A Fault-Tolerant Network Design Based on Distributed Control System

Xiaoyu Ding [1], Muxuan Pan [1,*], Tao Ding [1]

[1] College of Energy and Power Engineering, Nanjing University of Aeronautics and Astronautics, Nanjing, 210016, China;

Abstract. Reliability is a critical factor in communication network design of distributed control systems. To address the insufficient fault tolerance of traditional bus networks in aero-engine applications, this paper proposes a fault-tolerant network design scheme based on a braided-ring topology. Smart communication nodes of the network were designed and implemented using FPGA. Design data conversion, validation, and mode switching functional modules in a modular manner, and conduct experiments. Experimental results validated that the proposed fault-tolerant network can maintain functionality under fault scenarios such as line interruptions or data errors, demonstrating broader fault-tolerant capabilities compared to conventional bus architectures.

Keywords: Aero-engine; distributed control system; braided ring; communication fault-tolerance.

1. Introduction

With the continuous performance improvement, aero-engine's control systems have evolved from centralized to distributed architectures[1-8]. Distributed control systems integrate signal processing and communication functions at local nodes, thereby alleviating the computational burden on the central controller[9-11]. However, the distributed architecture also brings new problems, such as increased complexity in system synchronization, insufficient fault tolerance for communication failures, and reduced reliability of communication buses in local harsh environments (e.g., high-temperature zones) within the engine[12]. In response to these issues, researchers worldwide have conducted extensive studies on fault-tolerant technologies[15-19].

To address the critical demand for communication fault tolerance in aero-engine distributed control systems, this paper proposes a braided ring availability integrity network design scheme, built upon the Time-Triggered Protocol (TTP), and completes the design of smart communication nodes. Finally, hardware-software integration of the smart communication node was carried out, and communication experiments under simulated fault scenarios were conducted, validating the fault-tolerant functionality of the braided-ring network topology.

2. Design of BRAIN and Smart Node for Fault Tolerance

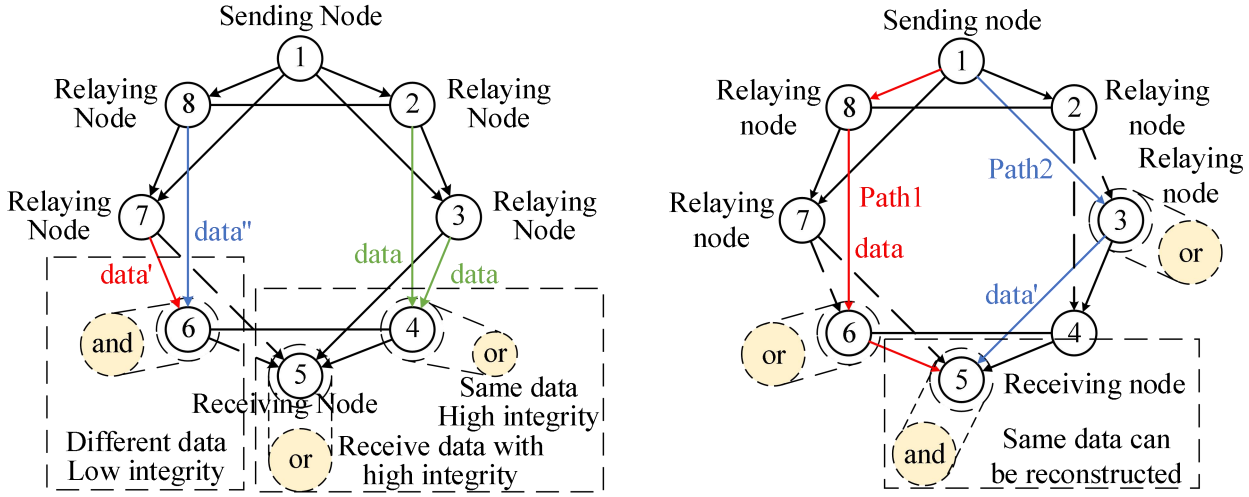
2.1 Design of BRAIN

2.1.1 Fault-tolerance mechanisms in BRAIN

The Braided-Ring Availability and Integrity Network (BRAIN) is an advanced communication topology based on a distributed system. Self-Checking Data Relaying and Data Reconstruction are the two primary fault-tolerant mechanisms through which BRAIN achieves high reliability and integrity.

