electronics

MDPI

Article

Real-Time Performance Benchmarking of RISC-V Architecture: Implementation and Verification on an EtherCAT-Based Robotic Control System

Taeho Yoo and Byoung Wook Choi *

Department of Electrical and Information Engineering, Seoul National University of Science and Technology, Seoul 01811, Republic of Korea; kjkj0581@seoultech.ac.kr

* Correspondence: bwchoi@seoultech.ac.kr; Tel.: +82-02-970-6412

Abstract: RISC-V offers a modular technical approach combined with an open, royalty-free instruction set architecture (ISA). However, despite its advantages as a fundamental building block for many embedded systems, the escalating complexity and functional demands of real-time applications have made adhering to response time deadlines challenging. For real-time applications of RISC-V, real-time performance analysis is required for various ISAs. In this paper, we analyze the real-time performance of RISC-V through two real-time approaches based on processor architectures. For real-time operating system (RTOS) applications, we adopted FreeRTOS and evaluated its performance on HiFive1 Rev B (RISC-V) and STM3240G-EVAL (ARM M). For real-time Linux, we utilized Linux with the Preempt-RT patch and tested its performance on VisionFive 2 (RISC-V), MIO5272 (x86-64), and Raspberry Pi 4 B (ARM A). Through these experiments, we examined the response times on the real-time mechanisms of each operating system. Additionally, in the Preempt-RT experiments, scheduling latencies were evaluated by means of the cyclictest. These are very important parameters for implementing real-time applications comprised of multi-tasking. Finally, in order to show the real-time capabilities of RISC-V practically, we implemented motion control of a six-axis collaborative robot, which was performed on the VisionFive 2. This implementation provided a comparative result of RISC-V's performance against the x86-64 architecture. Ultimately, the results indicated that the real-time performance of RISC-V for real-time applications was feasible. A noticeable achievement of this research is its first implementation of an EtherCAT master on RISC-V designed for real-time applications. The successful implementation of the EtherCAT master on RISC-V shows real-time capabilities for a wide range of real-time applications.

Keywords: RISC-V; FreeRTOS; Preempt-RT; RTOS; EtherCAT; embedded Linux

Citation: Yoo, T.; Choi, B.W. Real-Time Performance Benchmarking of RISC-V Architecture: Implementation and Verification on an EtherCAT-Based Robotic Control System. *Electronics* **2024**, *13*, 733. https://doi.org/10.3390/electronics13040733

Academic Editor: Luca Patanè

Received: 23 January 2024
Revised: 9 February 2024
Accepted: 9 February 2024
Published: 11 February 2024

1. Introduction

The surge in advanced intelligent and collaborative robots, along with automated control systems, has accelerated research in real-time control [1,2]. Embedded systems for real-time control must possess the capability to execute tasks within desired time frames using limited resources [3]. Based on their response to missed deadlines, these systems are classified into hard, soft, and firm real-time systems. Failure to meet deadlines in hard real-time tasks can have severe consequences, such as system failure or even industrial disasters [4]. In contrast, missing deadlines in soft and firm real-time tasks typically does not lead to system destruction or severe consequences. If a soft real-time task exceeds its deadline, it may result in a lower quality of service to users or the continuation of the task, whereas a firm real-time task may render the results useless or cause them to be ignored by the system [5,6]. Therefore, real-time applications for real-time control must be implemented considering these time-critical constraints. As the demands for these real-time applications grow, there is a corresponding increase in the complexity and expense of implementing and validating their functionalities. However, costs can be mitigated with

adequate technical support and detailed information about the instruction set architecture (ISA) for these systems.

RISC-V is an ISA designed to be simple and modular. Developed as open-source and royalty-free, it allows anyone to use and extend its capabilities freely [7]. This makes it a popular option for creating embedded systems in real-time environments, especially when budget constraints are a major consideration, due to its advantage of low-cost deployment [8,9]. Additionally, RISC-V can be modified to meet various constraints and requirements easily, allowing for the implementation of desired functionalities [10–12]. Leveraging these benefits, there is research into the use of RISC-V across various domains, including artificial intelligence (AI) and the Internet of Things (IoT) [13,14]. However, unlike ARM and x86-64, which benefit from advanced commercialization and a wealth of information, RISC-V, as a relatively new player in the market, encounters challenges due to limited technical support and information pertaining to the development of real-time applications [15].

With the escalating complexity and functional demands of real-time applications, meeting required response times has become increasingly challenging. Consequently, there is an increased need for development information to implement RISC-V based real-time systems that achieve these response times. In real-time applications comprised of multiple tasks, extensive data transfer occurs between tasks, making the performance of inter-task communication (ITC) crucial to overall real-time performance. ITC mechanisms facilitate data transfers between tasks, and each kernel can implement these mechanisms with slight variations in functionality and naming. The commonly used ITC mechanisms in real-time kernels include queues, semaphores, and mutexes. A queue is a mechanism used for message transfers between tasks, operating on a first-in, first-out (FIFO) basis. Semaphores are used to manage access to task resources. Operating systems typically categorize them into binary semaphores, with a count limit of one, and counting semaphores, with a higher count limit. Unlike mutexes, semaphores do not prioritize tasks, making them suitable for synchronizing operations between tasks. Finally, a mutex limits access to a resource to one task at a time, ensuring that the task holding the mutex can operate without interference from others [16–18].

In this paper, we compare and analyze the real-time performance capabilities of the inter-task communication mechanism on embedded boards based on RISC-V and other ISAs. For complex, multi-functional tasks in real-time applications, it is common to divide them into smaller tasks based on their specific roles. These smaller tasks rely on ITC to maintain data integrity and to ensure timely communication [19]. Specifically, ITC is used for facilitating communication among various tasks in applications that collect signals from sensors or other hardware, analyze and process these signals within a defined time frame, and subsequently operate actuators [20,21]. The kernel architecture for running these real-time applications differs based on the processor architecture and can be categorized into RTOS and real-time Linux, depending on the approach.

For RTOS applications, we utilized FreeRTOS [22] and applied it to HiFive1 Rev B (RISC-V) and STM3240G- EVAL (ARM M). For real-time Linux, we chose Linux with the Preempt-RT patch [23], conducting our experiments on VisionFive 2 (RISC-V), MIO5272 (x86-64), and Raspberry Pi 4 B (ARM A). Tables 1 and 2 show the specifications of the boards using FreeRTOS and Preempt-RT. We carried out experiments using both FreeRTOS and Linux with the Preempt-RT patch for each ISA to assess and compare their real-time performance capabilities across various processor architectures. In FreeRTOS, we focused on measuring the response times of the queue, stream buffer, message buffer, semaphore, and mutex. The results revealed that ITC mechanisms in FreeRTOS on RISC-V performed better than those on ARM M. For Preempt-RT, our measurements included the response times of the pipe, FIFO, message queue, semaphore, and mutex. Moreover, we evaluated the scheduling latency of each ISA using cyclictest [24] on Linux with the Preempt-RT patch. Across all tested metrics in Preempt-RT, x86-64 showcased the best real-time performance. Notably, RISC-V surpassed ARM A in all ITC mechanisms except FIFO. However, its

cyclictest results indicated the lowest performance. Despite this, the results demonstrated that RISC-V meets the performance requirements necessary for real-time applications. To validate RISC-V's capability in real-world scenarios further, we tested its real-time performance by controlling a collaborative robot. For a comparative analysis, we selected x86-64, recognized for its superior real-time performance, and implemented an EtherCAT master and a real-time motion control application on each ISA. Our results indicated that real-time application based on RISC-V adequately fulfills the performance demands for motion control of a collaborative robot.

The rest of the paper is organized as follows. Section 2 introduces previous related studies and highlights the differences from our work. Section 3 provides an overview of FreeRTOS, details the experimental setup for each ISA, explains the code used for real-time performance measurement, and presents the benchmarking results. Section 4 introduces Preempt-RT, discusses the experiments conducted to assess real-time performance capabilities on real-time Linux, and presents the benchmarking results. Section 5 describes the collaborative robot and experimental setup used to test RISC-V's performance in a real-time environment, explains the experimental methodology, outlines the architectures implemented, and displays the response times for each task, along with the positional and velocity differences for each joint. Finally, we summarize the benchmarking results and conclude the paper.

