ECOLE NATIONALE
SUPÉRIEURE
D'INFORMATIQUE

الجـــمهوريـــة الجزائريـــة الديمـــقراطيـــة الشـعبيـة
وزارة التعليـــم العـــالي والبـــحث العلمـــي
People's Democratic Republic of Algeria
Ministry of Higher Education and Scientific Research

Intelligent and Communicating Systems, ICS

2^{nd} Year Specialty SIL G1

Homework Report n°03

Title:

I2C-ZigBee Simulation

Studied by:

First Name: Yasmine

Last Name: DINARI

E_mail: ly_dinari@esi.dz

Contents

1 Theory **2**

1. The ZigBee Protocol 2

1.1. ZigBee networks components: 2

1.2. Supported topologies: 2

2. Zigbee's concurrent place in the IoT world 2

2.1. Zigbee's Key advantages: 2

2 Activity **4**

1. ZigBee simulation 4

1.1. Download XBee Module Library for Proteus 4

1.2. Create the Transmitter Project 4

1.3. Create the Receiver Project 5

1.4. Create Virtual COM Port Pair 7

1.5. Write and Upload the Code 7

1.6. Run the Simulation 10

2. I2c simulation 11

2.1. Create a New Project in Proteus 11

2.2. Set Up the Circuit 11

2.3. Write and Upload the Code 12

2.4. Run the Simulation 15

3 Conclusion **17**

Theory

1. The ZigBee Protocol

ZigBee is a low-power, reliable wireless **communication protocol** based on IEEE 802.15.4, ideal for IoT applications requiring **minimal** energy use. It operates mainly in the 2.4 GHz band, with data rates up to 250 kbit/s and a range of 10–100 meters. The protocol stack includes the Physical, MAC, Network, and Application layers.

1.1. ZigBee networks components:

ZigBee networks include:

- *Coordinator (ZC):* Manages the network (one per network).
- *Router (ZR):* Extends the network range and routes messages.
- *End Device (ZED):* Low-power, non-routing devices like sensors.

1.2. Supported topologies:

Star, Point-to-Point, and Mesh (for robust, scalable communication). ZigBee uses AES-128 encryption for security.

2. Zigbee's concurrent place in the IoT world

ZigBee is a leading protocol in the IoT industry, especially in smart homes and industrial applications. As of 2025, it powers around 85% of 802.15.4 mesh chipsets, with over a billion units sold worldwide.

2.1. Zigbee's Key advantages:

- *Low Power Consumption:* Enables devices to run for years on batteries.
- *High Scalability:* Supports networks with over 65,000 nodes.
- *Mesh Reliability:* Robust communication via self-healing mesh networks.
- Interoperability: Devices from different manufacturers work together seamlessly.

- *Widespread Adoption:* Used by major companies like Amazon, Ikea, Xiaomi, and Philips.

ZigBee mainly competes with Z-Wave, but differs by **being open-source**, while Z-Wave is **proprietary**. Though ZigBee lacks **native Internet Protocol (IP) support**, it **integrates** with cloud and external systems via gateways.

Activity

1. ZigBee simulation

This is comprehensive walkthrough of all the steps to follow to achieve this wireless communication using a pushbutton and LED:

1.1. Download XBee Module Library for Proteus

Download XBee Module Library for Proteus

- Download the XBee Library
- Extract the zip file to get XBeeTEP.LIB and XBeeTEP.IDX files
- Place these files in the libraries folder of Proteus (C:Files (x86)Electronics8 Professional)

Download Genuino/Arduino Library for Proteus

- Download the Arduino Library
- Extract and place the library files in the Proteus library folder

Install Eltima Virtual Port Driver

- Download and install Eltima Virtual Serial Port Driver
- This will be used to create virtual COM port pairs for the XBee modules

