EAST WEST UNIVERSITY

Group Work

Course Code: CSE406

Course Title: Internet of Things

Section: 01

Lab Title: UART Interface

Submitted By:

Name:Sabera Zannat Sheba

ID:2022-2-60-095

Other Group Members:

Name	ID
Tanjila Akter	2022-1-60-078
Ronjon Kar	2022-1-60-091
Suddip Paul Arnab	2022-1-60-356

Submitted To:

Dr. Raihan Ul Islam

Associate Professor

Department of Computer Science Engineering East West University

1. Introduction

In this experiment, we learned how to set up I2C communication between two ESP32 boards — one acting as a Master and the other as a Slave. I2C is a two-wire protocol that uses SDA (data line) and SCL (clock line) to transfer data between devices. It is commonly used in IoT systems for connecting sensors and microcontrollers.

The Master sends data packets to the Slave, and the Slave replies with an acknowledgment (ACK) after checking the data using a checksum. A comparative stress test was also performed using different clock speeds, message sizes, and time intervals to measure performance in terms of throughput, message rate, and error rate.

2. Objective

The main goals of this lab are to:

- Understand the basics of I2C communication including SDA, SCL, addressing, and speed modes.
- Connect two ESP32 DevKit boards using I2C (Master–Slave) communication.
- Write programs for the master and slave using Arduino IDE.
- Implement data transmission and acknowledgment (ACK) using callbacks.
- Perform a comparative stress test to analyze performance using different clock speeds, message sizes, and time intervals.
- Compute Throughput, Message rate, and Error rate, and select the best configuration.

3. Hardware Components

Component	Quantity	Description
ESP32 DevKit	2	Used as Master and Slave
Jumper wires	3	For SDA, SCL, and GND connections
Resistors	2	4.7kΩ–10kΩ pull-ups to 3.3V

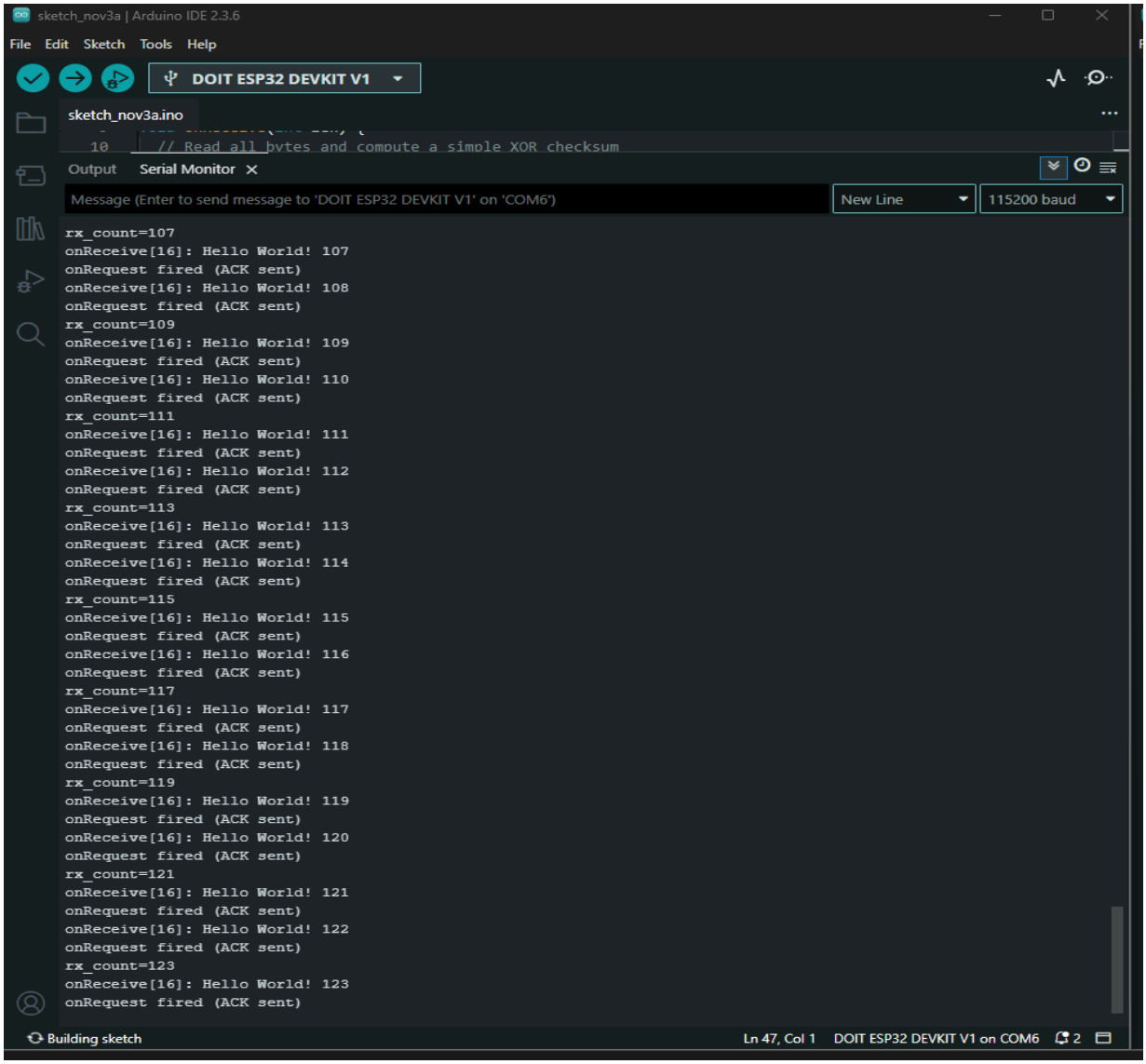
Connections:

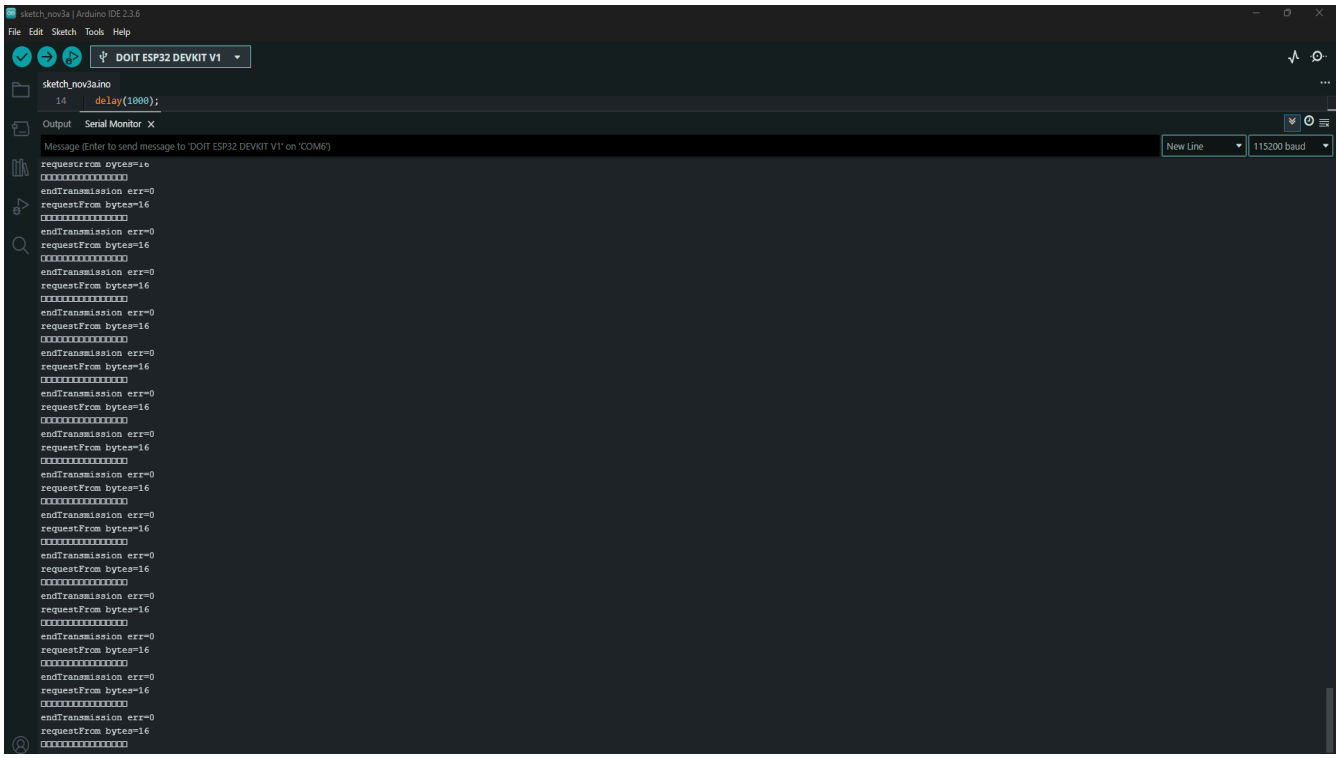
- Master GPIO21 (SDA) → Slave GPIO21 (SDA)
- Master GPIO22 (SCL) → Slave GPIO22 (SCL)
- GND ↔ GND (common ground)

