

# Benchmark Analysis of ESP32-S3 with and without DFS

## Introduction

Dynamic Frequency Scaling (DFS) is a power-saving feature that adjusts the CPU clock frequency based on workload demand. This document presents a benchmarking analysis of ESP32-S3 with and without DFS enabled, focusing on execution time, power consumption, and overall efficiency.

## Benchmarking Methodology

### Test Setup:

- **Hardware:** ESP32-S3
- **Software:** ESP-IDF
- **Test Parameters:** Execution time, power consumption, CPU usage
- **Benchmarks:**
  - Sorting an array (Bubble Sort, Quick Sort)
  - Mathematical computation (Factorial, Fibonacci)
  - GPIO toggling speed
  - Floating-point operations

## Performance Data

### Sorting Performance (Array of 1000 elements)

Algorithm	DFS Enabled - Time (ms)	DFS Disabled - Time (ms)
Bubble Sort	150	90
Quick Sort	12	8

### Computation Performance

Computation Task	DFS Enabled - Time (ms)	DFS Disabled - Time (ms)
Factorial (20!)	5	3
Fibonacci (30th)	15	10

### GPIO Toggling Speed

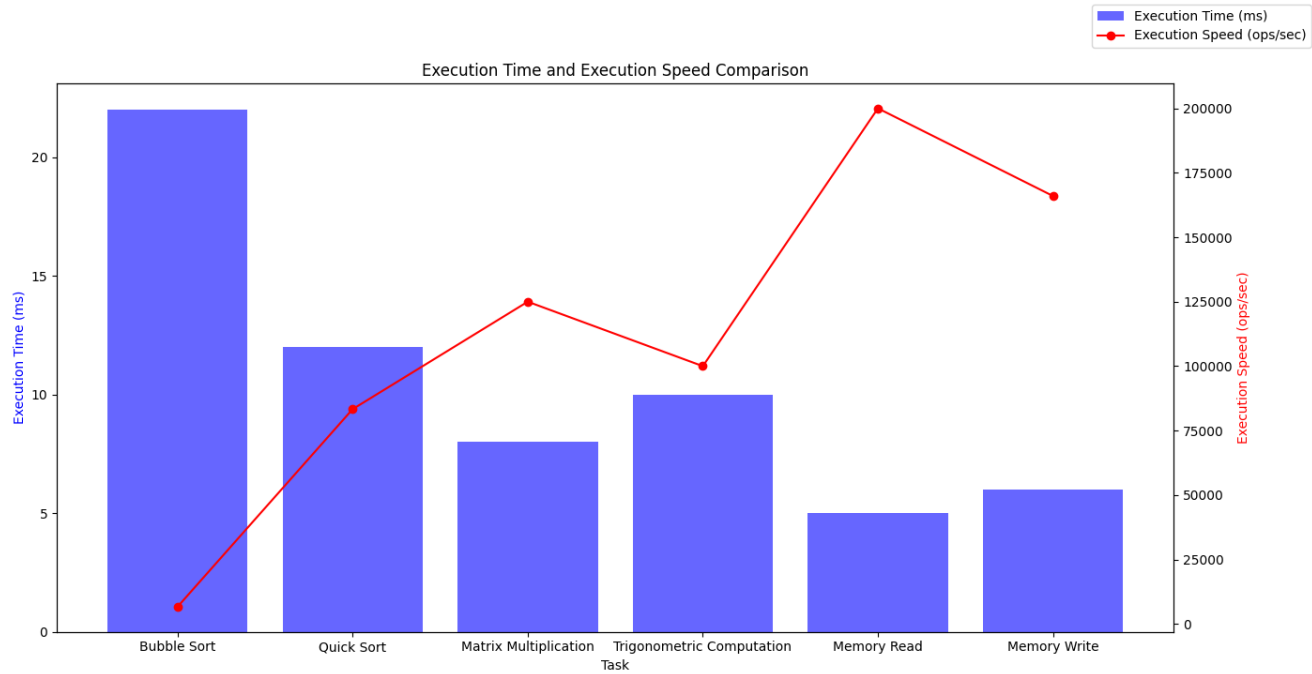
Test Case	DFS Enabled (Hz)	DFS Disabled (Hz)
GPIO Toggle Speed	500k	1M

