symmetry

MDPI

Article

Timing Comparison of the Real-Time Operating Systems for Small Microcontrollers

Ioan Ungurean [1,2]

[1] Faculty of Electrical Engineering and Computer Science; Stefan cel Mare University of Suceava, 720229 Suceava, Romania; ioanu@eed.usv.ro

[2] MANSiD Integrated Center, Stefan cel Mare University, 720229 Suceava, Romania

Received: 9 March 2020; Accepted: 1 April 2020; Published: 8 April 2020

Abstract: In automatic systems used in the control and monitoring of industrial processes, fieldbuses with specific real-time requirements are used. Often, the sensors are connected to these fieldbuses through embedded systems, which also have real-time features specific to the industrial environment in which it operates. The embedded operating systems are very important in the design and development of embedded systems. A distinct class of these operating systems is real-time operating systems (RTOSs) that can be used to develop embedded systems, which have hard and/or soft real-time requirements on small microcontrollers (MCUs). RTOSs offer the basic support for developing embedded systems with applicability in a wide range of fields such as data acquisition, internet of things, data compression, pattern recognition, diversity, similarity, symmetry, and so on. The RTOSs provide basic services for multitasking applications with deterministic behavior on MCUs. The services provided by the RTOSs are task management and inter-task synchronization and communication. The selection of the RTOS is very important in the development of the embedded system with real-time requirements and it must be based on the latency in the handling of the critical operations triggered by internal or external events, predictability/determinism in the execution of the RTOS primitives, license costs, and memory footprint. In this paper, we measured and compared the timing performance for synchronization throughout an event, semaphore, and mailbox for the following RTOSs: FreeRTOS 9.0.0, FreeRTOS 10.2.0, rt-thread, Keil RTX, uC/OS-II, and uC/OS-III. For the experimental tests, we developed test applications for two MCUs: ARM Cortex™-M4 and ARM Cortex™-M0+ based MCUs.

Keywords: real time systems; real time operating systems; task synchronization; microcontrollers

1. Introduction

In the automation systems designed and developed to monitor and control industrial processes, it is very important that they comply with the real-time requirements of the industrial installations. Typically, these systems contain industrial networks (fieldbuses) to which embedded devices are connected. These devices can acquire data from the sensors and receive/send data through the fieldbuses. The embedded systems used in the automation systems must have hard and/or soft real-time features and they must react in the imposed deadline to the data/events received throughout the fieldbuses or data acquired from the sensors connected to the embedded devices [1].

Usually, the design and development of the embedded systems, from the software point of view, is based on an embedded operating system. The real time operating systems (RTOSs) are a particular category of embedded operating systems that were designed to provide support in the design and develop of embedded systems with real-time capabilities. These operating systems have been developed especially for small microcontrollers (MCUs) on 8, 16, or 32 bits that are used to design and develop embedded systems [2]. Examples of RTOSs include FreeRTOS, RT-Thread, eCOS,

LynxOS, QNX, VxWorks, OSEK (Open Systems and their Interfaces for the Electronics in Motor Vehicles), uC-OS/II, uC-OS/III, KEIL RTX, etc. [2]. These operating systems are designed to provide a deterministic and predictable time in the handling of the internal or external events [3,4]. In addition, RTOSs are widely used in the design and development of applications based on the Internet of Things (IoT) and Industrial Internet of Things (IIoT) concepts [5].

RTOSs are part of a category of operating systems designed for embedded systems, especially for systems with a small memory for code and data. Usually, these RTOSs are designed for MCUs that do not use virtual memory [6]. Linux and Windows that use have real time characteristics but there are variants for real time systems for these systems. For example, RTLinux is a patch for Linux with real time capabilities where real time tasks do not use virtual memory and have direct access to the hardware [7]. Windows Embedded Industry is the Windows based operating system for embedded systems with real time capabilities. We will consider small MCUs, the MCUs that do not have virtual memory and cannot use operating systems based on Linux, Windows, or Android (for example, these operating systems can be used on ARM Cortex Ax MCUs). With these MCUs, specialized applications with real-time capabilities can be developed.

This paper aims to make a comparison of RTOSs that are used on small MCUs, such as those based on ARM Cortex Mx architectures. These MCUs use RTOSs with a small memory footprint and without a memory management system such as virtual memory. Applications based RTOS for small MCUs are used in domains such as automotive [8], industrial automation [9], telecommunications [10], avionics [11], military systems [12], Internet of things, Industrial Internet of things [13], and so on. An area of applicability is that of a symmetry concept by developing embedded systems for data acquisition, data compression, pattern recognition, diversity, and sustainability. Usually, the software applications for these domains are a combination of soft and hard real-time tasks [6].

Several features, services, and capabilities are used to compare RTOSs. The most important ones are the maximum time for deactivation of the interrupts and the worst-case execution time (WCET) for tasks and RTOS services such as the routines of service of the interruptions, the time of executing the receipts, etc. [14,15]. Other features used in the comparison are modularity, scalability, memory footprint, latency, response time, and jitter [6,14]. An RTOS provides the basic services for developing multitasking software applications. The fundamental service provided by an RTOS is task management. Each task has associated a priority and the task with the highest priority from the ready state will enter in running state. There are two types of scheduling: preemptive and non-preemptive. At the pre-emptive scheduling, a task from the running state can be preempted by a higher priority task that enters in the ready state, while at the non-preemptive scheduling, the task from the running state must voluntarily release the processor. Most RTOSs use pre-emptive scheduling because it allows for achieving a shorter response time for critical operations. In addition, most RTOSs include synchronization and inter-task communication services such as semaphores, events, mailboxes, and message queues [6].

RTOSs can be compared from several perspectives, some of them are: the time for synchronization and inter-task communication or the response time of the task with the highest priority after the expected event occurs. In [14], we presented a first comparative test for the time of task context switching where task switching is triggered by an event, semaphore, or mailbox. In this paper, we measured, compared, and analyzed the timing performances for the following RTOSs: FreeRTOS 9.0.0, FreeRTOS 10.2.0, rt-thread, Keil RTX, uC/OS-II, and uC/OS-III. All of these RTOSs support preemptive scheduling.

For testing, we used two MCUs: ARM Cortex™-M4 and ARM Cortex™-M0+ based MCUs. The paper extends the experimental tests presented in [14]. In [14], there are measured and compared timing performances for task context switching from a lower priority task to a higher priority task triggered by an event, semaphore, and mailbox. In this paper, the tests are completed by measuring the timing performances for the task context switching from a higher priority task to a lower priority task

and the timing performance for the primitives (non-blocking) used to wait and send/signal an event, semaphore, and mailbox. Furthermore, we use the newest compiler provided by the KEIL MDK-ARM.

The main contribution of this paper is that the experimental tests are performed in actual MCUs, and not just simulations using software tools as found in related works. A test pin (an output GPIO (general-purpose input/output) selected according to the MCU and board used) is used to signal the beginning and the end of the operation for which we want to measure the time and the time is measured with an oscilloscope on this test pin. This paper aims to compare the timing for synchronization of the tasks for the most used RTOSs on small microcontrollers. To achieve this goal, RTOSs are executed on the same hardware configuration.

This paper is structured as follows: Section 2 presents some performances tests presented in the specialized literature and the motivation for the target RTOS's selection, Section 3 describes the test setup. Section 4 contains discussions related to the experimental results. The conclusions are drawn in Section 5.

2. Related Works

Interesting studies related to the market for embedded systems are published by EETimes.com and Embedded.com every year. Unfortunately, the last study was published in 2017 and it has not updated [16]. In the last market study, they conclude that 68% of the ongoing embedded projects use and embedded operating system, and of these, 41% use an open-source embedded operating system. If we exclude the embedded system based on Linux or Windows, the most used RTOS are FreeRTOS (20% of the ongoing embedded projects that use an operating systems), in house solution (19%), Texas Instruments RTOS (5%), Texas Instruments DSP/BIOS (5%), Micrium uC-OS/III (5%), Keil RTX (4%), Micrium uC-OS/II (4%), and Wind River VxWorks (4%). From embedded operating systems based on Windows or Linux, the most used are Embedded Linux (22%), Debian (13%), Microsoft (Windows Embedded 7/Standard) (8%), Microsoft (Windows 7 Compact or earlier) (5%), and Angstrom (3%).

