HIGH-SPEED FILE DOWNLOAD to SPIFFS on ESP32

Embedded Firmware Design

A PROJECT REPORT

Submitted by

SHAIKH MOHAMMAD UMAR

BACHELOR OF ENGINEERING

in

ELECTRONICS & COMMUNICATION ENGINEERING

Government Engineering College Bharuch

September 2025

PROJECT OVERVIEW

The goal of this project is to create an ESP32 Dev Kit high-performance firmware solution that supports secure, high-speed HTTPS-based file downloads and optimum storage in the on-board SPI Flash File System (SPIFFS).

The firmware will be design to provide a minimum transfer rate of 400 KBps during simultaneous download and writing operations to ensure reliable data transfer even under limited embedded system resources. The implementation utilizes the ESP-IDF framework and incorporates the following core capabilities:

- HTTPS Client File Access: Secure file access from public URLs.
- Buffer Optimization: Tailored buffer sizes to ensure optimal throughput while keeping memory usage under control.
- SPIFFS Storage Management: Optimized write operations with monitoring for available space.
- Error Handling: Crash recovery for Wi-Fi disconnections, server timeouts, and flash memory constraints.

This project also has a performance validation framework, taking detailed logs of download speeds and write throughput. The validation ensures compliance with the 400 KBps benchmark and shows resistance to real-world error conditions.

In conclusion, the deliverable demonstrates how an ESP32-based embedded system can be optimized to handle secure, high-speed, and fault-tolerant data processing, applicable to industry use cases like OTA updates, IoT data acquisition, and edge device content delivery.

TABLE OF CONTENT

Contents

PROJECT OVERVIEW 2
TABLE OF CONTENT 3
INTRODUCTION 4
Problem Statement: 4
Block Diagram 4
SSL certificate: 4
SPIFFS partition: 4
partitions.csv: 5
Menu Configuration 5
Steps to compile, Flash and test firmware: - 7
Optimization: - 8
Result 9
Reference 10

INTRODUCTION

Problem Statement:

Develop a firmware application for the ESP32 Dev Kit that downloads a file via HTTPS from a public URL and writes it to SPIFFS (SPI Flash File System) at a speed of at least 400KBps.

Block Diagram

SSL certificate:

If you want to download any file from a website using HTTPS on ESP32, you must also give ESP32 the SSL certificate of that website. This certificate is like an online ID card which tells ESP32 “yes, this website is real and safe.” To get this certificate from your computer, you don’t need any heavy software, only CMD is enough.

- Firstly we need to download OpenSSL.exe in system we use.
- Then open respective .bin folder in CMD window
- Next to get certificate type “openssl s_client -showcerts -connect <your website>”
- Paste it to your .pem file in VS code. Also add EMBED_TXTFILES "server_cert.pem" in Cmake File.

SPIFFS partition:

As we are using custom partition so we need to select custom partition table in menuconfig. Run idf.py menuconfig in terminal, go to: **Partition Table → Partition Table**. By default, it shows "Single factory app, no OTA" or "Factory app, two OTA partitions". Change it to **Custom partition table CSV**. Then it will ask for the **CSV file path**. Normally you keep it in your project root folder as `partitions.csv`

partitions.csv:

This was the table which was added in code.

# Name	Type	SubType	Offset	Size	Flags
nvs	data	nvs	0x9000	0x5000	
otadata	data	ota	0xe000	0x2000	
app0	app	ota_0	0x10000	0x180000	
spiffs	data	spiffs	0x190000	0x260000	

