OPTIMIZING REALTIME PERFORMANCE FOR EMBEDDED SYSTEM IN 3 AXIS DYANOMETER: A COMPARISION BETWEEN TECHNIQUES

TỐI ƯU HÓA HIỆU NĂNG THỜI GIAN THỰC CHO HỆ THỐNG NHÚNG TRONG THIẾT BỊ ĐO LỰC BA THÀNH PHẦN: SO SÁNH GIỮA CÁC KỸ THUẬT

Bui Long Vinh, Nguyen Ngoc Toan*, Dao Minh Tuan, Do Hoang Nam, Phan Quoc Hien, Phan Dang Hoang

School of Mechanical Engineering, Hanoi University of Science and Technology

**Email: Toan.NN232031M@sis.hust.edu.vn*

ABSTRACT

The highly accurate, high-speed acquisition of cutting force data is a fundamental prerequisite for the real-time monitoring and optimization of machining processes. To date, the predominant focus of research has been on the mechanical design and calibration accuracy of dynamometers, with comparatively limited attention devoted to the optimization of embedded data acquisition architectures.

This study addresses this gap by presenting a systematic performance comparison of four distinct data acquisition strategies implemented on a three-axis real-time force measurement system in turning process. The architectures: (i) Polling, (ii) Polling with Interrupts, (iii) a Real-Time Operating System (RTOS) with Interrupts, and (iv) an RTOS with Direct Memory Access (DMA) utilizing a double buffering mechanism were evaluated on an STM32 microcontroller platform integrated with a 24-bit ADC operating at 32 kSPS.

Performance was quantitatively assessed based on CPU utilization and sampling rate stability. The empirical results demonstrate that while conventional Polling and Interrupt-driven methods achieve the target sampling rate, they do so at the cost of nearly 100% CPU utilization. The RTOS-Interrupt architecture offered a substantial improvement, reducing the CPU load to approximately 97.51%. Most significantly, the RTOS-DMA configuration demonstrated superior efficacy, achieving an average CPU load time of 87.23% while maintaining high sampling rate stability (33,145.09 ± 36.64 samples/s).

These findings validate that the RTOS-DMA architecture provides an optimal balance between high data throughput and computational efficiency. This approach effectively decouples data acquisition from the CPU, thereby enabling parallel execution of critical tasks such as signal processing and communication, establishing it as the most suitable architecture for robust, multi-tasking smart machining monitoring systems.

Keywords: *Embedded system; Dynamometer; Turning process; Real-time operating system; Direct memory access embedded architecture; Real-time force measurement.*

TÓM TẮT

Việc thu nhận dữ liệu lực cắt với độ chính xác cao và tốc độ lớn là điều kiện tiên quyết cho công tác giám sát và tối ưu hóa quá trình gia công theo thời gian thực. Cho đến nay, phần lớn các nghiên cứu chủ yếu tập trung vào thiết kế cơ khí và độ chính xác hiệu chuẩn của thiết bị đo lực (dynamometer), trong khi việc tối ưu hóa kiến trúc thu thập dữ liệu trên hệ thống nhúng vẫn chưa được quan tâm đầy đủ.

Nghiên cứu này nhằm lấp đầy khoảng trống đó thông qua việc so sánh hiệu năng một cách có hệ thống giữa bốn chiến lược thu thập dữ liệu khác nhau, được triển khai trên hệ thống đo lực ba trục thời gian thực trong quá trình tiện. Các kiến trúc bao gồm: (i) Polling, (ii) Polling kết hợp Interrupts, (iii) Hệ điều hành thời gian thực (RTOS) kết hợp Interrupts, và (iv) RTOS kết hợp truy cập bộ nhớ trực tiếp (DMA) sử dụng cơ chế đệm kép. Các kiến trúc này được đánh giá trên nền tảng vi điều khiển STM32, tích hợp bộ chuyển đổi tương tự số (ADC) 24 bit hoạt động ở tốc độ 32 kSPS.