1.2. Create the Transmitter Project

Start a New Project in Proteus

- Open Proteus and create a new project for the transmitter

Add Components

- Add an Arduino Uno board (or Genuino Uno board) from the component library
- Add an XBee module (search for ”XBee Module” in the component library)

- Add a pushbutton and connect it between pin 2 and ground

Connect the XBee Module to Arduino

- Connect TX pin of Arduino to TX pin of XBee
- Connect RX pin of Arduino to RX pin of XBee

Note: *In Proteus, XBee modules use TX to TX and RX to RX connections (unlike real world hardware)*

Configure the XBee Module

- Double-click on the XBee module to open its properties
- Set the COM port to COM1
- Set the baud rate to 9600

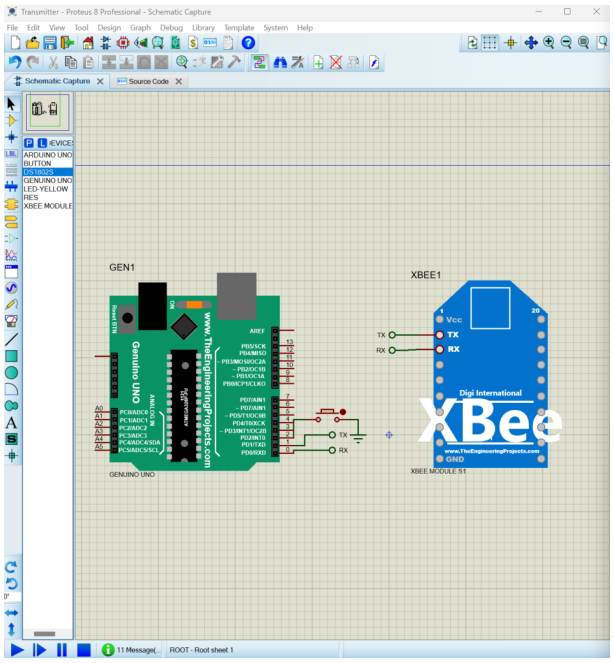

Figure 2.1: Transmitter Project Setup

1.3. Create the Receiver Project

Start Another Project in Proteus

- Create a new project for the receiver

Add Components

- Add an Arduino Uno board (or Genuino Uno board)
- Add an XBee module
- Add an LED-AQUA (active component) and connect it to pin 12
- Add a 220Ω current-limiting resistor in series with the LED

Connect the XBee Module to Arduino

- Connect TX pin of Arduino to TX pin of XBee
- Connect RX pin of Arduino to RX pin of XBee

Configure the XBee Module

- Double-click on the XBee module
- Set the COM port to COM2
- Set the baud rate to 9600

