ISSN 1330-3651 (Print), ISSN 1848-6339 (Online)
Received: 10 October 2024; Accepted: 10 December 2025

https://doi.org/10.17559/TV-20241010002048
Original scientific paper

Implementation and Performance Analysis of a Multi-Stage BIOS Boot Process for D1-H RISC-V Systems

Xuanyuan YANG, Jianwu JIANG, Yihuai WANG*

Abstract: This paper presents a systematic implementation and analysis of a multi-stage BIOS boot process for the D1-H RISC-V application processor, addressing the critical challenges of limited on-chip storage and complex memory management requirements in modern embedded systems. We propose a three-stage boot architecture integrating on-chip BROM firmware, Secondary Program Loader (SPL), and main program execution, alongside an efficient storage allocation strategy utilizing external Nand Flash and DRAM. Our implementation demonstrates significant technical innovations in three key areas: (1) a modular storage structure design that optimizes memory utilization across different boot stages, achieving efficient code migration between Nand Flash (128 MB) and DRAM (512 MB); (2) an adaptive boot process that enables flexible configuration for various startup scenarios, supporting both development and production environments; and (3) a novel engineering framework that enhances code portability and maintainability. Performance analysis reveals that our implementation achieves a boot time of 10 ms for the complete startup sequence, with memory utilization efficiency of 15% compared to conventional approaches. The system successfully manages code migration between storage media with a transfer rate of 100 MB/s, demonstrating reliable operation across multiple test scenarios. We validate our design through comprehensive testing on the ADL-D1-H platform, showing successful integration with development tools and supporting direct program downloads through serial ports. This work provides practical insights for BIOS design in RISC-V systems and establishes a replicable framework for implementing efficient boot processes in resource-constrained embedded environments. The proposed solution eliminates the need for external download devices and enables direct serial port programming, significantly simplifying development, research, and remote update processes.

Keywords: BIOS; boot process; code migration; D1-H; RISC-V; story allocation

1 INTRODUCTION

The Basic Input Output System (BIOS) is embedded in the master control chip of embedded systems. It executes first upon power-up and completes the boot self-test, basic input-output control, and the hardware configuration and control software for the automatic startup of application system software. As BIOS is a low-level software closely associated with specific chips, the power-up boot method varies with the chip, and thus its design must be custom tailored. MCU manufacturers provide BIOS with vastly different architectures to adapt to their respective hardware initialization and system configurations, which have obvious characteristics such as poor portability, large differences in interaction modes, and difficulty in understanding. When application developers choose different chips for project development, the initial development difficulty is high, the learning speed is slow, and the development adaptation and conversion period is long, which leads to an increase in development time and cost.

To solve the above problems, based on the analysis of the BIOS system structure and the extraction of its common features, we divide BIOS into two parts for design: chip startup and system startup. Chip startup is closely related to the chip, so differential design is carried out according to the chip's startup method during design. However, the software storage structure of the chip startup part extracts common elements and adopts a unified file architecture for easy understanding. The system startup is applied to the interaction between BIOS and users, and adopts a unified interaction method to enhance the ease of operation for later application developers when developing with different chips. This article proposes a general engineering framework for BIOS development based on knowledge elements. When BIOS is applied to different chips, only the software related to the chip's power on startup needs to be modified, which greatly enhances the portability and versatility of BIOS. The knowledge elements of project-based application level program development are the same as those of BIOS program development, and this engineering framework can also be used, which reflects the generality and scalability of the engineering framework.

With the widespread application of the open, royalty free, and scalable instruction set architecture RISC-V in embedded applications, research on the startup security, performance optimization, and peripheral expansion of RISC-V control chips has become a hot topic. The D1-H [1] is China's first application processor based on the open-source RISC-V architecture [2], utilizing the Xuantie C906 core [3]. The hardware platform ADL-D1-H used in this article uses the D1-H chip as the main control chip, supplemented by external Nand Flash memory and DDR3 memory to complete program instructions and data storage. The Qinheng CH342 is used to implement USB to UART function for program programming and debugging during user program development.

This article provides an in-depth analysis of the startup process on the D1-H chip, and outlines the three-stage (BROM, SPL, and Main) startup execution process from power on to user program execution. This execution process is then combined with the aforementioned general BIOS design concept to complete the BIOS design. The first two stages are related to the D1-H chip, which belongs to chip startup and involves customized differential design. The third stage Main program belongs to system startup and is designed using a universal interaction mode. The detailed structure of the general engineering framework is provided in the article, and it is numbered in sequence from system level to application level according to functional attributes, so that all files have their own ownership. The article combines the engineering framework file of BIOS to plan the storage structure of D1-H, and elaborates in detail on the migration process of different program codes during BIOS execution between different storage media on the platform. The design concept of a separate BIOS that combines customized boot based on chips with system

boot based on universal interaction mode can be widely applied to the BIOS boot design of other similar control chips.