Table 1. Specifications of the boards using FreeRTOS.

Hardware	Microcontroller Unit	Architecture	RTOS	Kernel Version
HiFive1 Rev B	SiFive FE310-G002 32-bit	RISC-V RV32IMAC	FreeRTOS	10.2.0
STM3240G-EVAL	STM32F407IGH6 with ARM Cortex-M4F 32-bit	ARMv7-M	FreeRTOS	10.2.0

Table 2. Specifications of the boards using Preempt-RT.

Hardware	Microprocessor Unit	Architecture	Real Time Kernel	Kernel Version
VisionFive 2	JH7110 64-bit quad-core	RISC-V U74-MC	Preempt-RT	5.15.0-rt17+
Raspberry Pi 4 B	Broadcom BCM2711 ARM Cortex-A72 64-bit quad-core	ARMv8-A	Preempt-RT	5.15.92-rt57-v8+
MIO-5272	Intel Core I7 6600U	x86-64	Preempt-RT	5.15.79-rt54

2. Related Work

Recent research has focused extensively on analyzing the performance capabilities of RTOS and real-time Linux across various architectures. In one such study [25], the performance of RTOS on ARM M4 and M0 microcontrollers, with a particular emphasis on context switching in systems that support preemptive scheduling. This included FreeRTOS, RT-thread, Keil RTX, and uC/OS-II, and the evaluation was based on events, semaphores, and mailboxes. The study provided insights for choosing an RTOS that is well-suited for small microcontrollers by comparing various RTOS types on similar architectures. However, these experiments were limited to the ARM M architecture and focused exclusively on RTOS synchronization mechanisms. In contrast, our experiment broadens the scope by comparing the real-time performance capabilities of task communication mechanisms across various ISAs, including RISC-V, taking into account the data size transmitted.

In a previous study [26], experiments were carried out to measure execution times and latencies. These included comparing the performance capabilities of two forms of RTOS,

specifically FreeRTOS and uC/OS-III. The measurements focused on real-time latencies, semaphore and mutex operation durations, and the frequency of sustained interrupts. Furthermore, additional tests examined thread creation and mutex locking behaviors to detect any functional issues in the operating systems. Another study [27] focused on the performance of RTOS in IoT devices, measuring the time required for task-switching due to semaphores and message queues, time durations when acquiring and releasing a fixed-size memory region, and the response time of a task to an interrupt service routine. However, as the computational demands and development costs of embedded systems rise, real-time Linux is increasingly being adopted in various fields [23]. Therefore, in our study, we aim to aid in the design and development of real-time applications for embedded systems by providing insights into the real-time performance of not just RTOS, but also the Preempt-RT kernel, across different ISAs.

One study [28] conducted a comparison of the timing accuracy and interrupt latency between a compounded real-time operating system (cRTOS), Preempt-RT, and Xenomai. In other research [29], the timing latencies of Linux patched with Preempt-RT were assessed on different ARM architectures, specifically Raspberry Pi 3 and BeagleBone AI. Researchers [29] also delved into the scheduling latency of real-time Linux on multi-core general-purpose processors, offering development guidelines for real-time applications suited to Xenomai and Preempt-RT kernels. Additionally, experiments were undertaken on scheduling latency using cyclictest on the RISC-V architecture with the Preempt-RT patch [30]. Despite these studies, there is a noticeable lack of information about the development of real-time applications for RISC-V when compared to other ISAs. To address this gap, we implemented real-time applications on both RISC-V and x86-64, providing insight into real-time performance capabilities for robot motion control.

In this paper, we compare various real-time performances across multiple ISAs to demonstrate the performance of the RISC-V architecture for real-time applications with a multi-tasking structure. To validate the practicality of our experimental findings, we implemented an EtherCAT master on both RISC-V and x86-64 to control a six-axis robot. This paper contributes by offering valuable information about real-time performance outcomes, aiding in the selection of an appropriate ISA for real-time systems. It also highlights the potential for significant time and cost savings in the development of real-time applications utilizing the RISC-V.

3. Real-Time Performance Measurement on FreeRTOS

FreeRTOS, an open-source RTOS, is suitable for small embedded systems that are required to undertake multi-tasking [31]. It features a microkernel architecture and is primarily used in microcontrollers and small microprocessors [32]. Its popularity in real-time application development projects stems from its simplicity, compact size, portability, user-friendliness, and compatibility with various hardware architectures. Additionally, the robust FreeRTOS community offers substantial technical support, with system bugs and errors swiftly addressed, providing significant benefits for development [33].

In this section, we outline the uniform RTOS environments and experimental methodologies applied across various ISAs and present the corresponding results. The hardware used in the experiments and the experimental environment are detailed in Table 1. To evaluate the real-time performance of RISC-V and other ISAs in RTOS, we measured inter-task communication response times in FreeRTOS. For the RISC-V assessments, we used the HiFive1 Rev B board, which is equipped with a SiFive FE310-G002 32-bit microcontroller unit. This board is capable of speeds up to 320 MHz and includes 2 kB of SRAM and 4 MB of flash memory. In the ARM M case, our experiments were conducted on the STM3240G-EVAL board, hosting an STMicroelectronics STM32F407IGH6 32-bit microcontroller unit with a maximum speed of 168 MHz and 2 MB of SRAM. Our development environment consisted of Visual Studio Code IDE and PlatformIO, employed as a cross-platform development extension [34]. Due to the limited availability of the FreeRTOS kernel version for the HiFive1

Rev B board in our setup, we aligned the kernel version on the STM3240G-EVAL board to V10.2.0 for consistency.

The experimental results demonstrated that the real-time performance of RISC-V in all inter-task communication (ITC) mechanisms of FreeRTOS outperformed that of ARM. Notably, especially due to the latency being unaffected by data size, it can be inferred that RISC-V based systems are suitable for robust real-time systems.

3.1. Inter-Task Communication of FreeRTOS

In our test application, we measured the communication time between two tasks to assess the real-time performance of the ITC mechanism. Figure 1 illustrates the setup used to evaluate the real-time performance of the ITC mechanism on FreeRTOS. We assigned the priorities of the two tasks differently, with the highest being 8 and a slightly lower priority ranking at 7, conducting the tests for 10 ms. During the experiments, we connected the GPIO pins of each board to an oscilloscope and recorded measurements from over 1000 adequate samples. The real-time performance of the ITC mechanism for each ISA was then determined by calculating the average of these measured response times. The methodology for measuring the ITC mechanism's response time adheres to the approach outlined in [35], with the specific pseudocodes provided in Algorithms 1 and 2.

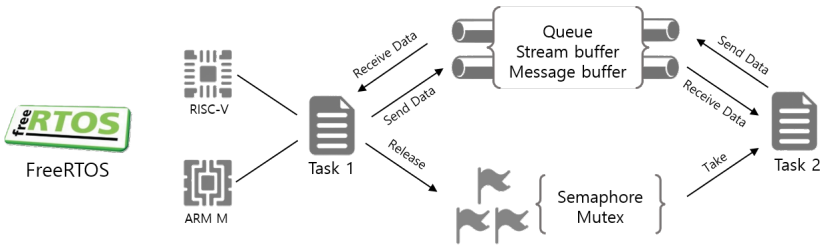

Figure 1. Diagram for measuring the real-time performance of the ITC mechanism on FreeRTOS.

Algorithm 1: Pseudocode for measuring the response times of the queue, stream buffer, and message buffer.

```
Data: N message size;
Function Task_1:
    while (1) do
        Sending a message of size N to task_2;
        GPIO pin on;
        Receiving a message of size N from task_2;
        GPIO pin off;
        Delay for the duration of the period time;
    end
Function Task_2:
    while (1) do
        Receiving a message of size N from task_1;
        Sending a message of size N to task_1;
    end
```

Algorithm 2: Pseudocode for measuring the response times of the semaphore and mutex.

```
Function Task_1:
    while (1) do
        releasing semaphore/mutex;
        GPIO pin on;
        Delay for the duration of the period time;
    end
Function Task_2:
    while (1) do
        waiting for semaphore/mutex acquisition;
        GPIO pin off;
    end
```

3.1.1. Queue

In our experiment, we measured the time required for a task to send data to a queue and for another task to receive and return the data. We utilized two separate queues for sending and receiving data, enabling efficient communication between the tasks. Unlike previous studies [36] that measured RTOS performance using a single task to send a fixed-size message to itself, our experiment aimed to evaluate message communication performance capabilities in a multi-tasking real-time application. We conducted tests with messages of varying sizes, handled by two tasks. To assess how the response time varied with different data sizes, we incrementally increased the data size sent through the queue, starting with 60 bytes and increasing to 600 bytes in 60-byte steps.