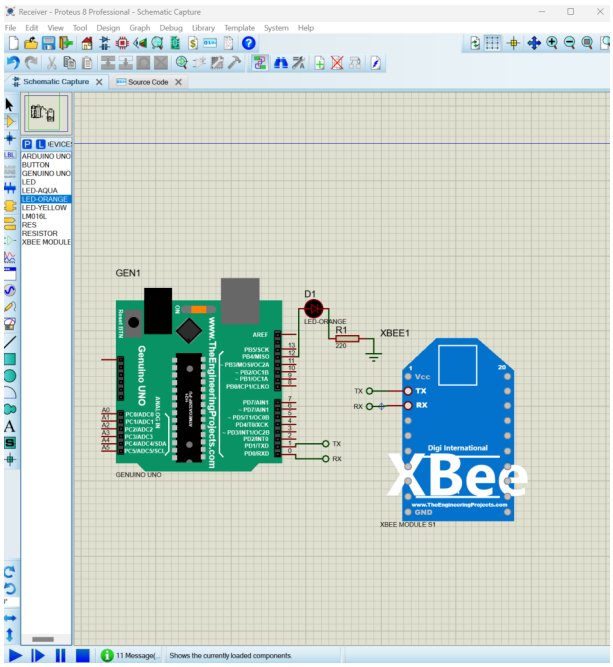

Figure 2.2: Receiver Project Setup

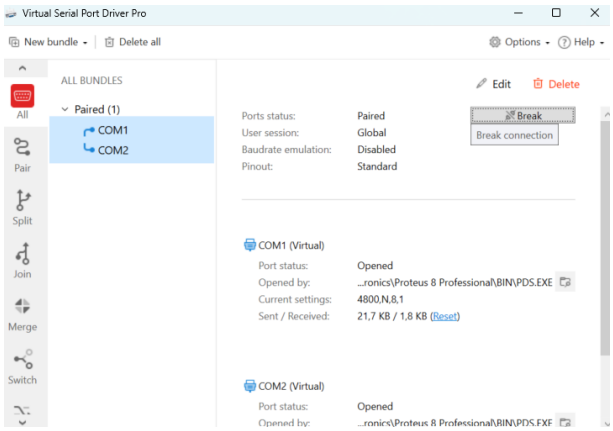

Figure 2.3: Port Pairing

1.4. Create Virtual COM Port Pair

Launch Eltima Virtual Serial Port Driver

- Open the Virtual Serial Port Driver application

Create a Paired Bundle

- Select ”Pair” on the left pane
- Click ”Add a new pair” button
- Select COM1 as the first port and COM2 as the second port
- Click ”Create” to establish the virtual connection

Verify Port Pairing

- Ensure the status shows ”Paired” for the COM1-COM2 bundle
- This virtual connection allows the XBee modules to communicate with each other

1.5. Write and Upload the Code

Transmitter Code

- Click on the ”Source Code” icon in Proteus
- Enter the following code for the transmitter:

```
/* Transmitter.ino file for ZigBee communication
 * Created: sam. mai 3 2025
 * Processor: Arduino Uno
 * Compiler: Arduino AVR (Proteus)
 */

#pragma GCC push_options
#pragma GCC optimize ("O1")

// Define pin for pushbutton
const int buttonPin = 2;  // Pushbutton connected to pin 2

// Variables
int buttonState = 0;      // Current state of the button
int lastButtonState = 0;  // Previous state of the button

void setup() {
  // Initialize serial communications at 9600 bps
  Serial.begin(9600);

  // Initialize the pushbutton pin as input with pull-up resistor
  pinMode(buttonPin, INPUT_PULLUP);

  // Print startup message
  Serial.println("ZigBee Transmitter Ready");
  Serial.println("Button on pin 2, using INPUT_PULLUP mode");
}

void loop() {
  // Read the state of the pushbutton (inverted due to pull-up)
  buttonState = !digitalRead(buttonPin);

  // Check if button state has changed
  if (buttonState != lastButtonState) {
    // If button is pressed (HIGH)
    if (buttonState == HIGH) {
      Serial.println("Button pressed - sending ON command");
      Serial.write('1');  // Send '1' character to turn LED ON
    } else {
      Serial.println("Button released - sending OFF command");
      Serial.write('0');  // Send '0' character to turn LED OFF
    }

    // Update the last button state
    lastButtonState = buttonState;

    // Small delay to prevent bouncing
    delay(50);
  }
}

#pragma GCC pop_options
```

