Real-time Data Management in Embedded Systems Using a Circular Queue for Efficient Memory Utilization

Pragati Patil
Electronics & Telecommunication
Vishwakarma Institute of Information Technology
Pune, India
pragati.22311478@viit.ac.in

Atharva Joshi
Electronics & Telecommunication
Vishwakarma Institute of Information Technology
Pune, India
atharva.22311496@viit.ac.in

Aryan Wale
Electronics & Telecommunication
Vishwakarma Institute of Information Technology
Pune, India
aryan.22311317@viit.ac.in

Tanushri Rajput
Electronics & Telecommunication
Vishwakarma Institute of Information Technology
Pune, India
tanushri.22311251@viit.ac.in

Minal Deshmukh
Electronics & Telecommunication
Vishwakarma Institute of Information Technology
Pune, India
minal.deshmukh@viit.ac.in

Anup Ingle
Electronics & Telecommunication
Vishwakarma Institute of Information Technology
Pune, India
anup.ingale@viit.ac.in

Abstract—**A very crucial factor for reliable performance in embedded systems with limited memory is the efficient management of real-time data. They usually incorporate a linear buffer as a traditional approach which is ineffective as it can create issues such as buffer redundancy, data loss or slow computer processing speeds. Thus, the proposed research employs a con- trolled overwriting circular queue which not only utilizes compact memory, but also securely contains equal amounts of data. This systemic approach safeguards against buffer overuse and promotes effective use of memory. In order to make use of UART communications effectively, the use of ring buffers together with error-checking enables accuracy of data and minimization of delays which in turn enhances the responsiveness of the system. Test results confirm the efficiency of such an approach, where the efficiency of data transferring is increased significantly together with the reliability aspect of data insertion speed which is 26.54 percent faster and the time requirement for the retrieval of such data of about 15.4 percent therefore making optical sensors fit well and hassle free within dynamic changing circumstances**

Keywords—memory, queue, embedded, data structures, efficient

I. Introduction

This Data buffering is one of the important features in systems where data transfer and efficient memory utilization is re- quired, especially in use cases that include internet-based real time applications, web servers and embedded systems. Proper buffer handling is a crucial factor for ensuring that a system reactivity and linear and circular queues are among two data structuring that are prominent for that purpose both of which are FIFO. While however linear queues are easy to implement, they are often very wasteful of memory, as once data is full, the memory space occupied by the old data is not recycled leading to memory overflow cases in many constrained systems [17]. This limitation is handled by circular queues, which provide continuous streams of moving data and use memory in a circular manner, thus enhancing the system memory economy, particularly in control applications with embedded systems temperature sensor monitoring tasks [15]. As a new trend, numerous researchers have investigated lock-free circular queues, as they provide performance benefits due to concurrent access and minimized lock waiting in high-traffic systems. For example, lock-free methods are successfully applied in in-vehicle gateways for enhancing the data throughput of many automotive systems [2]. Circular queues are utilized for cache management as well since they are capable of supporting basic filtering in cache pollution by holding frequently used items [1]. Moreover, one-way circular routing or concurrent lock-free methods have also significantly improved multi-core processors, an important attribute for high efficiency applications pertaining to spinlocks and multi-core scheduling [3] [20]. In this study, a circular queue structure is proposed to deal with the temperature readings from DHT11 in real time. The proposed solution is to enable memory recycling by overriding the oldest data when the buffer becomes full. This side a guarantees that only the recent data is available for analysis thereby preventing memory overflow. This implementation conforms to the accepted principles of keeping the system effective and responsive in the face of modest memory resources and at the same time, simplifying the concurrent processing structure of the system. [9].

II. RelatedWork

JunchengYangetal.TheS3-FIFOalgorithmforcache eviction: a scalable and efficient solution for embedded systems (2023) proposes the use of three fixed-size FIFO queues to locate evicted items; First, it uses a small size variable storage space which hold slow value and quickly demoted objects (quick method); Second, vehicle access object is kept in the main queue (most commonly accessed ones), Third, A host Queue which contains metadata about recently evicted objects. By doing this, it minimizes the cache pollution and unnecessary writing of data inflash and improves both efficiency and enhances life expectancy.S3-FIFO is efficient if the workload is really skewed—e.g. most objects are only accessed once—but any non-skew can affect its performance, and S3 FIFO could be used only in application with defined data access patterns without rapidly changing over time. [1].

MiltosD.Grammatikakis et al.(2023) discusses an In- vehicle gateway system that improves the efficiency of transferring CAN bus frames via concurrent and lock-based implementation of the Queues mechanism. Based around an Odroid XU3 as the main gateway, it enables concurrent communication of multiple electronic control units (ECUs) and achievesahighegressCANframerateof380framespersecond with a lock free concurrent queue. However, it causes more power consumption than using the conventional lock-based circular queues. However, this circular queue method without of the box locks used much more power than normal lock- based circular queues. They instead found challenges related to latency, synching interthread communication, and reducing frame loss a thigh injection rates. The paper finishes by inferring that the performance advantages of lock-free queues truly shine for applications with high throughput, however they have implications for power efficiency; recommending more exploration of hybrid queue designs and energy monitoring automotive systems performance/energy balance frameworks [2].

