T.R.
ESKISEHIR OSMANGAZI UNIVERSITY

DEPARTMENT OF ELECTRICAL-ELECTRONICS ENGINEERING
AND
DEPARTMENT OF COMPUTER ENGINEERING

INTRODUCTION TO MICROCOMPUTERS
TERM PROJECT

HOME AUTOMATION SYSTEM

Project Group Members:

151220212123, Yiğit DOMBAYLI, Electrical and Electronics Engineering
152120211092, Canan MUTLU, Computer Engineering
151220222120, Tunahan ŞANAL, Electrical and Electronics Engineering
152120211102, Nurefşan Ceren DOĞAN, Computer Engineering
151220192079, Yusuf İNAN, Electrical and Electronics Engineering
152120211089, Efe Duhan ALPAY, Computer Engineering

December 2025

Index

1. Introduction

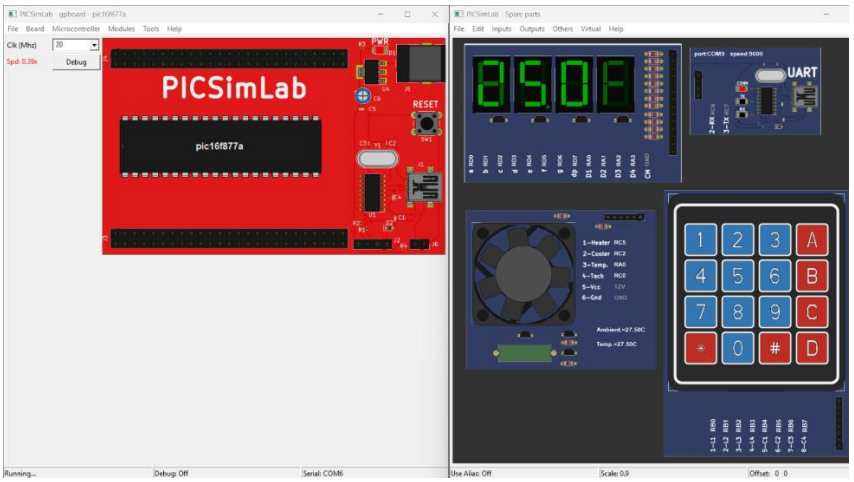
This project encompasses a comprehensive home automation system designed using a PIC16F877A microcontroller. The system manages system components automatically or manually through various hardware by analyzing environmental data and user preferences. You can access all the source codes from the following link: GitHub

2. Design

2.1 Board#1 Hardware Architecture and Pin Configuration

The design of Board #1 is built on the PICSimLab simulator using the "gpboard" architecture. The diagram on PIC16F877A of the hardware components is as follows:

- **LM35 Temperature Sensor (PORTA):** Connected to the RA0 (AN0) analog input pin to measure ambient temperature.
- **7-Segment Display (PORTB/D):** 4-digit common cathode display; it uses PORTD pins (RD0-RD7) for segment data (a-g, dp), and PORTA (RA1-RA3) along with PORTD pins for digit selection (D1-D4) by scanning (multiplexing).
- **Keypad (PORTB):** The 4x4 matrix keypad is connected to pins RB0-RB7 for row and column scanning operations.
- **UART IO (PORTC):** RC6 (TX) and RC7 (RX) pins are used for transferring system data to the PC interface and receiving remote commands. Data communication format: Serial communication is carried out at 9600 baud rate, 8-bit data, and 1 stop bit (8N1) format, and the COM9-COM10 pair is used.
- **Fan & Heater Actuators (PORTC):** For temperature control according to automation decisions; the Heater is connected to the RC5 pin and the Cooler is connected to the RC2 pin.

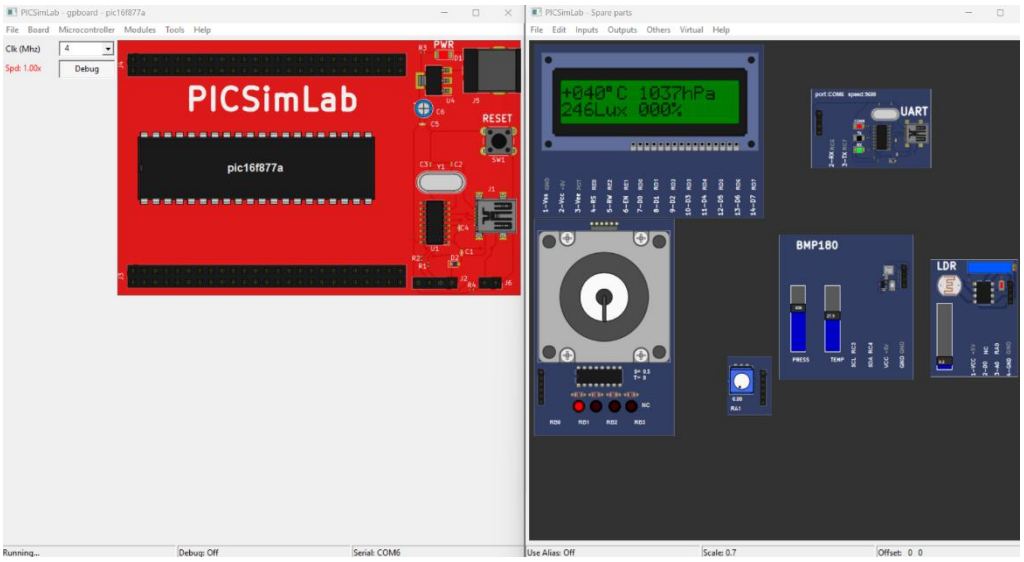


Board_1 Hardware Architecture

2.2 Board#2 Hardware Architecture and Pin Configuration

The design of Board#2 is structured using the "gpboard" architecture on the PICSimLab simulator, just like Board#1. This architecture offers a standardized hardware layer that allows peripherals to work harmoniously with the microcontroller. The processing of sensor data in the system and the control of the actuators are carried out in real time through the specific I/O ports defined on this structure. The wiring diagram on the PIC16F877A of the hardware components is as follows:

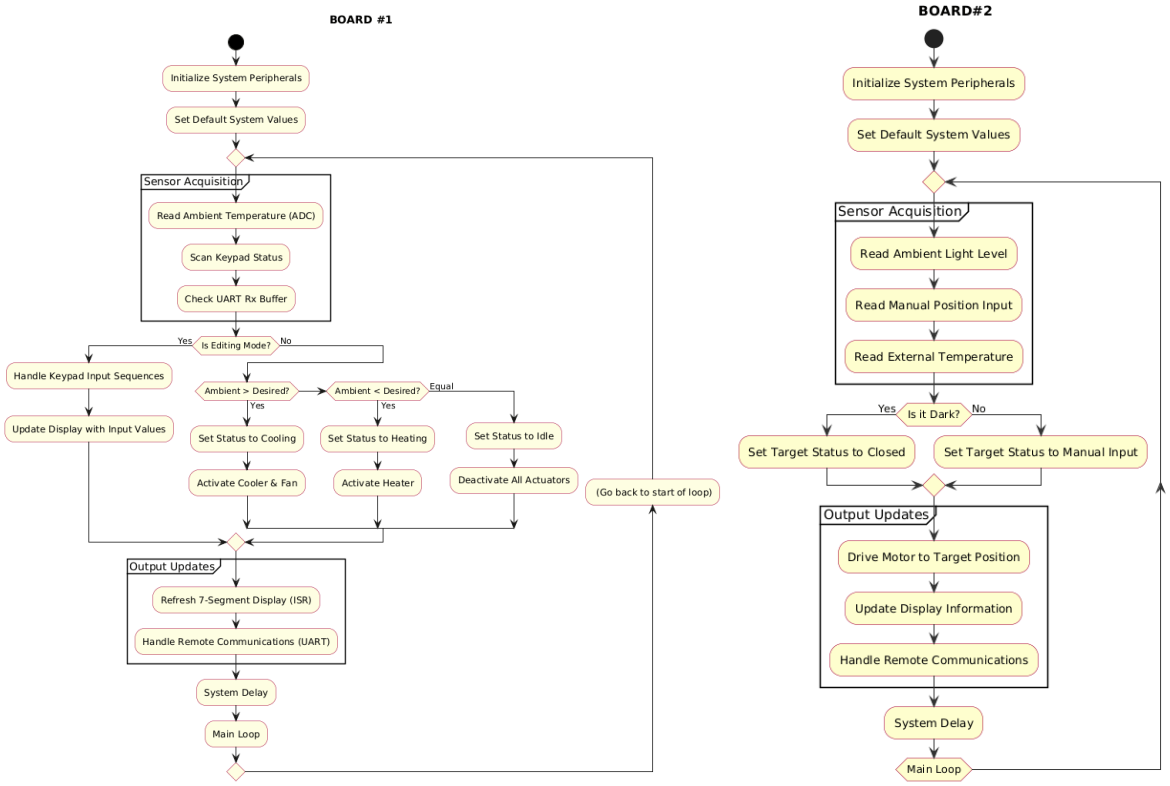
- **Stepper Motor (PORTB):** Controlled by ULN2003 driver via RB0-RB3 pins.
- **LDR & Potentiometer (PORTA):** RA0 (AN0) is connected to the ADC channel for light intensity and RA1 (AN1) for light position input.
- **BMP180 (PORTC):** RC3 (SCL) and RC4 (SDA) pins are used for the software I2C (bit-banging) protocol.
- **LCD Display (PORTD/E):** The 2x16 text display is connected to the RD0-RD7 (Data) and RE0-RE1 (Control) pins.
- **UART IO (PORTC):** RC6 (TX) and RC7 (RX) pins are used for serial communication between the microcontroller and the PC interface. Data transmission is carried out at a baud rate of 9600, 8-bit data and 1 stop bit format (8N1), and the COM6-COM7 pair is used.



Board_2 Hardware Integration

2.3 UML Software Flowchart

The working logic of the system consists of the stages of collecting data in an infinite loop, making automation decisions and updating the outputs. In this section, the general flow and decision mechanism of all components of the system is presented in vertical format.



