The Design and Real-Time Optimization of an EtherCAT Master for Multi-Axis Motion Control

Jianjun Zhang 1,2,*, Manjiang Xia 1, Han Li 1, Shasha Li 1 and Juan Shi 3

1 School of Electrical Engineering and Automation, Henan Polytechnic University, Jiaozuo 454099, China; xiamanjian@home.hpu.edu.cn (M.X.); lihaniss@163.com (H.L.); 10460230389@hpu.edu.cn (S.L.)
2 Henan International Joint Laboratory of Direct Drive and Control of Intelligent Equipment, Jiaozuo 454003, China
3 Air Defence and Missile Defence Institute, Air Force Engineering University, Xi’an 710051, China; shijuan@mail.nwpu.edu.cn
* Correspondence: sitiandji@hpu.edu.cn

Abstract: To address the issues of low bandwidth, weak real-time performance, and poor synchronization in traditional fieldbuses for multi-axis motion control, a solution for the implementation of an EtherCAT master based on the IgH EtherCAT Master open-source software framework and an embedded hardware platform is proposed. On a hardware platform centered around the AM64x Sitara processor, a Linux real-time operating system based on the Xenomai real-time kernel is constructed, and the IgH master framework is ported to realize a high-performance EtherCAT master. The configuration process of the EtherCAT bus is detailed, a master application program is developed, and methods for the real-time performance optimization of the master—such as exclusive CPU usage by the master process and the optimization of the network card driver—are proposed. Finally, experiments are conducted on a six-axis servo control platform, with the packet analysis of the periodic EtherCAT data frames sent by the master. The experimental results show that the optimized master, under a high-speed communication cycle of 500 microseconds, maintains maximum jitter within 20 microseconds and average jitter within 1 microsecond, meeting the requirements for high-precision multi-axis motion control.

Keywords: embedded platform; EtherCAT; IgH; Linux real-time system; motion control; optimize

1. Introduction

With the rapid development of industrial automation, the demand for efficient, reliable data communication and real-time performance in modern industrial systems is continuously increasing. Although traditional fieldbuses have played a significant role in past industrial automation applications, their shortcomings are gradually becoming apparent, primarily including low communication speeds [1], which can become a bottleneck in large-scale and complex industrial systems, affecting data transmission and real-time performance [2]; small data transmission volumes, with only a limited amount of data being transmitted per cycle, making them less flexible for applications involving large-scale data acquisition and real-time control [3]; and poor synchronization performance, with significant differences in synchronization time between servos during multi-axis motion control [4]. With the development of Industry 4.0, more and more new-generation industrial communication technologies and standards are emerging. These new technologies typically feature higher communication speeds, greater transmission distances, stronger anti-interference capabilities, and better interoperability, gradually replacing traditional fieldbus applications [5]. Many companies have subsequently introduced their own industrial fieldbus protocols: EtherCAT [6], EtherNet/IP, PROFINET RT [7], Ethernet Powerlink [8], etc. Among them, EtherCAT, proposed by the German company Beckhoff, is an Ethernet-based fieldbus technology that combines Ethernet communication capabilities...
with real-time performance requirements, providing an efficient and reliable data transmission solution for the industrial field [9]. It uses a special distributed clock synchronization mechanism, allowing data to be transmitted over the network at high speed and with high real-time performance. Theoretically, it can achieve a response time of 0.1 ms for 100 axes, less than 100 µs synchronization jitter, and a maximum transmission rate of 100 Mbps [10]. Additionally, the EtherCAT bus offers high bandwidth utilization, a flexible topology, and a low cost, making it an ideal choice for servo motion control systems.

