Lab Task – Communication between two NodeMCU (ESP8266) using UART interface

Lab Description: Exploring UART Communication with NodeMCU ESP8266

Objective

In this lab, students will:

1. Understand **UART (Universal Asynchronous Receiver-Transmitter)** communication and its role in serial data transfer.
2. Connect two NodeMCU ESP8266 boards using NodeMCU expansion boards and a breadboard to establish UART communication.
3. Perform a stress test to measure:
    - **Amount of data** transferred (throughput in bytes per second).
    - **Transfer speed** (messages per second).
    - **Error rate** (percentage of failed or corrupted messages).
4. Analyze the impact of baud rate, message size, and transmission interval on performance.

Background: What is UART Communication?

UART (Universal Asynchronous Receiver-Transmitter) is a serial communication protocol used to transfer data between two devices, such as microcontrollers, sensors, or computers. It is widely used due to its simplicity and reliability for short-distance, low-speed communication.

Key Features of UART

- **Asynchronous**: No shared clock signal; both devices must agree on a **baud rate** (bits per second) to synchronize data transfer.
- **Two-Wire Interface**:
    - **TX (Transmit)**: Sends data from one device.
    - **RX (Receive)**: Receives data on the other device.

    - A common **GND (ground)** connection is required.

- **Data Format**: Data is sent in packets (typically 8-bit bytes) with a start bit, optional parity bit, and stop bit(s).
- **Baud Rate**: Common rates include 9600, 38400, and 115200 bits per second. Higher rates increase speed but may introduce errors if wiring or hardware is unreliable.
- **Full-Duplex**: Both devices can send and receive simultaneously (if wired TX→RX and RX→TX).

UART in NodeMCU ESP8266

- The NodeMCU ESP8266 has a hardware UART (pins D1/TX, D2/RX) but shares it with USB Serial for debugging.
- **SoftwareSerial** is used to create a virtual UART on other pins (e.g., D5/TX, D6/RX), allowing simultaneous communication and debugging.
- SoftwareSerial is slower and less reliable at high baud rates but suitable for educational purposes.

Lab Setup

Materials

- 2x NodeMCU ESP8266 boards (e.g., NodeMCU 1.0 ESP-12E)
- 2x NodeMCU expansion boards (base boards with pin headers or terminal blocks)
- 1x Breadboard
- Male-to-male jumper wires
- 2x USB cables (for power and programming)
- Computer with Arduino IDE installed
- Optional: Multimeter for wiring verification

Hardware Setup

Students will connect two NodeMCUs via their expansion boards using a breadboard to establish UART communication.

1. **Prepare NodeMCUs and Expansion Boards**:

- Plug each NodeMCU into its expansion board, ensuring pins align (e.g., D5 to D5, GND to GND).
- The expansion boards provide easy access to pins via headers or terminal blocks.

2. **Wiring on Breadboard**:
   - Use SoftwareSerial on **D5 (GPIO14, TX)** and **D6 (GPIO12, RX)** to avoid conflicts with USB Serial.
   - Connect the NodeMCUs as follows:
     - **NodeMCU 1 (Master)**:
       - D5 (TX) → Breadboard row 10A → NodeMCU 2 D6 (RX, row 10F).
       - D6 (RX) → Breadboard row 11A ← NodeMCU 2 D5 (TX, row 11F).
       - GND → Breadboard GND rail.
     - **NodeMCU 2 (Slave)**:
       - D5 (TX) → Breadboard row 11F → NodeMCU 1 D6 (RX, row 11A).
       - D6 (RX) → Breadboard row 10F ← NodeMCU 1 D5 (TX, row 10A).
       - GND → Breadboard GND rail (same as NodeMCU 1).
   - **Diagram** (text-based):

```
NodeMCU 1 (Expansion Board 1)        Breadboard        NodeMCU 2 (Expansion Board 2)
D5 (TX) ----------------> Row 10A ================= Row 10F ----------------> D6 (RX)
D6 (RX) <---------------- Row 11A ================= Row 11F <---------------- D5 (TX)
GND --------------------> GND Rail <==============> GND Rail <---------------- GND
```

3. **Power**:
   - Connect each NodeMCU to a USB cable, plugged into a computer or powered USB hub.
   - The expansion boards pass power (5V, regulated to 3.3V by NodeMCU) to the boards.

4. **Wiring Tips**:
   - Ensure jumpers are secure in breadboard rows and expansion board headers.

- Keep wires short (<10 cm) to minimize noise.
- Verify GND continuity with a multimeter to avoid communication issues.
- Double-check D5/D6 pin labels on expansion boards, as mislabeling can occur.

