Available online at www.sciencedirect.com

ScienceDirect

IFAC PapersOnLine 55-4 (2022) 292–297

Measuring the Performance of FreeRTOS on ESP32 Multi-Core

Jakub Arm *, Ondřej Baštán, Ondrej Mihálik, Zdeněk Bradáč

Department of Control and Instrumentation, Faculty of Electrical Engineering and Communication, Brno University of Technology, Brno, Czech Republic (e-mail: Jakub.Arm@vut.cz, Ondrej.Bastan@vut.cz, ondrej.mihalik@ceitec.vutbr.cz, bradac@vutbr.cz)

Abstract: Real-Time Operating System (RTOS) executing on multi-core architectures still links to a lot of unresolved issues. The article presents potential problems and discusses measurement techniques indicating the performance and determinism. The advantages and disadvantages of individual measurement approaches are discussed, which cover a range from non-invasive techniques to techniques that require the cooperation of a monitored application. On top of that, the parameters indicating the performance and properties of RTOS running on multi-core are defined. Among them, the measuring of the semaphore taking, and task period jitter are described deeply. The operations are measured on ESP32-WROOM-32 development kit equipped with XTENSA dual-core processor running FreeRTOS. The results show unexpectedly high values of the switching context time and jitter when the rescheduling to another core was forced compared to time values measured on single core. Consequently, the unexpected rescheduling to other core increases the execution time of FreeRTOS operations. The work should facilitate the improvement of FreeRTOS implementation when running on multi-core architecture.

Keywords: Multi-core, RTOS, switching context, task jitter, determinisms.

1. INTRODUCTION

With increasing demands on computing power and diversity of operations, the complexity of deployed processor structures grows. When it is no longer possible to increase the computational power by the speed of the operations performed (i.e., the processor frequency), it is a logical step to increase the number of cores (CPUs) on one processor system. The speed of a given operation does not increase linearly with the number of added CPUs, but according to Amdahl's law.

According to the DIN 44 300 (DIN 44 300, 1985), real-time operating mode of a computer system are permanently ready for the processing of data arriving from the outside, thus, its results will be available within predetermined periods of time. Classical RTOS mostly work on a single-core platform, while ensuring the pseudo-parallel running of individual threads (tasks). On multi-core architectures, there may be true parallel concurrency of tasks, resulting in multiple situations leading to incorrect system behaviour. These situations mostly stem from the fact that the system has limited resources (memory, cache, peripherals), while individual CPUs access these subsystems simultaneously. As a result, spatial and temporal anomalies arise.

In the embedded world, multiprocessor structures are used in microcontrollers or microprocessors with 32-bit or more word length. In the lower class, we often come across variants that a simpler processor with lower computing power is complemented by a more complex processor with higher performance. Thus, each processor core is designed for different types of operations (fast feedback control vs. mathematical operations with double numbers). This approach is especially striking in heterogeneous structures, so-called SoC (System on the Chip), where one CPU can be designed to run RTOS, the other CPU can run Linux OS, and signal processing can be performed using a gate array.

Deterministic multi-core RTOS scheduler algorithms already exist [Mistry et al. (2014)], which use a global queue approach or a queue for each kernel separately. Mathematically, however, determinism can only be proved on an architecture containing a CPU and main memory. However, most microprocessors contain other components, such as cache memory (L1 and L2), flash accelerator, branch prediction subsystem, multi-stage pipeline, and even an advanced bus matrix. All these subsystems accelerate computing power, but increase the complexity of evaluating deterministic behaviour, respectively, make it impossible to evaluate the operation of RTOS offline in analytical ways.

Evaluation of RTOS system properties (hardware + OS + applications) is an issue that is practically solved mainly by manufacturers in terms of quality assurance of required capabilities. This area is also researched scientifically to define valid parameters, accurate evaluation of measured parameters, determine appropriate measurement methods, and moreover, to bring mathematical proof of the reliability of the measurements on different architectures. In the commercial sphere, so far there are several benchmarks in the field of RTOS, which provide a specific view of the performance and properties of RTOS [Champagne et al.

10.1016/j.ifacol.2022.06.048

(2019)], but for deeper analysis, it is necessary to use some of the tracing techniques.

Although end-to-end measurement of RTOS operations indicates its performance and properties, these values cannot be calculated in the WCET (Worst-Case Execution Time) analysis, which was first reported in [Puschner and Goat (1989)]. More accurate values can only be obtained using statistical or hybrid method, which are based on a detailed analysis of application and target architecture [Nowotsch (2014)]. [Atanassov et al. (2001)], however, showed that the statistically calculated values of the WCET analysis may differ from the measured values (terms exhibited in Figure 1) because in calculations it is often not possible to accurately mathematically describe all aspects of a given hardware.