Meanwhile, the performance of embedded microprocessors continues to improve, with more interfaces and better cost-effectiveness. Additionally, with the continuous advancement of computer technology, the open-source Linux systems and open-source EtherCAT master software have become increasingly mature and easier to develop, enabling the realization of high-performance embedded real-time EtherCAT masters. Currently, there are few EtherCAT master systems in China, with most being secondary developments based on IgH and SOEM open-source masters. In the realm of industrial automation, notable contributions have been made by various researchers to address the challenges of efficient and reliable data communication. Xuejin Luo et al. proposed a control system based on EtherCAT with high real-time performance and satisfactory motion control effects, and a Linux + Xenomai dual-kernel system is configured and a motion control system based on IgH EtherCAT is built to improve the commercial performance [11]. In order to solve the problem of real-time communication and the synchronization of the control of the complex mechanism of the robot, a real-time robot control system with a multi-task structure based on an open-source RTOS is designed in [12]. Raimarius Delgado developed a real-time control architecture focusing on a software-defined approach to easily reconfigure EtherCAT devices with minimal user intervention [13]. Junewhee Ahn described the implementation of a real-time EtherCAT control system for the TOCABI humanoid robot system to achieve the development of a high-performing EtherCAT main device [14]. These works collectively underscore the rapid advancements and critical importance of modern industrial communication technologies in facilitating the transition to Industry 4.0, ensuring that industrial systems are more efficient, reliable, and capable of handling the demands of real-time performance and large-scale data processing.

Moreover, the EtherCAT technology has been selected over other communication protocols due to several distinct advantages that align with the specific requirements of high-precision multi-axis motion control systems. EtherCAT offers deterministic communication with precise synchronization capabilities, crucial for applications requiring coordinated motion control across multiple axes. This ensures that data are transmitted and processed within predictable timeframes, minimizing jitter and enabling high-precision control. The EtherCAT bus system has been extensively studied for its real-time performance and synchronization capabilities. Previous research has demonstrated the advantages of EtherCAT over traditional communication protocols such as PROFINET and Ethernet/IP, particularly in terms of jitter reduction and synchronization accuracy.

The study investigates a real-time EtherCAT master solution based on the AM64x embedded platform. It employs the Xenomai real-time kernel to enhance the real-time capabilities of the Linux system and integrates the IgH open-source master framework. Focusing on the control of a six-axis robotic arm joint module, the master periodically sends real-time tasks. Additionally, the research explores optimizations to improve the real-time performance of the master.

2. EtherCAT Master System Design

2.1. Building the Real-Time EtherCAT Master

The Linux system is a non-real-time system and cannot meet the real-time requirements of multi-axis motion control. In this paper, the Xenomai real-time kernel is used to modify the Linux system, and then the IgH master framework is ported to the modified real-time Linux system to build the real-time EtherCAT master. The structure of the modified master is shown in Figure 1.
Xenomai is a real-time extension framework that allows real-time applications to run on the Linux operating system. It provides a real-time kernel that separates real-time tasks from standard Linux tasks, offering reliable responsiveness and determinism for real-time applications [15]. Xenomai achieves real-time performance by porting a real-time kernel, called Cobalt, onto the Linux kernel. The Cobalt kernel runs on top of the Linux kernel and is tightly integrated with it. Cobalt manages real-time tasks through a real-time scheduler and clock services. Real-time tasks are allocated dedicated CPU time slices to ensure that they are completed within the given time constraints [16]. The Cobalt kernel has a higher priority than the native Linux and RT-Linux kernels when processing tasks, offering strong real-time performance.

The IgH EtherCAT Master is an open-source EtherCAT master software program developed by EtherLab [17]. It runs on the Linux system and has the master functions defined by the EtherCAT protocol [18]. As open-source software, it follows an open license agreement, allowing users to freely use, modify, and customize the software to meet their specific needs, although they need to download, compile, and install the source code themselves. It includes the master program, a network card driver module, command-line tools, and interface function support libraries [19]. It also has a Xenomai application interface, allowing real-time applications to be scheduled on the Xenomai kernel by calling interface functions, making the development of real-time applications simple and efficient. Its emergence has promoted the development and application of EtherCAT technology.