The main function of Self-Checking Data Relaying is to detect data errors or node functional failures. This fault-tolerant mechanism is illustrated in Fig. 1(a): during data transmission, the sending node transmits data simultaneously via both the direct link and the indirect link. A relaying node validates the data before relaying it to subsequent nodes and, based on the validation result, transmits an integrity flag to indicate whether the data is error-free. The receiving node collects all data packets along with their respective integrity flags and subsequently selects the data with the highest integrity.

The function of Data Reconstruction is to tolerate multiple benign faults (such as node failures or link interruptions) within the communication network. Fig. 1(b) illustrates the reconstruction process under multiple benign fault scenarios. When an smart node or a local link fails, this mechanism enables segmented reorganization of the communication path, thereby enhancing data availability. The receiving node compares all received data and selects the consistent data set as the reconstructed data.



(a)Self-Checking Data Relaying (b)Data Reconstruction

Fig.1 Fault-Tolerance Mechanisms in BRAIN

For the smart nodes to perform data checking, relaying, and reception functions, several key modules must be designed, such as a data serial/parallel conversion module, a node mode selection module, and a BRAIN data checking module. The detailed design of these modules will be elaborated upon in subsequent sections.

2.2 Design of Smart Node

2.2.1 Design of data checking module

The fault-tolerant functionality of BRAIN is implemented by relaying nodes performing checking on data from different paths. In this work, an FPGA is employed to realize parallel data checking, leveraging the mapping capability of look-up tables (LUTs) to enhance the real-time performance of data processing. To ensure compatibility between processor data and the FPGA's inherent LUT6 primitives, a two-stage cascaded LUT structure is adopted in this design. The checking function is achieved by configuring the truth tables within the LUTs.Behavioral and timing simulations were conducted on the data checking module to validate its functional correctness and real-time performance.

To verify the logical correctness of the module, behavioral simulation was carried out by altering input signals A and B and observing the output signal AEQB. As shown in the behavioral simulation results in Fig. 2, when input signals A and B are unequal, the output is 0; when A and B are equal, the output is 1. These results confirm the correct logical functionality of the data checking module: it asserts a valid output (1) for matching inputs and remains invalid (0) for mismatched inputs.

To evaluate the real-time performance of the module, timing simulation was performed by dynamically changing input signals A and B during operation and monitoring the checking time. The timing simulation results are presented in Fig. 3. When inputs A and B change, the module initiates checking at the rising clock edge and produces the output after a delay of 4.52 ns. The results demonstrate that the module delay is within an acceptable range—the checking time is shorter than the clock period, allowing the result to be correctly output within the next cycle.

Through the above behavior and timing simulation, it has been verified that the data checking module meets the functional logic and timing requirements of the BRAN smart node.