Submission Guidelines for UART Communication Lab

Overview

In this lab, you explored **UART (Universal Asynchronous Receiver-Transmitter)** communication by connecting two NodeMCU ESP8266 boards using expansion boards and a breadboard. You performed a stress test to measure the performance of UART communication by varying baud rates, message sizes, and transmission intervals. Your task was to quantify:

- **Amount of Data Transferred**: Throughput in bytes per second.
- **Transfer Speed**: Messages per second.
- **Error Rate**: Percentage of failed or corrupted messages.

Your submission must include the **Arduino code** used for the stress test and a **report** analyzing the results. This document outlines the requirements, structure, and expectations for your submission to ensure a thorough evaluation of your findings.

Submission Components

Your submission must consist of two parts:

1. **Arduino Code**: The sketches used for NodeMCU 1 (Master) and NodeMCU 2 (Slave).
2. **Lab Report**: A detailed analysis of the stress test results, including throughput, transfer speed, and error rate.

1. Arduino Code

Submit the Arduino sketches that you used to perform the UART stress test. These sketches should match the functionality described in the lab, where NodeMCU 1 sends messages of varying sizes and intervals, and NodeMCU 2 echoes them back, with metrics logged to the Serial Monitor.

Requirements

- **Files**:
    - Submit two .ino files:
        - NodeMCU1_Master_StressTest.ino: The Master sketch for NodeMCU 1.
        - NodeMCU2_Slave_StressTest.ino: The Slave sketch for NodeMCU 2.
    - Ensure the files are named exactly as specified to avoid confusion.
- **Content**:
    - The Master sketch should:
        - Send messages with sequence numbers at different baud rates (e.g., 9600, 38400, 115200), message sizes (e.g., 10, 50, 100 bytes), and intervals (e.g., 0ms, 10ms, 100ms).
        - Verify responses from NodeMCU 2 and calculate throughput, message rate, and error rate.
        - Output results to the Serial Monitor.
    - The Slave sketch should:
        - Echo received messages back to NodeMCU 1.
        - Support baud rate changes (e.g., via Serial commands like BAUD:9600).
        - Log received and sent messages to the Serial Monitor for debugging.
- **Code Quality**:
    - Include comments explaining key sections (e.g., test loop, error handling).
    - Ensure the code is functional and matches the logged results in your report.
    - Use SoftwareSerial on pins D5 (TX) and D6 (RX) to align with the lab setup.

Submission Format

- Upload the codes it to your github repository in a folder named Lab 3.
- Do not paste code directly into the report unless specifically requested.

2. Lab Report

The lab report is a written document analyzing the results of your UART stress test. It should provide a clear, structured evaluation of the **amount of data transferred** (throughput), **transfer speed** (messages per second), and **error rate** (percentage of failed or corrupted messages).

Report Structure

The report should be submitted as a **PDF** or **Word document** (e.g., John_Doe_UART_Lab_Report.pdf) and follow this structure:

1. **Title Page**:
    - Title: "UART Communication Stress Test with NodeMCU ESP8266"
    - Your name, student ID, course name, and date (July 7, 2025).
    - Lab section or group number (if applicable).
2. **Introduction** (1 paragraph):
    - Briefly explain **UART communication** (e.g., asynchronous serial protocol, TX/RX pins, baud rate).
    - State the lab's objective: to measure throughput, transfer speed, and error rate using two NodeMCUs connected via expansion boards and a breadboard.
    - Mention the test parameters (baud rates: 9600, 38400, 115200; message sizes: 10, 50, 100 bytes; intervals: 0, 10, 100ms).
3. **Methodology** (1–2 paragraphs):
    - **Hardware Setup**:
        - Describe how you connected the NodeMCUs (e.g., NodeMCU 1 D5 → NodeMCU 2 D6, NodeMCU 2 D5 → NodeMCU 1 D6, shared GND via breadboard).
        - Mention the use of expansion boards and breadboard for wiring.
        - Note any challenges (e.g., wiring issues resolved).
    - **Software Setup**:

- Explain that you used Arduino IDE to program NodeMCU 1 (Master) and NodeMCU 2 (Slave).
- Describe how you logged output using CoolTerm to files (e.g., nodemcu1_output.txt, nodemcu2_output.txt).
- Mention baud rate syncing (e.g., sending BAUD:38400 to NodeMCU 2).

4. **Results** (2–3 pages):
    - **Data Collection**:
        - Provide a table summarizing the stress test results for each test case (baud rate, message size, interval). Include:
            - **Throughput** (bytes/second).
            - **Message Rate** (messages/second).
            - **Error Rate** (% of failed or corrupted messages).
        - Example table:

```
| Baud Rate | Message Size | Interval | Throughput (bytes/s) | Message Rate (msg/s) | Error Rate (%) |
|-----------|--------------|----------|----------------------|----------------------|----------------|
| 9600      | 10 bytes     | 0 ms     | 1200                 | 50                   | 0.5            |
| 9600      | 50 bytes     | 10 ms    | 1500                 | 30                   | 1.0            |
| 38400     | 10 bytes     | 0 ms     | 3500                 | 100                  | 2.0            |
| ...       | ...          | ...      | ...                  | ...                  | ...            |
```

        - Attach or reference the log files (nodemcu1_output.txt, nodemcu2_output.txt) as evidence.
    - **Observations**:
        - Discuss trends, e.g.:
            - How throughput increases with baud rate or message size.
            - How message rate decreases with larger messages.