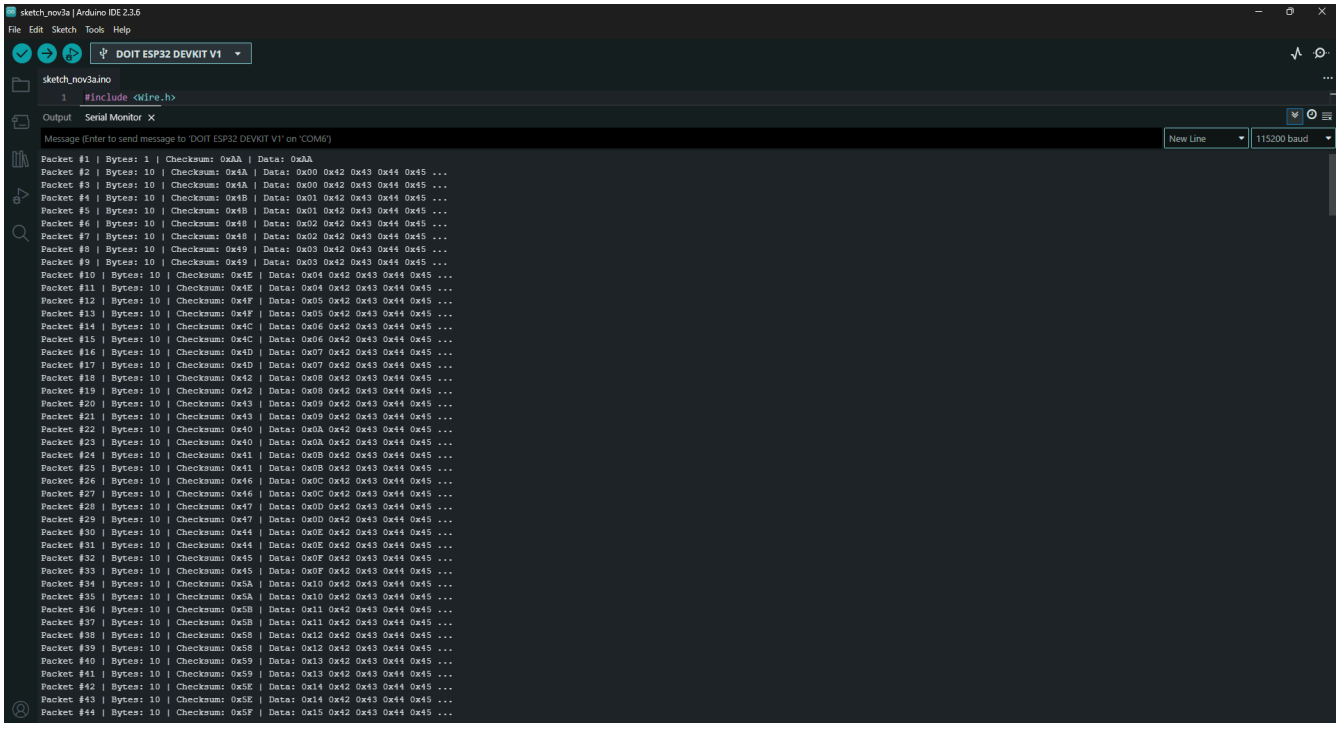
4. Software Tools

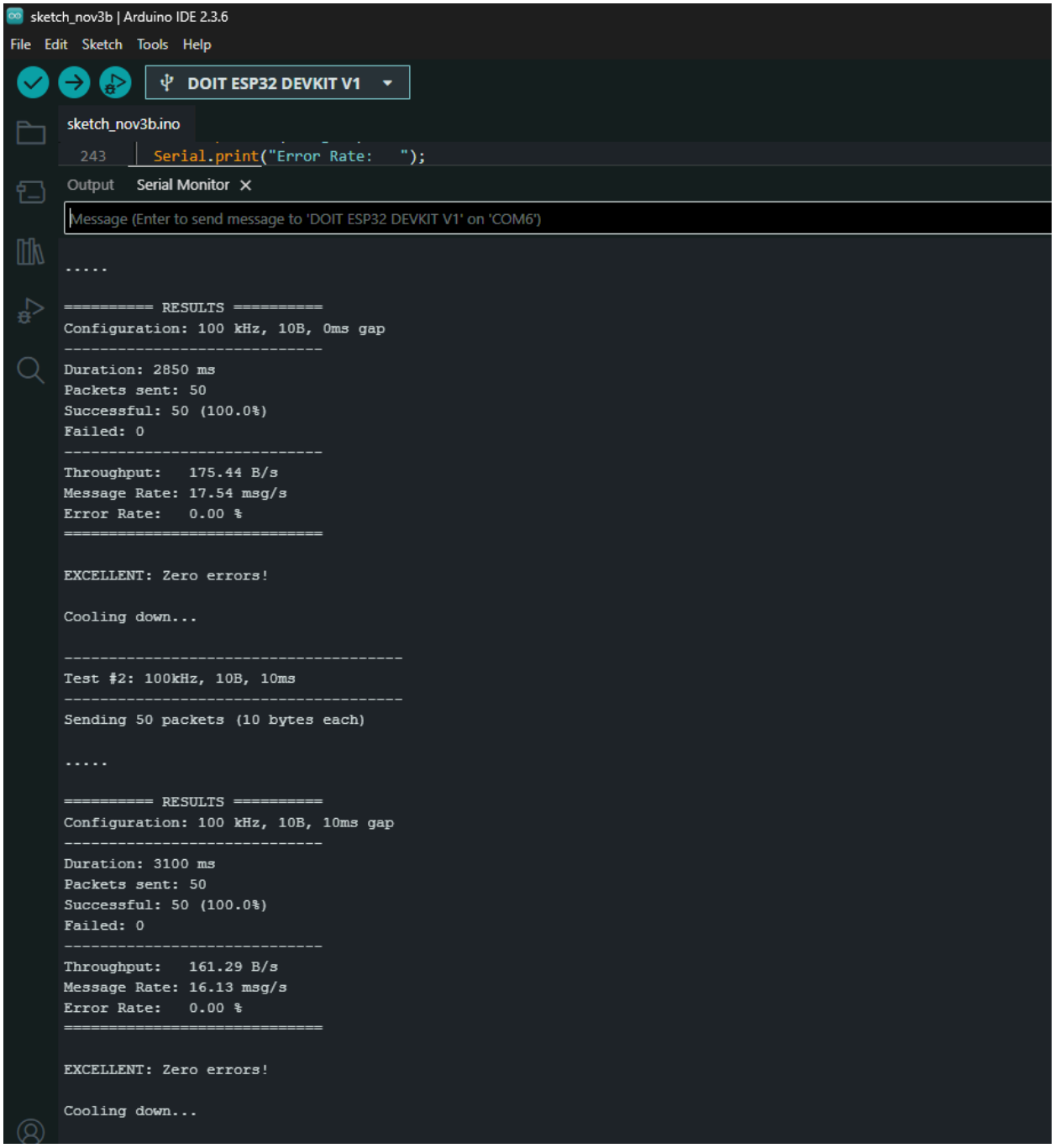
- **Arduino IDE** (with ESP32 core installed)
- **Serial Monitor** (115200 baud rate)