Hiệu năng hệ thống được đánh giá định lượng dựa trên hai tiêu chí chính: mức độ sử dụng CPU và độ ổn định của tần số lấy mẫu. Kết quả thực nghiệm cho thấy, mặc dù các phương pháp Polling truyền thống và điều khiển bằng Interrupts có thể đạt được tần số lấy mẫu mục tiêu, nhưng điều này đi kèm với mức sử dụng CPU gần như 100%. Kiến trúc RTOS kết hợp Interrupts mang lại sự cải thiện đáng kể khi giảm tải CPU xuống còn khoảng 97,51%. Đáng chú ý nhất, cấu hình RTOS–DMA thể hiện hiệu quả vượt trội, với mức tải CPU trung bình 87,23%, đồng thời duy trì độ ổn định cao của tần số lấy mẫu (33.145,09 ± 36,64 mẫu/s).

Những kết quả này khẳng định rằng kiến trúc RTOS–DMA cung cấp sự cân bằng tối ưu giữa thông lượng dữ liệu cao và hiệu quả tính toán. Cách tiếp cận này giúp tách rời quá trình thu thập dữ liệu khỏi CPU, từ đó cho phép thực thi song song các tác vụ quan trọng như xử lý tín hiệu và truyền thông, và được xác định là kiến trúc phù hợp nhất cho các hệ thống giám sát gia công thông minh, đa nhiệm và có độ tin cậy cao.

Từ khóa: *Hệ thống nhúng; Thiết bị đo lực; Quá trình tiện; Hệ điều hành thời gian thực; Truy cập bộ nhớ trực tiếp; Kiến trúc hệ thống nhúng; Đo lực thời gian thực.*

1. INTRODUCTION

Accurate and high-speed measurement of cutting forces is critical for understanding tool-workpiece interactions and optimizing machining performance. Multi-component dynamometers have been developed to capture the three orthogonal forces tangential, feed, and radial during turning operations. Rizal et al. introduced a three-axis strain-gauge-based turning dynamometer achieving a dynamic bandwidth of 766 Hz and a sensitivity of 0.001 N, enabling precise real-time observation of transient cutting forces [1]. Similarly, Yaldız and Unsal proposed a piezoelectric-based dynamometer capable of measuring forces with a bandwidth up to 1 kHz, ensuring high linearity and repeatability [2]. More recent works by Gómez et al. focused on displacement-based and low-cost dynamometers to improve portability and dynamic range [3], [4]. However, most existing studies emphasize mechanical ☞

design and calibration accuracy, while limited attention has been given to real-time data acquisition and embedded implementation for online monitoring. Moreover, prior studies in embedded real-time systems have shown that interrupt latency and DMA-based data movement play a decisive role in achieving deterministic timing, reducing CPU load, and lowering power consumption under high-throughput sensing workloads [5], [6].

Meeting real-time requirements in embedded systems demands effective scheduling and data-transfer mechanisms. Polling-based data acquisition is straightforward to implement but becomes inefficient at high sampling rates. RTOS such as FreeRTOS or µC/OS-II enable deterministic task scheduling and priority management [7], [8]. Interrupt-driven acquisition improves responsiveness by minimizing CPU (Central Processing Unit) idling, while DMA enables high-speed parallel data transfer, reducing CPU load and power consumption [9]. Y. R. Shen et al. demonstrated that combining RTOS scheduling with DMA offloading achieves significant reductions in transfer latency and jitter while maintaining stability across multi-channel sensor platforms [10]. However, existing research rarely provides comparative evaluation of different scheduling and data-transfer strategies within the same embedded system for dynamometer applications. To address this gap, this work systematically compares four architectures: (i) polling alone, (ii) polling combined with interrupts, (iii) RTOS with interrupts, and (iv) RTOS with DMA. The study assesses latency CPU utilization, and sampling rate stability during three-component force acquisition in turning, providing insights into the optimal configuration for real-time embedded dynamometer systems.