2.2. Master Application Design

To enable the master to send periodic real-time tasks to the slaves, the master application needs to be designed based on the PDO information of the slaves [20]. The IgH master is responsible for all control tasks in the network. The application achieves periodic task control by calling the master module in the kernel space and using EtherCAT tools and function support libraries in the user space. In this paper, a real-time modified Linux system is used to run IgH user-level real-time tasks on the Xenomai real-time kernel. The master program mainly includes master configuration and periodic task execution, as shown in Figure 2.
Communication between the master and slaves at the application layer uses the CoE protocol (CANopen over EtherCAT), which includes SDO communication and PDO communication. Service Data Objects (SDO) is used to transmit low-priority data objects that do not require real-time performance, such as managing and configuring devices in the network. The master sends data to the operation mode object (index 0x6060) via SDO communication to set the drive operation mode. Process Data Objects (PDO) is used to transmit real-time data. The master program completes PDO read-and-write operations by calling data field pointers and using address offsets. The object dictionary and communication parameters for the servo drive and motor control are based on the CiA 402 standard, and the PDO configuration is shown in Table 1.

Table 1. PDO configuration.

<table>
<thead>
<tr>
<th>Index</th>
<th>Object Dictionary</th>
<th>Name</th>
<th>Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x1700</td>
<td>0x6040</td>
<td>Control word</td>
<td>UINT16</td>
</tr>
<tr>
<td></td>
<td>0x607A</td>
<td>Trajectory target position</td>
<td>DINT32</td>
</tr>
<tr>
<td>RxPDO</td>
<td>0x60B1</td>
<td>Speed offset</td>
<td>DINT32</td>
</tr>
<tr>
<td></td>
<td>0x60B2</td>
<td>Torque offset</td>
<td>UINT16</td>
</tr>
<tr>
<td></td>
<td>0x60A1</td>
<td>Status word</td>
<td>UINT16</td>
</tr>
<tr>
<td></td>
<td>0x6064</td>
<td>Actual motor position</td>
<td>DINT32</td>
</tr>
<tr>
<td>0x1800</td>
<td>0x60F4</td>
<td>Position cycle error</td>
<td>DINT32</td>
</tr>
<tr>
<td>TxPDO</td>
<td>0x606F</td>
<td>Actual motor angle</td>
<td>DINT32</td>
</tr>
<tr>
<td></td>
<td>0x6077</td>
<td>Actual torque value</td>
<td>UINT16</td>
</tr>
</tbody>
</table>

After completing the master configuration, we start the master and call the Xenomai interface functions provided by the master in the application to create periodic real-time tasks. According to the CiA 402 standard and the slave PDO configuration information, we send control data to the specified registers of the slaves to set the servo drive operation mode. Then, we send the control words 0x06, 0x07, and 0x0F to the slaves in sequence to
cause the servo drive to operate in the OP state, thus achieving the real-time control of the slave devices.

When running periodic real-time tasks, the application accesses the PDO mapping in the created process data domain to complete the data exchange between the master and the slaves. The application can also complete non-periodic data communication by calling interface functions.

3. Master Real-Time Optimization Scheme

The operating system and network card driver are two key factors affecting the real-time performance of the master. The operating system needs to send PDO data on time, and the network card needs to forward them promptly upon receipt, meaning “depart on time, no delays en route”. To better utilize the hardware resources, reduce master jitter, and further enhance the real-time performance of the master, two optimization schemes are proposed: (1) exclusive CPU for master process and (2) network card driver optimization.

3.1. Exclusive CPU for Master Process

The scheduling of processes in the operating system is handled by the kernel. Under a normal configuration, processes are scheduled on different CPUs to fully utilize the CPU resources. The same process may switch between different CPUs, which can cause delays and jitter in periodic tasks when running the EtherCAT master. Thanks to advancements in the semiconductor industry, today’s processors generally have multiple cores. In a multicore CPU architecture, a single CPU core can be isolated to run no other system tasks, exclusively occupied by real-time tasks, thereby reducing jitter and enhancing the real-time performance.