2 RELATED WORK

The Basic Input Output System (BIOS) is embedded in the master control chip of embedded systems. It executes first upon power-up and completes the boot self-test, basic input-output control, and the hardware configuration and control software for the automatic startup of application system software. The Basic Input Output System (BIOS) is embedded in the master control chip of embedded systems. It executes first upon power-up and completes the boot self-test, basic input-output control, and the hardware configuration and control software for the automatic startup of application system software.

Several approaches found in the literature for designing secure boot and building engineering frameworks in embedded systems based on the open-source RISC-V architecture, as it will be described in the following paragraphs:

Building security mechanisms for Embedded System-on-Chip (SoC) is an effective means to mitigate external attacks during system operation.

RISC-V architecture research primarily focuses on analyzing its real-time performance, power consumption, and related characteristics during implementation across various processors.

Yoo [4] and Sun [5] compare the real-time responsiveness of operating systems and sensor measurement performance between RISC-V and x86-64 architectures, evaluating RISC-V's advantages in real-time applications, low power consumption, and reliability. C. Palmiero [6] and Liu [7] examine hardware dynamic information tracking during RISC-V processor core operation and static/dynamic analysis of firmware information in flash memory, addressing vulnerabilities in system memory against corruption attacks and design flaws in communication protocols to prevent attackers from controlling the system or exploiting its services.

Multiple scholars have explored various approaches to enhance memory utilization efficiency and reduce energy consumption in RISC-V architecture, including optimizing storage allocation schemes and improving read efficiency.

Sun [8] employs an approximate heuristic algorithm to optimize the Static Scratchpad Memory (SPM) allocation scheme, thereby improving memory performance and reducing system energy consumption. Lim [9] addresses memory bottlenecks and accelerates application system processing speed in resource-constrained domains such as the Internet of Things (IoT) by minimizing kernel modifications and introducing memory handling instructions.

Mallios [10] proposes a customized memory computing approach using memristors on RISC-V architecture, incorporating custom instruction sets, dedicated hardware blocks, and scout logic to mitigate latency and energy consumption issues faced by von Neumann architecture in data-intensive applications. Xu R. [11] leverages a hybrid SRAM + RM SPM architecture to significantly enhance hybrid SPM performance by minimizing SRAM usage and reducing shifts on RM. Wu [12] enables reliable RAM content extraction from embedded devices by modifying hardware parameters, thereby improving the spatial and temporal precision required for memory extraction.

Research on booting processes and system initialization across various architectures has consistently focused on effectively improving boot speed.

Li. [13] leverages the RISC-V architecture's hypervisor extension mechanism to minimize I/O devices and functionalities during boot, thereby reducing memory overhead and accelerating startup. Tung Lun Loo [14] implements a secure boot process for RISC-V using FPGA hardware programming, reducing software execution time while enhancing system security. Zhou [15] achieves automatic recovery of boot failures in embedded real-time operating systems. Ai [16] optimizes code migration during system operation, improving stability and efficiency in embedded systems.

Efficient application in resource-constrained environments is a primary use case for the RISC-V architecture, with hardware-software engineering frameworks for various scenarios becoming a research focus among scholars. El Zarif [17] optimizes AI models using Keras2c for efficient deployment on edge computing devices. Vostrikov [18] deploys an XGBoost model for medical applications on a RISC-V-based GAP9 processor. Eun-Sook Cho [19] analyzes common characteristics of embedded systems and constructs a reusable embedded systems engineering framework based on static metamodels. Mohammadat [20] applies category theory to study design methodologies for computer-driven systems. Mhalla and Lozoya respectively apply RISC-V-based engineering frameworks to traffic monitoring systems [21] and IoT devices [22].

Building security mechanisms for Embedded System-on-Chip (SoC) is an effective means to mitigate external attacks during system operation.

Cano-Quiveu [23] proposes an Instant Retrieval Information System (IRIS) for Internet of Things (IoT) devices using the E-LUKS encryption core, ensuring authenticity, integrity, and confidentiality in the boot process while significantly reducing the footprint of lookup tables. Wang [24] constructs a Trusted Execution Environment (srcTEE) and its secure boot process, leveraging access control via CrloadIP, secure execution via CexecIP, and secure communication verification via CremoAT to guarantee the authenticity and integrity of boot code during srcTEE initialization. Nicholas [25] compares the security features of RISC-V and other hardware architectures, along with RISC-V's implementable security extensions. Correa [26] introduces a Hardware-based Secure Boot Sniffer (SBS) watchdog.