There are also other studies for the RTOS market such as the one published by MarketWatch Company in 2019 that presents the recent trends, size, growth, top manufacturers, and forecast to 2024 related to RTOS systems [17].

In [18], the authors present a Benchmark of Real Time Operating Systems. They focused ion FreeRTOS, RTEMS, uC/OS-III, and Linux and they measured the overhead for semaphore and message queue services in different scenarios on an MPCore Cortex A−7 900 MHz MCU. In this case, the best performances (low overhead) are achieved by uC/OS-III and the weaker performances by Linux.

A comparison between a multicore RTOS and VxWorks is presented in [19]. The evaluation of performances is performed in a simulator. The paper highlight the performances gain for a multicore RTOS is. In [20], a performance evaluation of the RTOS for robotics is presented. Benchmarks and comparisons of the RTOSs that use CMSIS-RTOS layer are presented in [21]. The tests are performed for the RTX, X RT Kernel, FOSS Free and open-source software), and ChibiOS. The authors conclude that the use of the CMSIS(Cortex Microcontroller Software Interface Standard)-RTOS layer do not generate overhead for the RTOSs. In [21], the authors present an evaluation of the performances of real time systems for vision based navigation. The tests are performed on a Rasberry Pi 2 model B device on PREEMPT_RT and Xenomai kernels. The authors conclude that the best performances are achieved by the Xenomai kernel. In [22], a comparison related performances of the FreeRTOS and uC/OS-III is presented. The experimental tests are performed on Renesas RX63N MCU for the memory footprint, latency, and service performances. In almost all of the tests, the uC/OS-III outperforms the performances of the FreeRTOS. All of these comparisons are performed using simulators or different software tools. In this paper, we wanted to compare the most used RTOS for small MCUs in actual and modern MCUs using a GPIO output pin as a test pin to measure the time for different operations. By this method, we can measure the real-time of the operations without other influences and as close as possible to the real cases. RTOSs will be executed on the same hardware platform for a more accurate comparison of performance.

In this paper, we focused on four RTOSs for small MCUs: FreeRTOS, Micrium uC-OS/II and Micrium uC-OS/III, Keil RTX, and rt-thread. FreeRTOS [23] is an open-source RTOS widely used in the embedded systems project. Micro-Controller Operating Systems (uC-OS) is a commercial RTOS [24]. Keil RTX is a royalty-free RTOS included in the KEIL MDK-ARM tools [25]. rt-thread is an open source RTOS [26].

We chose these RTOSs because they are at the top in the study [14], to which we added rt-thread because we used it in previous projects. In addition, the targets of these RTOSs are small MCUs such as those based on ARM Cortex M0, M3, and M4 architectures. Focusing on small microcontrollers, no Linux or Windows-based operating systems were used because they target systems that are more complex and, with small exceptions, cannot be executed on small MCUs. For example, Linux can be executed on Arm Cortex M4 microcontrollers, but this variant is not used in practice for developing dedicated devices with real-time capabilities. For the selected RTOSs, there are many examples of real-time applications developed, and uC/OS-II and uC/OS-III have certifications for the development of applications in the avionics, industrial control, and medical fields [27]. In [28], the authors propose a system in the manufacturing IoT environment that includes aggregation nodes based on an Arm Cortex M4 MCU and rt-thread RTOS. An evolution and an analysis of the real-time behaviors of the FreeRTOS is presented in [29]. In [30], the authors present an implementation and performances of a low-level control of omnidirectional mobile robot based on Keil RTX RTOS and an ARM Cortex M4 MCU. In [31], an Electromagnetic transmitter for EM-MWD System is presented. The device is developed with TMS320F2S12 MCU (digital signal processors based on Armv8.1-M architecture) and uC/OS-III RTOS.

All of these examples highlight the fact that the selected RTOSs are used in the development of real applications with real-time capabilities and present additional arguments for their selection, in addition to the study presented in [14].

3. Experimental Setup

In [14], we presented some experimental results related to the time for context switching between a low priority task to a higher priority task for the uC-OS/II, FreeRTOS 9.0.0, rt-thread, and Keil RTX triggered by an event, semaphore and mailbox on two MCUs: STM32F407IG ARM Cortex™-M4 MCU, and STM32L053R8 ARM Cortex™-M0+ MCU. In this paper, we complete the experimental tests with the time for the context switching between a high priority task to a lower priority task (blocking wait in the high priority task) and the time for the primitives used to send and receive an event, semaphore, and mailbox. Furthermore, we perform the experimental tests on two additional RTOSs: FreeRTOS10.2.0 and uC-OS/III.

In the current section, we will describe and explain the experimental software applications when the event is used as a synchronization mechanism between two tasks. For the events, we tested the following scenarios: scenario 1—signal an event causing a context switch, scenario 2—wait for an event causing context switching, scenario 3—signal an event without context switching (unblocking signal), and scenario 4—wait for an available event (unblocking wait). In these scenarios, a test pin (an output GPIO selected according to the MCU and board used) will be used to signal the beginning and the end of the operation for which we want to measure the time. The time will be measured with an oscilloscope on the test pin. For each scenario, we have a software application. This software application is uploaded to the MCU flash in order to be executed.

The software diagram of the test software applications for the scenario 1 is shown in Figure 1 (it is similar with the test software application used in [14]). The software applications consist of two tasks with distinct priorities and the RTOSs are configured to use preemptive scheduling. In the test software applications, the task with the lower priority, at a period of 1 ms, sets the test pin to 0 (LOW/FALSE) and signals an event. The higher priority task waits, in an infinite loop, the event with infinite time-out and when it receives the event, it sets the test pin to 1 (HIGH/TRUE). Thus, with an oscilloscope on the

test pin, it is possible to determine the time for the task context switching triggered by the task with a lower priority by sending the event waited by a task with the higher priority.

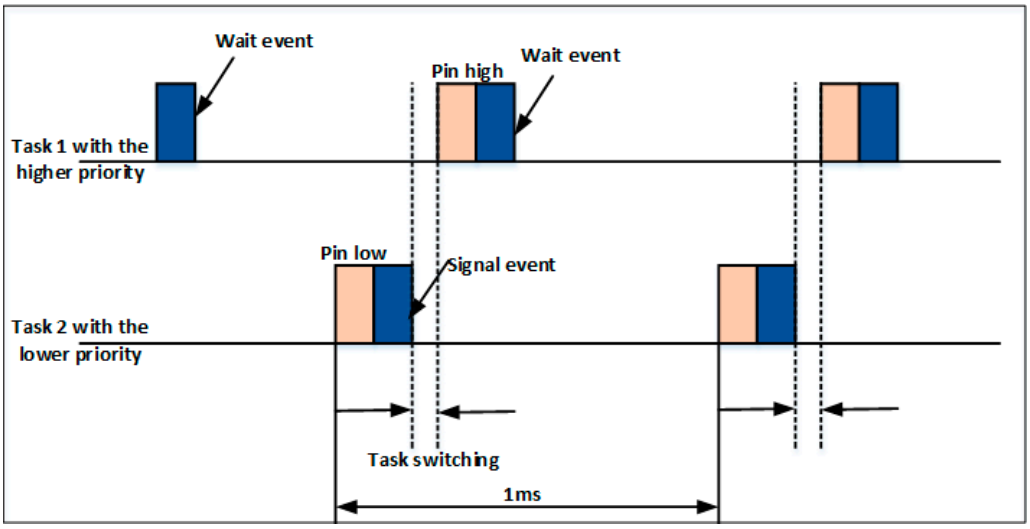


Figure 1. The software diagram of the test software applications for scenario 1.

Figure 2 presents the software diagram of the test software applications for the scenario 2. The software applications consist of two tasks with distinct priorities and preemptive scheduling will be used. The higher priority task sets the test pin to 0 (LOW/FALSE) and it waits an event with a time-out of 1 ms. The lowest priority task consists of an infinite loop that sets the test pin to the one logic. By calling the primitive for the event wait, a context switch is triggered and the lower priority task, which is in the ready state, will pass to the running state and it will set the test pin to 1 (HIGH/TRUE). With this configuration and by using the test pin, we can measure the time for task context switching from a higher priority task to a lower priority task.