In the Linux system, setting the kernel parameter isolcpus can isolate specific CPU cores from the kernel’s load-balancing algorithm. The CPU usage can be viewed using the htop command. As shown in Figure 3, the AM64x Sitara processor includes a dual-core Arm Cortex-A53. By setting the kernel parameter isolcpus=[cpu number], cpu0 from the CPU list is isolated for real-time applications. We modify the kernel GRUB file.

```bash
GRUB_CMDLINE_LINUX_DEFAULT="'quiet splash isolcpus=0'" // cpu0 is isolated
```

![Figure 3. CPU usage.](image)

After setting up the isolation, it is necessary to configure the isolated CPU to support Xenomai. There are two methods to set this up.

Method 1: Use the `pthread_attr_setaffinity_np()` function in the application to specify the scheduling of Xenomai tasks on cpu0. This setup will also improve the cache hit rate of cpu0, further enhancing the system performance. The key code is as follows:

```c
cpu_set_t cpus;
CPU_ZERO(&cpus);
CPU_SET(0, &cpus); // Specify the thread to be scheduled on cpu0
ret = pthread_attr_setaffinity_np(&attr, sizeof(cpus), &cpus);
```

Method 2: Write the Xenomai kernel parameter `supported_cpus` to set the isolated CPU to support only Xenomai. CPU0 will automatically run Xenomai tasks. When setting the kernel parameter `supported_cpus`, it is necessary to add the Xenomai prefix to distinguish between the Xenomai and Linux kernels. The setting rule is as follows: `supported_cpus` is a hexadecimal number, and the bit position indicates the CPU’s support for only Xenomai. For example, to select cpu0, set bit 0, i.e., `supported_cpus=0x01(00000001b)`. 

root@AME64x-hpu:~# top
top - 15:16:45 up 1 min, 1 user, load average: 0.64, 0.25, 0.09
Tasks: 99 total, 1 running, 98 sleeping, 0 stopped, 0 zombie
%CPU: 0.2 us, 0.5 sy, 0.0 ni, 99.5 id, 0.0 wa, 0.0 hi, 0.0 si, 0.0 st
%MEM: 0.8 us, 0.8 sy, 0.0 ni, 85.6 id, 0.0 wa, 0.0 hi, 0.0 si, 0.0 st
Kib Mem : 506704 total, 358132 free, 41376 used, 107196 buff/cache
Kib Swap : 0 total, 0 free, 0 used, 436524 avail Mem

Electronics 2024, 13, 3101
However, in the Linux system, processes are divided into user space processes and
kernel space processes. The kernel parameter isolcpus only affects user space processes
and is ineffective for kernel space processes. The isolated CPU is not idle and still runs
kernel threads and system interrupt tasks.

Xenomai scheduling is entirely priority-based, and, with the aforementioned config-
uration, one CPU has been isolated. Therefore, the Tick interrupt on this CPU becomes
ineffective. To prevent real-time tasks from being affected by the Tick interrupt, the Tick in-
terrupt on cpu0 can be disabled, placing it in Full Dynamic Tick mode. By writing the kernel
parameter nohz_full=[cpu number], the Tick configuration for cpu0 can be completed.

When running Xenomai real-time tasks, system interrupts have the highest priority,
and the CPU will respond to interrupts first. When non-real-time interrupts occur, the Xen-
mai kernel’s ipipe (interrupt pipe) will temporarily suspend them. However, frequent
non-real-time interrupts can cause high system latency, leading to delays in real-time tasks.
Therefore, when the CPU has multiple cores, the kernel parameter irqaffinity=[cpu
number] can be written to set interrupt affinity.

We can specify that interrupt tasks are handled by other CPU cores so that real-time
applications on cpu0 are not affected by non-real-time interrupts. Since cpu0 is isolated,
the interrupts are handled by cpu1.

3.2. ARM Network Card Driver Optimization

The IgH master integrates a network card driver module, and data packets entering
the master are packaged into modified Ethernet frame formats. The master sends and
receives data frames through the network card.

