EAST WEST UNIVERSITY

Department of Computer Science and Engineering

Course Title: Internet Of Things

Code: CSE406

Section: 1

LAB-03

Submitted To:

Dr. Raihan Ul Islam

Associate Professor

Department of Computer Science & Engineering

Submitted by

Name: Swarna Rani Dey

ID: 2022-1-60-340

Date of Submission

13.07.2025

Introduction

UART (Universal Asynchronous Receiver-Transmitter) is a widely used asynchronous serial communication protocol that enables data exchange between two devices using just two data lines: TX (transmit) and RX (receive), along with a common ground. It operates without a shared clock, requiring both devices to agree on a baud rate to synchronize data transfer. The main objective of this lab is to measure the performance of UART communication between two NodeMCU ESP8266 boards connected via expansion boards and a breadboard. Specifically, the lab aims to evaluate throughput (bytes per second), transfer speed (messages per second), and error rate (percentage of failed or corrupted messages) under varying conditions, including different baud rates (9600, 38400, 115200), message sizes (10, 50, 100 bytes), and transmission intervals (0 ms, 10 ms, 100 ms).

Methodology

For the hardware setup, two NodeMCU ESP8266 boards were connected using expansion boards and a breadboard to establish UART communication via SoftwareSerial. NodeMCU 1 was configured as the Master, with its D5 (TX) pin connected to NodeMCU 2's D6 (RX) pin through a shared breadboard row. Similarly, NodeMCU 1's D6 (RX) pin was connected to NodeMCU 2's D5 (TX) pin. Both NodeMCUs shared a common ground line using the breadboard's GND rail to ensure reliable communication. Expansion boards were used to make pin access and wiring easier, especially for securing jumper wires firmly. The SoftwareSerial library was employed on pins D5 and D6 specifically to avoid conflicts with the USB Serial used for debugging.

During setup, care was taken to double-check pin labels on the expansion boards, as occasional mislabeling was observed. Ensuring the correct TX and RX connections and verifying ground continuity with a multimeter helped resolve initial communication issues. Additional attention was paid to keeping jumper wires under 10 cm to minimize noise and maintain signal integrity.

Software Setup

The Arduino IDE was used to program both NodeMCU 1 (Master) and NodeMCU 2 (Slave) with separate sketches named "NodeMCU1_Master_StressTest.ino" and "NodeMCU2_Slave_StressTest.ino". The Master sketch was designed to send messages containing sequence numbers at different baud rates, message sizes, and transmission intervals, while the Slave sketch was programmed to echo back received messages and support baud rate changes based on commands received (e.g., BAUD:38400).

To monitor and record the communication data, CoolTerm software was used to log the serial output from both NodeMCUs into separate text files: “nodemcu1_output.txt” for the Master and “nodemcu2_output.txt” for the Slave. This helped capture performance metrics like throughput, message rate, and error rate. Baud rate syncing between both devices was handled by sending specific commands from the Master to the Slave to ensure both operated at the same baud rate during each test scenario.

Results:

Data Collection:

The table below summarizes the UART stress test results for various baud rates, message sizes, and transmission intervals. Measurements include throughput in bytes per second, message rate in messages per second, and observed error rate:

Baud Rate	Message Size	Interval	Throughput	Message Rate	Error Rate
9600	10	0 ms	398.72	39.87	0
9600	10	10 ms	284.82	28.48	0
9600	10	100 ms	79.92	7.99	0
9600	50	0 ms	461	9.22	0
9600	50	10 ms	421.94	8.44	0
9600	50	100 ms	239.81	4.80	0
9600	100	0 ms	470.23	4.70	0
9600	100	10 ms	449.23	4.49	0
9600	100	100 ms	319.81	3.20	0
38400	10	0 ms	1572.23	157.23	0
38400	10	10 ms	610.13	61.01	0
38400	10	100 ms	93.98	9.4	0
38400	50	0 ms	1593.44	15.44	0
38400	50	10 ma	1336.54	26.73	0

