Technical Test Assignment

TCP/IP Communication Linux and FreeRTOS Test Equipment

1. Purpose of the Assignment

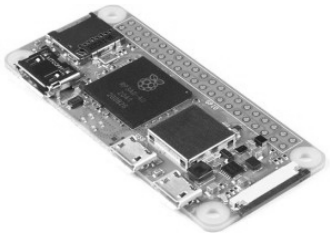
The purpose of this assignment is to evaluate the candidate's ability to:

- Develop network applications under **Embedded Linux**
- Understand and implement **TCP/IP client–server communication**
- Interoperate with an embedded device running **FreeRTOS**
- Write clean, maintainable, and well-documented embedded code
- Analyze network behavior, reliability, and error handling in constrained systems

2. Task Overview

AJAX provides to the candidate two embedded device boards:

1. **Device A (**Raspberry Pi Zero / Zero 2 W board**)** – runs **Embedded Linux**
2. **Device B** (Wi-Fi ESP32-CAM) – runs **FreeRTOS** with a TCP/IP stack (e.g., lwIP)



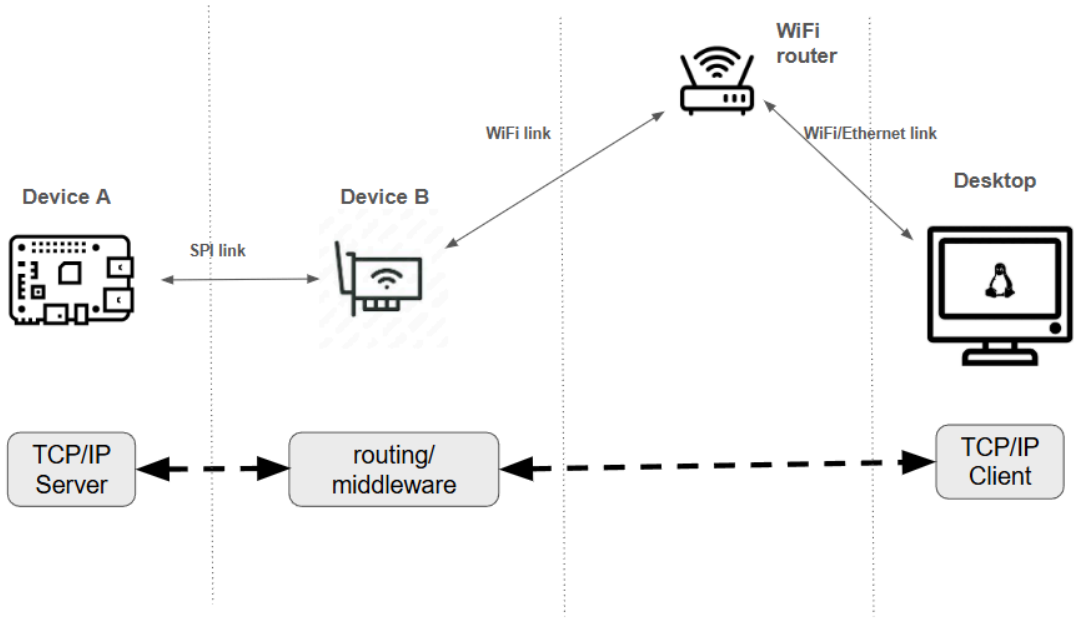
Additionally, you need (**not provided** by AJAX):

1. PC with desktop Linux
2. Wi-Fi router supports **Wi-Fi 802.11g/n 2.4GHz**

The test task is:

1) Setup SPI connectivity between the devices (parameters of connection is defined by developer)
2) Develop and bring-up network solution that allows the transfer some network data between Device A and Linux Desktop, using device B as specific router/bridge (links speed up to 600 kbit/s)
3) Develop and implement a simple TCP/IP client (application of Desktop Linux) – server solution (server application of Device A) that transfers test data (raw binary and text) from one device to the other over an IP network. The protocol of communication must be clearly defined and justified in your design.

Simple diagram of test environment:



4. Functional Requirements

4.1 Communication

- Use **TCP/IP**
- One device acts as a **server**, the other as a **client**
- Connection must support:
    - Establishing a TCP connection

    - Sending test data
    - Receiving and validating data
    - Graceful connection close

4.2 Test Data

- Transmit structured test data, for example:
    - Text string
    - Binary packet with header + payload
- Data must include:
    - Sequence number
    - Payload length
    - Simple integrity check (CRC or checksum)

4.3 Reliability

- Handle:
    - Connection timeouts
    - Reconnection attempts
    - Partial reads/writes
    - Invalid or corrupted data

5. Non-Functional Requirements

5.1 Embedded Linux Side

- Programming language: **C or C++**
- Use **POSIX sockets API**

- Application must:
    - Compile with GCC
    - Run in user space
    - Log key events (connection, errors, data transfer)

5.2 FreeRTOS Side

- Use the available TCP/IP stack (e.g., **lwIP**)
- Code must:
    - Run as one or more FreeRTOS tasks
    - Avoid blocking the scheduler indefinitely
    - Respect memory constraints
- Networking API usage must be documented

6. Performance and Resource Constraints

- Measure and report:
    - Data transfer latency
    - Throughput (approximate)
- Memory usage on FreeRTOS side should be minimized
- CPU utilization should be reasonable for both devices

7. Error Handling and Robustness

Your solution should demonstrate:

- Defensive programming

- Clear error reporting
- Recovery from network disruptions
- Proper cleanup of resources (sockets, memory, tasks)

8. Deliverables

The candidate must provide:

1. **Source Code**
    - Linux application
    - FreeRTOS networking code
2. **Build Instructions**
    - Toolchain used
    - Build and flashing steps
3. **Architecture Description**
    - Client/server roles
    - Data format
    - Task/thread model
4. **Test Report**
    - Test scenarios executed
    - Observed behavior
    - Known limitations

9. Result Analysis

The analysis section must include:

- Confirmation of successful data transfer
- Packet loss or retransmission observations
- Behavior under:
    - Network disconnection
    - Server restart
- Identified bottlenecks or weaknesses

10. Evaluation Criteria

The solution will be evaluated based on:

Criterion	Description
Correctness	Functional TCP communication
Code Quality	Readability, structure, comments
Embedded Awareness	Efficient use of RTOS resources
Error Handling	Stability and robustness
Documentation	Clarity and completeness

11. Optional Extensions (Bonus)

- TLS-based secure communication
- Configuration via command-line arguments or config file
- Support for multiple clients
- Packet retransmission at application level
- Logging via syslog (Linux side)

12. Time Expectations

Estimated completion time: **12–16 hours**