These two scenarios can be found in real software applications. For example, a task can wait for data from a communication line and when data are received, an event, mailbox, or semaphore is sent to the task that waits for data. This can trigger a task context switch to execute the task that waits for data. Or, a task can expect an event from another peripheral, such as a digital input, and a task context switch must be triggered (with an event, mailbox, or semaphore) to handle that event.

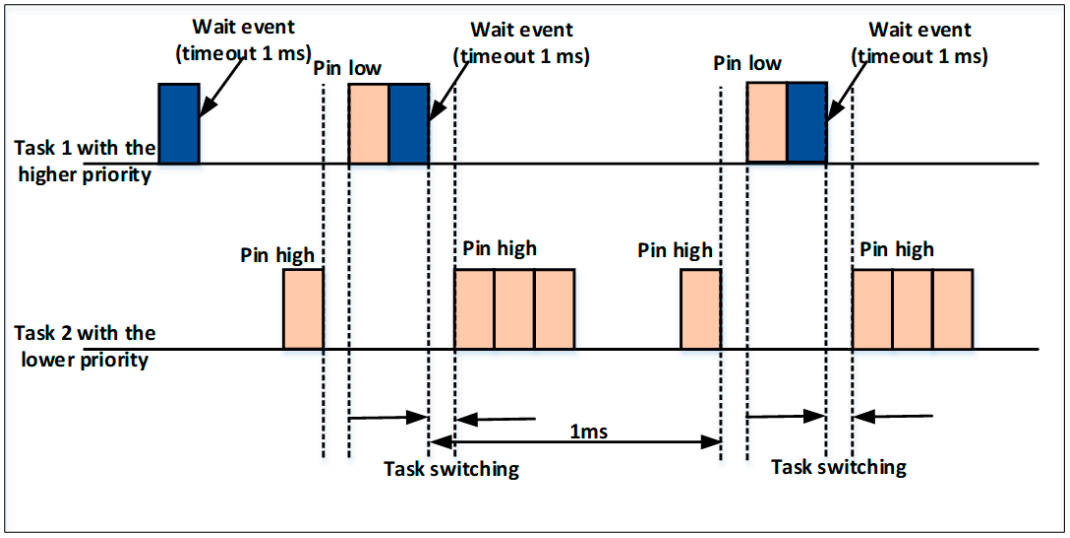


Figure 2. The software diagram of the test software applications for scenario 2.

Figure 3 presents the software diagram of the test software applications for the scenario 3. The software applications consist of only one task. In this case, every 1 ms, the test pin is set to 0 (LOW/FALSE), and an event (not expected by any task) is triggered, whereupon the test pin is set to 1 (HIGH/TRUE). After these operations, the primitive for waiting an event is called to receive and clear the event. In this scenario, with the help of the test pin, the execution time for the primitive for event triggering can be measured.

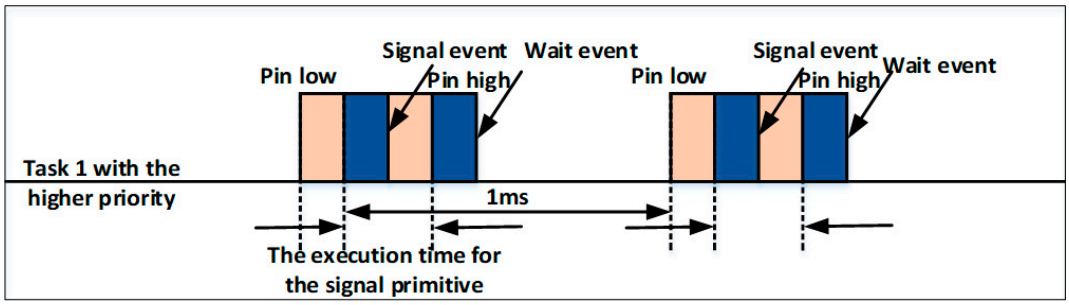


Figure 3. The software diagram of the test software applications for scenario 3.

Figure 4 presents the software diagram of the test software applications for the scenario 4. The software applications consist of only one task. In this scenario, we want to measure the execution time for the primitive used to receive an existing event. The software applications are similar with the third scenario, with differences that the test pin is set to 0 (LOW/FALSE) before the call for the waiting primitive and is set to 1 (HIGH/TRUE) after this call.

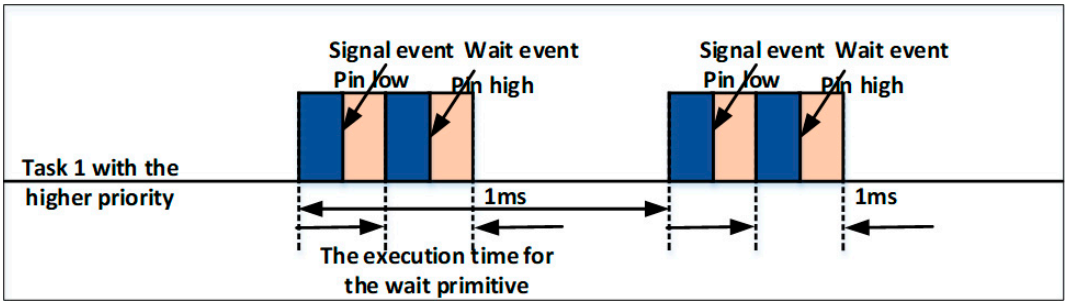


Figure 4. The software diagram of the test software applications for scenario 4.

Test software applications with the same configuration were developed when semaphores or mailboxes are used as synchronization mechanisms, using the waiting and signalling primitives specific to each mechanism. We developed a software application for each of the chosen RTOSs on two MCUs: STM32F407IG ARM Cortex™-M4 MCU and STM32L053R8 ARM Cortex™-M0+ MCU. We used the following versions of RTOSs: uC-OS/II V2.92.11, uC-OS/III V3.05.00, FreeRTOS 9.0.0, FreeRTOS 10.2.0, rt-thread V2.1.1, and Keil RTX V4.82.0. For all RTOSs, the most code is in C because the purpose of these RTOSs is to provide better portability. Using the same compiler, they can be compared under the same conditions, with no optimized code. For each synchronization mechanism, a software application has been developed for each MCU, and from these software applications, the scenarios are activated with conditional compilation.

The tests were performed on two MCUs: STM32F407IG ARM Cortex™-M4 MCU and STM32L053R8 ARM Cortex™-M0+ MCU. The following development kits were used for these MCUs: KEIL MCBSTM32F400 and STM32 NUCLEO-L053R8. These two architectures are not similar although both are based on a 32-bit RISC processor. ARM Cortex M0 + used a Von Neumann architecture with instruction pipelining of two stages. The ARM Cortex M4 used a Harvard architecture with Instruction pipelining of three stages. For the STM32F407IG ARM Cortex™-M4 MCU, the system

clock was setup at 168 MHz (the source is a high-speed external clock of 25MHz with PLL activated) and for the STM32L053R8 ARM Cortex™-M0+ MCU, the system clock was setup at 32 MHz (the source is high-speed internal clock with PLL activated). The purpose is to compare the performances of the RTOSs on the same MCU, not on different MCUs. We used two MCUs to see whether performance differences between RTOSs are the same on different MCUs. On the two chosen microcontroller architectures, a wide range of devices with hard/soft real-time capabilities can be designed and developed based on real-time operating systems.

RTOSs were configured with fully preemptive scheduling, without round-robin (a single task on a priority level) and with the internal tick clock of 1 ms. For all software applications, we used the latest compiler (V5.06 update 6 build 750) included in the KEIL MDK-ARM Pro 5.29 tools with compiler optimization level 3 (O3).

The configurations for compiler are the same for all software applications and we used the same functions to handle the test GPIO pin. For the STM32L053R8 ARM Cortex™-M0+ MCU we used the PC port, pin 13 as test pin, and for STM32F407IG ARM Cortex™-M4 MCU we used the GPIOH port, pin 3 as test pin.

For this reason, the differences related to the time performances are due only to the way of implementation of each RTOS. Regarding the measured values, the measurements errors are generated by the oscilloscope (we used the PicoScope 2205MSO—that provides a vertical resolution up to 12 bits and a time base accuracy of ±100 ppm).