Figure 2 displays the experimental results. We observed that with an increase in the data size, the average response time ranged from 9.84 us to 12.85 us for HiFive1 Rev B and from 64.72 us to 334.70 us for STM3240G-EVAL. Notably, the queue's response time was consistently shorter in RISC-V, and the escalation in the response time as the data size was increased was also relatively minimal.

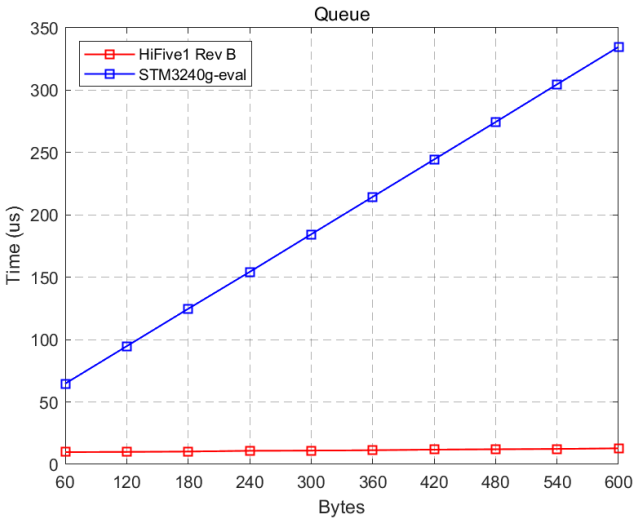

Figure 2. Results of the response time of the queue on FreeRTOS.

3.1.2. Stream Buffer

A stream buffer, a mechanism used to transfer data of varying lengths as a byte stream between two tasks, was employed in our experiment. We utilized two stream buffers, one for sending and the other for receiving data between the tasks, to measure the time required for data exchange. To determine how the response time varied with different data sizes, we increased the data size by 60 bytes in each step of the experiment.

Figure 3 presents the results of these tests. We observed that the average response time ranged from 13.05 us to 32.75 us for HiFive1 Rev B and from 71.40 us to 342.04 us for STM3240G-EVAL.

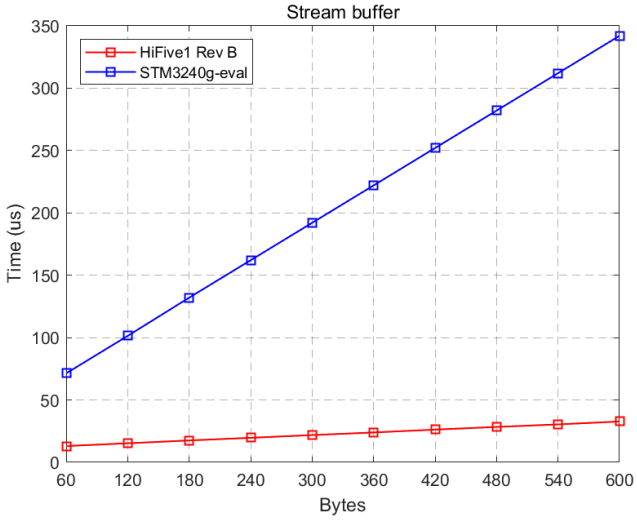

Figure 3. Results of the response time of stream buffer on FreeRTOS.

3.1.3. Message Buffer

In our experiments, we focused on the message buffer, a mechanism specifically designed to transmit variable-length discrete messages between two tasks. We used two message buffers, one for sending and the other for receiving data, to measure the time required to exchange data. To analyze how the response time varied with different data sizes, we systematically increased the data size in increments of 60 bytes.

The outcomes of these experiments are depicted in Figure 4. We noted that the average response time increased from 3.10 us to 9.64 us for HiFive1 Rev B and from 18.31 us to 108.51 us for STM3240G-EVAL.

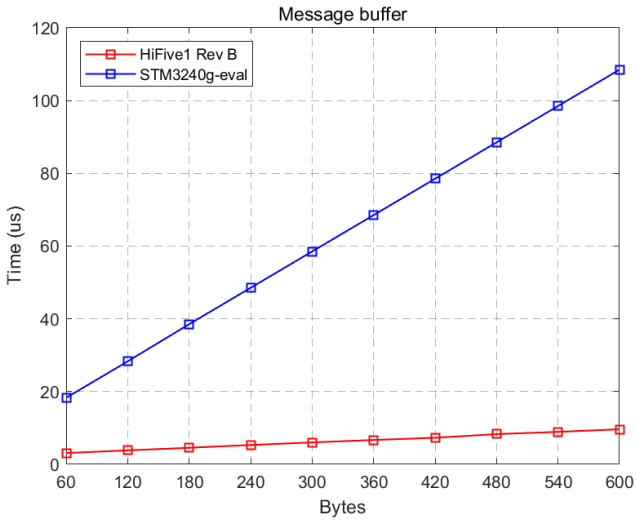

Figure 4. Results of the response time of the message buffer on FreeRTOS.

3.1.4. Semaphore and Mutex

In FreeRTOS, semaphores are categorized into counting and binary semaphores based on the number of preemptible semaphores. Counting semaphores control access to a resource by limiting the number of semaphores available for resource preemption, restricting access once the count reaches zero. On the other hand, binary semaphores and mutexes are used to prevent simultaneous access to a resource by two tasks. In this experiment,

we measured the time taken for a semaphore or mutex to be released and then acquired between two tasks [37].

The results are displayed in Figure 5. The average response times for HiFive1 Rev B were 4.74 us for counting semaphores, 4.68 us for binary semaphores, and 4.90 us for mutex. For STM3240G-EVAL, these times were 12.64 us, 27.97 us, and 13.68 us, respectively. Notably, the binary semaphore experiment on STM3240G-EVAL showed highly unstable signals, with a significant standard deviation of 68.8 us. To investigate potential errors in the ARM's binary semaphore related to the FreeRTOS version, we repeated the experiment using version 10.4.4 without any code changes. This led to stable binary semaphore signals, showing a response time of 12.50 us and a remarkably low standard deviation of 0.002 us. Additionally, version 10.4.4 displayed a modest improvement in the real-time performance for both counting semaphores and mutex compared to version 10.2.0. Table 3 presents the experimental results for semaphores and mutexes, including FreeRTOS version 10.4.4.

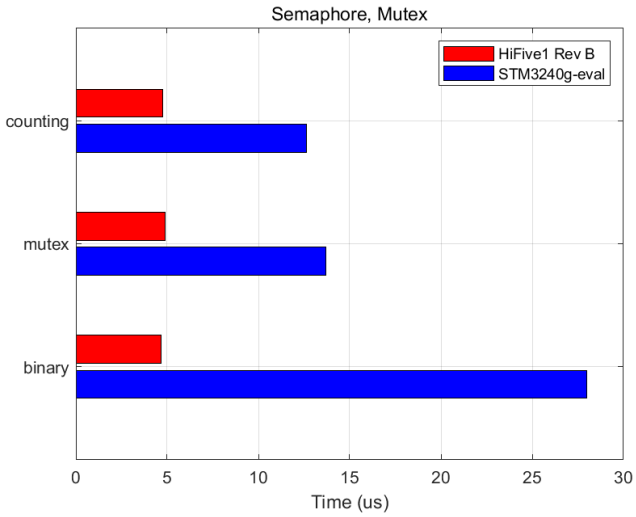

Figure 5. Results of the response times of semaphores and mutexes on FreeRTOS.

Table 3. Results of the response times of semaphores and mutexes, including FreeRTOS V10.4.4.

