Design and FPGA Implementation of a Watchdog Timer for Safety-Critical Embedded systems

Dr. Jamuna M[1], Nagasmruthi R A[2], Sanjana Balaramsingh Hajeri[3], Sanjana K[4], Sanya S[5]
[1]*Assistant Professor, Bangalore Institute of Technology*
[2,3,4,5] *Dept. of ETE, Bangalore Institute of Technology*

***Abstract*—Reliable supervision mechanisms are essential in safety-critical embedded systems where uninterrupted operation is mandatory. This paper presents the design and implementation of a hardware-based windowed watchdog timer using FPGA for monitoring microcontroller-driven sensor applications. The proposed system integrates an FPGA-based watchdog, an Arduino Nano for environmental sensing using a DHT11 sensor, and a MATLAB-based supervisory interface for real-time visualization and recovery.**

Unlike conventional watchdog timers that rely solely on timeout counters, the proposed architecture incorporates configurable frame and service windows, pattern-based verification, and UART-based supervision. The FPGA validates servicing signals and detects early, late, or incorrect refresh attempts. Upon detecting abnormal behaviour, a reset pulse is generated and transmitted to MATLAB, which initiates automatic recovery of the Arduino application.

The system was evaluated under various operating scenarios including normal execution, delayed servicing, incorrect pattern transmission, UART communication delay, and forced microcontroller freeze. Experimental and simulation results confirm accurate fault detection and reliable closed-loop recovery. The integration of deterministic hardware timing and software-based corrective action enhances robustness and makes the architecture suitable for industrial monitoring, environmental sensing, and long-duration embedded systems.

***Index Terms*—FPGA, Watchdog Timer, Windowed Watchdog, Safety-Critical Systems, UART Communication, Embedded Systems, Fault Detection, MATLAB Supervision**

I. INTRODUCTION

Embedded systems deployed in real-time and safety-critical environments such as industrial automation, medical instrumentation, automotive control, and aerospace systems are required to operate continuously with high reliability and minimal human intervention. In such applications, deterministic timing behavior and fault tolerance are critical design requirements. Any unexpected system halt caused by software deadlocks, infinite loops, stack overflows, memory corruption, communication timeouts, or delayed sensor responses may lead to degraded performance or catastrophic failure. Therefore, fault detection and automatic recovery mechanisms must be incorporated at the architectural level.

One of the most widely adopted fault-recovery mechanisms is the watchdog timer (WDT). A watchdog timer is a dedicated supervisory hardware module that monitors the correct execution of the embedded processor. It operates as an independent timer that must be periodically serviced (refreshed) by the application software within a predefined time interval. If the processor fails to issue the refresh signal within the expected time window, the watchdog timer interprets this as abnormal system behavior and initiates a corrective action, typically a system reset.

The fundamental operating principle of a watchdog timer is illustrated in Figure 1. The watchdog timer is connected to the processor's reset input line and functions independently of the main execution flow. During normal operation, the application periodically writes to a specific watchdog register or toggles a refresh signal to reset the internal countdown counter before it expires. As long as this periodic refresh occurs within the allowed interval, the system continues normal execution. However, if the processor becomes unresponsive and fails to service the watchdog before the timeout period elapses, the

watchdog asserts a reset signal. This reset forces the processor to restart, thereby recovering the system from transient faults.

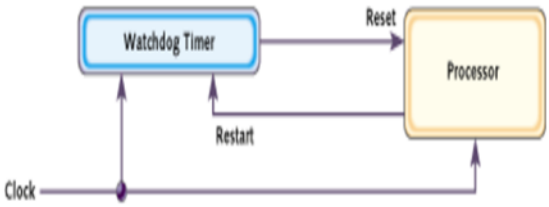

Figure 1: A typical watchdog timer setup

In Figure 1, the watchdog timer is connected to the processor's reset line. This figure shows how periodic refresh signals prevent timeout, while failure to service results in system reset.

The servicing process, commonly referred to as "kicking the dog," is illustrated in Figure 2.