For each chosen MCU, there are developed three software applications for each RTOS, one for mailboxes, one for semaphores, and one for events. These software applications consist of 2 tasks and, with conditional compilation, one of the 4 scenarios defined in this section is activated. Some examples for the code for the tasks are shown in Appendix A for FreeRTOS when semaphores are used, Appendix B for uC/OS-II RTOS when events are used, Appendix C for rt-thread when mailboxes are used, and Appendix D for Keil RTX when events are used. With the help of the RTOS_TEST macro, one of the 4 scenarios is activated. In addition, from these examples, you can see that the same functions are used to handle the test pin. It can be observed that the operations are executed periodically, at every 1 ms, to capture any jitter that may occur. The used oscilloscope is capable to detect and to measure this jitter. The jitter is very important in determining the worst-case execution time (WCET), which is very important for hard real-time systems.

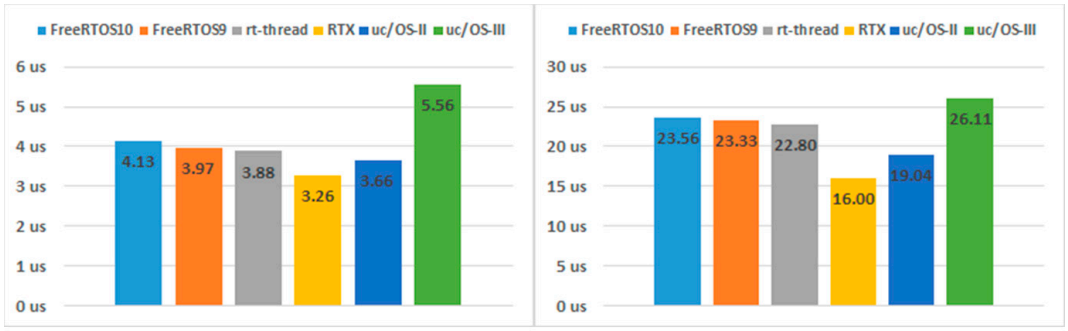
4. Experimental Results and Discussions

In this section, the results obtained for the tests described in the previous section are presented. The tests were performed on two MCUs: STM32F407IG ARM Cortex™-M4 MCU and STM32L053R8 ARM Cortex™-M0+ MCU.

For the measurements, we used the PicoScope 2205MSO oscilloscope. The operations are periodically triggered at every 1 ms and the presence of jitter can be detected on the oscilloscope (the jitter is determined by the oscilloscope, not by software). The test pin was connected to the analog part of the oscilloscope, we used a threshold of 3 V was used to detect the start of the operation (to detect the transition from 0 V to 3.3 V), and we used a timebase of 5 µs/div.

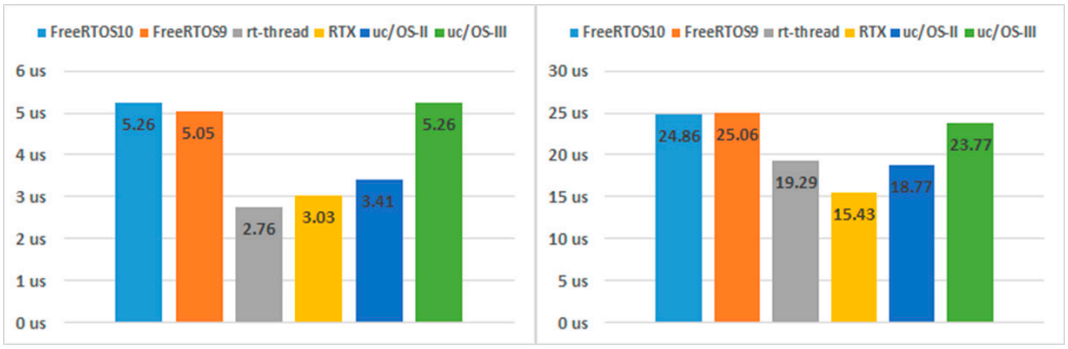
All operations are performed periodically and the oscilloscope can easily detect the present of jitter. The jitter of 1 ms clock tick is not present because each operation is triggered at the end of the clock tick interrupt and is much shorter by 1 ms. All measurements were performed in a series. The uncertainty budget of the time measurement is generated by the errors generated by the PicoScope 2205MSO oscilloscope that provides a vertical resolution up to 12 bits and a time base accuracy of ±100 ppm. For a timebase of 5 us, the uncertainty budget of the time measurement is of ± 0.5 ns.

Figure 5, Figure 6, and Figure 7 present the results obtained for the first scenario when a task context switch is triggered by an event, semaphore, and mailbox. From these figures, it can be observed that the smallest latency is obtained by Keil RTX when events are used, by rt-thread and Keil RTX when semaphores are used, and by FreeRTOS0 and Keil RTX when mailboxes are used. It can be seen that there are small differences between STM32F407IG ARM Cortex ™ -M4 and STM32L053R8 ARM Cortex ™ -M0+ MCUs regarding latencies obtained by RTOS systems (comparison between RTOSs on the same MCU). In the first case, no jitter appears in the latency measurement (the task switching is periodically triggered at every 1 ms and the presence of jitter could be observed on the oscilloscope).



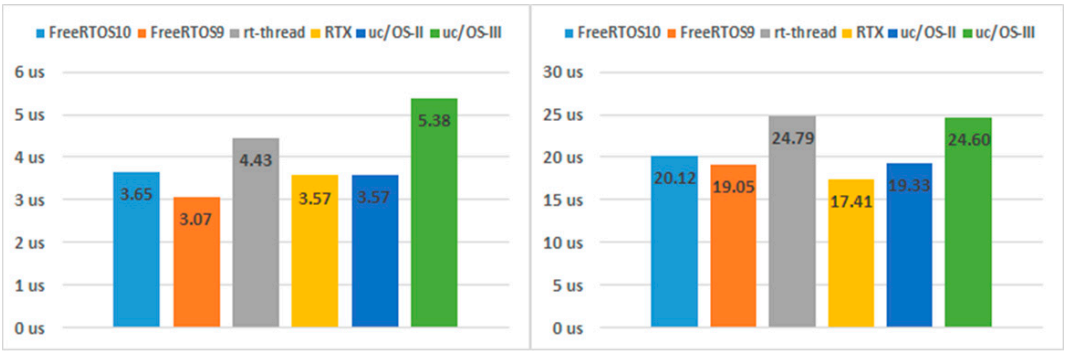
(a) STM32F407IG ARM Cortex™-M4 MCU **(b) STM32L053R8 ARM Cortex™-M0+ MCU**

Figure 5. The time for task switching from the lowest priority task to the higher priority task triggered by an event (scenario 1).



(a) STM32F407IG ARM Cortex™-M4 MCU **(b) STM32L053R8 ARM Cortex™-M0+ MCU**

Figure 6. The time for task switching from the lowest priority task to the higher priority task triggered by a semaphore (scenario 1).



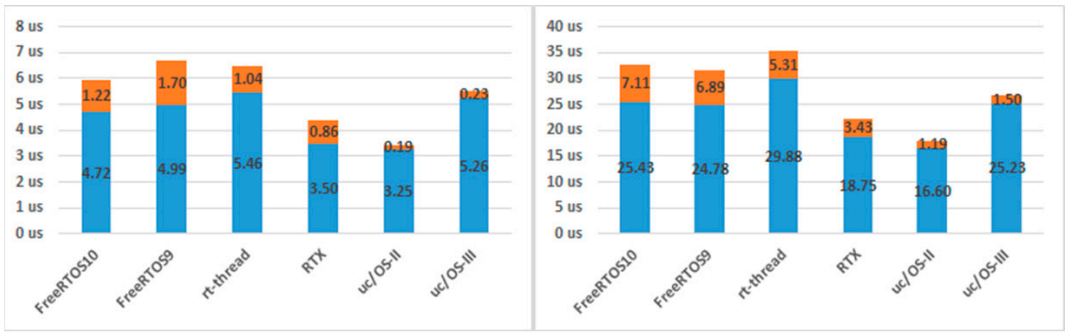
(a) STM32F407IG ARM Cortex™-M4 MCU **(b) STM32L053R8 ARM Cortex™-M0+ MCU**

Figure 7. The time for task switching from the lowest priority task to the higher priority task triggered by a mailbox (scenario 1).