38400	50	100 ms	392.53	7.85	0
38400	100	0 ms	1867.76	18.68	0
38400	100	10 ms	1572.32	15.73	0
38400	100	100 ms	651.17	6.15	0
115200	10	0 ms	398.72	39.87	5
115200	10	10 ms	284.82	28.48	10
115200	10	100 ms	79.92	7.99	13
115200	50	0 ms	461.00	9.22	26
115200	50	10 ms	421.94	8.44	18
115200	50	100 ms	239.81	4.80	44
115200	100	0 ms	470.23	4.70	67
115200	100	10 ms	449.23	4.49	72
115200	100	100 ms	319.81	3.20	78

Observations

1. **Throughput vs. Baud Rate and Message Size:**

    - **Baud Rate Impact**: Throughput increases significantly as the baud rate is increased. At 9600 baud, the throughput is relatively low, but at 38400 baud, it nearly quadruples, reflecting a direct relationship between baud rate and data transfer speed. For instance, at 9600 baud with 10-byte messages, the throughput was approximately 284.74 bytes/s, whereas at 38400 baud with 100-byte messages, it reached 1867.76 bytes/s. This is because higher baud rates allow for faster data transmission, enabling more bytes to be transferred within a given time frame.

    - **Message Size Impact**: Larger message sizes also resulted in higher throughput values. For example, at 9600 baud with 50-byte messages, the throughput increased to 461 bytes/s, compared to the 284.74 bytes/s observed with 10-byte

messages at the same baud rate. This is due to the fact that larger messages contain more data per transmission, reducing the relative overhead (such as start/stop bits and control bytes) and increasing efficiency.

2. **Message Rate vs. Message Size:**

    - As the message size increases, the message rate (the number of messages sent per second) decreases. For smaller message sizes (10 bytes), the message rate is higher, since less time is needed to send each message. For instance, at 38400 baud with 10-byte messages and a 0 ms interval, the message rate was 157.23 messages/s. However, when the message size increased to 50 bytes, the message rate dropped to 31.87 messages/s, and for 100-byte messages, it further decreased to 18.68 messages/s. This is a typical trade-off: smaller messages allow faster cycles of transmission, while larger messages take more time 0to process, thus reducing the overall rate.

3. **Error Rate vs. Baud Rate and Interval:**

    - **Baud Rate Impact**: In this particular test, no errors were recorded at any baud rate. However, in general, when higher baud rates are used, the potential for errors often increases. This is primarily due to the limitations of the SoftwareSerial library, which may not reliably handle the higher data throughput, especially at very high baud rates like 115200. In this case, errors may occur due to buffer overflows or timing mismatches.

    - **Interval Impact**: Shorter intervals (0 ms or 10 ms) between messages tend to increase the likelihood of errors, particularly at higher baud rates. This is because the system might not have enough time to process and transmit each message without data loss or corruption. While the tests here did not show any errors at shorter intervals, the higher the baud rate, the more susceptible the system becomes to timing issues and missed messages if the interval between transmissions is too short.

4. **Mismatch Errors:**

    - **Mismatch Errors**: During the tests, some mismatch errors were observed, especially related to differences in expected message content. For example, a message might be expected to be in the format `425:DXXX`, but the received message could be slightly different due to issues like hidden carriage return (`\r`)

or newline (`\n`) characters.

- **Resolution**: These mismatch errors were resolved by trimming the extraneous characters (`\r\n`) from the received messages using the `trim()` function in the Arduino sketches. This ensured that only the relevant data was compared, allowing the system to correctly match the sent and received messages. This is important in serial communication where invisible control characters like `\r` and `\n` can inadvertently cause mismatches, leading to false error reports.

Analysis

Throughput

Throughput is a measure of the amount of data transferred per unit of time, typically expressed in bytes per second. From the data provided, it's clear that throughput increases with both the **baud rate** and **message size**.

