Real Time Operating Systems for Embedded Applications: Design and Implementation

Aisyah Binti Ahmad[1], Setiawan Prabowo[2]*
[1,2]Universitas Bhayangkara Jakarta Raya, Jalan Raya Perjuangan, Jawa Barat, Indonesia

Keywords:
Embedded Systems;
Real-Time Scheduling;
RTOS Design;
Task Management;
Embedded Applications

Corresponding Author Email:
prab.seti.wan@dsn.ubharajaya.ac.id

DOI: 10.31838/JIVCT/02.01.05

Received : 15.11.2024
Revised : 18.12.2024
Accepted : 15.01.2025

Abstract

All industries rely on real time operating systems (RTOS) embedded applications because they deliver deterministic behaviour and accurate timing in the most critical systems. This thorough guide targets those who are developing time sensitive embedded projects and are working as developers or engineers. And yet, the complexity and prevalence of embedded systems are growing ever higher, and the demand for robust, reliable, and efficient operating systems has never been greater. All RTOS are mission critical in autonomous vehicles, medical devices and industrial automation and all RTOS require reliable performance under strict timing constraints. Here, in this article, we explain the key concepts of RTOS, major components with design pattern and best practices to optimise and test. If you are new to real-time systems or want to increase your current skill level, this guide is for you in developing high performance embedded applications.

How to cite this article: Ahmad AB, Setiawan Prabowo S (2025). Real Time Operating Systems for Embedded Applications: Design and Implementation. Journal of Integrated VLSI, Embedded and Computing Technologies, Vol. 2, No. 1, 2025, 37-45

The Real-Time Operating Systems Understanding.

Real time operating systems refer to the specialized software platforms that are meant to be run on a hardware which posses an ability to manage the resources of the said hardware along with a task with a given time frame. General purpose operating systems have high throughput but lose determinacy or predictable response times — RTOS are different. Determinism: To guarantee deterministic and deterministic execution times for sensing/processing of critical tasks. Often come down to minimizing event occurrence to system response delays (low latency). Resource allocation based on task's importance and urgency: priority based scheduling. On the other hand, preemptibility: high priority tasks can interrupt the running of low priority tasks. On the minimal kernel the overhead is reduced and the work performed is more efficient.[1-3]

Types of Real-Time Systems

Typically real time systems are classified according to the havoc created if the timing deadlines are not met.

1. Results will potentially be catastrophic, deadlines need to be miss; critical (e.g., aircraft control systems, medical devices).
2. Missed deadlines are allowed in some cases (e.g. video conferencing, industrial process control) but the system utility vanishes fast if they occur (e.g. reduced quality of the video).
3. They are soft real-time systems: they do not cause critical failures (e.g. periodic tasks in a multimedia streaming system, or task completion, for example, in a home automation system) but they do cause system quality degradation (i.e., their deadlines are almost always missed).

Knowing these distinction when picking an RTOS and designing a production embedded application for use cases are a must.[4-5]

RTOS Architecture and Components

One such example is that of a real time operating system, where a lot of parts come together to make the system deterministic and efficient in resource management at the same time. This article will explore which of these elements are combined when designing an RTOS.

Kernel

Any RTOS's kernel is the heart of the system: it performs the core system and resource management functions. The kernel handles such things as task scheduling, interrupt processing and task communication in between tasks. In RTOS design, there are two primary kernel architectures that are commonly used:

1. Monolithic kernel utilizes more memory but the services in the system run in kernel space, and hence the executability is faster.
2. Low level minimal core services are run in kernel space and the additional functionality is run with low level services as user space processes for improving fault isolation and modularity.

Task Scheduler

One of such important components is task scheduler deciding about when tasks run and how. RTOS schedulers usually implement its algorithms based on the priorities of tasks so that high priority tasks are responded immediately. Common scheduling algorithms include: Priority is assigned according to frequency, which is called Rate Monotonic Scheduling (RMS). Latest Deadline First (EDF): Items are ordered by their priority, depending on their next scheduling deadline.

