Proceedings of 2019 IEEE
International Conference on Mechatronics and Automation
August 4 - 7, Tianjin, China

Design of a Dual-core Processor Based Controller with RTOS-GPOS Dual Operating System

Yuansong Sun[1,3], En Li[1], Guodong Yang[1], Zize Liang[1], Rui Guo[2]

1. The State Key Laboratory of Management and Control for Complex Systems, Institute of Automation Chinese Academy of Sciences, Beijing 100190, China
2. State Grid Shandong Electric Power Company, Jinan 250001, China
3. The University of Chinese Academy of Sciences, Beijing 100049, China
{sunyuansong2017, en.li, guodong.yang, zize.liang}@ia.ac.cn & guoruihit@qq.com

***Abstract* - The controller is one of the key components of the robot system. The general-purpose controller usually uses multi-processor scheme with master-slave structure or distributed structure. This kind of scheme is complex and high-cost. It is difficult to adapt to the design requirements of embedded miniaturized robots. In view of this situation, this paper proposes a controller design based on dual-core dual-operating system, solving various real-time and non-real-time tasks in robot control through a single chip. According to the design requirements, a dual-core chip ZYNQ works as the core of the controller hardware platform. Linux and FreeRTOS are transplanted as master-slave system on different core, and the asymmetric multi-processing (AMP) architecture is introduced as the working framework of the dual-core processor. After that, the key technologies of ZYNQ running under the AMP architecture are analyzed, including the allocation of shared resources, the dual-core booting mode and the dual-core communication mode. Finally, using the open source architecture OpenAMP to transplant the Linux+FreeRTOS dual system and achieve parallel running. The communication capability and real-time performance of the dual system are tested, and the results verify the effectiveness of the controller.**

Index Terms - robot controller; embedded system; GPOS; RTOS; AMP;

I. INTRODUCTION

The robot is a typical integrated system with mechatronics and control technologies. The past robot controller research was usually aimed at specific problems. However, the robot controller software includes non-real-time tasks such as human-computer interaction and path planning, as well as real-time tasks such as motion control and emergency operations. The past robot controller has poor reusability, long development cycle, and high cost. They can't balance different types of tasks. In order to solve this problem, some present robot controllers use a multi-processor architecture to achieve both versatility and real-time performance, but this results in higher cost and lower communication efficiency. To enhance the versatility of robot controllers, some robotics teams had proposed modular robot controller designs. A modular controller based on ARM+FPGA was one of the options. This controller was characterized by modularity and quick configuration [1]. This solution made full use of ARM's high-performance computing ability and FPGA's fast configuration ability by using ARM as the upper controller and FPGA as the lower interface. At the same time, in order to control the robot in real time, the real-time operating system was running on ARM. This design improves the reusability of the hardware platform and shortens the development time. But the disadvantage is that there is no special communication path between ARM and FPGA, which easily affects the overall performance improvement. FPGA will lead to an increase in overall cost while its hardware acceleration characteristics cannot be exerted. The Beijing University of Aeronautics and Astronautics proposed a modular controller architecture based on dual-core dual operating systems [2]. They formed a hybrid robot operating system with RTOS and GPOS. The GPOS contains non-real-time ROS code that runs on Linux, while the real-time operating system contains real-time ROS code running on Nuttx. The hybrid ROS runs on a dual-core x86 architecture chip. GPOS and RTOS run respectively on different CPU. This hybrid operating system was used in controllers for six-degree-of-freedom modular manipulators, and its performance was demonstrated by software testing experiments [3][4]. The scheme can balance both versatility and real-time performance, but the system is complex and difficult to implement. In summary, the research on robot controllers has achieved a lot of results. However, current research lacks a concise, lightweight and low-cost solution that combines versatility and real-time.

In this paper, a robot controller scheme that can balance the versatility and real-time of the controller is proposed. This paper designs a controller hardware platform based on the dual-core chip, which enables GPOS and RTOS to run on two processors in parallel. The RTOS guarantees the real-time requirements in robot control and the GPOS makes full use of software resources. It enables the controller to perform both general and real-time tasks simultaneously. The hardware of the controller is light and low-cost, and the system architecture is concise, which has certain advantages compared with the existing controller scheme.

