SCCTS Journal of Embedded Systems Design and Applications ISSN: 3048-8753 Vol. 2, No. 2, 2025 (pp. 1-8)
RESEARCH ARTICLE ECEJOURNALS.IN

Fault-Tolerant Embedded Systems: Reliable Operation in Harsh Environments Approaches

Ferreira Martins Carvalho[1], Teusner Perscheid[2]*

[1,2]Polytechnic School, Pontifical Catholic University of Rio Grande do Sul, Porto Alegre, Rio Grande do Sul 90619-900, Brazil

Keywords:
Embedded Systems;
IoT (Internet of Things);
Real-time Operating Systems;
Microcontrollers;
Hardware-Software Co-Design

Corresponding Author Email:
percsd@ifsc.edu.br

DOI: 10.31838/ESA/02.02.01

Received : 01.12.24
Revised : 01.03.25
Accepted : 01.05.25

Abstract

Embedded systems are playing an increasingly important role in delivering critical applications across industries ranging from aerospace and defense to telecommunications to healthcare. These systems operate in harsh environments where failure carry severe consequences. Consequently, fault tolerance has become a critical necessity for embedded systems to guarantee uninterrupted and steady operation under both hardware and software faults. In this paper, we discuss different approaches and techniques to improve the reliability and availability of fault tolerant embedded systems working under hard operational conditions. In the following, we will study the basics of fault tolerance, look at different fault tolerant architectures and discuss the approaches of building robust embedded systems that can bear failures and work properly.

How to cite this article: Carvalho FM, Perscheid T (2025). Fault-Tolerant Embedded Systems: Reliable Operation in Harsh Environments Approaches. SCCTS Journal of Embedded Systems Design and Applictions, Vol. 2, No. 2, 2025, 1-8

Fault Tolerance in Embedded Systems

Fault tolerance means that a system will operate correctly in the presence of faults or errors. For embedded systems, without fault tolerance, system reliability, availability, and safety will be higher at risk in mission critical applications.

Before diving deeper into fault-tolerant approaches, it's essential to understand some key concepts:

1. **Fault:** An abnormal or defective system component that can cause a failure.
2. **Error:** The unexpected behavior of a system, usually caused by a fault.
3. **Failure:** Failure of a system to meet its required functions in a required manner, within the accepted levels of time, and cost, verified by tests on specific outputs.
4. **Reliability:** The likelihood that a system will accomplish its desired result for a given time and under stated circumstances.
5. **Availability:** An indication of the amount of time a system is capable of performing its intended functions and is operational.[1-4]

Importance of Fault tolerance in Embedded Systems

In general, embedding electronic systems in environments with limited or no human access is quite common. In such scenarios, fault tolerance becomes critical for several reasons:

Safety: In automotive systems, or medical devices applications, failures may result in severe risk of human life.

Continuous Operation: Many of these embedded systems, for example in telecommunication or in industrial control, have to be uninterrupted.

Cost Reduction: The use of fault tolerance can reduce the maintenance or replacement frequency, greatly reducing overall system cost.

Reliability in Harsh Environments: Aerospace and defense applications often require embedded systems which must endure extreme conditions while still functioning.

Classification and Fault Models

To formulate effective fault tolerant strategies there is need to understand the types of faults

and how these faults can corrupt embedded systems.

Types of Faults

Hardware Faults: Among them are defective physical components (processors, memory, or I/O devices), and malfunctions within the software system itself.

Software Faults: Errors in program code, logic or design which can cause system failures.

Transient Faults: Cause: random one time faults like electromagnetic interference.

Permanent Faults: Faunts that persist until repair or replacement of faulty component.

Intermittent Faults: Faults that are often difficult to reproduce, and often are found happening on a sporadic basis.

Duration based fault classification

Transient Faults: These faults come and go in a short period of time. Occasionally they are caused by external factors, such as power fluctuations or radiation.

Intermittent Faults: However, these faults are in a state of emerging, disappearing and reemerging over time. Diagnosis can be tricky and such faults may be an early harbinger of faults about to become permanent.

Permanent Faults: Until the faulty component is repaired or replaced they remain. They can also be caused by wear and tear, manufacturing defect or severe damage.

These fault types and classifications are important to understand in order to design appropriate fault tolerant mechanisms for embedded systems.

Embedded System Fault Tolerant Architectures

The main objective of fault tolerant architectures is to achieve system reliability and availability presence of faults. There are several architectural approaches to take in utilized embedded systems to attain fault tolerance.