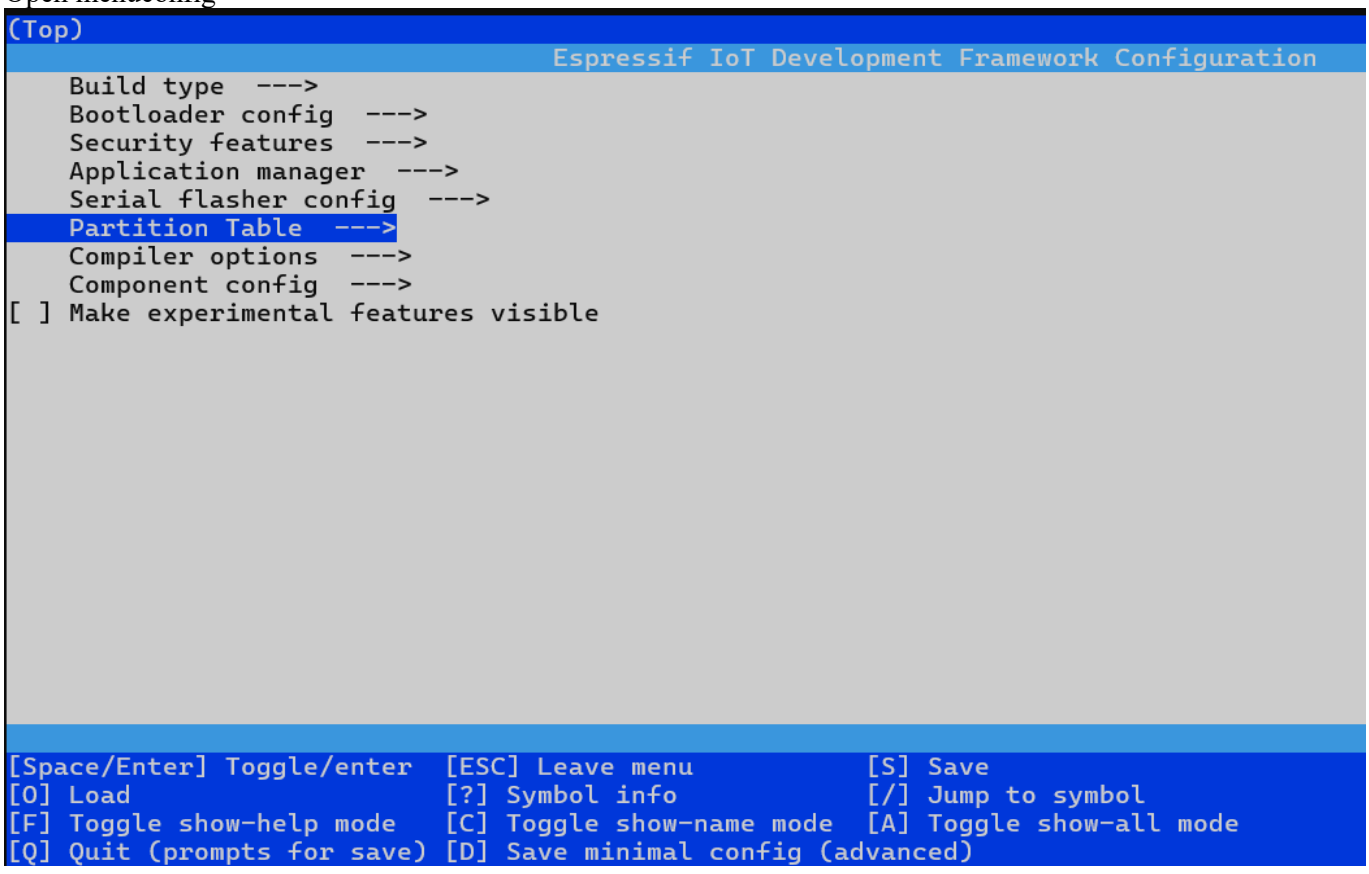
- **nvs (Non-Volatile Storage)**
  - Type: data, Subtype: nvs
  - Address: 0x9000, Size: 0x5000 (20 KB approx.)
  - Used for storing key-value pairs (like WIFI credentials, configs).
- **otadata (OTA Data Partition)**
  - Type: data, Subtype: ota
  - Address: 0xe000, Size: 0x2000 (8 KB)
  - Keeps track which firmware slot is active (app0/app1) in OTA.
- **app0 (Application Partition)**
  - Type: app, Subtype: ota_0
  - Address: 0x10000, Size: 0x180000 (1.5 MB approx.)
  - This is where your main firmware binary gets flashed.
- **spiffs (SPI Flash File System)**
  - Type: data, Subtype: spiffs
  - Address: 0x190000, Size: 0x260000 (~2.3 MB)
  - This is your "hard disk" inside flash. You can store files (cert.pem, config.json, downloaded files, etc.).