Listing 2.1: Transmitter code

Receiver Code

- Click on the "Source Code" icon in the receiver project
- Enter the following code:

```
/* Receiver.ino file for ZigBee communication
 * Created: sam. mai 3 2025
 * Processor: Arduino Uno
 * Compiler: Arduino AVR (Proteus)
 */

#pragma GCC push_options
#pragma GCC optimize ("O1")
// Define pin for LED
const int ledPin = 12;     // LED connected to pin 12
void setup() {
  // Initialize serial communications at 9600 bps
  Serial.begin(9600);
  // Initialize the LED pin as output
  pinMode(ledPin, OUTPUT);
  digitalWrite(ledPin, LOW);  // Initially turn off the LED
  // Print startup message
  Serial.println("ZigBee Receiver Ready");
  Serial.println("LED connected to pin 12");
  // Test LED at startup
  Serial.println("Testing LED...");
  digitalWrite(ledPin, HIGH);
  delay(1000);
  digitalWrite(ledPin, LOW);
  Serial.println("LED test complete");
}

void loop() {
  // Check if data is available from serial (XBee)
  if (Serial.available() > 0) {
    // Read the incoming byte
    char command = Serial.read();

    // Process the command
    if (command == '1') {
      Serial.println("Received ON command - Turning LED ON");
      digitalWrite(ledPin, HIGH);  // Turn LED ON
    }
    else if (command == '0') {
      Serial.println("Received OFF command - Turning LED OFF");
      digitalWrite(ledPin, LOW);   // Turn LED OFF
    }
  }
}

#pragma GCC pop_options
```

Listing 2.2: Receiver code

Compile the Code

- Click the ”Build” button to compile your code
- If you encounter ”Failed to set firmware property” error, manually set the path to the generated .hex file
- The hex file is typically located at: C: / Users/[Username]/AppData/Local/Temp/VSM Studio/[random]/ARDUINO_UNO_1 / Debugging/Debug.hex

1.6. Run the Simulation

Start the Simulation

- Ensure both projects are open
- Start the simulation in both projects
- Verify that the COM ports are properly paired and showing ”Opened” status in Eltima

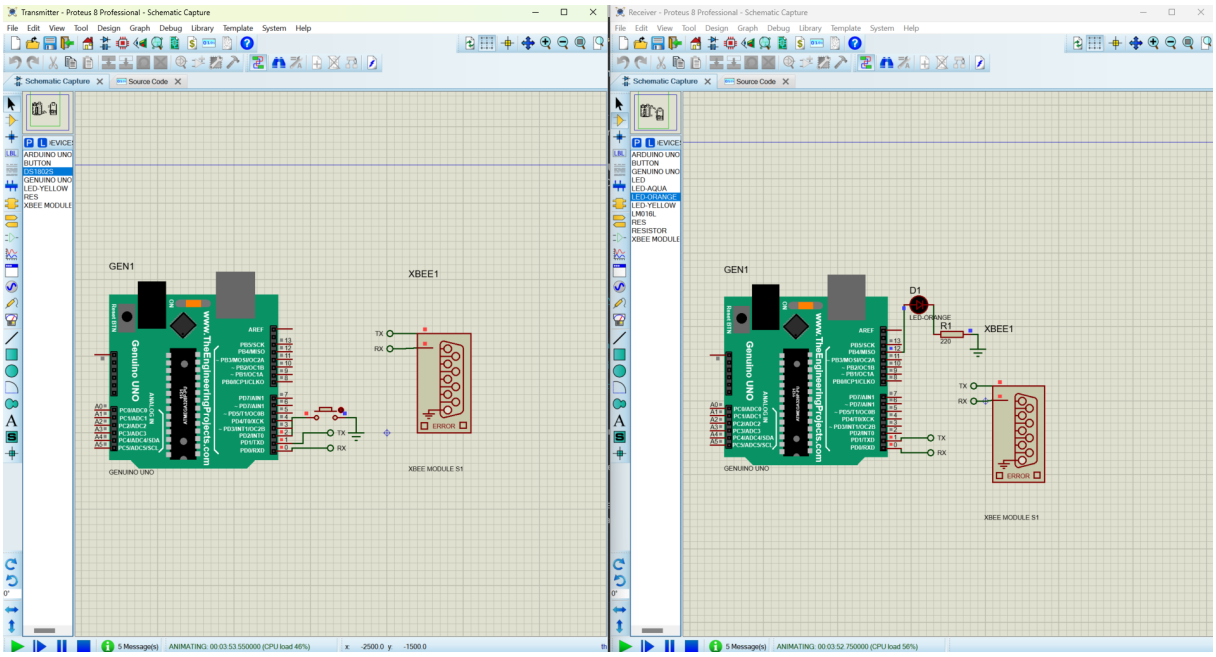

Figure 2.4: Simulation Inactive

Test the Communication

- Click on the pushbutton in the transmitter circuit
- Observe the LED in the receiver circuit turning on when the button is pressed