	HiFive1 Rev B (10.2.0)				STM3240G-EVAL (10.2.0)				STM3240G-EVAL (10.4.4)			
ITC	**Mean (us)**	**Min (us)**	**Max (us)**	**Sdev (us)**	**Mean (us)**	**Min (us)**	**Max (us)**	**Sdev (us)**	**Mean (us)**	**Min (us)**	**Max (us)**	**Sdev (us)**
Counting Semaphore	4.74	4.73	4.74	0.004	12.64	12.63	12.64	0.003	12.39	12.39	12.40	0.002
Mutex	4.90	4.73	4.95	0.037	13.68	13.12	13.73	0.1	13.51	13.50	13.52	0.004
Binary Semaphore	4.68	4.65	4.73	0.019	27.97	0.07	334.71	68.8	12.50	12.50	12.51	0.002

4. Real-Time Performance Measurement of Preempt-RT

The Preempt-RT patch consists of a series of kernel modifications aimed at minimizing latency and enhancing determinism for tasks on Linux [38]. By applying the Preempt-RT patch to a Linux kernel, real-time tasks can be executed alongside non-real-time tasks in the same user space. This integration offers features such as symmetrical multiprocessing (SMP) and priority inheritance to avoid priority inversion outcomes [39,40]. Furthermore, the development of real-time applications is facilitated by its compatibility with existing libraries and tools that comply with the POSIX standard [23].

In this section, we describe the real-time Linux environment and experimental methods for various ISAs, and their corresponding experimental results. To evaluate the real-time performance of RISC-V and other ISAs in a real-time Linux environment, we patched Linux with Preempt-RT and measured the response time of the ITC mechanism. Figure 6 depicts

the setup for assessing the real-time performance of the ITC mechanism in Preempt-RT. We also conducted a cyclictest to evaluate the scheduling latency of each ISA. For consistency across experiments, we applied the Preempt-RT patch to a Debian-based Linux version 5.15, suitable for each board, within a desktop environment. This setup was then cross-compiled and installed on all boards.

The hardware used and the versions of the real-time kernel are detailed in Table 2. For these experiments, the RISC-V VisionFive 2 board with a StarFive JH-7110 64-bit microprocessor unit at 1.5 GHz and 8 GB SDRAM was utilized. The ARM A ISA was represented by Raspberry Pi 4 B, equipped with a Broadcom BCM2711 64-bit microprocessor unit at 1.5 GHz and 8 GB SDRAM. The x86-64 setup involved the MIO-5272 board, featuring a Intel Core i7 6600U 64-bit microprocessor unit at 2.6 GHz and 16 GB of SDRAM. The experimental findings indicate that in Preempt-RT, x86-64 offered the best real-time performance for the ITC mechanism, and except for FIFO, RISC-V outperformed ARM A. In the cyclictest results, x86-64 showed superior performance, while RISC-V recorded the lowest outcome. More detailed information is well-documented in the references [41].

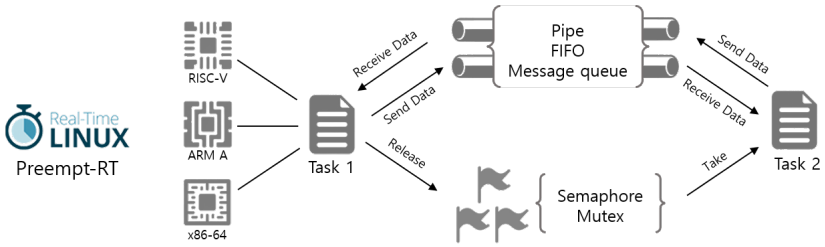

Figure 6. Diagram for measuring the real-time performance of the ITC mechanism in Preempt-RT.

4.1. Inter-Task Communication of Preempt-RT

In the test application to measure the real-time performance of the ITC mechanism in Preempt-RT, we measured the communication time between two tasks, similar to the experiment conducted previously in FreeRTOS. We set the corresponding task priorities to 95 and 90 and established a 10 ms period for the experiment, which ran for a total duration of 30 min. The response time of the ITC mechanism was captured using the high-resolution time function clock_gettimer(), and the data were saved to a file for analysis. The real-time performance capabilities of the ITC mechanism for each ISA were then determined by calculating the average of these recorded response times.

The results reveal that for all ITC mechanisms, RISC-V based systems exhibit higher latency compared to x86 systems. However, they show superior performance compared to ARM based systems. Additionally, the results demonstrate relatively consistent performance regardless of data size. Therefore, it can be confirmed that RISC-V based systems are beneficially applicable in real-time systems with multitasking structures.

4.1.1. Pipe

A 'pipe' is a mechanism that facilitates the transmission of messages without set boundaries by means of a byte stream between two tasks [42]. In our experiment, we used two pipes, one for sending and the other for receiving data, to measure the time required to exchange data. To observe the variations in the response time with different data sizes, we conducted the experiments while gradually increasing the data size from 4 to 3072 bytes. The outcomes of these experiments are depicted in Figure 7. The average response time for VisionFive 2 was shorter than that of Raspberry Pi 4 B but longer than that of MIO-5272.

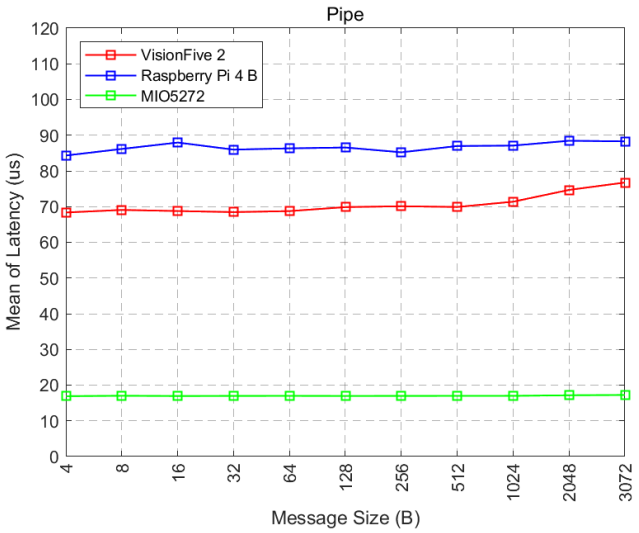

Figure 7. Results of the response time of the pipe on Preempt-RT.

4.1.2. FIFO

FIFO, functioning similarly to a pipe, is a mechanism used for message transmission between two tasks. It can be handled like a file in a file system, akin to a regular file [43]. In our experiment, we utilized two FIFOs, one for sending and the other for receiving data, to measure the time needed to complete the data exchange. To assess how the response time varied with different data sizes, we progressively increased the data size from 4 to 3072 bytes in each test. The results of these experiments are presented in Figure 8. We observed that the average response time for VisionFive 2 was less than that of Raspberry Pi 4 B but greater than that of MIO-5272.

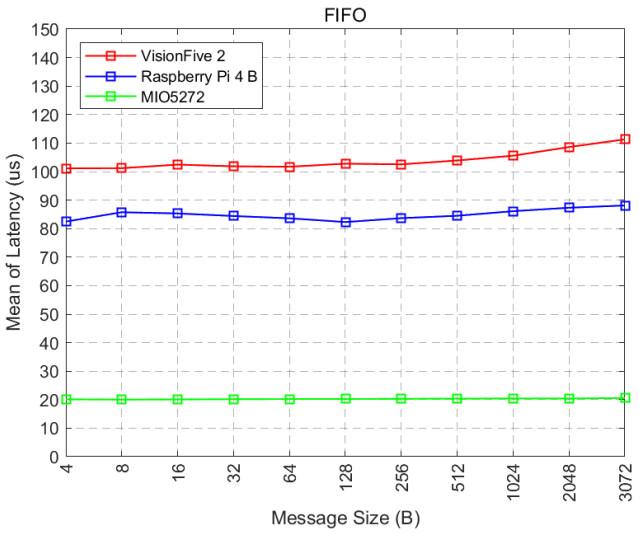

Figure 8. Results of the response time of FIFO on Preempt-RT.

4.1.3. Message Queue

A message queue is a mechanism used for transmitting fixed-length messages between two tasks, ensuring that the receiver receives complete messages as sent by the sender. In our experiment, we used two message queues for sending and receiving data to measure the time required to exchange data. To determine how the response time varied with different data sizes, we incrementally increased the data size from 4 to 3072 bytes during these tests. The outcomes of these experiments are illustrated in Figure 9. We found that the average response time for the VisionFive 2 was less than that of the Raspberry Pi 4 B but greater than that of the MIO-5272.