Architectures based on Redundancy

One of the basic ideas in fault tolerant design is redundancy, i.e. when you do not use all of your components, but use extra components instead to maintain functionality in case of failures.

Hardware Redundancy: This means replicating essential hardware so backups are made, in case it fails. Common approaches include:

Triple Modular Redundancy (TMR): Each module does the same thing, all 3 modules are identical, and only the majority of the modules are allowed to fuse a signal into the output.

Dual Modular Redundancy (DMR): It uses 2 identical modules that operate in parallel, and a comparator to compare and detect difference.

Software Redundancy: In this case, we are talking of using several software's to do the same thing. Techniques include:

N-Version Programming: A voting mechanism to select the correct output among multiple independently developed version of a software running concurrently.

Recovery Blocks: If the primary algorithm fails an acceptance test then there are alternative algorithms to do a function.

Time Redundancy: It is the repeated computations for transient errors detecting and correcting.

Information Redundancy: This consists of augmenting information in data to detect and correct errors, or error correcting codes and checksums, for example.

Architectures for Fault-Tolerant Systems

Distributed Systems: By distributing system functionality across multiple nodes, the system can tolerate some of its nodes failing without stopping operation.

Self-Checking Pairs: Each consist of two modules running in parallel and their outputs are compared. If a discrepancy is seen, the system will switch to a backup pair or a safe state.

Watchdog Timers: They monitor system activity and take a reset or recovery action if specified expected events are not seen within a specified time frame.

Fault-Tolerant Networks: The redundant communication paths and protocols to guarantee reliable data transfer in the situation where network faults occur.[5-8]

Fault Detection and Diagnosis Techniques

The implementation of fault tolerant strategies on embedded systems depends on practical fault detection and diagnosis. These techniques facilitate fault identification as quickly and accurately as possible for the administration of appropriate recovery actions (Table 1).

Table 1: Fault-Tolerant Techniques for Embedded Systems

Technique	Purpose
Redundant Components	Redundant components involve duplicating critical system components to ensure continued operation in case of failure of one unit.
Error Detection Codes	Error detection codes identify data transmission errors and trigger corrective actions to prevent data corruption and system failures.
Watchdog Timers	Watchdog timers monitor system behavior and reset the system if it becomes unresponsive, ensuring reliable operation in harsh conditions.
Self-Healing Systems	Self-healing systems automatically detect faults and reconfigure themselves to restore normal operation without requiring manual intervention.
Recovery Algorithms	Recovery algorithms restore system functionality after a failure by rerouting operations or restarting processes to minimize downtime.
ECC (Error Correction Codes)	Error correction codes are used to detect and correct errors in data transmission or memory, ensuring the integrity of information within the system.

Fault Detection Utilizing Hardware

Built-In Self-Test (BIST): Self testing integrated circuits that detect faults while in operation or at startup.

Watchdog Timers: Hardware timers that are used to keep an eye on system activity and reset the system if not refreshed from time to time by the software.

Error-Detecting Codes: Coding schemes, like parity checks or cyclic redundancy checks (CRC), that can be implemented in hardware to detect data errors.

Comparators: Redundant hardware circuits that compare output from redundant modules in order to detect discrepancies.

Fault Detection based on software

Assertion Checking: Putting runtime checks in the code to check that during runtime that some conditions are satisfied.

Control Flow Checking: Looking at the control flow taken by the program during its execution and monitoring something when it is different from what we expected it to be.

Data Flow Analysis: To detect the anomalies, you must analyze how data is used and changed through out the program.

Exception Handling: Putting in place tactics to intercept and pattern unexpected error or exception occurrence while running a program.

Fault Diagnosis Techniques

Signature Analysis: Find faults in the system by comparing the system's behavior against known good signatures.

Fault Trees: Analysing possible failure modes and their causes using logical diagrams.

Expert Systems: Using artificial intelligence techniques to bolster diagnoses of faults by symptoms and historical data.

Model-Based Diagnosis: Mathematical models of the system are used to predict its behaviour and compare it with observed behaviour to identify faults.

Fault Recovery and System Reconfiguration

It's the next critical step once a fault is detected and diagnosed: Recover from the fault and reconfigure the system so that functionality is restored. These fault recovery and system reconfiguration strategies are essential in a fault tolerant embedded system.[9-12]

Fault Recovery Techniques

Checkpoint and Rollback: Saving of system state periodically and roll back to a known good state when an incident event is encountered.