- The LED should turn off when the button is released

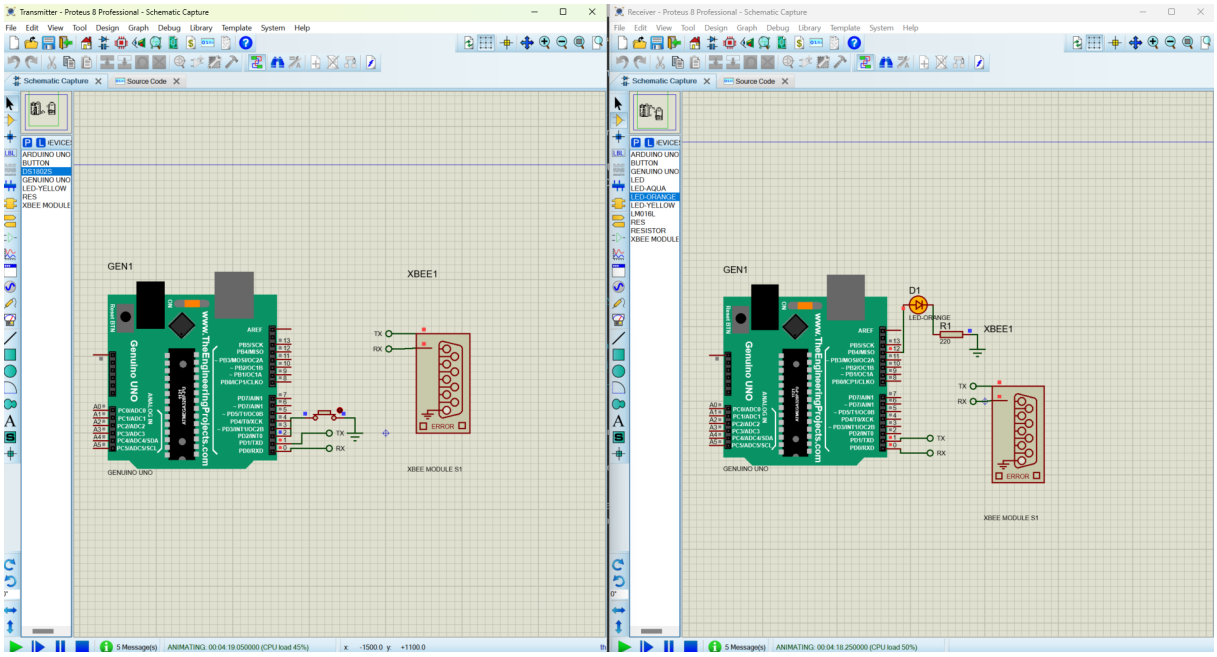

Figure 2.5: Simulation Active

2. I2c simulation

This is a comprehensive walkthrough of all the steps to follow to achieve communication between two Arduino boards using I2C protocol with a pushbutton and LED:

2.1. Create a New Project in Proteus

- Open Proteus and create a new project for the I2C simulation
- Add two Arduino Uno boards from the component library

2.2. Set Up the Circuit

Connect the I2C Lines

- Connect pin A4 (SDA) of both Arduino boards together
- Connect pin A5 (SCL) of both Arduino boards together