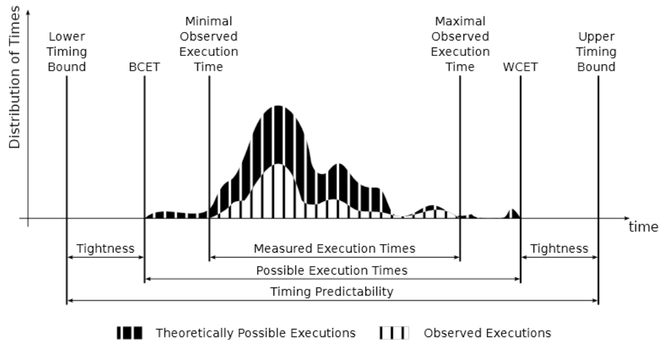

Figure 1: Timing analysis terminology [Wilhelm et al. (2008)]

2. RTOS IN MULTI-CORE

When deploying RTOS to a multi-core platform, there are multiple ways to operate RTOS depending on the architecture. These methods are divided into:

- SMP (Symmetric Multiprocessing) - indicates an architecture where multiple CPU units are connected to the main memory via a common bus. All cores are equal in access to peripherals. RTOS usually runs in only one instance and must be adapted to be able to use all cores, i.e., to run tasks on all cores. The advantage of this approach is easy scalability. The downside is the requirement to control access to shared objects (memory, disk, peripherals) and cache coherency problem.
- AMP (Asymmetric Multiprocessing) - indicates the fact that there is a hierarchical relationship between CPUs because some CPUs have access to peripherals or are superior by the OS. The parent kernel monitors the function of other kernels and determines the division of tasks. Therefore, an RTOS can only run on one or more cores in multiple instances.
- NUMA (Non-Uniform Memory Access) - indicates an architecture where some cores have faster access to the given peripherals (memory, IO). For such an architecture, RTOS can reduce the access of some cores to more distant resources, thereby optimizing computing power and avoiding interferences (they do not have to happen or rarely).

2.1 Readiness of individual representatives

RTEMS (Real time Executive for multiprocessor Systems) – is a free open-source solution embodying the monolithic kernel that supports multi-processor, TCP/IP, and filesystem while still offering minimum executable sizes below 20 kB. It adopts the POSIX (Portable Operating System Interface) standard. RTEMS organizes cores into nodes and annotates objects as global for all (local and remote) nodes or local, which are valid only for the local nodes. On Altera DE2 development board running 50 MHz Nios II/e soft-core, IRQ latency was measured as value of 0,12 ms, post object as value of 2,98 ms and receive object as value of 2,96 ms. [Dahlqvist (2009)]

FreeRTOS is an open-source RTOS originally designed for single-core architectures. Modification for multi-core architectures takes place at the community level, therefore, the implementation is not complete. Anyway, the modification is already functional; the SMP concept is supported by the RTOS. Due to openness, many scientists are also trying to create an adaption [Mistry et al. (2014)]. FreeRTOS defines its own API, while the application is transferable only to SafeRTOS.

QNX is the commercial microkernel based RTOS. It is designed to scale on ARM and x86 multicore architecture. It supports asymmetric multiprocessing and symmetric multiprocessing, as well as bound multiprocessing.

2.4 RTOS parameters

The definition of RTOS is relatively vague, therefore there is no clear definition of parameters reflecting the real-time properties of the OS. The most significant indicators are the time of context switching (or its histogram), the jitter of the periodic task, the time of response to interrupt, the time of execution of RTOS functions, the time of memory allocation and the time of memory work. By evaluating the time of these operations, it is possible to get an overview of RTOS performance on a given platform. Importantly, it is necessary to investigate whether the times of these operations are limited, which is one of the conditions of RTOS determinism.

Context switching is the most common RTOS operation [Slanina et al. (2007a)]. The operation is characterized as the time between the beginning of the preemption of one thread and the end of the resume of the other thread. The procedure consists of the following steps:

- save the current thread context on the stack,
- save the current stack pointer in the threads' control block,
- switch to the system stack pointer,
- find the highest priority thread that is ready,
- switch to the new thread's stack,
- recover the new thread's context,
- return to the new thread and its previous PC.