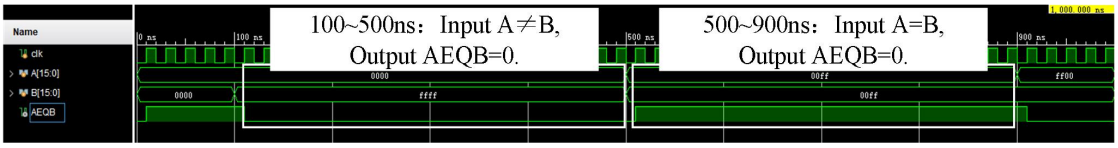


Fig. 2 Behavioral simulation of data checking

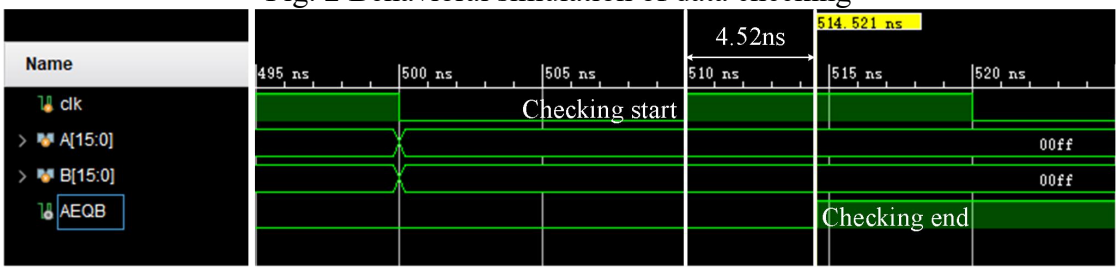


Fig. 3 Timing simulation of data checking

2.2.2 Serial/Parallel data conversion module design

BRAIN employs a differential serial protocol at the physical layer, while the smart nodes process parallel data. To address this mismatch in data transmission methods, this section presents the design of a data conversion module to enable bidirectional conversion between serial and parallel data.

In BRAIN communication, signal transmission requires a 10 MHz clock drive, with conversion operations performed in units of 10 clock cycles. This module implements data conversion control through the design of a baud rate clock and a conversion clock. During data reception, the system generates preconfigured baud rate and conversion clocks based on the start bit's level transition. Serial data is then shifted bit-by-bit into the register under the control of the baud rate clock. For data transmission, the system pre-generates corresponding baud rate and conversion clocks. Parallel data begins loading on the rising edge of the conversion clock and is subsequently shifted out bit-by-bit to the serial port driven by the baud rate clock.

Simulation tests were conducted to validate the correctness of the conversion module design. First, the function of converting serial data to parallel data was verified. The simulation results, shown in Fig. 4, demonstrate that by varying the input signal values to simulate serial data, these values are captured into the register under the baud rate clock. After a conversion clock, the 8 valid data bits in the register are output in parallel to the processor. For the parallel-to-serial conversion simulation, the results in Fig. 5 show that parallel data is loaded into the shift register at the start of the conversion clock. Driven by the baud rate clock, the data is output serially one bit at a time until all bits are completely transmitted to the serial port.

The simulation results in Figs. 4 and 5 indicate that the data conversion module correctly performs bidirectional conversion between serial and parallel data. Furthermore, at the 10 Mbps communication rate, the conversion cycle duration is 1 μs, which matches the communication rate, thereby validating the functional correctness of the designed data conversion module.