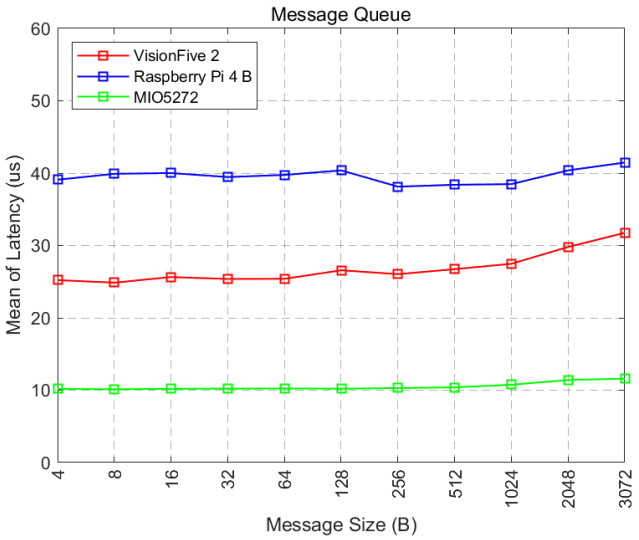

Figure 9. Results of the response time of the message queue on Preempt-RT.

4.1.4. Semaphore and Mutex

We measured the time it took for two tasks to release and subsequently acquire a semaphore or mutex. The results of this experiment are displayed in Figure 10. For both the semaphore and mutex, the average response times were 10.84 us and 10.01 us for VisionFive 2, 18.29 us, and 18.11 us for Raspberry Pi 4 B. The corresponding times were 4.92 us and 4.90 us for MIO-5272.

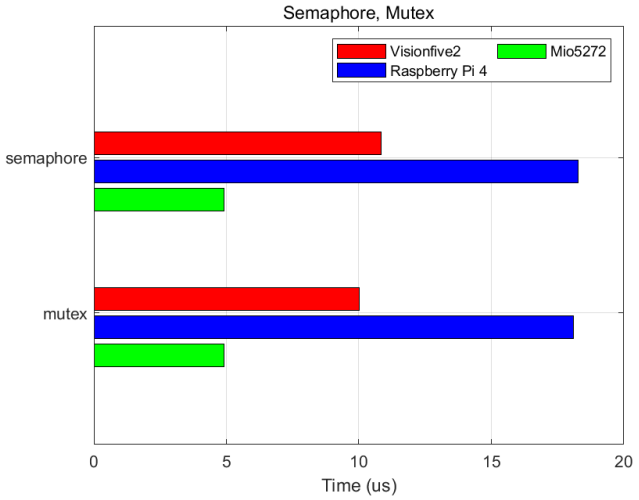

Figure 10. Results of the response time of semaphore and mutex on Preempt-RT.

4.2. Scheduling Latency

In a real-time Linux environment, we employed the cyclictest benchmarking tool to evaluate the scheduling latency of each ISA. Cyclictest measures the difference between a thread's intended wake-up time and the time it wakes up for each ISA [44]. A previous study [30] used cyclictest on a Preempt-RT patched Linux system to assess the real-time performance of RISC-V. However, our experiment expanded this analysis to include not only RISC-V but also ARM A and x86-64, enabling a comprehensive comparison of their real-time performance capabilities. The measurements for each ISA were carried out over a two-hour period with 1 ms intervals.

For cyclictest, we used the -m option to prevent page faults and the -S option for efficient testing in SMP (symmetric multiprocessing) systems ($cyclictest -m -S -p97 -i1000 -N -d0 -D2h). Our results showed that MIO- 5272 exhibited the shortest average scheduling latency of 3.92 us, whereas VisionFive 2 had the longest at 32.69 us. In terms of the standard

deviation, MIO-5272 also had the smallest value at 0.20 us, while VisionFive 2 had the largest outcome at 7.02 us. A summary of the cyclictest results is provided in Figure 11 and Table 4.

The results demonstrate that the latency of RISC-V based systems exhibits higher mean and standard deviation values for latency compared to systems based on the other two ISAs. However, with a maximum latency of 202.984 us and a mean value of 32.690 us, it is evident that these results are suitable for real-time system applications. The validation of this performance evaluation (for practical applicability) will be further verified in the subsequent section through motion control with a collaborative robot.

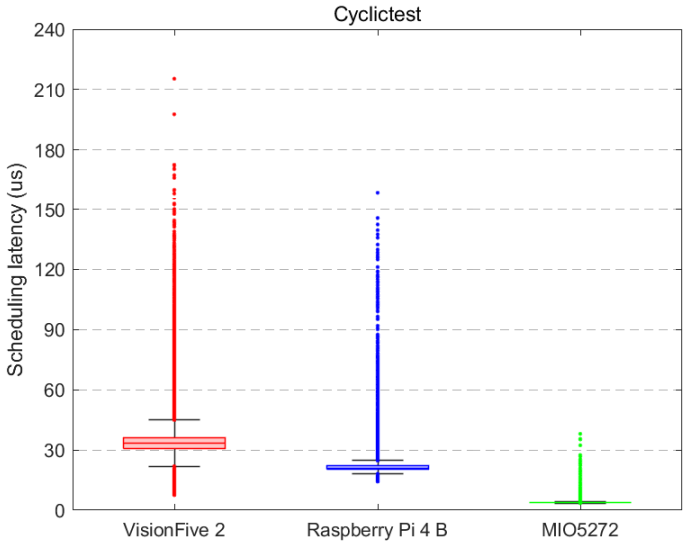

Figure 11. Boxplot of observed scheduling latencies for various ISAs measured with cyclictest on a Preempt-RT patched Linux system.

Table 4. Results of scheduling latencies for various ISAs measured with cyclictest on a Preempt-RT patched Linux system.

Hardware	Mean (us)	Min (us)	Max (us)	Std (us)
VisionFive 2	32.690	7.662	202.984	7.023
Raspberry Pi 4 B	21.472	14.311	158.704	1.440
MIO-5272	3.919	3.489	38.051	0.202

5. Real-Time Performance Experiment on a Collaborative Robot

In this section, we describe the experimental environment used to assess real-time Linux motion control of a collaborative robot based on RISC-V. Figure 12 displays the six-axis collaborative robot from Neuromeka that was utilized in our experiments [45]. We detail the implementation procedure and results of the executed real-time application. By assessing the real-time performance of each ISA via the ITC mechanism and with cyclictest experiments within the Preempt-RT framework, we were able to ascertain that RISC-V meets the requisite performance benchmarks for real-time application deployment. Accordingly, we developed a real-time application for collaborative robot motion control and examined its response times to ensure the ISA's compatibility with real-time operational demands.

To construct the real-time motion control system, our platform of choice was Preempt-RT, which is consistent with our previous experimental environment. For the EtherCAT master implementation, we adopted IgH EtherCAT, an open-source fieldbus protocol [41,46]. In the ensuing experiments, we selected VisionFive 2 on RISC-V and MIO-5272 of x86-64,

which is recognized for its superior real-time performance. These boards were patched with Preempt-RT and subsequently integrated with the IgH EtherCAT master.

In prior research, referenced as [47], the focus was on EtherCAT master's implementation and a real-time performance evaluation for a Python-based servo motor control application. This paper directly evaluates RISC-V's applicability in real-time systems by evaluating response times during a collaborative robot motion control test. Figure 13 shows the architecture of the EtherCAT network with Preempt-RT for motion control. Commands dispatched from the EtherCAT master for motion control traverse the Ethernet driver, subsequently reaching the robot's six actuators via the network interface. The EtherCAT application interface provides the requisite libraries and tools pivotal for EtherCAT motion control applications, while the EtherCAT master core contains crucial data pertaining to both master and slave entities.

The EtherCAT master core was designed to abstract system-level routines related to EtherCAT, utilizing the IgH EtherCAT application interface and library. The master base is responsible for configuring the EtherCAT master according to the connected slaves. Furthermore, data acquisition and control for each slave should be executed in line with its specific slave base. The real-time functionalities intrinsic to the Preempt-RT kernel manifest through RT-POSIX. Hence, every task integral to the motion control application functions as a real-time task, operated via POSIX threads within the RT-POSIX environment.