        - How error rate changes with baud rate or interval (e.g., higher errors at 115200 baud).
    - Note any mismatch errors (e.g., Expected '425:DXXX', Got '425:DXXX') and their resolution (e.g., trimming \r\n).

5. **Analysis** (1–2 pages):
    - **Throughput**:
        - Compare throughput across baud rates and message sizes.
        - Explain why larger messages or higher baud rates increase throughput (e.g., more data per transmission).
        - Discuss limitations (e.g., SoftwareSerial's buffer size, breadboard noise).
    - **Transfer Speed**:
        - Analyze message rate trends (e.g., higher rates with smaller messages or shorter intervals).
        - Discuss trade-offs between speed and reliability.
    - **Error Rate**:
        - Identify conditions with high error rates (e.g., 115200 baud, 0ms interval).
        - Explain possible causes (e.g., SoftwareSerial buffer overflow, wiring noise).
        - If mismatch errors occurred, discuss their cause (e.g., hidden \r\n) and how they were mitigated.
    - **Best Configuration**:
        - Recommend the optimal baud rate, message size, and interval for low errors and high performance (e.g., 38400 baud, 50 bytes, 10ms interval).
    - **Challenges**:
        - Describe any issues (e.g., wiring problems, baud rate syncing) and how you resolved them.

6. **Conclusion** (1 paragraph):
    - Summarize key findings (e.g., “38400 baud with 50-byte messages at 10ms intervals achieved 3500 bytes/s with 2% error rate”).
    - Reflect on UART’s suitability for data transfer and limitations of SoftwareSerial.
    - Suggest improvements (e.g., use hardware UART, improve wiring, or add error correction).
7. **References** (if applicable):
    - Cite any resources used (e.g., NodeMCU documentation, Arduino SoftwareSerial library reference).
8. **Appendix** (optional):
    - Include snippets from nodemcu1_output.txt or nodemcu2_output.txt showing key results or errors.
    - Attach log files if required by the instructor.

Report Format

- **Length**: 4–6 pages (excluding title page and appendix).
- **Format**: PDF or Word document, 12-point font, 1-inch margins, single or 1.5 spacing.
- **File Name**: YourName_UART_Lab_Report.pdf (e.g., John_Doe_UART_Lab_Report.pdf).
- **Figures/Tables**:
    - Include the results table (mandatory).
    - Optionally, add graphs (e.g., throughput vs. baud rate) created in a spreadsheet or Python.
- **Submission**: Upload to the course platform (e.g., Canvas, Blackboard) by the deadline.