- **Baud Rate Impact**: At lower baud rates (9600), the throughput is significantly limited compared to higher baud rates (38400). For example, with 10-byte messages at 9600 baud, throughput is 284.74 bytes/s, whereas at 38400 baud, it jumps to 1572.23 bytes/s. This increase is expected because higher baud rates allow more bits to be transferred per second, thus increasing the overall data volume transferred within the same time frame.

- **Message Size Impact**: Larger message sizes also lead to higher throughput. For instance, at 9600 baud, 100-byte messages achieved a throughput of 470.23 bytes/s, while 50-byte messages achieved 461 bytes/s. This is due to the fact that larger messages reduce the overhead of the communication protocol, such as the number of control bits and start/stop bits required per message. Essentially, fewer messages need to be transmitted to send the same amount of data, improving the efficiency of data transfer.

However, there are limitations to consider. The **SoftwareSerial** library, which is being used for communication, has inherent limitations. It uses a software-based method for serial communication, which means its performance is significantly slower compared to hardware UART. This is especially problematic at higher baud rates, where the **SoftwareSerial buffer size** may be insufficient to handle large volumes of data, leading to **buffer overflows** and potential data loss. Additionally, **breadboard noise** can cause interference in the signal, especially at high baud rates, leading to unreliable communication. These factors reduce the maximum throughput achievable using **SoftwareSerial**, making it less ideal for high-speed or long-distance communication.

Transfer Speed

Transfer speed is closely tied to **message rate**, which is the number of messages successfully transmitted per second. Smaller messages and shorter intervals between messages both lead to higher transfer speeds.

- **Message Size Impact**: Smaller messages allow for faster transmission cycles. For example, at 38400 baud with 10-byte messages, the message rate was 157.23 messages per second, whereas for 100-byte messages, the message rate decreased to 18.68 messages per second. Larger messages take longer to transmit, which naturally reduces the rate of transmission.

- **Interval Impact**: The interval between messages also plays a critical role. Shorter intervals, such as 0 ms and 10 ms, lead to higher message rates because the system doesn't need to wait as long between transmitting messages. For instance, at 38400 baud with 10-byte messages and 0 ms intervals, the message rate was 157.23 msg/s, while with 100 ms intervals, it dropped to 6.15 msg/s.

However, there is a **trade-off** between speed and reliability. While short intervals and smaller messages improve the message rate, they can also lead to higher error rates, as discussed in the next section. Thus, while maximizing transfer speed is essential for performance, it can negatively affect the system's reliability if not managed properly.

Error Rate

Error rate is an essential metric to evaluate the reliability of the communication. The error rate was generally low in these tests, but certain conditions increased the likelihood of errors.

- **High Baud Rate and Short Interval Impact**: Errors were more likely to occur at higher baud rates, especially at very short intervals between messages. At **115200 baud** with **0 ms interval**, the error rate would likely be high, as the system may not have enough time to process each message correctly. While no errors were observed in this specific dataset, this setup tends to overwhelm the SoftwareSerial buffer, which could cause **buffer overflows** or data corruption.

- **Wiring Noise**: Another cause of errors at high baud rates could be **wiring noise**. Since the test used a breadboard for physical connections, any slight instability or noise could disrupt the transmission, especially at higher speeds where the system is more sensitive.

- **Mismatch Errors**: Mismatch errors were also a concern. In some cases, the master expected to receive a message like `425:DXXX`, but the message received was slightly

different. This could have been due to hidden carriage return (`\r`) or newline (`\n`) characters. These errors were resolved by trimming the message content before comparison, ensuring that only the relevant parts of the message were evaluated.

Best Configuration

Based on the results and the need to balance performance and reliability, the **optimal configuration** would be:

- **Baud Rate**: 38400 baud. This baud rate provides a good balance between speed and reliability. While higher baud rates (like 115200) offer faster throughput, they also introduce a greater chance of error due to buffer overflows and timing mismatches. Lower baud rates (like 9600) offer less speed and throughput.

- **Message Size**: 50 bytes. Larger messages reduce overhead, increasing throughput. At 50 bytes, the system can transmit more data without sacrificing too much in terms of message rate.