Listing 1: Kicking the dog

```
uint16 volatile * pWatchdog =
   (uint16 volatile *) 0xFF0000;
main(void)
{
      hwinit();
      for (;;)
      {
         *pWatchdog = 10000;
         read_sensors();
         control_motor();
         display_status();
      }
}
```

Figure 2: Watchdog servicing ("kicking the dog") concept

The servicing operation, commonly referred to as "kicking the dog," is illustrated in Figure 2. In this process, the application software periodically resets the watchdog counter during execution of critical tasks or at predefined checkpoints in the control loop. The refresh mechanism ensures that only correctly executing software can maintain system operation. If task execution is disrupted due to timing violations, logic errors, or communication failures, the absence of the refresh signal causes the watchdog counter to overflow, resulting in an automatic reset.

Conventional watchdog timers embedded within microcontrollers are typically simple timeout-based modules with limited configurability. Their timeout intervals are often coarse-grained and dependent on software-controlled timing mechanisms. Moreover, traditional watchdogs generally detect only system inactivity (i.e., failure to refresh) but cannot identify scenarios where the watchdog is serviced too frequently due to software malfunction. Such implementations may therefore be insufficient for high-integrity systems requiring stricter supervision.

To address these limitations, this work implements a hardware-based windowed watchdog timer on an FPGA platform. Unlike conventional watchdogs, a windowed watchdog defines both a minimum and maximum allowable refresh interval. The refresh signal must occur within a predefined time window; servicing the watchdog either too early or too late is treated as a fault condition. This approach enhances fault detection capability by identifying abnormal execution timing patterns in addition to complete system stalls. The FPGA-based implementation provides high configurability, deterministic timing control, and hardware-level independence from the main processor. The watchdog module is integrated with an Arduino-based control system and MATLAB interface to establish a closed-loop supervisory architecture. This integration enables real-time monitoring, parameter tuning, and validation of system reliability under various fault scenarios, thereby improving overall system robustness and operational safety.

II. LITERATURE REVIEW

Embedded systems deployed in real-time and safety-critical applications demand high reliability, deterministic timing behavior, and autonomous fault recovery capabilities. In domains such as industrial automation, automotive control, healthcare monitoring, and distributed sensing systems, uninterrupted operation is essential to ensure system safety and functional correctness. Software anomalies including deadlocks, infinite loops, timing violations, stack overflows, and communication timeouts can cause processor stalls or unpredictable execution behavior. If such failures remain undetected, they may propagate through the system and result in performance degradation or unsafe conditions.

Watchdog timers (WDTs) are widely employed as supervisory mechanisms to detect abnormal processor behavior and initiate system recovery. A conventional watchdog timer monitors processor activity by requiring periodic refresh signals within a

predefined timeout interval. Failure to service the watchdog within the specified duration triggers a reset, thereby restoring system functionality. While internal watchdog modules integrated within microcontrollers provide basic fault detection, they typically rely on fixed timeout intervals and software-controlled servicing mechanisms. Consequently, their capability to detect subtle timing anomalies, such as premature servicing or partial task failures, is limited.

To enhance reliability and improve fault coverage, hardware-based and windowed watchdog architectures have been proposed. A windowed watchdog enforces both lower and upper timing bounds for refresh operations, enabling detection of early and late servicing faults. Implementation on reconfigurable hardware platforms such as Xilinx FPGAs allows deterministic timing control, architectural flexibility, and independence from the monitored processor. Furthermore, integration with embedded platforms like Arduino and analytical environments such as MATLAB enables closed-loop supervision, real-time parameter tuning, and performance validation.

Motivated by these considerations, this work proposes a hardware-based windowed watchdog timer implemented on an FPGA and integrated within a closed-loop supervisory framework. To contextualize the proposed approach and identify existing research gaps, a comprehensive review of related literature on watchdog architectures and fault detection mechanisms is presented in the following section.

M. Jaceintha and D. Kasturi," Design of an Improved Watchdog Timer for Safety-Critical Applications," Industrial Engineering Journal, vol. 52, no. 5, pp. 1235–1240, May 2023.