3. Implementation Details

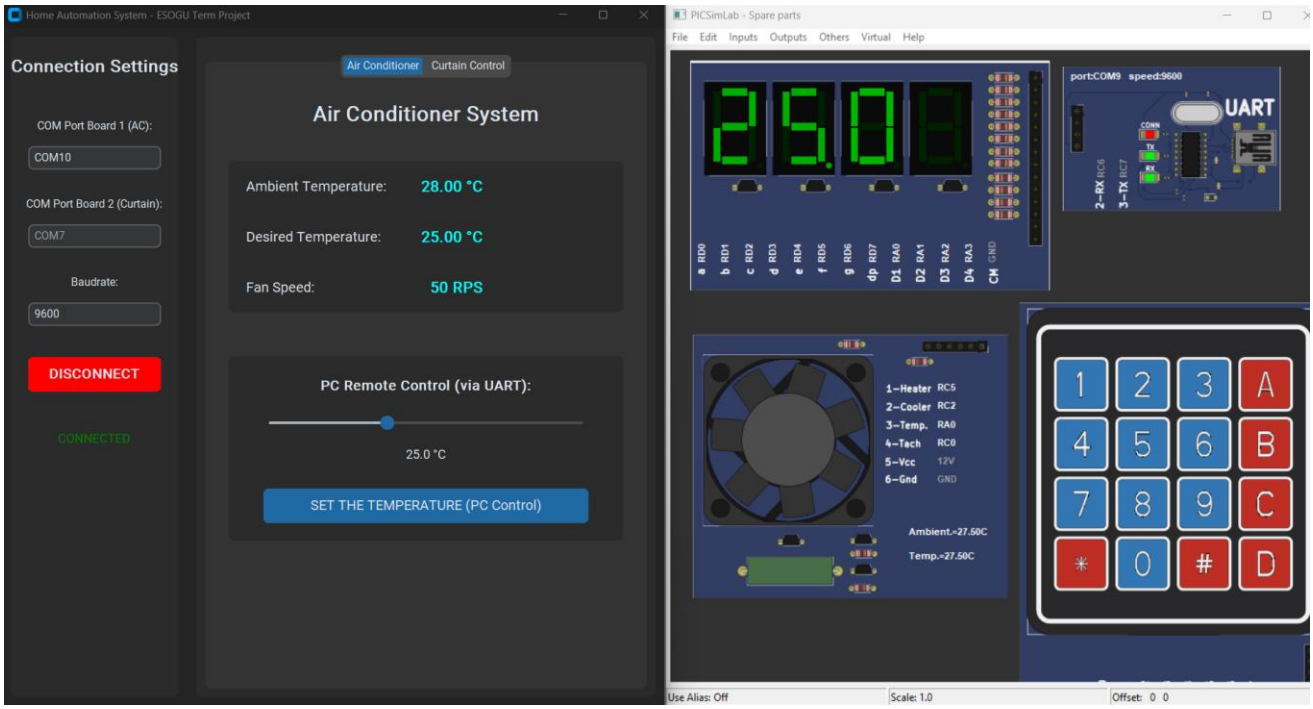
The implementation of the system has been designed in a modular structure in line with the specified requirements. All microcontroller codes for Board#1 are written in MPLAB X IDE v5.35, all microcontroller codes for Board#2 are written in Assembly language in MPLAB X IDE v6.25 environment, and directly generated .hex files can be used in simulation, testing and evaluation stages.

3.1 Board #1 Implementation (Home Air Conditioner System)

Board #1 is configured to manage the air conditioning processes of the home.

- **Temperature Control:** Ambient Temp is read periodically from the LM35 sensor on the RA0 pin. By comparing it with the target temperature value (Desired Temp) determined by the user; If the target temperature is higher than the environment, the fan is stopped and the heater is activated, otherwise the fan is turned on and the heater is turned off.
- **User Interaction (Keypad):** The user switches to target temperature input mode via interrupt by pressing the 'A' key. Entrances are limited to '10.0' to '50.0' degrees. The '*' key serves as a decimal separator, while the '#' key serves as an affirmation.

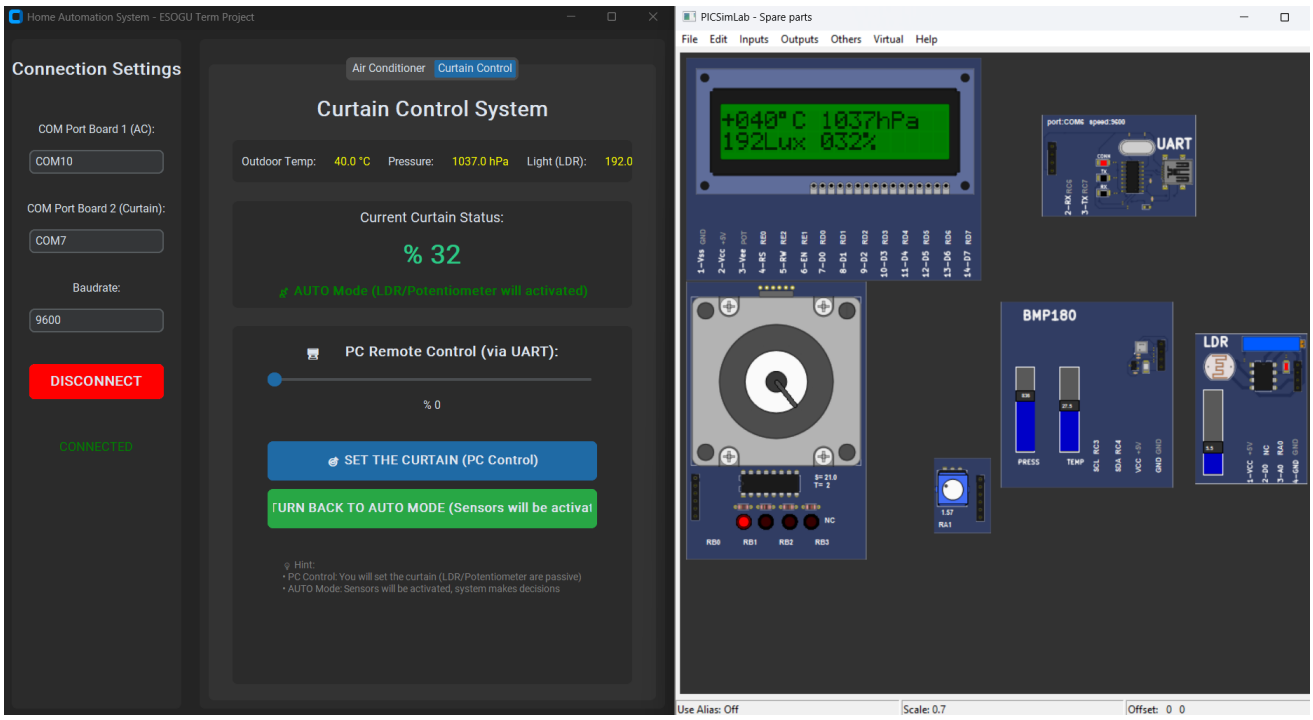
- **Visualization:** Target temperature, ambient temperature, and fan speed (rps) values are displayed on the 7-segment display alternately at 2-second intervals.
- **Serial Communication:** COM9-COM10 virtual port pair is used for data exchange with the PC interface. The data is transmitted at a rate of 9600 baud.



