PROCEEDINGS OF THE YEREVAN STATE UNIVERSITY

Physical and Mathematical Sciences 2026, **60**(1), p. 54–63

Informatics

OPERATIONAL ENVIRONMENT FOR DYNAMIC LOADING AND MANAGEMENT OF MODULAR APPLICATIONS IN EMBEDDED SYSTEMS

M. V. AVETISYAN [*], K. A. AVETISYAN [**]

Chair of Programming and Information Technologies, YSU, Armenia

Ensuring modularity under strict resource constraints is a critical challenge in the design of embedded systems. This paper presents "Part Only RTOS", an operational environment developed for devices based on 8-bit ATmega2560 microcontrollers. The core feature of the proposed system is the ability to dynamically load and execute application modules (in .bin format) from an SD card without requiring full hardware reprogramming. The developed three-tier architecture, comprising the Launcher, Bootloader, and Kernel, ensures reliable software validation via unique pattern matching and efficient memory utilisation. Experimental results demonstrate that the system kernel occupies less than 2.5 *KB* of Flash memory, and the loading time for a 10 *KB* module is approximately 132 *ms*. This solution enables the creation of flexible and high-performance modular embedded systems on low-power microcontroller platforms.

https://doi.org/10.46991/PYSUA.2026.60.1.054

MSC2020: Primary: 68M11; Secondary: 68N01.

Keywords: Real-Time Operating System (RTOS), microcontrollers, Dynamic Loading, Modularity, Bootloader, Embedded Systems, ATmega2560, Memory management.

Introduction. The rapid evolution of the Internet of Things (IoT), wireless sensor networks (WSN), and autonomous embedded devices has fundamentally transformed the requirements for resource management in low-power computing environments. Despite the emergence of high-performance 32-bit architectures, 8-bit microcontrollers, particularly the Microchip AVR ATmega series, remain a cornerstone of industrial and educational applications due to their cost-effectiveness, deterministic behaviour, and low power consumption [1, 2].

[*] E-mail: mkhitar.avetisyan@edu.ysu.am

[**] E-mail: kajik.avetisyan@ysu.am

However, the traditional monolithic firmware approach, where the entire application logic is compiled and flashed as a single binary presents, a significant bottleneck in modern modular systems. In such architectures, any functional update or the addition of a new task requires a full memory erase and a complete reflashing cycle via external programming hardware or In-System Programming (ISP) interfaces [3]. This lack of flexibility limits the adaptability of devices in remote or field-deployed environments, where physical access for reprogramming is often impractical.

While several Real-Time Operating Systems (RTOS) have been adapted for AVR platforms, such as FreeRTOS or specialised kernels discussed by Logutenko et al. [4], they often suffer from high overhead in terms of Flash and SRAM footprint. Moreover, most existing solutions do not natively support dynamic application loading from external storage (e.g., SD cards) on 8-bit platforms, primarily due to the architectural limitations of Harvard architecture microcontrollers regarding code execution from non-Flash regions and the complexity of runtime memory management [5, 6].

The proposed "Part Only RTOS" addresses these challenges by introducing a hybrid, modular operational environment designed specifically for the ATmega2560. The primary innovation of this system lies in its ability to dynamically load, validate, and execute modular applications (.bin files) from an SD card without requiring a full system reboot or external computer intervention. This is achieved through a coordinated three-tier architecture:

A High-Level Launcher: Providing a user interface via UART for file system navigation and module selection.

A Specialized Bootloader: Responsible for sector-specific Flash writing and integrity verification using unique "Kernel Pattern Matching."

A Lightweight Kernel: Managing task execution, crash detection via "Stack Canary" mechanisms [7], and future energy-aware scheduling [8].

By implementing a strict memory footprint (under 2.5 *KB* for the Kernel) and achieving sub-200 *ms* loading times for 10 *KB* modules, this research demonstrates that modular flexibility and RTOS-like capabilities can be efficiently integrated into resource-constrained 8-bit environments. The following sections detail the problem formalisation, the technical implementation of the dynamic loading mechanism, and the experimental results regarding memory fragmentation and power efficiency.