II. CONTROLLER SCHEME

A. Controller System Structure

The dual operating system robot controller is designed to enable two operating systems to run in parallel on one controller chip, making full use of the structural advantages of

978-1-7281-1699-0/19/$31.00 ©2019 IEEE

the multi-core processor to achieve complementary functions of the two systems. The structure of the controller is shown in Fig. 1.

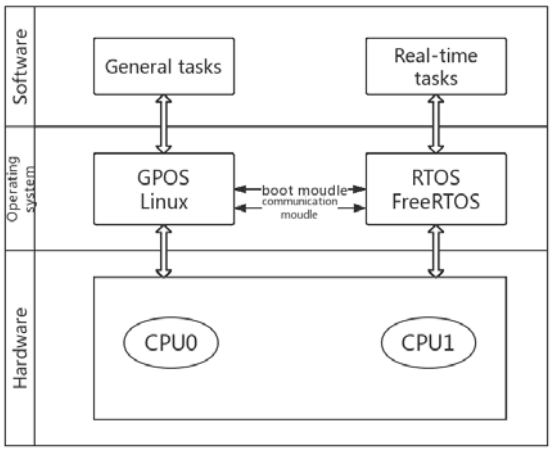


Fig. 1 Dual operating system robot controller system structure.

At the hardware layer, it requires a dual-core or multi-core processor that can run two different operating systems in parallel and support inter-core communication.

At the operating system layer, it requires suitable GPOS and RTOS. They need to be portable and reliable. At the same time, they need to be able to run on our chosen hardware platform, which means they have a board support package that supports the hardware platform.

At the application software layer, since there are no specific control objects by now, the current application software layer is used to test the controller.

B. Controller Hardware Design

According to the requirements, GPOS and RTOS should run in parallel and be able to communicate. Therefore, the hardware platform needs to meet the following conditions: dual-core or multi-core, support for heterogeneous systems, communication channels between multiple cores and enough memory space

After comparing the performance of the current mainstream embedded processor chips, combined with the design requirements of the robot controller, the ZYNQ-7000 AP Soc from Xilinx is selected, and the controller hardware platform based on ZYNQ-7020 chip is designed [5]. The ZYNQ chip integrates a dual Cortex-A9 ARM core and FPGA to form a fully programmable chip. Each CPU has separate memory management units and caches that allow them to run independently. The AXI bus is used in the dual-core processor to enable dual-core communication with low latency. In general, the hardware architecture of ZYNQ is very suitable for this design purpose. The dual-core processors in PS have their own set of basic devices, which makes sure they can work together and run independently. It is suitable for the heterogeneous multi-core architecture.

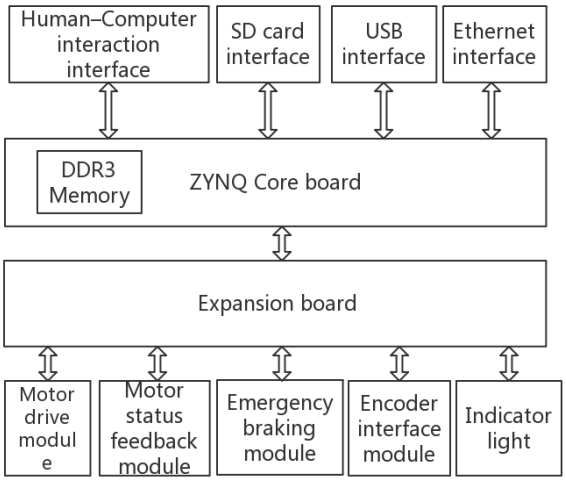
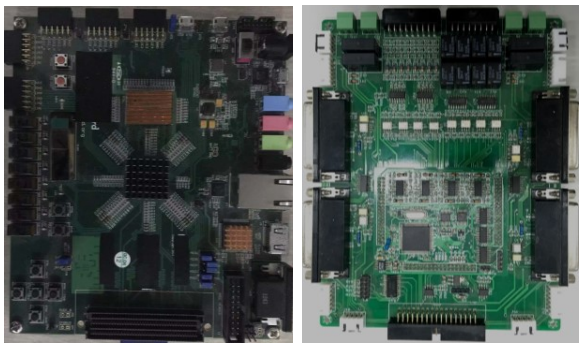


Fig. 2 Hardware structure.