3.2 Board #2 Implementation (Curtain Control System)

Board #2 manages curtain automation based on light intensity and outdoor data.

- **Stepper Motor Control:** The curtain mechanism is modeled over 5 full turns (1000 steps). A clockwise (CW) rotation of the motor opens the curtain (towards the 0% position), while a counterclockwise (CCW) rotation closes the curtain (towards the 100% position). The system is calibrated to take 10 steps for every 1% change.
- **LCD Display:** On the 2x16 LCD screen, data is presented in the following format:
  - Line 1: +40C 1037hPa (Outside temperature and pressure)
  - Line 2: 220Lux 71% (Luminous intensity and curtain opening ratio)
- **BMP180 Sensor and I2C Technical Challenges:** The BMP180 sensor, which operates at 3.3V under normal conditions, operates in an overload state at the 5V level due to the lack of pull-up resistors in the simulation environment. In addition, since PICSimLab requires Bit-Banging I2C (software scheduling) instead of a hardware I2C (MSSP) module, the academic advisor was informed of the limitations encountered during the implementation of this protocol with Assembly.
- **Serial Communication:** PC connection for this board is made via COM6-COM7 virtual port pair.

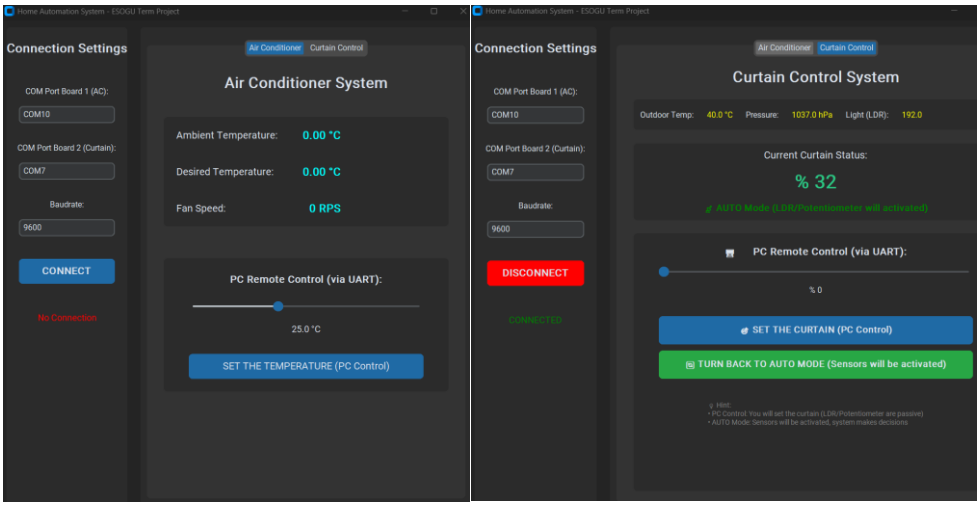


3.3 API and GUI Application

A Python-based API layer and GUI were developed using the 'CustomTkinter' library to provide a modern, central control hub. This interface transforms complex hardware commands into intuitive visuals while managing units through dedicated API classes.

1. **Autonomous Mode:** In this mode, the system operates based on sensor data (e.g., automatically closing curtains when LDR levels drop below a threshold). The GUI reflects these live changes, showing the system's independent decisions.
2. **Manual Mode:** Users can override autonomous logic by interacting with GUI components such as sliders and buttons. When a user sets a target temperature or curtain position via the interface, the API sends specific override commands to bypass sensor inputs and prioritize settings.

- **Data Synchronization:** The "Common Data" fields within the microcontrollers are kept in synchronization with the GUI elements. Through recurring update parameters such as curtainStatus is fetched and displayed on the dashboard , ensuring the user always sees the most current state.



System Interface Components

3.4 Special Implementation Notes for End User

To ensure the system operates correctly and to replicate the results presented in this report, strict adherence to the specified software versions is required. The project has been developed and tested in the following environment:

- **Simulation Software:** PICSimLab_0.9.2_241005_win64
- **API Environment:** python-3.14.2-amd64

Deployment Instructions: The compiled .hex files generated in MPLAB X IDE must be loaded directly into the microcontroller instances within PICSimLab. For the user interface to communicate effectively with the simulated ports, the Python API must be executed using the version specified above.

Virtual Serial Port Configuration: Since the communication between the PICSimLab simulation and the Python API relies on UART, a virtual null-modem emulator (specifically 'com0com') must be configured. The port pairs defined in the source code (COM6-COM7 for Board#2 and COM9-COM10 for Board#1) must be active and paired before launching the application.

Python Dependencies: The user interface requires specific third-party libraries to function properly, particularly for serial communication handling. Ensure that the 'pyserial' library and other dependencies listed in the 'requirements.txt' file are installed via pip prior to execution

Documentation & Support: A comprehensive demonstration video detailing the setup and usage steps is included in the submitted ZIP archive.