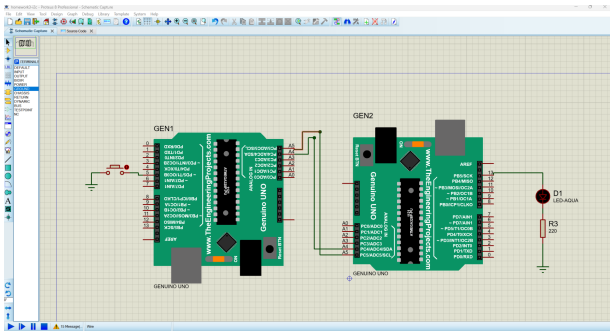

Figure 2.6: Set up of I2C communication

Add Components

- On the Master Arduino (left one): Connect a pushbutton between pin 5 and ground
- On the Slave Arduino (right one): Connect a LED (active component) to pin 13
- Add a 220Ω current-limiting resistor between the LED and ground

2.3. Write and Upload the Code

Master Arduino Code (with Pushbutton)

- Click on the ”Source Code” icon in Proteus
- Enter the code for the master Arduino (with pushbutton)

```
/* Main.ino file generated by New Project wizard
 *
 * Created:   sam. mai 3 2025
 * Processor: Arduino Uno
 * Compiler:  Arduino AVR (Proteus)
 */









#pragma GCC push_options
#pragma GCC optimize ("O1")
#include <Wire.h>

const int buttonPin = 5;  // Pushbutton connected to pin 5
int buttonState = 0;      // Current state of the button
int lastButtonState = 0;  // Previous state of the button

void setup() {
  Wire.begin();           // Initialize as master
  Serial.begin(9600);     // Start serial for debugging
```

```
  pinMode(buttonPin, INPUT_PULLUP);  // Set button pin as input with pull-up
  Serial.println("I2C Master Ready");
}

void loop() {
  // Read the state of the pushbutton (inverted due to pull-up)
  buttonState = !digitalRead(buttonPin);

  // Check if button state has changed
  if (buttonState != lastButtonState) {
    // If button is pressed (HIGH)
    if (buttonState == HIGH) {
      Serial.println("Button pressed - sending ON command");
      Wire.beginTransmission(8);  // Transmit to device #8
      Wire.write(1);              // Send value 1 to turn LED ON
      Wire.endTransmission();     // Stop transmitting
    } else {
      Serial.println("Button released - sending OFF command");
      Wire.beginTransmission(8);  // Transmit to device #8
      Wire.write(0);              // Send value 0 to turn LED OFF
      Wire.endTransmission();     // Stop transmitting
    }

    // Update the last button state
    lastButtonState = buttonState;

    // Small delay to prevent bouncing
    delay(50);
  }
}

#pragma GCC pop_options
```

Listing 2.3: Master Arduino code

Slave Arduino Code (with LED)

- Click on the "Source Code" icon for the second Arduino
- Enter the code for the slave Arduino (with LED)

```
/* Main.ino file generated by New Project wizard
 *
 * Created:   sam. mai 3 2025
 * Processor: Arduino Uno
 * Compiler:  Arduino AVR (Proteus)
 */




#pragma GCC push_options
#pragma GCC optimize ("O1")
```

```
14
15 /* Main.ino file generated by New Project wizard
16  *
17  * Created:   sam. mai 3 2025
18  * Processor: Arduino Uno
19  * Compiler:  Arduino AVR (Proteus)
20  */
21
22
23
24
25
26 #pragma GCC push_options
27 #pragma GCC optimize ("O1")
28
29 #include <Wire.h>
30
31 const int ledPin = 13;     // LED connected to pin 13
32
33 void setup() {
34   Wire.begin(8);           // Initialize as slave #8
35   Wire.onReceive(receiveEvent); // Register receive event
36   Serial.begin(9600);      // Start serial for debugging
37   pinMode(ledPin, OUTPUT);
38   digitalWrite(ledPin, LOW);  // Initially turn off the LED
39   Serial.println("I2C Slave Ready");
40 }
41
42 void loop() {
43   // Nothing to do here - everything happens in the receiveEvent function
44   delay(100);
45 }
46
47 // Function that executes whenever data is received from master
48 void receiveEvent(int howMany) {
49   while (Wire.available()) {
50     char c = Wire.read();  // Receive byte as a character
51     Serial.print("Received: ");
52     Serial.println(c);
53
54     if (c == 1) {
55       Serial.println("Turning LED ON");
56       digitalWrite(ledPin, HIGH);
57     } else {
58       Serial.println("Turning LED OFF");
59       digitalWrite(ledPin, LOW);
60     }
61   }
62 }
63
64
65 #pragma GCC pop_options
66
67
68 #pragma GCC pop_options
```