(a) (b)

Fig. 3 Hardware platform. (a) kernel board, (b) expansion board.

The controller hardware platform consists of the kernel board and the expansion board. The core board is based on the ZYNQ processor, and the external parts include DDR3 memory, human-computer interaction interface module, SD card interface, USB interface, and Ethernet interface. The DDR memory will be used as shared memory for dual-core communication. The kernel board is the core of this design. The dual system will be built and run on the kernel board. It is also responsible for interacting with the host computer. The resources on expansion board include motor drive module, motor status feedback module, emergency braking module, encoder interface module, and indicator light. The expansion board is used to interact with robot actuator and sensor. The hardware structure and material object are shown in Fig. 2 and Fig. 3.

The operating mode of multi-core processors has symmetric multiprocessing mode (SMP) and asymmetric multiprocessing mode (AMP) [6]. The AMP is characterized in that each core in a multi-core processor can be viewed independently. According to the requirements, AMP is selected as the operating mode of multi-core processors.

C. *Operating System Selection*

At present, the operating systems commonly used by robot controllers include general-purpose operating systems (GPOS) such as Windows, Linux, Android, Ubuntu, ROS (based on Ubuntu), and real-time operating systems (RTOS) such as μC/OS, FreeRTOS, and VxWorks. This design requires a GPOS system and an RTOS system. They need to be open source, tailorable, and portable.

The current GPOS includes Windows, Linux, Android, and so on. Among them, embedded Linux is the best choice from the perspective of easy portability and tailor ability. Linux is currently the most widely used embedded general-purpose operating system. Embedded Linux comes from a standard Linux system. It is widely used, and its reliability is fully verified.

Current RTOS includes μC/OS, VxWorks, FreeRTOS, and the like. In this design, the processor resources are limited, so we need to choose a small operating system. In addition, the Xilinx official IDE provides support for FreeRTOS, which greatly facilitates software development on real-time systems, so FreeRTOS is selected. FreeRTOS is an open source, lightweight real-time system. Its kernel consists of only three files, and the main part is used for task scheduling. FreeRTOS uses the task scheduler for task control and controls tasks according to their priority. Therefore, users need to set the priority of each task according to specific needs.

Therefore, Linux and FreeRTOS are selected as GPOS and RTOS respectively. The ZYNQ chip supports both systems, so they can be used in this design.

III. DUAL OPERATING SYSTEM DESIGN BASED ON OPENAMP

A. *OpenAMP Overview*

OpenAMP is an open source AMP framework developed by the Multicore Association (MCA). The current OpenAMP can support a variety of mainstream operating systems, including Linux, Nucleus, FreeRTOS, μC/OS, VxWorks and more.

OpenAMP provides three functional modules: Remoteproc, RPMsg, and virtIO. virtIO is a virtualized communication standard that provides a communication interface for upper layer software by virtualizing the slave device. Remoteproc is a software interface module used by the main processor to control the life cycle of the slave processor. RPMsg is a software interface module that provides communication channels for heterogeneous multi-core systems. Remoteproc and RPMsg are the software interface modules we mainly use, which are responsible for dual-core startup and communication. The key technologies of ZYNQ running under the OpenAMP architecture is analyzed next section.

B. *Key Technologies of AMP Architecture*

1) Dual-core shared resources: The main shared resources of the dual-core include 512KB L2 cache, 256KB on-chip memory (OCM), DDR memory, interrupt controller GIC, clock, etc. In our design, the L2 cache is used by the main CPU. The DDR memory is divided according to the size of the system on each CPU so that the two systems can only identify the fixed size area on the DDR, and a small area is divided into shared memory. On-chip memory OCM is used by dual cores.