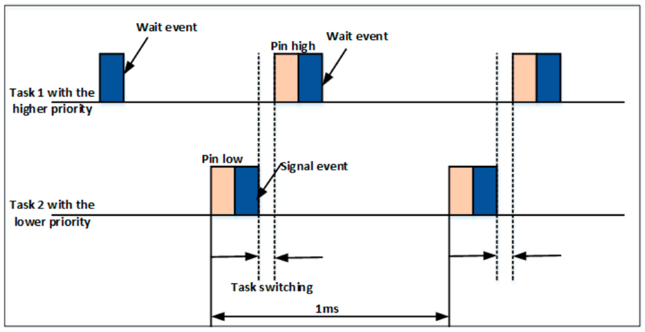

Fig.1: Task Scheduler

Interrupt Handler

The management of interrupts in real time systems must be very efficient in the pursuit of keeping the latency low and quick response to external events in order to use other resources and keep some of the critical processing on holdMinimize interrupt service routine (ISR) execution time for Embedded Applications. Real Time operating systems (RTOS) are at the core of modern embedded applications controlling precise timing and deterministic behaviour in such critical system domains as semiconductor manufacturing, communications, home and industrial automation and aerospace. This all encompassing guide delves into the details associated with RTOS design and implementation, complete with information that is of much benefit to developers and engineers working on time sensitive embedded projects.

In conjunction with the tremendous increase in the complexity and ubiquity of embedded systems, there has never been a stronger need for robust and efficient, yet predictable, operating systems. RTOS enables reliable performance under stringent timing points, which range from autonomous vehicles to medical devices to industrial automation. This article will cover the basics of RTOS, including core principles, main parts, design patterns as well as optimizations and testing best practices. This guide is for beginners in the field of real-time systems or for those who wish to further develop their understanding of this topic; it provides keen understanding in implementing high performance embedded applications.[6-8]

Understanding Real-Time Operating Systems

Real time operating system (RTOS) is a specialized software platform that takes care of hardware resource management and task execution under a guaranteed time frame. While general purpose operating systems are designed to maximize overall throughput, RTOS aim for systems that have deterministic behavior and it is possible to predict their response time.

Key Characteristics of RTOS

Deterministic Execution: for determinism i.e. predictable execution time for critical tasks. Low Latency – Reducing the delay between the occurrence of event and the system response. Scheduling resources based on the priority of task importance and urgency. Preemptibility: Interrupting lower priority tasks by higher priority tasks. Streamlined core functionality with minimal kernel: to cut down on overhead and run more efficiently

Types of Real-Time Systems

Mostly, real-time systems are categorized depending severity of the consequences of missing timing deadlines (deadlines):

1. Absolute adherence to deadlines is critical and missing deadlines can cause potentially

catastrophic behavior (e.g., aircraft control systems, medical devices).

2. Occasional deadlines are missed, but the utility of the system rapidly degrades (e.g., video conferencing, industrial process control).
3. Missed deadlines will impact the system quality, but no critical failures will happen to the system (such as multimedia streaming, home automation, etc.)

It is important to understand these basic differences so that it can be determined if certain RTOS are appropriate choices and so that they can be utilized to program robust embedded applications according to various use scenarios and requirements.

RTOS Architecture and Components

A real time operating system is said to consist of number of components, which are connected, and that operates together to guarantee determinism and good resource management. Now let us explore the key elements that constitute the basis of the RTOS design. The core of any RTOS is the kernel that is responsible for basic system functions and resource management. Task scheduling, processes interrupt handling and provides intertask communication are some of the things the kernel does. There are commonly two main kernel architectures employed in RTOS design:

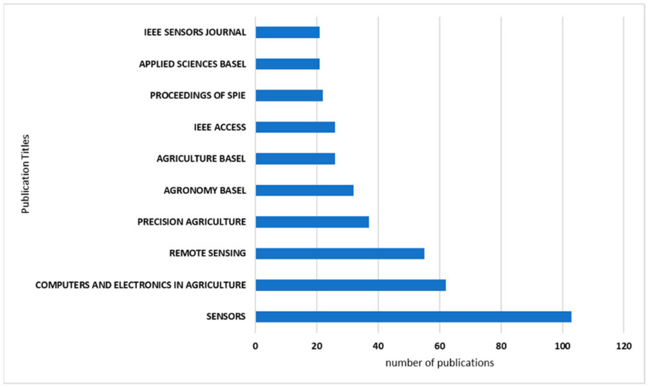

Fig. 2: RTOS Architecture and Components

1. All system services in kernel space for faster execution, but also with larger memory footprint
2. Minimal core services run in kernel space and more functionality is implemented as user space processes providing better modularity and fault isolation than a monolithic kernel does.

That is how important the task scheduler is; as it decides what tasks to run and when they should be run. Priority based algorithms are usually used by real time schedulers to assure that a high priority task be serviced immediately. Common scheduling algorithms include: First scheduling is Rate Monotonic Scheduling (RMS) which schedules the tasks according to the estimation of their priority. Dynamically prioritised according to upcoming deadlines (Earliest Deadline First, EDF). Scheduling by Time, one of the types of embedded systems, executes the tasks on a fixed time basis.

Interrupt Handler

Interrupt handling is of utmost importance for low latency in real-time systems, and so should be as efficient as possible. The design purpose of RTOS interrupt handlers is: Reduce the time spent in interrupt service routine (ISR) execution. Shift non critical processing to lower priority work to complete them. The memory management in RTOS is predictable allocation and deallocation of resources. Key features include: Compile time pre allocation of memory to avoid run time fragmentation. Prevents the task memory space from being corrupted. Deterministic allocation algorithms are required algorithms for which memory operations behavior must be consistent (Table 1).

RTOSs offer a number of mechanisms for tasks to communicate and synchronize via: Semaphores used as a method of resource management and signaling among tasks. To asynchronously exchange data between the task via the message queues. Event flags – task synchronization conditioned on satisfaction of certain constraints. Real time operating system for the embedded systems is to be designed and implemented using these core components.[9-11]

Real-Time Systems and Determinism

Determinism pins real time systems with critical tasks that must execute within their respective time windows regardless of system load or otherwise. In this process, both software and hardware aspects involved to design the system to achieve deterministic behavior.

Hardware requirements for determinism

In order to maintain deterministic performance, it is necessary to select appropriate hardware components.

1. **Processors:** Pick CPUs with predictable instruction execution times and little pipeline stalls.
2. **Memory:** Equipped with high speed low latency memory with consistent access time.

Table 1: Real-Time Operating Systems

Challenge	Description	Impact
Task Scheduling and Prioritization	Ensuring that tasks are scheduled according to priority, while meeting real-time constraints.	Incorrect task scheduling can lead to delays or system crashes, affecting the reliability of embedded systems.
Real-Time Constraints and Deadlines	Guaranteeing that tasks meet deadlines without overloading system resources.	Failure to meet deadlines can compromise system functionality and lead to performance degradation.
Memory and Resource Management	Efficiently managing memory and system resources in resource-constrained embedded devices.	Inefficient memory usage can lead to system crashes or reduced performance in embedded systems.
Power Consumption	Reducing power consumption while maintaining the performance of real-time applications.	High power consumption can reduce battery life in portable devices, limiting system uptime.
Scalability and Flexibility	Designing systems that can scale according to different application requirements and hardware platforms.	Scalability issues may limit the deployment of the system in different embedded environments or applications.

3. Always go for peripherals that are deterministic in response time and have efficient interrupt handling.
4. **Timer:** Very precise task scheduling and timing measurements can be accomplished with high resolution timers.

Determinism Measurement and Analysis

Developers can use different tools and techniques to verify and improve system determinism.

1. Worst case execution time (WCET) analysis; determine the amount of time at worst before finishing a task in the worst case for these circumstances.
2. **Jitter analysis:** Find the possible sources of non determinism by measuring the variation of the execution time of a task.
3. **Tracing and profiling:** Collecting timing information using special tools to trace out and profile performance bottlenecks.
4. **Stress testing:** We needed to be able to stress test the system with high loads and change conditions, and we needed the system to exhibit deterministic behavior under all cases.