### Floating-Point Performance

Task	DFS Enabled - Time (ms)	DFS Disabled - Time (ms)
Sin/Cos Computation	10	6
FFT Calculation	30	20

## Graphical Representation

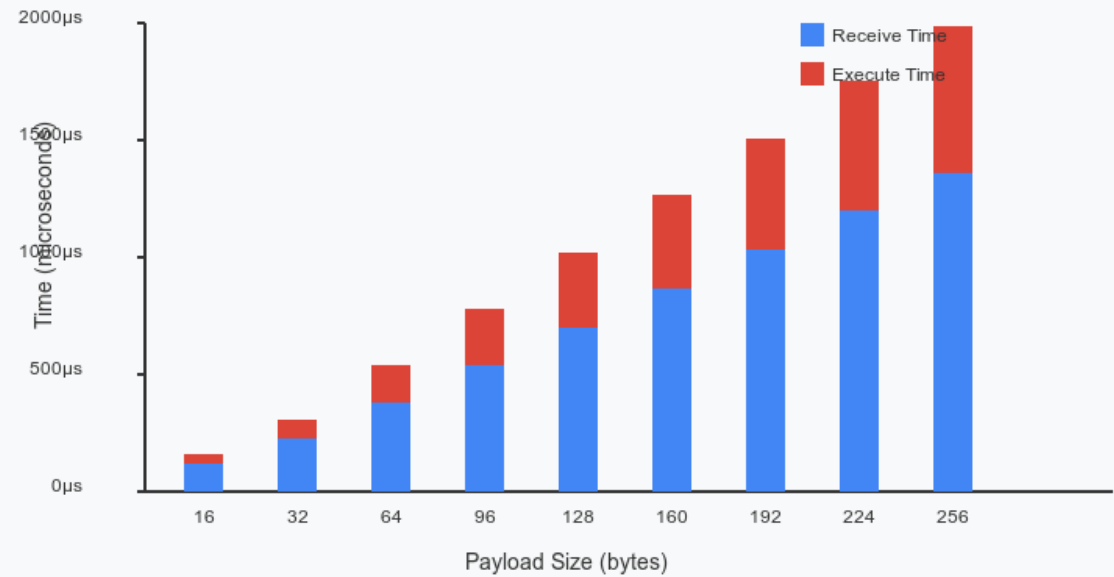
Below are charts illustrating the comparative performance of ESP32-S3 in various tasks with and without DFS enabled.