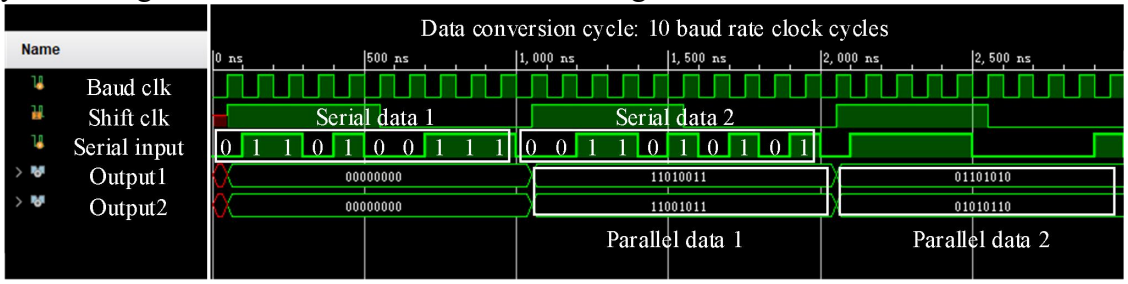


Fig.4 Functional simulation of serial input and parallel output

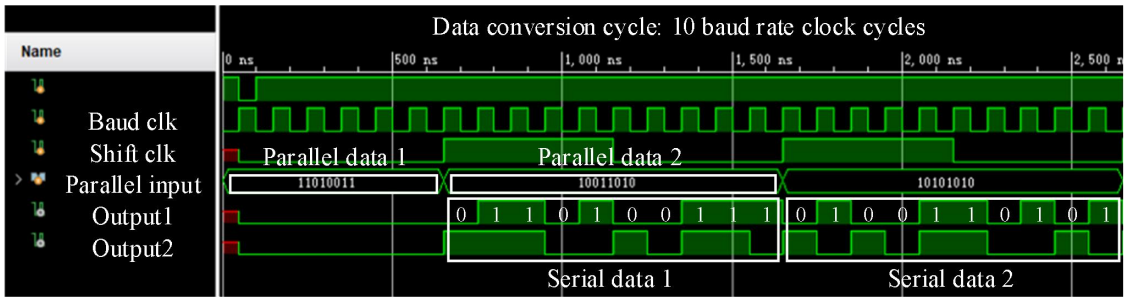


Fig.5 Functional simulation of parallel input and serial output

2.2.3 Design of the bit-width conversion module

To realize the checking function within the BRAIN smart nodes, a bit-width conversion module is required to enable mutual conversion between 8-bit and 16-bit parallel data.

During data concatenation, starting from the initial state 0, the module stores the first collected 8-bit data into a temporary register and transitions to state 1. Upon the arrival of the second 8-bit data, it is concatenated with the 8-bit data in firstbyte to form a 16-bit parallel data output, after which the module returns to state 0. During data splitting, the module latches the 16-bit parallel data to be split and outputs the higher 8 bits, then immediately transitions to state 1. In state 1, it outputs the lower 8 bits and returns to state 0.

To validate the correctness of the data bit-width conversion functionality, simulation tests were conducted on the designed module.

In the data concatenation simulation, the values 0xAB, 0xCD, 0x56, and 0x78 were input consecutively, followed by the non-consecutive input of 0x11 and 0x22. The simulation results are shown in Fig. 12. Analysis of the results indicates that when 8-bit data is input consecutively, the module correctly outputs the concatenated 16-bit data based on the flag. When 8-bit data is input non-consecutively, the module successfully concatenates the valid data according to the flag, preventing repeated sampling of data from the register.

In the data splitting simulation, the values 0xABCD and 0x5678 were input consecutively, followed by the non-consecutive input of 0xA5B5 and 0x5A5B. The simulation results are shown in Fig. 13. Analysis reveals that when 16-bit valid data is input consecutively, the module's data latching mechanism prevents the data currently being split from being overwritten, ensuring the sequential splitting of consecutively input 16-bit data. When 16-bit parallel data is input non-consecutively, the module also successfully selects valid data for splitting based on the flag.