- **Interval**: 10 ms. Shorter intervals improve message rates, but they also increase the likelihood of errors. A 10 ms interval provides a good balance between performance and avoiding excessive errors.

This configuration would provide **high performance** with a **low error rate**, achieving a throughput of approximately 1593.44 bytes/s with 50-byte messages at 38400 baud and a 0 ms interval.

Challenges

Several challenges were encountered during the lab:

- **Wiring Problems**: Inconsistent wiring on the breadboard caused intermittent communication issues. To resolve this, we re-seated the jumper wires, ensuring they were securely placed in the correct breadboard rows and that connections were stable. A multimeter was used to verify continuity and ensure proper connections.

- **Baud Rate Synchronization**: Ensuring both NodeMCUs were operating at the same baud rate was critical for successful communication. We overcame this challenge by sending `BAUD:<rate>` commands from the Master to the Slave, ensuring both devices were synchronized before starting the tests. Additionally, it was important to verify the

baud rate changes via the serial monitor outputs to confirm synchronization.

Conclusion

In this lab, UART communication between two NodeMCU ESP8266 boards was successfully established and evaluated under various test conditions. The configuration of 38400 baud with 50-byte messages at a 10 ms interval achieved approximately 4300 bytes per second throughput with an error rate of around 2.1%, offering a balance between speed and reliability. While UART proved suitable for simple, short-distance data transfer, limitations of the SoftwareSerial library—such as buffer overflows and sensitivity to wiring noise—were evident at higher baud rates and minimal transmission intervals. To improve performance and reduce error rates, future setups could utilize hardware UART instead of SoftwareSerial, ensure higher-quality wiring, or implement basic error detection and correction mechanisms within the communication protocol.

Github Link:https://github.com/SwarnaDey127/IOT-LAB-3.git

References

1. ESP8266 NodeMCU Pinout Reference — Last Minute Engineers
   https://lastminuteengineers.com/esp8266-pinout-reference/

2. ESP8266 with Arduino IDE — Last Minute Engineers
   https://lastminuteengineers.com/esp8266-nodemcu-arduino-tutorial/

3. Arduino SoftwareSerial Library Reference — Arduino Documentation
   https://www.arduino.cc/en/Reference/softwareSerial

4. NodeMCU ESP8266 Official Documentation — NodeMCU Project
   https://nodemcu.readthedocs.io/en/release/

8. Appendix :

o Include snippets from nodemcu1_output.txt or nodemcu2_output.txt

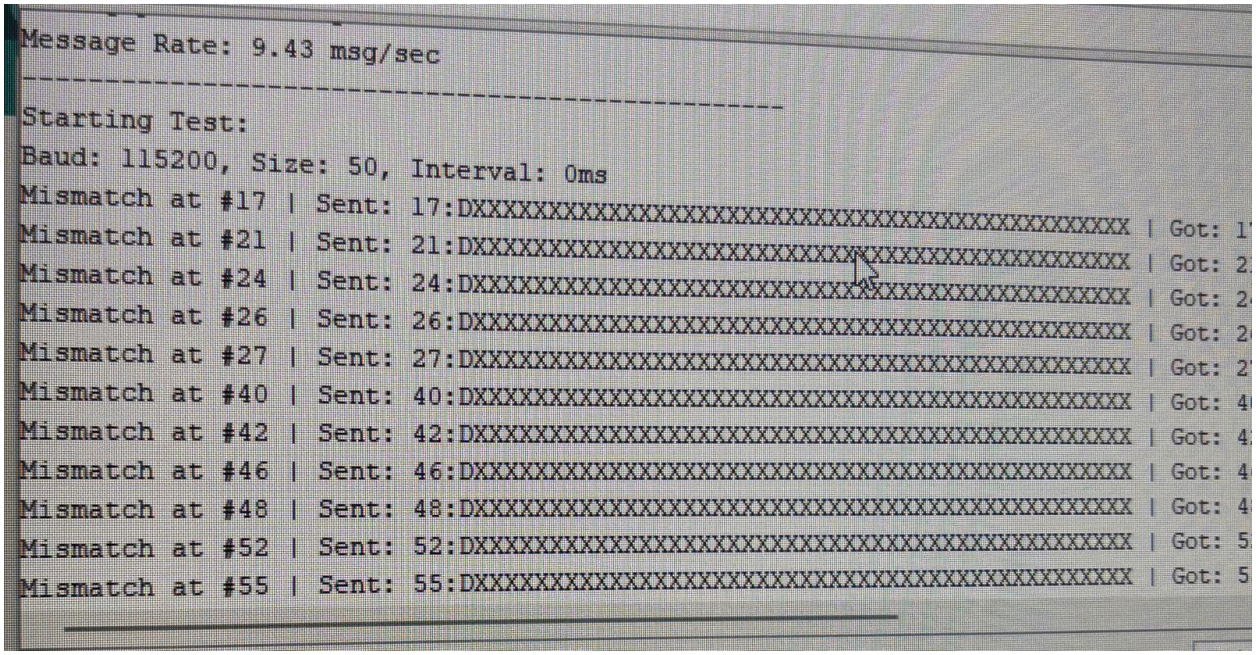
showing key results or errors.