2) Dual-core boot method: During the dual core boot process, the slave processor is booted under the control of the master processor. The Remoteproc module of OpenAMP provides a software interface to manage the remote processor's lifecycle for software applications running on the main processor. The boot process is shown in Fig. 4.

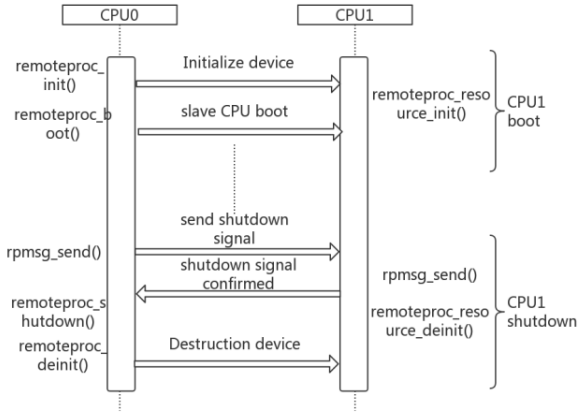


Fig. 4 Dual-core boot process under OpenAMP.

3) Dual-core communication method: Shared memory is used for communication in the AMP architecture. Dual cores have the right to read and write to the same memory area [7]. In this design, DDR is used as the shared memory and reading and writing interrupts are set as the inter-core communication mechanism. The communication method is shown in Fig. 5.

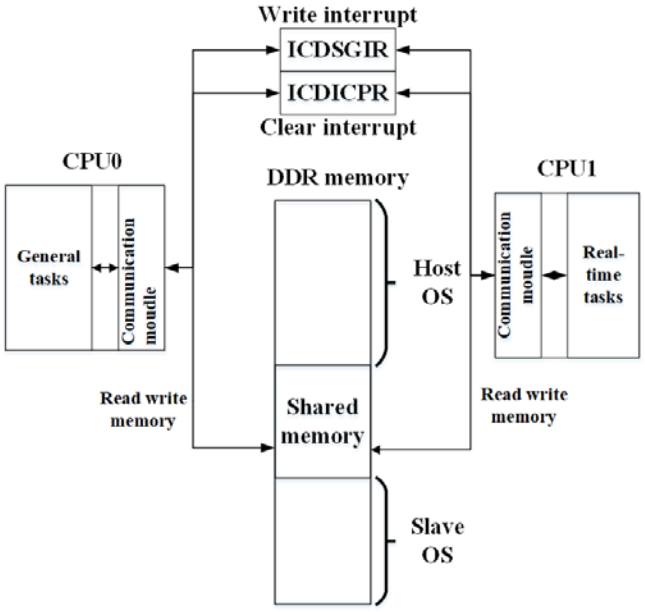


Fig. 5 Dual-core communication method under OpenAMP.

C. AMP Architecture for Linux+FreeRTOS

The Xilinx development environment supports OpenAMP. Petalinux is used to build the AMP architecture of Linux and FreeRTOS under the OpenAMP framework. Xilinx SDK is used to build communication software.

In this design, Linux is the host operating system running on CPU0 while FreeRTOS is the slave operating system running on CPU1. At the same time, in order to verify that the two systems can run in parallel and communicate, the verification software is needed. The host operating system Linux sends data to the slave operating system FreeRTOS. After receiving the data, FreeRTOS will send it back. Linux checks the sending and receiving data to verify whether the communication is valid. The modules provided by OpenAMP is used to complete the configuration of the system.

D. Software Design in Linux

The host operating system Linux runs on CPU0. When the new application is created in SDK, Linux and CPU0 are selected to be the environment and the processor respectively.

Applications can control devices by reading and writing files in Linux. The communication module RPMsg exists in the form of files in the Linux system. The communication between CPU0 and CPU1 is realized by reading and writing the file corresponding to the RPMsg module.

In this program, the open(rpmsg_dev) function is used to get the file identifier of the RPMsg module. The ioctl () function is used to make necessary configuration for the IO channel, such as the baud rate of the transmission. The malloc() function is used to allocate space for the payload of the data. The write(fd,i_payload) function and read(fd,r_payload) function are used to send and receive data respectively. At last, write(fd, shutdown_msg) is used to send a shutdown command to CPU1 to shut it down. The running flowchart of the software is shown in Fig. 6.