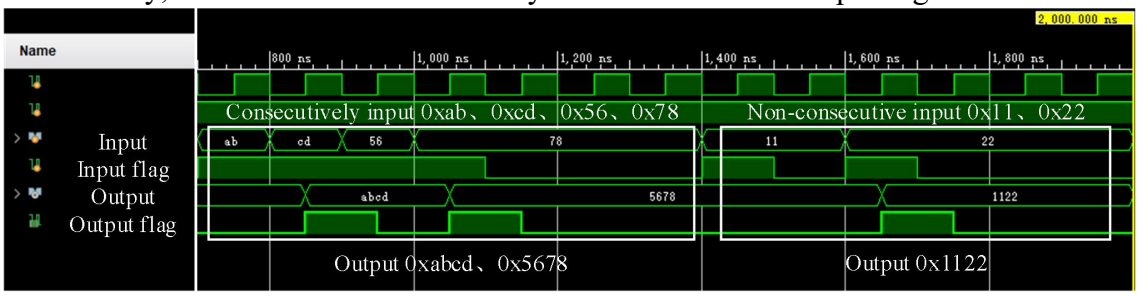


Fig.6 Functional simulation of data splicing

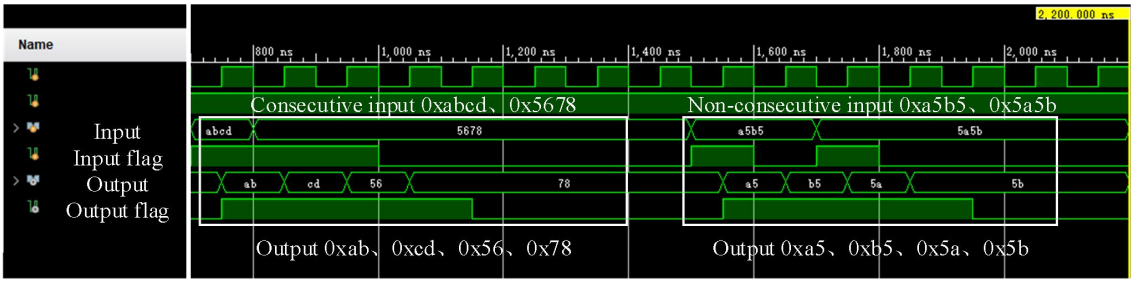


Fig.7 Functional simulation of data splitting

2.2.4 Design of node mode selection module

During the operation of BRAIN, smart nodes perform distinct functions depending on their operational mode. Consequently, a mode selection module is required to enable nodes to switch between different data processing modes based on their current functional role.

This module implements node mode switching through the design of a mode selection identifier. First, considering the number of nodes in BRAIN and the data types on the bus, an 8-bit field was selected for the mode selection identifier. The least significant 4 bits are designated as the relaying number, and the most significant 4 bits serve as the identity ID. Secondly, a unique node ID must be configured within the module. The designed 4-bit node ID ensures all smart nodes in BRAIN have distinct identities, guaranteeing data is correctly identified by the receiving node. Finally, the logic for identifying the relaying number was designed. In this module, the number for primary relaying nodes is set to 1. When a received relaying number is 1, the node switches to primary relaying mode. After completing the configuration of these module parameters, the logical functions of the node were designed around the mode selection identifier. When a node transmits data under the scheduler of the protocol controller, the module simultaneously sends the mode selection identifier. The most significant 4 bits of this identifier contain the node ID corresponding to the target recipient node, while the least significant 4 bits contain the relaying number. For a sending node, this is set to 0x1. If a node receives a mode selection identifier, it selects its mode based on the information within the identifier and increments the received relaying number by one.

Based on the above logic design, the node can switch its mode accordingly. If the least significant 4 bits (relaying number) of the received identifier are 0x1, the node acts as a primary relaying node: it increments the number and forwards both the data and the updated identifier to subsequent nodes. When the node receives other numbers, it processes and forwards the data according to the logic of the checking module described previously. The most significant 4 bits (node ID) determine the final data reception: when a node's own ID matches the node ID in the received frame, it switches to receive mode, regardless of the relaying number.

After completing the module design, its logical functionality was verified through simulation. The input flag signal values were varied to simulate the node receiving different mode selection identifiers, specifically 0x31, 0x62, and 0x52. The node's own identity was set to 0x5 during simulation.The simulation results are shown in Fig. 8. Analysis of these results demonstrates that: when the identifier is 0x31 (relaying number is 1), the node operates in primary relaying mode; when the identifier is 0x62 (relaying number is 2), the node switches to checking-relaying mode; when the identifier is 0x52 (node ID is 5, matching the node's own identity), the node switches to receiving mode.

The simulation results confirm that the designed mode selection module correctly performs mode switching based on the received identifier.