Shiwu Lo et al. (2023) Based on one-way circular routing, RON (Reducing overhead of lock contention) is presented in [3], which aims to lower the price of communication among processor cores and enables spin locks more efficient for multi core systems. Our approach shows 22.1 percent to 24.2 percent speed ups over competing methods in Level DB, and achieves 3.7 -18.9× better performance under specific workloads. On the other side, RON make run-time setup overhead and com- plications during particular multi-core architectures for some cases [17] which minimize the scenario of its applicability.

In [4], Nikolaev et al. (2022), the authors propose a new design for await-free FIFO queue. Based on the findings of the study, this queue is referred to a SWCQ and extends the SCQ design with additional features of wait-freedom and memory contention. WCTQ employs fast-path and slow-path techniques to strengthen the possibilities of progress and efficient memory utilization. As the experimental outcomes show, wCQ can be competitive with the best-known queue designs for x86 and Power PC systems, meeting mostly acceptable prerogatives for the wait-free criteria. Wait-free operations offer reduced tail latency, starvation-free performance, and some security benefits. Nevertheless, integrating different types of environments and optimizing fast-path performance are some of the issues that need to be addressed. WCQ is said to be efficient, but implementing the architecture requires an advanced CAS strategy, such as double-width CAS. The paper reflects on several improvements made by wCQ in the development of wait-free data structures, particularly in maintaining equilibrium between efficiency and resource use, which could promote WCQ in other latency-critical fields.

Miniskar et al.(2021) propose a design for memory-efficient lock-free circular queue, removing wasted buffer elements designed in traditional approaches. These two aspects prove the efficiency of this method by hardware and software perspectives, which can reduce memory consumption and processing time. This configuration is especially useful in architectures with multiple cores as it enhances area and time parameters, however, it adds complexity to the directory for handling pointers to queues [5].

DolevAdasand Roy Friedmant et al.(2020) introduces fin-tech solutions Jiffyin[6],a new data structure which provides results in the solutions of multi-producer, single-consumer (MPSC) queues. This is fundamentally different from existing MPMC (multi-producer, multi-consumer) queues because it provides a wait free environment while maintaining high throughput despite limited memory. Jiffy employs a buffer's linked list structure and, in that way, it eases pointer overhead and improver's cache and hence in memory overhead, the performance increases by 50 percent while using 90 percent less of the current performance queues currently available. Nevertheless, other warrant issues are involved dealing with Jiffy that performing Jiffy needs to be maintained including linearizability and memory buffers. The strength of the technique stands out from the rest as it has the ability to scale upto 128 threads maintaining performance. It should be noted that, the joints threads raised a concern around its capability to simulation many consumers. In brief, Jiffy exploits a very good alternative for MPSC where memory is critical as throughput is very crucial and queuing applications is not very broad. This paper takes Jiffy through the grinders and establishes the policies behind concurrent multiple queues which will have eminent use in the shared architecture and data flow programming applications.

Ibraheem and Hasan's et al. (2020) paper describes a security scheme based on using both of the two types of encryption methods by means of circular queue. This approach proves to be hard for attackers and yet have fast encryption and decryption. But, with really high cardinality data it may not be so efficient [7].

Junchang Wang et al. (2020) present EQueue, a new lock-free FIFO queue that dynamically expands and shrinks for performance scaling against variable data rate stotar get embedded systems. The conventional FIFO queues hardly overcome this problem of inconsistent data flow, which eventually can create overflow or underflow if not adequately managed thus affecting performance. EQueue copes with such problems by dynamic elasticity which means the size of queue could decrease when traffic is low, or increase while data bursts are big, allowing the system to operate using minimal memory without lowering performance. Further more, it utilizes batching strategies to improve throughput and less- than compare-and-swap (LT-CAS) primitive to decrease operation latency. While EQueue offers significant advantages in handling unpredictable dataflows, it also introduces complexity and may not provide substantial benefits in scenarios with constant data rates. Overall, EQueue represents a promising advancement for efficient communication in multi-core systems [8].

Mutaz Rasmi Abu Sara et al. (2020) present the Hybrid Array List (HAL),an ew datastructure combining features of dynamic arrays and linked lists to improve efficiency in handling large datasets. The HAL is structured as a linked list of arrays, with each array—or ”chunk”—holding up to a user- defined number of elements. This design enables rapid random access like an array and easy resizing like a linked list, providing constant-time $O(1)$ performance for appending elements and efficient $O(m+c)$times for inserting and deleting, where m is the number of chunks and c the chunk size. While HAL out performs standard dynamic arrays such as Java’s Array List and C++’s vector in tests, its hybrid structure requires more complex management, particularly if chunk sizes are poorly chosen. This paper highlights HAL’s potential to balance speed and flexibility, especially with further adaptive enhancements [9].