2. SYSTEM COMPONENT

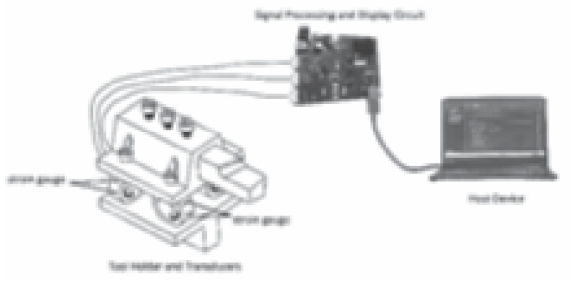

Figure 1. System component overview

The proposed system is implemented on an STM32 microcontroller platform integrated with an external precision ADC (ADS131M03) to realize a three-axis real-time measurement system operating at 32 kSPS. The hardware architecture is designed to ensure deterministic timing, high sampling accuracy, and efficient data throughput.

The ADS131M03 is a 3-channel, 24-bit delta-sigma analog-to-digital converter capable of simultaneous sampling at rates up to 64 kSPS per channel, providing synchronized acquisition of tri-axial cutting force components with high resolution and low noise (Texas Instruments, 2021). The ADC communicates with the STM32 MCU through a high-speed SPI interface configured to operate at 8 MHz. Each data frame from the ADC consists of 72 bits, corresponding to the 24-bit samples of three channels.

To efficiently handle the large data stream generated by the ADC, the STM32 microcontroller employs the Direct Memory Access (DMA) peripheral in Peripheral-to-Memory mode. This allows the DMA controller to automatically transfer incoming SPI data directly into the MCU's RAM without CPU intervention, thus freeing the Cortex-M4F/M7 core for digital signal processing (DSP) and

other real-time tasks. For data transmission to the host PC, a second DMA channel is configured in Memory-to-Peripheral mode to send processed data via a high-speed UART interface. The UART is operated at a baud rate of 3 Mbps, configured for 8 data bits, no parity, and 1 stop bit (8N1), ensuring low-latency and high-throughput data transfer.

The achievable data rate R_d can be estimated as:

$$R_d = \frac{8}{10} \times \text{Baudrate} = \frac{8}{10} \times 3{,}000{,}000 = 2{,}400{,}000 \text{ bps}$$

Since each data sample requires 72 bits, the theoretical maximum sampling throughput is:

$$f_{max} = \frac{2{,}400{,}000}{72} \approx 33.33 \text{ kSPS}$$

This value exceeds the target rate of 32 kSPS, confirming that the communication subsystem can fully sustain the required real-time acquisition performance without data loss or transmission bottlenecks. The combination of DMA-driven SPI input and DMA-driven UART output forms a fully pipelined data path, enabling near-continuous data streaming between the sensor front end and the host processor.

3. METHOD

3.1. Polling

Polling is one of the most fundamental techniques used for data acquisition and peripheral control in embedded systems such as STM32 microcontrollers. In this approach, the CPU continuously checks the status of a peripheral such as an ADC (Analog-to-digital Converter), SPI (Serial Peripheral Interface), or I²C (Inter-Integrated Circuit) to determine whether new data are available or a specific condition has been met. Once the required data are ready, the CPU retrieves them, processes the results, and then resumes other tasks such as updating an OLED display or transmitting data via UART. After completing these operations, the CPU returns to the polling loop to repeat the process. This sequential and repetitive mechanism allows straightforward control of data flow but comes with notable trade-offs in efficiency [11].