If the threads are located on different cores, an IPC (Inter Process Communication) operation enters the process, which is provided by a separate thread or by hardware using shared memory.

The jitter of the periodic task should be as small as possible [Slanina et al. (2017)], and its histogram must ensure little variance. The important fact is that all values are within a certain range. On a multi-core, the values may differ due to rescheduling the thread to another core [Docekal et al. (2017)]. However, even in this case, the histogram of the jitter values must satisfy the above statements.

2.5 Performance methods

From scrutiny, there are many defined ways to evaluate the performance and properties of RTOS [Slanina et al. (2018)]. Most approaches come from academia and are tailored to suit the use case. Over time, however, some freely available benchmarks have also stabilized, which serve as a quick insight into RTOS performance in terms of its real-time properties:

- Rhealstone (1989) - this is the average of task switching time, task preemption time, interrupt latency time, semaphore shuffling time, deadlock breaking time, and intertask message latency [Kar et al. (1989)]. Using a proper implementation, it is also useful on multi-core architecture.
- Hartstone (1990) - is a set of operational requirements for a synthetic application used to test hard real time systems under the synthetic load (Whetstone benchmark). There are 5 test series comprising of periodic tasks, harmonic frequencies, non-harmonic frequencies, harmonic frequencies with aperiodic processing, harmonic frequencies with synchronization. [Halang et al. (2000)]
- Dhrystone (1996) - detects overhead RTOS for defined workload in Round Robin, Task priority preemption, Semaphore, Memory alloc / dealloc, Interrupt latency, and Message passing scenarios [McRae (1996)]. The scenarios can be adapted on multi-core easily.
- RT-Test (2005) - measures operation times such as Message queue latency, Semaphore latency, Mutex latency, Signal latency, Signal round trip, and Cyclic test. Currently, the Linux OS and its RT variants are supported without modification [Linux Foundation (2019)]. Using a proper implementation, it is also useful on multi-core architecture.
- Thread-Metric (2008) - a portable solution that measures times of frequent operations, such as Cooperative context switch, Preemptive context switch, Interrupt processing, Message passing, Semaphore processing, and Memory alloc / dealloc [Express Logic (2007)]. Using a proper implementation, it can be adapted to multi-core architecture due to the modifications in the portable layer.

Furthermore, the time analysis can be performed from the created model. The advantage of this approach is the ability to use a large amount of computing power, i.e., to achieve a more accurate result. On the other hand, the disadvantage is the creation of such a model that the obtained results are credible. The process of modelling can be automated; thus, a timing model might be automatically derived from measurements on the hardware using methods from automata learning. [Reineke (2017)]

2.6 RTOS application monitoring techniques

Application Level Tracing is a feature of ESP-IDF providing the features of the fast log data transmission from ESP32 to a host via UART or JTAG interface with minimal overhead on a program execution. The library supports some levels of tracing from the sending application specific data in binary or textual representation, through the FreeRTOS events tracing, to the source code coverage using Gcov. When UART is not sufficient due to its low speed (up to 4 Mbps but limited to 921600 bps), JTAG interface can be used instead; moreover, the hardware tracing buffers are used (Figure 2)

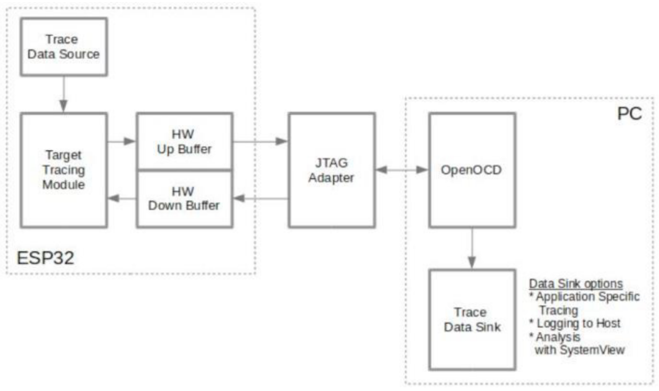

Figure 2: Tracing components using JTAG [Espressif (2021)]

Profilation is a software monitoring technique that measures the overall execution time of every software function. Evaluation might point out the infinite loops or other problems related to the increased time consuming. Profilation may be used to the measuring of the function execution time if it is linked to the execution count value.

RTOS events tracing is semi-invasive monitoring technique resulting in the sequence of executed RTOS functions (task suspended, task delay, semaphore taken, etc.). The execution time values can be calculated using timestamps. The method is powerful and reveals the incorrect function sequence. However, the RTOS must cooperate by providing event hooks. Also, the monitoring data amount grows rapidly and needs a fast communication channel to collect the data; usually, it utilizes the JTAG interface.