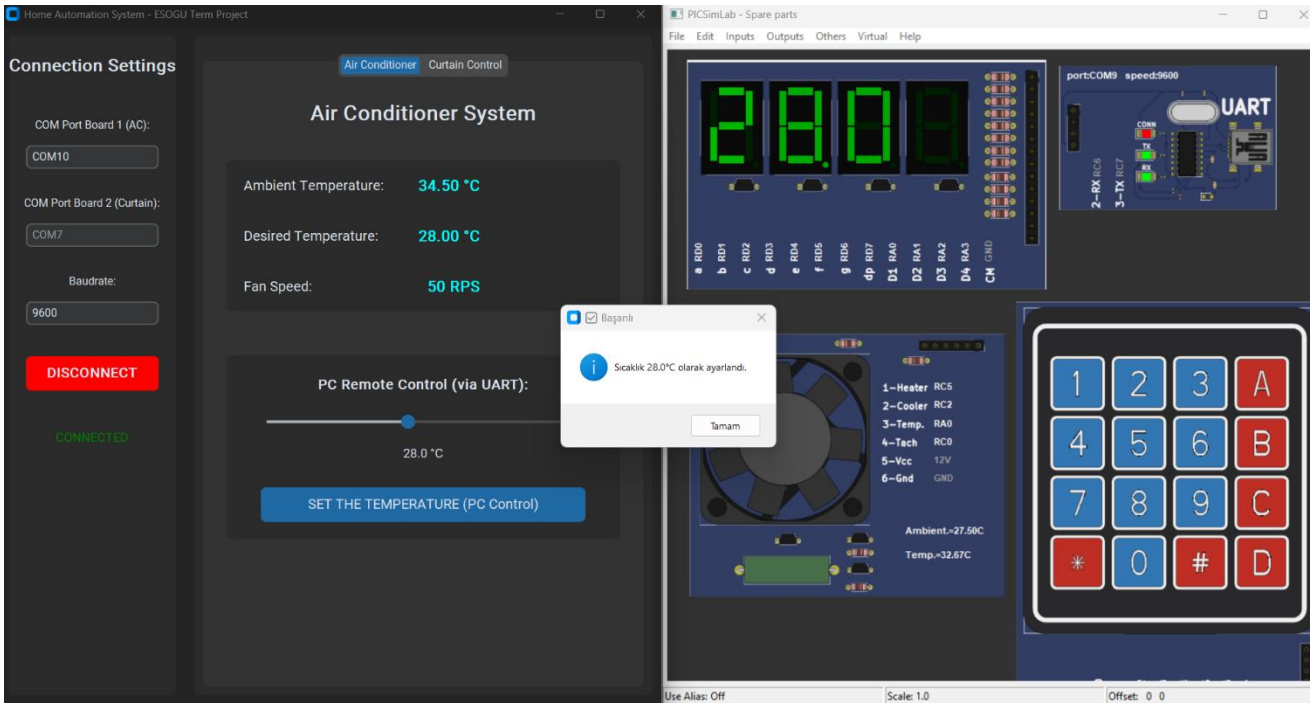
Compatibility Warning: Please note that due to potential differences in libraries and simulation behavior, trying to run this project on different versions of PICSimLab or Python may result in communication errors or system instability. We strongly recommend using the versions listed above to avoid compatibility issues.

3.5 Task Assignments

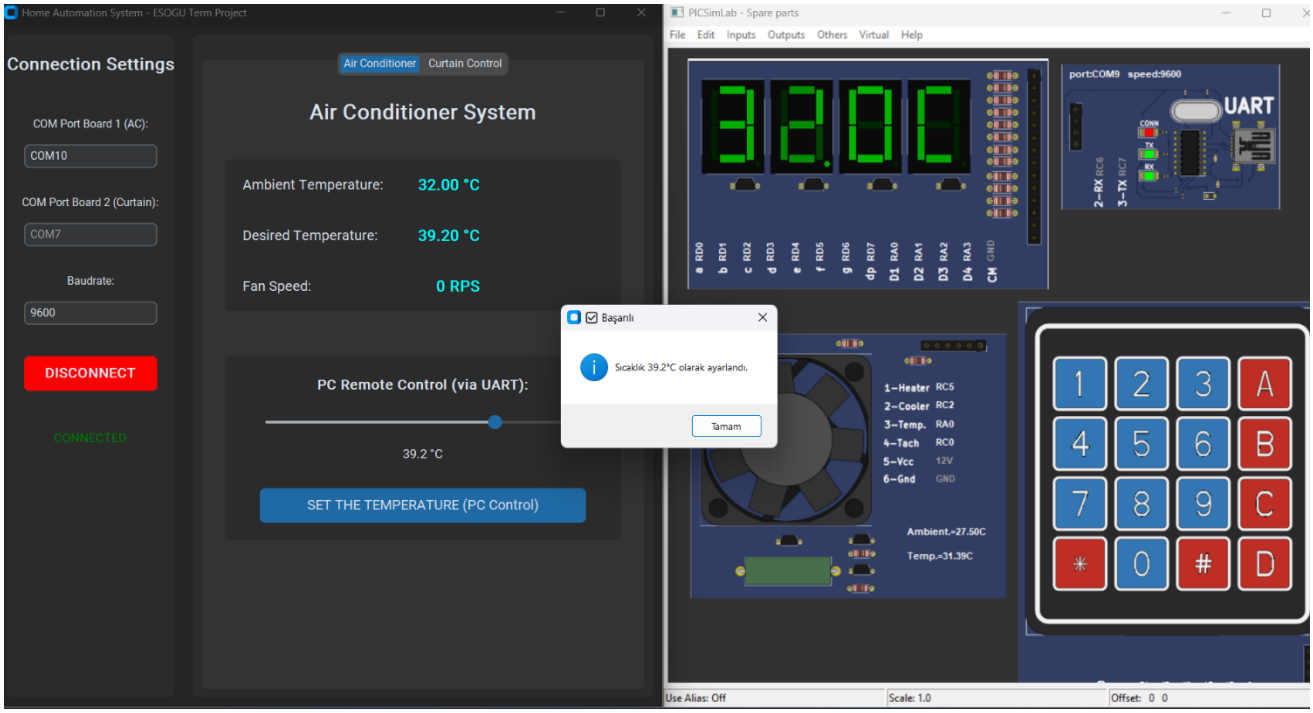
The distribution of tasks in the project process is stated in the table below:

Group Member	Tasks
Yiğit DOMBAYLI	7-Segment Display and UART Integration
Canan MUTLU	API Architectural Design (Board#1)
Tunahan ŞANAL	Hardware Integration (Board#2)
Nurefşan Ceren DOĞAN	API Architectural Design (Board#2)
Efe Duhan ALPAY	Keypad Integration
Yusuf İNAN	Heater & Cooler Integration

3.6 Sample Outputs

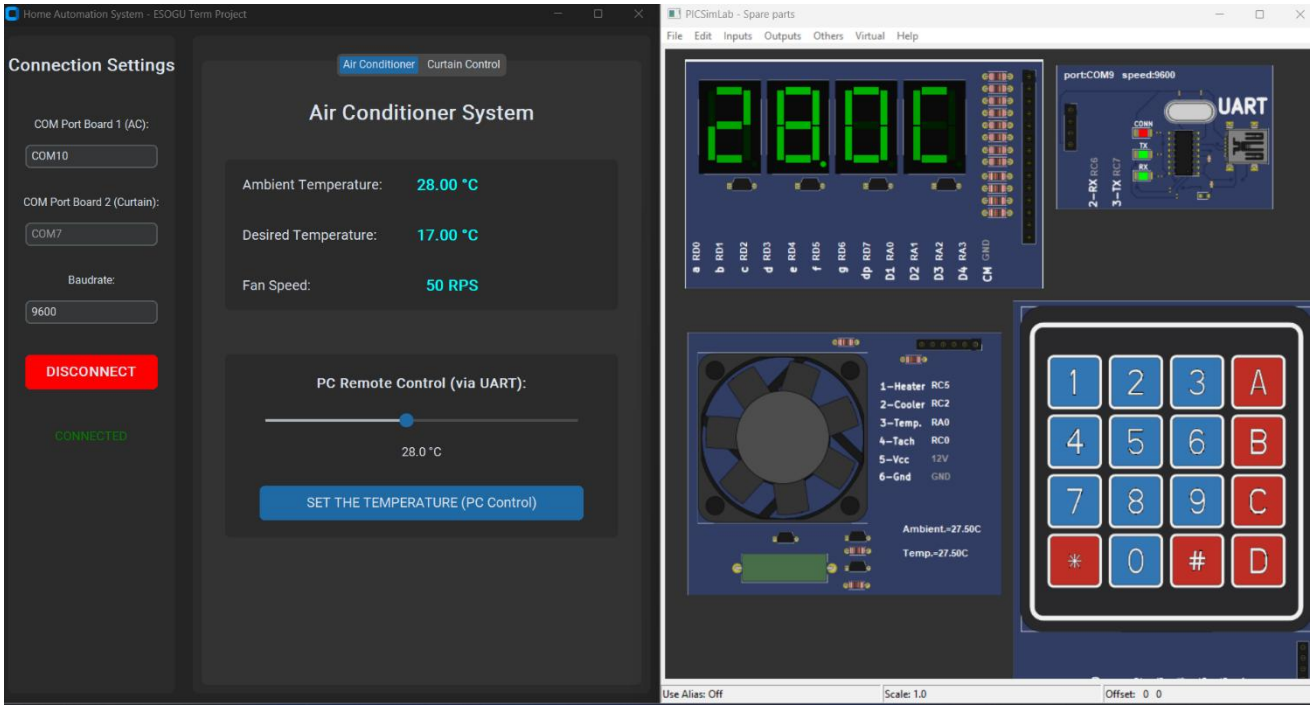


Scenario 1: Cooling Mode Activation via PC Interface Test Condition: The ambient temperature is 34.50 degrees Celsius. The user sets the desired temperature to 28.00 degrees Celsius using the PC interface slider and clicks the "SET THE TEMPERATURE" button. Expected Output: Since the ambient temperature is higher than the desired value, the system activates the Cooler/Fan (indicated by 50 RPS fan speed). The Python API displays a confirmation popup stating "Temperature set to 28.0 degrees Celsius".