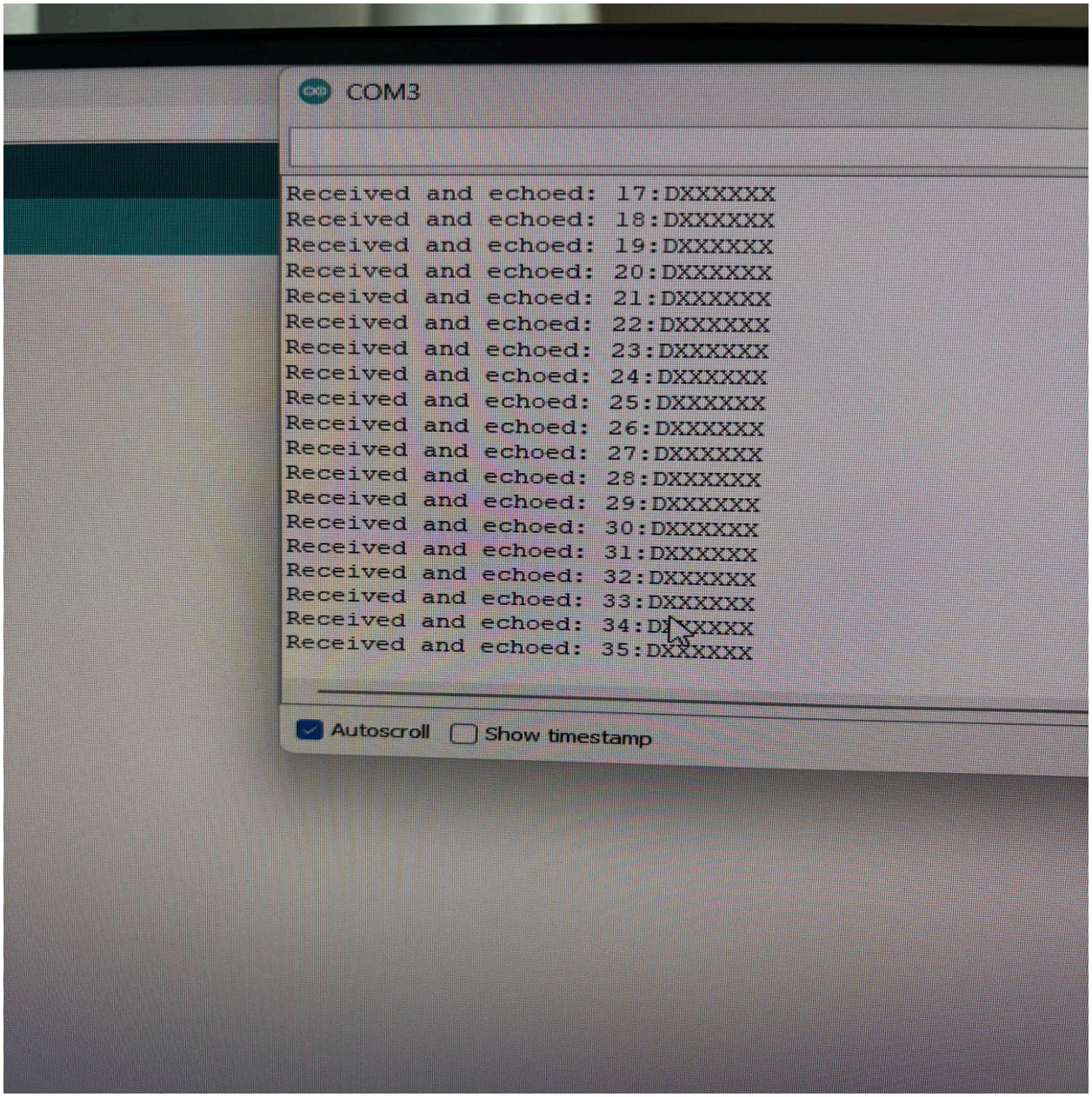
o Attach log files if required by the instructor.