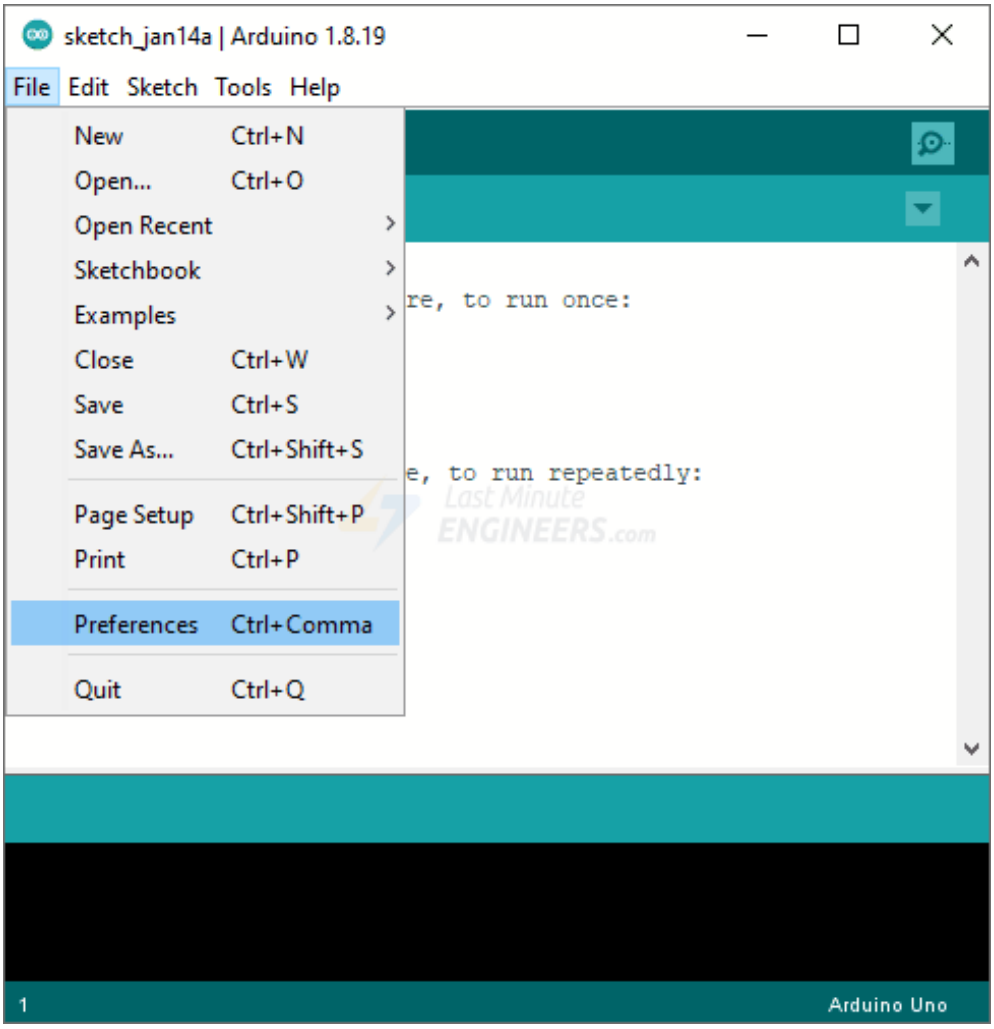
Submission Guidelines

- **Files**:
    - NodeMCU1_Master_StressTest.ino

    - NodeMCU2_Slave_StressTest.ino
    - YourName_UART_Lab_Report.pdf
    - Optional: nodemcu1_output.txt, nodemcu2_output.txt
    - Combine all files in a folder named Lab3 in your git hub repository and submit the link in google classroom.
- **Integrity**:
    - Submit your own code and results. Plagiarism (e.g., copying code or log files) will result in penalties.
    - If you modified the provided sketches, explain changes in the report's methodology.