Forward Recovery: Race to correct the fault and try to continue operation without rolling back, or use (error correcting) codes to reproduce unreliable data.

Retry Mechanisms: In case of transient faults repeating a failed operation.

Handling: Application of software routines to recover from specific fault conditions.

System reconfiguration strategies

Dynamic Resource Allocation: Providing availability of critical resources until component failures.

Graceful Degradation: Preventing system functionality reduction to significant operations only when full recovery from an event is not possible.

Hot Swapping: Doing so while the system does not shut down.

Adaptive Fault Tolerance: Dynamically adjusting fault tolerant mechanisms to current system condition and fault history.

Software Fault Tolerance Techniques.

Embedded systems possess an equally important aspect of redundant software. There exist techniques for improving the reliability and robustness of embedded software.

Defensive Programming

Input Validation: Checking in thoroughly and validating all of its inputs to avoid unexpected behavior.

Error Handling: Using error detection and handling mechanisms throughout the code.

Boundary Condition Checking: To make sure everything stays in acceptable ranges and limits.

Resource Management: Manages system resources such as memory and processing time to avoid them hitting the resource failures.

Design Diversity

N-Version Programming: Dividing the critical software components into multiple versions implemented with different algorithms or languages of programming.

Recovery Blocks: Altetnataling facit functil kritiku med kravteste som svar på.

Design Patterns for Fault Tolerance: Using software design patterns which increase system reliability and fault tolerance.

Runtime Monitoring and Self Checking

Assertions: Verification that certain program conditions are met during runtime program execution.

Watchdog Processes: Monitoring the health and activity of critical system components and implementing software processes for that purpose.

Heartbeat Mechanisms: Regular status updates of how system components are performing and in good health.

Runtime Verification: Detecting faults in the behavior of system with reference to formal specifications or models.[13-15]

Fault Tolerant Communication in Embedded Systems

Distributed embedded systems have to have reliable communication. The fault tolerant communication protocols and architecture provide techniques to ensure correct data transmission in the presence of network faults and network failures (Figure 1).

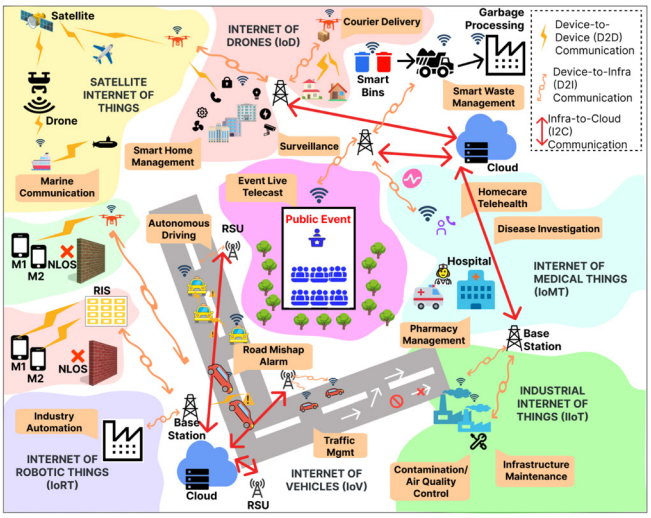

Fig. 1: Fault Tolerant Communication in Embedded Systems

Redundancy channels

Dual-Channel Communication: Transmitting data over two independent channels such that failures of one channel do not affect the other, and detect and recover from channel failures.

Multi-Path Routing: Dynamically choosing alternative paths on the fly in the event of a loss of connectivity through the network (link failures).

Channel Bonding: To increase reliability or increase bandwidth through combining multiple physical channels into a single logical channel.

Network Protocols with Fault Tolerance

Time-Triggered Protocols: Schedules that use predetermined communication protocols to ensure deterministically and fault tolerant data transmission.

Reliable Multicast: A set of protocols which guarantee reliable message delivery to a number of recipients in the face of network faults.

Error-Correcting Codes: Coding schemes for detecting and correcting errors in transmission without retransmission.

Fault-Tolerant Middleware

Group Communication Systems: Reliable and ordered message delivery middleware for groups of processes or nodes.

Fault-Tolerant CORBA: Built in fault tolerance features of Common Object Request Broker Architecture (CORBA).

