`...), then returning to the file's submenu.

> **CAUTION**
>
> `RDP=2`, any `OTP` write and `PRODUCT_STATE` are **irreversible operations**. Always try them on a test board first, before applying them to a production unit.

SECTION 06

Updating the programmer's own firmware

When NORTHED releases a new firmware version for the programmer, download the corresponding `FIRMWARE.HEX` file from the NORTHED website (northed.es) and copy it to the **root** of the microSD card, replacing any previous `FIRMWARE.HEX`. Do not modify the file: the programmer checks its integrity before applying it and rejects it if anything is wrong.

1. Insert the card with the new `FIRMWARE.HEX` in its root.
2. In the menu, enter **SELECT**: on detecting the file, the programmer shows the self-update screen instead of the usual target list.
3. A version comparison appears, for example `CURRENT V1.0` / `NEW V1.1`, with `OK: YES SEL: NO`. Press **OK** to continue or **SELECT** to cancel (`CANCELLED`).
4. The programmer writes the new version with a progress bar, verifies its integrity (`VERIFYING CRC`) and applies the change (`SWAPPING BANK`), restarting automatically (the logo appears again).
5. Entering SELECT again with the SD card in place should now show the new version.

Safe against power loss

The firmware in use is not touched until the new version has been fully written and verified. The system also allows "updating" to an earlier version if needed; the comparison is shown either way so you can decide.

If the file is not valid

Before the memory is touched, messages such as `HEX FORMAT`, `BAD ADDRESS`, `NO FW INFO`, `NOT PROG FW`, `WRONG HW`, `BAD SIZE` or `SIZE MISMATCH` are shown: the file is not valid for this device and the current firmware is left untouched. If the failure happens during writing (`ERASE FAIL`, `PROGRAM FAIL`, `CRC FAIL`, `OPT BYTES FAIL`), the firmware that was in use stays intact and the device keeps working normally; the update can be retried.

> Remember that as long as there is a **`FIRMWARE.HEX`** in the root of the SD card, that name is reserved for the self-update: target firmware always goes in **`STM/<family>/`** or **`ESP/<variant>/`** (section 3).

SECTION 07

USB mode

Reading and writing the SD card from the computer

OPTIONS → USB MODE turns the programmer into a card reader: when the USB cable is connected to the computer, the microSD appears as an ordinary removable drive, with no need to take the card out of the device. It lets you copy new firmware and `.OPT` files, or delete the ones you no longer need, straight from the computer's file explorer.

It works with **Windows, macOS and Linux**: it is a standard USB mass-storage device, so it needs no drivers or additional software on any of the three systems.

To exit, hold **SELECT for 1 second**: USB is disconnected and the card is mounted internally again, so the new files already appear under **SELECT**.

While USB mode is active the menu does not access the SD card (and vice versa): the programmer and the computer never use it at the same time. If no card is inserted when entering this mode, `NO SD CARD` is shown.



Recommended practice

Keep only the firmware in production on the card: the menu lists everything it finds, and a folder with many files makes two-button navigation slower.

SECTION 08

STM32 family compatibility

The `STM/` folders cover the whole current STM32 range (there are no "F5"/"F6" families: they were never made as STM32). Single-letter folders (`G`, `L`, `H`) **auto-detect** the exact chip within that letter without forcing anything; numbered folders (`F1`, `G0`, `L4`, `H7`...) **force** that specific family and give `WRONG CHIP` if the connected chip belongs to another one.

FOLDER	CHIPS COVERED	STATUS
`F0`	STM32F0xx	Written and checked in software (same engine as F1); not verified on real hardware
`F1`	STM32F1xx	**Field-verified**, the project's reference family
`F2`	STM32F2xx	Not verified on real hardware
`F3`	STM32F3xx	Not verified on real hardware (same engine as F1)
`F4`	STM32F4xx	Not verified on real hardware
`F7`	STM32F7xx	Not verified on real hardware
`G`	Any STM32G0/G4 (auto-detects)	As detected, see G0/G4
`G0`	STM32G0xx	**Field-verified**
`G4`	STM32G4xx	Not verified on real hardware
`L`	Any STM32L0/L1/L4/L5 (auto-detects)	As detected, see L0/L1/L4/L5
`L0`	STM32L0xx	Not verified; very slow programming (word by word, EEPROM-like)
`L1`	STM32L1xx	Not verified; same speed limitation as L0
`L4`	STM32L4/L4+/G4 (same controller)	Not verified on real hardware
`L5`	STM32L5xx	Not verified; works only with **TrustZone disabled** on the target
`H`	Any STM32H5/H7 (auto-detects)	As detected, see H5/H7
`H5`	STM32H5xx	Not verified as a target (although it is the same flash controller the programmer uses for its own self-update, which is already tested)
`H7`	STM32H7xx	Not verified; the highest-risk family (dual bank, 256-bit word)

> Everything marked "not verified" is written according to ST's reference manuals, but has not yet been tested on real silicon. Before using it in production, try it first on a low-cost board of that family.

