PROJECT ABSTRACT

Modbus Device Simulator for Industrial IoT Testing

A Modbus RTU/TCP device simulator: live telemetry (temperature, pressure, flow) mapped to a full register/coil map, a master console for composing real requests with byte-level frame inspection and CRC-16 decode, scenario presets and fault injection for gateway testing. Built to order with source, protocol documentation, testing guide and complete viva kit.

01. Title

Modbus Device Simulator for Industrial IoT Testing

02. Introduction / Background

Testing an industrial IoT gateway against real Modbus hardware is slow and expensive: you need the PLC or sensor, the wiring, and you cannot easily make it fail on demand to check your error handling. A simulator solves this — a virtual Modbus slave that answers requests exactly like the real device, with telemetry you can drive into alarm states and faults you can inject at will. This project builds that simulator as an interactive tool: a device model generating realistic telemetry (motor temperature with thermal drift, line pressure, coolant flow) mapped to holding/input registers and coils, a browsable register map, and a master console where requests are composed and inspected byte by byte, including CRC-16.

03. Problem Statement

Gateway and SCADA-client developers need real protocol frames to develop against, and real hardware is scarce, slow to reconfigure and incapable of failing on demand. This project provides a faithful Modbus slave in the browser — correct RTU/TCP framing, live registers, scenario presets and fault injection — so protocol handling, register parsing and failure behaviour can be built and tested without any hardware.

04. Objectives

- Simulate a Modbus slave with live telemetry (temperature, pressure, flow) driven by a drift-plus-noise device model, ticking every 400 ms.
- Expose a full register map: 6 holding registers, 2 input registers, 4 coils, 2 discrete inputs with Modbus address notation, live values, hex view and R/W flags.
- Provide a master console composing real requests for function codes 01/02/03/04/05/06/16 over Modbus RTU or TCP.
- Render every request/response frame as annotated hex with CRC-16 highlighted and a plain-English decode.
- Inject faults: dropped responses, exception replies, added latency.
- Ship scenario presets: normal operation, overheating motor, pressure spike, flaky comms.
- Deliver the protocol documentation, testing guide and full viva kit (report, PPT, Q&A;).

05. Proposed Methodology

A device model updates temperature, pressure and flow every 400 ms from drift-plus-noise dynamics toward scenario-dependent targets; each value is scaled into its Modbus register (e.g. temperature ×10 into holding register 40001), and coils reflect digital outputs like motor run and alarm. The master console builds a request PDU for the chosen function code, wraps it in RTU framing (slave ID + CRC-16/Modbus, polynomial 0xA001) or a TCP MBAP header. The virtual slave parses the frame, validates the CRC, executes the read/write against the device model, and builds the response. Fault-injection hooks sit in the response path; scenario presets retarget the telemetry dynamics.

06. System Architecture / Working

Three panes work together: the register map panel shows every address live with read/write access flags (R/W registers and coils are clickable to write — setpoints and control bits respond immediately in the telemetry); the master console composes and sends requests, rendering both frames as annotated hex with timing and a decoded explanation; and a live trend chart plots temperature, pressure and flow over a rolling 48-second window. Fault toggles and scenario presets sit alongside, so a tester can drive the motor into overheat, watch the alarm coil flip, and verify how a gateway handles the injected timeouts or exceptions.

07. Hardware / Software Requirements

- Any modern web browser with JavaScript and HTML5 canvas
- No server, no libraries — single HTML file, zero dependencies
- Optional: real serial/TCP hardware only for the documented extension path (the browser simulator opens no ports)
- Git

08. Expected Outcomes

- A simulator speaking correct Modbus RTU/TCP framing with CRC-16 validation.
- A live, writable register map and byte-level frame inspector for protocol learning.
- Reproducible failure scenarios (timeouts, exceptions, latency, alarm states) for gateway test plans.
- Protocol documentation (framing, function codes, CRC, data model) and a scenario/fault-injection testing guide.
- A full viva kit: report on Modbus theory and simulator design, PPT and Q&A; on Modbus vs MQTT, RTU vs TCP, CRC and registers/coils.

09. Applications

- Developing and testing IoT gateways, SCADA clients and dashboards without hardware.
- Teaching industrial protocols, binary framing and test engineering.
- Failure-mode testing that real hardware cannot provide on demand.

10. Future Scope

- Real serial-port and TCP-socket slave backends for hardware-in-the-loop testing.
- Modbus diagnostics (function code 08) and FIFO (24) support.
- Multi-slave topologies with address ranges 1–247.

- Recorded session export for regression test suites.