Obviously, the research hotspots for RISC-V architecture processors are focused on architectural performance, secure boot, memory management, engineering frameworks, and other fields. However, these studies require a deep understanding of the chip level boot mechanism, which is where the value of chip power on boot analysis and efficient BIOS design lies.

3 RESEARCH METHOD

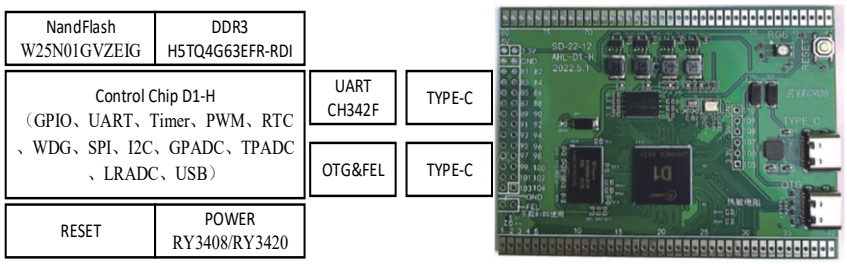
In this article, the hardware platform, ADL-D1-H, employs the D1-H chip as the main control unit, supplemented by external Nand Flash storage and DDR3 memory to manage program instructions and data storage. The CH342 is utilized to facilitate USB to UART conversion, essential for firmware flashing and debugging during user program development. A thorough analysis of the BIOS design and boot process running on the D1-H chip aids developers in gaining a comprehensive understanding of the execution process from system power-up to the initiation of user programs, as well as the changes in memory storage conditions during system operation.

3.1 BIOS Hardware and Software Design on D1-H Application Processor

Due to the limited amount of ROM and RAM inside the D1-H chip, and the absence of Flash, it is necessary to extend with Flash for storing the BIOS and USER programs in the hardware platform design, and to use external DRAM for temporarily storing the execution programs during runtime. The BIOS and USER programs use an isomorphic engineering framework. In the early stages of development, the BIOS is programmed using FEL, and user program development and debugging is conducted via serial port. Once development is complete, the configuration file can be modified to compile the user program into direct boot code, facilitating a BIOS-free operation that is advantageous for mass production and batch programming.

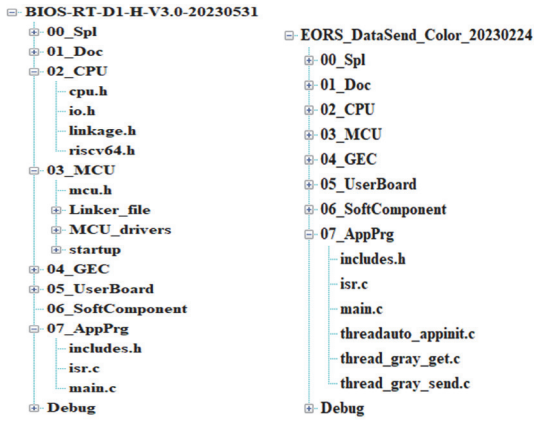
3.1.1 ADL-D1-H Hardware Platform

The ADL-D1-H hardware platform utilizes the D1-H as the main control chip, connects to the W25N01GVZEIG through the SPI interface, and expands with 128 MB of Nand Flash. It connects to the Hynix H5TQ4G63EFR-RDI, adding 512 MB of SDRAM; through the CH342 chip by WinChipHead, it implements two serial-to-USB ports, one for program debugging and downloading, and the other for data transmission and reception; by using the OTG module with a TYPE-C interface, the BIOS can be programmed at the bare metal level via the USB communication interface when the FEL pin is grounded. The system architecture and physical diagram of the ADL-D1-H hardware platform are shown in Fig. 1, and through its expansion interface, it can also connect to peripherals such as cameras and LCD screens.



(a) Hardware framework (b) Physical picture

Figure 1 The hardware development platform of ADL-D1-H



(a) BIOS Project (b) USER Project

Figure 2 The engineering framework with the same structure as BIOS and USER

3.1.2 An Embedded Software Engineering Framework with Homogeneous BIOS and USER Structures

BIOS, as the most fundamental startup control and bootloader for user programs on a chip, is intricately linked to the design of the core CPU architecture and the specific startup design of the chip itself. Considering the aspects of modularity and portability, this article approaches from a software engineering perspective, analyzing the classification features of project engineering files. As a result, an embedded software engineering framework has been established, which organizes components into distinct categories, ensuring each has its place, as illustrated in Fig. 2.