SECTION 09

ESP32/ESP8266 compatibility and boot pins

To flash an ESP over serial, the module must enter its **ROM bootloader**. That is achieved by holding the **EN/CH_PD** pin high (chip enabled) while the **boot (strapping)** pin is pulled low during reset. The programmer's connector has two fixed signals for this — `ENABLE_ESP32` (to EN/CH_PD) and `BOOT_ESP32` (to the boot pin) — but **which physical GPIO of the module BOOT_ESP32 must go to changes with the chip**:

MENU FOLDER	CHIPS GROUPED	BOOT PIN	VERIFIED
ESP32	Classic ESP32 (ESP32-D0WD)	**GPIO0**	**Yes**, the variant already tested in the field
ESP32-D	ESP32-D0WD (alias of the classic)	**GPIO0**	Same driver as ESP32
ESP32-S	ESP32-S2, ESP32-S3	**GPIO0**	Not verified
ESP32-C	ESP32-C2, C3, C5, C6, C61	**GPIO9**	Not verified
ESP32-H	ESP32-H2	**GPIO9**	Not verified
ESP8266	ESP8266	**GPIO0** (+ GPIO15 to ground and GPIO2 high, usually already fixed on the module)	Not verified

In every case, `ENABLE_ESP32` goes to the module's **EN / CH_PD** pin (high = chip enabled). When building an adapter or cable for a specific module, connect `BOOT_ESP32` as shown in the table column; if the wrong pin is used, the module does not enter flashing mode and the programmer shows `NO ESP (UART)` (no response).

Automatic chip check

After syncing, the programmer identifies the real chip connected. If it clearly belongs to a different family than the chosen folder, it stops with `WRONG CHIP` and the detected name, without flashing; if the chip is not in the table (new silicon), it does not block.

Verification

Available across the ESP32 family (it compares the chip's MD5 against the file's). The **ESP8266 cannot verify**: its ROM does not implement that command and `NO VERIFY / ROM WITHOUT MD5` is shown (this is not a failure, it is a limitation of the chip itself).

UART speed

By default it syncs at 115200 and steps up to 460800 for flashing (faster); the ESP8266 does not support this speed change and always works at 115200 even with the FAST option set.

SECTION 10

Troubleshooting: on-screen messages

MESSAGE	MEANING	WHAT TO DO
`SD... does not appear when pressing OK in SELECT`	The OK button was not detected	Press again; check the button
`SD: NO CMD0 RESP`	The card does not respond at all	Check that the SD card is properly inserted and check the wiring/contacts
`SD: CMD8 NO RESP / SD: CMD58/16 FAIL`	Failure at a specific step of the SD start-up	Try another card; check the contacts
`SD: ACMD41 TIMEOUT`	The card responds but does not leave the idle state	Check the SD card's power supply
`FAT: READ FAIL`	The file system cannot be read	Try another card
`FAT: NO FS`	The card is not formatted as FAT16/FAT32 (e.g. exFAT)	Reformat as FAT32
`WRONG CHIP + detected name`	The connected chip does not belong to the family/variant of the chosen folder	Enter the correct folder for that chip
`UNLOCK FAIL / ERASE FAIL / PROGRAM FAIL (STM32)`	Failure while unlocking, erasing or writing over SWD; a second diagnostic line is shown	Retry; if it persists, try OPTIONS → SPEED → STM32 → SLOW; check the SWD wiring
`NO ESP (UART)`	The ESP does not respond over the UART	Check the UART wiring and the EN/BOOT pins (section 9); check the module's power supply
`SPI ATTACH FAIL (classic ESP32)`	The classic ESP32's ROM rejected that command (uncommon)	Retry the session
`NO VERIFY / ROM WITHOUT MD5`	Normal ESP8266 limitation, not a failure	—
`NO DRIVER`	That family/variant does not have flashing support yet (sections 8 and 9)	Use another family or wait for a firmware update

MESSAGE	MEANING	WHAT TO DO
`OPT FAIL + reason`	The .OPT file is invalid or asks for something not applicable to that family (section 5); the target is not touched	Correct the .OPT file according to the reason shown
`HEX FORMAT / BAD ADDRESS / NO FW INFO / NOT PROG FW / WRONG HW / BAD SIZE / SIZE MISMATCH`	These appear only when self-updating the programmer (section 6); the FIRMWARE.HEX file is not valid	Download the file again from the NORTHED website
`NO SD CARD (in USB mode)`	No card is inserted	Insert the microSD card
`Reset when connecting to the target board`	The programmer restarts when it is connected to the target	Add capacitors to the output line or use the PROG-CAP accessory board
Red LED blinking in patterns of 1 to 4 flashes with a pause, while not flashing	Serious internal failure of the programmer itself	Note the number of flashes and contact NORTHED support

LED after an operation

Solid = successful; blinking = failed (it switches off automatically when the next programming or verification starts).