To achieve an effective real time operating system you need to adhere to proven design patterns and best practices. These guidelines assist developers develop robust, maintainable and efficient RTOS based applications. Centralized resource control. menting an effective real-time operating system requires adherence to proven design patterns and best practices. These guidelines help developers create robust, maintainable, and efficient RTOS-based applications. Benefits: Simpler access and improved resource utilization. Can be used to implement interrupt handler and task notifications, it provides support of the event driven architecture. Controls the complex system behaviors and transitions, promotes loose coupling between components. A good match for protocol stacks and device drivers Improves code readability as well as its maintainability. Guarantees that there is only one instance of a critical system component. For the real time operating system, scheduler, memory manager and device driver.e are commonly used; so design patterns and best practices to be adhered to are proven. These guidelines aid developers in building rock solid and maintainable as well as performant RTOS based applications.[12-15]

Common RTOS Design Patterns

Teams that have mastered the Above Listed Best Practice have developed Small finsable tasks, completed them while focusing on the functions of a thing, to correctly handle and recover from errors, and to be consistent with naming of tasks and related resources. Depending on how big the data is and based on what timing requirement, choose suitable IPC mechanisms. All blocking operation can handle timeout and can inherit from Prioritization to prevent priority inversion.

‘Priority inheritance is used to prevent priority inversion problems, and when possible, the overhead free priority enforcement strategy based on priority

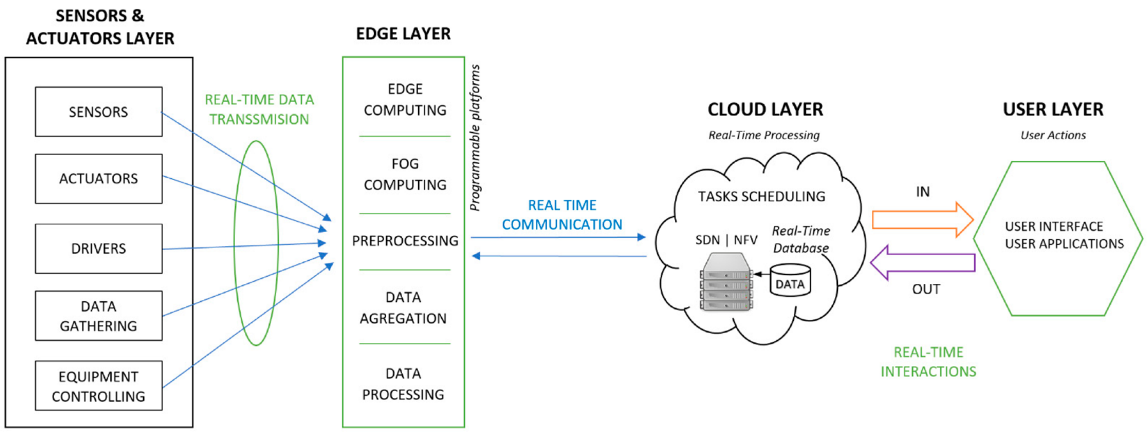

Fig. 3: Common RTOS Design Patterns

preemptive queues is used; otherwise, resource locking has to be carefully implemented to prevent deadlock.' Frequently allocated/deallocated objects could be allocated out of resource pools, Memory alignment could be chosen to facilitate optimal performance, Execution times for ISRs could be reduced by deferring non critical processing to tasks. Control system responsiveness by means of interrupt priorities Synchronize proper ISRs (Figure 3).[16-18]

Can detect task overruns, system hangs or trying to conserve as much time spent on each computation as possible without deviating from time units used at any other point in the system. Last year, the national median wage topped £35,000 per year, but 56% of Britain's top G&T schools spent less than £2,000 more on their staff. When the priority tasks are running slowly, they become the goal of the time slicing. Comprehensive logging and tracing mechanisms are provided, and RTOS aware debugging tools are used for better debugging; thus, real-time operating system must follow proven design patterns and best practices. For creating robust, maintainable and efficient applications using RTOS, these guidelines acts as useful.[19-21]