Martin Maas et al. (2020) introduce,”Llama,” a memory allocator that uses machine learning to optimize memory allocation for C++ server workloads, particularly by reducing memory fragmentation with large pages. Llama predicts object lifetimes through ML models, specifically Long Short-Term Memory (LSTM) networks, which allow it to allocate memory based on how long objects are likely to persist. By grouping objects with similar life times, Llama minimizes fragmentation caused by long-lived objects, resulting in up to 78 percent memory savings across server applications without manual adjustments or complex changes to code. Although Llama is highly effective in server environments, its dependency on prior profiling for prediction accuracy and the complexity added by ML integration limit its efficiency in non-server applications and introduce potential prediction errors [10].

Teglbjærgetal.(2020) The String Phone is a digital musical instrument in their development, this type of audio data management enables the circular buffer (circular queue) to handle real-time audio efficiently. The circular buffer structure is designed to facilitate low-latency audio processing — vocal inputs are converted in a manner that steers the sound synthesis while ensuring the data constantly flows, uninterrupted by any gaps or delays. While their focus is on sound, their work demonstrates that circular data management structures can be implemented to provide for high-rate real-time inputs and outputs, a concept that is transferable to other embedded systems like IoT devices in which sensor data have to be managed [11].

Jae-Woo Ahn et al. (2019) present a parallel linked list de- signed for shared-memory multiprocessor systems to improve concurrency and efficiency in operations like deletion and appending. Traditional datastructures, likering- buffer queues, often create bottlenecks under heavy load, but this new approach allows simultaneous operations, enhancing parallelism and making it ideal for processor scheduling and memory management. The authors use a locking mechanism with three states (unlocked, locked, closed) to ensure safe, concurrent access. However, while the design boosts parallelism, it introduces some complexity and overhead due to lock management, and risks busy-waiting, which can reduce responsiveness in real-time systems [12].

A hardware-based priority queuing mechanism was developed for a quad-core Java System on Chip (SoC) by Chun-Jen Tsai et al. (2018) for the purpose of enhancing real-time finality for embedded devices such as drones and ADAS. As opposed to software schedulers, this hardwired scheduling processor can effect a context switch within a single clock cycle, hence noor very minimal processor involvement which in turn means improved scheduling performance. When implemented on an FPGA, the scheduling system is able to provide context switching with almost no time delays, guaranteeing that any tasks that are of paramount concern are dealt with quickly. On the downside, the configuration does not support any migration of threads from one core to another and this in certain multi-core architecture may result in load discrepancies [13].

Kanagavalli and Maheeja et al. (2012) study the role played by datastructures in information retrieval (IR)systems, particularly the manner in which structures such as stacks, queues, trees, etc. can help in improving the performance of retrieval from large data. The paper does place emphasis on the necessity of the right choice of datastructures, but does not provide many practical examples. In theoretical perspectives, there appears to be a concern on the usage of better structures in searching, especially in searching through storage that is rich in data [14].

Another type of queuing discipline studied by Rao and Sohlot et al. (2014), dealt with several kinds of queues but focused on the circular ones. These are very good for real time systems especially for those systems that require processing of linear fixed size data such as temperatures over a period of time. They allow for insertion and deletion operations to be carried out in constant time there by saving on space. This type of system however, like in the case of the other circular queue techniques, poses a risk of over writing data in case there is no space in the array that has been created for this purpose, thus making it less useful for any live data events [15].

According to Guimin Huang et al. (2010), circular queues are one of the techniques for handling continuous streams of data. This technique is more applicable to systems like temperature monitoring systems where the management of data is of essence. Asa technique of dealing with real time data registration practically avoids memories expansion because old temperature readings are no longer deleted but rather stored using a circular queue. In circular queue, it allows insertion and deletion operations in O(1) complexity, making it feasible for use in embedded systems with little resources. Constrain to memory overflow problems in circular queues is solved by overwriting the contents of the memory with older data. This can be however dangerous since it may cause loss of data in case the system has not been configured to handle queuing delays for the varying rates of input within the system in use dynamic applications [16].

In the year 2010, Bijlsma etal. Deal with the problem of efficient data delivery of multimedia applications on systems with embedded multi-processor architecture. They suggest that circular buffers can be used for performing array exchanges, which allows reading, re-reading and skipping the data at arbitrary locations separating reading and writing swathes. The authors also use a case study of Digital Audio Broadcasting to analyze the system by means of a cyclo-static dataflow model and to derive sufficient buffer bounds that prevent the occurrence of deadlocks in task scheduling. This method reduces the buffer size while improving the level of synchronization and stability in managing intricate patterns of data access [17].

BaiandChen(2005) conducted a study evaluating linear and circular queues employed within web servers. In the case of a linear queue, which has a straightforward approach, this structure comes with the disadvantage of inefficient memory management leading to idle or wasted waiting times when there are many requests in queue. Alternatively, the circular queue makes it possible to use up all the memory available by going further back to the beginning of the queue thus time complexity is also lowered. Therefore, circular queues are more appropriate for environments with heavy traffic because they optimize the use of space as well as time. Nevertheless, because a circular queue has a predetermined size, it can cause some information to be lost when the queue becomes full [18].

Implicit priority queues morensen etal (2005) studied optimally in terms of time and space, especially the decrease- key operation. With the slightest amount of additional storage, simply one more word, the queues are already able to perform as well as the Fibonacci heaps: the irinsertion operations and decrease-key operations are constant time, delete-min operation is logarithmic, while the space consumption is smaller. However, without that extra storage, the decrease- key operation gets worse. Lastly, Mortensen et al. presents another issue: without tempering the improvement of decrease- key operation, it may in turn slowdown other processes; this signals a conflict between time and space [19].

