

ESP8266 Non-FOTA Flash Layout: WEMOS D1 Mini Pro (16MB)

Framework: esp-open-sdk (NON-OTA Configuration)

This only applies to NON-FOTA firmware versions. The Espressif SDK family **does not natively support a 16 MB flash mapping profile** for application code execution. Instead, it handles a 16 MB chip (like the one on the WEMOS D1 Mini Pro) by addressing it as a **4 MB (32 Mbit) layout profile**. The remaining 12 MB is invisible to the linker script and can only be accessed manually using raw sector reading/writing APIs (`spi_flash_read`).

Flash execution

In order to flash ESP8266 the GPIO0 should be connected to GND. This is D3 on Wemos D1 Mini Pro.

Non-OTA Binary Structure

In a non-OTA layout, your build outputs two primary binary files:

1. **eagle.flash.bin** (Flashed at 0x00000): Contains code initialized straight into internal RAM—specifically IRAM (code required to execute with no flash latency, interrupt handlers) and DRAM (initialized global/static variables).
2. **eagle.irom0text.bin** (Flashed at offset 0x10000 or 0x20000): Contains main application code and SDK functions tagged with `ICACHE_FLASH_ATTR`. It remains in flash memory and is read dynamically via the Xtensa instruction cache (irom0).

Non-OTA Flash Map (Mapped as 4MB / 32Mbit)

In a traditional **Non-OTA** setup (referred to as `eagle.flash.bin` and `eagle.irom0text.bin` compiled via `esp-open-sdk`), the flash is mapped using **Flash Map 4** (or 6 depending on the exact SDK version configuration).

Because it is targeted as a 4 MB envelope, the default critical system parameters are pushed to the **very end of the first 4 MB segment**, not the physical end of the 16 MB chip.

Offset Address	Size	Binary Target / Description
0x00000	~40 KB	eagle.flash.bin (SDK Boot, initialized data, IRAM segments)
0x10000	Max 1 MB	eagle.irom0text.bin (Main application code executing from Flash)
0x11000 to 0x3FB000	~3 MB	Free User Space / Filesystem Space

Offset Address	Size	Binary Target / Description
0x3FC000	12 KB	esp_init_data_default.bin (System & RF Initialization parameters)
0x3FF000	12 KB	blank.bin (System Parameter Save Area / Wi-Fi settings store)
0x400000 to 0xFFFFF	12 MB	Unmapped by SDK (Raw access only via SPI flash API)

2. Crucial `esp-open-sdk` Compilation & Flashing Quirks

The Linker Script (`.ld`) Constraint

Your application compilation must use the correct linker script. Look inside your sdk directory (ld/) and ensure you target:

```
eagle.app.v6.ld
```

This forces the compiler to build the code assuming a single application image (Non-OTA) rather than splitting it into user1.bin and user2.bin layout profiles typical of an OTA environment.

The `esptool.py` Flash Size Trick

When flashing an esp-open-sdk firmware onto the WEMOS D1 Mini Pro, you must be careful with the -fs (flash size) parameter. If you set -fs 16MB, the Espressif system configuration files will be expected at the absolute end of the 16MB space (0xFFC000), which will cause a **boot loop** (rf_cal_size error) because the SDK logic is expecting them at the 4MB mark.

To successfully flash a Non-OTA open-sdk project on this board, force a 4MB size parameter during upload:

```
esptool.py --port /dev/ttyUSB0 --baud 460800 write_flash \
    -fm dio -fs 4MB \
    0x00000 eagle.flash.bin \
    0x10000 eagle.irom0text.bin \
    0x3FC000 esp_init_data_default.bin \
    0x3FF000 blank.bin
```

Note: If you explicitly want to map out the hidden 12 MB for custom partition spaces or filesystems, you must flash esp_init_data_default.bin at 0xFFC000 and blank.bin at 0xFFFF000, but you must manually ensure your custom open-sdk Makefile overrides the default FLASH_SIZE constants to prevent SDK crashes.

ESP-OPEN-SDK Firmwares

First initial flash

```
# Full erase first
esptool.py --port /dev/ttyUSB0 --baud 115200 erase_flash

# Flash esp-open-sdk/sdk/bin/esp_init_data_default.bin at 0x3fc000
esptool.py --port /dev/ttyUSB0 --baud 115200 write_flash
    --flash_mode dio --flash_size 32m
    0x000000 firmware/0x000000.bin
    0x100000 firmware/0x100000.bin
    0x3fc000 /home/oscar/esp8266/esp-open-
sdk/sdk/bin/esp_init_data_default.bin
```

Subsequential flash

```
esptool.py --port /dev/ttyUSB0 --baud 115200 write_flash
    --flash_mode qio --flash_size 32m
    0x000000 firmware/0x000000.bin
    0x100000 firmware/0x100000.bin
```

EASY ESP Firmware

```
sudo python esptool/esptool.py erase_flash
sudo python esptool/esptool.py -p /dev/ttyUSB0 -b 460800 write_flash -fm qio
0x000000 0x000000.bin
```

NODEMCU Firmware

Finally got the Wemos D1 mini pro with 16 MByte flash.

```
esptool.py erase_flash
esptool.py -b 460800 write_flash -fm qio 0x000000 bin/0x000000.bin 0x10000
bin/0x100000.bin
```

And finally it was resolved with dev version of NodeMcu prepared in <https://nodemcu-build.com/>

My best result is 17 modules in the BIN. Many was unsuccessful I reported result in GitHub (with modules) - <https://github.com/nodemcu/nodemcu-firmware/pull/1529>

Also their own fixed esptool used - <https://github.com/nodemcu/nodemcu-firmware/tree/dev/tools>

LINKS

- <https://github.com/nodemcu/nodemcu-firmware/tree/dev>
- <http://nodemcu.readthedocs.io/en/dev/en/>
- <https://github.com/nodemcu/nodemcu-firmware/pull/1529>
- <https://nodemcu-build.com/stats.php>
- <http://tiestvangool.ghost.io/2016/08/13/flash-nodemcu-esp8266/>
- <http://www.letscontrolit.com/wiki/index.php/ESPEasy#Stable>

From:

<https://wiki.oscardegroot.nl/> - **HomeWiki**

Permanent link:

<https://wiki.oscardegroot.nl/doku.php?id=esp:esp8266:open-sdk:flash-nonfota-firmware&rev=1790000564>

Last update: **2026/09/21 14:22**

