Introduction

In this lab, we were able to experiment with the UART (Universal Asynchronous Receiver-Transmitter) community in this lab with two nodes MCU ESP8266. UART The UART protocol is a serial type of protocol where data can be sent and received through two signal wires: TX and RX. The idea was to initiate the communication between the boards using SoftwareSerial on pins D5 and D6 and through a stress test by measuring throughput bytes/sec, transfer speed, messages/sec, and the error rate in %. At 9600, 38400 and 115200 baud, a fixed-sized message of 10, 50 and 100 bytes was used with the intervals at 0, 10 and 100 ms.

Methodology

Hardware Setup

Two NodeMCU boards were mounted on expansion bases and connected via jumper wires through a breadboard:

TX/RX Wiring:

1. NodeMCU 1 (Master) D5 (TX) → NodeMCU 2 D6 (RX)
2. NodeMCU 1 (Master) D6 (RX) ← NodeMCU 2 D5 (TX)
3. Common GND connected via the breadboard ground rail.

Software Setup

We used the Arduino IDE with the ESP8266 board support installed. Two sketches were developed:

Node1_Master_StressTest.ino (Master): Sends test messages and calculates metrics.
NodeMCU2_Slave_StressTest.ino (Slave): Echoes back received messages and logs outputs.
Baud rates were synchronized by sending commands such as BAUD:38400. Serial outputs were logged using CoolTerm and saved for analysis.

Explanation of Code

Node1 UART:

The code is Node1 -UART.ino, and the current temperature reads as red lines on the UART port as a simple UART master, on NodeMCU 1. It communicates through software serial on a pair of pins D5 and D6, where D5 will transmit, and D6 receive. During the setup process, the master both initialization of the hardware serial port (used to debug) and the software serial port (used to communicate via UART) and prints a message saying that the master is live. Within the loop, it broadcasts the message of Hello form nodeMCU 1, at every two seconds interval and waits to receive any response of the slave node. When

UART line gets a corresponding response, it reads the message and displays the message on the Serial Monitor.

Node2 UART:

The code is the Node2-UART.ino code which is the slave program on the NodeMCU 2. It is the same pin arrangement and uses SoftwareSerial as well. To the set up section, once the serial communication has been initialized, the slave loop continuously looks on inbound UART message. When it receives a message sent by NodeMCU 1, it reads the data, and prints the message received on the Serial Monitor so that one can verify that the message is received correctly by the receiving node, and sends a predefined message, "Hi fro NodeMCU 2" back. This configuration provides a simplex two-way conversation with a source (master) transmitting a message and the source receiving message (slave).

NodeMCU1_Master_StressTest:

Node1_Master_StressTest.ino script is an improved form of the master program. It carries out stress test messages of different sizes (10, 50, 100 bytes) at various baud rates (9600, 38400, 115200) and with different time delays (0ms and 10ms and 100ms the time between the sendings) between each other. This program is aimed at analyzing the level of the UART communication efficiency in various conditions. In every configuration, the master emits a sequence-tagged message and awaits an echo response by slave and monitors whether the message was received successfully, a mismatch was received or a timeout. The test is operated under ten (10) seconds constant period per configuration. It measures and prints data (after each test) on throughput (in bytes per second), message rate (in messages per second) and errors rate expressed as a percentage. Communication efficiency and reliability can be measured accurately by use of a number sequence and precise timing.

NodeMCU2_Slave_StressTest:

NodeMCU2_Slave_StressTest.ino script is supposed to be able to interact with stress test master script. It will listen to the hardware Serial and receive such commands such as "BAUD:38400", after which it will switch its software serial baud rate too, to allow dynamic testing at various rates. When in the main program it is scanning the SoftwareSerial input to see if it has new messages in it, scrolls them, stripping off unnecessary whitespace or characters, and then re-outputs them back to the master just like received. This enables the master to check the integrity of messages. Also, all the messages received and echoed are displayed in the Serial Monitor so as to help in debugging and confirmation. The aggregate working of these four programs forms up a very solid configuration towards experimenting and testing UART communication between two NodeMCUs.

These sketches handle message transmission, echoing, error checking, and metric calculation. The master also sends a unique ID with each message to track responses.

Output

Observations

- The higher the baud rate and size of messages, the more the throughput.
- At higher baud rates the rate of errors rises.
- Due to the mismatch of echo and buffer overflow when using SoftwareSerial, 115200 and 0ms happened frequently.
- The command string was effective and efficient in synchronising baud rate provided it is formatted correctly.

Analysis

Throughput

The baud rates were not critical (e.g. 115200/sec) could transfer more data to a given end per second, however, at some point, the SoftwareSerial buffer became a limiting factor.

Transfer Speed

By increasing the frequency of messages the message rate became larger at higher baud but the introduction of errors occurred. 10ms had the sweet spot between spiciness and rigidity.

Error Rate

Errors were caused by:

- SoftwareSerial limitations
- Noise due to longer wires
- Mismatch in expected vs. received message formats

We resolved mismatch errors by trimming extra characters and validating packet IDs.

Discussion

The lab illustrated the operation of UART and how to achieve credible communication with NodeMCUs. We came across the fact that although high-baud rates facilitate quick data transmission, increasing error

rates as well are inevitable particularly when small spaces and large messages are combined. The best performance came out with a mid-range setting. Hardware UART or buffer-handling should be used in further versions, in order to decrease the possibility of errors.