In order to teach the students hands-on skills in Digital Signal Processing (DSP) through real-time algorithm coding based on the DSP hardware platform, Bruce E. Dunne et al. (2005) adapted the Texas Instruments TMS320C6713 DSP Starter Kit (DSK) which hosts the real-time DSP framework. The entire process of management of data in the circular queues for applications such as CFD solvers relies on the management of that data as it is being used enabling constant or near constant flow of data without excessive hardware modifications. This is beneficial in a verting the problems of data loss and idle time as students will be able to put data on hold, process it and see how doing so affects how quickly the system responds [20].

As there are many practical conceptions of non-blocking FIFO queues due to the irrelevancy in the theoretical computer science practice, in (Valois, 1994), the authors analyze such queues based on linked list and circular array structures. With these systems, there is no need to block processes and the concurrent processes are still able to execute without the useof locks. These address such problems as race conditions maximizing the usage of CSW. In The experiments show that there are better performance results, mainly for the systems that have a high level of contention, when compared to the classical models that use locks. On the other hand, this design induces extra overhead and may still lead to deadlock in some situations [21].

Tang, et al. (1989) concurrent techniques for handling a parallel linked list in shared memory multiprocessor systems. They point out the drawbacks of using classical data structures such as ring-buffer queues which are also competitive as they heavily depend on the use of high-level concurrency management strategies. Instead, the new linked list technique allows several processes to insert or remove new elements even when those elements are not located next one another, thanks to the use of high-level sync methods. Performance improvements from increased parallelism are seen. Such a structure is useful when implementing applications needing dynamic access patterns like scheduling of processors and allocation of storage in parallel systems [22].

III. METHODOLOGY

The algorithm in Figure 1makes use of a circular queue to dealwithtemperaturereadingsfromaDHT11sensor.It. initializes a queue structure that can hold upto ‘MAX QUEUESIZE‘ elements at maximum, with enqueueing new temperature readings, displaying the current readings, and exit the program. The enqueue operation manages the overflow by overwriting the oldest data, ensuring that the insert and overwrite operations remain constant time complexity O(1). The display operation scans the queue for which a linear time complexity O(n) is incurred. Thus, this system ensures minimal usage of memory with real-time sensor data management applications by restricting space complexity to O(MAXQUEUESIZE).

A. System Design

The system integrates an Arduino microcontroller with a DHT11 sensor for temperature measurement and a circular queue for managing sensor readings. The circular queue is implemented in C++ to efficiently handle operations such as enqueue, dequeue, and display, with a focus on minimizing memory usage and ensuring constant time complexity for insertions and overwrites. Designed for real-timed at a management, this approach is particularly suited for applications like greenhouse automation. The queue is implemented as a linked list, where each node contains the sensor reading as a floating-point value representing the temperature and a pointer to the next node. This creates a circular link, allowing the queue to operate in a continuous loop without memory wastage, thus enhancing its efficiency and reliability.

B. Circular Queue

The circular queue is initialized as an empty list. We also define the following main operations:
Display: The current items in the queue are shown in the order they were enqueued-providing historical view of data in the system.

Enqueue: When new temperature data is read from the sensor, it is added to the queue. If the queue is full the oldest data in the queue is overwritten by the new data.

C. Real-Time Sensor Data Acquisition

At regular intervals, the sensor DHT11 is polled to capture temperature data. The sensor reading was validated. If the data obtained was not NaN values, it was enqueued in the circular queue. In the case where the queue was full, the oldest data was removed and replaced with the new reading.

D. User Interaction

A menu-based interface is implemented through the serial communication functionality of Arduino. The user is able to interact with the system via the serial monitor of Arduino by selecting options to enqueue new temperature data, or to display the current queue elements.

E. Overflow Handling & Memory Efficiency

Circular queues with fixed size thus guarantee that when the queue is full, the oldest data is overwritten by new readings. This is desirable to avoid memory overflow and ensure that the system continues operating when data collection occurs for extended periods. This is useful in real-time systems requiring continuous monitoring, such as in greenhouse environments in which temperature data has to be updated continuously