So out of 4 MB flash:

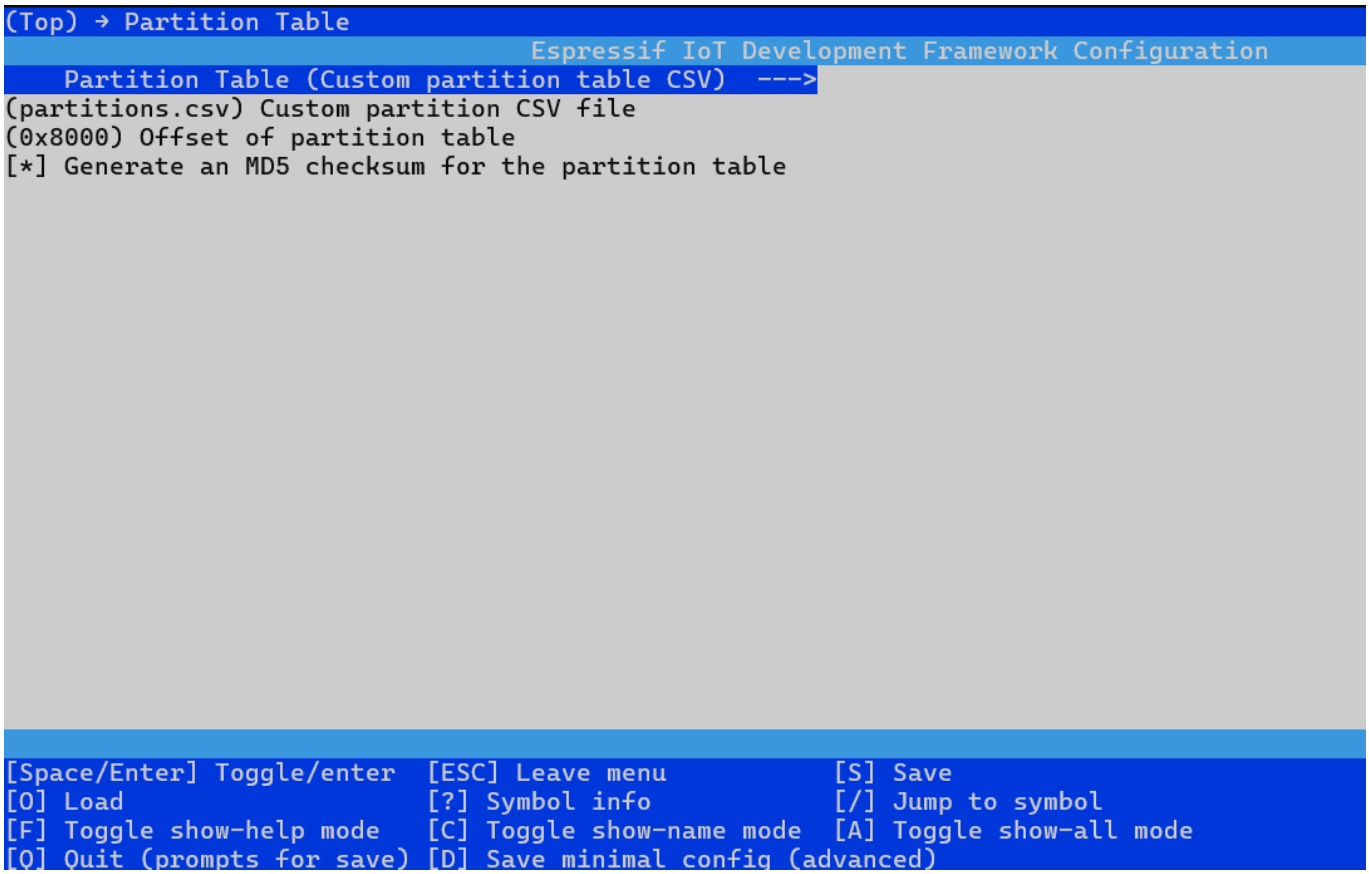
- Small chunks (NVS, OTA data) are reserved at start,
- 1.5 MB goes to firmware,
- Remaining ~2.3 MB is left for SPIFFS.

