PROJECT DESIGN DOCUMENT

Multi-Node Token-Ring Network Over UART

Prepared for learning purposes
Prepared by: Ameed Othman, Embedded Software Developer
10 December 2024

1. INTRODUCTION

1.1 Purpose

This document defines the architectural blueprint and communication protocol for the UART-based token-ring network. It details the logical layers, frame formats, state machines, and timing strategies that will guide subsequent implementation activities.

1.2 Scope

The scope includes all firmware layers (from the UART interface to the application-level message exchange), as well as integration with the Zephyr RTOS. This design provides a foundation that can be scaled and adapted as the network grows or gains new features.

2. SYSTEM ARCHITECTURE

2.1 Layered Architecture Overview

The system is structured into four primary layers:

Hardware Abstraction Layer (HAL):

- **Responsibilities:** UART initialization, interrupt handling, and fundamental RX/TX operations.
- **Outputs:** Provides a clean, hardware-agnostic UART interface to upper layers.

Data Link/Token Management Layer (DLL):

- **Responsibilities:** Implements token passing, frame construction/deconstruction, CRC checks, and error handling (e.g., token loss, corrupted frames).
- **Outputs:** Presents functions for sending/receiving validated frames and managing the token lifecycle.

Application Layer:

- **Responsibilities:** Interacts with sensor drivers, retrieves or sets configuration parameters, and packages application-level data into frames. Interprets incoming frames to update settings or report received data.
- **Outputs:** High-level data structures representing sensor readings, configuration commands, or diagnostic information.

RTOS Integration Layer:

- **Responsibilities:** Maps protocol logic onto Zephyr tasks, semaphores, and message queues. Ensures deterministic scheduling for token handling and data processing.
- **Outputs:** Provides reliable timing and synchronization guarantees that enable the token-ring to function predictably in real-time.

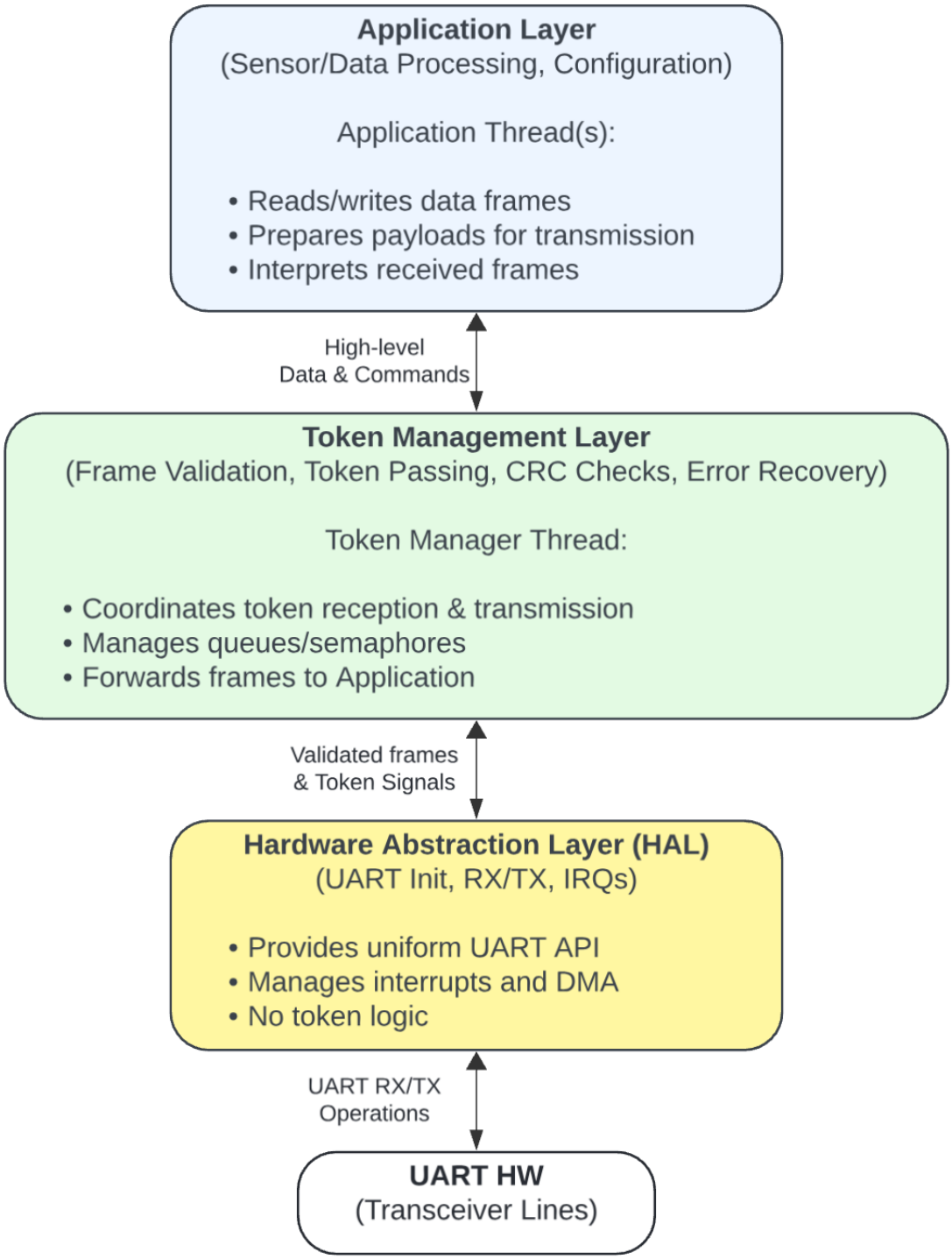

Figure 1: Block Diagram (Layered Architecture)