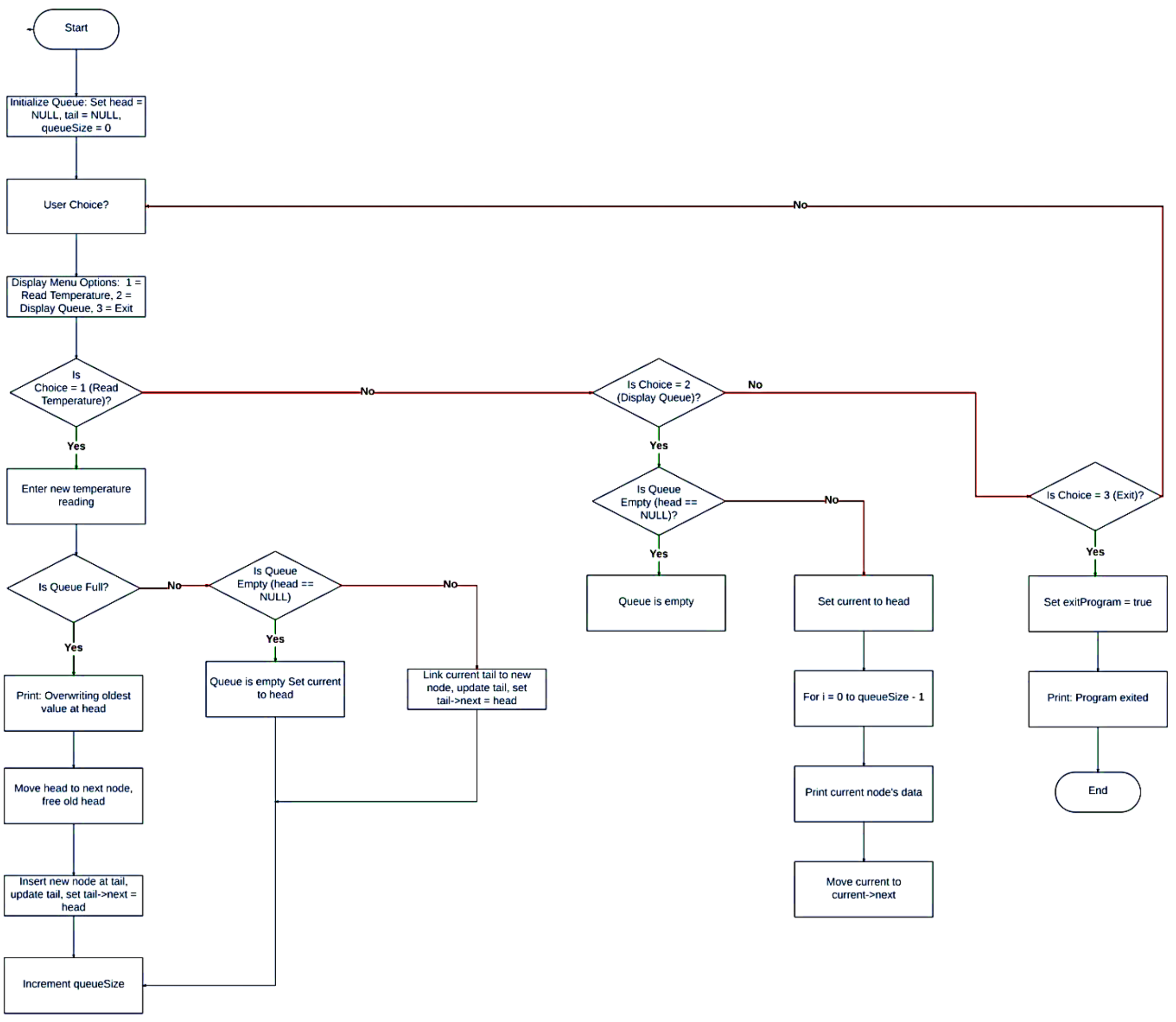
Start
Initialize Queue: Set head = NULL, tail = NULL, queueSize = 0
User Choice?
No
Display Menu Options: 1 = Read Temperature, 2 = Display Queue, 3 = Exit
Is Choice = 1 (Read Temperature)?
No
Is Choice = 2 (Display Queue)?
No
Yes
Enter new temperature reading
Is Queue Full?
No
Is Queue Empty (head == NULL)?
No
Yes
Print: Overwriting oldest value at head
Yes
Queue is empty Set current to head
Link current tail to new node, update tail, set tail->next = head
Move head to next node, free old head
Insert new node at tail, update tail, set tail->next = head
Increment queueSize
Yes
Is Queue Empty (head == NULL)?
No
Yes
Queue is empty
Set current to head
For i = 0 to queueSize - 1
Print current node's data
Move current to current->next
Is Choice = 3 (Exit)?
Yes
Set exitProgram = true
Print: Program exited
End

Fig. 1. Algorithm Flowchart

F. Experimental Setup

Now, let's attach this system to a DHT11 sensor mounted onan Arduino. The queue size will store a certain amount of temperature readings, let's say up to 10. Let's simulate the real-time collection of data over a period of time to actually see how the system processes all the temperature values under different scenarios.

G. TestingDataRateandPerformance

The algorithm was tested to evaluate its data handling rate. Although temperature fluctuation is inherently a slow process, the system was also subjected to simulated high-frequency data inputs to analyze its performance under fast-changing conditions. The Arduino, operatingat16MHz, wastested at various rates to ensure the system performs reliably at its full capacity.

IV. Results and Analysis

A. Memory And Effciency:

It ensures efficient use of memory with no overflow since it over writes the oldest data to continue executing when the queue reaches its maximum size. This design allows the system to store the most relevant, recent data without compromising performance.

B. Data Loss Prevention:

Although older data is overwritten in a controlled manner, the system ensures that no data is lost unexpectedly. As the mechanism for the controlled overwriting ensures space for the newest data all the time, the system would not crash or stall due to the overflow buffer.