Listing 2.4: Slave Arduino code

Compile the Code

- Click the ”Build” button to compile your code for both Arduinos
- If you encounter ”Failed to set firmware property” error, manually set the path to the generated .hex files
- The hex files are typically located at: C:/Users/[Username]/AppData/Local/Temp/VSM Studio/[random]/ARDUINO_UNO_3/Debug/Debug.hex and C:/Users/[Username]/AppData/L Studio/[random]/ARDUINO_UNO_4/Debug/Debug.hex

2.4. Run the Simulation

Start the Simulation

- Ensure both Arduino boards are properly configured
- Start the simulation in Proteus

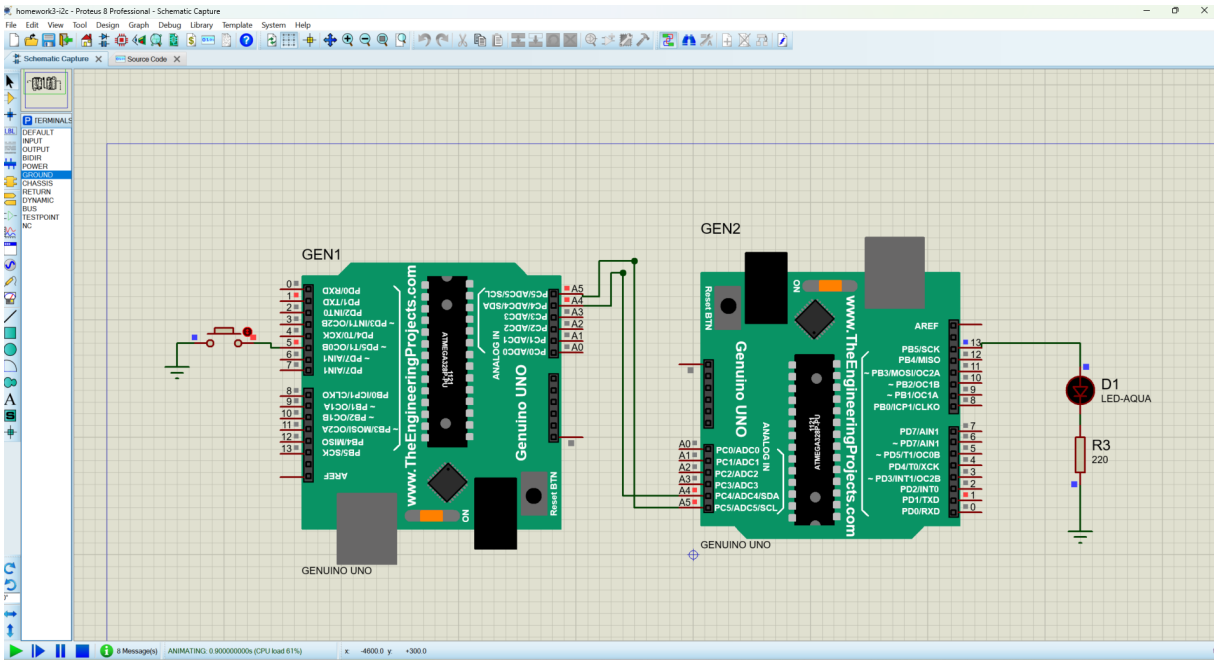

Figure 2.7: Inactive I2C Simulation

Test the Communication

- Click on the pushbutton connected to the Master Arduino
- Observe the LED on the Slave Arduino turning on when the button is pressed
- The LED should turn off when the button is released