Problem Description. The architecture of 8-bit AVR microcontrollers, such as the ATmega2560, is based on a modified Harvard architecture, where program memory (Flash) and data memory (SRAM) are physically separate. Traditional firmware deployment models utilise the entire Flash space as a monolithic block. This creates several critical issues in modular development:

1. **Fixed Functionality:** Once the hex file is written to Flash, the device is locked into a specific set of tasks. Changing a single logic module requires re-compiling the entire project and reflashing the chip.

2. **Memory Volatility and Safety:** Implementing dynamic loading requires a reliable method to jump between code segments. Without a supervisory kernel,

a jump to an uninitialized or corrupted Flash address can lead to undefined behaviour or permanent system hangs [9].

3. **Validation Gap:** On-the-fly loading from external media (SD card) bypasses the standard verification tools of IDEs. Thus, a mechanism is needed to ensure that the binary file on the SD card is compatible with the resident Kernel version.

To solve these, the Part Only RTOS formalises the application as a "Module" that must reside in a predefined Flash section, linked against a specific library (*kernel_lib*), which embeds a 27-byte validation signature.

Proposed System Architecture and Solution. The proposed system is divided into three functional domains to ensure isolation between the management layer and the application logic. The proposed environment is based on a three-tier architecture designed to decouple the hardware-specific operations from the application logic. The structural diagram and the high-level data flow between these components are illustrated in Fig. 1.

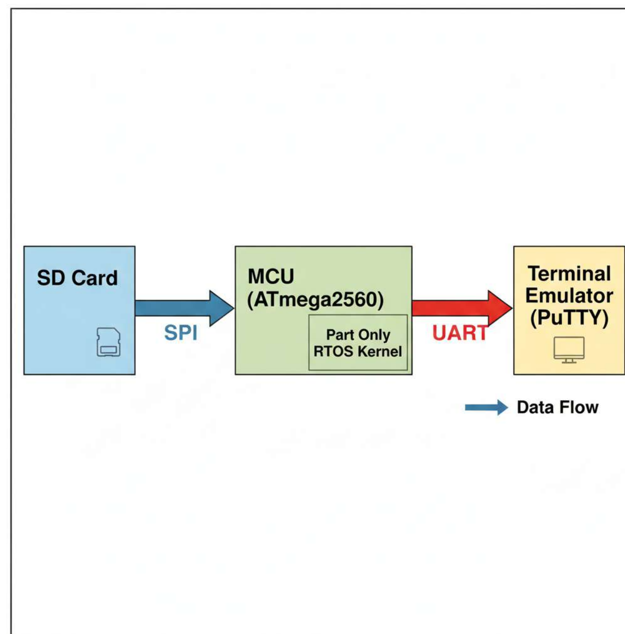

Fig. 1. Structural diagram and data flow of the system.

Module Structure and Execution Model. The implementation of dynamic modularity on the 8-bit ATmega2560 architecture is based on a fixed-address execution model. In this approach, each application module is developed as a standalone binary image, specifically linked to start at the base Flash address of 0×0000. By utilising a shared interface through "kernel.h" and a static library "kernel.a", the system ensures that all external function calls and hardware references are resolved during the compilation phase. This static relocation strategy is crucial for resource-constrained systems as it eliminates the overhead of a dynamic loader or a Memory Management Unit (MMU).

The Application Binary Interface (ABI) follows standard C-calling conventions, where the kernel exposes essential system services like UART communication and timer management via the shared library. When a new module is loaded, the Bootloader overwrites the application section of the Flash memory, including the Interrupt Vector Table (IVT). This mechanism allows the newly loaded module to take partial control over the hardware interrupts. While the module can define custom Service Routines for specific application-level tasks, critical system interrupts-such as the system tick for the scheduler and UART communication-remain managed by the Kernel's resident section. By preserving these core vectors in the protected Bootloader and Kernel space, the system ensures that modular applications cannot inadvertently destabilize the underlying operational environment, maintaining a robust separation between user-level logic and system-level services.