Menu Configuration

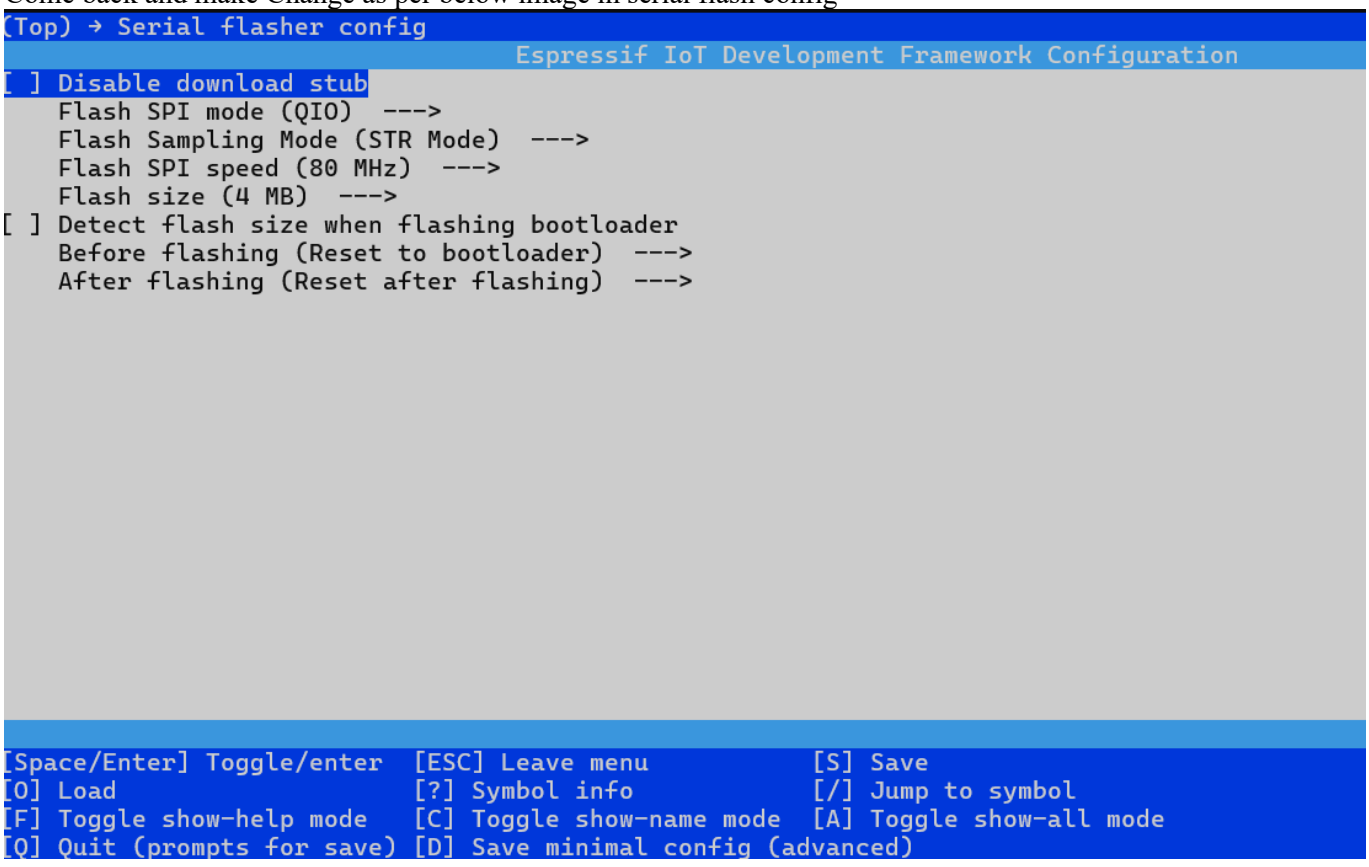
Open menuconfig



Select custom partition table



Come back and make Change as per below image in serial flash config



SAVE <ESC + Y>

Steps to compile, Flash and test firmware: -

Run this from **ESP-IDF Command Prompt** (not VS Code, not PowerShell):

```
C:\Espressif\frameworks\esp-idf-v5.4.2\install.bat
```

This script will:

- Download required Python if missing.
- Create the virtual environment (`C:\Espressif\python_env\idf5.4_py3.13_env`).
- Install all Python dependencies (`pip install -r requirements.txt`).
- After install finishes, run:
- `C:\Espressif\frameworks\esp-idf-v5.4.2\export.bat`
- If successful, your prompt will show `[esp-idf]` at the beginning.
- Check versions:
    - `idf.py --version`
    - `python –version`
- Correct usage is:
    - PowerShell: export.ps1
    - CMD: export.bat

  USE `idf.py` in terminal to know each command

1. Install ESP-IDF in VS Code

- Install **VS Code** from official site.
- Go to Extensions (**Ctrl+Shift+X**) → install **ESP-IDF**.
- Click ESP-IDF logo → **Install ESP-IDF** → choose **Express Install**.
  (It downloads ESP-IDF, Python, Ninja, CMake, etc. one time setup).

2. Create Project

- In ESP-IDF menu → **Create Project** → choose `hello_world`.
- Save in folder (e.g., `C:\Users\acer\proj`).
- Or run:
- `idf.py create-project my_project`

3. Select COM Port (ESP32 on COM8)

- Check in **Device Manager** → **Ports** (e.g., COM8).
- In VS Code → Command Palette (**Ctrl+Shift+P**) → ESP-IDF: Select Port → COM8.
- Or CLI:
- `idf.py set-target esp32`
