International Journal of Computing and Digital Systems
ISSN (2210-142X)
Int. J. Com. Dig. Sys. 6, No.4 (July-2017)
http://dx.doi.org/10.12785/ijcds/060405

Real-Time Operating System FreeRTOS Application for Fire Alarm Project in Reduced Scale

Luca de Oliveira Turci[1]

[1] *Department of Control and Automation Engineering, Universidade Federal de Ouro Preto, Ouro Preto, Brazil*

Received 1 Feb. 2017, Revised 8 Apr. 2017, Accepted 19 Jun. 2017, Published 1 July 2017

Abstract: Uncontrolled fires are responsible for many harmful damages and human losses. Hence, the design of a system in order to reduce such problems may be considered relevant. The use of real-time operating systems can present a possibility to achieve significant results and improvements in fire alarm projects, considering that fires are examples of critical time systems in which the time of response is extremely important. In the proposed project, an algorithm was developed by using Arduino Nano and FreeRTOS open source kernel in order to accomplish such task in a reduced scale project. Some tests, such as jitter, latency, and worst case response time are also carried out to evaluate the performance of the real-time system proposed in the present project.

Keywords: Arduino, Fire alarm, Fire sensor, FreeRTOS, Real-time system operating, Jitter, Latency

1. INTRODUCTION

Since the early days of the mankind, fire has been an important source of security and comfort for the human race. Humans have been using fire to cook food, keep warm, and light up [1]. Moreover, fire has also been massively used in the industry and during the power generation in combustion processes, which are responsible for a wide range of energy transformation in the world.

However, uncontrolled fires are considered as one of the most common causes of deaths and losses nowadays. An uncontrolled fire can easily spread itself and cause a total catastrophe, devastating everything ahead [2]. According to [3], 37.000 fires at industrial or manufacturing properties were reported each year in U.S. fire departments during 2009-2013, leading 18 civilian to deaths, 279 civilian injuries, and $1 billion in losses.

Fire may be considered as a fast form of energy and self-sustained oxidation process that emits heat and light in varying intensities. The chemical reactions and changes of the state in the weak double bond of molecular oxygen (O_2), to the stronger bonds in the combustion products carbon dioxide (CO_2) and water (H_2O) release this hot and fast form of stored chemical energy [1].

A wide variety of factors may lead to a quick emergence of uncontrolled fires in the industrial environment. Situations as high fuel loads per unit of area, transport of toxic products, leakage of flammable liquids or gases, greasy or dusty electric motors or machines, overloaded outlets, broken power tools, and exposed wiring are able to ignite easily with the least carelessness.

Analogously, a fire may be considered a system which should be replied in a critical time due to the threats it may cause. Real-time operating systems (RTOS) are time dependent systems that have to respond to external or internal reactions in a particular fraction of time. In other words, RTOS must process the input data and give the desirable output within a stipulated time. RTOS are divided into two groups of systems: hard real-time systems and soft real-time systems.

In soft real-time systems, the deadline is not compulsory for every task every time. However, the system cannot miss the deadline for every task, since the performance of the system would become impaired. The best examples of soft real-time systems are videos and audio calls from internet users.

In hard real-time systems, such as fire alarm systems, medical monitoring systems, and remote satellite imaging systems, the system is purely deterministic and limited by the time. If the deadline or time of response is missed, the consequences may be irreversible and the system performance will fail [4].

RTOS have been used in plenty of embedded system for years in the areas of Automation and Computer Science. Such systems have been developed to support embedded algorithms in military devices, defense systems, and softwares used to control large switching

E-mail address: gpv.automacao@gmail.com

systems. RTOS are normally implemented to be used in hard real-time systems [5].

The development of RTOS has increased quickly over the last decades. The applications of these systems are considered robust and always represent a challenge for designers. These systems must guarantee satisfying timing constraints whereas executing complex tasks [6], so they have to be well designed.

There are plenty of fire alarm projects installed in buildings and industries, including different devices and technologies. However, these projects are often outdated and require improvements in order to obtain a better execution. In the present project, a hard real-time system for fire alarm and fire suppression is proposed in a reduced scale to improve the performance of the time response of these systems implemented in buildings and industries. The block diagram of the fire alarm system is displayed in Fig. 1.

Such project was developed using Arduino Nano and the open source kernel RTOS FreeRTOS. An alarm, a cooler acting as an actuator, a blue LED simulating fire suppression actuator, a lighter, a YS-17 flame sensor, and a MQ-2 semiconductor sensor for combustible gas were used as well.