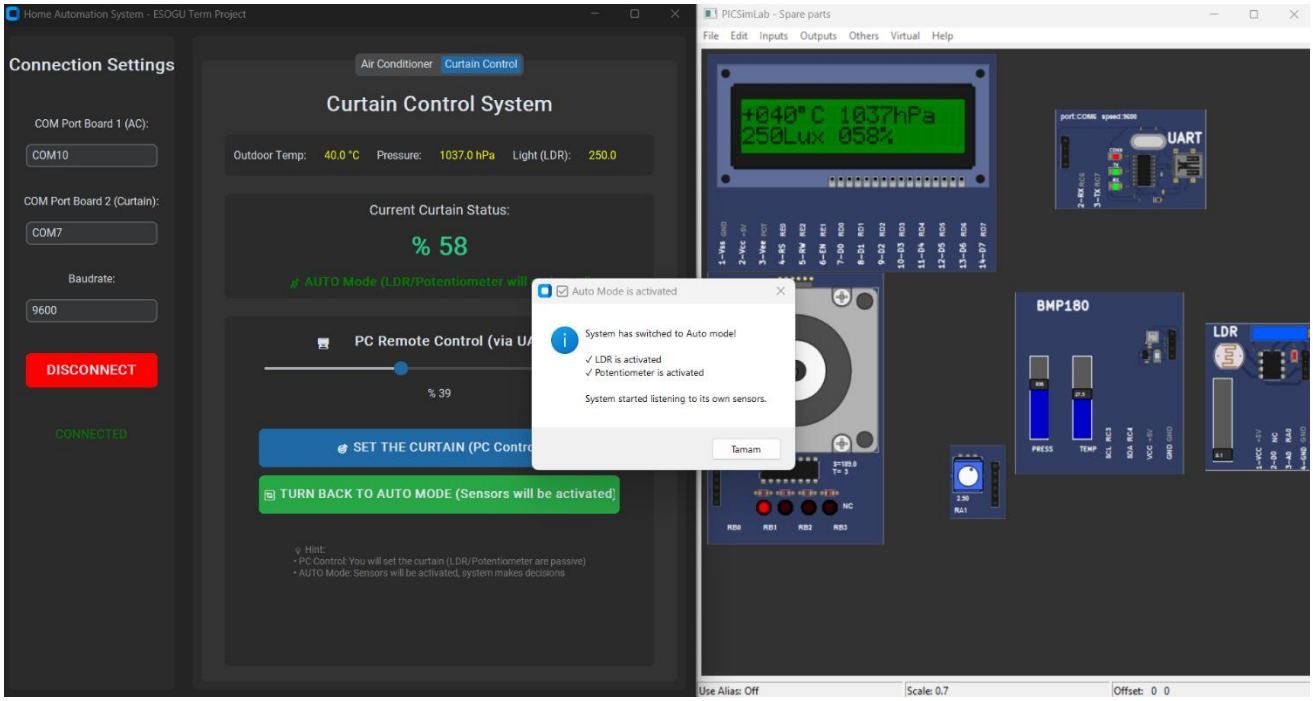


Scenario 2: Heating Mode Activation via PC Interface Test Condition: The ambient temperature is read as 32.00 degrees Celsius. The user sets a high target temperature of 39.20 degrees Celsius via

the PC interface. Expected Output: The system logic compares the values (Target > Ambient) and activates the Heater (RC5 LED is ON) while keeping the fan turned off (0 RPS). The 7-segment display on the simulation board updates to show the set value.

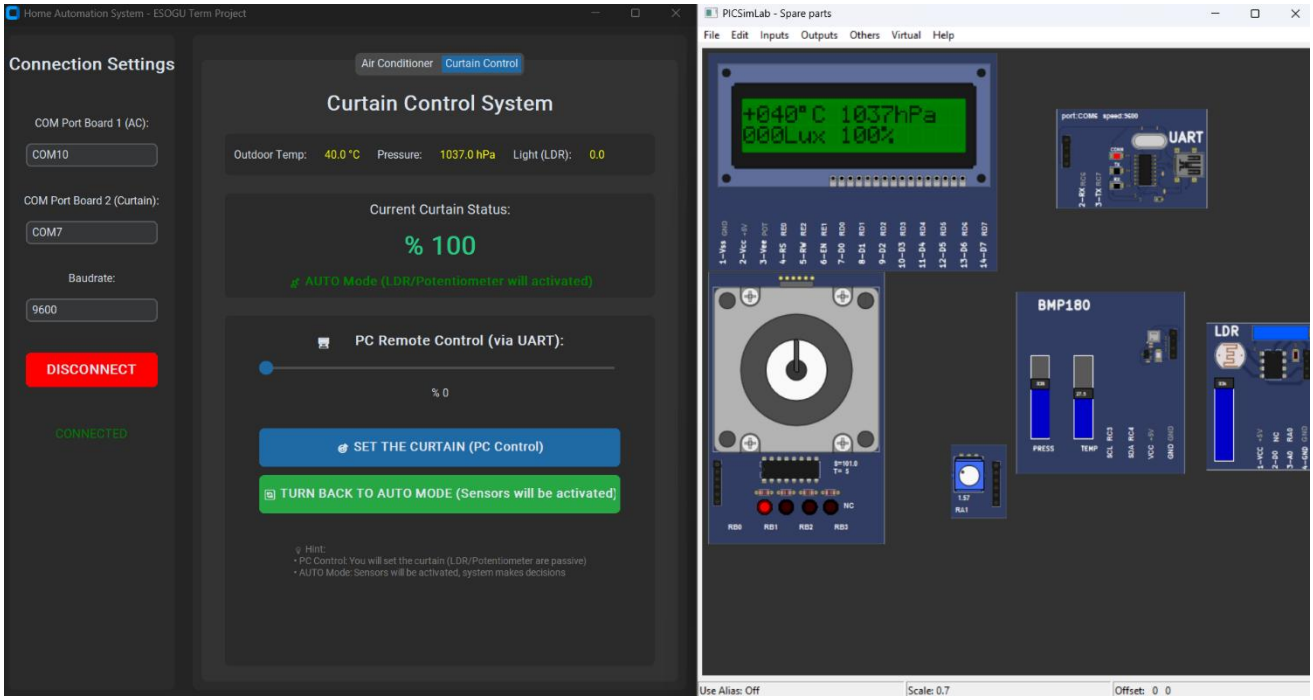


Scenario 3: Extreme Cooling Test via Keypad/Manual Input Test Condition: The target temperature is set to a low value of 17.00 degrees Celsius, while the ambient temperature remains at 28.00 degrees Celsius. Expected Output: The system detects a significant temperature difference requiring cooling. The Cooler/Fan is activated at maximum speed. The 7-segment display multiplexes to show the new target (17.00) and the current ambient temperature.