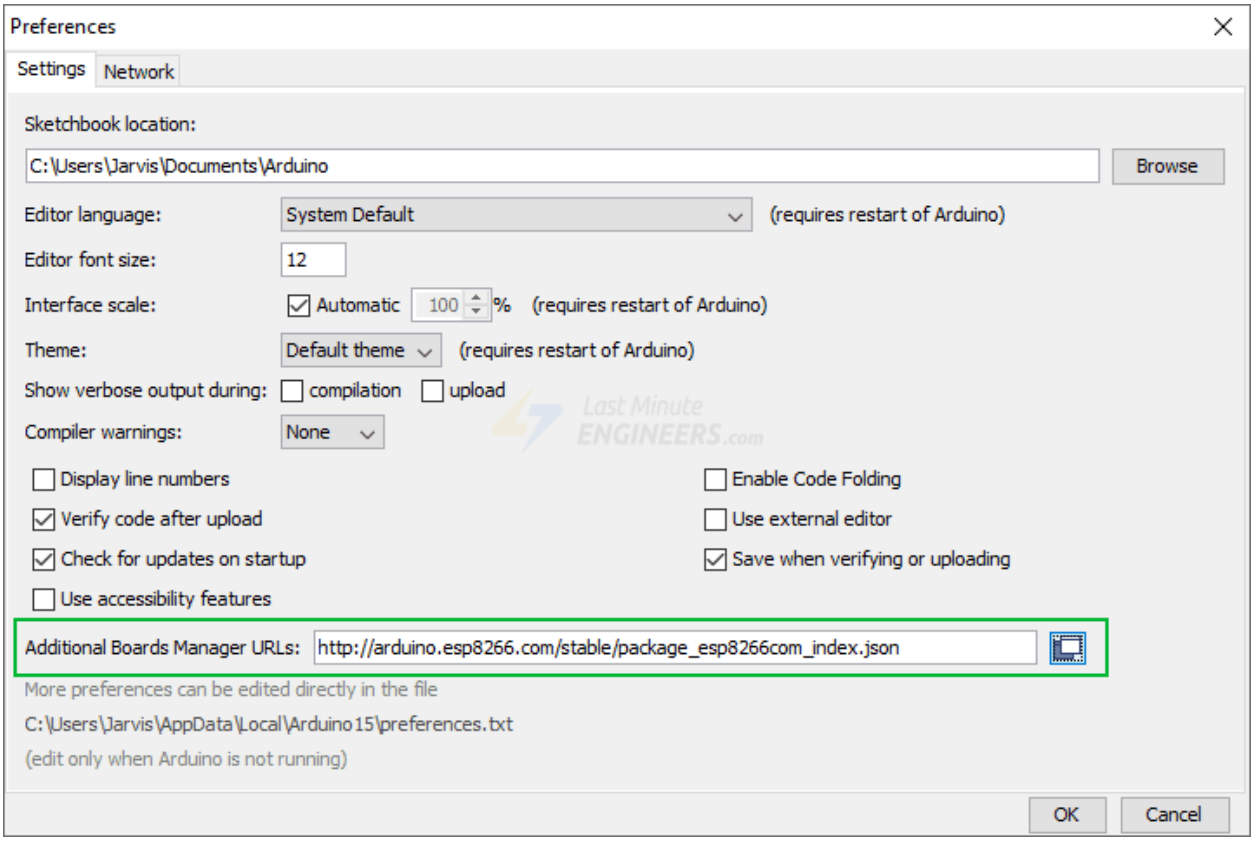
Installing the ESP8266 Arduino Core

Launch the Arduino IDE and navigate to File > Preferences.

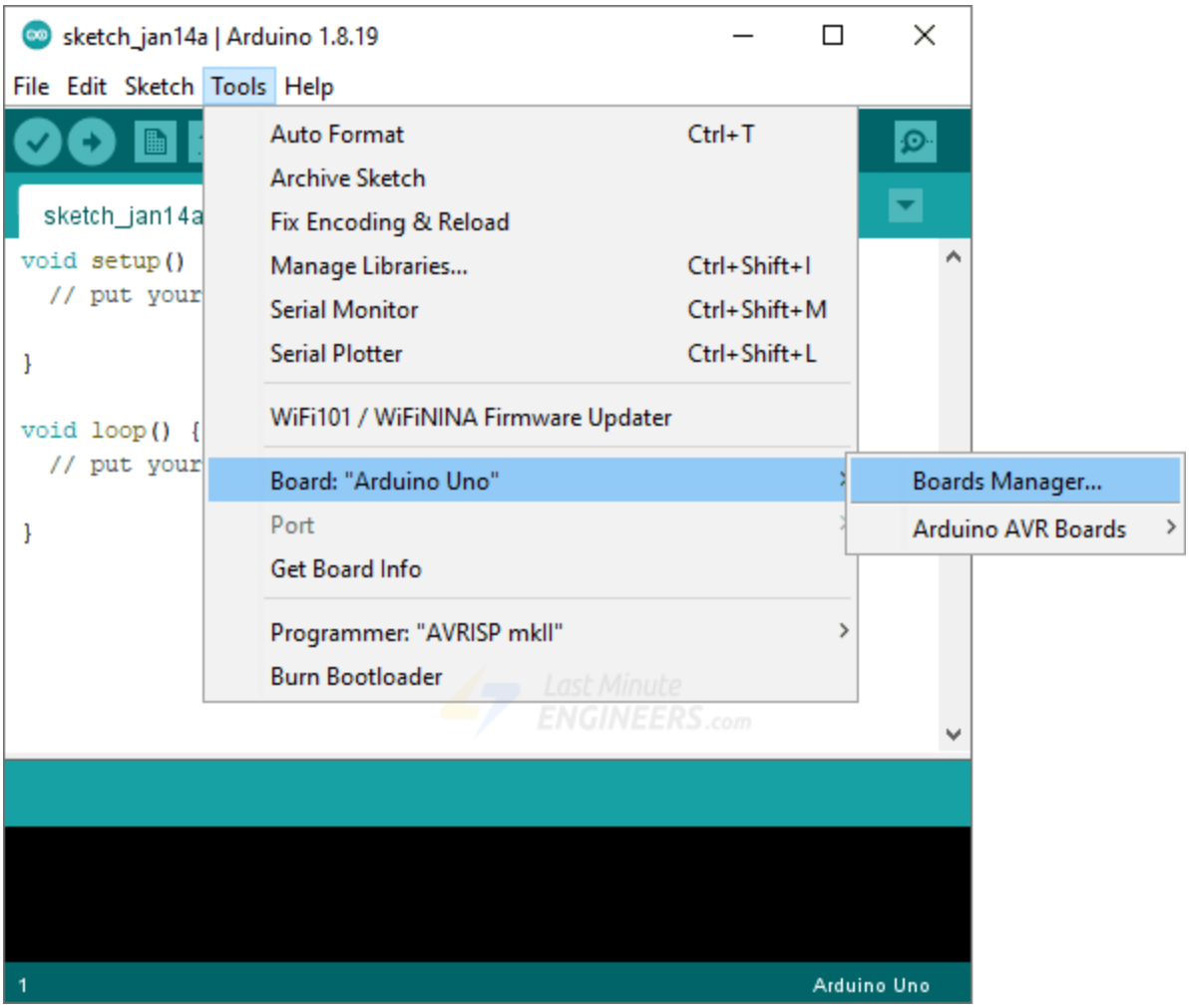


Fill in the “Additional Board Manager URLs” field with the following. Then, click the “OK” button.