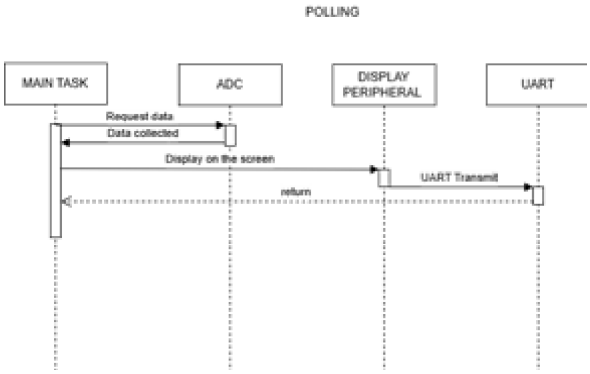

Figure 2. Sequence diagram for reading force data with polling method

The main advantage of the polling approach lies in its simplicity and transparency. Since the execution sequence follows a predictable top-down flow, debugging and verifying the correctness of program behavior are relatively straightforward. Polling is particularly suitable for simple or low-latency applications where the sampling rate is moderate, and system complexity is limited. Its deterministic nature makes it easy to reason about system timing, which is beneficial during early-stage development and testing. However, polling suffers from significant inefficiencies in CPU utilization. Because the processor must continuously check whether new data are available, it effectively remains active 100% of the time, even during idle or waiting periods. ☞

This behavior leads to unnecessary power consumption and limits the system's ability to handle concurrent tasks. Moreover, polling introduces blocking conditions: if a peripheral fails to provide valid data due to signal loss, transmission error, or sensor malfunction the CPU may remain indefinitely stalled in the waiting loop, preventing the execution of subsequent processes. These limitations make pure polling unsuitable for systems that require strict real-time performance or multitasking responsiveness.

To overcome these drawbacks, polling can be enhanced by integrating interrupt mechanisms. In this combined approach, the system assigns a flag to indicate the availability of new data. When the data transfer or conversion is completed, the peripheral hardware triggers an interrupt to set this flag. The CPU, instead of continuously polling the device, periodically checks the flag to determine whether valid data are ready for processing. This hybrid method allows the processor to perform other tasks in the meantime, improving resource utilization while still maintaining a relatively simple control structure.

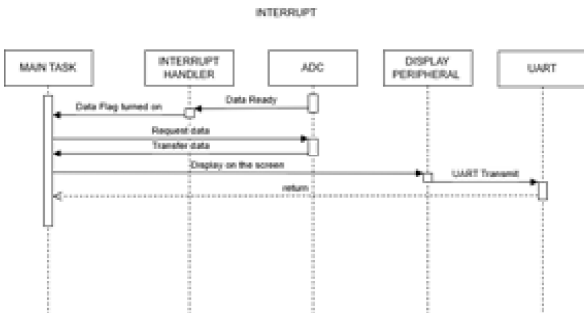

Figure 3. Sequence diagram for reading force data with polling and interrupt method

Compared to pure polling, the polling–interrupt combination offers improved responsiveness and reduced blocking time, as the interrupt immediately signals the CPU when new data are available. The approach also minimizes data loss or corruption because data are fetched only when the corresponding flag is set, ensuring synchronization between acquisition and processing. However, since the CPU must still check the flag regularly, the overall system load remains relatively high, and CPU utilization can still approach full capacity under certain conditions. Furthermore, designing such systems requires careful balancing between interrupt frequency and polling intervals to prevent unnecessary context switching or timing jitter.

In summary, while polling provides simplicity and predictability, its inefficiency limits its applicability in high-performance embedded systems. The hybrid polling–interrupt approach represents an intermediate solution that improves system responsiveness and reliability, though at the cost of increased software complexity and partial CPU occupation. These characteristics make it a useful baseline for comparison against more advanced scheduling strategies such as RTOS and DMA-based architectures in real-time measurement applications.

3.2. RTOS

A RTOS is designed to manage hardware resources, execute multiple concurrent tasks, and meet strict timing constraints required in real-time embedded systems [12]. Although a MCU (microcontroller unit) can only execute one instruction stream at any given time, an RTOS performs rapid context switching among tasks, creating the illusion of true parallelism [13]. Each task in an RTOS is typically assigned a specific function, which improves modularity and enhances code clarity. This modular structure also simplifies debugging

and maintenance, since individual tasks can be independently tested, modified, or reused in other applications [14]. Moreover, the RTOS scheduler allows developers to assign priorities to different tasks, ensuring that time-critical processes are executed first. Such flexibility facilitates scalable and maintainable system design, especially in applications requiring deterministic timing, such as multi-axis machining force measurements [15].