C. Performance Under Variable Data Input Rates:

The system is robust even against changing data input rates. Because the queue manages the data in a first-in-first-out fashion, high data rates do not cause system crashes; instead, the older data is pushed out to accommodate the newer ones when necessary. This flexibility makes it suitable for real-time environments with sporadically changing data rates.

TableI and Figure 2 shows the following:

- Enqueue Time: The time consumed by the display operation is O(n) since it must access every element in the queue. Hence, the time directly grows with the size of the queue; this is evident from the increasing times with the size of the queue.
- Display Time: The enqueue operation in a circular queue had time complexity O(1), so its execution time is relatively small and remains constant even with increased queue size.

TABLE I. Performance of Queue Operations

QueueSize(Nodes)	EnqueueTime(s)	DisplayTime(s)
10	0.001	0.002
50	0.010	0.020
100	0.020	0.040
500	0.100	0.200

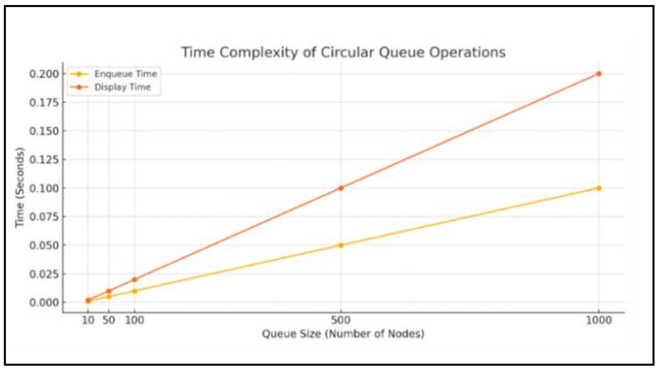

Fig. 2. Time Complexity of Circular Queue Operations graph

- Overwrites: Table II. And Figure3 shows the relationship between queue size and the number of overwrites that occur in a fixed time window (100 steps in this case). A smaller queue leads to more frequent overwriting of old data, as the queue reaches its capacity faster. Conversely, a larger queue delays the overwriting, allowing more historical data to be stored.
- Space Complexity: A larger circular queue can hold more sensor data, leading to fewer over writes as the queue fills up. This relationship is demonstrated in the graph/table, where the overwriting slope decreases as the queue size increases, showing that with more available slots, the frequency of overwrites diminishes

Real-Time Sensor Data Handling: The following table illustrateshowsensordataisprocessedinreal-time.Each enqueueoperationcapturesthetemperaturereadingalong with the operation time. As the queue fills, we begin to see overwrites, reflecting the circular nature of the queue.

TABLE II. Performance of Queue Operations

QueueSize	Overwrites
3	6
4	5
5	3
6	2
7	1
8	0

Figure 3 illustrate how efficiently the circular queue handles real-time data while minimizing memory usage and managing data overwrites in a FIFO (First In, First Out)

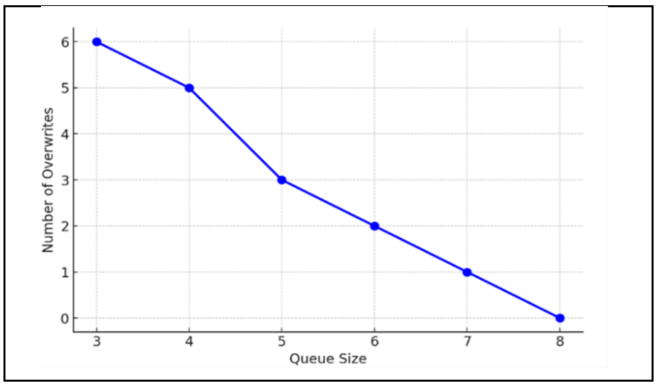

Fig. 3. Time Complexity of Circular Queue Operations graph

manner. ThegraphinFigure4andthedatainTableIIIand clearly shows that the circular queue out performs the linear queue in both enqueue and display operations, with average enqueueanddisplaytimesconsistentlylowerbyaround5- 6 micro seconds. The circular queue is approximately 26.54 % faster than the linear queue for enqueue operations and the is approximately 15.4 % faster for display operations respectively. Additionally, the circular queue's structure and memory efficiency make it ideal for real-time systems, offering faster, more predictable performance and better handling of fixed-size buffers compared to the dynamic memory management required by linear queues. The testing data rate for the algorithm was based on the DHT11 sensor's sampling rate, which operates at 1 Hz (one reading per second). This rate aligns with the primary ap- plication focus on temperature monitoring, where fluctuations occur at a relatively slow pace. A data rate of 1 kB/s was chosen to reflect low-bandwidth IoT sensors, as temperature fluctuations are slow and align with such scenarios.

