T.R.

ESKISEHIR OSMANGAZI UNIVERSITY DEPARTMENT OF ELECTRICAL-ELECTRONICS ENGINEERING

AND

DEPARTMENT OF COMPUTER ENGINEERING

INTRODUCTION TO MICROCOMPUTERS TERM PROJECT

HOME AUTOMATION

152120221055 BUĞRA AYRANCI COMPUTER ENGINNERING
152120231091 SALİH EREN COMPUTER ENGINNERING
151220182059 BORAN YILDIRIM ELECTRICAL-ELECTRONICS ENGINEERING
151220222094 ALPER ENES GÜNDÜZ ELECTRICAL-ELECTRONICS ENGINEERING

January 2026

Table of Context

1.Introduction ... 3
2.Design ... 4
  2.0 Task Assignments ... 4
2.1 Board #1 ... 4
  2.2. Hardware Architecture and Pin Connections ... 4
  2.3. Software Design and Module Logic ... 5
  2.3.1 Temperature Control Logic ... 5
  2.3.2 Keypad Interface ... 6
  2.3.3 Display Multiplexing ... 6
  2.3.4. Communication Protocol (UART) ... 7
2.4 Board #2: Smart Curtain Control System ... 8
  2.4.1 Hardware and Connection Architecture ... 8
  2.4.2 Software Algotrithm and Operational Logic ... 8
  2.4.3 Analog Reading ... 8
  2.4.4 Decision Mechanism ... 9
  2.4.5 Motor Driving Technique and Mechanical Ratio ... 9
  2.4.6 Communication and Display ... 10
  2.4.7 Safety And Error Management ... 12
2.5 API ... 13
  2.5.1 Software Architecture (API) ... 13
  2.5.2 Graphical User Interface (GUI) Design ... 14
  2.5.3. Main Menu ... 14
  2.5.4 Air Conditioner Control (Board #1) ... 14
  2.5.5 Curtain Control(Board #2) ... 15
3. Conclusion ... 16

1.Introduction

The objective of this project is to design and develop a comprehensive "Home Automation" system utilizing PIC microcontrollers. The system aims to control various home automation sensors and drivers, such as temperature regulation mechanisms and curtain control systems, while allowing users to monitor and manage these parameters via a personal computer (PC) interface.

The hardware architecture of the project is divided into two distinct subsystems, simulated using the PICSimLab environment. Board #1 is designated as the "Home Air Conditioner System". It manages peripherals including a heater, cooler, fan (tachometer), keypad, and a 7-segment display to regulate and visualize the ambient temperature based on user-defined setpoints. Board #2 functions as the "Curtain Control System". This subsystem utilizes a step motor, LDR light sensor, BMP180 pressure/temperature sensor, and an LCD screen to automate curtain movements in response to light intensity and manual inputs.

In terms of software architecture, the project requires a dual-layer approach. All control software running on the PIC16F877A microcontrollers is implemented using Assembly language to ensure precise low-level management of the peripherals. On the PC side, the custom Application Programming Interface (API) and the client application are developed using Python. This implementation facilitates Serial (UART) communication between the PC and the microcontrollers, ensuring seamless data exchange and system control. This report outlines the design, implementation, and testing phases of both the hardware simulation and the software modules required to achieve these functionalities.

2.Design

2.0 Task Assignments

Group Member	Tasks
Buğra Ayrancı	Board #1 UART,Temperature System,Report
Salih Eren	Board #1 7-Segment Display,Keyboard,Report
Boran Yıldırım	Board #2 Peripheral Configuration,Report
Alper Enes Gündüz	Board #2 Curtain Control System Design and UART Integration,Report

https://github.com/SalihEREN0/asm The link of github repository

2.1 Board #1

The design of Board #1 focuses on a Home Air Conditioner System controlled by a PIC16F877A microcontroller. The system is designed to monitor the ambient temperature, allow the user to set a desired temperature via a keypad, and automatically control a heater or cooler to maintain the desired temperature. Additionally, the system displays the current status on a 7-segment display and communicates with a PC interface via UART for remote monitoring and control.

2.2. Hardware Architecture and Pin Connections

The hardware design connects four main peripherals to the microcontroller: the Temperature System, the Keypad, the 7-Segment Display, and the UART module. The specific pin assignments used in the simulation are detailed in Table 1 below, complying with requirement.