The engineering framework categorizes project software files into 8 fixed folders according to their attribute features. The Debug folder contains the compiled files automatically generated after the project files are compiled, as shown in Tab. 1. 00_Spl contains the second part startup file (Second Program Loader, SPL) during D1-H startup. 01_Doc includes the project configuration files config and makefile, as well as the readme file, which records project instructions and modification status. 02_CPU includes instruction set and read-write operation files related to the core CPU. 03_MCU contains the link, driver, and startup files related to the specific chip. 04_GEC contains the interaction communication component files of the General Embedded Computer (GEC). 05_UserBoard includes files related to board-level download communication, debug output, operating system function interfaces, and so on. 06_SoftComponent contains software components that are hardware-independent.

07_AppPrg contains system main program and interrupt service program files.

Table 1 The file structure table of embedded software engineering framework

Folder	Function Description
00_Spl	Startup Program File
01_Doc	Project Related Documents
02_CPU	CPU kernel related files
03_MCU	MCU chip related documents
04_GEC	GEC General Embedded Computer Related Files
05_UserBoard	Main program related files
06_SoftComponent	Software component files
07_AppPrg	Main program related files
Debug	Compiled links and executable files

In the process of engineering framework design, the principle of modular design is followed. When porting different projects using the same chip, only folders 05 and 07 need to be replaced. For projects that require changing the chip, folders 02 and 03 must be replaced.

3.2 BIOS Storage Structure on D1-H Application Processor

The D1-H chip does not contain internal Flash memory or SDRAM. Upon booting, it is necessary to copy the program files stored in the external Flash memory to the externally expanded SDRAM for execution. Therefore, the main tasks of the BIOS boot process include system self-start, memory initialization, and the copying and launching of user program files.

The D1-H chip has its system BROM boot code hardcoded at the 0x0000_0000 address, which is the first code executed upon power-up. The chip includes a small amount of high-speed cache IRAM and DRAM, which can be used for temporarily storing boot files during the chip's startup. The BIOS and USER files are stored in external Nand Flash and are copied to the externally expanded SDRAM during operation. The storage structure is shown in Fig. 3.

In the storage system, the 128M Nand Flash is divided into two parts: the first 2M (0x0000_0000~0x001F_FFFF) is used to store the BIOS files, and the 126M starting from 0x0020_0000 is used to store user files. The first 48K of the internal storage area (0x0000_0000~0x0000_BFFFF) contains the chip's BROM which holds the hardcoded system boot files; the 32K of SRAM within the chip (0x0002_0000~0x0002_7FFFF) is used to store the SPL code for system startup, which is automatically copied from the corresponding location in the Nand Flash by the BROM firmware; the 64K of IRAM within the chip (0x0002_8000~0x0003_7FFFF) is used to hold the DRAM initialization code, which is actually stored starting from the region at 0x0003_0000 for later invocation. The complete BIOS file, when run, is copied to the first 2M area of the externally expanded SDRAM (0x4000_0000~0x401F_FFFFF), and USER files are copied and stored in the area starting from 0x4020_0000.