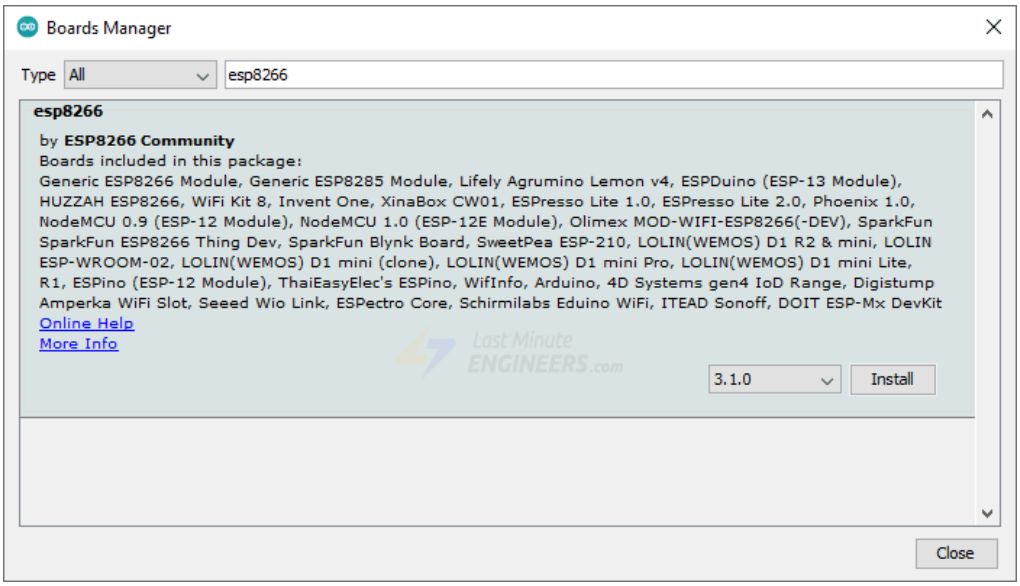
http://arduino.esp8266.com/stable/package_esp8266com_index.json



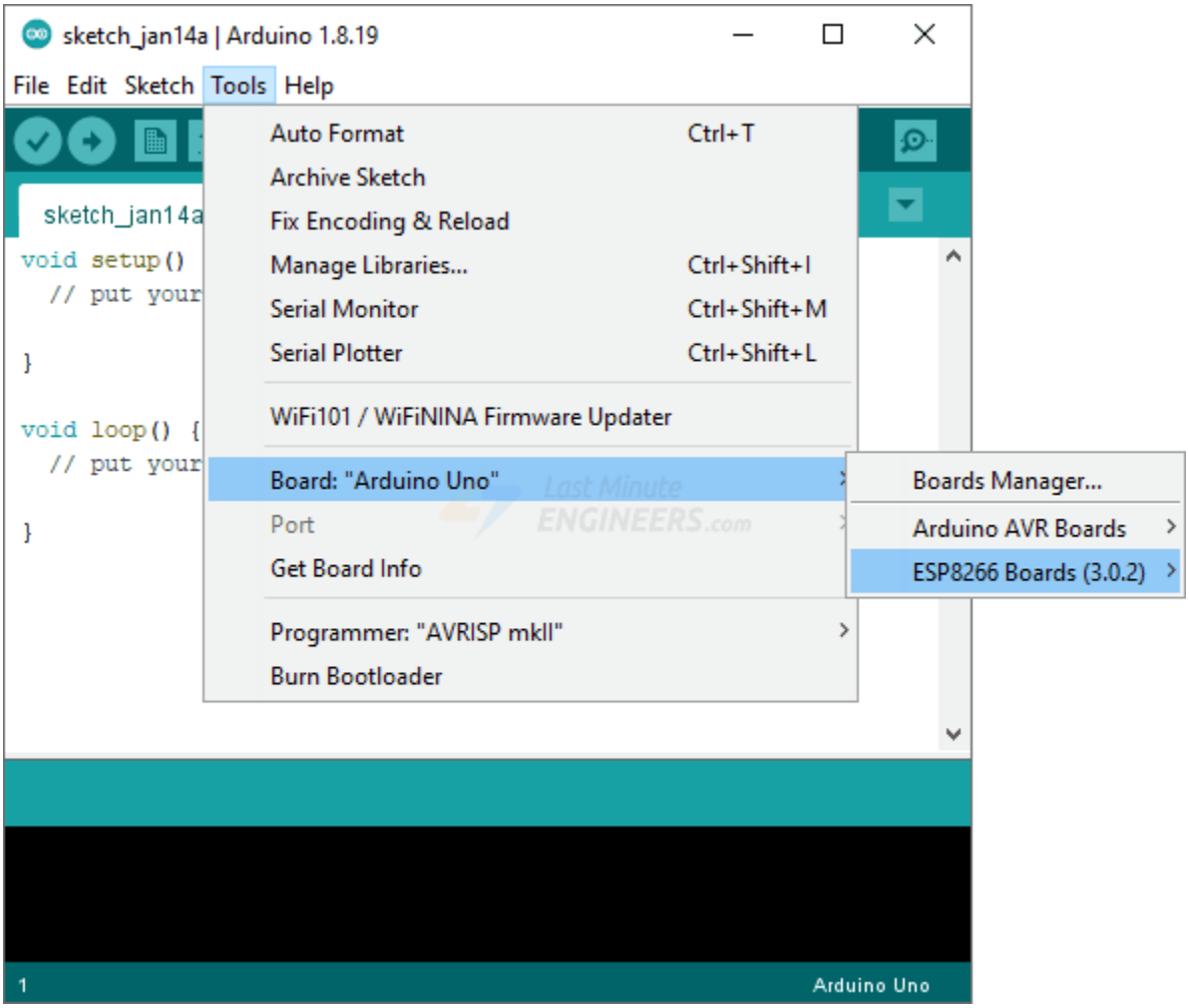
Now navigate to Tools > Board > Boards Manager...