Message-Oriented Middleware: Allows for a consistent reliable mode of message queuing and delivery.

Constraint Satisfaction in Real-Time Fault Tolerant Embedded Systems

Embedded systems have such real time requirements most of whose operations have a importance equal to their correctness, of which they have to occur in sometimes very short periods. Fault tolerance with real time constraints have certain unique challenges that are well understood when integrated, and such integration demands very specialized approaches.

Real-Time Fault Detection

Timing-Based Fault Detection: Tracking time execution and detecting deviations from the expected timing behaviour.

Deadline Monitoring: Mechanisms to implement in order to detect the missed deadlines in the real time tasks.

Real-Time Error Detection Codes: Using error detection schemes that can be computed and checked with short turn around time.

Real-Time Fault Recovery

Time-Bounded Rollback: Understanding specific techniques for rollback recovery with guarenteed maximum recovery time bounds.

Real-Time Checkpointing: Checkpoining strategies development that decreases overhead and provides timely recovery.

Adaptive Fault Tolerance: Dynamic fault tolerant mechanisms based on current system load and timing constraints.

Fault Tolerance Scheduling

Fault-Tolerant Scheduling Algorithms: Scheduling algorithms that take into account the actions of fault recovery while still meeting real time deadlines.

Mixed-Criticality Systems: Scheduling strategies that utilize a critical task priority while maintaining fault tolerance to all system components.

Overload Management: Reaching techniques for tackling temporary overloads due to fault recovery actions while preserving the critical real time tasks (Table 2).

Fault Tolerant Embedded System Testing and Verification

Rigorous testing and verification process is needed to verify the effectiveness of the fault tolerant mechanisms in the embedded systems. These processes validate the system's' ability to detect, diagnose, and recover from faults for a variety of conditions.

Fault Injection Techniques

1. **Hardware Fault Injection:** Adding faults into hardware components to see how systems respond.
2. **Software Fault Injection:** It is to modify software or data to simulate different fault conditions.
3. **Network Fault Injection:** Testing fault-tolerant network protocols by simulating failure communication or error.

Table 2: System Characteristics for Fault-Tolerant Embedded Systems

Characteristic	Importance
Robustness	Robustness ensures that the system can withstand environmental stresses such as extreme temperatures, humidity, and vibrations.
Reliability	Reliability is crucial to ensure that the system consistently performs its intended function, even under harsh or unpredictable conditions.
Scalability	Scalability ensures that fault-tolerant embedded systems can be expanded or adapted to meet growing demands or new requirements.
Real-Time Response	Real-time response guarantees that the system can react instantly to changes in the environment, ensuring safety and operational continuity.
Autonomous Operation	Autonomous operation allows the system to function independently, with minimal human intervention, even when failures occur in the environment.
Resource Management	Resource management ensures efficient use of limited resources (e.g., power, memory) while maintaining fault tolerance and optimal system performance.

Formal Verification Methods

4. **Model Checking:** To verify that the system meets given fault tolerance properties using formal models.
5. **Theorem Proving:** Mathematical proof techniques for the correctness of fault tolerant algorithms and protocols.
6. **Runtime Verification:** Ensuring adherence of the system execution to some fault tolerance specification.

Simulation and Emulation

7. **Hardware-in-the-Loop Simulation:** Testing of the fault tolerant embedded systems in the simulated environments using real hardware components.
8. **Software-in-the-Loop Simulation:** Fault tolerant software components have been evaluated in simulated system environment.
9. **Fault-Tolerant System Emulators:** Constructing specialized emulators capable of dissecting different fault scenarios and connecting to the system response and emulation.

Case Studies: Fault Tolerant Embedded Systems in Practice

Real world implementation of the fault tolerant embedded systems gives us a view of the practical challenges that might be experienced and the strategies that are successful. In this section, case studies taken from different domains with fault tolerance sensitivity are presented.[16-18]

Aerospace Systems

Flight Control Systems: Fault tolerant architectures used for safe and reliable flight control in modern aircraft are studied.

Satellite Systems: Fault tolerant design of long term satellite operations in the harsh space environment.

Automotive Systems

Advanced Driver Assistance Systems (ADAS): Investigating fault tolerant methods to be used in automotive safety critical applications.

Electric Vehicle Battery Management: Study of fault tolerant strategies of maintaining and safeguarding high capacity battery systems.

Industrial Control Systems

Nuclear Power Plant Control: The study of fault tolerant designs used in nuclear power plant control systems to assure safe operation.