3. ESP32 FREERTOS MEASUREMENT

ESP32-WROOM-32E is a module containing a microcontroller embodying the XTENSA microprocessor ESP32-D0WD-V3 Dual-Core 32-bit LX6 with a 7-stage pipeline that operates at up to 240 MHz. The CPU has

CoreMark score 994.26 CoreMark; 4.14 CoreMark / MHz. The microprocessor has 448 kB ROM, 520 kB RAM, Floating Point Unit, 32 interrupt vectors, and DSP instructions, such as a 32-bit multiplier and 32-bit divider. [Espressif (2021)]

Execution time of a thread depends strongly on the context due to interference on shared resources. However, the RTOS operations must not be depended on any other execution context. In the following scenarios, the operation time will be measured using end-to-end methodology; thus, the priority and task settings must ensure that the timestamping (start and end) is executed just after without any other task.

The time measurement is burdened with the systematic errors due to the execution time of the time function and the precision. The error is minimized by utilizing the low-level assembler function which only reads the tick counter value. The function ensures the best precision of the measurement because its value is increased by every tick of the processor clock; thus, it is increased by value of 240 every microsecond.

3.1 Testing scenarios

There are idle tasks (priority 0 – the lowest) and IPC tasks (priority 24 – the highest) on each core after start. Measurement task priorities are within these limits. The idle task does not contain useful code and does not disturb in the presence of higher priority tasks. During the measurement, the watchdog tasks are switched off and the processor frequency is set to a maximum value of 240 MHz.

Task switch time will be measured using two tasks pinned to two cores. The first task has higher priority and lower period. The second task has lower priority and higher period. The measurement algorithm is as follows; the first task waits for the semaphore synchronization while the second task starts the measurement and signals the semaphore. The Figure 3 exhibits the measurement algorithm in more details.

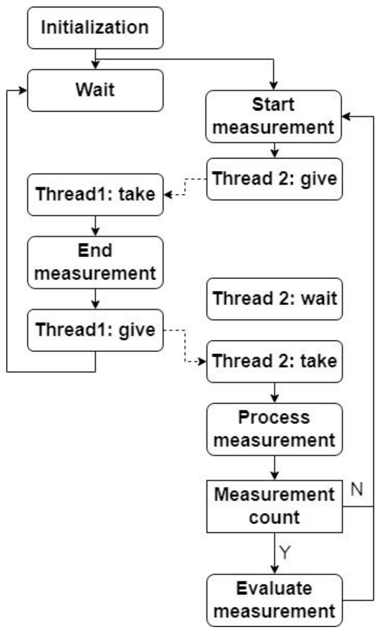

Figure 3: The measurement algorithm of the task switch

Periodic task jitter stems for the range of the periodic task measuring. The periodic task is created using the function *vTaskDelayUntil*, which is recommended and should ensure the most precise periodicity of the task using delay. The measurement is performed for one task that is toggled between cores using auxiliary threads with the higher priority and heavy work. The heavy work is simulated by the cycle of ten operations (multiplication) in periodic loop of 2 ms. The auxiliary threads force the scheduler to re-plan the measuring thread to the other core. The place of the measurement is situated just after the return of the delay function. Then, the auxiliary thread executions are toggled using *vTaskSuspend* and *vTaskResume* functions. The Figure 4 exhibits the measurement algorithm in more details.

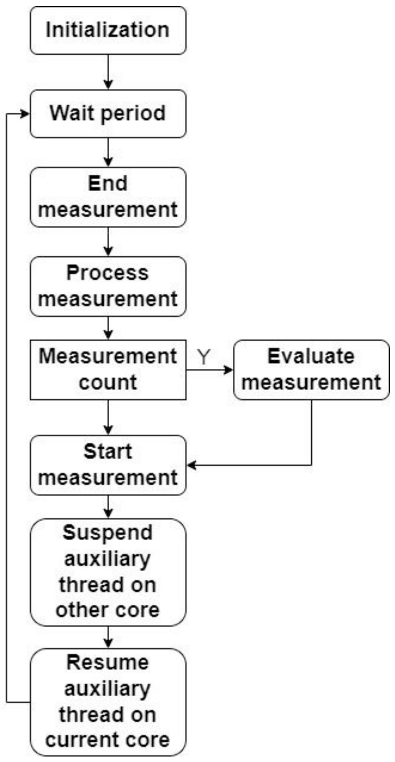

Figure 4: The measurement algorithm of the periodic task

4. RESULTS

Task switch time using the semaphore on one core is evaluated as follows:

Parameter	Value [ms]
Average	0.012
Variance	0.000
Minimum	0.012
Maximum	0.012

The results confirm the expected behaviour with a minimum variance.