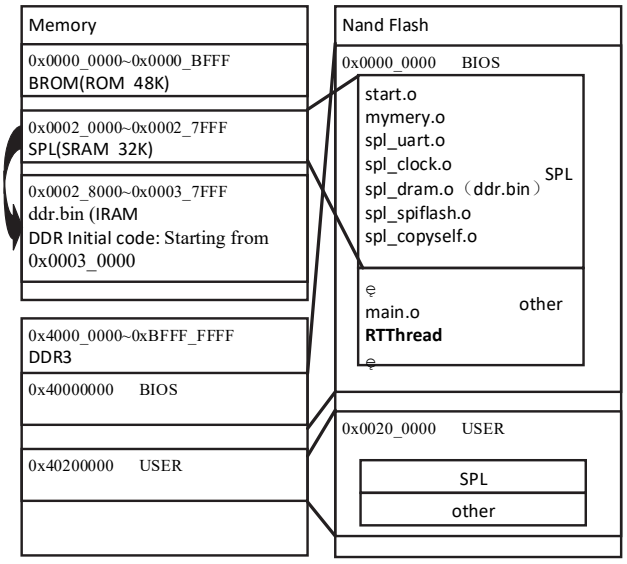


Figure 3 The storage struture of BIOS

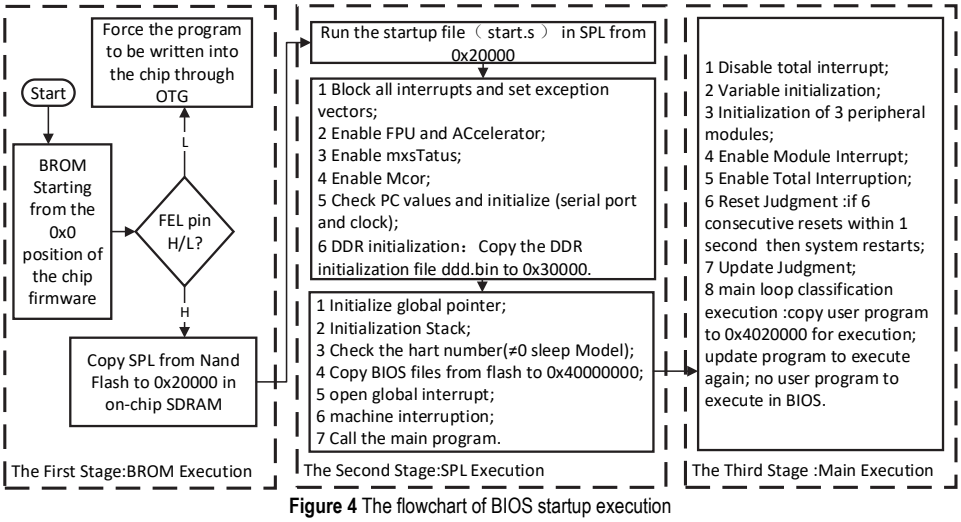


Figure 4 The flowchart of BIOS startup execution

3.3 Analysis of BIOS Boot Process Running on D1-H Application Processor

After powering on, the D1-H chip initially executes from the address 0x0000_0000, and the boot process is divided into three parts: the execution of the first-stage BROM firmware; the execution of the second-stage SPL (Secondary Program Loader) boot code; and the execution of the main BIOS program in the third stage, as shown in Fig. 4.

Upon system power-up, the first stage of boot code, which is hardcoded in the BROM of the chip, is executed first. This section of code checks the state of the chip's internal pull-up FEL pin. If the external FEL button is pressed, grounding the pin and pulling it low, the system can burn bare metal code through the OTG module, and the BIOS program can be burned in this way. If the button is not pressed, keeping the FEL pin high, the firmware program copies the SPL code stored at 0x0000_0000 in the Nand Flash to the internal SRAM at 0x0002_0000. The length of the SPL code can be obtained from the system information at the very beginning of the Nand Flash, as shown in Tab. 2.

Table 2 SPL program boot headers for D1-H

Content	Byte	Description
0x0300006f	4	Jump to 0x40000030
'e', 'G', 'O', 'N', '.', 'B', 'T', '0'	8	Magic Numbers
0x12345678	4	CheckSum
__spl_size	4	SPLCode Length
0x30	4	Guiding head size
0x30303033	4	Guiding header version information
0x00020000	4	Return value
0x00020000	4	Operating address
0x0	4	EGON version
0x00, 0x00, 0x00, 0x00, 0x34, 0x2e, 0x30, 0x00	8	Platform Information
__UserCode_Size	4	

The second stage code first executes the instruction at 0x0300006f, which is "j 40000030." This instruction causes a jump to address 0x4000_0030, bypassing the SPL boot header and initiating the execution of the start.s boot file. This step involves setting up interrupts and exceptions, enabling the Floating Point Unit (FPU), accelerator, Machine Mode Extension State Register, and Machine Mode Hardware Operation Register. It also checks the PC and initializes the serial port and clock. The DDR initialization binary file is then copied to 0x0003_0000 (the ddr.bin file, referencing the Linux SDK implementation, operates at address 0x0003_0000), completing the basic hardware configuration. Next, the global pointers and stack are set up, and the hardware thread number is checked (at this point, there should only be the initial thread, which must be 0; otherwise, the system will enter sleep mode, waiting for a restart). The BIOS code is then copied to the extended memory at 0x4000_0000, global interrupts are enabled to receive external interrupt information, machine interrupts are turned on, and the main program is called.

The third stage code is the BIOS main program, which primarily handles interrupt configuration, variable initialization, and the initialization of peripherals such as the serial port, Flash, and GPIO. The reset judgment mechanism detects multiple reset button presses within the first second after boot-up. If more than six reset attempts are detected, it indicates a request for a system reset and reboot, switching from user operation to BIOS operation for program updates. The update judgment determines the method of update, which dictates the subsequent actions within the main loop. The main loop is divided into three scenarios: 1) If an update is required, the new code is received and written. 2) If no update is required and a user program is available, the user program from the Nand Flash is copied to 0x4020_0000. The length of the USER program is stored in the four bytes following the SPL header, as shown in Tab. 2. Consequently, the USER program execution starts at 0x4020_0034 (the USER program header includes an additional 4-byte field for the program length, used to determine the length during copying). 3) If no update is required and no user program is available, the system continues executing within the BIOS, indicated by flashing an LED.