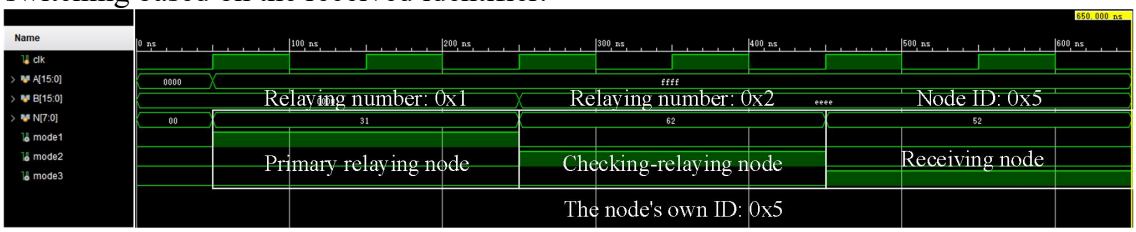


Fig.8 Functional simulation of mode selection module

2.3 Hardware-Software Integration of the node

Based on the previously described BRAIN fault-tolerant mechanisms and the design of the smart node functional modules, this section proposes a hardware architecture for the smart communication node, as illustrated in Fig. 9. The architecture primarily consists of a signal conversion module, a network communication module, an smart processing module, and a system power supply module, among others. The physical hardware implementation of the smart communication node, designed according to the above scheme, is shown in Fig. 10.

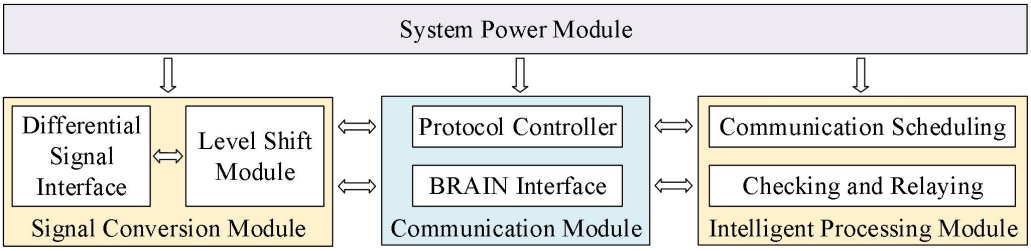


Fig.9 Hardware architecture of smart node

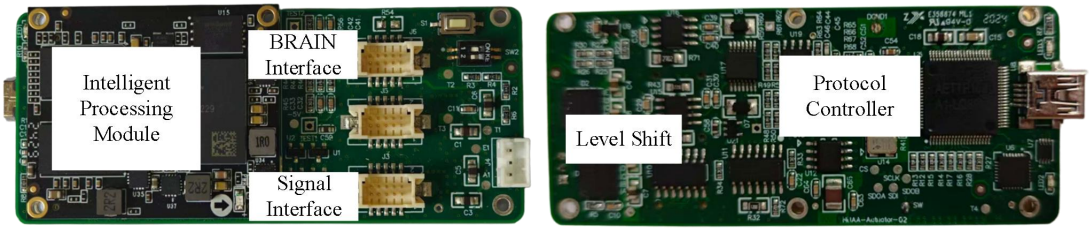


Fig.10 Smart communication node

Based on the design of the intelligent node functional modules from the previous section, an FPGA visual design was created, integrating the functional modules for serial/parallel data conversion, bit-width conversion, mode selection, and data checking described earlier. Following the completion of hardware development and program design, the intelligent communication node was integrated, and the program was solidified into the hardware's FLASH memory.

3. Experiments and Results

Based on the designed hardware circuits and software algorithms described previously, fault tolerance verification was conducted on the integrated BRAIN system. The experimental setup is shown in Fig. 11. After confirming correct equipment connections and proper functionality, communication experiments under different simulated scenarios were carried out to evaluate the system's fault-tolerant performance.