Common RTOS Design Patterns

Debugging and Profiling

This paper demonstrates how these design patterns and practice can be applied to augment existing real time operating systems for embedded applications to be more reliable, more efficient and more maintainable. These guidelines are nice to begin with, so that RTOS implementation will be successful as many domains and use cases as possible. For embedded application, these operating systems must have the best timing for embedded application. This part looks at different RTOS products and efficiency response enhancement techniques. It reduces overhead of calling functions that are frequently used & small. Reduces the amount of call stack used for function calls. The messages should not be too elaborate (or not too nuanced) for the sake of code bloat. It also maintains the code maintainability and ensures proper documentation. Wen known design patterns and best practices towards writing an effective real time operating system. These guidelines assist the developers in creating robust, maintainable, and efficient RTOS based application .[22-24]

Common RTOS Design Patterns

Sensitive data can be communicated using strong encryption and when data integrity matters, use message authentication codes (MAC). Use secure protocol, industry standards (i.e TLS, DTLS), handle management and validation of proper certificate. If hardware based key storage is available, use it, otherwise implement the secure key generation and storage, and key rotation mechanisms. Access Control mprovides strong authentication means of the user accessing the system iv. Strong encryption for sensitive data transmission. Establish proper access controls on system resources and enforce those access controls. A principle of least privilege for task permissions. Implement tamper evident logging mechanisms securing logs of critical system events and system access attempts validate and sanitize all data including data coming from trusted sources avoid using dangerous C functions always use safe string handling functions implement proper buffer size management and bounds checking (Table 2):

If the space allocated is smaller than the required memory then the array is truncated and rest of the

Table 2: Implementing Real-Time Operating Systems

Solution	Description	Benefit
Preemptive and Cooperative Scheduling	Using both preemptive and cooperative scheduling methods to handle real-time tasks efficiently.	Improves task management and ensures system responsiveness in real-time applications.
Priority-based Scheduling Algorithms	Implementing priority-based scheduling algorithms (e.g., Rate Monotonic Scheduling) to ensure deadlines are met.	Helps prioritize critical tasks and ensures timely execution, improving system reliability.
Memory Partitioning and Management	Utilizing memory partitioning to allocate resources efficiently and avoid resource conflicts in embedded systems.	Maximizes available memory and ensures efficient resource utilization in embedded systems.
Low-Power Design Techniques	Adopting low-power techniques such as dynamic voltage scaling and sleep modes to reduce power consumption.	Extends battery life while maintaining system performance, crucial for portable devices.
Modular and Configurable RTOS	Designing a modular RTOS architecture that can be configured for different types of embedded applications.	Enables easy adaptation of the RTOS to various embedded applications, improving system versatility.

array contents is neglected. We check if the written bytes are less than required memory and otherwise ignore the rest of the bytes in case they are more. Some of the following are nested Buffers : If the first layer buffer has enough space(the buffer size is allocated properly), then there more bytes can be written in the first layer until the complete memory required is reached.

It implements secure practices for data allocation and deallocation (Makes sure that sensitive data from memory is zeroed out when it's not used anymore) It also strongly encrypts sensitive data before transmission. Security Testing Validation Penetration test RTOS as well as applications regularly. To identify the vulnerabilities we simulate various attack scenarios. Allows you to fuzz input handling as well as communicating interfaces and potential crashes of your rockets under malformed input. Code Reviews Write regular security relevant code reviews. rong encryption for sensitive data transmission. By providing continuous Security Monitoring, you help guarantee that devices will not be offering malicious software updates, services and even rogue scanning devices to your systems and to our service endpoint. Run time security monitoring and anomaly detection. Provide plan to respond to an incident in case of security breach and strong encryption of sensitive data transmission. Security considerations for such RTOS based embedded systems can be addressed and the system make more resilient to such attacks. Security, therefore, is not a static instance, but on ongoing process of assessing, revising and changing around, regular review, as new vulnerabilities and threats arise. In the current world where embedded applications are being connected by network, such a wide range of security approach is indispensable both by hardware and software [25]-[26].