- `idf.py -p COM8`

4. Build Project

- In VS Code: Command Palette → **ESP-IDF: Build Project**.
- Or CLI:
- `idf.py build`
- First build takes time. Success = *"Build finished"*.

5. Flash Firmware

- In VS Code: Command Palette → **ESP-IDF: Flash**.
- Or CLI:
- `idf.py -p COM8 flash`

6. Monitor Logs

- After flashing: Command Palette → **ESP-IDF: Monitor**.
- Or CLI:
- `idf.py -p COM8 monitor`
- See live logs (WiFi connect / HTTPS download / SPIFFS save).
- Stop with **Ctrl+]**.

7. Menuconfig (Change Settings)

- Run:
- `idf.py menuconfig`
- Change SPIFFS Partitions setup, flash config, etc.
- Press **S** (save), then **Q** (quit).
- Build again after changes.

8. Typical Workflow

1. Edit code in `main/`.
2. `idf.py build`
3. `idf.py -p COM8 flash`
4. `idf.py -p COM8 monitor`

Optimization: -

Goal and baseline

- Target: ~400 KB/s sustained HTTPS download to SPIFFS.

Code-level optimizations

- Larger, fewer I/O operations:
  - Network read size standardized to CHUNK = 16 KB (best balance for heap/TLS on ESP32).
  - File I/O buffered with setvbuf(..., _IOFBF, 16 KB) to batch flash writes and reduce stalls.
- HTTP client tuning:
  - Keep-alive enabled.
  - HTTP RX/TX buffers set to 8 KB to conserve heap while maintaining steady flow.
  - Timeout increased to 20 s to absorb TLS + network jitter on larger buffers.
- Wi-Fi throughput hints:
  - Power-save disabled.
  - TX power maximized: esp_wifi_set_max_tx_power (78).
- Task scheduling:
  - Download task given higher priority and pinned to the secondary core to keep up with network and flash I/O.