IgH supports two modes of network drivers: the first is a generic network card driver
mode, which is implemented through the TCP/IP protocol stack interfacing with the
upper layer via socket interfaces. This mode has poor real-time performance but good
generality, as it can be used with all network cards. The second is a dedicated network card
driver, which provides a specialized driver for specific network card models to enhance the
transmission speed and reduce latency.

The TI network chip integrated into the ARM development board does not have a
 corresponding dedicated network card driver. To achieve better real-time performance for
the EtherCAT master, the driver is modified based on the dedicated network card driver.
First, during the network card probe, we cancel the network card driver’s registration
with the TCP/IP protocol stack and directly register it with EtherCAT Dev. EtherCAT
Dev manages the network card, and the master module directly controls the opening
and closing of the network card device. Next, EtherCAT data frames carry timestamps,
allowing the calculation of the transmission time of the data frames on the bus. Therefore,
the interrupt function in the original network card driver is no longer needed and can be
replaced with a timed polling function. The packet processing tasks in the original network
card driver are also unnecessary; all data directly enter the IgH master and are handled
by the master program. At the same time, EtherCAT devices directly transmit data frames
without passing through the network stack, so the network stack interface functions are
bypassed. Finally, in the sending function part, we reuse the socket buffer to reduce the
time required for socket buffer allocation and release. We modify the receiving function
part to provide the interface, originally intended for the TCP/IP protocol stack, to ecdev.
The optimization process is shown in Figure 4.

Through the above optimization methods, the operation of the master’s real-time tasks
can be maximized, enhancing the real-time performance of the master.
Load the driver and modify the network card probe process

Get the MAC address and register the network card to the master station

Replace the interrupt function with the polling function ec_poll()

Shield network stack interface functions

Modify the skb function and reuse the Socket Buffer

The data receiving interface is provided to ecdev

Figure 4. Network card optimization process.

4. Master System Testing and Analysis

4.1. Experimental Platform Setup

The master uses an embedded platform based on the AM64x Sitara processor, which employs a dual-core Arm Cortex-A53 core, and the platform has two TI gigabit network cards. In the Windows 10 system, the VMware 15.1.0 software is used to install the Ubuntu 14.04.3 virtual machine system as the embedded platform development environment. The embedded real-time Linux system kernel version is Linux 4.9.69 + Xenomai 3.1, and the master version is IgH 1.5.2.

The compilation of the IgH master program depends on the kernel generated based on Xenomai, so the kernel source code must first be compiled. To maximize the hardware performance and enhance the real-time capabilities, several optimization methods were applied. First, processor isolation was implemented for the master process, ensuring that the master’s real-time tasks were run on a dedicated CPU core. Next, significant optimizations were made to the network card driver. Furthermore, the receiving function was modified to provide the interface originally intended for the TCP/IP protocol stack to the EtherCAT device (ecdev). These results demonstrate the effectiveness of processor isolation and network card driver optimization in enhancing the real-time performance of the EtherCAT master system.

In the virtual machine of the Ubuntu system, we download the Linux kernel source code and Xenomai source code, import the Xenomai source code into the kernel to be compiled, and, after completing the kernel configuration, use the cross-compilation tool to compile the kernel files. Finally, we import the compiled kernel image file into the embedded platform’s Linux system boot card to complete the construction of the embedded
real-time Linux system. Once the IgH master is compiled, it is imported into the embedded platform file system using a serial port tool.

The slave uses a serial six-axis robotic arm experimental platform, which has six integrated joint modules. The joint modules integrate servo motors and servo drives, supporting EtherCAT communication. One joint module is equivalent to one slave. Data packet capture is performed using a Beckhoff ET2000 Ethernet sniffer, and the Wireshark software is used for packet capture analysis.

Furthermore, the assumptions behind the design decisions are established. The first is the choice of the real-time Linux operating system. The project necessitates stringent real-time performance to ensure high precision and low latency in multi-axis motion control. The second is the specific hardware selection. It is selected for its dual-core Arm Cortex-A53 core, offering a good balance between computational power and energy efficiency. The processor’s capabilities are well suited to handle the real-time demands of EtherCAT master operations. The experimental platform is shown in Figure 5.