This paper [1] reviews advancements in watchdog timer designs for safety-critical VLSI systems. It highlights the limitations of traditional watchdog architectures and this work focuses on configurable watchdog architectures, the proposed system extends supervision by integrating MATLAB-based monitoring and automated recovery along with FPGA-level timing verification. The proposed design, implemented using Verilog HDL on Xilinx 14.7, offers enhanced reliability, reduced system costs, and better adaptability for real-time embedded applications.

V. R. Devi and J. Sreedhar" Design and Implementation of an Improved Watchdog Timer for Memory Applications," in Proc. 2023 Global Conf. on Information Technologies and Communications (GCITC), Karnataka, India, Dec. 2023, pp. 1–4.

This paper [2] presents an enhanced windowed watchdog timer architecture tailored for memory-based embedded systems. Designed for fault detection and autonomous system recovery, the timer operates independently of the processor using a dedicated clock. It features programmable service and frame windows, fault-tolerant mechanisms, and integration with memory units. Simulation and FPGA implementation using Xilinx Vivado demonstrate its reliability and adaptability for real-time systems.

V. Parekh, P. Divya, K. Srilatha, and P. Chitra," Design of the Configurable Watch Dog Timer using FPGA in Space Probe Application," Int. J. Innovative Technology and Exploring Engineering (IJITEE), vol. 8, no. 10, pp. 3555–3558, Aug. 2019.

This paper [3] presents the design of a configurable, processor-independent watchdog timer implemented on FPGA for space probe applications. It addresses the limitations of traditional watchdogs by introducing service, frame, and controller windows to improve timing accuracy and error detection. Unlike the reported architecture, the proposed design additionally validates servicing patterns and coordinates recovery actions through a closed-loop FPGA–MATLAB interface for system parameters such as temperature, pressure, and heat, thereby enhancing system reliability in radiation-prone environments.

R. K. Unni, V. P. Vijayanand, and Y. Dilip," FPGA Implementation of an Improved Watchdog Timer for Safety-Critical Applications," in Proc. 31st Int. Conf. on VLSI Design and 17th Int. Conf. on Embedded Systems, Pune, India, Jan. 2018, pp. 55–60, doi: 10.1109/VLSID.2018.37. Department of Electronics and Telecommunication Engineering, B.I.T. 11 FPGA Implementation of Watchdog Timer for Safety- Critical Application 2025-2026

This paper [4] proposes a processor-independent, windowed watchdog timer architecture implemented on FPGA, designed for safety-critical embedded systems. Unlike conventional watchdogs, this design supports configurable service and frame windows, multiple fault detection mechanisms, and real-time failure classification. Simulation and hardware fault

injection validate its effectiveness, demonstrating robust fault handling and efficient system resets in high-reliability environments such as aerospace and defence.

III. MATERIALS AND METHODS

Overview of the Proposed Methodology

The overall FPGA-based watchdog architecture is illustrated in Fig. 3. The design consists of a UART interface, pattern comparator, configuration registers, frequency divider, service window generator, and reset pulse logic. The architecture is fully hardware-implemented to ensure deterministic timing supervision independent of processor software execution.

The UART input serves as the communication interface through which servicing frames and configuration commands are received. Configuration writes update internal control registers, while service frames are validated before being accepted by the watchdog core.

1) Configuration Registers

The configuration registers provide runtime adaptability and allow parameter tuning without modifying the hardware description. The following parameters are stored:

- Frame Window Length (FWLEN): Defines the maximum permissible time interval between two valid servicing events.
- Service Window Length (SWLEN): Defines the valid servicing interval within the frame window.
- Clock Divider Value: Determines the effective watchdog timing resolution by dividing the system clock.
- Failure Mode Settings: Selects the type of response (e.g., immediate reset, latched failure flag, interrupt generation).

By parameterizing these registers, the watchdog can be adapted to different real-time constraints and application-specific timing requirements.

2) Pattern Comparator

To prevent unintended or accidental servicing, a two-step pattern validation mechanism is implemented. The pattern comparator verifies a predefined servicing sequence:

- Step 1: Reception of 0xAAAA
- Step 2: Reception of 0x5555

Only if the patterns are received in the correct order within the valid time window is the servicing request accepted. Any deviation in sequence, incorrect data, or incomplete transmission results in rejection of the service attempt. This mechanism enhances robustness against communication noise, software corruption, and unintended register writes.