Validation of the RTOS based Systems, Testing

Reliable and correct real time operating systems are key to the success of embedded applications. Therefore deployment of a system can only occur after elements of the system have undergone extensive testing and validation where the elements are checked and fixed prior to deployment. In this section, I looked into a bunch of strategies and methods for testing a RTOS based systems. Tests for individual RTOS modules (encoded by Isolation of components for testing with mocking frameworks: scheduler, memory manager) Test all edge cases and extreme conditions for each component and and. We should check a proper error handling, error recovery mechanism and also try to achieve a high code coverage in unit tests. Among the verified is communicating between the RTOS components, verification of the proper data flow and communication between modules. Apply the filter algorithms on target hardware.[27-28]

Future Trends in RTOS development

Real time operating systems are being used today for many emerging technologies and application domains, and the growing landscape of real time operating systems has been developed to respond to these needs. This section investigates some of the major trends for development of RTOS and its importance to the embedded systems.

Integration of artificial intelligence, machine learning adaptive scheduling integration of machine learning algorithms; Use of AI models for predictive maintenance, fault detection etc; AI inference at the edge; Optimized RTOS designs; GPU Controllers and Neural Processing Units.

Clouds can be used to efficiently manage the provide control of AI accelerators and neural processing units. Enabling integration of AI driven decision making into the critical control loops (safety) is increasingly enabled in continuously developing technologies and critical application domains to meet the increasing demands of emerging technologies and applications. In this section, we discuss some of the major trends which are likely to influence the future of RTOS development and the land they might leave in the territory of embedded devices.

Support for running multiple OSes with different criticality levels under Virtualization Resource partitioning for isolation of safety critical and non safe critical tasks Coexistence of legacy Real Time Operating Systems and modern operating systems Migration of existing system and its upgrades to be enjoyed step by step. Sensitive operations executed in secure environments is increasingly being followed by time operating systems to meet the demands of emerging technologies and application domains.- time operating systems in order to satisfy the needs to emerging technologies and application domains, is continuously evolving to further reach improvements in the isolation and protection of virtualization. This article examines a few of the most important trends in the development of RTOS and explores how they may change the way embedded systems are constructed.

This implies saving in the low power consumption hence this saving can be applied to battery operated IoT devices. Power management and sleep mode optimization Native support of 5G and other emerging wireless protocols. For generation of 5G mobile devices, the complete cloud wireless backbone IoT communication optimized network stacks, and for IoT devices, secure and reliable mechanisms for the firmware update are supported. Atomic updates (as well as rollback in mission critical systems is continually evolving to keep up with the new demands of new technologies and application domains. Some of the trends in RTOS development that are important in shaping the future of RTOS and its impact on embedded systems are discussed in this section.

Quantum Computing Readiness

Quantum Resistant Cryptography Post quantum cryptographic algorithms integration. Quantum inspired optimization techniques for scheduling, it is preparing the security features of the RTOS for the quantum realm. Complex decision making use potential of quantum annealing Deep integration with hardware security modules (HSM), and trusted execution environment; The use of emerging CPU security features to improve protection. Anomaly detection and threat prevention through using machine learning Real time intelligence based security policies Evaluating blockchain technologies for IoT data management with security and audibility. De-centralized system distributed trust models. Efficient time series data handling and visualization support.

Conclusion

The main concept of sketch algorithm as a tool for approximating aggregate statistics quickly. We will first state an overview of the basic idea of this algorithm. So, we will also learn about a certain powerful kind of sketch algorithm, namely Count Sketch. We will then discuss functions of the technique that are not under its supervision per se, such as counting multisets. Once we have understood the Streaming analytics in real time. Although prognostics and health management (PHM) capabilities of integration For early fault detection and system optimization, real time analysis.g systems are continually growing to keep pace with the need of emerging technologies and application domains. This section discusses some of the main trends in the area of the RTOS development of, and the ways it may affect the embedded systems.