E. Software design in FreeRTOS

The slave operating system FreeRTOS runs on CPU1. When the new application is created in SDK, FreeRTOS and CPU1 are selected to be the environment and the processor respectively.

After the application is created, the processor precompilation constant needs to be added to activate the configuration of CPU1 in AMP architecture and the OpenAMP library needs to be added.

The application in FreeRTOS exists in the form of tasks. The task scheduler controls all tasks according to the priority of each task, and only one task can be run at the same time. In this program, the task needs to constantly detect whether Linux has passed data and if so, return the data. The remoteproc_resource_init() function is used to initialize the slave device, including the RPMsg module. The hil_poll() function is used to get the data from CPU0. The rpmsg_send() function is used to send the data back to CPU0. This task will keep running until the data received by FreeRTOS is the CPU1 shutdown signal. Then remoteproc_resource_init() function is used to release the resource and vTaskDelete() function is used to end the running of this task in FreeRTOS. The running flowchart of the software is shown in Fig. 7.

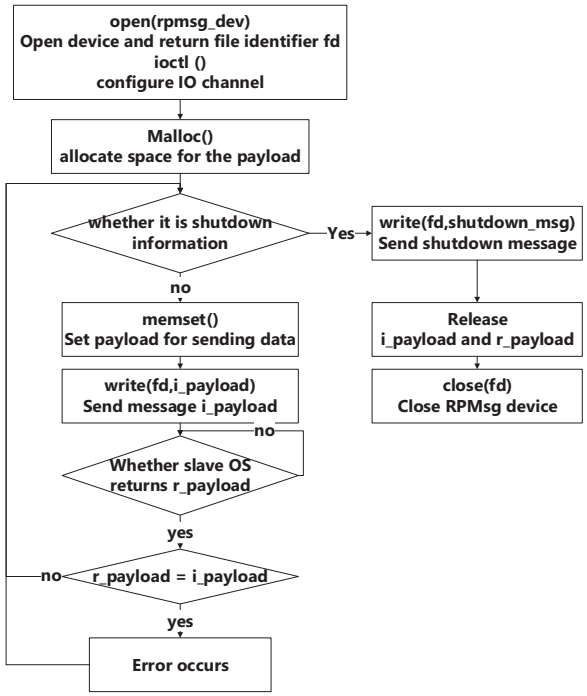


Fig. 6 Flowchart of communication software in Linux.

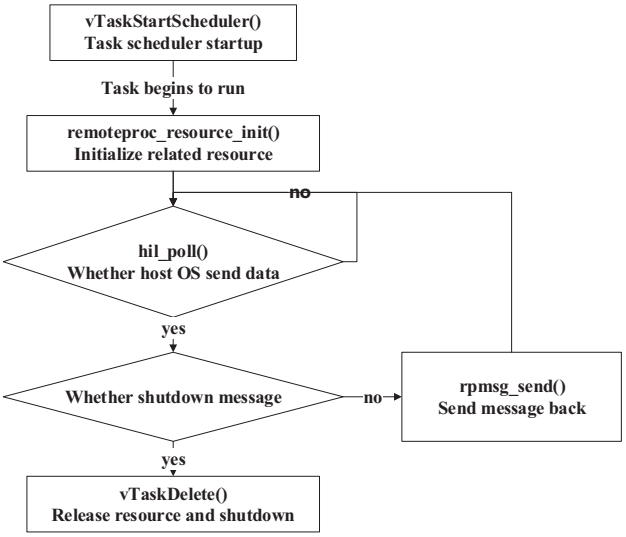


Fig. 7 Flowchart of communication software in FreeRTOS.

The communication software of the two systems is developed using the Xilinx SDK and then the operating system is configured using the Xilinx Petalinux.

F. Configuration of The Operating System

Xilinx Petalinux is used to build the AMP architecture, which includes the following steps.