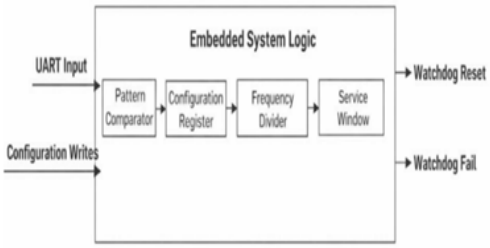

Figure 3: Block diagram of FPGA watchdog architecture

3) Frame Window and Service Window Generation

- The watchdog timing logic is structured using a dual-window mechanism to improve fault detection coverage.
- Frame Window (FW): The frame window defines the maximum allowable time interval between two successive valid servicing events. If no valid service occurs before the frame counter expires, the watchdog asserts a failure condition.
- Service Window (SW): The service window represents a valid servicing interval within the frame window. Servicing is permitted only when the internal counter lies within this predefined window. Any servicing attempt outside this window is considered invalid.
- This windowed architecture enables detection of the following fault conditions:
- Early Servicing: Service request occurs before the service window opens.
- Late Servicing: Service request occurs after the service window closes but before frame timeout.
- Missing Servicing: No valid service request within the frame window.
- If any of the above violations occur, the watchdog failure signal (WDFAIL) is asserted. Depending on the configured failure mode, the reset output (RSTOUT) is generated through the reset pulse logic. The reset pulse width is

controlled to ensure reliable system reinitialization.
- The frequency divider ensures that watchdog timing resolution is independent of the system clock frequency, allowing scalable implementation across different FPGA platforms.

B. System Architecture
- The complete closed-loop supervisory architecture is illustrated in Fig. 4. The system integrates an Arduino Nano-based embedded node, the FPGA-based windowed watchdog, and a MATLAB-based monitoring and recovery interface.
- The architecture operates as follows:

1) The Arduino Nano performs two primary functions:
- Periodic transmission of heartbeat signals.
- Acquisition and transmission of sensor data (e.g., temperature from DHT11 sensor).

2) The MATLAB interface running on the PC:
- Receives UART data from the Arduino.
- Monitors system health parameters.
- Generates validated servicing sequences (0xAAAA → 0x5555).
- Forwards servicing commands to the FPGA watchdog.
- Initiates recovery actions when failure events are detected.

3) The FPGA watchdog module:
- Receives servicing patterns via UART.
- Validates the servicing sequence using the pattern comparator.
- Enforces frame and service window constraints.
- Generates WDFAIL and RSTOUT signals when violations occur.

This separation ensures that the supervisory mechanism remains hardware-deterministic while higher-level monitoring and analytics are performed in software.

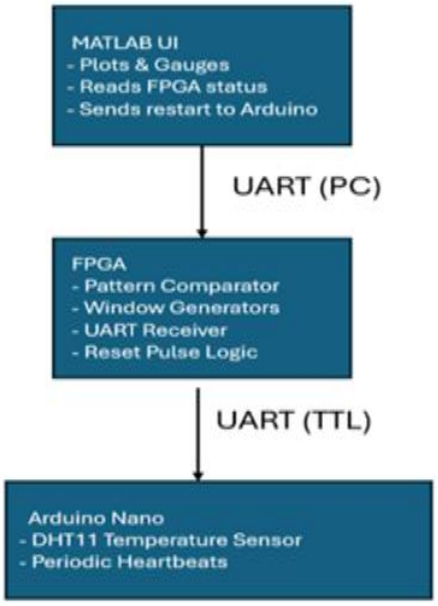

Figure 4: Overall closed-loop architecture.

Figure 5 illustrates the bidirectional data flow among the Arduino, MATLAB, and FPGA modules.
- The Arduino transmits sensor readings and heartbeat frames to MATLAB.
- MATLAB processes the incoming data and determines whether servicing conditions are satisfied.
- If the system is operating normally, MATLAB forwards the correct servicing sequence to the FPGA.
- The FPGA validates and accepts the service if timing constraints are satisfied.
- In the event of a timeout or window violation, the FPGA asserts a failure signal.
- Upon detecting this condition, MATLAB issues a restart command to the Arduino, restoring normal system operation.