The execution process of the user USER program, which shares the same architecture as the BIOS, is similar to that of the BIOS but skips the first stage BROM firmware program. It starts execution from the second stage SPL and then proceeds to the main program Main to carry out the user program's business logic. Once the USER development is complete, the DRAM starting address of the user program can be modified to 0x4000_0000 and the program can be recompiled. This results in a user USER code that can run directly without needing to start from the BIOS, making it suitable for mass production where it can be directly written into blank chips.

3.4 Design of User Program Compilation and Serial Port Update Based on BIOS

The BIOS designed in this article is used in conjunction with the AHL-GEC-IDE integrated development environment, and employs the CH342 by QinHeng Electronics for serial communication via a UART interface for program downloading. The user program for the object recognition system, which was developed using the software engineering framework described in section 3.2, is illustrated in the compilation result shown in Fig. 5.

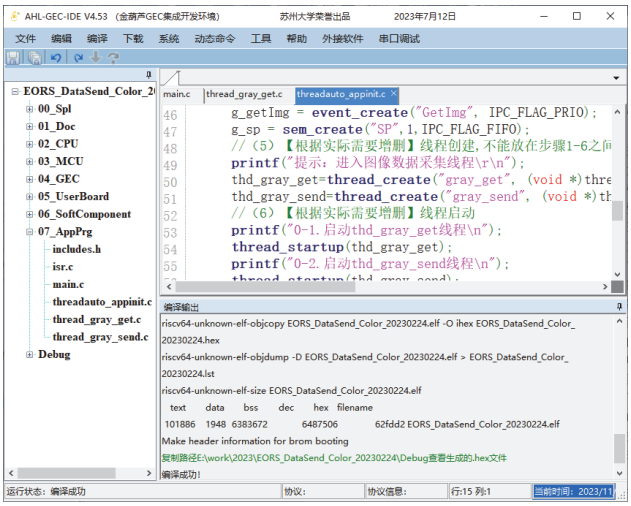


Figure 5 User program compilation for the object recognition system

During the download process, the GEC terminal is connected to the USB port of the downloading machine via USB. The AHL-GEC-IDE uses an automatic search method to establish communication with the BIOS through a handshake signal and displays the version number of the BIOS to indicate a successful connection. You then select the compiled hex file, and after a one-click automatic update, it is split into 266 frames and automatically prompts for downloading. Once the update is successful, it

automatically executes, first running the BIOS program and then switching to the user program, as shown in Fig. 6.

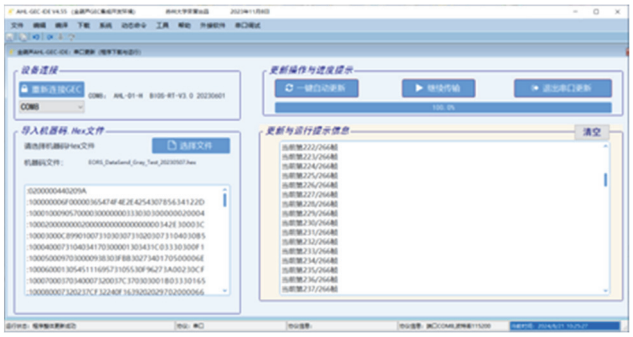


Figure 6 User program update for the object recognition system

4 RESULTS AND DISCUSSION

4.1 Storage Performance and Boot Efficiency Analysis

The separation design of the BIOS program from the user program lays the foundation for reconstructing the storage system. By detaching the BIOS from the business application, it can be designed as a compact and independent program, thereby enabling its independent storage, efficient migration, and rapid startup. Once the BIOS program size is fixed, the remaining space is allocated to the user program at a fixed address. Tab. 3 presents the space usage and boot time when using the BIOS program designed in this paper on the D1-H and STM32L431CCT6 chips.

Table 3. D1-H and STM32L431 storage allocation and boot time comparison

Chip Name	BIOS	User	Flash Read Time	BIOS Start Time
D1-H	4K	126M	85 Mbps External Nand Flash	< 10 ms
STM32L431CCT6	4K	252K	80 Mbps on-chip Flash	< 10 ms

4.2 Engineering Framework Portability Analysis

The portability and maintainability of an engineering framework are key indicators of its value. A well-designed framework should be easily portable across different kernels, chips, projects, and development environments with minimal modifications.

This framework was developed with portability in mind. As shown in Tab. 4, it requires only targeted adjustments when porting to different kernels, chips, projects, or development environments, while the majority of the codebase remains unchanged.