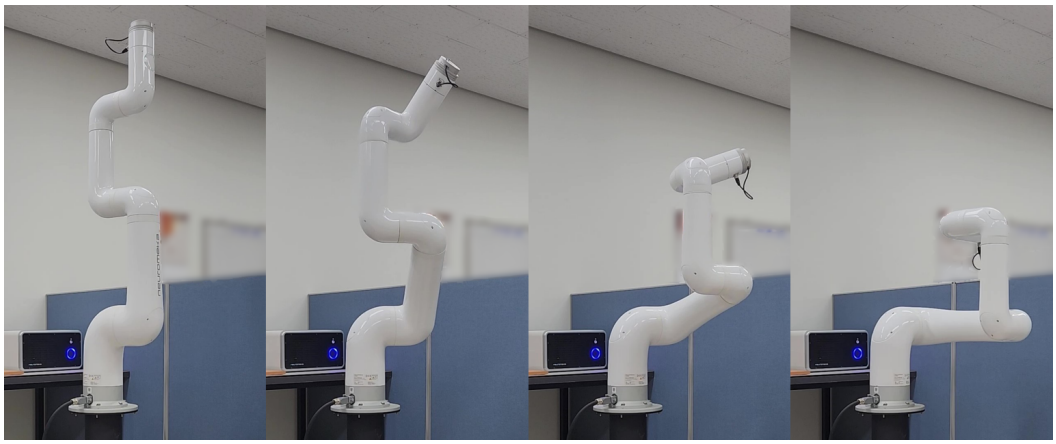

Figure 12. Motion control with Indy7 collaborative robot.

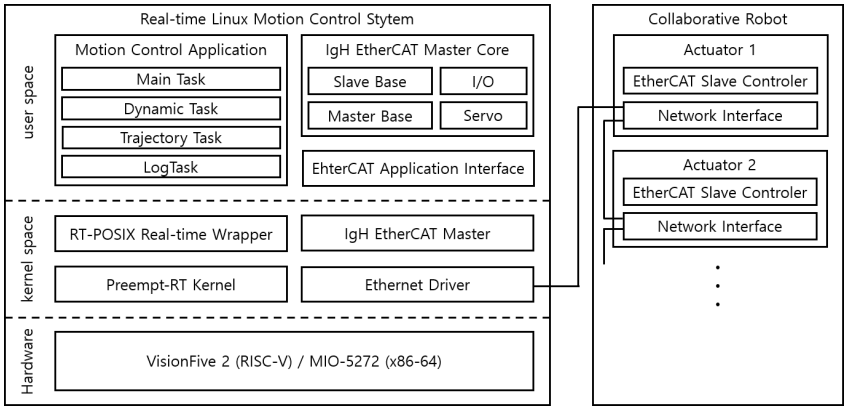

Figure 13. Architecture of the EtherCAT network with Preempt-RT for motion control.

Motion Control of a Collaborative Robot

To ensure the efficient operation of the motion control application, its functionalities were modularly partitioned and decomposed across multiple tasks, culminating in a multi-task implementation. Figure 14 shows the architecture of the motion control application. The dynamic task is responsible for computing inverse dynamics and leverages the open-

source Kinematics and Dynamics Library (KDL) provided by Orocos. The trajectory task specifies the designated trajectory and velocity for each joint. The main task controls Indy7's operations, utilizing the IgH EtherCAT protocol. It receives control data from message queues managed by individual tasks. Finally, the log task logs the response times of each task, encompassing its own, and integrates the encoder data, subsequently storing this information in a designated file.

In the experiment conducted here, motion control involved rotating each motor angle of Indy7 [45] within a range from 0 to 90 degrees. Concurrently, the real-time performance was evaluated by examining the response times associated with each task.

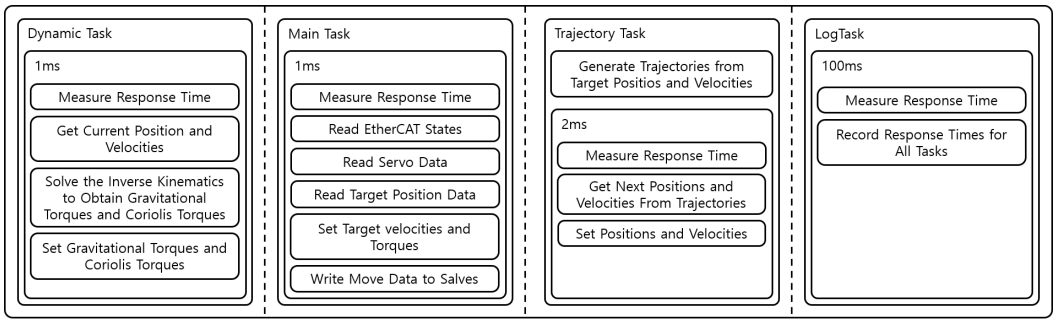

Figure 14. Motion control application architecture.

Table 5 shows the experiment task configurations and the corresponding response time results. The data reveal that, concerning the application response times across both architectures, the dynamic task had the shortest durations, whereas the log task registered the longest. Specifically, for VisionFive 2, the response times for the dynamic task and log task were 112.71 us and 329.50 us, respectively. In contrast, for MIO-5272, these times were 95.89 us for the dynamic task and 164.08 us for the log task.

Table 5. Configuration and response time results of motion control tasks.

Task	Response Time (us)		Period (ms)	Priority
	VisionFive 2	MIO-5272		
Dynamic Task	112.71	95.89	1	95
Main Task	292.14	152.99	1	90
Trajectory Task	319.04	159.97	2	85
Log Task	329.50	164.08	100	70

Figure 15 depicts the discrepancies between the reference and actual trajectories for each joint of the Indy7, controlled by VisionFive 2. In this figure, the deviation in degrees for each joint of the collaborative robot remains within a 2-degree margin. The experimental outcomes for x86-64 closely mirror those of RISC-V, thus only the velocity data for the joints are shown. Table 6 presents the velocity discrepancies between the reference and actual trajectories for each joint when comparing the two ISAs. The occurrence of negative min values in Table 6 is attributed to measurements taken in reverse directions. Consequently, the average difference in joint velocity was the highest in Joint 5, showing 0.2843 deg/s in RISC-V and 0.2874 deg/s in x86-64. Additionally, the maximum difference in joint velocity was the highest in Joint 4, with RISC-V showing 12.9488 deg/s and x86-64 showing 13.0634 deg/s. The positional and velocity values for each joint were derived from encoder sensor feedback data.

The experimental results indicate that the hardware maintained a positional variance of less than two degrees across all joints. Nonetheless, notable velocity discrepancies were evident at the initiation and culmination of the motion sequences. These fluctuations likely arose from real-time adjustments and optimizations of the velocity controller to

ensure accurate and smooth motion. Notably, both VisionFive 2 and MIO-5272, built on x86-64 ISA, and recognized for their superior real-time performance, effectively performed motion control for Indy7. This underscores the potential feasibility of employing RISC-V in real-time applications, particularly for collaborative robot management.