This closed-loop recovery mechanism ensures:
- Hardware-level timing supervision
- Software-level monitoring and analytics
- Automated recovery without manual intervention

The architecture thus provides enhanced fault detection capability compared to traditional standalone watchdog implementations, while maintaining configurability and deterministic behavior required for real-time embedded applications.

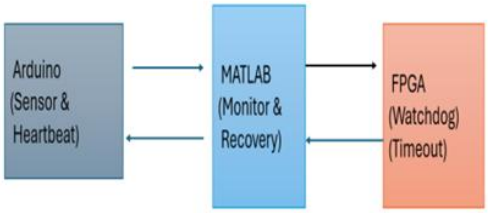

Figure 5: Closed-loop data flow

IV. RESULTS AND DISCUSSION

The proposed FPGA-based windowed watchdog system was evaluated under multiple operational scenarios to validate its functional correctness, timing determinism, and fault detection capability. The test conditions included:

1. Normal servicing within the valid service window
2. Delayed servicing beyond the frame window
3. Incorrect pattern injection (invalid servicing sequence)
4. Forced microcontroller freeze (no servicing condition)

These scenarios were designed to emulate real-world software faults such as timing violations, communication corruption, and processor hang conditions.

A. RTL Simulation Results

The synthesized Register Transfer Level (RTL) structure of the memory module is shown in Fig. 6. It illustrates the inferred memory block with clearly defined clock, reset, address, and data ports. The presence of synchronous clocking and proper reset connectivity confirms correct synthesis of the memory architecture. The structural representation verifies that the design adheres to synchronous digital design principles and that no unintended combinational loops or latch inferences are present.

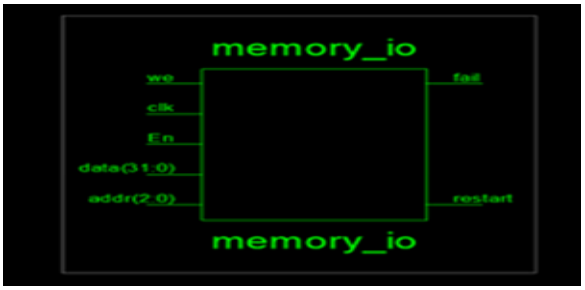

Figure 6: RTL view of memory module

The integrated waveform after module instantiation is shown in Figure 7**.** It demonstrates synchronized transitions of internal counters, memory read/write signals, configuration registers, and control logic. The waveform confirms proper inter-module communication between the UART receiver, configuration registers, window generator, and pattern comparator. The deterministic alignment between clock edges and signal transitions validates timing coherence across the integrated system.

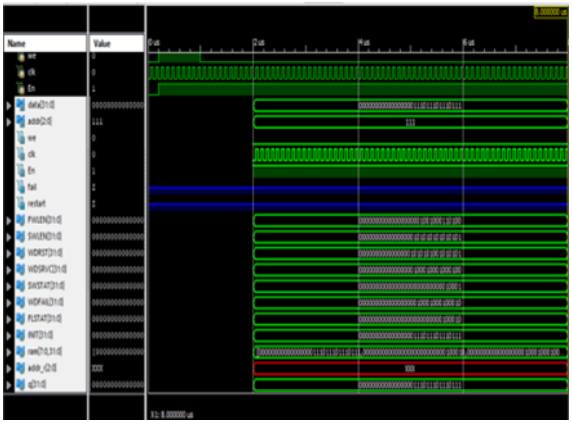

Figure 7: Integrated system waveform after module instantiation

The complete system waveform under nominal operating conditions is illustrated in Fig. 8. It shows stable operation of the clock divider, frame counter, service window generator, and reset logic. The divided clock output exhibits uniform periodicity, confirming correct frequency scaling. The frame counter increments monotonically and resets only upon valid servicing. Under normal servicing conditions, no watchdog failure (WDFAIL) or reset output (RSTOUT) assertions are observed, confirming correct window validation and system stability.