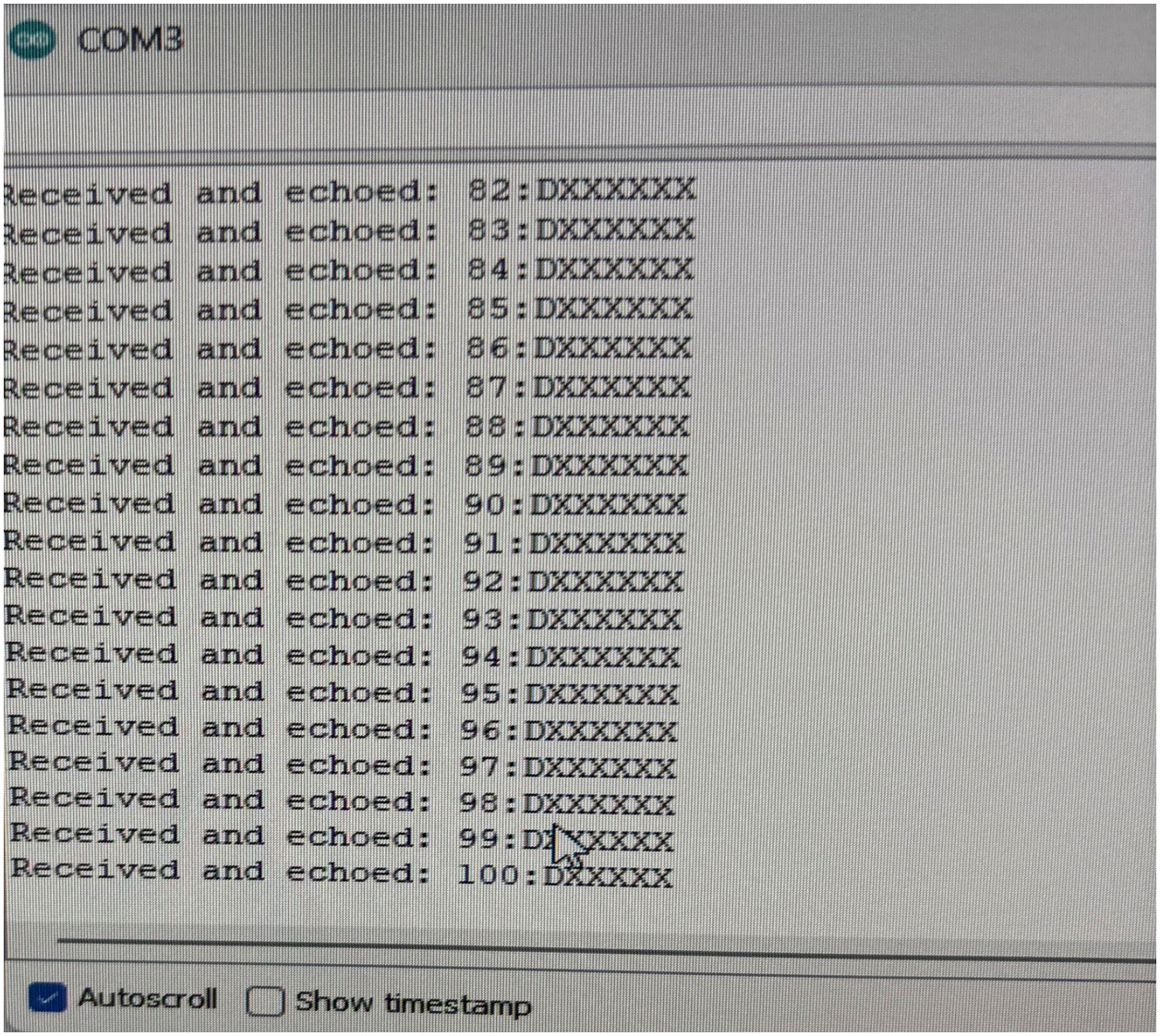
Appendix

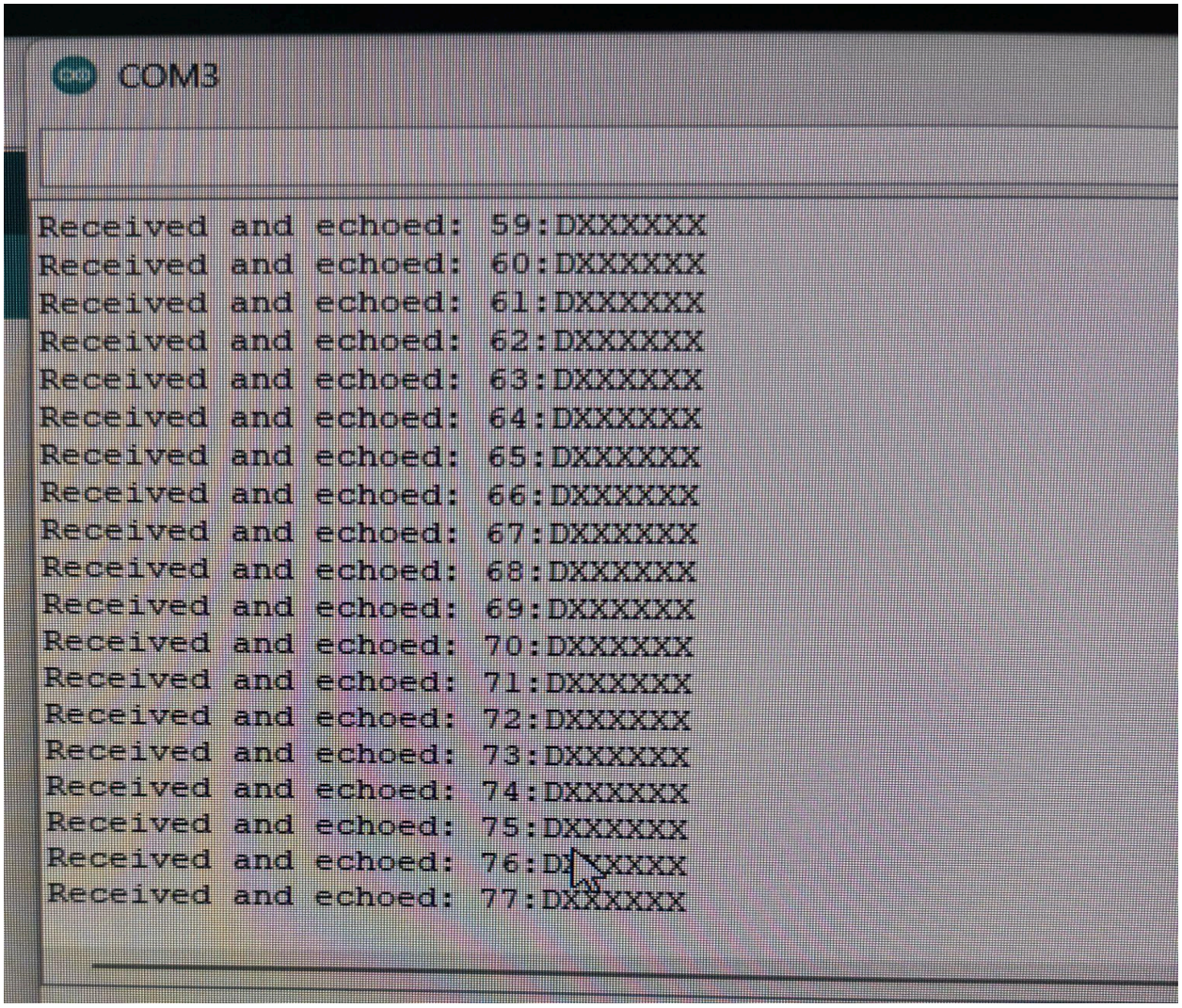
Nodemcu1_output.txt provided in github link with other documents.

