DOI: 10.35598/mcfpga.2026.018

Performance Efficiency of C/C++ and TinyGo Programming Languages for ESP32 Microcontrollers

Hlib Serbskyi
Department of Media Engineering and Information Radio Electronic System
Kharkiv National University of Radio Electronics
Kharkiv, Ukraine
hlib.serbskyi@nure.ua

Andrii Volokh
Scientific adviser
ORCID 0009-0003-5375-8835
Department of Media Engineering and Information Radio Electronic System
Kharkiv National University of Radio Electronics
Kharkiv, Ukraine
andrii.volokh@nure.ua

***Abstract* — Due to their low power consumption and integrated wireless interfaces, ESP32 microcontrollers are widely used in Internet of Things (IoT) devices. Alongside C/C++, the primary development language for this platform, alternatives such as Rust and TinyGo are gaining popularity by offering higher levels of memory safety and code reliability. This article presents a performance comparison between C/C++ and TinyGo, specifically through the implementation of CRC32 and Fast Fourier Transform (FFT) algorithms. The results demonstrate that TinyGo provides performance close to that of C/C++, while offering concise syntax and efficient concurrency mechanisms (goroutines). This makes TinyGo a promising tool for embedded systems development, particularly where effective multitasking and minimal latency are required.**

Keywords — Microcontroller, ESP32, C/C++, TinyGo, Performance efficiency.

I. Introduction

In IoT devices and embedded systems, microcontrollers (MCUs) are utilized due to their compact size, low power consumption, and sufficient performance for signal processing tasks. When building such systems, MCUs typically perform functions for collecting, processing, and transmitting information to upper-level devices (gateways or cloud servers). Thanks to its high-performance dual-core 32-bit architecture, integrated wireless interfaces, extensive range of available features, and robust software support, the ESP32 MCU is a popular choice for IoT systems based on its cost-to-performance ratio [1].

While C/C++ remains the primary language for programming ESP32 MCUs, languages such as Rust, TinyGo, and MicroPython are also supported. Consequently, there is significant scientific interest in comparing the complexity and efficiency of alternative languages for ESP32 programming, particularly through the implementation of common data and signal processing algorithms. This study evaluates the performance of the Cyclic Redundancy Check (CRC32) and Fast Fourier Transform (FFT) algorithms on the ESP32, implemented in C/C++ and TinyGo.

II. Methods and Results of Performance Comparison

A. Specifics of Programming Language Implementation

The choice of a programming language for microcontrollers (MCUs) is determined by several critical aspects: the compiler's level of support for the processor's architectural features; the availability of Hardware Abstraction Libraries (HAL) for peripheral interfacing; memory management mechanisms; the potential for deep code optimization; and the specifics of interaction with the runtime environment or a Real-Time Operating System (RTOS).

C/C++ is the de facto standard for ESP32 MCUs due to official manufacturer support from Espressif within the ESP-IDF framework. The language provides direct access to hardware resources via HAL libraries. Memory management in C/C++ is primarily manual or deterministic (through standard library mechanisms), while multitasking is implemented via integration with FreeRTOS.

TinyGo is a specialized version of the Go language tailored for embedded systems [2]. It offers a simplified syntax compared to C++ and features automatic memory management. A key feature of TinyGo is its built-in support for concurrency (goroutines) at the language scheduler level, allowing for multitasking without the need for an external OS. However, due to its pursuit of universality, TinyGo currently has limited support for specific ESP32 peripherals, notably Wi-Fi and Bluetooth modules, as well as power-saving modes (Sleep Modes).

B. Selection of Algorithms for Performance Testing

To evaluate the performance of the ESP32 using C/C++ and TinyGo, the implementations of the CRC32 and FFT algorithms were selected. These algorithms were chosen based on the following criteria: their inclusion in existing benchmarks oriented towards embedded systems testing (e.g., MiBench); the availability of stable and well-tested implementations within the ESP-IDF open-source libraries (the framework provided by Espressif) [3]; and the ability to

easily verify the algorithm's correctness after porting it to TinyGo.

For testing purposes, an ESP32-WROOM-32 development board was used (Fig. 1), with the CPU frequency set to 160 MHz, representing one of the standard operating frequencies used in many practical applications.