Process Control in Chemical Plants: The design of fault tolerant architectures for safe and efficient chemical processing operations.

Medical Devices

Implantable Medical Devices: Assignments studying fault tolerant design in pacemakers and other implantable medical devices for continued reliable operational modes.

Critical Care Equipment: Faault tolerant approaches in life support systems, and other critical medical equipment.

Fault Tolerant Embedded Systems : Future Trends

Fault tolerant embedded system will further evolve with change in technology, having change in challenges. The second part of this chapter explores emerging trends and future directions in the field (Figure 2).

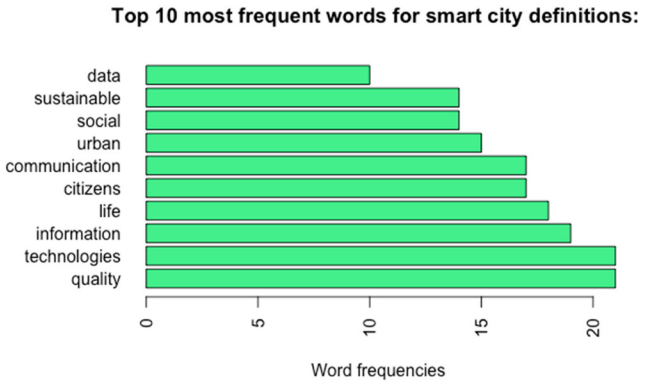

Fig. 2: Fault Tolerant Embedded Systems

Fault Tolerant Systems Using AI and Machine Learning

Predictive Fault Detection: Machine learning algorithms to predict potential fault before they happen.

Adaptive Fault-Tolerant Systems: Building systems that can learn and adapt the fault tolerant strategies from experience.

AI-Assisted Fault Diagnosis: Applying artificial intelligence techniques to enhance the accuracy and faster fault diagnosis.

Edge Computing and Internet of Things (IoT)

Distributed Fault Tolerance: Fault tolerant architecture development for large scale distributed IoT systems.

Edge-Based Fault Recovery: Going through edge, reducing latency as well as making system responsive with fault recovery mechanisms.

Secure Fault-Tolerant IoT: To integrate security management with fault tolerant designs of IoT devices and networks.

Fault Tolerant Quantum Computing

Quantum Error Correction: Study of fault tolerant techniques for quantum computing systems.

Hybrid Classical-Quantum Systems: Fault tolerant architectures combining classical and qautum computing eslements.

Fault-Tolerant Quantum Communication: Analysing methods for tolerant communication in the face of noise and error.

Conclusion

Finally, fault tolerant embedded systems are indispensable in guaranteeing the reliability and safety of the critical applications of different industries. The implementation of these systems with strong fault detection, diagnosis and recovery mechanisms allows them to still function in harsh environments as well as in the event of hardware or software failures. Operating beyond the limits of modern technology, complex computing systems range from spacecraft and industrial control applications to stock exchange systems and manufacturing plants — all reliant on fault tolerant, dependable computing resources.

References:

1. Kwon, O., & Yoo, S. K. (2021). Interoperability reference models for applications of artificial intelligence in medical imaging. Applied Sciences, 11(6), 2704.
2. Lee, S., Lee, T., Jin, G., & Hong, J. (2008). An implementation of wireless medical image transmission system on mobile devices. Journal of Medical Systems, 32, 471-480.
3. Leidy, N. K., Beusterien, K., Sullivan, E., Richner, R., & Muni, N. I. (2006). Integrating the patient's perspective into device evaluation trials. Value in Health, 9(6), 394-401.
4. Lee, H. G., Nam, S., & Chang, N. (2003, March). Cycle-accurate energy measurement and high-level energy characterization of FPGAs. In Fourth International Symposium on Quality Electronic Design, 2003. Proceedings. (pp. 267-272). IEEE.
5. Li, F., Lin, Y., He, L., Chen, D., & Cong, J. (2005). Power modeling and characteristics of field programmable gate arrays. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 24(11), 1712-1724.
6. Vallabhuni, R. R., Yamini, G., Vinitha, T., & Reddy, S. S. (2020, September). Performance analysis: D-Latch modules designed using 18nm FinFET Technology. In 2020 International Conference on Smart Electronics and Communication (ICOSEC) (pp. 1169-1174). IEEE.
7. Liang, H., Chen, Y. C., Luo, T., Zhang, W., Li, H., & He, B. (2015, September). Hierarchical library based power estimator for versatile FPGAs. In 2015 IEEE 9th International Symposium on Embedded Multicore/Many-core Systems-on-Chip (pp. 25-32). IEEE.
8. Liang, H., Chen, Y. C., Luo, T., Zhang, W., Li, H., & He, B. (2015, September). Hierarchical library based power estimator for versatile FPGAs. In 2015 IEEE 9th International Symposium on Embedded Multicore/Many-core Systems-on-Chip (pp. 25-32). IEEE.
9. Paul, M., & Petrov, P. (2010, June). I-cache configurability for temperature reduction through replicated cache partitioning. In 2010 IEEE 8th Symposium on Application Specific Processors (SASP) (pp. 81-86). IEEE.
10. Sanchez, D., & Kozyrakis, C. (2012). Scalable and efficient fine-grained cache partitioning with vantage. IEEE Micro, 32(3), 26-37.
11. Suo, G., & Yang, X. J. (2009, August). Balancing parallel applications on multi-core processors based on cache partitioning. In 2009 IEEE International Symposium on Parallel and Distributed Processing with Applications (pp. 190-195). IEEE.
12. Vallabhuni, R. R., Sravya, D. V. L., Shalini, M. S., & Maheshwararao, G. U. (2020, July). Design of Comparator using 18nm FinFET Technology for Analog to Digital Converters. In 2020 7th International Conference on Smart Structures and Systems (ICSSS) (pp. 1-6). IEEE.
13. Ferreira Martins, H., Carvalho de Oliveira Junior, A., Dias Canedo, E., Dias Kosloski, R. A., Ávila Paldês, R., & Costa Oliveira, E. (2019). Design thinking: Challenges for software requirements elicitation. Information, 10(12), 371.
14. Kuula, S., Haapasalo, H., & Kosonen, J. M. (2019). Three phases of transforming a project-based IT company into a lean and design-led digital service provider. IEEE software, 37(2), 41-48.
15. Higuchi, M. M., & Nakano, D. N. (2017). Agile design: A combined model based on design thinking and agile methodologies for digital games projects. Revista de Gestão e Projetos, 8(2), 109-126.
16. Vallabhuni, R. R., Sravana, J., Pittala, C. S., Divya, M., Rani, B. M. S., & Vijay, V. (2021). Universal shift register designed at low supply voltages in 20 nm FinFET using multiplexer. In Intelligent Sustainable Systems: Proceedings of ICISS 2021 (pp. 203-212). Singapore: Springer Singapore.
17. Bäck, T., Fogel, D. B., & Michalewicz, Z. (1997). Handbook of evolutionary computation. Release, 97(1), B1.
18. Bai, K., & Shrivastava, A. (2010, October). Heap data management for limited local memory (llm) multi-core processors. In Proceedings of the eighth IEEE/ACM/IFIP

international conference on Hardware/software codesign and system synthesis (pp. 317-326).

19. Shaik, S. (2021). Wideband rectangular patch antenna with DGS for 5G communications. National Journal of Antennas and Propagation, 3(1), 1–6.
20. Vardhan, K. V., & Musala, S. (2024). Thermometer Coding-Based Application-Specific Efficient Mod Adder for Residue Number Systems. Journal of VLSI Circuits and Systems, 6(2), 122–129. https://doi.org/10.31838/jvcs/06.02.14
21. Pradeep, M., Abinya, R., Sathya Anandhi, S., & Soundarya, S. (2017). Dynamic smart alert service for women safety system. International Journal of Communication and Computer Technologies, 5(2), 58-66.
22. 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
23. 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
24. 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
25. Surendar, A. (2024). Emerging trends in renewable energy technologies: An in-depth analysis. Innovative Reviews in Engineering and Science, 1(1), 6-10. https://doi.org/10.31838/INES/01.01.02
26. Prasath, C. A. (2024). Energy-efficient routing protocols for IoT-enabled wireless sensor networks. Journal of Wireless Sensor Networks and IoT, 1(1), 1-7. https://doi.org/10.31838/WSNIOT/01.01.01.
27. Abdullah, D. (2024). Enhancing cybersecurity in electronic communication systems: New approaches and technologies. Progress in Electronics and Communication Engineering, 1(1), 38–43. https://doi.org/10.31838/PECE/01.01.07.