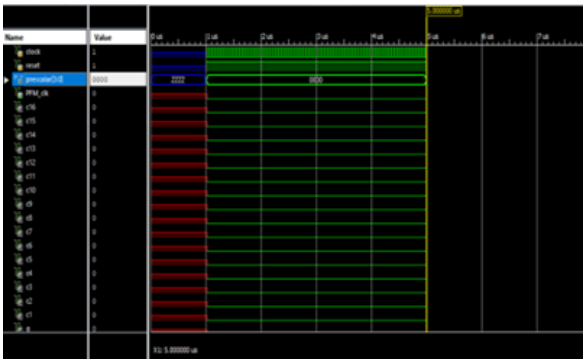

Figure 8: Integrated watchdog system waveform

B. Timer and timeout verification

The timer behaviour configured for maximum period operation is presented in Fig. 9. It shows the watchdog counter incrementing until it reaches the programmed maximum frame window length (FWLEN). Upon reaching the configured threshold without receiving a valid service pattern, the timeout logic is activated. The WDFAIL signal is asserted, followed by generation of the reset pulse through the reset pulse logic block. The waveform verifies:

- Accurate counter incrementation with respect to the divided clock
- Correct comparison against programmed FWLEN
- Deterministic assertion of failure and reset signals

The reset pulse width is observed to be stable and consistent with the configured failure mode settings, ensuring reliable system reinitialization.

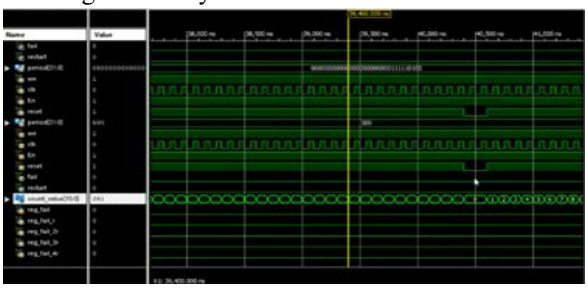

Figure 9: Timer simulation with maximum period value

C. Fault Injection Analysis

To evaluate robustness, fault conditions were intentionally injected:

1. Delayed Servicing:

When the servicing sequence was transmitted after the service window closed but before frame expiration, the watchdog correctly detected a late servicing violation and asserted WDFAIL.

2. Incorrect Pattern Injection:

Injection of incomplete or incorrectly ordered patterns (e.g., 0x5555 before 0xAAAA) resulted in rejection by the pattern comparator. The frame counter continued incrementing, eventually triggering a timeout.

3. Forced Microcontroller Freeze:

During simulated processor hang (no heartbeat transmission), the watchdog frame counter reached its maximum value and initiated automatic reset, demonstrating effective detection of missing servicing.

These results confirm that the proposed windowed watchdog not only detects complete system stalls but also identifies early and late servicing faults, which are typically undetected by conventional timeout-only watchdogs.

D. Closed-Loop Recovery Validation

In the integrated Arduino–MATLAB–FPGA setup, timeout events generated by the FPGA were successfully reported to the MATLAB interface. Upon detection of WDFAIL, MATLAB transmitted a restart command to the Arduino, restoring normal operation. This validates the effectiveness of the closed-loop supervisory mechanism in providing automated recovery.

Overall, simulation and hardware validation results demonstrate that the proposed FPGA-based windowed watchdog architecture achieves deterministic timing supervision, improved fault coverage, and reliable recovery capability suitable for real-time embedded applications.

Communication Stability Test

The communication reliability between the Arduino, MATLAB interface, and FPGA watchdog module was experimentally evaluated under multiple operating scenarios. The observed results are summarized in Table 1.

Table 1: Communication Stability Test

Test Condition	Arduino → MATLAB	MATLAB → FPGA	Result
Idle reset (normal)	Stable	Stable	No watchdog fault
Sensor read interval (2 s)	Stable	Stable	No watchdog fault
Temporary UART delay (simulated)	Occasional skip	Forward delay	Watchdog timeout triggered
Forced Arduino freeze	No data	No data	FPGA asserts WDFAIL

Table 1 indicates that the system maintains stable and reliable communication under normal operating conditions, including periodic sensor acquisition and idle reset states. No unintended watchdog events were observed during nominal operation.