Hardware Interface and Simulation Model. The physical layer consists of the ATmega2560 MCU, an SD card module interfaced via the Serial Peripheral Interface (SPI), and a UART-to-USB bridge for user interaction. Before physical prototyping, a full-scale simulation model was developed in Proteus VSM to validate the SPI communication timings and Flash page-writing cycles.

Software Integrity and Validation Logic. The security model of "Part Only RTOS" currently utilises a 27-byte unique identifier KERNEL_LIB:BUILD_NUMBER:0.1 embedded within the binary modules. It is important to characterize this mechanism primarily as a structural validation tool rather than a comprehensive cryptographic defence. This signature ensures that the Bootloader only recognises and processes files specifically compiled for the compatible version of the Kernel, preventing the accidental execution of foreign or malformed binaries.

To mitigate the risks of data corruption during SD card read operations or potential unauthorised modifications, the architecture is designed to support the implementation of a Cyclic Redundancy Check (CRC-32). By calculating and verifying the CRC checksum during the page-by-page flashing process, the system can ensure high-level data integrity without the prohibitive computational costs of asymmetric encryption. This tiered approach to security allows the system to remain lightweight while providing a clear path for future security optimisation.

Dynamic Loading Mechanism (The Launcher). The Launcher is the primary interface between the user and the file system. Using the Petit FatFs (PFF) library [3], it scans the root directory of the SD card for files with the .bin extension. The loading process follows a strict sequence:

1. *Power On*: The Launcher identifies the selected file and reads the header.

2. *Signature Verification*: It searches for the 27-byte "Kernel Pattern." If the pattern is not found, the loading is aborted to prevent "bricking" the device with incompatible code.

3. *Bootloader Handover*: Once verified, control is passed to the Bootloader section (starting at address $0 \times 3E000$).

The Specialised Bootloader. Unlike standard bootloaders used for ISP, this bootloader is optimised for internal Flash-to-Flash transfers. It reads sectors from the SD card and uses the `spm` (Store Program Memory) instruction to write the data into the Application Flash section in 256-byte pages (Fig. 2).

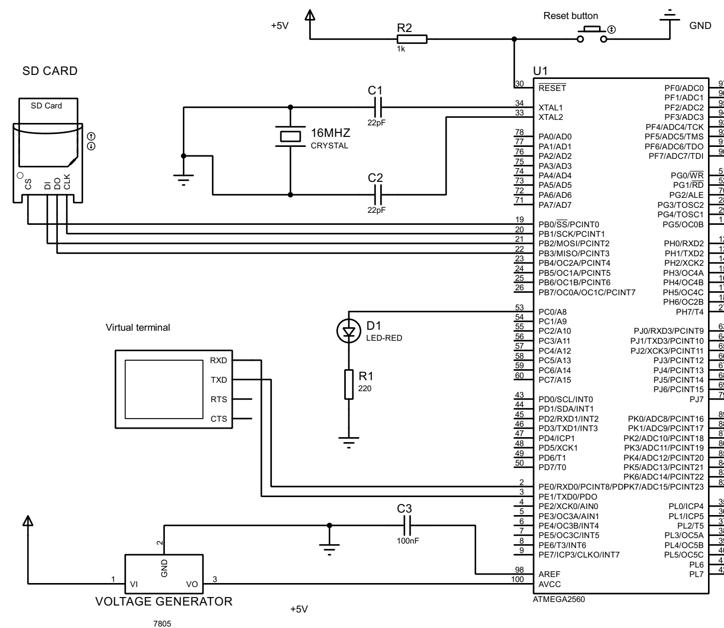

Fig. 2. Proteus simulation model of the Part Only RTOS hardware environment, including SD card, SPI interface, red-led indicator, reset button, voltage generator, 16 *Mhz* Crystal and UART monitoring virtual terminal.