Fig. 1 – Illustration of ESP32-WROOM-32 DevKit

The performance evaluation was based on averaging the results of 100 iterations across varying input data sizes (from 0 to 1024 bytes for CRC32 and a corresponding number of points for FFT). The C/C++ code was compiled with the -O2 optimization level to maximize execution speed. The C/C++ software implementation relied on FreeRTOS mechanisms, whereas TinyGo utilized its built-in goroutine scheduler, enabling multitasking without the need for external frameworks [4].

C. Results of Practical Implementation and Testing

Fig. 2 presents the testing results for the average execution time of the CRC32 algorithm on the ESP32, implemented in C and TinyGo. As the results indicate, the TinyGo implementation proved to be as fast as, or even slightly faster than, the C implementation in terms of execution speed.

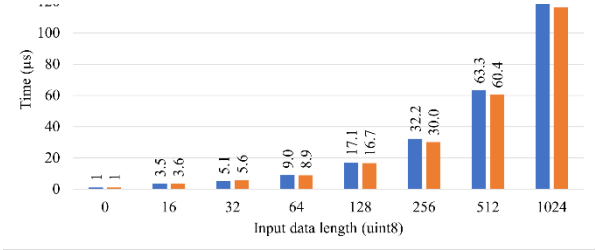

Fig. 2 - Average execution times of the CRC-32 algorithm

Fig. 3 presents the testing results for the average execution time of the FFT algorithm on the ESP32, implemented in C and TinyGo. The input data for the FFT algorithm testing were passed as complex integers. Despite the shorter execution time of the algorithm implemented in C, the use of TinyGo demonstrated only slightly lower performance.

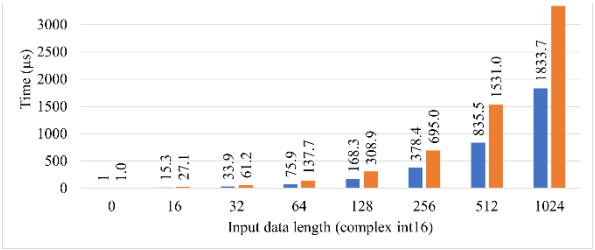

Fig. 3 - Average execution times of the FFT algorithm

In the comparative analysis of the TinyGo and C/C++ implementations, an extremely low standard deviation of iteration execution times was observed (near zero within the margin of measurement error). This stability in timing metrics is attributed to the efficiency of the built-in TinyGo scheduler and the minimization of task context-switching overhead compared to full-featured real-time operating systems (RTOS). The high determinism of execution time in TinyGo programs represents a significant advantage for developing real-time systems based on the ESP32 MCU, where the predictability of system response to external events is critically important.

III. CONCLUSIONS

This study conducted a comparative performance analysis of the C/C++ and TinyGo programming languages for the ESP32 MCU, using CRC32 and FFT algorithms as examples. The testing results demonstrated that TinyGo provides a level of performance that is fully comparable to C/C++ implementations and, in certain cases, matches it in terms of efficiency.

A significant advantage of TinyGo proved to be its high execution time determinism, achieved by minimizing runtime overhead, which makes the language a promising choice for real-time systems. Furthermore, as a higher-level language compared to C/C++, TinyGo substantially simplifies the development process and accelerates software production for the ESP32 architecture without significant performance trade-offs.

REFERENCES

[1] A. Maier, A. Sharp, Y. Vagapov “Comparative Analysis and Practical Implementation of the ESP32 Microcontroller Module for the Internet of Things” // In Proceedings of the 2017 Internet Technologies and Applications (ITA), IEEE: Piscataway, NJ, USA, 2017. PP. 143–148.

[2] TinyGo. URL: https://tinygo.org/ (accessed on: 27.03.2026).

[3] Espressif DSP Library. URL: https://docs.espressif.com/projects/esp-dsp/en/latest/esp-dsp-library.html (accessed on: 31.03.2026).

[4] Why Go Instead of Rust? URL: https://tinygo.org/docs/concepts/faq/why-go-instead-of-rust/ (accessed on: 31.03.2026).