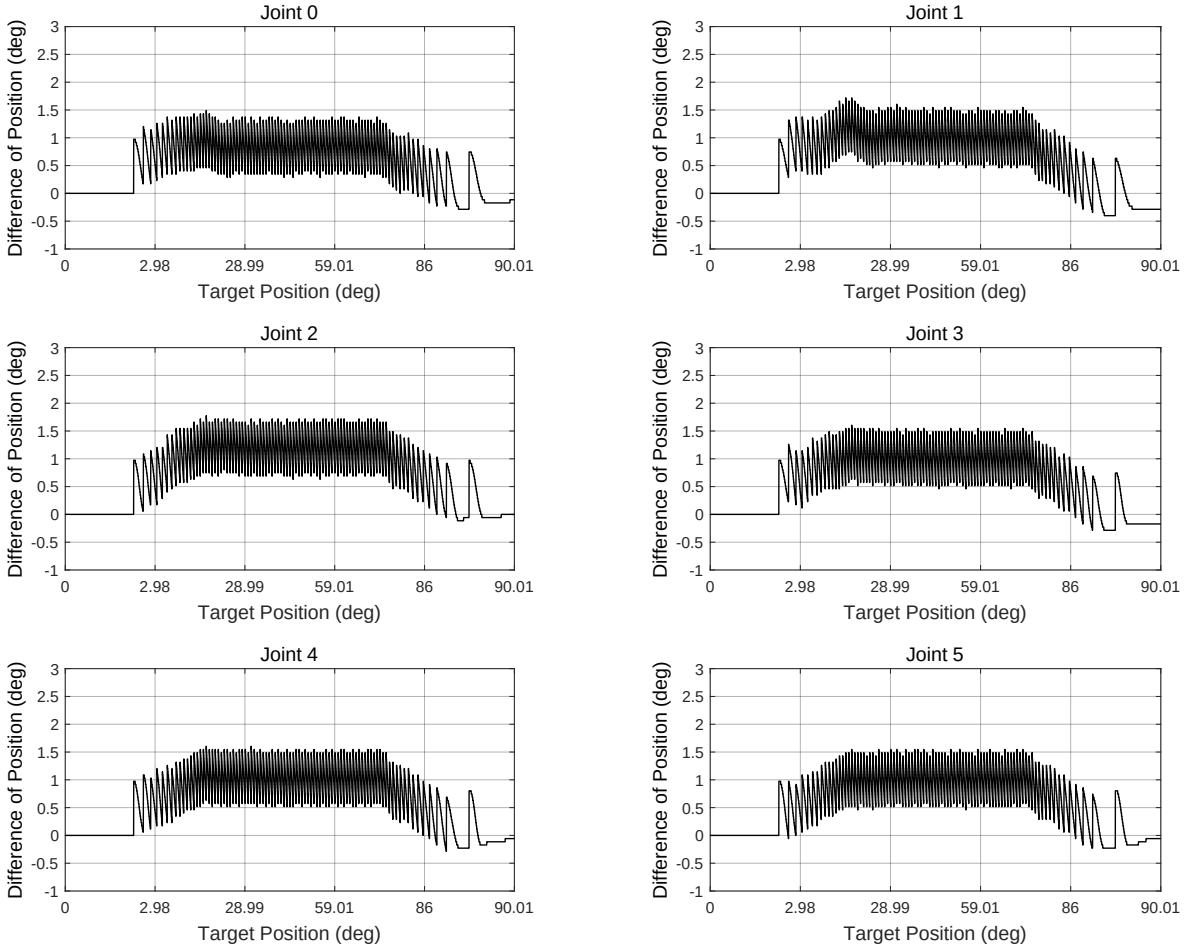

Figure 15. Results of the position difference observed from 0 to 90 degrees for all joints of a collaborative robot controlled on RISC-V.

Table 6. Results of the velocity difference for all joints observed while controlling a collaborative robot from 0 to 90 degrees on two ISAs.

	VisionFive 2				MIO-5272			
Joint	**Mean (deg/s)**	**Min (deg/s)**	**Max (deg/s)**	**Sdev (deg/s)**	**Mean (deg/s)**	**Min (deg/s)**	**Max (deg/s)**	**Sdev (deg/s)**
Joint 0	−0.2282	−8.8236	6.9901	1.8447	−0.2356	−8.9381	6.9901	1.8670
Joint 1	−0.2519	−7.7349	6.9901	2.2061	−0.2484	−7.5057	6.9901	1.7420
Joint 2	−0.2448	−11.2873	6.9901	2.2061	−0.2355	−11.3446	6.9901	2.2189
Joint 3	−0.2347	−11.6883	6.9901	2.0839	−0.2403	−12.2613	7.1620	2.1197
Joint 4	−0.2595	−12.9488	6.9901	2.2177	−0.2627	−13.0634	6.9901	2.2180
Joint 5	−0.2843	−12.0894	6.9901	2.0947	−0.2874	−12.3186	6.9901	2.1172

6. Conclusions

In this paper, we analyzed the real-time performance of RISC-V and other ISAs to provide development metrics for real-time system applications. We employed two real-time approaches, FreeRTOS and Linux with Preempt-RT. For FreeRTOS, we evaluated the

ITC mechanism's response time on both HiFive1 Rev B (RISC-V) and STM3240G-EVAL (ARM M). Meanwhile, with Preempt-RT, we assessed the ITC mechanism's response time and determined the scheduling latency using cyclictest across three types of hardware: VisionFive 2 (RISC-V), Raspberry Pi 4 B (ARM A), and MIO-5272 (x86-64). Following our analysis, we successfully implemented the IgH EtherCAT master on both RISC-V and the outperforming x86-64 ISA. Utilizing a multi-task real-time application, we controlled the motion of a six-axis collaborative robot, known as Indy7, to assess the real-time capabilities of both ISAs. Our findings indicate that while RISC-V excels in real-time performance, particularly with the ITC mechanism in RTOS, its performance and scheduling latency in real-time Linux were comparatively low. However, with regard to real-time robot control applications, RISC-V's performance stood competitively alongside those of other ISAs. For the robot control application, task data communication was facilitated through message queues, which proved to be more efficient in RISC-V than other ITC mechanisms and exhibited the smallest performance gap when compared to the top-performing ISA, x86-64. Thus, RISC-V based applications employing message queues are capable of achieving the required performance standards for controlling collaborative robots in a real-time environment.

In future research, leveraging the insights obtained from our real-time performance analysis in this study, we aim to expand the scope of RISC-V applications within real-time environments, particularly for overseeing diverse robotic systems such as mobile robots. Furthermore, we intend to explore the feasibility of integrating advanced intelligent control systems, harnessing deep learning and machine learning techniques, in RISC-V based real-time applications.

Author Contributions: Conceptualization, T.Y. and B.W.C.; methodology, T.Y.; software, T.Y.; validation, T.Y.; investigation, T.Y.; resources, B.W.C.; data curation, T.Y.; writing—original draft preparation, T.Y.; writing—review and editing, T.Y. and B.W.C.; visualization, T.Y.; supervision, B.W.C.; project administration, B.W.C.; funding acquisition, B.W.C. All authors have read and agreed to the published version of the manuscript.

Funding: This work was financially supported by SeoulTech (Seoul National University of Science and Technology).

Data Availability Statement: The relevant data and source code are available upon request from the corresponding author.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Davis, R.I.; Burns, A. A survey of hard real-time scheduling for multiprocessor systems. *ACM Comput. Surv. (CSUR)* **2011**, *43*, 1–44. [CrossRef]
2. Stankovic, J.A. Strategic directions in real-time and embedded systems. *ACM Comput. Surv. (CSUR)* **1996**, *28*, 751–763. [CrossRef]
3. Akesson, B.; Nasri, M.; Nelissen, G.; Altmeyer, S.; Davis, R.I. An empirical survey-based study into industry practice in real-time systems. In Proceedings of the 2020 IEEE Real-Time Systems Symposium (RTSS), Houston, TX, USA, 1–4 December 2020; pp. 3–11.
4. de Matos Pedro, A.; Pereira, D.; Pinho, L.M.; Pinto, J.S. Logic-based schedulability analysis for compositional hard real-time embedded systems. *ACM SIGBED Rev.* **2015**, *12*, 56–64. [CrossRef]
5. Liu, D.; Hu, X.S.; Lemmon, M.D.; Ling, Q. Firm real-time system scheduling based on a novel QoS constraint. *IEEE Trans. Comput.* **2006**, *55*, 320–333. [CrossRef]
6. Song, I.; Kim, S.; Karray, F. A real-time scheduler design for a class of embedded systems. *IEEE/ASME Trans. Mechatronics* **2008**, *13*, 36–45. [CrossRef]
7. Asanović, K.; Patterson, D.A. *Instruction Sets Should Be Free: The Case for Risc-v*; Tech. Rep. UCB/EECS-2014-146; EECS Department, University of California: Berkeley, CA, USA, 2014.
8. Patterson, D. Reduced instruction set computers then and now. *Computer* **2017**, *50*, 10–12. [CrossRef]
9. Torres-Sánchez, E.; Alastruey-Benedé, J.; Torres-Moreno, E. Developing an AI IoT application with open software on a RISC-V SoC. In Proceedings of the 2020 XXXV Conference on Design of Circuits and Integrated Systems (DCIS), Segovia, Spain, 18–20 November 2020; pp. 1–6.
10. Patterson, D.; Waterman, A. *The RISC-V Reader: An Open Architecture Atlas*; Strawberry Canyon: Berkeley, CA, USA, 2017.