Some tests were accomplished in order to analyze the performance of the project and evaluate the operation of the FreeRTOS platform. Some measurement experiments as jitter, latency, and the worst case response time (WCRT) of the system were also carried out. It is important to assess these parameters, because they may help in making decisions about the scheduling of new tasks as well as ensuring that all tasks are respecting the respective deadlines.

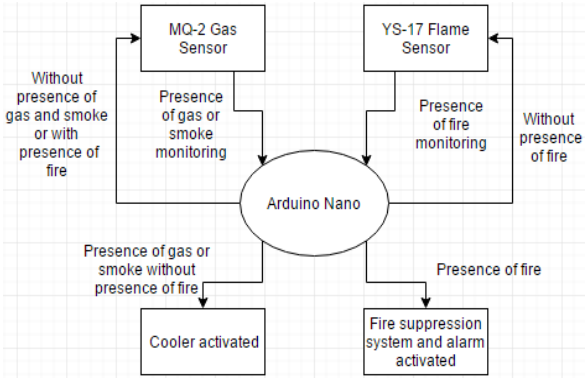


Figure 1. Block diagram of the system

The remainder of this paper is organized as follows. Section 2 shows the background and related works, section 3 describes the analysis of the system design. Section 4 explains the algorithm features, section 5 demonstrates the experiments, and section 6 displays the obtained results and the conclusion.

2. Background and Related Works

Many studies and researches in computer science, control and automation, electronics, electrical, and technology fields have been carried out in order to minimize and have a better control of the impacts caused by undesired fires. Tabirca et al. [7] used an algorithm and equations to provide a study for fire hazard safety in building environments. Wang et al. [8] used multisensory technology aiming to assist building managers in fire emergency management and fire brigades in fire rescue. Planas-Cuchi et al. [9] carried out a survey of the origin, type, and consequences of fire accidents in process plants and in the transportation of hazardous materials.

Furthermore, considering the development and improvements of fire alarm systems, many projects have also been carried out by researchers and designers. Dong et al. [10] designed a wireless automatic fire alarm system to achieve rapid fire detection and alarm and state supervision of fire-fighting facilities with low power consumption. Shu-guang [11] constructed a wireless fire alarm system based on ZigBee technology to overcome the limitations of a cable alarm system and avoid high power consumption of other wireless communication technology. Jing and Jingqi [12] deployed Bayesian network model (a graphical network based on probabilistic inference) into a fire alarm system to overcome the problem of false alarm and missing alarm caused by information uncertainty existing in the fire alarm system. Zheng et al. [13] proposed a fire detection system model and calculating model of fuzzy neural network for processing fire signal based on the characteristic of fire detection signal and the requirements of fire detection system.

Using RTOS and their applications or aiming to enhance the performance of such systems, many other studies and projects have been carried out. Qin et al. [14] used RTOS µCOS-II platform to realize the state of charge (SoC) estimation of a battery management system for lithium-Ion batteries of electric vehicles. Proctor and Shackleford [15] analyzed the impacts of jitter on stepper motor control and some techniques to reduce jitter. Salem et al. [16] deployed RTOS applications for electric control applications. Vetromile et al. [17] evaluated the pros and cons of RTOS scheduler implementation from software and hardware. Harkut and Ali [18] used RTOS and FPGA technology in order to develop a novel fuzzy logic based adaptive hardware scheduler for multiprocessor systems that minimizes the processor time for scheduling activity. Sharma, Elmiligi, and Gebali [19] explored different methods to evaluate the performance of RTOS based on the estimation of WCRT. Othman et al. [20] compared different data rate services using four scheduling algorithms in the uplink of LTE system using the uplink Vienna simulator.

3. ANALISYS OF THE CURRENT SYSTEM DESIGN

Fig. 2 displays the components of the system, the proper connections and the actual position of each device in the assembly. The list of equipment and components used in the present project to perform the tests and build the fire alarm assembly are displayed in Table I.