Filesystem and flash

- SPI flash already at 80 MHz QIO, which minimizes flash latency.

Wi-Fi/AP environment tips

- Keep device close to AP.

Memory and build configuration

- Kept buffers within heap limits after observing malloc failures with very large chunks:
  - Settled on 16 KB network chunk + 16 KB file buffer, 8 KB HTTP buffers.
- Eliminated IRAM/DRAM pressure:
  - Switched from RAM app to normal flash app build.
  - Disabled unnecessary "*_IN_IRAM" options and heavy debug features (e.g., GDB stub, Ethernet) to free internal RAM.

Summary

- We improved throughput by reducing per-chunk overhead, avoiding Wi-Fi power save gaps, batching flash writes, right-sizing buffers to fit heap, and removing IRAM/DRAM pressure in the build. We also ensured robust HTTPS by attaching a proper trust source and handling TLS timing prerequisites.

```
I (5460) wifi:dp: 1, bi: 102400, li: 3, scale listen interval from 307200 us to 307200 us
I (5470) wifi:dp: 2, bi: 102400, li: 4, scale listen interval from 307200 us to 409600 us
I (5470) wifi:AP's beacon interval = 102400 us, DTIM period = 2
I (6480) esp_netif_handlers: sta ip: 10.244.100.62, mask: 255.255.255.0, gw: 10.244.100.137
I (6480) https_spiffs_dl: Got IP: 10.244.100.62
I (6480) https_spiffs_dl: Connected to AP
I (6480) main_task: Returned from app_main()
I (6480) https_spiffs_dl: Preparing HTTP client for URL: https://speed.hetzner.de/1MB.bin
I (6890) wifi:<ba-add>idx:0 (ifx:0, a2:ef:44:db:5e:18), tid:0, ssn:4, winSize:64
I (8230) https_spiffs_dl: Content-Length: 138
I (8230) https_spiffs_dl: Download complete (read returned 0)
I (8230) https_spiffs_dl: Total bytes written: 138 bytes
I (8240) https_spiffs_dl: Elapsed time: 0.000 s
I (8240) https_spiffs_dl: Throughput: 774.52 KB/s
I (8240) https_spiffs_dl: Throughput target met: 774.52 KB/s >= 400 KB/s
I (8250) https_spiffs_dl: Download task finished
```

Result

Looking for SPIFFS file from ESP32 use below command in terminal

`esptool.py --chip esp32 --port COM<your-port> --baud 921600 read_flash 0x190000 0x260000 spiffs_dump.bin`

```
PS C:\Users\acer\proj> esptool.py --chip esp32 --port COM8 --baud 921600 read_flash 0x190000 0x
260000 spiffs_dump.bin
esptool.py v4.9.0
Serial port COM8
Connecting....
Chip is ESP32-D0WD-V3 (revision v3.1)
Features: WiFi, BT, Dual Core, 240MHz, VRef calibration in efuse, Coding Scheme None
Crystal is 40MHz
MAC: 00:4b:12:2f:fa:b8
Uploading stub...
Running stub...
Stub running...
Changing baud rate to 921600
Changed.
Configuring flash size...
2490368 (100 %)
2490368 (100 %)
Read 2490368 bytes at 0x00190000 in 31.1 seconds (640.1 kbit/s)...
Hard resetting via RTS pin...
PS C:\Users\acer\proj>
```

This will give you `spiffs_dump.bin` file in project folder
Hence, we got our file downloaded successfully

Reference

GitHub Repo: - https://github.com/espressif/esp-idf/tree/v5.4.1/examples

Guide 1: - SPIFFS

Guide 2: - FreeRtos

Guide 3: - HTTPS

Guide 4: - DOCs

Guide 5: - https://esp32tutorials.com/esp32-spiffs-esp-idf/

Partitions: - partition.html