5. Results


sketch_nov3a | Arduino IDE 2.3.6
File Edit Sketch Tools Help
DOIT ESP32 DEVKIT V1
sketch_nov3a.ino
// Read all bytes and compute a simple XOR checksum
Output Serial Monitor
Message (Enter to send message to 'DOIT ESP32 DEVKIT V1' on 'COM6')
New Line
115200 baud
rx_count=107
onReceive[16]: Hello World! 107
onRequest fired (ACK sent)
onReceive[16]: Hello World! 108
onRequest fired (ACK sent)
rx_count=109
onReceive[16]: Hello World! 109
onRequest fired (ACK sent)
onReceive[16]: Hello World! 110
onRequest fired (ACK sent)
rx_count=111
onReceive[16]: Hello World! 111
onRequest fired (ACK sent)
onReceive[16]: Hello World! 112
onRequest fired (ACK sent)
rx_count=113
onReceive[16]: Hello World! 113
onRequest fired (ACK sent)
onReceive[16]: Hello World! 114
onRequest fired (ACK sent)
rx_count=115
onReceive[16]: Hello World! 115
onRequest fired (ACK sent)
onReceive[16]: Hello World! 116
onRequest fired (ACK sent)
rx_count=117
onReceive[16]: Hello World! 117
onRequest fired (ACK sent)
onReceive[16]: Hello World! 118
onRequest fired (ACK sent)
rx_count=119
onReceive[16]: Hello World! 119
onRequest fired (ACK sent)
onReceive[16]: Hello World! 120
onRequest fired (ACK sent)
rx_count=121
onReceive[16]: Hello World! 121
onRequest fired (ACK sent)
onReceive[16]: Hello World! 122
onRequest fired (ACK sent)
rx_count=123
onReceive[16]: Hello World! 123
onRequest fired (ACK sent)
Building sketch
Ln 47, Col 1
DOIT ESP32 DEVKIT V1 on COM6


sketch_nov3a | Arduino IDE 2.3.6
File Edit Sketch Tools Help
DOIT ESP32 DEVKIT V1
sketch_nov3a.ino
14 delay(1000);
Output Serial Monitor
Message (Enter to send message to 'DOIT ESP32 DEVKIT V1' on 'COM6')
New Line
115200 baud
requestFrom bytes=16
endTransmission err=0