3. PROTOCOL SPECIFICATION

3.1 Frame Types

- **Token Frame:**
  - Format (example):
    - Start Delimiter (1 byte): 0xAA
    - Token ID (1 byte): A unique identifier for the token's circulation sequence
    - CRC (2 bytes): 16-bit CRC for error detection
- **Data Frame:**
  - Format (example):
    - Start Delimiter (1 byte): 0xBB
    - Node ID (1 byte): Identifies the transmitting node
    - Payload Length (1 byte): Length of the following payload
    - Payload (N bytes): Sensor readings, configuration commands, or diagnostic data
    - CRC (2 bytes): 16-bit CRC to ensure integrity

3.2 Timing Constraints

- Token Rotation Interval: Define a target interval (e.g., 50 ms for a ring of three nodes). Each node must pass the token within a specified maximum delay (e.g., 5 ms after finishing its data transmission).
- Timeouts: If a node does not receive a token within a predefined period (e.g., 2x the expected token rotation interval), it initiates a token regeneration sequence.

4. STATE MACHINE AND BEHAVIOR

4.1 Node States

- **IDLE:** Node awaits the arrival of a token frame.
- **TOKEN_RECEIVED:** Node validates token frame and prepares data frame if needed.
- **DATA_TRANSMISSION:** Node sends its data frame and then transitions to token forwarding.
- **TOKEN_FORWARDING:** Node transmits a fresh token frame to the next node.
- **ERROR_RECOVERY:** Triggered by timeouts or CRC failures; node may regenerate the token or re-sync after an error.

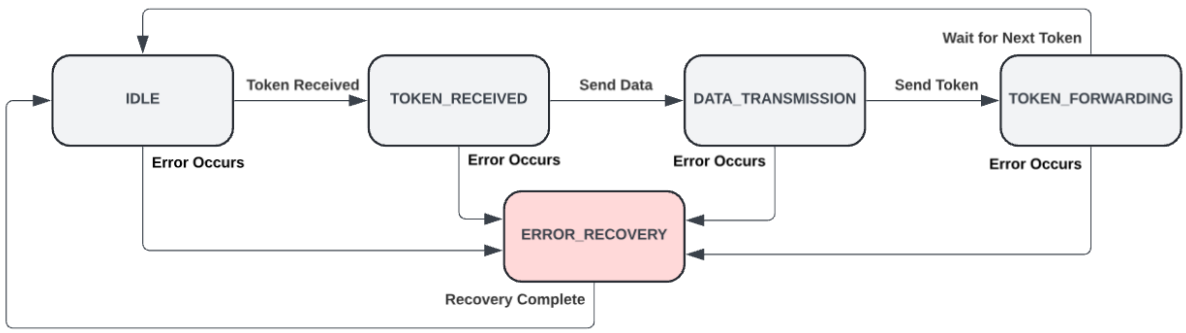

Figure 2: State Machine Diagram

5. RTOS INTEGRATION

5.1 Task/Thread Structure

- **Token Manager Thread:** High-priority thread handling UART interrupts, token arrival, and token dispatch. Uses semaphores to coordinate with the application.
- **Application Thread:** Processes incoming data frames, reads sensors, and prepares outgoing payloads. May run at a slightly lower priority than token management to ensure timely token handling.

5.2 Synchronization and Queues

- **Message Queues:** Used for passing received data frames from the Token Manager to the Application Layer.
- **Semaphores:** Employed to ensure only one thread modifies shared buffers at a time, and to signal when the application has new data ready for transmission.

6. SCALABILITY AND EXTENSIBILITY

6.1 Adding More Nodes

By adhering to this layered architecture, adding a fourth or fifth node involves simply extending the UART ring and updating configuration parameters. The token passing logic remains unchanged, as it inherently supports multiple nodes.

6.2 Additional Features

Future enhancements, such as node discovery or dynamic configuration of baud rates, can be integrated by adding payload commands interpreted at the Application Layer and corresponding state transitions in the Token Management Layer.

7. PRELIMINARY INTERFACE SPECIFICATION

7.1 HAL API Example

- hal_uart_init()
- hal_uart_write(uint8_t *data, size_t len)
- hal_uart_read(uint8_t *buf, size_t *len)

7.2 Token Management API

- token_manager_process_frame(uint8_t *frame, size_t len)
- token_manager_send_data_frame(uint8_t node_id, uint8_t *payload, size_t payload_len)
- token_manager_init() to start token rotation (if this node is designated as a start node)

7.3 Application Layer API

- app_send_sensor_data(uint8_t *data, size_t len) for preparing outbound frames
- app_receive_data(uint8_t *data, size_t len) callback triggered when a new valid data frame is available