TABLE I. USED EQUIPMENT AND COMPONENTS

1 Arduino Nano board
1 MQ-2 sensor
1 Cooler
1 YS-17 sensor
1 Alarm
1 Blue LED
1 Lighter
1 Oscilloscope
1 Function generator

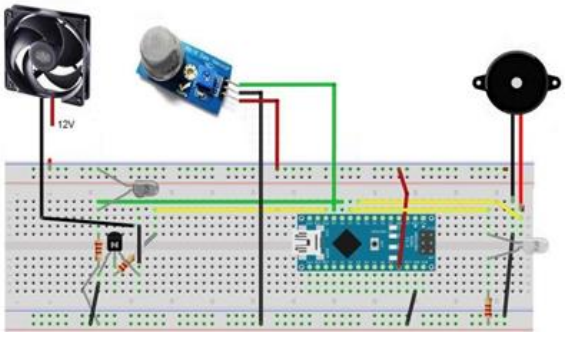


Figure 2. Fire alarm project components

A. YS-17 flame sensor

The YS-17 flame sensor is extremely sensitive to flame and radiation, so when a small flame of wavelength within the range of 760nm-1100nm approximates to this sensor, it emits a digital signal response in high level (5V) to an input pin of the Arduino Nano board. Then, an output pin of the Arduino Nano board sends a digital signal response in high level to the alarm and the blue LED. The alarm starts ringing and the blue LED starts flashing, simulating a fire suppression actuator.

YS-17 is capable of detecting flames in a radius of 100cm from its base. Fig. 3 shows two samples of this sensor.

Figure 3. YS-17 sensor samples

B. MQ-2 combustile gas sensor

Taking into account the MQ-2 combustible gas sensor, when a small quantity of gas or smoke is detected by MQ-2 sensor, it emits a digital signal with high level to an input pin of the Arduino Nano. Then, an output pin of the Arduino Nano returns a digital signal with high level to the cooler, which is turned on in order to spread off such gas.

Jiru [21], Hua et al. [22], and Min [23] suggested the design of alarm monitoring systems using MQ-2 gas sensor. This sensor is used for gas leakage detection in housing and industry, as it is suitable for detecting of LPG, butane, propane, methane, alcohol, hydrogen, and smoke. MQ-2 is capable of distinguishing gases from smokes, generating different output voltages. Fig. 4 displays a sample of MQ-2 sensor.

Figure 4. MQ-2 sensor sample

C. Arduino Software and Hardware

Arduino is an open-source platform used for many designers. The microcontroller Arduino is widely used in many control systems because of its range of advantages such as feasibility, simplicity, ease of use, cost, and size [24]. The Arduino may be programmed in simplified languages, such as C/C++ and has a friendly integrated development environment (IDE) which is able to run on all major operating systems. Further, this device may compute approximately 300,000 lines of program code per second and there is a large online community with a great amount of accessible knowledge where worldwide users enable rapid prototyping and debugging. It is also possible to find custom Arduino libraries and useful support for the platform in its community [25].

Taking into account its low cost and flexibility, the Arduino Nano board was selected to be the brain of the present project. Displayed in Fig. 5, this device is used to control the fire alarm project and it was programmed for working with 0V in low state and 5V in high state. The Arduino Nano is based on the ATmega328 and it is compatible with the FreeRTOS platform.

Figure 5. Arduino Nano board

D. FreeRTOS Overview

FreeRTOS is a popular open source RTOS kernel on top of which embedded applications can be built to meet their hard real-time requirements. Such kernel receives over 75,000 downloads per year [26]. By using FreeRTOS, it becomes possible to organize applications

as a collection of independent threads (or tasks) of execution. On a single processor (only one core), only one thread may be processed at any time. Then, the scheduler of this kernel is responsible for deciding which thread should be processed by examining the priority assigned to each thread. This kernel is designed to be small, simple, easy to port, and maintainable. Some of the main features of FreeRTOS are described [27].

- Very small memory footprint, low overhead, and very fast execution.
- Ideally suited to embedded real-time applications that use microcontrollers;
- Tick-less option for low power applications;
- FreeRTOS may be assigned as a hard real-time system or soft real-time system by selecting appropriate task and interrupt priorities.
- The scheduler may be preemptive or cooperative.
- Asynchronous events may invoke scheduler decision points.
- The scheduler algorithm is highest priority first and scheduler decision points also occur at regular clock frequency.
- FreeRTOS allows the creation of binary semaphores.
- Tasks are either non-blocking or will block after a fixed period of time.
- Tasks are independent modules.
- Tasks may be tested in isolation.
- No processing time is wasted by polling for events that have not occurred.
- The kernel is responsible for executing timing, resulting in fewer interdependencies between modules and allowing the software to evolve in a predictable way.
- Codes are executed only when there is some event that must be done.
- The idle task is created automatically and are used to measure processing capacity or to place the processor into a low-power mode, enhancing the efficiency gain and decreasing the power consumption.
- The scheduler may be suspended;
- There are three heap models as part of distribution.
- The disabling of interrupts handles the critical section processing.
- Mutexes, recursive mutexes, counting semaphores, binary semaphores, queues, software timers, event groups, stack overflowing checking, and tick hook functions may be implemented using FreeRTOS.
- Is supplied as a set of C programming language source files.

