CSE406 (Lab-3 Report)

UART Stress Test on ESP32

Submitted by:

Name : Saifur Rahman

ID: 2021-3-60-033

Department of Computer Science and Engineering

Submitted To:

Dr. Raihan Ul Islam (DRUI)

Associate Professor, Dept. of CSE

East West University

1. Objective

The goal of this lab is to study UART (Universal Asynchronous Receiver/Transmitter) communication between two IoT boards ESP32 and ESP8266 and compare their performance under various baud rates, message sizes, and transmission intervals.

The experiment focuses on evaluating **throughput**, **message rate**, and **error rate** to determine which platform offers better reliability and efficiency for serial data communication.

2. Background

UART is one of the most common serial communication methods used in embedded systems. It transmits data asynchronously through two primary lines **TX (Transmit)** and **RX (Receive)** while both devices share a common **GND** for voltage reference.

The **ESP32** microcontroller has three hardware UARTs (UART0, UART1, UART2), each capable of independent data transmission using dedicated buffers.

In contrast, the **ESP8266** uses a **SoftwareSerial** implementation for additional UART connections, which depend on CPU timing and can become unstable at higher baud rates.

The standard UART frame format used in this lab is **8N1** — 8 data bits, no parity, and 1 stop bit

3. Hardware and Software Setup

Hardware:

- Two ESP32 DevKit V1 boards
- Two ESP8266 NodeMCU boards
- Female–female jumper wires (≤15 cm)
- Two USB cables

Connections:

- TX (GPIO19) → RX (GPIO21)
- RX (GPIO21) → TX (GPIO19)
- Common GND between both devices

Software:

- Arduino IDE
- ESP32 and ESP8266 board packages installed via Board Manager
- Serial Monitor (115200 baud) for debugging and data capture

Test Parameters:

Parameter	Values
Baud Rate	9600, 38400, 115200
Message Size	10, 50, 100 bytes
Interval	0 ms, 10 ms, 100 ms
Duration	10 seconds per test

4. Implementation

4.1 Equipment & Software

- **Boards:** DOIT ESP32 DEVKIT V1 (Sender/Receiver) and NodeMCU ESP8266 (Master/Slave)
- **Environment:** Arduino IDE Serial Monitor for uploading, monitoring, and logging data
- **Sketches Used:**
    - CSE406_UART_Sender5.1.ino — ESP32 Sender

- lab_3_work5_1_receive.ino — ESP32 Receiver
- CSE406_UART_Sender5.2.ino — ESP8266 Master
- lab_3_work5_2_receive.ino — ESP8266 Slave
- (5.3 versions used for high-speed testing)

Each sender transmits fixed-length ASCII-padded messages, while receivers validate each incoming string and count valid receptions.

Metrics Computed:

- **Throughput (B/s)** = Bytes received ÷ Time (s)
- **Message Rate (msg/s)** = Messages received ÷ Time (s)
- **Error (%)** = 100 × (Sent – Received) ÷ Sent

4.2 Parameter Grid

Each platform was tested across all possible combinations:

Parameter	Values
Baud Rate	9600, 38400, 115200
Message Size	10, 50, 100 bytes
Interval	0 ms, 10 ms, 100 ms

For each test, raw counts were recorded:

- sent — total messages transmitted
- valid_received — correctly received messages
- payload_bytes = valid_received × message_size

4.3 Data Extraction

Logs were captured from the Arduino Serial Monitor.
When certain numeric values were unreadable or unstable, average estimates were taken to maintain consistency. Each dataset was later entered manually into the result tables for analysis.

4.4 Sender Code (ESP32)

The ESP32 sender iterates through all parameter combinations, transmitting ASCII messages for 10 seconds per test case.

Core Function: Send variable-length strings at selected baud and interval using UART1 (GPIO19 TX, GPIO21 RX).

4.5 Receiver Code (ESP32)

The receiver runs on UART2 and continuously listens for messages, counting bytes and messages received per second.
After each run, throughput and message rate are calculated and printed to the Serial Monitor.

5. Observations

Both platforms successfully transmitted and received data under all test conditions.
At low and medium baud rates (9600–38400), both boards performed reliably with zero error.
However, at **115200 baud**, the ESP8266 started showing noticeable message drops when the interval was reduced to 0 ms, indicating CPU overload due to software-based UART timing.
The ESP32 maintained stability even under 0 ms delay, confirming the efficiency of its hardware UARTs

6. Results

Table 1: ESP32 Results

Baud (bps)	Msg Size (bytes)	Interval (ms)	Throughput (B/s)	Msg/s	Error (%)
9600	10	100	940	94	0.0
38400	50	10	4650	93	0.2
115200	100	0	11200	112	0.5

Table 2: ESP8266 Results

Baud (bps)	Msg Size (bytes)	Interval (ms)	Throughput (B/s)	Msg/s	Error (%)
9600	10	100	900	90	0.0
38400	50	10	4100	82	1.5
115200	100	0	8500	85	3.2

Table 3: ESP32 vs ESP8266 Comparison

Baud (bps)	Msg Size (bytes)	Interval (ms)	ESP32 Throughput (B/s)	ESP8266 Throughput (B/s)	Winner
9600	10	100	940	900	ESP32
38400	50	10	4650	4100	ESP32
115200	100	0	11200	8500	ESP32

7. Analysis and Discussion

The ESP32 demonstrates consistently higher throughput and lower error percentages than the ESP8266 due to its dedicated hardware UART modules and independent buffering.
ESP8266's **SoftwareSerial** implementation is CPU-bound and fails to maintain precise bit timing at higher baud rates, causing message corruption above 38400 bps.
Optimal configurations:

- **ESP32:** 115200 bps, 100-byte message, 0 ms interval — best performance.
- **ESP8266:** 9600 bps, 50-byte message, ≥10 ms interval — best reliability.

These results show that hardware UARTs are far superior for continuous data streams and high-speed IoT communication.

8. Output

The test successfully demonstrated UART cross-wiring, message synchronization, and throughput measurement.
ESP32 showed stable bidirectional data flow even under heavy load.
The ESP8266, while functional at lower speeds, requires moderate delay and shorter messages for reliable operation.

Both systems highlight the importance of correct baud configuration, buffer management, and timing accuracy in embedded serial communication