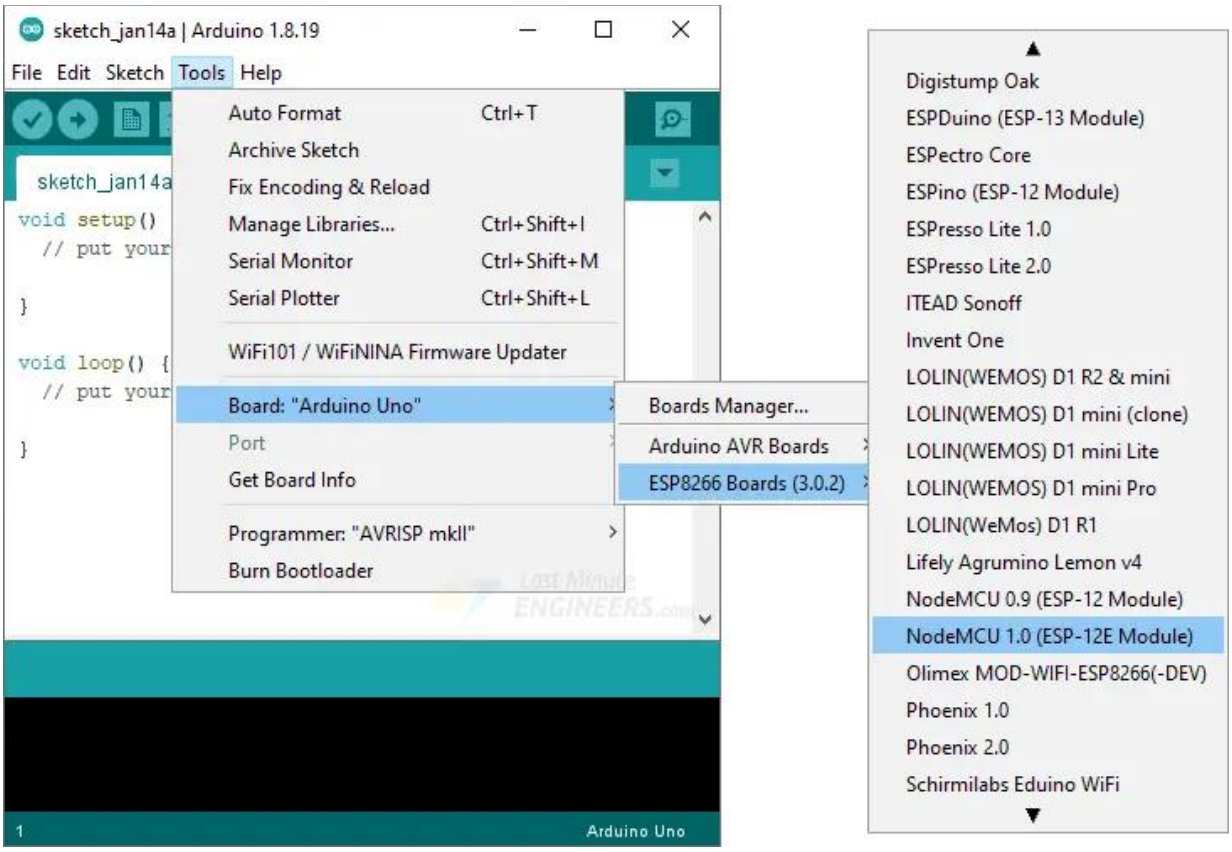
Filter your search by entering ‘esp8266‘. Look for ESP8266 by ESP8266 Community. Click on that entry and then choose Install.