Many examples of applications and researches using FreeRTOS are available. Qaralleh et al. [28] used FreeRTOS to exploit an ARM Generic Interrupt Controller (GIC). Rutagangibwa and Krishnamurthy [29] applied FreeRTOS applications in industrial control systems.

In the present paper, a code in terms of thread blocks independent of each other using FreeRTOS was implemented. Each thread is responsible for monitoring a respective function presented in the fire alarm project. The features of each thread are displayed and explained in topic 4 Algorithm.

4. ALGORITHM

An algorithm was developed in Arduino IDE platform using the FreeRTOS libraries. The algorithm has 4 threads and 2 binary semaphores, since 2 out of 4 threads are considered handlers that have the function of receiving the binary semaphores.

Generally, in synchronization events, an interrupt service routine (ISR) of the binary semaphores is used to unblock a task when an interrupt happens [30]. Fig. 6 shows ISR giving a binary semaphore to Task 2 in order to unblock this task and put the Task 1 in a blocked state, waiting for the semaphore to start running again.

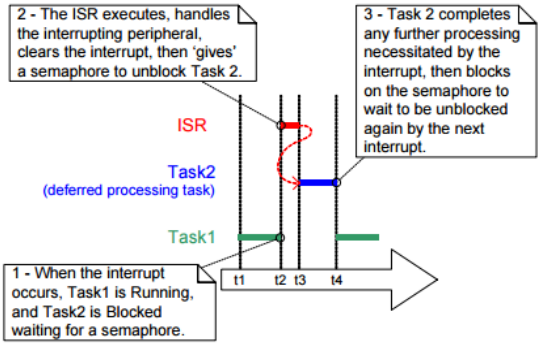


Figure 6. ISR giving a binary semaphore to Task 2 [30]

A. Threads

Threads in FreeRTOS are called tasks. Each thread is a small program in its own right. Normally, a thread runs forever within an infinite loop and must not be allowed to return from their implementing function. The threads handled by FreeRTOS are normally characterized by stack size and priority [30].

The scheduler used in FreeRTOS provides determinism by allowing the user to assign the priority of each thread. Hence, the scheduler handles the threads using the priority to know which thread will run next. The scheduler will always process the highest priority thread which is able to run within a defined period of time. In

this project, the scheduler is based on preemptive scheduling.

A task is considered in the blocked state always when it is waiting for an event. There are two ways in which tasks may enter in the blocked state, by temporal events or synchronization events. Temporal event is considered as a delay period expiring, or an absolute time to be reached and may be used intentionally by the designer of the algorithm, calling the function *void vTaskDelayUntil (TickType_t *pxPreviousWakeTime, TickType_t xTimeIncrement);*. The parameter *pxPreviousWakeTime* is used to set a fixed frequency at which a task will execute periodically and hold the last time that a task was left in the blocked state. The parameter *xTimeIncrement* sets the time that a task will be waiting in the blocked state. This time is converted in ticks by using the macro *pdMS_TO_TICKS()*.

The prototype *void ATaskFunction (void *pvParameters)* is responsible for implementing the task whereas the function *xTaskCreate() API* is used to create a new task. I developed the 4 tasks as follows:

1) *xTaskCreate(vHeatSensor, "Thread 1", 200, NULL, 1, NULL)*; This task has priority 1, the highest priority of the system and is responsible for detecting the presence of fire in the system as well as giving the Binary Semaphore 1 to the ISR in order to activate the Handler 1. This thread must occupy the CPU immediately and start running whenever some presence of fire is detected by the YS-17 flame sensor. After giving the Binary Semaphore 1 to the ISR, this task is placed into the blocked state for 500 ms, using the function *vTaskDelayUntil(&xLastWakeTime,(500/ portTICK_PERIOD_MS));*.