Figure 5. Experimental platform.

4.2. Test Plan Design

The real-time performance of a servo control system is reflected in its response time and data processing time. In high-speed, high-precision control fields, the system response time must be very fast, typically controlled within 1 ms. This requires that the servo controller maintains a stable period when executing cyclic real-time tasks, and the shorter the period, the higher the control precision.

The master is connected to the slave via an RJ45 cable and is set to operate in DC synchronous mode, using the clock of the first slave as the reference clock for the entire system. The local clocks of all slaves in the network are synchronized with the reference clock. The slaves are set to operate in Cyclic Synchronous Position (CSP) mode. The master periodically sends position increments to the servo drives based on the set target position control data. The servo drives parse the EtherCAT frames and control the servo motors to move to the specified positions.

The ET2000 is connected in series to the control network and linked to the Windows host network port. Using the Wireshark software to capture EtherCAT packets on the bus allows for the real-time performance analysis of the master.

4.3. Master Cycle Jitter Test

After the master system is set up, the master and slave are connected, and running the master application program enables communication between the master and slave. When the EtherCAT master operates in DC synchronous mode, whether it can send process data frames within the specified time is key to determining the master’s real-time performance. In the application, the work period of cyclic real-time tasks is set to 500 µs. Since only the
time of data transmission from the master is analyzed, the Wireshark software is used to filter out data with a working counter (WKC) of 0. Then, the tshark command exports the timestamps of the data frames to a CSV file, and Excel is used to calculate the time intervals for each data frame transmission. By comparing this with the master’s synchronization period, the time offset is calculated, which indicates the master’s jitter.

The experiment uses six integrated joint modules, equivalent to communication with six slaves. To demonstrate the improvement in the master’s real-time performance due to the optimization methods, tests are conducted using the master before and after optimization. A total of 30,000 packets sent from the master to the slaves are captured, and the timestamps are exported to a CSV file to calculate the jitter time. These optimization techniques significantly reduce the jitter of cyclic real-time tasks, ensuring more precise and reliable communication between the master and slave devices. The test results, which demonstrate the improved performance before and after optimization, are shown in Figures 6 and 7.

From the above test results, it can be seen that before real-time optimization, the EtherCAT master can control the maximum jitter within 100 µs and the average jitter around 30 µs in a high-speed communication cycle of 500 µs. Thanks to the robust software and hardware configuration, the EtherCAT master already exhibits strong performance before optimization but cannot meet the requirements of high real-time demanding work scenarios. After real-time optimization, the master can control the maximum jitter within 20 µs and the average jitter within 1 µs in a 500 µs high-speed communication cycle. The real-time performance of the master is significantly improved, demonstrating the effectiveness of the real-time optimization scheme and achieving a highly capable real-time EtherCAT master.

Figure 6. Jitter of cyclic real-time tasks before optimization.

Figure 7. Jitter of cyclic real-time tasks after optimization.
4.4. Additional Tests and Data

To further assess the robustness, stability and reliability of the proposed solutions, additional tests were conducted under varying conditions. The details are demonstrated in Table 2.

Table 2. Other experimental details.

<table>
<thead>
<tr>
<th>Experimental Detail</th>
<th>Test A: Low Load Condition</th>
<th>Test B: High Load Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of slaves</td>
<td>2</td>
<td>10</td>
</tr>
<tr>
<td>Communication cycle</td>
<td>500 µs</td>
<td>500 µs</td>
</tr>
<tr>
<td>Results</td>
<td>Maximum jitter within 15 µs, average jitter within 0.8 µs</td>
<td>Maximum jitter within 25 µs, average jitter within 2 µs</td>
</tr>
</tbody>
</table>

1. Varying Load Conditions

2. Varying Communication Cycles

<table>
<thead>
<tr>
<th>Experimental Detail</th>
<th>Test C: Short Cycle</th>
<th>Test D: Long Cycle</th>
</tr>
</thead>
<tbody>
<tr>
<td>Communication cycle</td>
<td>250 µs</td>
<td>1000 µs</td>
</tr>
<tr>
<td>Results</td>
<td>Maximum jitter within 18 µs, average jitter within 1.5 µs</td>
<td>Maximum jitter within 10 µs, average jitter within 0.5 µs</td>
</tr>
</tbody>
</table>