References:

1. Fenick, S., & Joiner, H. F. (1992, February). So Little Time To Measure It. In *Proeedingsof Tnanual National Conference On Ada Technology* (p. 220).
2. Holt, R. C. (1972). Some deadlock properties of computer systems. *ACM Computing Surveys (CSUR)*, *4*(3), 179-196.
3. Howard Jr, J. H. (1973). Mixed solutions for the deadlock problem. *Communications of the ACM*, *16*(7), 427-430.

4. Ready, J. (1986). VRTX: A real-time operating system for embedded microprocessor applications. *IEEE Micro*, *6*(04), 8-17.
5. Stankovic, J. A., & Ramamritham, K. (1987, December). The design of the Spring kernel. In *RTSS* (Vol. 87, pp. 146-157).
6. Arnold, & Berg. (2006). A modular approach to real-time supersystems. *IEEE Transactions on Computers*, *100*(5), 385-398.
7. Evans, R. J., & Franzon, P. D. (1995). Energy consumption modeling and optimization for SRAM's. *IEEE Journal of Solid-State Circuits*, *30*(5), 571-579.
8. Kirovski, D., Lee, C., Potkonjak, M., & Mangione-Smith, W. H. Synthesis of Power-Efficient Memory-Intensive Systems-on-Chip.
9. Furumochi, K., Shimizu, H., Fujita, M., Akita, T., Izawa, T., Katsube, M., ... & Kawamura, S. (1996, February). A 500 MHz 288 kb CMOS SRAM macro for on-chip cache. In *1996 IEEE International Solid-State Circuits Conference. Digest of TEchnical Papers, ISSCC* (pp. 156-157). IEEE.
10. Madhumitha, K., Ganesh, E. N., Vallabhuni, R. R., Vinu, M. S., Dandekar, T., Jain, S., Anitha, S., Karunya, S., & Kannan, V. (2023). *Wireless router* (Design No. 379514-001). The Patent Office Journal, 19/2023, India.
11. Smith, M. J. S. (1997). *Application-specific integrated circuits* (Vol. 7, pp. 1-1). Boston: Addison-Wesley.
12. Stewart, D. B., Schmitz, D. E., & Khosla, P. K. (2002). The Chimera II real-time operating system for advanced sensor-based control applications. *IEEE Transactions on Systems, Man, and Cybernetics*, *22*(6), 1282-1295.
13. Stewart, D. B., Volpe, R. A., & Khosla, P. K. (1997). Design of dynamically reconfigurable real-time software using port-based objects. *IEEE Transactions on software engineering*, *23*(12), 759-776.
14. Hergenhan, A., & Heiser, G. (2008, November). Operating systems technology for converged ECUs. In *6th Emb. Security in Cars Conf.(escar). Hamburg, Germany: ISITS*.
15. Kinebuchi, Y., Koshimae, H., Oikawa, S., & Nakajima, T. (2006). Virtualization techniques for embedded systems. In *Proceedings of the Work-in-Progress Session: The 12th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications* (RTCSA), Sydney, Australia.
16. Kinebuchi, Y., Sugaya, M., Oikawa, S., & Nakajima, T. (2008). Task grain scheduling for hypervisor-based embedded system. In *Proceedings of the 2008 10th IEEE International Conference on High Performance Computing and Communications (HPCC)* (pp. 190–197). IEEE Computer Society.
17. Sajithabanu, S., Adhoni, Z. A., Niranjan, S. K., Vallabhuni, R. R., Saif, M. A. N., Dhanasekaran, S., Gnanasaravanan, S., & Kannan, V. (2023). *IoT-enabled file storage racks* (Design No. 379414-001). *The Patent Office Journal*, 18/2023, India.
18. Schoeberl, M. (2009). *JOP reference handbook: building embedded systems with a java processor*.
19. Schoeberl, M. (2010, November). Time-predictable chip-multiprocessor design. In *2010 Conference Record of the Forty Fourth Asilomar Conference on Signals, Systems and Computers* (pp. 2116-2120). IEEE.
20. Meghanathan, S. B. N. (2005). A survey of contemporary real-time operating systems. *Informatica*, *29*(2).
21. Laplante, P. A. (2004). *Real-time systems design and analysis* (Vol. 3). New York: Wiley.
22. Walls, C. (1996). RTOS for microcontroller applications. *Electronic Engineering*, *68*(830), 57-59.
23. Schmidt, D. C., Deshpande, M., & O'Ryan, C. (2002, January). Operating System Performance in Support of Real-Time Middleware. In *WORDS* (pp. 199-206).
24. Shukla, B. K., Mahilraj, J., Anandaram, H., Vallabhuni, R. R., Neethidevan, V., Narasimhulu, N., Humnekar, T. D., & Kannan, V. (2023). *IoT-based weather reporting system* (Design No. 378815-001). *The Patent Office Journal*, 15/2023, India.
25. Pothuganti, K., Haile, A., & Pothuganti, S. (2016). A comparative study of real time operating systems for embedded systems. *International Journal of Innovative Research in Computer and Communication Engineering*, *4*(6), 12008.
26. Pothuganti, K., Haile, A., & Pothuganti, S. (2016). A comparative study of real time operating systems for embedded systems. *International Journal of Innovative Research in Computer and Communication Engineering*, *4*(6), 12008.
27. Noble, B. D., Satyanarayanan, M., Narayanan, D., Tilton, J. E., Flinn, J., & Walker, K. R. (1997). Agile application-aware adaptation for mobility. *ACM SIGOPS Operating Systems Review*, *31*(5), 276-287.
28. Noble, B. (2000). System support for mobile, adaptive applications. *IEEE Personal Communications*, *7*(1), 44-49.
29. Rahim, R. (2024). Quantum computing in communication engineering: Potential and practical implementation. *Progress in Electronics and Communication Engineering, 1*(1), 26–31. https://doi.org/10.31838/PECE/01.01.05
30. Rahim, R. (2024). Adaptive algorithms for power management in battery-powered embedded systems. *SCCTS Journal of Embedded Systems Design and Applications, 1*(1), 25-30. https://doi.org/10.31838/ESA/01.01.05
31. Sadulla, S. (2024). Optimization of data aggregation techniques in IoT-based wireless sensor networks. *Journal of Wireless Sensor Networks and IoT, 1*(1), 31-36. https://doi.org/10.31838/WSNIOT/01.01.05
32. Ariunaa, K., Tudevdagva, U., & Hussai, M. (2025). The need for chemical sustainability in advancing sustainable chemistry. *Innovative Reviews in Engineering*