2) *xTaskCreate (xGasSensor, "Thread 2", 200, NULL, 2, NULL);* This task has priority 2 in the system and is responsible for monitoring the presence of gas or smoke. This thread is responsible for giving the Binary Semaphore 2 to the ISR in order to activate the Handler 2. The Thread 2 is enabled since the MQ-2 gas sensor detects some presence of gas or smoke in the system. However, as Thread 1 has the highest priority of the system, Thread 1 is the most important thread and must be running whenever some presence of fire is detected. In this case, even if some presence of gas or smoke is detected, Thread 2 would not be able to run, because only one thread should occupy the CPU any time. After giving the Binary Semaphore 2 to the ISR, this task is placed into the blocked state for 500 ms, using the function *vTaskDelayUntil(&xLastWakeTime, (500 / portTICK_PERIOD_MS));*.

3) *xTaskCreate (vFireSuppression, "Handler 1", 200, NULL, 2, NULL);* This task has priority 2 and is responsible for taking *xBinarySemaphore* from the ISR to activate the alarm and turn on the blue LED indicating the presence of fire in the system.

4) *xTaskCreate (vActivateAlarm, "Handler 2", 200, NULL, 3, NULL);* This task has priority 3 and is responsible for taking *xBinarySemaphore2* from the ISR and activating the cooler to spread the gas.

B. Binary Semaphores

Basically, the binary semaphore may be considered as a queue with a length of one. This queue may be either full or empty. Hence, the name binary emerges. When ISR gives the binary semaphore to the task, the queue is considered full. Otherwise, when ISR takes the semaphore from the task, the queue is considered empty. When a queue is considered full, only the task that has the binary semaphore may be running, unless a task with higher priority causes an interrupt process and immediately owns license to occupy the CPU. On the other hand, when a queue is considered empty, any task may start running any time it is called by an event. The same process occurs in the algorithm of the current project, when two binary semaphores are handled by ISR.

In Fig. 7, the Task 2 remains waiting for the semaphore, which is given by Task 1. When an interrupt occurs, the semaphore is finally given and Task 2 start running. An interrupt happens when determined event is recognized by the system. For instance, the presence of fire or the presence of smoke or gas detected by the sensors are events that cause interrupts.

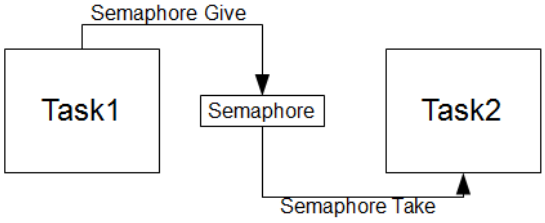


Figure 7 . Semaphore application

The binary semaphores were implemented as follows:

1) *vSemaphoreCreateBinary(xBinarySemaphore);* This semaphore is responsible for activating the Handler 1. ISR always gives this binary semaphore to Handler 1 whenever some presence of fire is detected by the sensor of flame in the Thread 1.

2) *vSemaphoreCreateBinary(xBinarySemaphore2);* This semaphore is responsible for activating the Handler 2. ISR gives this binary semaphore to Handler 2 whenever some presence of gas or smoke is detected by the sensor of gas in the Thread 2, unless there is some presence of fire in the system, because Thread 1 has the highest priority in the system and must occupy the CPU and start running immediately.

5. EXPERIMENTS AND RESULTS

The analysis of the performance of a RTOS is extremely important and may not be a trivial assignment. According to [31], it is not possible to measure the features of a RTOS with reliability without an external device. A qualitative analysis of the performance of FreeRTOS kernel was carried out.

The worst case response time of a task and worst case response time of an interrupt are considered together the most important characteristic of a real-time system. Besides that, the most important specification of a RTOS is the amount of time that the interrupts are disabled, so interrupt latency is a component which must be analyzed to obtain a good idea of the real-time capabilities [31].

According to Barabanov [6], Aroca and Caurin [31], Ganssle [32], and Franke [33] a PC parallel port is used to obtain an interrupt and give a response to this interrupt, as one of the most accurate methods used to measure execution time through output ports. An external signal generator varying in frequency from 1kHz to 30kHz is used to generate an external stimulus and an oscilloscope is also used in order to measure the latency to handle interrupts, measure the jitter to find a random variation from one latency measurement to another one, and obtain the worst case response time to evaluate the quality of the system, as shown in Fig. 8.