The latency measurement is performed using a GPIO pin, and the time for port handling is the same for each RTOS. Furthermore, each RTOS has the same latency generated by the access of the GPIO port that is connected to the peripheral bus of the MCU. Usually, embedded systems must react to external events that are received via peripherals (GPIO, ADC, UART, CAN, etc.) and the reaction can be sent further through peripherals (GPIO, DAC, UART, CAN, etc.). For this reason, we consider that the use of a GPIO pin to measure latency can bring us closer to the real functioning of the real-time system that was developed based on an RTOS.

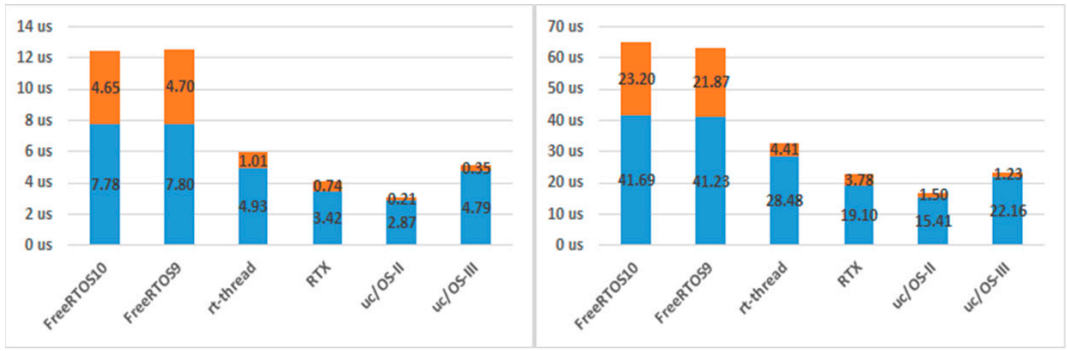
The results for the second scenario are presented in Figure 8, Figure 9, and Figure 10. As can be seen from the figures, in this case, jitter is present during task switching. The jitter results from the fact that the second task sets the test pin to 1 (HIGH/TRUE) in an infinite loop and it is preempted by the higher priority task while it sets the pin. The preemption moments can influence the time when the task is resumed and sets the test pin to 1 (HIGH/TRUE). In addition, there is an influence generated by the interrupt service routine for the clock tick. From the figures, we can see that the smallest latency and the smallest jitter is obtained by uC/OS-II and the highest latency is obtained by the rt-thread in the case of events and mailboxes and by FreeRTOS9 in the case of semaphores.

In Figure 11, Figure 12, and Figure 13 the results for the third scenario are presented. In this case, the time is measured when an event is triggered, a semaphore is issued, or a message is sent. For events and semaphores, the lowest latency is obtained for rt-thread and for mailboxes, the lowest latency is obtained for uC/OS-II. For events and semaphores, the highest latency is obtained for FreeRTOS10, and for mailboxes, Keil RTX obtains the lowest latency.



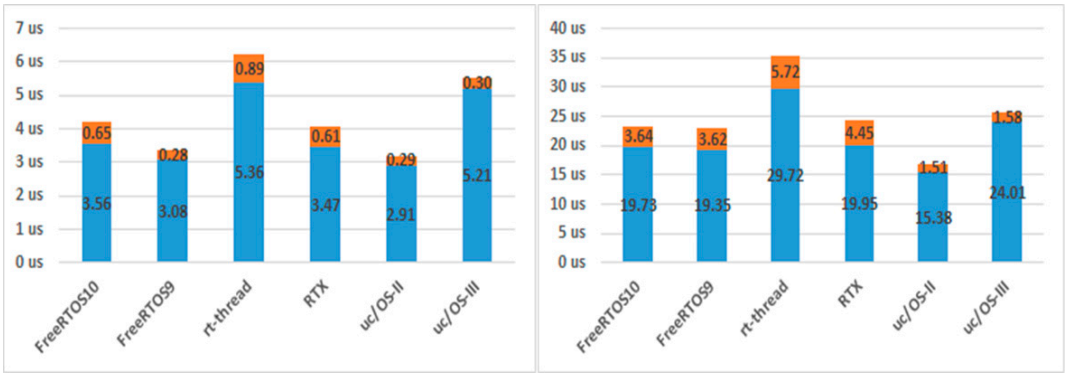
(a) STM32F407IG ARM Cortex™-M4 MCU **(b) STM32L053R8 ARM Cortex™-M0+ MCU**

Figure 8. Time for task switching from the lowest priority task to the higher priority task triggered by an event (the values from the top are the jitters) (scenario 2).



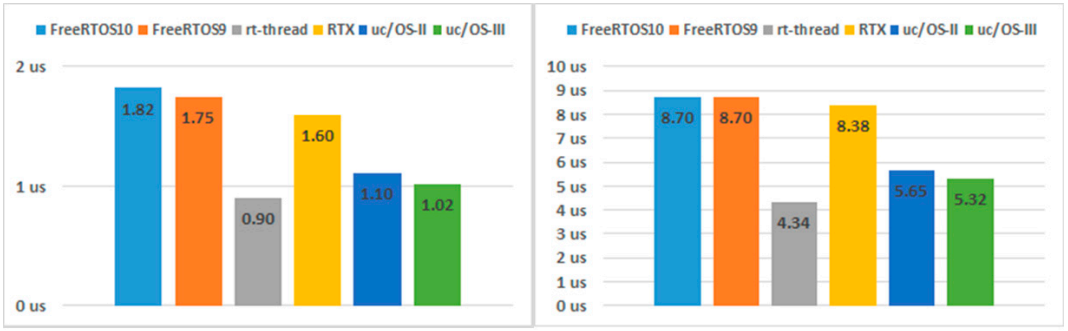
(a) STM32F407IG ARM Cortex™-M4 MCU **(b) STM32L053R8 ARM Cortex™-M0+ MCU**

Figure 9. Time for task switching from the lowest priority task to the higher priority task triggered by a semaphore event (the values from the top are the jitters (scenario 2).



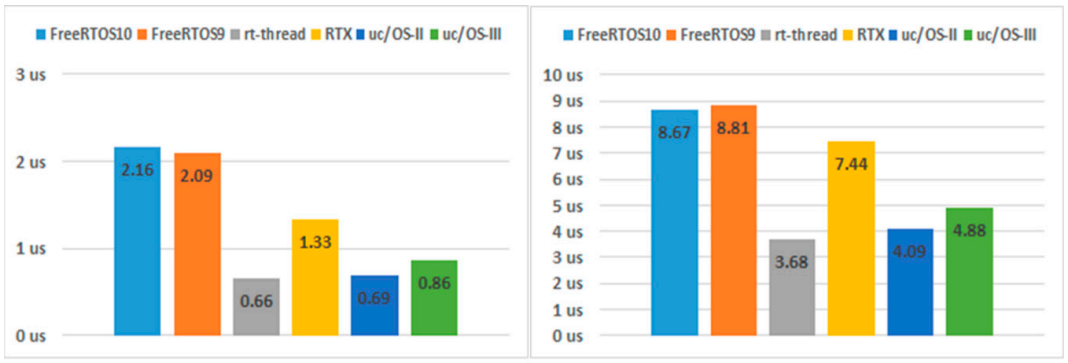
(a) STM32F407IG ARM Cortex™-M4 MCU **(b) STM32L053R8 ARM Cortex™-M0+ MCU**

Figure 10. The time for task switching from the lowest priority task to the higher priority task triggered by a mailbox (the values from the top are the jitters) (scenario 2).