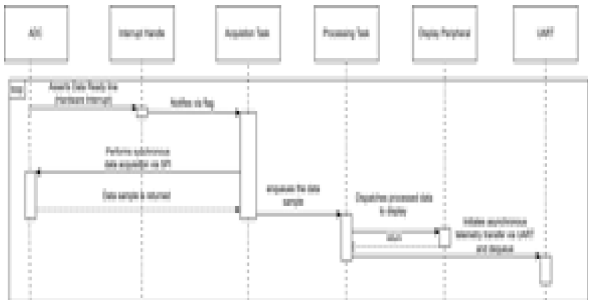

Figure 4. Sequence diagram for reading force data with RTOS method

However, this approach introduces additional system overhead. Each task requires its own stack and memory allocation, which can become a constraint in MCUs with limited on-chip memory. Furthermore, the frequent context switching between tasks consumes CPU cycles and increases latency [16]. Therefore, the implementation of RTOS in resource-limited embedded systems requires careful optimization of memory usage and task scheduling policies to balance responsiveness and efficiency.

To enhance real-time responsiveness, RTOS can be combined with interrupt-driven mechanisms. In this configuration, when an interrupt occurs, the CPU temporarily suspends its current operations to execute an ISR (Interrupt Service Routine). Instead of performing heavy computation within the ISR, it only signals the RTOS to activate the appropriate task for processing [17]. This hybrid approach significantly improves reaction time because the interrupt notifies the system of time-sensitive events immediately, while the RTOS ensures structured and prioritized task execution. The main advantage lies in minimizing blocking time and avoiding long ISR routines that could otherwise delay other critical processes [18]. Nevertheless, this design increases programming complexity, as developers must carefully partition workloads between ISRs and tasks to maintain determinism and prevent race conditions.

A further improvement in performance can be achieved by integrating DMA (Direct Memory Access) with the RTOS. DMA controllers enable peripherals to transfer data directly to and from memory without CPU intervention, effectively offloading repetitive data movement tasks. In an RTOS-DMA combined system, the DMA transfers a complete block of data such as digitized signals from an ADC and then triggers an interrupt to inform the RTOS that the data is ready for processing. This mechanism allows data acquisition and computation to occur in parallel, significantly reducing CPU workload and improving real-time throughput [19]. By freeing the CPU from routine data transfers, system energy efficiency is also enhanced. However, such designs require a deep understanding of the internal operation of DMA controllers, interrupt priorities, and RTOS task scheduling. Improper synchronization between DMA completion and task execution may lead to data corruption or missed samples, especially in high-frequency signal acquisition systems. Consequently, although combining RTOS with DMA provides high efficiency and responsiveness, it demands careful hardware-software co-design and thorough validation to ensure reliability in real-time embedded measurement applications [20]. ☞

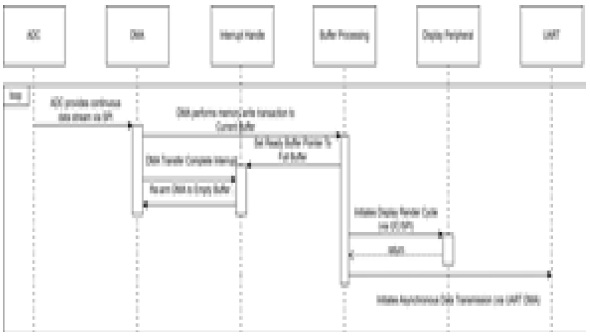

Figure 5. Sequence diagram for reading force data with RTOS and DMA method

To address synchronization challenges and prevent data corruption or loss during continuous data acquisition, a double buffering mechanism is employed. Instead of using a single memory array for incoming data, two buffers: Buffer A and Buffer B operate alternately in a ping-pong fashion. While new data are being transferred into Buffer A via DMA, the system simultaneously processes the previously acquired data stored in Buffer B. Once the DMA transfer to Buffer A is complete, the roles of the two buffers are swapped, allowing the DMA to begin writing to Buffer B while Buffer A is being processed. This alternating mechanism ensures that DMA never overwrites data currently being accessed by the RTOS task, and vice versa. It effectively eliminates memory access contention between the data acquisition and processing threads, enabling uninterrupted, high throughput operation. By maintaining a continuous flow of acquisition and computation, double buffering enhances system stability and guarantees real-time responsiveness even under heavy data rates.