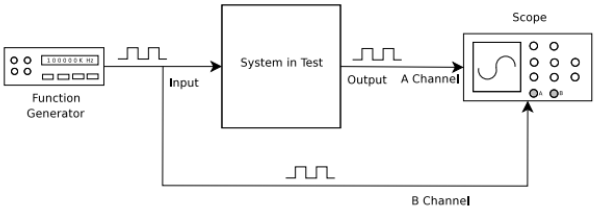


Figure 8. Experiment setup [31]

According to Karim, Prevost, and Rad [34], latency may be considered as the time difference between the moment that an interrupt happens and the moment that a response is generated from the associated interrupt handler. This parameter is analyzed externally, considering the RTOS under test in conjunction with the external equipment used. High latencies are not desired and may debilitate the system performance.

Jitter may be considered as a random variation between each latency value and is obtained from several latency measurements. The jitter is a parameter that can cause a notorious impact in a RTOS. In order to figure out the jitter, the time variance between two consecutive latency measurements is calculated. The greatest value encountered is pointed out as the worst jitter of the system.

WCRT may be described as the inverse of the maximum interruption frequency with reliability. In order to obtain such parameter, the input signal frequency from the function generator should be slightly incremented and the output signal should start changing. Then, the maximum interruption frequency that is readable on the oscilloscope screen with reliability is used to calculate the WCRT.

In total, 50 independent samples of measurement were taken. The Fig. 9 and Fig. 10 show the results of latency and WCRT, respectively, for most significant sample that has been obtained.

The Table II displays the results obtained during the tests accomplished using FreeRTOS.

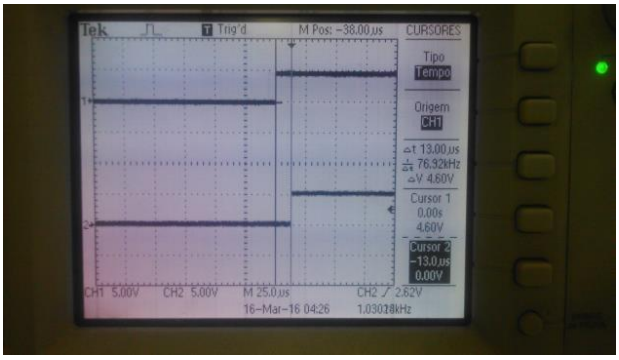


Figure 9. Obtained results for latency

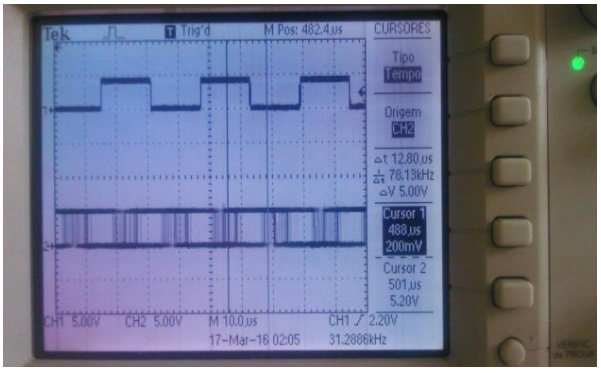


Figure 10. Obtained results for WCRT

TABLE II. OBTAINED RESULTS FOR THE MOST SIGNIFICANT SAMPLE USING FREERTOS

Jitter	Latency	WCTR (*1/f*)
25 µS	13µS	3.19×10^{-5}s

The values showed in the Table II are considered within the criteria that defines a hard real-time system as a system which has a jitter no higher than 100µs in tasks that has cycles of up to 10ms, according to OMAC (Open Modular Architecture for Control) user group [35].

6. CONCLUSION

In the present project, the performance of the FreeRTOS kernel using Arduino Nano board was analyzed together with external equipments (oscilloscope and function generator) and some practical methods through several parameters. The explanation about the importance of using RTOS in critical time systems as fire alarm projects was also accomplished and proved.

Considering the circumstances of the experiments, it is possible to conclude that the FreeRTOS kernel really presented determinism and reliability. Such kernel may be considered useful and flexible with free open source libraries. The FreeRTOS libraries are written in C programming language, being very easy to become acquainted with.

The kernel presented satisfactory results, meeting temporal requirements within the criteria that defines a hard real-time system as a system with a jitter no higher than 100µs in tasks with cycles up to 10ms [35]. The tests also showed that FreeRTOS has a good task scheduler, presenting little changes in the measurements with very low times.