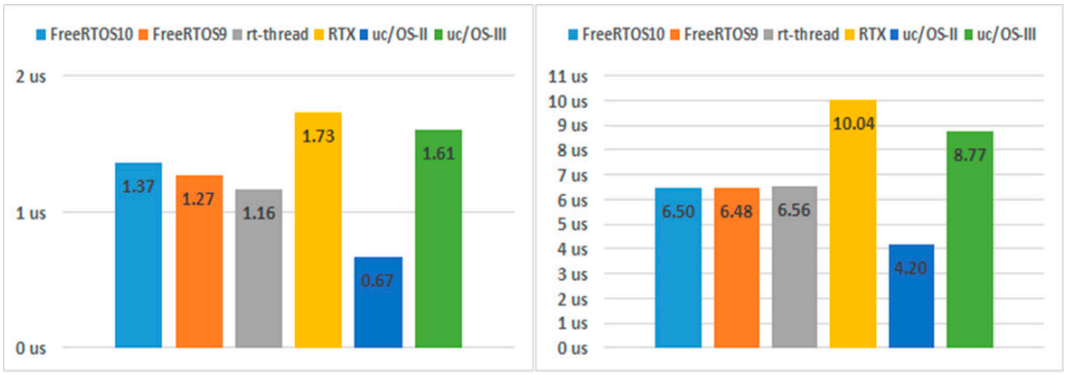
(a) STM32F407IG ARM Cortex™-M4 MCU **(b) STM32L053R8 ARM Cortex™-M0+ MCU**

Figure 11. The execution time for the primitive that signal an event that is not waited by the other task (unblocking send) (scenario 3).



(a) STM32F407IG ARM Cortex™-M4 MCU **(b) STM32L053R8 ARM Cortex™-M0+ MCU**

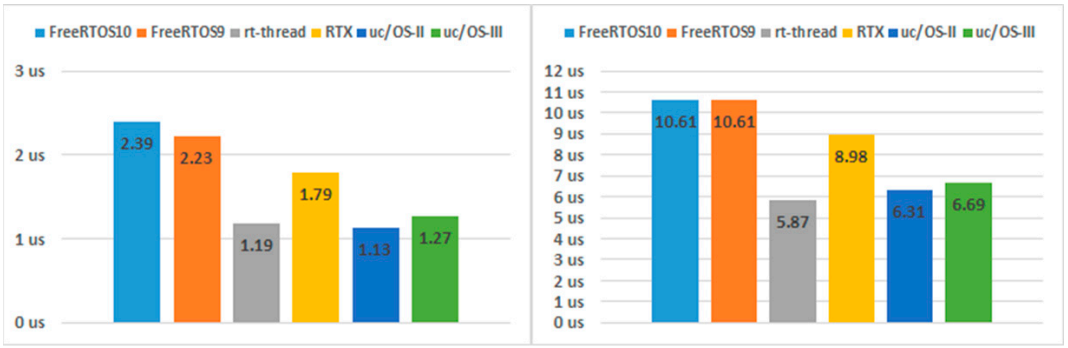
Figure 12. The execution time for the primitive that signal a semaphore that is not waited by the other task (unblocking send) (scenario 3).



(a) STM32F407IG ARM Cortex™-M4 MCU **(b) STM32L053R8 ARM Cortex™-M0+ MCU**

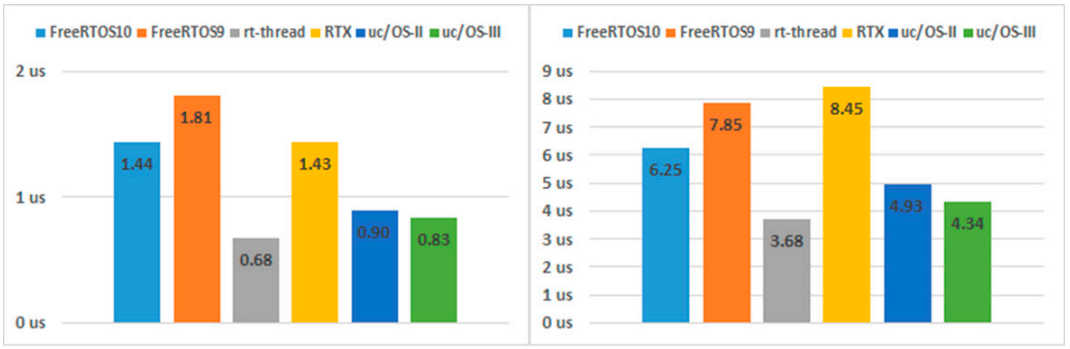
Figure 13. The execution time for the primitive that send a mailbox that is not waited by the other task (unblocking send) (scenario 3).

Figure 14, Figure 15, and Figure 16 present the results for the fourth scenario. In this case, it is measured the time when an event, semaphore or mailbox is received. The results are similar to the previous scenario, in the sense that for events and semaphores, the highest latency is obtained for FreeRTOS10 and for the mailboxes Keil RTX obtains the lowest latency.



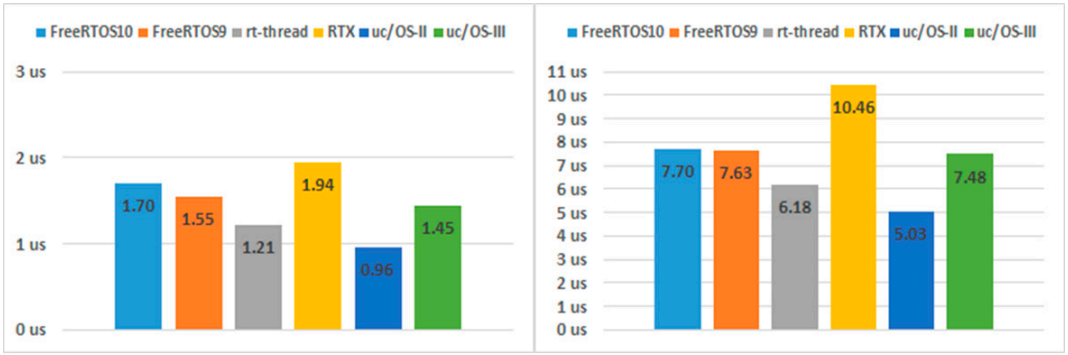
(a) STM32F407IG ARM Cortex™-M4 MCU **(b) STM32L053R8 ARM Cortex™-M0+ MCU**

Figure 14. The execution time for the primitive used to receive an existing event (unblocking wait) (scenario 4).



(a) STM32F407IG ARM Cortex™-M4 MCU **(b) STM32L053R8 ARM Cortex™-M0+ MCU**

Figure 15. The execution time for the primitive used to receive an existing semaphore (unblocking wait) (scenario 4).



(a) STM32F407IG ARM Cortex™-M4 MCU **(b) STM32L053R8 ARM Cortex™-M0+ MCU**

Figure 16. The execution time for the primitive used to receive an existing mailbox (unblocking wait) (scenario 4).

The measurement error is the same for all cases because the same oscilloscope is used. All RTOSs use supervisor call (SVC) interrupt to perform task switching, but there are differences in how the directives for communication mechanisms are executed, and how the queues of different events (event, semaphore, and mailbox) are accessed internally.

Furthermore, scenarios for measurement of the energy consumption can be made, but this is dependent on the hardware platform used and as long as the MCU and other components enter the low power consumption mode. On the same hardware platform, energy consumption is directly proportional to the execution time that is measured in the scenarios proposed in this article.

Analyzing the results for the four scenarios, we can say that the best RTOS performers are uC/OS-II, Keil RTX, and rt-thread. FreeRTOS, which is the most used RTOS in embedded projects, does not have the best latency performances but the performances are close to the other RTOSs tested in our case. The differences appear in the way of implementation of the access to internal queues for events, semaphores, and mailboxes.

Also, when a RTOS is selected to design and develop applications with real-time capabilities, other elements such as licensing mode, memory footprint, predictability, certifications, and support provided must be considered. Usually, there are preferred mature RTOSs that have proven their functionality and efficiency in developing other applications with real-time capabilities.