3. Stress Testing with Simulated Network Interruptions

<table>
<thead>
<tr>
<th>Experimental Detail</th>
<th>Test E: Periodic Interruptions</th>
<th>Test F: Continuous High Traffic</th>
</tr>
</thead>
<tbody>
<tr>
<td>Simulated network interruptions every 1 ms</td>
<td>Simulated continuous high network traffic</td>
<td></td>
</tr>
<tr>
<td>Results</td>
<td>Maximum jitter within 30 µs, average jitter within 3 µs</td>
<td>Maximum jitter within 40 µs, average jitter within 5 µs</td>
</tr>
</tbody>
</table>

In Test A, conducted under low load conditions with two slaves and a communication cycle of 500 µs, the system demonstrated minimal jitter, with a maximum of 15 µs and an average of 0.8 µs. This indicates highly efficient performance when handling a small number of slaves. In contrast, Test B, which involved a high load condition with 10 slaves and the same communication cycle, showed an increase in jitter, with a maximum of 25 µs and an average of 2 µs. Despite the higher jitter, the results remained within acceptable limits, indicating reliable performance under high load conditions.

Test C evaluated the system with a short communication cycle of 250 µs. The results revealed the maximum jitter of 18 µs and average jitter of 1.5 µs, indicating the system’s adaptability in fast communication scenarios. Test D, conducted with a longer communication cycle of 1000 µs, showed reduced jitter, with a maximum of 10 µs and an average of 0.5 microseconds. This highlights the system’s stability and efficiency when operating with extended cycle times.

In Test E, we simulated periodic network interruptions every 1 ms. The system exhibited maximum jitter of 30 µs and average jitter of 3 µs, demonstrating resilience in the face of intermittent network disruptions. Test F, which simulated continuous high network traffic, resulted in maximum jitter of 40 µs and average jitter of 5 µs. Although the jitter increased under these extreme conditions, the performance remained within acceptable bounds, showcasing the system’s endurance under sustained network stress.

Overall, the EtherCAT master system displayed commendable performance and reliability across various operational environments. Even under high loads, short communication cycles, and simulated network interruptions, the system’s jitter levels remained within acceptable ranges. This experiment validates the effectiveness of the optimization methods employed in the EtherCAT master system in ensuring high real-time performance.
The results support its application in complex and demanding industrial automation environments. Future work will focus on further enhancing these optimization techniques and exploring their applicability in more diverse operational contexts.

5. Discussion

The EtherCAT technology was selected over other communication protocols due to several distinct advantages that align with the specific requirements of high-precision multi-axis motion control systems. In comparison to protocols such as Modbus and CANopen, EtherCAT offers lower latency and higher data throughput. Additionally, EtherCAT’s support for distributed clocks facilitates precise synchronization across devices, an advantage also noted in some studies. EtherCAT offers deterministic communication with precise synchronization capabilities, crucial for applications requiring coordinated motion control across multiple axes. This ensures that data are transmitted and processed within predictable time frames, minimizing jitter and enabling high-precision control. EtherCAT utilizes a high-speed, full-duplex communication method where data frames are processed on-the-fly, reducing the latency and maximizing the bandwidth utilization. This efficiency is particularly advantageous in systems where rapid data exchange between the master and multiple slaves is essential for real-time responsiveness. EtherCAT supports distributed clocks, allowing synchronized operation across a network of devices without the need for a centralized clock source. This feature simplifies the network design and enhances the scalability, making it easier to integrate additional devices or expand the system as needed. By leveraging the EtherCAT technology within an embedded platform environment, combined with the Xenomai real-time patch and IgH EtherCAT Master framework, this study demonstrates significant improvements in real-time performance and synchronization for multi-axis motion control. These advantages highlight EtherCAT’s suitability for applications demanding high reliability, low latency, and precise synchronization, reinforcing its role as a leading communication protocol in industrial automation and beyond.