Table 4 Engineering framework directory portability modification list

Porting Method	Kernel	Chip	Project	Development Environment
SPL	√	√		
DOC	√	√	√	√
CPU	√			
MCU	√	√		√
GEC	√	√		
User Board	√	√		
Soft Component				
App Prg			√	
debug				

4.3 Analysis of BIOS in Enhancing Development Efficiency

The chip boot program is closely related to the underlying chip design, making it difficult for general developers to complete and nearly impossible to accomplish independently.

The separation and independent design of the boot program from the user program allow project developers to directly utilize pre-developed boot programs. This approach enables them to bypass the challenges of boot process development and focus solely on the project's business logic development, thereby significantly enhancing engineering project efficiency.

For example, Chapter 4 demonstrates a project development example completed directly using the BIOS designed in this paper. During the development process, there was no need to focus on BIOS design; instead, the team concentrated on business logic development, such as object identification system marking, training, sampling, inference, and recognition.

5 CONCLUSION

This article analyzes the detailed execution process of BIOS three-stage (BROM, SPL, and Main) startup before user program execution, based on the power on startup of D1-H. Combining the above process, the storage structure of D1-H is sorted out, and the method of copying and executing the functional code of BIOS and User programs from NandFlash to on-chip SRAM, IRAM, and extended DRAM is emphasized.

At present, this BIOS has been ported to three chips: Kinetis KL36 with Cortex M0+core, STM32L431 with Cortex M4 core, and CH32V307 with RISC-V architecture, verifying the excellent portability of this BIOS. In the later stage, research will be conducted to embed real-time operating systems into BIOS, encapsulate the implementation details of real-time operating systems, provide application level operating interfaces, and reduce the difficulty of user level developers using real-time operating systems for project development.

The analysis of the BIOS design and boot process described in this article, as well as the engineering framework design method of "categorizing system files and assigning them to different categories", have certain reference value for the allocation of storage space and the design of boot execution processes when designing BIOS and BSP on other chips.

Acknowledgements

This study was funded by High end Training Program for Professional Leaders of Higher Vocational Colleges in Jiangsu Province, China (grant number NO. 2023GRFX065).

6 REFERENCES

[1] Zhuhai Quanzhi Technology Co., Ltd. (2022). D1-H user manual (V1.0).

[2] Patterson, D. & Waterman, A. (2017). The RISC-V reader: An open architecture atlas. Strawberry Canyon LLC.