The embedded system Arduino Nano demonstrated a good performance with simplicity, low cost, reliability, and feasibility. The board has a very small size, demonstrating a good capability of being installed in small fire alarm projects. As displayed in the experiments, this device may be considered useful for the proposed application and other applications involving RTOS.

The MQ-2 sensor presented enough performance, being able to differentiate gases from smoke. The YS-17 flame sensor also presented a good acting, respecting the time and deadlines of the respective threads.

Finally, it is possible to conclude that there is a very rich field involving RTOS applications for embedded tasks. RTOS are also very important to handle critical time systems, specially for hard real-time systems, as fire alarm project, which becomes useless after missing the deadline.

REFERENCES

[1] E. K. Addai, S. K. Tulashie, J. Annan, I. Yeboah, "Trend of fire outbreaks in Ghana and ways to prevent these incidents," Safety and Health at Work, vol. 7, iss. 4, pp. 284-292, Dec. 2016.

[2] P. Navitas, "Improving resilience against urban fire hazards through environmental design in dense urban areas in Surabaya, Indonesia," Procedia - Soc. Behav. Sci., vol. 135, pp. 178-183, Aug. 2014.

[3] R. Campbell, "Fires in industrial or manufacturing properties," Nat. Fire Protection Assoc., Apr. 2016.

[4] A. Damm, J. Reisinger, W. Schwabl, H. Kopetz, "The real-time operating system of MARS," ACM SIGOPS Oper. Sys. Review, vol. 23, iss. 3, pp. 141-157, Jul. 1989.

[5] D. Culbert, "System and method for dynamic resource management across tasks in real-time operating systems," U.S. Patent 5 838 968, Nov. 17, 1998.

[6] M. Barabanov, "A linux-based real-time operating system ," M.S. thesis, Comp. Science, New Mexico Inst. of Mining and Technol., Socorro, New Mexico, 1997.

[7] S. Tabirca, L. T. Yang, T. Tabirca, "Fire hazard safety optimization," in Int. Conf. Comput. Sci., ICCS 2015, Procedia Comput. Sci., vol. 51, pp. 2759-2763.

[8] X. Wang, S. Lo, H. Zhang, W. Wang, "A novel conceptual fire hazard ranking distribution system based on multisensory technology ," in 2013 Int.Conf. on Performance-based Fire and Fire Protection Eng., pp. 567-576.

[9] E. Planas-Cuchi, H. Motiel, J. Casal, "A survey of the origin, type and consequences of fire accidents in process plants and in the transportation of hazardous materials," Trans. IChemE, vol. 75, Part B, pp. 3-8, Feb. 1997.

[10] W. Dong, L. Wang, G. Yu, Z. Mei, "Design of wireless automatic fire alarm system," Procedia Eng., vol. 135, pp. 413-417, Feb. 2016.

[11] M. Shu-guang, "Construction of wireless fire alarm system based on Zigbee technology," Procedia Eng., vol. 11, pp. 308-313, May 2011.

[12] C. Jing, F. Jigqi, "Fire alarm system based on multi-sensor bayes network," vol. 29, pp. 2551-2555, Feb. 2012.

[13] Q. Yu, D. Zheng, Y. Fu, A. Dong, "Intelligent fire alarm system based on fuzzy neural network," in Int. Sys. App., 2009. ISA 2009. Int. Work. on IEEE, May 2009.

[14] H. He, H. Qin, Y. Shui, K. Oleksandr, "Lithium-ion battery SoC estimation with UKF and RTOS µC/OS-II platform," in Int. Conf. on Applied Energy, ICAE 2014, vol. 61, pp. 468-471.

[15] F. M. Proctor, W. P. Shackleford, "Real-time operating system timing jitter and its impact on motor control," Proceedings of the SPIE Sens. Cont. Intell. Manuf.II, vol. 4563, pp. 10-16, Oct. 2001.

[16] A. K. B. Salem, S. B. Othman, H. Abdelkrim, S. B. Saoud, "RTOS for SoC embedded control applications, " in Des. Technol. Integra. Sys. Nanoscale Era, 2008.

[17] M. Vetromile, L. Ost, C. A. M. Marcon, C. Reif, F. Hessel, "RTOS scheduler implementation in hardware and software for real time applications, " in Rapid System Prototyping, 2006. Seventeenth IEEE Int. Workshop.