4. EXPERIMENT AND RESULTS

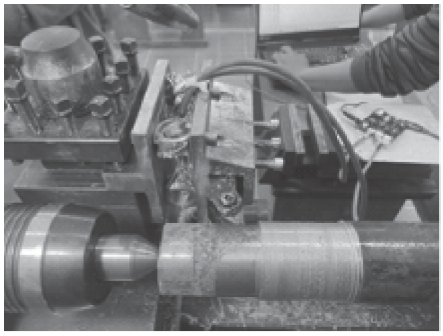

Figure 6. Experimental set-up

The experimental evaluation was conducted on a conventional lathe integrated with a three-component force dynamometer, which was mounted directly onto the machine tool to monitor cutting forces during turning operations. The sensor output was conditioned and sampled by an ADC embedded within the MCU, and the acquired data were subsequently transmitted to a host computer for post processing and visualization. Thirty experimental trials were performed to quantitatively compare the performance of different data acquisition architectures Polling, Interrupt, RTOS combined with Interrupt, and RTOS combined with DMA under identical cutting and sampling conditions.

The first experiment measured the percentage of CPU idle time using an oscilloscope to capture system activity. In the pure polling and interrupt-driven configurations, the CPU exhibited almost continuous activity, corresponding to an idle time of approximately 0%, indicating full CPU occupation during the acquisition cycle. In contrast, the RTOS-based architectures demonstrated significant improvements. Statistical analysis using a Student's t-distribution across 30 samples showed that the RTOS combined with Interrupt achieved an average CPU load of 97.51% ±

0.45% at a 98% confidence level. The RTOS combined with DMA configuration yielded the best result, with an average CPU load percentage of 87.23% ± 2.15%, indicating minimal processor engagement during acquisition and substantial efficiency gains.

A second experiment evaluated the sampling rate stability by measuring the number of valid samples acquired per second from the ADC. The results were analyzed under a normal distribution with a 98% confidence interval. The RTOS and Interrupt configuration achieved a mean sampling rate of 33,163.37 ± 33.74 samples/s, while the RTOS + DMA achieved 33,145.09 ± 36.64 samples/s. In comparison, the pure Interrupt configuration recorded 33,160.35 ± 42.37 samples/s, and the Polling configuration yielded 33,102.05 ± 85.19 samples/s, showing higher variability and reduced consistency.

The experimental findings indicate that while traditional Polling and Interrupt-based methods can achieve comparable sampling rates to RTOS-based implementations, they do so at the cost of full CPU utilization. Such architectures are unsuitable for modern embedded dynamometer systems, where the processor must concurrently handle multiple tasks such as data logging, filtering, and communication. The integration of an RTOS substantially improved system efficiency by providing deterministic scheduling and task prioritization. In particular, the RTOS + Interrupt configuration achieved a notable reduction in CPU load, which allows concurrent task execution without compromising data acquisition. However, the RTOS + DMA configuration demonstrated the most remarkable performance, with high stability, validating its superior suitability for real-time measurement environments.

5. CONCLUSION

Based on the experimental analysis, the proposed architecture combining RTOS and DMA proved to be the most efficient and reliable solution for high-performance embedded force measurement systems. This configuration achieved an optimal balance between throughput and computational efficiency maintaining nearly maximum sampling rates while minimizing CPU load. By decoupling data transfer from CPU control, the RTOS + DMA architecture enables parallel execution of data acquisition and processing tasks, significantly improving system scalability and power efficiency. These results demonstrate that employing RTOS with DMA is a practical and robust approach for developing next generation smart machining monitoring systems capable of handling multi-task workloads in real-time environments.❖

References:

[1]. M. Rizal, J. A. Ghani, M. Z. Nuawi, Husni, and Husaini, "*Design and construction of a strain-gauge-based dynamometer for a 3-axis cutting force measurement in turning process*". Journal of Mechanical Engineering and Sciences, vol. 12, no. 4, pp. 4072-4087, 2018.