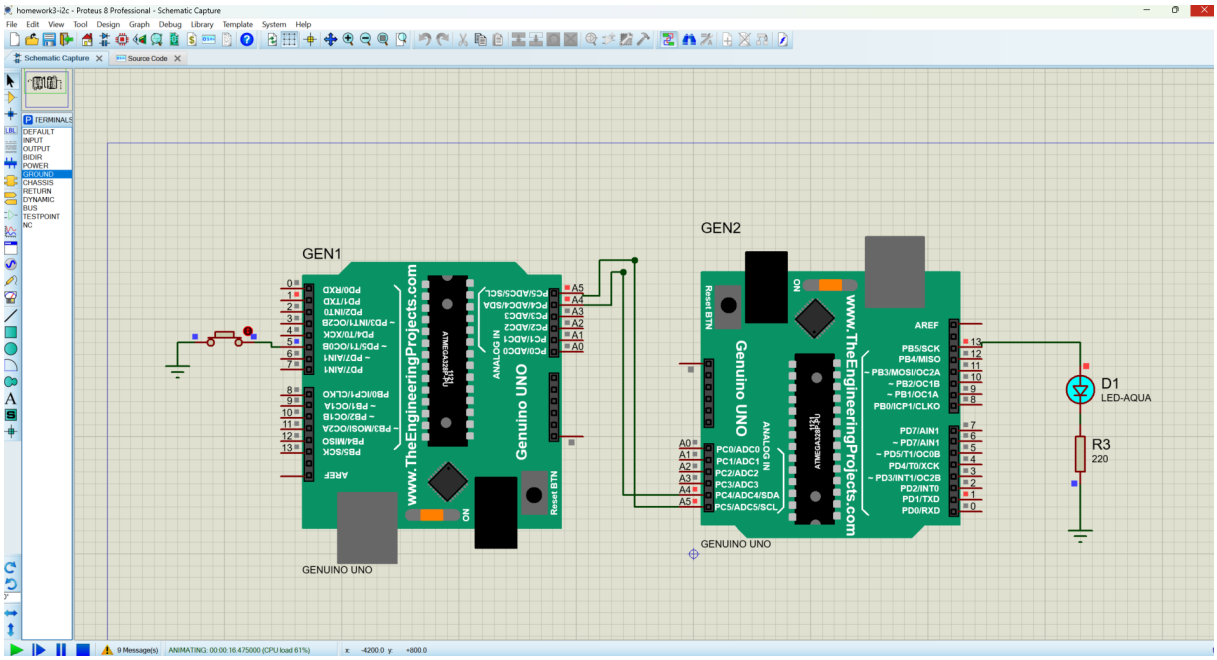

Figure 2.8: Active I2C Simulation

Conclusion

The ZigBee and I2C protocols represent two distinct approaches to device communication in IoT applications. ZigBee offers wireless connectivity through a mesh network topology, requiring XBee modules and virtual COM port pairing to establish communication between devices.

In contrast, I2C provides a simpler wired solution using only two signal lines (SDA and SCL), enabling direct communication between closely positioned components without the need for COM port pairing.

I2C's addressing system allows multiple devices to share the same bus, while its synchronous nature with a dedicated clock signal (SCL) differs fundamentally from ZigBee's wireless asynchronous approach.

Both protocols successfully demonstrated the fundamental IoT interaction pattern of controlling an LED with a pushbutton across different Arduino boards. While ZigBee excels in wireless applications requiring extended range, I2C provides an efficient solution for compact systems with minimal wiring requirements.

These implementations highlight how different communication protocols can be selected based on specific application needs, balancing factors such as power efficiency, communication range, and system complexity.