After installing the ESP8266 Arduino Core, restart your Arduino IDE and navigate to Tools > Board to ensure you have ESP8266 boards available.



Now select your board in the Tools > Board menu (in our case, it's the NodeMCU 1.0 (ESP-12E Module)). If you are unsure which board you have, select the Generic ESP8266 Module.



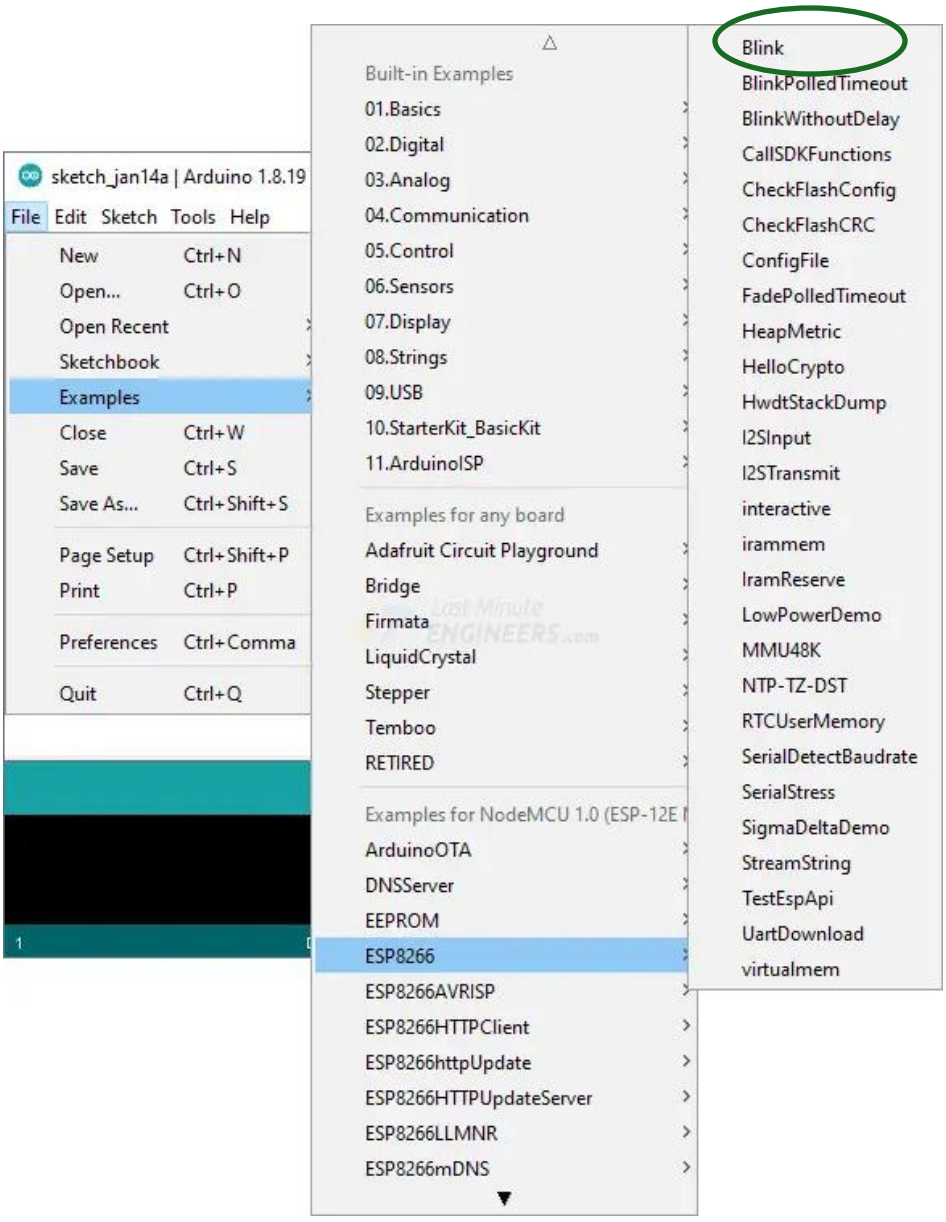
Finally, connect the ESP8266 NodeMCU to your computer and select the Port.

Testing the Installation

Once you've finished the preceding steps, you are ready to test your first program with your ESP8266! Launch the Arduino IDE. If you disconnected your board, plug it back in.

The ESP8266 Arduino core includes several examples that demonstrate everything from scanning for nearby networks to building a web server. To access the example sketches, navigate to File > Examples > ESP8266.

You will see a selection of example sketches. You can choose any of them to load the sketch into your IDE and begin experimenting.



Last Minute ENGINEERS.com

Details About ESP8266

https://lastminuteengineers.com/esp8266-pinout-reference/