and Science, 2(2), 33-40. https://doi.org/10.31838/INES/02.02.05

33. Abdullah, D. (2024). Strategies for low-power design in reconfigurable computing for IoT devices. *SCCTS Transactions on Reconfigurable Computing, 1*(1), 21-25. https://doi.org/10.31838/RCC/01.01.05
34. Abdullah, D. (2024). Design and implementation of secure VLSI architectures for cryptographic applications. *Journal of Integrated VLSI, Embedded and Computing Technologies, 1*(1), 21-25. https://doi.org/10.31838/JIVCT/01.01.05
35. Prasath, C. A. (2023). The role of mobility models in MANET routing protocols efficiency. *National Journal of RF Engineering and Wireless Communication, 1*(1), 39-48. https://doi.org/10.31838/RFMW/01.01.05
36. El-Saadawi, E., Abohamama, A. S., & Alrahmawy, M. F. (2024). IoT-based optimal energy management in smart homes using harmony search optimization technique. *International Journal of Communication and Computer Technologies, 12*(1), 1-20. https://doi.org/10.31838/IJCCTS/12.01.01
37. Soh, H., & Keljovic, N. (2024). Development of highly reconfigurable antennas for control of operating frequency, polarization, and radiation characteristics for 5G and 6G systems. *National Journal of Antennas and Propagation, 6*(1), 31–39.