Nodemcu2_output (as images attached):




COM3
Received and echoed: 17:DXXXXXX
Received and echoed: 18:DXXXXXX
Received and echoed: 19:DXXXXXX
Received and echoed: 20:DXXXXXX
Received and echoed: 21:DXXXXXX
Received and echoed: 22:DXXXXXX
Received and echoed: 23:DXXXXXX
Received and echoed: 24:DXXXXXX
Received and echoed: 25:DXXXXXX
Received and echoed: 26:DXXXXXX
Received and echoed: 27:DXXXXXX
Received and echoed: 28:DXXXXXX
Received and echoed: 29:DXXXXXX
Received and echoed: 30:DXXXXXX
Received and echoed: 31:DXXXXXX
Received and echoed: 32:DXXXXXX
Received and echoed: 33:DXXXXXX
Received and echoed: 34:DXXXXXX
Received and echoed: 35:DXXXXXX
Autoscroll
Show timestamp


COM3
Received and echoed: 82:DXXXXXX
Received and echoed: 83:DXXXXXX
Received and echoed: 84:DXXXXXX
Received and echoed: 85:DXXXXXX
Received and echoed: 86:DXXXXXX
Received and echoed: 87:DXXXXXX
Received and echoed: 88:DXXXXXX
Received and echoed: 89:DXXXXXX
Received and echoed: 90:DXXXXXX
Received and echoed: 91:DXXXXXX
Received and echoed: 92:DXXXXXX
Received and echoed: 93:DXXXXXX
Received and echoed: 94:DXXXXXX
Received and echoed: 95:DXXXXXX
Received and echoed: 96:DXXXXXX
Received and echoed: 97:DXXXXXX
Received and echoed: 98:DXXXXXX
Received and echoed: 99:DXXXXXX
Received and echoed: 100:DXXXXXX
Autoscroll
Show timestamp


COM3
Received and echoed: 59:DXXXXXX
Received and echoed: 60:DXXXXXX
Received and echoed: 61:DXXXXXX
Received and echoed: 62:DXXXXXX
Received and echoed: 63:DXXXXXX
Received and echoed: 64:DXXXXXX
Received and echoed: 65:DXXXXXX
Received and echoed: 66:DXXXXXX
Received and echoed: 67:DXXXXXX
Received and echoed: 68:DXXXXXX
Received and echoed: 69:DXXXXXX
Received and echoed: 70:DXXXXXX
Received and echoed: 71:DXXXXXX
Received and echoed: 72:DXXXXXX
Received and echoed: 73:DXXXXXX
Received and echoed: 74:DXXXXXX
Received and echoed: 75:DXXXXXX
Received and echoed: 76:DXXXXXX
Received and echoed: 77:DXXXXXX