System Resilience and Power Interruption Handling. A critical consideration for embedded systems is operational stability during power fluctuations. The "Part Only RTOS" architecture is specifically designed to be resilient against power failures during the flashing process. Because the Bootloader is physically isolated in the high-memory section $(0 \times 3E000)$ and does not self-overwrite, it serves as a permanent recovery environment. If an interruption occurs while a module is being written to Flash, the system will not "brick"; instead, upon power restoration, the hardware is programmed to initialise the Bootloader. The Bootloader then detects the corrupted state through signature mismatch and allows the user to re-authorise the loading sequence from the SD card. This ensures that the core logic remains secure and the device remains maintainable even in unstable industrial environments.

Main Result. To evaluate the performance and efficiency of the Part Only RTOS, a series of experiments was conducted using the hardware configuration. The primary metrics focused on memory footprint, module loading latency, and system stability during task execution.

Memory Utilisation Analysis. One of the most significant achievements of this kernel is its minimal impact on the ATmega2560's limited resources. The kernel occupies approximately 2560 bytes of Flash, which is roughly 1% of the total available space. The SRAM usage was analysed using static analysis of the ELF sections and runtime monitoring. As shown in Fig. 3, more than 94% of the SRAM remains available for modular applications and dynamic heap allocation.

```
PuTTY (inactive)
Bootloader: Starting
Bootloader: Using default LAUNCHER.BIN
Bootloader: Starting application

=== Part only real time OS Launcher v1.0 ===
Binary files found:
1) LAUNCHER.BIN  size: 5302  bytes
2) INT.BIN  size: 1348  bytes

Enter application number to run.
2
Loading: INT.BIN
Loading application to Flash...
Bootloader: Starting
Bootloader: Loading INT.BIN
Bootloader: Starting application
Kernel App Started
Type 'quite' for exit
Enter command:
[KERNEL] System is running fine...
```

Fig. 3. Operational Workflow of Part Only RTOS.

As shown in Tab. 1, the kernel footprint is optimised to leave more than 94% of the SRAM (7761 bytes) available for modular applications and dynamic heap allocation.

Table 1

Kernel Memory Footprint on ATmega2560

Memory Type	Segment	Size, Bytes	% of Total	Description
Flash	Kernel Code	2560	∼1.0	Core logic & drivers
SRAM	.data	350	∼4.27	Initialized variables
SRAM	.bss	79	∼0.96	Uninitialized variables
SRAM	.noinit	2	∼0.02	Stack Canary
Total SRAM	Used by Kernel	431	∼5.26	7761 bytes free

Loading Performance and Latency. The dynamic loading speed is a critical factor for the system's responsiveness. We measured the time taken from the moment the user confirms the selection in the Launcher to the moment the application starts execution. The total loading time for a standard 10 *KB* module was recorded as 132 *ms* (± 4 *ms*). This duration includes:

- SD card sector seeking and reading.
- Pattern matching and validation.
- Flash page erasing and writing (SPM cycles).

Experimental Results. The experimental results for the module loading process are summarised in Tab. 2. As demonstrated, there is a linear correlation between the binary size and the loading duration. For instance, a 10 *KB* module is loaded and initialised in 132 *ms*, which characterises the system's efficiency in handling dynamic modularisation on 8-bit hardware.To evaluate the performance of the proposed system, loading latency was measured using the internal 16-bit Timer1, providing a resolution of 1 μs. The loading process encompasses SD card initialisation, file system overhead via Petit FatFS, and the sequential flashing of 256-byte pages. Our benchmarks characterise a linear relationship between module size and loading time. Using a Class 10 SD card, the primary bottleneck was identified as the Flash memory write cycle (approximately 4.5 *ms* per page) rather than the SPI transfer speed. At an SPI clock frequency of 4 *MHz*, the system achieves a consistent throughput, allowing developers to precisely estimate the boot-up time based on the binary footprint of their specific application modules.