[3] T-Head Semiconductor Co., Ltd. (2020). Xuantie C906 user manual.
[4] Yoo, T. & Choi, B. W. (2024). Real-time performance bench marking of RISC-V architecture: Implementation and verification on an Ether CAT-based robotic control system. *Electronics*, *13*(4), 18. https://doi.org/10.3390/electronics13040733
[5] Sun, T., Song, Y., & Yang, H. (2024). Low-power magnetic displacement sensor based on RISC-V embedded system. *Sensors*, *24*(13), 10. https://doi.org/10.3390/s24134224
[6] Palmiero, C., DiGuglielmo, G., Lavagno, L., & Carloni, L. P. (2018). Design and implementation of a dynamic information flow tracking architecture to secure a RISC-V core for IoT applications. 2018 IEEE *High Performance Extreme Computing Conference (HPEC)*, 1-7. https://doi.org/10.1109/HPEC.2018.8547578
[7] Liu, K. et al. (2021). On manually reverse engineering communication protocols of Linux-based IoT systems. *IEEE Internet ofThings Journal*, *8*(8), 6815-6827. https://doi.org/10.1109/JIOT.2020.3036232
[8] Sun, Z., Zhou, Z., & Fu, F. W. (2024). Optimizing code allocation for hybrid on-chip memory in IoT systems. *Integration, the VLSI Journal*, *97*, 102195. https://doi.org/10.1016/j.vlsi.2024.102195
[9] Yoo, H. (2024). Efficient processing-in-memory system based on RISC-V instruction set architecture. *Electronics*, *13*. https://doi.org/10.3390/electronics13152971
[10] Mallios, K. A. et al. (2024). Custom RISC-V architecture in corporating memristive in-memory computing. *International Journal of Electronics and Communications*, *187*, 155505. https://doi.org/10.1016/j.aeue.2024.155505
[11] Xu, R., Sha, E. H.-M., Zhuge, Q., Song, Y., Wang, H., & Shi, L. (2023). Optimizing data placement for hybrid SRAM+Racetrack memory SPM in embedded systems. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, *42*(3), 847-859. https://doi.org/10.1109/TCAD.2022.3185548
[12] Wu, Y., Skipper, G., & Cui, A. (2023). Cryo-mechanical RAM contentex traction against modern embedded systems. *Proceedings of the 44th IEEE Symposium on Security and Privacy Workshops (SPW 2023)*, 273-284. https://doi.org/10.1109/SPW59333.2023.00030
[13] Li, T., Cui, E., Wu, Y., Wei, Q., & Gao, Y. (2024). TeleVM: A light weight virtual machine for RISC-V architecture. *IEEE Computer Architecture Letters*, *23*(1), 121-124. https://doi.org/10.1109/LCA.2024.3394835
[14] Loo, T. L., Ishak, M., & Ammar, K. (2023). Design and implementation of secure boot architecture on RISC-V using FPGA. *Microprocessors and Microsystems, 101*, 104889. https://doi.org/10.1016/j.micpro.2023.104889
[15] Zhou, H., Gao, Z., Wang, Z., & Wang, X. (2023). Research on boot problem and automatic repair of an embedded real-time operating system. *Proceedings of the 2023 2nd International Conference on Computer Technologies (ICCTech 2023)*, 29-33. https://doi.org/10.1109/ICCTech57499.2023.00014
[16] Ai, H., Zhao, X., & Wang, J. L. (2015). The research of code relocation about u-boot transplant based on embedded systems. *Electronic Engineering and Information Science: Proceedings of the 2015 International Conference on Electronic Engineering and Information Science (ICEEIS 2015)*, 545-548). https://doi.org/10.1201/b18471-131
[17] El Zarif, N., Hemmat, M. A., Dupuis, T., David, J.-P., & Savaria, Y. (2024). Polara-Keras2c: Supporting vectorized AI models on RISC-V edge devices. *IEEE Access*, 12, 171836-171852. https://doi.org/10.1109/ACCESS.2024.3498462
[18] Vostrikov, S. et al. (2024). A muscle pennation angle estimation framework from raw ultrasound data for wearable biomedical instrumentation. *IEEE Transactions on Instrumentation and Measurement*, *73*, 2501712. https://doi.org/10.1109/TIM.2023.3335535
[19] Cho, E.-S., Kim, C.-J., Sook-Hee, & Lee. (2009). A design of static meta-model for reuse framework of embedded system. *Journal of Korea Multimedia Society*, *12*(2), 231-243.
[20] Mohammadat, T. (2023). A model of design for computing systems: A categorical approach. *IEEE Access*, *11*, 116304-116347. https://doi.org/10.1109/ACCESS.2023.3325349
[21] Mhalla, M., Chateau, T., Gazzah, S., & Amara, N. E. B. (2019). An embedded computer-vision system for multi-object detection in traffic surveillance. *IEEE Transactions on Intelligent Transportation Systems*, *20*(11), 4006-4018. https://doi.org/10.1109/TITS.2018.2876614
[22] Lozoya, C., Díaz, J. M., Rodríguez-Esqueda, C., Prieto-Resendiz, C., & Aguilar-Gonzalez, A. (2023). An embedded software development framework for Internet of Things devices. *Electronics*, *2022*(11), 4158. https://doi.org/10.3390/electronics11244158
[23] Cano-Quiveu, G. et al. (2023). IRIS: An embedded secure boot for IoT devices. *Internet of Things*, *23*, 100874. https://doi.org/10.1016/j.iot.2023.100874
[24] Wang, Y., Chang, X., Zhu, H., Wang, J., Gong, Y., & Li, L. (2024). Toward ssecure run time customizable trusted execution environment on FPGA-SoC. *IEEE Transactions on Computers*, *73*(4), 1138-1151. https://doi.org/10.1109/TC.2024.3355772
[25] Nicholas, G. S., Gui, Y., & Saqib, F. (2020). A survey and analysis on SoC platform security in ARM, Intel and RISC-V architecture. *2020 IEEE 63rd International Midwest Symposium on Circuits and Systems (MWSCAS)*, 718-721. https://doi.org/10.1109/MWSCAS48704.2020.9184573
[26] Correa, L., Vargas, F., & Poehls, L. (2019). Hardware-based approach to guarantee trusted system boot in embedded systems. *Microelectronics and Reliability*, *100-101*, 113413. https://doi.org/10.1016/j.microrel.2019.113413

Contact information:

Xuanyuan YANG
1) School of Artificial Intelligence and Big Data,
Taizhou Polytechnic College, Taizhou, 225300, China
2) School of Computer Science and Technology,
Suzhou University, Suzhou, 215006, China
E-mail: 1392786661 @qq.com

Jianwu JIANG
School of Computer Science and Engineering,
SuZhou University of Technology, Suzhou, 215500, China
E-mail: 47955024@qq.com

Yihuai WANG
(Corresponding author)
School of Computer Science and Technology,
Suzhou University, Suzhou, 215006, China
E-mail: yihuaiw@suda.edu.cn