Under abnormal conditions, such as simulated UART transmission delay or forced microcontroller freeze, the watchdog mechanism responded appropriately. Communication delays resulted in missed or delayed servicing sequences, leading to correct timeout detection. In the case of complete processor freeze, the absence of servicing data caused the FPGA watchdog to assert the WDFAIL signal. These results confirm the effectiveness of the proposed supervisory architecture in detecting communication failures and ensuring timely fault recovery.

Watchdog Operational States

The operational behavior of the implemented watchdog system was experimentally evaluated under multiple scenarios, including normal execution, timeout detection, and post-fault recovery.

Normal system operation is illustrated in Fig. 10. Under this condition, stable bidirectional communication between the Arduino and the FPGA is observed. The servicing sequence is transmitted within the configured service window, and no watchdog fault indication is generated. The frame counter resets appropriately upon each valid service event, confirming correct timing supervision.

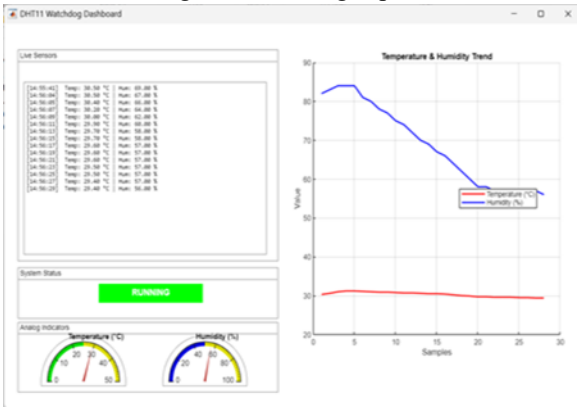

Figure 10: Normal running state of the watchdog system

To validate timeout detection capability, the Arduino application was intentionally forced into a freeze condition. In the absence of valid servicing within the programmed frame window, the FPGA watchdog detected the violation and asserted a timeout condition, as shown in Fig. 11. The waveform demonstrates the watchdog counter exceeding the configured frame window duration, followed by assertion of the WDFAIL signal, indicating correct fault detection.

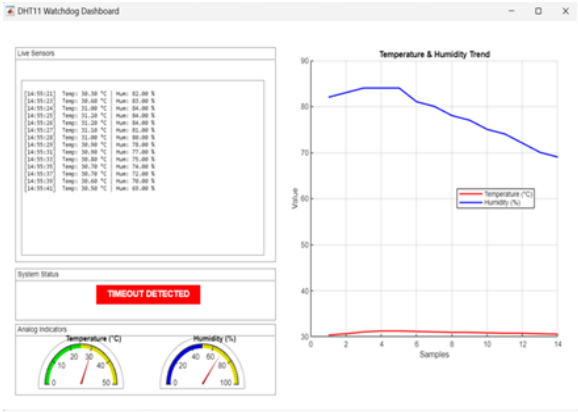

Figure 11: Watchdog timeout detected state

Following timeout detection, the recovery mechanism was triggered through the MATLAB supervisory interface. Upon receiving the failure indication, MATLAB transmitted a restart command to the Arduino to restore system functionality. The recovery sequence is depicted in Fig. 12, where normal heartbeat transmission and servicing resume after reset. The frame counter returns to nominal operation, confirming successful closed-loop restoration.

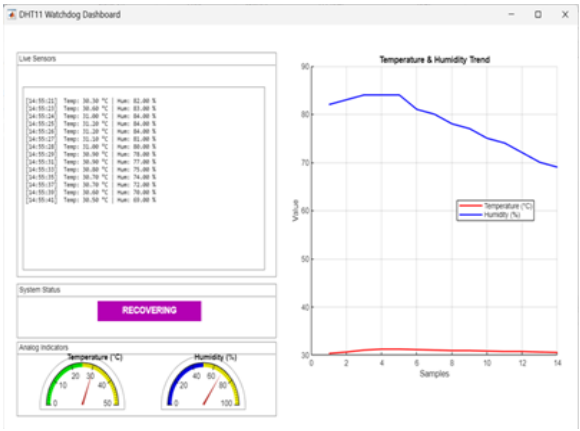

Figure 12: Watchdog recovery state

These experimental results validate the reliability of the proposed windowed watchdog architecture in detecting timing violations and autonomously restoring system operation, making it suitable for real-time and safety-critical embedded applications.