[18] D. G. Harkut, M. S. Ali, "Adaptive fuzzy hardware scheduler for real time operating system, " Int. J. Com. Dig. Sys. 5, no. 6, Nov. 2016.

[19] M. Sharma, H. Elmiligi, F. Gebali, "Performance evaluation of real-time systems, " Int. J. Com. Dig. Sys. 4, no.1, Jan. 2015.

[20] A. Othman, S. Y. Ameen, A. Al-Omary, H. Al-Rizzo, "Comparative performance of subcarrier schedulers in uplink LTE-A under high users' mobility, " Int. J. Com. Dig. Sys. 4, no. 4, Oct. 2015.

[21] P. Jiru, "Design of intelligent and monitoring system," in Instrumentation, Measurement, Computer, Communication and Control (IMCCC), 2013 Third International Conf. on.

[22] W. Hua, W. Cheng, Z. Guangyuan, F. Weidan, "Design of distributed wireless security alarm system," in Computer Sci. & Serv. Sys. (CSSS), 2012 Int.Conference on.

[23] L. Min, "The design of SMS alarm dystem on CORTEX M3 + SIM900A," Robots & Intelligent Sys. (ICRIS), 2016 Int. Conf. on.

[24] Q. A. Al-Haija, M. A. Tarayrah, H. Al-Qadeeb, A. Al-Lwaimi, "A tiny RSA cryptosystem based on Arduino microcontroller useful for small scale networks, " Procedia Comuter Sci., vol. 34, pp. 639-646, 2014.

[25] A. S. Ali, Z. Zanzinger, D. Debose, B. Stephens, "Open source building science sensors (OSBSS): a low-cost Arduino-based platform for long-term indoor environmental data collection, " Build. and Environ., vol. 100, pp. 114-126, May 2016.

[26] J. Mistry, M. Naylor, J. Woodcock, "Adapting FreeRTOS for multicore: an experience report," in Wiley InterScience. [Online]. Available: https://www.cs.york.ac.uk/fp/multicore-freertos/spe-mistry.pdf

[27] R. Goyette, "FreeRTOS Overview," in An Analysis and Description of the Inner Workings of the FreeRTOS, Ottawa, Canada: Carleton Univ., pp. 2-4, Apr. 2007.

[28] E. Qaralleh, D. Lima, T. Gomes, A. Tavares, S. Pinto, "HcM-FreeRTOS: hardware-centric FreeRTOS for ARM multicore," in Emerging Techn. & Factory Automation (ETFA), 2015 IEEE 20th Conf.on.

[29] V. Rutagangibwa, B. Krishnamurthy, "A survey on the implementation of real time systems for industrial automation aplications," IJIRST –Int. J. Innovative Res. in Sci. & Techn., vol. 1, iss. 7, pp. 174-177, Dec. 2014.

[30] R. Barry, "Resources manegement, " in Using the FreeRTOS Real Time Kernel: a Practical Guide, Renesas RX600 ed., Real Time Engineers Ltd., 2011.

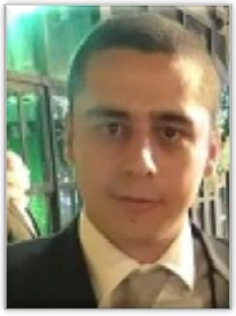
[31] R. V. Aroca, G. Caurin, "A real time operating systems (RTOS) comparison, " Universidade de São Paulo (USP), São Carlos, Brazil. [Online]. Available: http://www.lisha.ufsc.br/wso/wso2009/papers/st04_03.pdf

[32] J. Ganssle, "The firmware handbook," 1st ed., Elsevier, 2004.

[33] M. Franke, "Seminar paper: a quantitative comparison of realtime linux solutions, " Germany: Chemnizt Univ. Techn., Department of Comput. Sci., Mar. 2007.

[34] S. M. A. Karim, J. J. Prevost, P. Rad, "Efficient real-time mobile computation in the cloud using containers, " Int. J. Com. Dig. Sys. 5, no. 1, Jan. 2016.

[35] J. Hatch, "Windows CE real-time performance architecture, " In Windows Hardware Eng. Conf., 2006.



Luca de Oliveira Turci received the Control and Automation Engineering degree from Universidade Federal de Ouro Preto, Ouro Preto, Brazil in 2016.
He was a member of Control and Automation Engineering department whereas developed this project.