[2]. B. S. Yaldız and F. Unsal, "*A dynamometer design for measurement of the cutting forces on turning*". Measurement, vol. 39, no. 2, pp. 80-89, 2006.

[3]. M. F. Gómez and T. Schmitz, "*Low-cost, constrained-motion dynamometer for milling force measurement*". Manufacturing Letters, vol. 29, pp. 65-69, 2020.

[4]. M.F.Gómez and T. Schmitz, "*Displacement-based dynamometer for milling force measurement*". Procedia Manufacturing, vol 34, pp. 867-875, 2019.

[5]. S. Alonso, J. M. López, J. L. Díaz, and ☞

D. F. García, "*Interrupt latency accurate measurement in multiprocessing embedded systems by means of a dedicated circuit*". Electronics, vol. 13, no. 9, p. 1626, 2024.

[6]. C. C. Dobrescu, I. González, D. Carneros-Prado, and C. Nugent, "*Direct memory access-based data storage for long-term acquisition using wearables in an energy-efficient manner*". Sensors, vol. 24, no. 15, p. 4982, Aug. 2024.

[7]. G. C. Buttazzo, "*Hard Real-Time Computing Systems*", 3rd ed.. Springer, 2011.

[8]. I. Ungurean, "*Timing comparison of the real-time operating systems for small microcontrollers*". Symmetry, vol. 12, no. 4, p. 592, Apr. 2020.

[9]. M. A. Ahmed, A. Aljumah, and M. G. Ahmad, "*Design and implementation of a direct memory access controller for embedded applications*". International Journal of Technology, vol. 10, no. 2, pp. 309-319, 2019.

[10]. Y. R. Shen, B.-Y. Huang, and D. Cascaval, "*Efficient sharing of linked DMA channels on multi-sensor devices by LDMA task scheduler*". In Proc. 15th ACM Int. Systems and Storage Conf. (SYSTOR), Jun. 2022, pp. 87-97.

[11]. K. Langendoen, J. Romein, R. Bhoedjang, and H. Bal, "*Integrating polling, interrupts, and thread management*". In Proc. 6th Symposium on the Frontiers of Massively Parallel Computation (Frontiers '96), 1996, pp. 13-22.

[12]. G. C. Buttazzo, "*Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications*". Springer, 2011.

[13]. N. I. Rafla and D. Gauba, "*Hardware implementation of context switching for hard real-time operating systems*". IEEE.

[14]. F. Hessel, V. d. Rosa, I. Reis, R. Planner, C. Marcon and A. Susin, "*Abstract RTOS Modeling for Embedded Systems*". IEEE, 2004.

[15]. T. Wang, Q. Liu and L. Wang, "*An RTOS-based embedded CNC system*". IEEE, 2010.

[16]. M.-Y. C. Ya-Shu Chen, "*On-line energy-efficient real-time task scheduling for a heterogeneous dual-core system-on-a-chip*". Elsevier, vol. 59, no. 4-5, pp. 234-244, 2013.

[17]. P. Barry and P. Crowley, "*Modern embedded computing: designing connected, pervasive, media-rich systems*". ACM SIGSOFT Software Engineering Notes, 2013.

[18]. Morteza Saberikamarposhti, Amir Masoud Rahmani, Sima Abolhassani Khajeh, "*Real-Time Scheduling in IoT Applications: A Systematic Review*". Sensor, p. 232, 2023.

[19]. M. K. R. a. P. Haris Turkmanović *ORCID, "*High Performance Software Architectures for Remote High-Speed Data Acquisition*". Electronics, 2023.

[20]. M. Kistler and D. Brokenshire, "*Detecting race conditions in asynchronous DMA operations with full system simulation*". IEEE, 2011.