Table 2

Module Loading Latency on ATmega2560 (at 4MHz SPI)

Module Size, *KB*	Flash Pages	Loading Time, *ms*	Transfer Rate, KB/s
1 *KB*	4	13.2	~75.7
2 *KB*	8	26.4	~75.7
4 *KB*	16	52.8	~75.7
8 *KB*	32	105.6	~75.7
10 *KB*	40	132.0	~75.7
16 *KB*	64	211.2	~75.7
32 *KB*	128	422.4	~75.7

Reliability and Error Handling. During the testing phase, the system was subjected to "corrupted module" tests. When a .bin file without the 27-byte signature was provided, the Launcher successfully aborted the process, preventing system failure. Additionally, the Stack Canary (the 2-byte .noinit segment) was monitored; it successfully triggered a system reset when a simulated recursive function attempted to overflow the stack, ensuring the integrity of the kernel.

Software and Hardware Environment. To ensure the reproducibility of the reported results, the following technical environment was used: C Programming Language (standard C), avr-gcc Compiler (version 7.3.0), Microchip ATmega2560 (16 *MHz* external crystal oscillator), SD/SDHC card (FAT16 formatted) using SPI, Proteus VSM version 8.17, and PuTTY terminal emulator.

Comparative Analysis. The "Part Only RTOS" offers a unique trade-off between the flexibility of dynamic loading and the minimal resource footprint of 8-bit architectures. Compared to traditional Arduino bootloaders (e.g., Optiboot), which only allow full firmware updates, our system recognises and loads independent functional modules without affecting the core kernel. Although sophisticated real-time kernels like FreeRTOS provide comprehensive task management, they lack native support for SD-based dynamic module loading on low-power MCUs such as the ATmega2560.

The proposed system fills this gap by prioritising modularity and recovery over complex task pre-emption, making it more suitable for field-maintainable industrial controllers, where a full re-programming is impractical. The functional comparison is provided in Tab. 3.

Table 3

Comparison of features between Part Only RTOS and existing solutions

Feature	Arduino Bootloader	FreeRTOS	Part Only RTOS
Dynamic Module Loading	no	no	yes (via SD)
Minimum RAM Footprint	very Low	6–10 *KB*	$<$ 1 *KB* (Kernel)
8-bit Optimization	yes	limited	high (specialised)
Update Mechanism	full reprogramming	compilation-based	modular execution

Conclusion. In this paper, we present the development of the Part Only RTOS, which proves that modular, dynamic application management is viable even on highly resource-constrained 8-bit platforms like the ATmega2560. By shifting from a monolithic firmware model to a three-tier architecture comprising the Launcher, Bootloader, and Kernel, we have achieved a system that allows for field-updates and multi-functional task execution without the need for external programming tools.

The experimental validation confirmed that the system maintains a minimal memory footprint, occupying only 5.2% of SRAM and 1% of Flash, while delivering high-speed module loading of approximately 132 *ms* for a 10 *KB* binary. Furthermore, the integration of Kernel Pattern Matching and Stack Canary mechanisms provides a robust safety layer against incompatible code and memory overflows, ensuring system stability in dynamic environments.

Received 11.02.2026
Reviewed 12.03.2026
Accepted 26.03.2026

REFERENCES

1. Williams E. *Make: AVR Programming* (1st Ed.). O'Reilly Media (2014).
2. Bondarenko D.N. *Vstraivaemye Mikrokontrolery AVR.* Elec.ru (2018) (in Russian).
3. Beningo J. *Bootloader Design for Microcontrollers in Embedded Systems*. Beningo Embedded Group (2015).
   https://www.beningo.com/
4. Li Q., Yao C. *Real-Time Concepts for Embedded Systems*. CMP Books (2003).
   https://doi.org/10.1201/9781420025552
5. Atienza D. *Dynamic Memory Management for Embedded Systems*. Springer (2015).
   https://doi.org/10.1007/978-3-319-10571-0