5. Conclusions

This paper presented an analysis and comparison in terms of timing performances for task synchronization throughout events, semaphore, and mailboxes for six RTOSs (uC-OS/II, uC/OS-III, FreeRTOS 9.0.0, FreeRTOS 10.2.0, RT-Thread, and Keil RTX) that are widely used to design and develop applications on small MCUs. We measured the time for task switching triggered by an event, semaphore, and mailbox and we compared the time achieved by the chosen RTOSs on ARM Cortex™-M4 and STM32L053R8 ARM Cortex™-M0+ based MCUs. In addition, we measured the latencies for the directives used to send/receive an event, semaphore, and mailbox. From the experimental results, we can conclude that the best performances are achieved for uC/ OS-II, Keil RTX, and rt-thread. The lower performances (close to the others RTOSs) are achieved by the FreeRTOS although it is the most used RTOS in embedded projects. It should be mentioned that the tests were performed for small MCUs, and for this reason, no variants of Linux based RTOSs have been tested. Although the timing is the most important parameter in a hard or a soft real-time system, we must consider that we need to include other criteria in selecting an RTOS, such as licensing, memory footprint, predictability, certifications, and support provided. The results presented in this paper may represent an indication for selecting an RTOS in terms of response time to the occurrence of a critical event (periodic or aperiodic), and which is the most efficient synchronization mechanism for the selected RTOS.

Conflicts of Interest: The author declare no conflict of interest. The funders had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript, or in the decision to publish the results.

Appendix A

This appendix contains the code for two tasks of the test software application for FreeRTOS RTOS when semaphores are used.

```
void Task1 (void *argument)
{
      /* Attempt to create a semaphore. */
      xSemaphore = xSemaphoreCreateBinary();
      while (1)
      {
#if (RTOS_TEST == TEST1)
    xSemaphoreTake(xSemaphore, 10000);
    PIN_On(0);
#elif (RTOS_TEST == TEST2)
           vTaskDelay(1);  // delay 1 ticks = 1ms
           PIN_Off(0);
           xSemaphoreTake(xSemaphore, 1);
           PIN_Off(0);
#elif (RTOS_TEST == TEST3)
           xSemaphoreGive( xSemaphore );
           vTaskDelay(1);  // delay 1 ticks = 1ms
           PIN_Off(0);
           xSemaphoreTake(xSemaphore, 10000);
           PIN_On(0);
#elif (RTOS_TEST == TEST4)
           vTaskDelay(1);  // delay 1 ticks = 1ms
           PIN_Off(0);
           xSemaphoreGive( xSemaphore );
           PIN_On(0);
           xSemaphoreTake(xSemaphore, 10000);

#else
           osDelay(1);
#endif
  }
}
void Task2 (void *argument)
{
      while (1)
      {
#if (RTOS_TEST == TEST1)
```

```
            vTaskDelay(1);  // delay 1 ticks = 1ms
            PIN_Off(0);
            xSemaphoreGive( xSemaphore );
#elif (RTOS_TEST == TEST2)
            PIN_On(0);
#elif (RTOS_TEST == TEST3)
            vTaskDelay(1);  // delay 1 ticks = 1ms
#elif (RTOS_TEST == TEST4)
            vTaskDelay(1);  // delay 1 ticks = 1ms
#else
            vTaskDelay(1);  // delay 1 ticks = 1ms
#endif
      }
}
```

Appendix B

This appendix contains the code for two tasks of the test software application uC/OS-II RTOS when events are used.

```
void Task1 (void *argument)
{
      OS_ERR  err;
      while (1)
      {
#if (RTOS_TEST == TEST1)
            // wait until bit 0 is 0
            OSFlagPend(flg_grp, 0x01, OS_FLAG_WAIT_CLR_ALL, 0, &err);
            PIN_On(0);
            OSFlagPost(flg_grp, 0x01, OS_FLAG_SET, &err); // set bit 0
#elif (RTOS_TEST == TEST2)
            PIN_Off(0);
            // wait until bit 0 is 0
            OSFlagPend(flg_grp, 0x01, OS_FLAG_WAIT_CLR_ALL, 2u, &err);
#elif (RTOS_TEST == TEST3)
            OSFlagPost(flg_grp, 0x01, OS_FLAG_CLR, &err); //reset bit 0
            OSTimeDlyHMSM(0,0,0,1);  // wait 1ms
            PIN_Off(0);
            // wait until bit 0 is 0
            OSFlagPend(flg_grp, 0x01, OS_FLAG_WAIT_CLR_ALL, 0, &err);
            PIN_On(0);
            OSFlagPost(flg_grp, 0x01, OS_FLAG_SET, &err); // set bit 0
#elif (RTOS_TEST == TEST4)
            OSTimeDlyHMSM(0,0,0,1); // wait 1ms
            PIN_Off(0);
```

```
            OSFlagPost(flg_grp, 0x01, OS_FLAG_CLR, &err); //reset bit 0
            PIN_On(0);
            // wait until bit 0 is 0
            OSFlagPend(flg_grp, 0x01, OS_FLAG_WAIT_CLR_ALL, 0, &err);
            OSFlagPost(flg_grp, 0x01, OS_FLAG_SET, &err); // set bit 0
#else
            osDelay(1);
#endif
  }
}
void Task2 (void *argument)
{
      OS_ERR  err;

      while (1)
      {
#if (RTOS_TEST == TEST1)
            OSTimeDlyHMSM(0,0,0,1);  // wait 1ms
            PIN_Off(0);
            OSFlagPost(flg_grp, 0x01, OS_FLAG_CLR, &err); //reset bit 0

#elif (RTOS_TEST == TEST2)
            PIN_On(0);
#elif (RTOS_TEST == TEST3)
            OSTimeDlyHMSM(0,0,0,1); // wait 1ms
#elif (RTOS_TEST == TEST4)
            OSTimeDlyHMSM(0,0,0,1); // wait 1ms
#else
            OSTimeDlyHMSM(0,0,0,1); // wait 1ms
#endif
  }
}
```

Appendix C

This appendix contains the code for two tasks of the test software application rt-thread RTOS when mailboxes are used.

```
void Task1 (void *argument)
{
      rt_uint32_t e;
      while (1)
      {
#if (RTOS_TEST == TEST1)
            rt_mb_recv(mailboxTest, &e, RT_WAITING_FOREVER);
```

```
            PIN_On(0);
#elif (RTOS_TEST == TEST2)
            rt_thread_delay(1);//wait 1ms
            PIN_Off(0);
            rt_mb_recv(mailboxTest, &e, 1u);
            PIN_Off(0);
#elif (RTOS_TEST == TEST3)
            rt_mb_send(mailboxTest,0);
            rt_thread_delay(1);//wait 1ms
            PIN_Off(0);
            rt_mb_recv(mailboxTest, &e, RT_WAITING_FOREVER);
            PIN_On(0);
#elif (RTOS_TEST == TEST4)
            rt_thread_delay(1);//wait 1ms
            PIN_Off(0);
            rt_mb_send(mailboxTest,0);
            PIN_On(0);
            rt_mb_recv(mailboxTest, &e, RT_WAITING_FOREVER);
#else
            osDelay(1);
#endif
  }
}

void Task2 (void *argument)
{
     while (1)
     {
#if (RTOS_TEST == TEST1)
            rt_thread_delay(1);//wait 1ms
            PIN_Off(0);
            rt_mb_send(mailboxTest,0);
#elif (RTOS_TEST == TEST2)
            PIN_On(0);
#elif (RTOS_TEST == TEST3)
            rt_thread_delay(1);//wait 1ms
#elif (RTOS_TEST == TEST4)
            rt_thread_delay(1);//wait 1ms
#else
            rt_thread_delay(1);//wait 1ms
#endif
  }
}
```

Appendix D

This appendix contains the code for two tasks of the test software application for Keil RTX RTOS when semaphores are used.

```
void Task1(void *argument)
{
      semaphore = osSemaphoreCreate(osSemaphore(semaphore), 0);
      while (1)
      {
#if (RTOS_TEST == TEST1)
            osSemaphoreWait(semaphore, osWaitForever);
            PIN_On(0);
#elif (RTOS_TEST == TEST2)
            osDelay(1);
            PIN_Off(0);
            osSemaphoreWait(semaphore, 1U);
            PIN_Off(0);
#elif (RTOS_TEST == TEST3)
            osSemaphoreRelease(semaphore);
            osDelay(1);
            PIN_Off(0);
            osSemaphoreWait(semaphore, osWaitForever);
            PIN_On(0);
#elif (RTOS_TEST == TEST4)
            osDelay(1);
            PIN_Off(0);
            osSemaphoreRelease(semaphore);
            PIN_On(0);
            osSemaphoreWait(semaphore, osWaitForever);
#else
            osDelay(1);
#endif
      }
}
void Task2 (void *argument)
{
      while (1)
      {
#if (RTOS_TEST == TEST1)
            osDelay(1);   // wait 1ms
            PIN_Off(0);
            osSemaphoreRelease (semaphore);
#elif (RTOS_TEST == TEST2)
            PIN_On(0);
```

```
#elif (RTOS_TEST == TEST3)
          osDelay(1);
#elif (RTOS_TEST == TEST4)
          osDelay(1);
#else
          osDelay(1);
#endif
     }
}
```