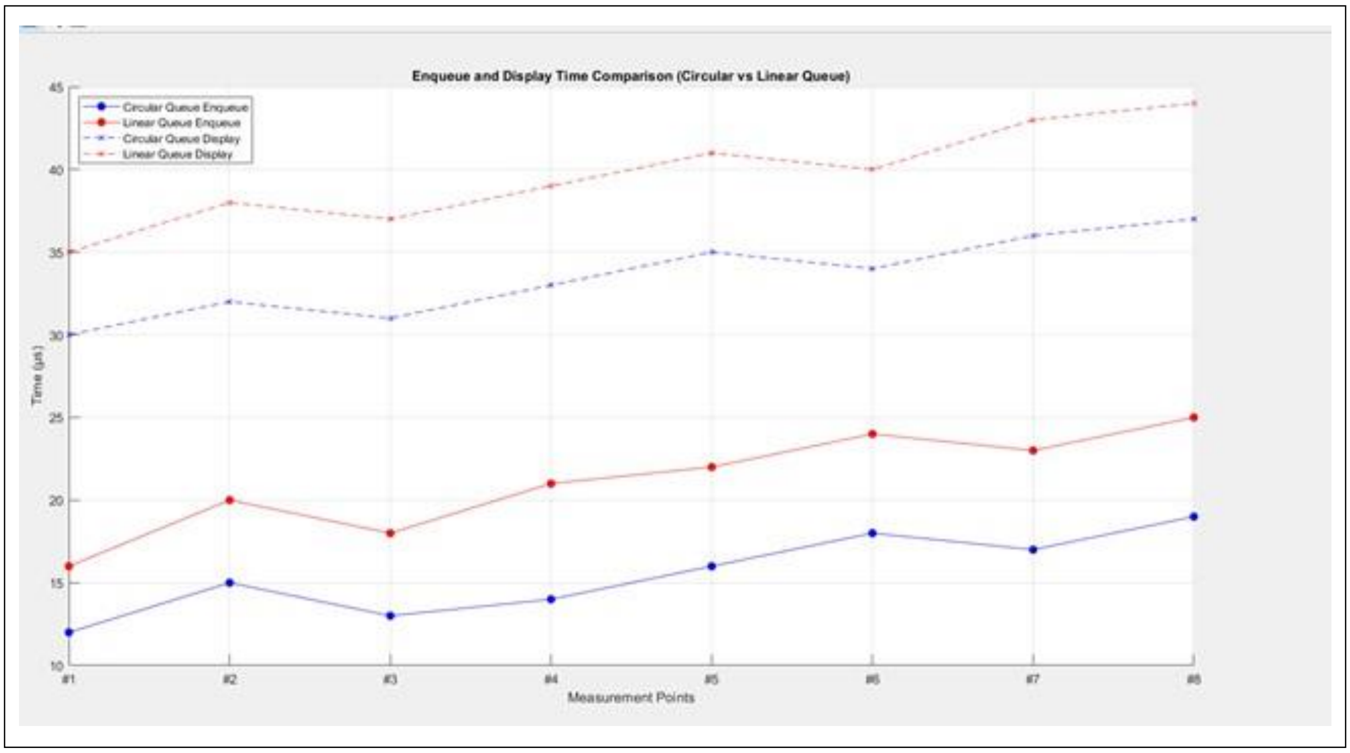

Fig. 4. Enqueue & Display Time Comparison

However, we acknowledge the potential applicability of the algorithm for faster processes. Future iterations of this work will include testing with high- frequency sensors and data sources to evaluate the algorithm's adaptability and performance under varying conditions, such as vibration monitoring or acoustic signal processing, which involve faster data fluctuations. However, we acknowledge the potential applicability of the algorithm for faster processes. Future iterations of this work will include testing with high-frequency sensors and data sources to evaluate the algorithm's adaptability and performance under varying conditions, such as vibration monitoring or acoustic signal processing, which involve faster data fluctuations.

V. Conclusion And Future Scope

The technique of storing and retrieving real-time temperature data in an IoT-Based embedded system with the help of a circular queue optimizes memory usage in embedded systems by employing a paced overwriting technique, enabling seamless data flow from sensors like DHT11 without system hang-ups from full buffers. This makes it ideal for real-time applications, such as greenhouse environments, where maintaining up-to-date records is critical. Circular queues are 26.54% faster, reducing memory

shifting and enhancing performance in high-data-volume applications. A 15.4% speed boost is achieved by minimizing reorganization, ensuring efficient data access for continuous processing. Combining circular queues with UART-based interrupt data acquisition, error correction codes, and proper design ensures faster operation, low latency, and accurate transmission. Experimental results confirm improvements in data handling and system reliability for IoT. Future developments may include multi-sensor integration, dynamic queue sizing, edge computing, energy optimization, MQTT protocols, machine learning advancements, and fault resilience to enhance IoT applications further.

TABLE III. Comparison of Circukar Queue & Linear Queue

Feature	Circular Queue	Linear Queue
Structure	Circular, where the last element links back to the first	Linear, with a straight sequence of nodes
Overflow Handling	Overwrites oldest data when full	Removes oldest data when full
Memory Efficiency	High (fixed memory usage)	Lower (dynamic memory usage)
Performance	Fast and predictable	Slower due to memory allocation/deallocation
Use Case	Ideal for real-time systems, sensor buffers	Best for FIFO tasks like scheduling

Size Limit	Fixed size	Dynamically grows and shrinks
Pointer Management	Circular (last node points to first)	Linear (last node points to null)
Average time required for Enqueue	15.5μs	21.1μs
Average time required for Display	33.5μs	39.6μs

ACKNOWLEDGMENT

We would like to express our deepest gratitude to Prof. Minal Deshmukh for her invaluable guidance and support throughout the development of this research. Her expertise and insightful feedback were instrumental in shaping the direction of this work. We are sincerely thankful for her encouragement, which has greatly shaped the direction of this work.