11. Ince, M.N.; Ledet, J.; Gunay, M. Building an open source Linux computing system on RISC-V. In Proceedings of the 2019 1st International Informatics and Software Engineering Conference (UBMYK), Ankara, Turkey, 6–7 November 2019; pp. 1–4.
12. Cannizzaro, M.J.; Gretok, E.W.; George, A.D. Risc-v benchmarking for onboard sensor processing. In Proceedings of the 2021 IEEE Space Computing Conference (SCC), Laurel, MD, USA, 23–27 August 2021; pp. 46–59.
13. Palmiero, C.; Di Guglielmo, G.; Lavagno, L.; Carloni, L.P. Design and implementation of a dynamic information flow tracking architecture to secure a RISC-V core for IoT applications. In Proceedings of the 2018 IEEE High Performance Extreme Computing Conference (HPEC), Waltham, MA, USA, 25–27 September 2018; pp. 1–7.
14. Kalapothas, S.; Galetakis, M.; Flamis, G.; Plessas, F.; Kitsos, P. A Survey on RISC-V-Based Machine Learning Ecosystem. *Information* **2023**, *14*, 64. [CrossRef]
15. Abraham, V.; Ranpariya, D.; Parikh, P.; Gajjar, S.; Shah, D. Exploration of FreeRTOS on a RISC-V Architecture. In *ICT Systems and Sustainability: Proceedings of ICT4SD 2022*; Springer: Singapore, 2022; pp. 1–11.
16. Walls, C. *Embedded RTOS Design: Insights and Implementation*; Newnes: Boulevard, UK, 2020.
17. Stevens, W.R.; Rago, S.A.; Ritchie, D.M. *Advanced Programming in the UNIX Environment*; Addison-Wesley New York: New York, NY, USA, 1992; Volume 4.
18. Raghavan, P.; Lad, A.; Neelakandan, S. *Embedded Linux System Design and Development*; CRC Press: Boca Raton, FL, USA, 2005.
19. Hambarde, P.; Varma, R.; Jha, S. The survey of real time operating system: RTOS. In Proceedings of the 2014 International Conference on Electronic Systems, Signal Processing and Computing Technologies, Nagpur, India, 9–11 January 2014; pp. 34–39.
20. Qian, K.; Wang, J.; Gopaul, N.S.; Hu, B. Low cost multisensor kinematic positioning and navigation system with Linux/RTAI. *J. Sens. Actuator Netw.* **2012**, *1*, 166–182. [CrossRef]
21. Cifuentes-Cuadros, A.A.; Romero, E.; Caballa, S.; Vega-Centeno, D.; Elias, D.A. The LIBRA NeuroLimb: Hybrid Real-Time Control and Mechatronic Design for Affordable Prosthetics in Developing Regions. *Sensors* **2023**, *24*, 70. [CrossRef]
22. Melot, N. Study of an Operating System: FreeRTOS. 2009. Available online: https://wiki.csie.ncku.edu.tw/embedded/FreeRTOS_Melot.pdf (accessed on 22 June 2023).
23. Reghenzani, F.; Massari, G.; Fornaciari, W. The real-time linux kernel: A survey on preempt_rt. *ACM Comput. Surv. (CSUR)* **2019**, *52*, 1–36. [CrossRef]
24. de Oliveira, D.B.; Casini, D.; de Oliveira, R.S.; Cucinotta, T. Demystifying the real-time linux scheduling latency. In Proceedings of the 32nd Euromicro Conference on Real-Time Systems (ECRTS 2020), Virtual, 7–10 July 2020.
25. Anh, T.N.B.; Tan, S.L. Real-time operating systems for small microcontrollers. *IEEE Micro* **2009**, *29*, 30–45. [CrossRef]
26. Peng, L.; Guan, F.; Perneel, L.; Timmerman, M. Behaviour and performance comparison between FreeRTOS and μC/OS-III. *Int. J. Embed. Syst.* **2016**, *8*, 300–312. [CrossRef]
27. Raymundo Belleza, R.; de Freitas Pignaton, E. Performance study of real-time operating systems for internet of things devices. *IET Softw.* **2018**, *12*, 176–182. [CrossRef]
28. Yang, C.F.; Shinjo, Y. Obtaining hard real-time performance and rich Linux features in a compounded real-time operating system by a partitioning hypervisor. In Proceedings of the 16th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments, Lausanne, Switzerland, 17 March 2020; pp. 59–72.
29. Adam, G.K.; Petrellis, N.; Doulos, L.T. Performance assessment of linux kernels with PREEMPT_RT on ARM-based embedded devices. *Electronics* **2021**, *10*, 1331. [CrossRef]
30. Jämbäck, M. Evaluation of Real-Time Linux on RISC-V Processor Architecture. Master's Thesis, Tampere University, Tampere, Finland, 2022.
31. Neuhard, Y. A Comparison of Real-time Operating Systems for Embedded Computing. Available online: https://es.cs.rptu.de/publications/datarsg/Neuh22.pdf (accessed on 11 July 2023).
32. Qutqut, M.H.; Al-Sakran, A.; Almasalha, F.; Hassanein, H.S. Comprehensive survey of the IoT open-source OSs. *IET Wirel. Sens. Syst.* **2018**, *8*, 323–339. [CrossRef]
33. Shammar, E.A.; Zahary, A.T. The Internet of Things (IoT): A survey of techniques, operating systems, and trends. *Library Hi Tech* **2020**, *38*, 5–66. [CrossRef]
34. Amos, B. *Hands-On RTOS with Microcontrollers: Building Real-Time Embedded Systems Using FreeRTOS, STM32 MCUs, and SEGGER Debug Tools*; Packt Publishing Ltd.: Birmingham, UK, 2020.
35. Koh, J.H.; Choi, B.W. Real-time performance of real-time mechanisms for rtai and xenomai in various running conditions. *Int. J. Control. Autom.* **2013**, *6*, 235–246.
36. Venkataraman, A.; Jagadeesha, K.K. Evaluation of inter-process communication mechanisms. *Architecture* **2015**, *86*, 64.
37. Alagalla, A.; Rajapaksha, U.S. Techniques of Enhancing Synchronization Efficiency of Distributed Real Time Operating Systems. In Proceedings of the 2022 2nd International Conference on Advanced Research in Computing (ICARC), Belihuloya, Sri Lanka, 23–24 February 2022; pp. 308–313.
38. Abbott, D. *Linux for Embedded and Real-Time Applications*; Elsevier: Boulevard, UK, 2011.
39. Litayem, N.; Saoud, S.B. Impact of the Linux real-time enhancements on the system performances for multi-core intel architectures. *Int. J. Comput. Appl.* **2011**, *17*, 17–23. [CrossRef]
40. Delgado, R.; Choi, B.W. New insights into the real-time performance of a multicore processor. *IEEE Access* **2020**, *8*, 186199–186211. [CrossRef]

41. Delgado, R.; Hong, C.H.; Shin, W.C.; Choi, B.W. Implementation and performance analysis of an EtherCAT Master on the latest real-time embedded Linux. *Int. J. Appl. Eng. Res.* **2015**, *10*, 44603–44609.
42. Silva, M.; Cerdeira, D.; Pinto, S.; Gomes, T. Operating systems for Internet of Things low-end devices: Analysis and benchmarking. *IEEE Internet Things J.* **2019**, *6*, 10375–10383. [CrossRef]
43. Kerrisk, M. *The Linux Programming Interface: A Linux and UNIX System Programming Handbook*; No Starch Press: San Francisco, CA, USA, 2010.
44. Alonso, S.; Lázaro, J.; Jiménez, J.; Bidarte, U.; Muguira, L. Evaluating latency in multiprocessing embedded systems for the smart grid. *Energies* **2021**, *14*, 3322. [CrossRef]
45. Tadese, M.; Pico, N.; Seo, S.; Moon, H. A Two-Step Method for Dynamic Parameter Identification of Indy7 Collaborative Robot Manipulator. *Sensors* **2022**, *22*, 9708. [CrossRef]
46. Delgado, R.; Choi, B.W.; Song, H. Application of etherCAT in microgrid communication network: A case study. In Proceedings of the 2018 International Conference on Platform Technology and Service (PlatCon), Jeju, Republic of Korea, 29–31 January 2018; pp. 1–6.
47. Cho, S.Y.; Delgado, R.; Choi, B.W. Feasibility Study for a Python-Based Embedded Real-Time Control System. *Electronics* **2023**, *12*, 1426. [CrossRef]