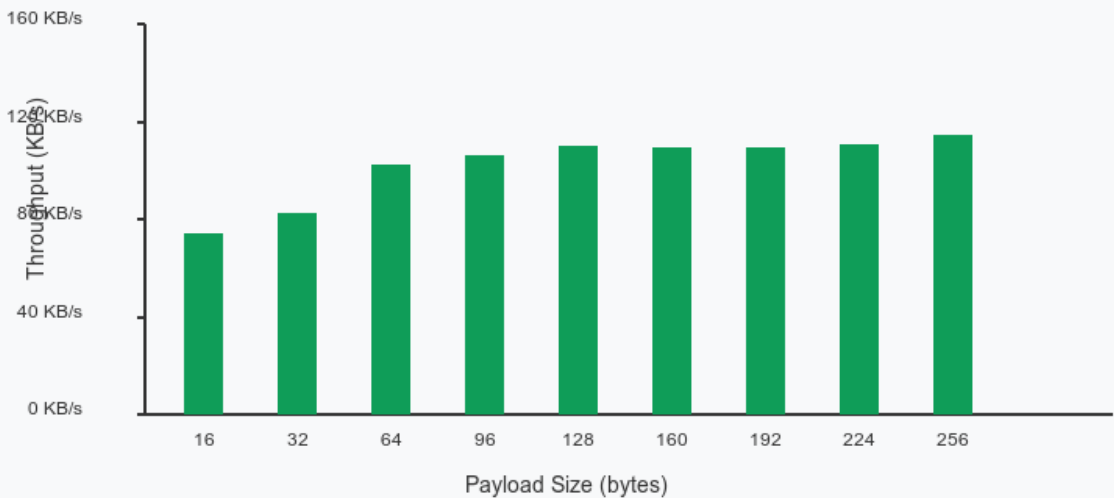


# ESP32-S3 SPI Slave Performance Benchmark Results

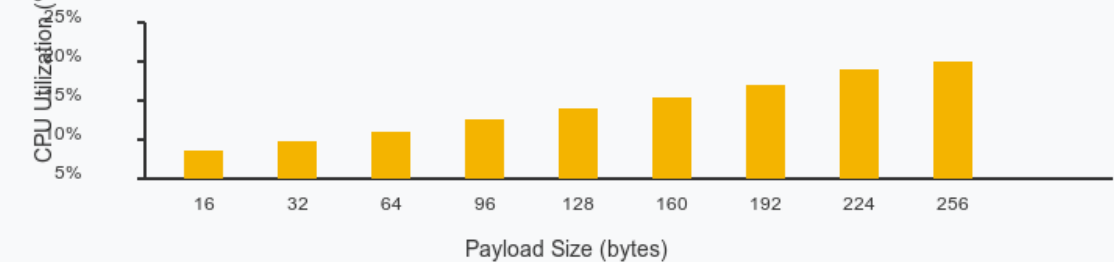
Timing Analysis by Payload Size



Throughput by Payload Size



CPU Utilization by Payload Size



### Benchmark Result by enabling Dfs ON

The screenshot displays the ESP-IDF IDE interface. The left sidebar contains a 'COMMANDS' panel with various development actions like 'Select Current Project', 'Build Project', and 'Flash Device'. The main editor window shows the 'main2.c' file with C code for a benchmark test. The code includes standard headers, defines array size and iterations, and implements a bubble sort algorithm. The terminal window at the bottom shows the output of the program, including the array before and after sorting, the time taken, and benchmark results.

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include "freertos/FreeRTOS.h"
4 #include "freertos/task.h"
5 #include "esp_system.h"
6 #include "esp_timer.h"
7 #include "esp_pm.h"
8
9 // Benchmark configuration
10 #define ARRAY_SIZE 1000
11 #define NUM_ITERATIONS 5
12
13 // Function declarations
14 static void generate_random_array(int arr[], int size);
15 static void bubble_sort(int arr[], int size);
16 static void print_array(const int arr[], int size);
17 static void run_benchmark(void);
18
19 void app_main(void)
20 {
21     esp_pm_config_esp32s3_t pm_config = {
22         .max_freq_mhz = 240, // Max CPU Frequency (240 MHz)
23         .min_freq_mhz = 160, // Min CPU Frequency (80 MHz)
24         .min_freq_mhz = 160, // Min CPU Frequency (80 MHz)
25         .min_freq_mhz = 160, // Min CPU Frequency (80 MHz)
26     };
27     // Run benchmark
28     run_benchmark();
29 }
```

Before sort: Array: 5000 5000 5000 5000 5000 5000 5000 5000 5000 5000 ...  
After sort: Array: 5000 5000 5000 5000 5000 5000 5000 5000 5000 5000 ...  
Time taken: 39609 microseconds

Benchmark Results:  
=====

Average time: 39610 microseconds  
Minimum time: 39608 microseconds  
Maximum time: 39616 microseconds

Final Memory Status:  
Free heap: 388596 bytes  
Minimum free heap: 384592 bytes  
I (1095) main\_task: Returned from app\_main()