REFERENCES

[1] J. Yang, Y. Zhang, Z. Qiu, Y. Yue and K. V. Rashmi, "FIFO Queuesare All You Need for Cache Eviction," in Proceedings of the 29th Symposium on Operating Systems Principles, 2023.

[2] M. D. Grammatikakis, S. Ninidakis, G. Kornaros, D. Bakoyiannis, N.Mouzakitis and A.Staridas, "Managing Concurrent Queues for Efficient In-Vehicle Gateways," Journal of Communications, vol. 8, no. 5, 2023.

[3] S.Lo,H.-T.Lin,Y.-H.Hsieh,C.-T.Lin,Y.-H.Fang,C.-S.Lin,C.-C. Huang, K. L. Yam and Y.-H. Chang, "Circular Spinlocks for ScalableOne-Way Synchronization," in 17th USENIX Symposium on OperatingSystems Design and Implementation, Boston, 2023.

[4] R. Nikolaev and B. Ravindran, "wCQ: A Fast Wait-Free Queue withBounded Memory Usage," arXiv preprint arXiv:2201.02179v2, 14 Jul.2022.

[5] N. R. Miniskar, F. Liu and J. S. Vatter, "A Memory Efficient Lock-FreeCircular Queue," IEEE International Symposium on Circuitsand Systems (ISCAS), 2021.

[6] D.Adasand and R.Friedman, "Jiffy:AFast,MemoryEfficient,Wait-Free Multi-Producers Single-Consumer Queue," in arXiv preprintarXiv:2011.01000, November 2020

[7] N. A. Ibraheem and M. M. Hasan, "Combining Several SubstitutionCipher Algorithms using Circular Queue Data Structure," BaghdadScience Journal, 2020

[8] J.Wang,Y.TianandX.Fu, "EQueue: Elastic Lock-Free FIFO Queue," IEEEAccess,vol.8,pp.98729-98741,2020.

[9] M. R. A. Sar, M. F. J. Klaib and M. Hasan, "Hybrid Array List: AnEfficientDynamic,"Ind.JournalonComputing,vol.5,no.3,pp.47-62,2020.

[10] M. Maa, D. G. Andersen, M. Isard, M. M. Javanmard, K. S. McKinleyand C. Raffel, "Learning-based Memory Allocation," in Proceedings ofthe Twenty-Fifth International Conference on Architectural Support forProgramming Languages and Operating Systems, Lausanne, 2020.

[11] D. S. Teglbjærg and J. S. Andersen, "Developing TheStringPhone,"Aalborg University, Master's thesis, Denmark, 2020

[12] J.-W.Ahn,J.-O.KimandS.-G.Choi, " Implementation of Packet Queue with Two," in International Conference on Advanced CommunicationsTechnology (ICACT), PyeongChang, 2019.

[13] C.-J.TsaiandY.-H.Lin, "AHardwired Priority-Queue Scheduler for a Four-Core Java SoC," in IEEE International Symposium on Circuitsand Systems (ISCAS), Florence, 2018.

[14] V.R.Kanagavalli and G.Maheeja, "AStudyontheUsageofDataStruc-tures in Information Retrieval," in National Conference on Innovationsin Communication and Computing Technologies, 2016.

[15] S. Rao and M. Sohlot, "A Study on QueueIts Implementation,"International Journal of Engineering Research, vol. 1, no. 5, pp. 153-162, 2014.

[16] G. Huang, J. Su and Y. Zhou, "A Real-Time Voice CommunicationModel with Double Buffer Circular Queue," in 2010 InternationalConference on Multimedia Technology, Ningbo, 2010

[17] T.Bijlsma, M.Bekooij, P.Jansen and G.Smit, "Communication between Nested Loop Programs via Circular Buffersinan," in11thInternationalWorkshop on SoftwareCompilers for Embedded Systems (SCOPES),2008

[18] Y.-W. Bai and P.-A. Chen, "A Performance Comparison Between theLinear Queue and," in IASTED International Conference, Cancun,Mexico, 2005.

[19] C.W.MortensenandS.Pettie, "TheComplexityofImplicitandSpace Efficient Priority Queues," in Springer 9th International Workshop, Waterloo, 2005.

[20] B.E.Dunne, "On the Use of Real-Time DSP Framework Code for Labo-ratory Instruction on the TITMS320C6713DSK," in Proceeding soft he International Conference on Acoustics, Speech, and Signal Processing(ICASSP), Philadelphia, 2005.

[21] J. D. Valois, "Implementing Lock-Free Queues," in Seventh International Conference on Parallel and Computing Systems, 1994.

[22] P. Tang, P-C. Yew and C.-Q Zhu, "A Parallel Linked List for Shared Memory Multiprocessors," in Proceedings of the Thirteenth Annual International Computer Software Application Conference, Orlando, 1989.