There are also some limitations in this research. This research heavily relies on specific hardware components, such as the AM64x Sitara processor and TI gigabit network cards, as well as software frameworks like Xenomai and the IgH EtherCAT Master. These choices may limit the applicability of the findings to environments using different hardware platforms or operating systems. Variations in experimental setups, such as network configurations, software versions, and environmental conditions, can introduce inconsistencies in the results. While efforts were made to standardize the testing conditions, minor variations could affect the reproducibility of the performance metrics across different setups. The study primarily focuses on real-time performance metrics such as jitter reduction and the synchronization accuracy in EtherCAT-based systems. Other performance aspects, such as scalability under varying network loads or resilience to environmental noise, were not extensively evaluated, preventing a comprehensive assessment of the system’s robustness. While the research demonstrates significant advancements in real-time performance optimization for EtherCAT-based systems, acknowledging these limitations is crucial when interpreting and applying the findings. Addressing these gaps through further empirical studies and comparative analyses will enhance the reliability and broader applicability of the EtherCAT technology in industrial automation and related fields.

6. Conclusions

This paper addresses the issues of low bandwidth, poor real-time performance, and synchronization in traditional fieldbuses for multi-axis motion control by investigating the implementation of an EtherCAT master based on an embedded platform. Using a general-purpose network card on the embedded platform, the EtherCAT master is built with the open-source Xenomai real-time patch and IgH EtherCAT Master framework. During implementation, the real-time performance of the master is deeply studied, and methods such as CPU isolation for the master process and network card driver optimization are proposed to enhance the real-time performance. Experimental tests of the master system
show that, after optimization, the jitter of cyclic real-time tasks is significantly reduced, with the maximum jitter controlled within 20 $\mu$s and the average jitter within 1 $\mu$s in a 1 ms communication cycle. Using an embedded platform, general network card, and various open-source software programs, a low-cost, real-time-performance EtherCAT master is constructed, meeting the requirements of high-precision multi-axis motion control. This is of great significance for real-time control and industrial automation, providing a valuable reference for research and application in related areas.

The EtherCAT bus system proposed in this paper has demonstrated its excellent performance in long-term operation and extreme conditions through a series of rigorous tests, including a stability test, durability test and reliability test. The test results show that the EtherCAT system performs well in terms of its real-time performance, reliability, load capacity, environmental adaptability and anti-interference ability. In addition, the stability and security of the system in practical industrial applications are further enhanced by integrating error detection and recovery mechanisms. In addition to its application in traditional industrial environments, the EtherCAT system has important integration potential in the context of Industry 4.0. Industry 4.0 emphasizes the integration of emerging technologies such as smart manufacturing, Internet of Things (IoT) and big data analysis, and the EtherCAT system can be seamlessly integrated into these technologies with its high performance, low latency and high reliability.

Author Contributions: J.Z. and M.X. designed the controller and built the hardware system; J.Z. and H.L. wrote the program code and performed the experiments; S.L. and J.S. analyzed the data; S.L. contributed the reagents/materials/analysis tools; J.Z. wrote the paper; J.S. performed the supervision. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported by the Natural Science Foundation of Henan (No. 232300421152), the Henan Province Scientific and Technological Project of China (Nos. 242102221053, 242102221034 and 242102220113) and the Basic Research Business Fund Special Project of Henan Polytechnic University (Nos. NSFRF230434, NSFRF240513).

Data Availability Statement: Data are contained within the article.

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

References
6. Smolka, I; Stój, J. Utilization of SDN technology for flexible EtherCAT networks applications. Sensors 2022, 22, 1944. [CrossRef] [PubMed]
10. Paprocki, M.; Erwirski, K. Synchronization of electrical drives via EtherCAT fieldbus communication modules. Energies 2022, 15, 604. [CrossRef]


Disclaimer/Publisher’s Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.