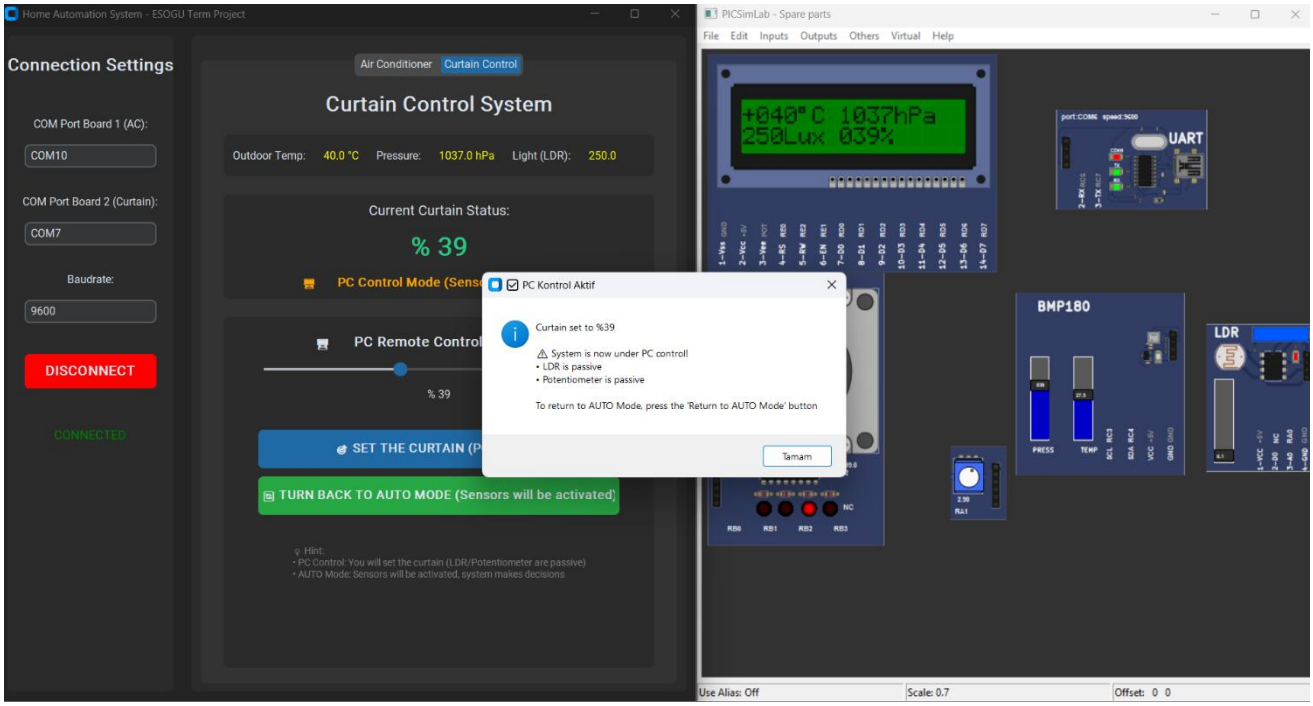


Scenario 4: Transition to Auto Mode (Sensor Activation) Test Condition: The user clicks the "TURN BACK TO AUTO MODE" button on the PC interface. Expected Output: The application

releases the manual override and re-activates the LDR and Potentiometer inputs. A confirmation popup appears stating "System has switched to Auto mode," indicating that the curtain position is now determined solely by environmental sensors.



Scenario 5: Night Mode / Low Light Cut-off Test Test Condition: The LDR sensor input (simulated via potentiometer) is reduced to 0.0 Lux (Voltage ~0V). Expected Output: The system identifies the condition as "Night Time" and automatically drives the curtain to the fully closed position (100%). The LCD screen updates instantly to show "000Lux" and "100%" status.



Scenario 6: Manual Curtain Control via PC Override Test Condition: The user manually drags the curtain slider to 39% on the PC interface. Expected Output: The system bypasses the sensor data

(LDR/Potentiometer are ignored) and locks the curtain to the specified position. The LCD screen displays "039%", and a popup warning confirms that the system is now under "PC Remote Control".

4 Conclusion

As a result, this project has successfully demonstrated that complex digital protocols and sophisticated real-time automation logic can be executed on the PIC16F877A microcontroller, despite its significantly constrained hardware resources. The primary objective was to bridge the gap between low-level hardware control and high-level user interfacing, and the final prototype stands as a testament to the efficiency of optimized Assembly programming. The project's success is deeply rooted in the disciplined distribution of tasks among team members. By developing the Keypad, 7-Segment Display, UART, and API modules simultaneously, we were able to transform individual components into a fully compatible and cohesive system architecture. This modular approach not only streamlined the development phase but also ensured that each subsystem was robust enough to handle its specific duty without interfering with the overall timing of the microcontroller. From a technical standpoint, the most significant achievement was the successful processing of BMP180 sensor data. Implementing I2C communication on a platform that lacks native hardware support for it required the development of a custom "Bit-Bang" I2C application in Assembly. Overcoming these hardware constraints to achieve accurate pressure and temperature readings represents the peak of our technical execution. Furthermore, the integration of a Python-based API to bridge the PIC16F877A with a modern PC interface proved that legacy hardware could still be a vital part of contemporary smart home ecosystems. During the teamwork process, we maintained a professional development workflow by utilizing GitHub for code version tracking. This allowed us to minimize integration errors and maintain a "single source of truth" for our modular code. We also encountered various simulation-specific challenges, such as voltage fluctuations and resistance limits within the virtual environment. Whether the system is operating in its autonomous mode—reacting to light levels via the LDR—or in manual mode via the PC interface, the motor positioning remains precise and reliable. This duality provides a versatile solution for modern environmental control. Looking ahead, this project serves as a foundational platform for more advanced iterations. Future studies aim to transition the system to a cloud-based IoT platform. By integrating ESP8266 modules for wireless connectivity and implementing PID control algorithms for smoother motor dynamics, the project could evolve from a localized automation tool into a globally accessible smart home solution. You can access all the source codes and technical documentation from the following link: https://github.com/TunahanSanal/MicroComputers_Term_Project_EEE_CENG.git