References

1. Richard, Z. *The Industrial Information Technology Handbook*, 1st ed.; CRC Press: Boca Raton, FL, USA, 2018; 1936p.
2. Anh, T.N.B.; Tan, S. Real-Time Operating Systems for Small Microcontrollers. *IEEEMicro* **2009**, *29*, 30–45. [CrossRef]
3. Cinque, M.; De Tommasi, G. Work-in-Progress: Real-Time Containers for Large-Scale Mixed-Criticality Systems. In Proceedings of the 2017 IEEE Real-Time Systems Symposium (RTSS), Paris, France, 5–8 December 2017; pp. 369–371.
4. Nuratch, S. A universal microcontroller circuit and firmware design and implementation for IoT-based realtime measurement and control applications. In Proceedings of the 2017 International Electrical Engineering Congress (iEECON), Pattaya, Thailand, 8–10 March 2017; pp. 1–4.
5. Zikria, Y.B.; Kim, S.W.; Hahm, O.; Afzal, M.K.; Aalsalem, M.Y. Internet of Things (IoT) Operating Systems Management: Opportunities, Challenges, and Solution. *Sensors* **2019**, *19*, 1793. [CrossRef] [PubMed]
6. Buttazzo, G.C. *Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications*; Springer: Berlin/Heidelberg, Germany, 2011.
7. De Oliveira, D.B.; de Oliveira, R.S. Timing analysis of the PREEMPT_RT Linux kernel. *Softw. Pract. Exp.* **2016**, *46*, 789–819. [CrossRef]
8. Zhang, X.; Lu, F.; Cui, M. A review of OSEK/VDX application verification methods. In Proceedings of the 5th International Conference on Communication and Information Processing (ICCIP'19). Association for Computing Machinery, New York, NY, USA, 15–17 November 2019; pp. 37–42.
9. Delgado, R.; Park, J.; Choi, B.W. Open Embedded Real-time Controllers for Industrial Distributed Control Systems. *Electronics* **2019**, *8*, 223. [CrossRef]
10. Belesioti, M.; Makri, R.; Fehling-Kaschek, M.; Carli, M.; Kostopoulos, A.; Chochliouros, I.P.; Neri, A.; Frosali, F. A New Security Approach in Telecom Infrastructures: The RESISTO Concept. In Proceedings of the 2019 15th International Conference on Distributed Computing in Sensor Systems (DCOSS), Santorini Island, Greece, 29–31 May 2019; pp. 212–218.
11. Zhang, K.; Wu, J.; Liu, C.; Ali, S.S.; Ren, J. Behavior Modeling on ARINC653 to Support the Temporal Verification of Conformed Application Design. *IEEE Access* **2019**, *7*, 23852–23863. [CrossRef]
12. Alexander, B.; Titarenko, L.; Mazurkiewicz, M. *Foundations of Embedded Systems*; Springer: Berlin/Heidelberg, Germany, 2019.
13. Ungurean, I.; Gaitan, N.C.; Gaitan, V.G. A Middleware Based Architecture for the Industrial Internet of Things. *KSII Trans. Internet Inf. Syst.* **2016**, *10*, 2874–2891.
14. Ungurean, I.; Gaitan, N.C. Performance analysis of tasks synchronization for real time operating systems. In Proceedings of the 2018 International Conference on Development and Application Systems (DAS), Suceava, Romanian, 24–26 May 2018; pp. 63–66.
15. Labrosse, J.J. *MicroC/OS II*; The Real Time Kernel; CMP Books: Lawrence, KS, USA, 2002.
16. Embedded Markets Study, Integrating IoT and Advanced Technology Designs, Application Development & Processing Environments. 2017. Available online: https://m.eet.com/media/1246048/2017-embedded-market-study.pdf (accessed on 14 February 2020).

17. Real-Time Operating Systems (RTOS) Market 2019: Highlights Recent Trends, Market Growth, Top Manufacturers Analysis, Business Opportunities and Demand. Available online: https://www.marketwatch.com/press-release/real-time-operating-systems-rtos-market-2019-highlights-recent-trendsmarket-growthtop-manufacturers-analysisbusiness-opportunities-and-demand-2019--09-09 (accessed on 14 February 2020).
18. Guillaume, C.; Dagenais, M. Benchmarking Real Time Operating Systems. 2019. Available online: https://amdls.dorsal.polymtl.ca/system/files/RTOS%20%20Benchmarking.pdf (accessed on 14 February 2020).
19. Lee, J.; Kim, S.C.; Woo, D.; Ma, Y.; Mah, P. Performance comparison of MRTOS and VxWorks in MultiBench benchmark suite. In Proceedings of the 2017 19th International Conference on Advanced Communication Technology (ICACT), Bongpyeong, Korea, 19–22 February 2017; pp. 871–873.
20. Pinto, M.L.; de Oliveira, A.S.; Wehrmeister, M.A. Real-Time Systems Evaluation for Robotics Using the Hart-ROS Benchmark. In Proceedings of the 2019 19th International Conference on Advanced Robotics (ICAR), Belo Horizonte, Brazil, 2–6 December 2019; pp. 290–295.
21. Renaux, D.P.B.; Pöttker, F. Performance evaluation of CMSIS-RTOS: Benchmarks and comparison. *Int. J. Embed. Syst.* **2016**, *8*, 452–463. [CrossRef]
22. Boltov, Y.; Skarga-Bandurova, I.; Kotsiuba, I.; Hrushka, M.; Krivoulya, G.; Siriak, R. Performance Evaluation of Real-Time System for Vision-Based Navigation of Small Autonomous Mobile Robots. In Proceedings of the 2019 10th International Conference on Dependable Systems, Services and Technologies (DESSERT), Leeds, UK, 5–7 June 2019; pp. 218–222.
23. Why RTOS and What Is RTOS? Available online: https://www.freertos.org/about-RTOS.html (accessed on 14 February 2020).
24. Real-Time Kernels: µC/OS-II and µC/OS-III. Available online: https://www.micrium.com/rtos/kernels/ (accessed on 14 February 2020).
25. RTX v5 Implementation. Available online: http://arm-software.github.io/CMSIS_5/RTOS2/html/rtx5_impl.html (accessed on 14 February 2020).
26. RT-Thread, RTOS. Available online: https://www.rt-thread.org/ (accessed on 14 February 2020).
27. Why Certification? Micrium. Available online: https://www.micrium.com/certification/why-safety-critical-certification/ (accessed on 14 February 2020).
28. Wang, M.; Zhang, Z.; Li, K.; Zhang, Z.; Sheng, Y.; Liu, S. Research on key technologies of fault diagnosis and early warning for high-end equipment based on intelligent manufacturing and Internet of Things. *Int. J. Adv. Manuf. Technol.* **2019**, 1–10. [CrossRef]
29. Guan, F.; Peng, L.; Perneel, L.; Timmerman, M. Open source FreeRTOS as a case study in real-time operating system evolution. *J. Syst. Softw.* **2016**, *118*, 19–35. [CrossRef]
30. Musyafani, M.I.; Ardilla, F.; Bachtiar, M.M. Architecture design of low level control omni directional robot with RTOS-RTX arm cortex-M4. In Proceedings of the 2016 International Electronics Symposium (IES), Denpasar, Indonesia, 29–30 September 2016; pp. 191–196.
31. Liu, K.; Li, X.; Wang, T.; Zhang, Y.; Li, L. New Electromagnetic Transmitter for EM-MWD System Based on Embedded RTOS: Uc-OS III. In Proceedings of the 2019 IEEE 4th International Conference on Signal and Image Processing (ICSIP), Wuxi, China, 19–21 July 2019; pp. 299–303.