Task switch time using the semaphore between cores is evaluated as follows:

Parameter	Value [ms]
Average	190.334
Variance	0.004
Minimum	190.330
Maximum	190.714

The values for the calculation of the mean and variance are converted into the data type *double*, while the minimum and maximum remain in the integer data type and are converted to *double* just before the final display.

The values are constant with very little variance during the measurement, which is probably due to the very deterministic algorithm of IPC and the fact that no other task runs except idle. On the other hand, the value of 190 ms is higher than expected. Therefore, the application may experience unexpected delays when threads are not pinned to a core because the scheduler may reschedule threads between cores randomly.

The values are subject to a systematic additive error in the form of the execution time of the give synchronization function. We also measured the function time as a value of 4.5 us. The granularity of the measurement counter is one unit; when the function is calling just after itself, the difference of the returned value is exactly one, thus, one clock tick happened (1/240 MHz).

Periodic task jitter of the task (period 10 ms) without the heavy work threads (rescheduling) is evaluated as follows:

Parameter	Value [ms]
Average	9.996
Variance	0.004
Minimum	9.632
Maximum	10.001

The results confirm the expected behaviour with a minimum variance.

Periodic task jitter of the task (period 10 ms) utilizing the heavy work threads toggling is evaluated as follows:

Parameter	Value [ms]
Average	8957.846
Variance	276.600
Minimum	213.006
Maximum	17700.333

The results show that when rescheduling a thread to a second core while waiting for periodicity, there is a significant time overhead. However, this overhead is most likely due to the nature of the auxiliary threads with the load, which can often lead to a situation that the measuring thread is being interrupted before the *vTaskDelayUntil* function is called; thus, the execution is prolonged greatly. To our knowledge, the FreeRTOS API does not yet include a function to move a task to another core. The measurement is therefore problematic due to the impossibility of ensuring such synchronization that the auxiliary thread starts to perform the load only when the measuring thread is already waiting.

5. LIMITATIONS AND FUTURE WORK

The performed measurements indicate the properties and performance of RTOS running on multi-core. The measurement took place directly on the target platform without the use of advanced monitoring techniques (e.g., in-situ monitoring) and the simulated load does not comply with any standard. These factors need to be considered when evaluating the results. In the future, we plan to use advanced monitoring techniques (profiling, RTOS function tracing) to ensure more accurate results and confirm the measurements. To achieve more accurately measure of the jitter, it will be better to use an RTOS that already provides the appropriate API function for task rescheduling or greatly improve the timing of the auxiliary tasks.

6. CONCLUSIONS

By measuring RTOS parameters on the target platform, the performance and properties can be found. There are already established approaches and benchmarks for a single-core platforms; some of these approaches can also be applied to multi-core architectures. As the most indicative operations, we identified the time of semaphore taken from one core to another and the jitter of the periodic task in the case of rescheduling to another core. We used ESP32 dual core equipped by FreeRTOS and found that the measured time values are unexpectedly higher compared to single core; however, the measurement of the task period, the time is strongly affected due to the simulated load forcing the rescheduling. From user perspective, FreeRTOS needs some improvements for multi-core usage or some of the more advanced monitoring techniques can make the measurement more trustful. Although the measured values are not fully confirmed, they point out the problem of the increased time when executing FreeRTOS on a multi-core architecture utilizing the IPC. The issue can cause unexpected problems when the user does not anticipate such RTOS behaviour (task rescheduling).

ACKNOWLEDGEMENT

The research was financially supported by Brno University of Technology. Part of the work was carried out with the support of core facilities of CEITEC – Central European Institute of Technology. This work was supported by the projects: FV40247 Cooperative robotic platforms for automotive and industrial applications; FV40196 Research and development of the monitoring of immobile persons who are tethered to the bed in terms of risk of suffocation – decubitus; FW03010100 Research and development of detection and monitoring of people with dense coverage in shopping centres in order to optimize shopping flows and, in the event of crises and epidemics, access control to these areas; FEKT-S-20-6205 Research in automation, cybernetics and artificial intelligence for Industry 4.0; 783119-1 SECREDAS Product Security for Cross Domain Reliable Dependable Automated System (H2020-ECSEL, EU). The above-mentioned grants and institutions facilitated efficient performance of the presented research and associated tasks.