sketch_nov3a | Arduino IDE 2.3.6
File Edit Sketch Tools Help
DOIT ESP32 DEVKIT V1
sketch_nov3a.ino
1 #include <Wire.h>
Output Serial Monitor
Message (Enter to send message to 'DOIT ESP32 DEVKIT V1' on 'COM6')
New Line
115200 baud
Packet #1 | Bytes: 1 | Checksum: 0xAA | Data: 0xAA
Packet #2 | Bytes: 10 | Checksum: 0x4A | Data: 0x00 0x42 0x43 0x44 0x45 ...
Packet #3 | Bytes: 10 | Checksum: 0x4A | Data: 0x00 0x42 0x43 0x44 0x45 ...
Packet #4 | Bytes: 10 | Checksum: 0x4B | Data: 0x01 0x42 0x43 0x44 0x45 ...
Packet #5 | Bytes: 10 | Checksum: 0x4B | Data: 0x01 0x42 0x43 0x44 0x45 ...
Packet #6 | Bytes: 10 | Checksum: 0x48 | Data: 0x02 0x42 0x43 0x44 0x45 ...
Packet #7 | Bytes: 10 | Checksum: 0x48 | Data: 0x02 0x42 0x43 0x44 0x45 ...
Packet #8 | Bytes: 10 | Checksum: 0x49 | Data: 0x03 0x42 0x43 0x44 0x45 ...
Packet #9 | Bytes: 10 | Checksum: 0x49 | Data: 0x03 0x42 0x43 0x44 0x45 ...
Packet #10 | Bytes: 10 | Checksum: 0x4E | Data: 0x04 0x42 0x43 0x44 0x45 ...
Packet #11 | Bytes: 10 | Checksum: 0x4E | Data: 0x04 0x42 0x43 0x44 0x45 ...
Packet #12 | Bytes: 10 | Checksum: 0x4F | Data: 0x05 0x42 0x43 0x44 0x45 ...
Packet #13 | Bytes: 10 | Checksum: 0x4F | Data: 0x05 0x42 0x43 0x44 0x45 ...
Packet #14 | Bytes: 10 | Checksum: 0x4C | Data: 0x06 0x42 0x43 0x44 0x45 ...
Packet #15 | Bytes: 10 | Checksum: 0x4C | Data: 0x06 0x42 0x43 0x44 0x45 ...
Packet #16 | Bytes: 10 | Checksum: 0x4D | Data: 0x07 0x42 0x43 0x44 0x45 ...
Packet #17 | Bytes: 10 | Checksum: 0x4D | Data: 0x07 0x42 0x43 0x44 0x45 ...
Packet #18 | Bytes: 10 | Checksum: 0x42 | Data: 0x08 0x42 0x43 0x44 0x45 ...
Packet #19 | Bytes: 10 | Checksum: 0x42 | Data: 0x08 0x42 0x43 0x44 0x45 ...
Packet #20 | Bytes: 10 | Checksum: 0x43 | Data: 0x09 0x42 0x43 0x44 0x45 ...
Packet #21 | Bytes: 10 | Checksum: 0x43 | Data: 0x09 0x42 0x43 0x44 0x45 ...
Packet #22 | Bytes: 10 | Checksum: 0x40 | Data: 0x0A 0x42 0x43 0x44 0x45 ...
Packet #23 | Bytes: 10 | Checksum: 0x40 | Data: 0x0A 0x42 0x43 0x44 0x45 ...
Packet #24 | Bytes: 10 | Checksum: 0x41 | Data: 0x0B 0x42 0x43 0x44 0x45 ...
Packet #25 | Bytes: 10 | Checksum: 0x41 | Data: 0x0B 0x42 0x43 0x44 0x45 ...
Packet #26 | Bytes: 10 | Checksum: 0x46 | Data: 0x0C 0x42 0x43 0x44 0x45 ...
Packet #27 | Bytes: 10 | Checksum: 0x46 | Data: 0x0C 0x42 0x43 0x44 0x45 ...
Packet #28 | Bytes: 10 | Checksum: 0x47 | Data: 0x0D 0x42 0x43 0x44 0x45 ...
Packet #29 | Bytes: 10 | Checksum: 0x47 | Data: 0x0D 0x42 0x43 0x44 0x45 ...
Packet #30 | Bytes: 10 | Checksum: 0x44 | Data: 0x0E 0x42 0x43 0x44 0x45 ...
Packet #31 | Bytes: 10 | Checksum: 0x44 | Data: 0x0E 0x42 0x43 0x44 0x45 ...
Packet #32 | Bytes: 10 | Checksum: 0x45 | Data: 0x0F 0x42 0x43 0x44 0x45 ...
Packet #33 | Bytes: 10 | Checksum: 0x45 | Data: 0x0F 0x42 0x43 0x44 0x45 ...
Packet #34 | Bytes: 10 | Checksum: 0x5A | Data: 0x10 0x42 0x43 0x44 0x45 ...
Packet #35 | Bytes: 10 | Checksum: 0x5A | Data: 0x10 0x42 0x43 0x44 0x45 ...
Packet #36 | Bytes: 10 | Checksum: 0x5B | Data: 0x11 0x42 0x43 0x44 0x45 ...
Packet #37 | Bytes: 10 | Checksum: 0x5B | Data: 0x11 0x42 0x43 0x44 0x45 ...
Packet #38 | Bytes: 10 | Checksum: 0x58 | Data: 0x12 0x42 0x43 0x44 0x45 ...
Packet #39 | Bytes: 10 | Checksum: 0x58 | Data: 0x12 0x42 0x43 0x44 0x45 ...
Packet #40 | Bytes: 10 | Checksum: 0x59 | Data: 0x13 0x42 0x43 0x44 0x45 ...
Packet #41 | Bytes: 10 | Checksum: 0x59 | Data: 0x13 0x42 0x43 0x44 0x45 ...
Packet #42 | Bytes: 10 | Checksum: 0x5E | Data: 0x14 0x42 0x43 0x44 0x45 ...
Packet #43 | Bytes: 10 | Checksum: 0x5E | Data: 0x14 0x42 0x43 0x44 0x45 ...
Packet #44 | Bytes: 10 | Checksum: 0x5F | Data: 0x15 0x42 0x43 0x44 0x45 ...



6.Discussion

- The experiment shows that higher I2C frequencies (400kHz) improve throughput significantly.

- Adding a small delay (10ms) between packets reduces bus congestion and error rate.

- Larger message sizes (50 bytes) increase throughput but may cause more errors if the bus is overloaded.

- The best performance is observed at 400kHz, 10B messages, 10ms delay, providing a stable and fast data exchange.

7. Conclusion

This lab successfully demonstrated I2C communication between two ESP32 boards using a master–slave setup. It provided practical knowledge on configuring I2C communication in the Arduino IDE and understanding how bus frequency, message size, and timing affect system reliability. Students also learned how to calculate and analyze throughput, message rate, and error rate to evaluate communication performance. Overall, the experiment showed that the recommended configuration offers high efficiency with minimal error, making it well-suited for reliable real-time IoT data transmission.