V. CONCLUSION

This paper presents a configurable FPGA-based windowed watchdog timer designed for safety-critical embedded applications. The proposed architecture combines hardware-level deterministic timing supervision with Arduino-based data acquisition and MATLAB-based monitoring and recovery, thereby establishing a closed-loop fault management framework. The implemented system effectively detects early servicing, late servicing, communication interruptions, and application freeze conditions. Experimental validation confirms accurate timeout detection, reliable assertion of watchdog failure signals, and successful automatic system restoration through supervisory intervention. By integrating deterministic FPGA timing control with software-level recovery mechanisms, the proposed approach enhances overall system robustness and fault coverage. The architecture is therefore well-suited for industrial monitoring systems and long-duration embedded applications that require autonomous fault detection and recovery capabilities.

REFERENCES

[1] Jaceintha M., Kasturi D., "Design of an Improved Watchdog Timer for Safety-Critical Applications," *Industrial Engineering Journal*, vol. 52, no. 5, pp. 1235–1240, May 2023.

[2] Devi V. R., Sreedhar J., "Design and Implementation of an Improved Watchdog Timer for Memory Applications," in *Proc. Global Conference on Information Technologies and Communications (GCITC)*, Bangalore, India, 2023, pp. 1–4. doi: 10.1109/GCITC60406.2023.10426468.

[3] Parekh V., Divya P., Srilatha K., Chitra P., "Design of the Configurable Watch Dog Timer using FPGA in Space Probe Application," *International Journal of Innovative Technology and Exploring Engineering (IJITEE)*, vol. 8, no. 10, pp. 3555–3558, Aug. 2019.

[4] Unni R. K., Vijayanand V. P., Dilip Y., "FPGA Implementation of an Improved Watchdog Timer for Safety-Critical Applications," in *Proc. 31st International Conference on VLSI Design and 17th International Conference on Embedded Systems (VLSID)*, Bangalore, India, Jan. 2018, pp. 55–60.

[5] S. N. Chau, L. Alkalai, A. T. Tai, and J. B. Burt, "*Design of a fault tolerant COTS-based bus architecture*," IEEE Transactions on Reliability, vol. 48, no. 4, pp. 351–359, Dec. 2018.

[6] V. B. Prasad, "*Fault tolerant digital systems*," IEEE Potentials, vol. 8, no. 1, pp. 17–21, Feb. 2019.

[7] J. Beningo, "*A review of watchdog architectures and their application to Cubesats*," Apr. 2017.

[8] A. Mahmood and E. J. McCluskey, "*Concurrent error detection using watchdog processors - A survey,*" IEEE Transactions on Computers, vol. 37, no. 2, pp. 160–174, Feb. 2019.

[9] J. Ganssle, "*Great watchdogs*," V-1.2, The Ganssle Group, updated January 2004, 2020.

[10] E. Schlaepfer, "*Comparison of internal and external watchdog timers' application note,"* Maxim Integrated Products, 2018.

[11] P. Garcia, K. Compton, M. Schulte, E. Blem, and W. Fu, "*An overview of reconfigurable hardware in embedded systems,*" EURASIP Journal on Embedded Systems, vol. 2014, no. 1, pp. 13–13, Jan.2018.

[12] G. C. Giaconia, A. Di Stefano, and G. Capponi, "*FPGAbased concurrent watchdog for real-time control systems*," Electronics Letters, vol. 39, no. 10, pp. 769–770, Jun. 2021.

[13] A. M. El-Attar and G. Fahmy, "*An improved watchdog timer to enhance imaging system reliability in the presence of soft errors,*" in Signal Processing and Information Technology, 2020 IEEE.

[14] B. S. Verma, S. S. Dhami, and H. S. Dhami, *"A Comparison of Various Watchdog Timer configurations for Safety-Critical Systems*," IEEE Access, vol. 6, pp. 35790- 35801, 2018.