REFERENCES

DIN 44 300 (1985). Informationsverarbeitung No. 9.2.11, 1985.

Wilhelm, R., Engblom, J., Ermedahl, A., Holsti, N., Thesing, S., Whalley, D., Bernat, G., Ferdinand, C., Heckmann, R., Mitra, T., Mueller, F., Puaut, I., Puschner, P., and Staschulat, J. (2008). The Worst-Case Execution Time Problem — Overview of Methods and Survey of Tools. ACM Transactions on Embedded Computing Systems (TECS), vol. 7, pp. 1–53, 2008. doi:10.1145/1347375.1347389.

Slanina, Z., & Srovnal, V. (2007a). Embedded linux scheduler monitoring. In Ieee international konference on emerging technologies and factory automation, etfa (pp. 760–763). doi:10.1109/EFTA.2007.4416851

Puschner, P., and Koza, C. (1989). Calculating the Maximum Execution Time of Real-Time Programs. Real-Time Systems Journal, pp. 159–176, 1989. doi:10.1007/BF00571421.

Nowotsch, Jan (2014). Interference-sensitive Worst-case Execution Time Analysis for Multi-core Processors. PhD thesis in Universität Augsburg.

Atanassov, P., Kirner, R., and Puschner, P. (2001). Using Real Hardware to Create an Accurate Timing Model for Execution-Time Analysis. IEEE Real-Time Embedded Systems Workshop.

Mistry, James & Naylor, Matthew & Woodcock, Jim. (2014). Adapting FreeRTOS for multicores: An experience report. Software: Practice and Experience. 44. 10.1002/spe.2188.

Docekal, T., & Slanina, Z. (2017). Control system based on freertos for data acquisition and distribution on swarm robotics platform. In 2017 18th international carpathian control conference, iccc 2017 (pp. 434–439). doi:10.1109/CarpathianCC.2017.797043

Champagne, G., & Dagenais, M. (2019). Benchmarking Real Time Operating Systems. Retrieved December, 2021, from: https://amdls.dorsal.polymtl.ca/files/RTOS%20%20Benchmarking.pdf

Kar, R. P., & Porter, K. (1989). Rhealstone-a real-time benchmarking proposal. Dr Dobb's Journal, 14(2), 14.

McRae, E. (1996). Benchmarking real-time operating systems. Dr Dobb's Journal, 21(5), 48-59.

Slanina, Z., & Docekal, T. (2018). Energy meter for smart home purposes. (Vol. 680, pp. 57–66). doi:10.1007/978-3-319-68324-9_7

Linux Foundation (2019). RT-Tests. Last changed March, 2019. Retrieved December, 2021, from https://wiki.linuxfoundation.org/realtime/documentation/howto/tools/rt-tests

Express Logic (2007). Measuring Real-Time Performance of an RTOS. Embedded Staff. Retrieved December, 2021, from https://www.embedded.com/measure-your-rtoss-real-time-performance/

Halang, Wolfgang & Gumzej, Roman & Colnaric, Matjaz & Druzovec, Marjan. (2000). Measuring the Performance of Real-Time Systems. Real-Time Systems. 18. 59-68. 10.1023/A:1008102611034.

Dahlqvist, Roger (2009). Predictable performance on a multiprocessor system with RTEMS. Master thesis at ICT/ECS KTH Royal Institute of Technology, Sweden. Retrieved December, 2021, from https://people.kth.se/~ingo/MasterThesis/Thesis_Dahlqvist2009.pdf

Reineke, Jan. (2017). Challenges for Timing Analysis of Multi-Core Architectures: Invited Talk at the 8th Workshop on DICE-FOPARA. Electronic Proceedings in Theoretical Computer Science. Uppsala, Sweden. 248. 4-5. 10.4204/EPTCS.248.3. Retrieved December, 2021, from http://cbr.uibk.ac.at/events/dice-fopara/slides/JR.pdf

Espressif (2021). ESP32 Series – datasheet. Retrieved December, 2021, from https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf

Mistry, James & Naylor, Matthew & Woodcock, Jim. (2014). Adapting FreeRTOS for multicores: An experience report. Software: Practice and Experience. 44. 10.1002/spe.2188.

Slanina, Z., & Docekal, T. (2017). Energy meter for charging stand in smart buildings. Journal of Telecommunication, Electronic and Computer Engineering, 9(2-5), 79–82.