(1) Establish the petalinux project as the host system Linux;

(2) Import the communication software, including the communication software running in Linux, the system firmware of FreeRTOS and its communication software. The compilation script needs to be modified accordingly.;

(3) Configure OpenAMP, including adding function modules, memory allocation, kernel boot address, and adding shared memory nodes in the device tree.

At this point, the construction of the dual system controller is completed.

IV. Performance Test

A. CPU Occupancy Test

In order to verify whether the two CPUs are running in the AMP state, the CPU occupancy test is needed.

Before starting the AMP, enter the cat command in the terminal to check the CPU status. Both CPUs are occupied by Linux system in SMP state. After starting the AMP, enter the command again and only CPU0 is occupied Linux. The results are shown in Table I. It can be verified that the CPU can run in the AMP state.

TABLE I
CPU OCCUPANCY TEST

Occupation status	CPU0	CPU1
SMP	Running	Running
AMP	Running	None

B. Dual-Core Communication Test

The test performs multiple data exchanges between the host and slave processors. If there is no data transfer error in the test, it can be verified that the AMP architecture of Linux+FreeRTOS is working properly and that there is efficient communication between the two processors. Each round of communication time is obtained by adding timing function to the communication software. The test shows that the communication time of each round increases as the size of the transmitted data increased. The communication time and accuracy curve are shown in Fig. 8. The measurement data is shown in Table II.

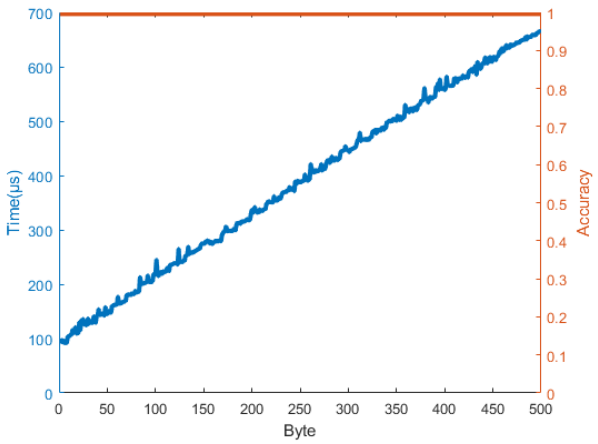


Fig. 8 Communication time and accuracy curve.

TABLE II
COMMUNICATION TIME

Maximum time(μs)	Minimum time(μs)	Average time(μs)
91.322	665.635	349.169

Through several repeated experiments, the data transmission can run reliably without error. It can be verified that the dual system architecture can operate and communicate effectively on dual-core processors.

C. Response Time Test

Rhealstone is a standard for testing the real-time performance of an operating system [8]. The system is tested by this standard below.

1) Task switching time test: The task switching time is the time required for the system to switch between two separate tasks that are in the ready state and have the same priority.

The test program creates two tasks with equal priority into the previous communication program in FreeRTOS. The currently running task will switch back and forth. The switching time is obtained by subtracting the previously measured pure communication time from the communication time including the task switching.

2) Preemption time test: Preemption time is the time it takes for the system to switch from a low priority task to a high priority task.

The test program creates two tasks in FreeRTOS. Task 2 with the higher priority runs first and then delay. Then task 1 will begin a counting process until Task 2 wakes up and preempts the operation of Task 1.

In order to obtain the pure preemption time, the processing time of the test program without delay is also measured. In this program no preemption will occur, so the measured time is the time that the rest of the program spends except for the preemption process. The preemption time can be obtained by subtracting two measured times.

Since the preemption time also includes the task switching time, the task switching time measured before should also be subtracted to obtain the pure preemption time.

3) Semaphore shuffle time test: The semaphore shuffling time is the time it takes from when task 1 releases a semaphore to task 2 that waits for that semaphore to be activated.

The test program creates two tasks with the same priority and a binary semaphore. Each task can either take or give the semaphore and then yield after either action. Task 1 starts first by getting the semaphore and then yield without giving it. Task 2 also needs to take the semaphore, so it blocks to wait. Task 1 re-runs and releases the semaphore, then yields again. Now task 2 can take the semaphore, then give it and end the program [9].