6. Zlatanov N. *Dynamic Memory Allocation and Fragmentation*. ESC Santa Clara (2015).
7. White E. *Making Embedded Systems*. O'Reilly Media (2011). https://doi.org/10.1002/9781119457503
8. Ehrlich P., Radke S. Energy-aware Software Development for Embedded Systems in HW/SW Co-design. *Workshop on SEES* (2013). https://doi.org/10.1109/DDECS.2013.6549823
9. Cha N. *Petit FatFs Module Specification*. [Online]. Available: http://elm-chan.org/fsw/ff/00index_p.html

Մ. Վ. ԱՎԵՏԻՍՅԱՆ, Ք. Ա. ԱՎԵՏԻՍՅԱՆ

ՄՈԴՈՒԼԱՅԻՆ ՀԱՎԵԼՎԱԾՆԵՐԻ ԴԻՆԱՄԻԿ ԲԵՌՆՄԱՆ ԵՎ ԿԱՌԱՎԱՐՄԱՆ ՕՊԵՐԱՑԻՈՆ ՄԻՋԱՎԱՅՐ ՆԵՐԿԱՌՈՒՑՎԱԾ ՀԱՄԱԿԱՐԳԵՐԻ ՀԱՄԱՐ

Ներկառուցված համակարգերի նախագծման մեջ ռեսուրսների սահմանափակության պայմաններում մոդուլյարության ապահովումը հանդիսանում է առանցքային խնդիր: Հոդվածում ներկայացվում է "Part Only RTOS" օպերացիոն միջավայրը, որը մշակվել է 8-բիթանոց ATmega2560 միկրոկոնտրոլլերների հիմքով սարքերի համար: Առաջարկվող համակարգի հիմնական առանձնահատկությունը SD կրիչից կիրառական մոդուլների (.bin ֆորմատով) դինամիկ բեռնման և գործարկման հնարավորությունն է՝ առանց սարքավորման ամբողջական վերածրագրավորման: Մշակված եռաստիճան ճարտարապետությունը (Launcher, Bootloader, Kernel) ապահովում է հուսալի վալիդացիա եզակի ձևանմուշների միջոցով և հիշողության արդյունավետ օգտագործում: Փորձարարական հետազոտությունները ցույց են տվել, որ համակարգը զբաղեցնում է ընդամենը 2.5 *KB* հիշողություն, իսկ 10 *KB* ծավալով մոդուլի բեռնման ժամանակը կազմում է 132 *ms*:

М. В. АВЕТИСЯН, К. А. АВЕТИСЯН

ОПЕРАЦИОННАЯ СРЕДА ДЛЯ ДИНАМИЧЕСКОЙ ЗАГРУЗКИ И УПРАВЛЕНИЯ МОДУЛЬНЫМИ ПРИЛОЖЕНИЯМИ ВО ВСТРАИВАЕМЫХ СИСТЕМАХ

Обеспечение модульности в условиях ограниченности ресурсов является ключевой задачей при проектировании встраиваемых систем. В статье представлена операционная среда "Part Only RTOS", разработанная для устройств на базе 8-битных микроконтроллеров ATmega2560. Основной

особенностью предлагаемой системы является возможность динамической загрузки и запуска прикладных модулей (в формате .bin) с SD-носителя без необходимости полной перепрошивки оборудования. Разработанная трехуровневая архитектура (Launcher, Bootloader, Kernel) обеспечивает надежную валидацию программного обеспечения с помощью уникальных сигнатур (паттернов) и эффективное распределение памяти. Экспериментальные исследования показали, что ядро системы занимает всего 2,5 *КБ* Flash-памяти, а время загрузки модуля объемом 10 *КБ* составляет 132 *мс*. Данное решение позволяет создавать гибкие и высокопроизводительные модульные встраиваемые системы на базе маломощных микроконтроллеров.