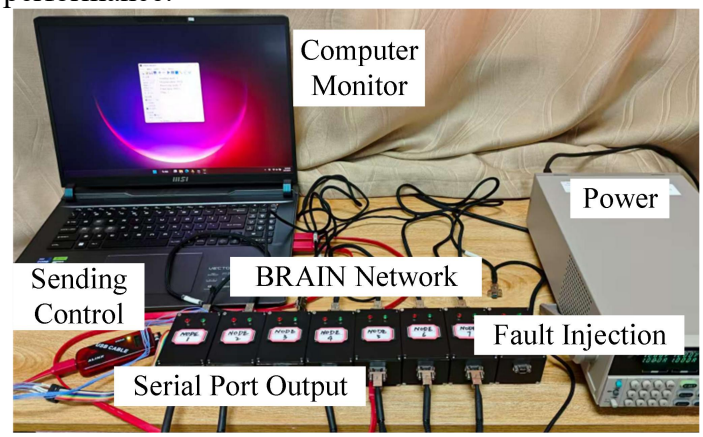


Fig.11 Experimental environment

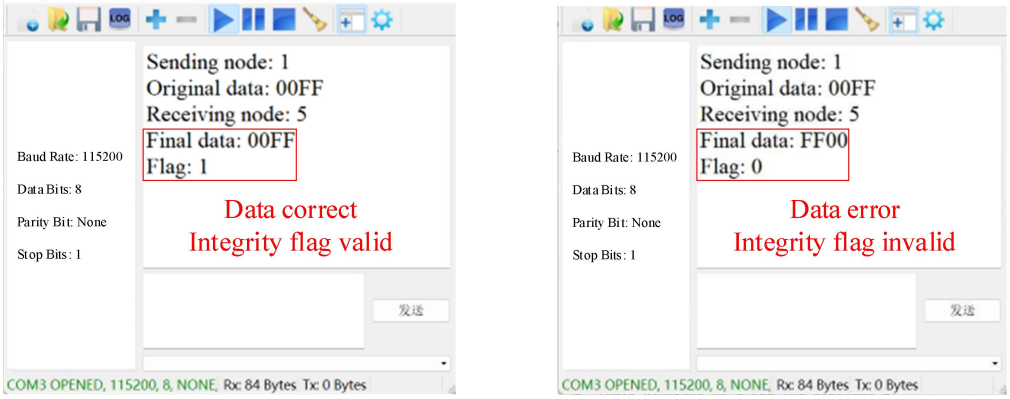
Considering the decentralized nature of BRAIN, Node 1 was configured as the transmitting node and Node 5 as the receiving node in this experiment. First, the functionality of the communication network under normal operating conditions was verified. Data 0x00FF was transmitted; after checking and relaying by the network nodes, the receiving node correctly received 0x00FF with an integrity flag of 1, indicating normal communication network operation.

To further validate the system's fault tolerance in the event of node failures, fault injections were performed according to the scenarios listed in Table 1, simulating the network's ability to tolerate benign faults (e.g., halted nodes, data errors). The transmitted data was 0x00FF. Faults were induced by either powering down the faulty node or instructing it to transmit different erroneous

data. Fig. 12 shows the results of successful and unsuccessful BRAIN fault tolerance. In subfigure (a), the receiving node's integrity flag is 1, indicating successful fault tolerance by BRAIN; in subfigure (b), the integrity flag is 0, indicating a fault tolerance failure. The fault tolerance results of BRAIN are shown in Table 1, where "T" indicates fault tolerance success and "F" indicates fault tolerance failure. The results demonstrate that the BRAIN system designed here can tolerate up to four node faults.

Table.1 Fault-tolerance performance of BRAIN under different faults

Number of Fault	Faulty Node IDs					
1	2	3	4	6	7	8
	T	T	T	T	T	T
2	2/8	2/7	2/6	3/7	3/6	4/6
	T	T	T	T	T	T
3	2/6/7	2/6/8	2/7/8	3/6/7	3/6/8	3/7/8
	F	T	F	F	T	F
4	2/3/6/7	2/3/6/8	2/3/7/8	2/4/6/7	2/4/6/8	3/4/6/7
	F	F	F	F	T	F



（a）Successful fault tolerance （b）Unsuccessful fault tolerance

Fig.12 Results of communication fault-tolerance

4. Conclusion

In response to the communication fault tolerance requirements of aero-engine distributed control systems, this paper conducted relevant research. A fault-tolerant network design scheme employing a braided-ring topology was proposed. Based on FPGA, a node relaying driver incorporating modules such as data checking, data conversion, and mode selection was designed, and a BRAIN communication system was established. Experimental results demonstrate that the system can tolerate multiple node failures and local communication link interruptions, exhibiting broader fault-tolerant capabilities compared to traditional bus architectures.