In order to obtain the pure semaphore shuffle time, the processing time of the test program without taking and giving semaphore is also measured. The preemption time can be obtained by subtracting two measured times. Additional switching time also needs to be subtracted.

4) Test results: The response time curve obtained by performing 500 tests is shown in Fig. 9. The statistical data is shown in Table 3.

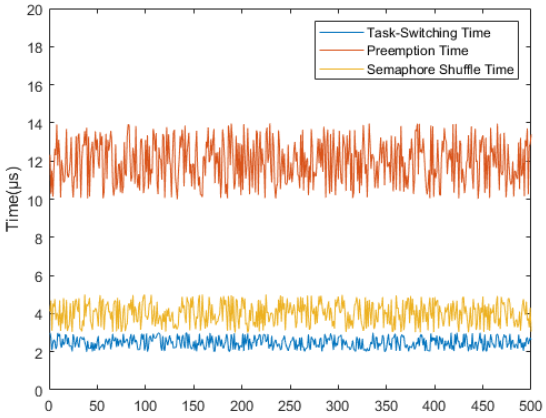


Fig. 9 Response time curve.

TABLE III
STATISTICAL DATA OF RESPONSE TIME

	Maximum time(μs)	Minimum time(μs)	Average time(μs)
task switching time	3.230	2.125	2.568
preemption time	14.016	10.352	12.241
semaphore shuffle time	5.153	3.252	4.055

By comparing with other RTOS, the response time of the dual-system is within the normal range, which can ensure real-time performance.

V. CONCLUSION

This paper designs a dual-core GPOS-RTOS dual operating system controller, which provides a robot controller solution that takes both general tasks and real-time tasks from the perspective of the embedded operating system. The feasibility of the dual-system architecture and the real-time performance are verified. In practical applications, GPOS is mainly responsible for human-computer interaction, robot path planning, and other tasks, while RTOS is responsible for real-time tasks such as motion control and emergency operations. In the future research, the proposed controller could be combined with actual robot to further verify the effectiveness of GPOS and RTOS in practical applications.

ACKNOWLEDGMENT

This work was supported by the National Key R&D Program of China (2018YFB1307400) and the National Natural Science Foundation of China (61873267, U1713224).

REFERENCES

[1] Li Li, Xingming Wu, Weihai Chen, "Implementation of Robot Motion Controller Based on ARM and FPGA," *Computer Measurement & Control*, vol. 9, no. 15, pp. 1162-1171, 2007.

[2] Wei H, Huang Z, Yu Q, "RGMP-ROS: A real-time ROS architecture of hybrid RTOS and GPOS on multi-core processor," *2014 IEEE International Conference on Robotics and Automation*, 2014, pp. 2482-2487.

[3] Yu Q, Wei H, "A novel multi-OS architecture for robot application," *2011 IEEE International Conference on Robotics and Biomimetics*, 2011, pp. 2301-2306.

[4] Wei H, Shao Z, Huang Z, "RT-ROS: A real-time ROS architecture on multi-core processors," *Future Generation Computer Systems*, vol.5, no. 8, pp. 171-178, 2015.

[5] Rajagopalan V, Boppana V, Dutta S, "Xilinx Zynq-7000 EPP: An extensible processing platform family," *2011 IEEE Hot Chips Symposium*, 2011, pp. 1-24.

[6] Chen X, Gu Y, Wang C, "Asymmetric multiprocessing for motion control based on Zynq SoC," *2016 IEEE International Conference on Field-programmable Technology*, 2016, pp. 315-318.

[7] Powell A A, *Performance of the Xilinx Zynq System-on-Chip interconnect with asymmetric multiprocessing*, Master Thesis, Temple University, Philadelphia, PA, USA, 2014.

[8] Kar R P, "Implementing the Rhealstone real-time benchmark," *Dr Dobbs Journal*, vol.15, no. 4, pp. 46-55, 1990.

[9] Boger T J, *Rhealstone benchmarking of FreeRTOS and the Xilinx Zynq Extensible Processing Platform*, Master Thesis, Temple University, Philadelphia, PA, USA, 2013.