The braided-ring topology designed in this work provides a foundation for further research. Subsequent efforts could focus on refining communication protocols and optimizing data relaying modes to further enhance the system's real-time performance.

References

[1] GAO Yahui, NI Yebin, JIANG Chengping, et al. Research status and prospect of aeroengine control systems and key technologies[J]. Journal of Nanjing University of Aeronautics & Astronautics, 2024, 56 (4): 577-596.

[2] Sisson P B, Faymon D K. Digital Control Brings Large Turbofan Benefits to the Regional Jetliner Turbofan Market[J]. 1994. DOI: 10. 1115/94-GT-130.

[3] Jaw L C, Garg S. Propulsion Controls Technology Development in the U. S. – A Historical Perspective[R]. NASA/TM—2005-213978, 2005: 1-20.

[4] SUN Jianguo. Prospects for aero-power control facing the 21st century[J]. Journal of Aerospace Power, 2001(02): 97-102.

[5] GUO Yingqing, ZHANG Hong. Overview of distributed control system for aero-engines[J]. Aeroengine, 2003(3): 52-55.

[6] Dennis Culley, Randy Thomas, Joseph Saus. Concepts for Distributed Engine Control[J]. 43rd AIAA/ASME/SAE/ASEE Joint Propulsion Conference & Exhibit, 2007.

[7] Pakmehr M, Mounier M, Fitzgerald N, et al. Distributed control of turbofan engines[R]. AIAA 2009-553, 2009:1-19.

[8] LI Ruichao. On Key Technologies of Distributed Engine Control System[D]. Xian: Northwestern Polytechnical University, 2019.

[9] YIN Yi. Overview of the Development of Smart Sensor Technology[J]. Microelectronics, 2018, 48(04): 504-507+519.

[10] Rama Y, Mike W, Alireza B. The Role of Various Real-time Communication Data Bus for Open System Distributed Engine Control Architectures for the Future[C]. San Diego, California: 47th AIAA/ASME/SAE/ASEE Joint Propulsion Conference & Exhibit. 2011: 6145.

[11] XU Ke. Distributed Control System for Aero-Engines[D]. Nanjing University of Aeronautics and Astronautics, 2004.

[12] H.A. Thompson, H. Benitez-Perez. A CANbus-based safety-critical distributed aeroengine control systems architecture demonstrator[J]. Microprocessors and microsystems 23 (1999) 345-355.

[13] LIU Dongdong, ZHANG Tianhong, CHEN Jian, YU Bing. Research on key characteristics of TTP/C protocol[J]. Computer Measurement & Control. 2012, 20(10): 2769-2772.

[14] CHEN Jian. Design and verification of TTP/C bus controller based on FPGA[D]. Nanjing University of Aeronautics and Astronautics, 2012.

[15] Hall B, Paulitsch M, Driscoll K. FlexRay BRAIN fusion a FlexRay-based braided ring availability integrity network[J]. SAE Transactions, 2007: 460-473.

[16] Hall B, Driscoll K, Paulitsch M, et al. Ringing out fault tolerance. a new ring network for superior low-cost dependability[C]//2005 International Conference on Dependable Systems and Networks (DSN'05). IEEE, 2005: 298-307.

[17] Koopman P, Driscoll K, Hall B. Selection of cyclic redundancy code and checksum algorithms to ensure critical data integrity[R]. United States. Department of Transportation. Federal Aviation Administration. William J. Hughes Technical Center, 2015.

[18] Driscoll K R, Hall B. Controller area network braided ring: U.S. Patent 11,652,663[P]. 2023-5-16.

[19] GUAN Yue. Research on communication bus of distributed control system for aeroengine[D]. Nanjing University of Aeronautics and Astronautics, 2013.

[20] Pakmehr M, Khamvilai T, Behbahani A R, et al. Applying Zero Trust Principles to Distributed Embedded Engine Control Systems[C]//AIAA AVIATION 2022 Forum. 2022: 3480.