Table 1: Pin Connections for Board #1 (PIC16F877A)

Module	**Component / Function**	**PIC Pin**	**Port**	**Type**
Temperature	LM35 Temperature Sensor	RA0	PORTA	Analog Input
System	Fan Tachometer	RA4	PORTA	Digital Input (T0CKI)
	Heater Control	RC4	PORTC	Digital Output
	Cooler (Fan) Control	RC5	PORTC	Digital Output
7-Segment	Segment 'a'	RD0	PORTD	Digital Output
Display	Segment 'b'	RD1	PORTD	Digital Output
	Segment 'c'	RD2	PORTD	Digital Output
	Segment 'd'	RD3	PORTD	Digital Output
	Segment 'e'	RD4	PORTD	Digital Output
	Segment 'f'	RD5	PORTD	Digital Output
	Segment 'g'	RD6	PORTD	Digital Output
	Decimal Point (dp)	RD7	PORTD	Digital Output
	Digit 1 Enable (D1)	RC0	PORTC	Digital Output

Module	Component / Function	PIC Pin	Port	Type
	Digit 2 Enable (D2)	RC1	PORTC	Digital Output
	Digit 3 Enable (D3)	RC2	PORTC	Digital Output
	Digit 4 Enable (D4)	RC3	PORTC	Digital Output
Keypad	Row 1 (L1)	RB0	PORTB	Digital Input/Interrupt
	Row 2 (L2)	RB1	PORTB	Digital Input
	Row 3 (L3)	RB2	PORTB	Digital Input
	Row 4 (L4)	RB3	PORTB	Digital Input
	Col 1 (C1)	RB4	PORTB	Digital Output
	Col 2 (C2)	RB5	PORTB	Digital Output
	Col 3 (C3)	RB6	PORTB	Digital Output
	Col 4 (C4)	RB7	PORTB	Digital Output
UART	Transmit (TX)	RC6	PORTC	Serial Output
	Receive (RX)	RC7	PORTC	Serial Input

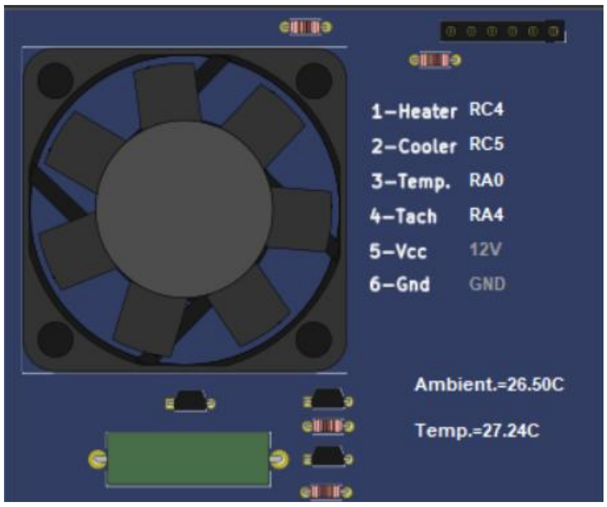
2.3. Software Design and Module Logic

The software is written in Assembly language and is structured into modular blocks to handle each peripheral independently.

2.3.1 Temperature Control Logic

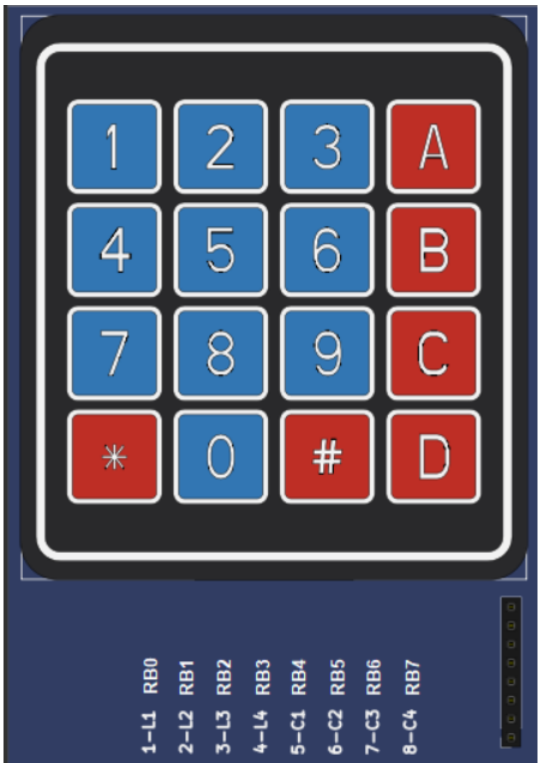
The system reads the analog value from the LM35 sensor via the ADC module on pin RA0. This value is converted to a temperature in Celsius. The control logic compares the Ambient Temperature with the Desired Temperature:

- If Desired Temp > Ambient Temp: The Heater (RC4) is turned ON and the Cooler (RC5) is turned OFF.
- If Desired Temp < Ambient Temp: The Heater (RC4) is turned OFF and the Cooler (RC5) is turned ON.
- The fan speed is calculated by counting pulses from the Tachometer (RA4) using Timer0.



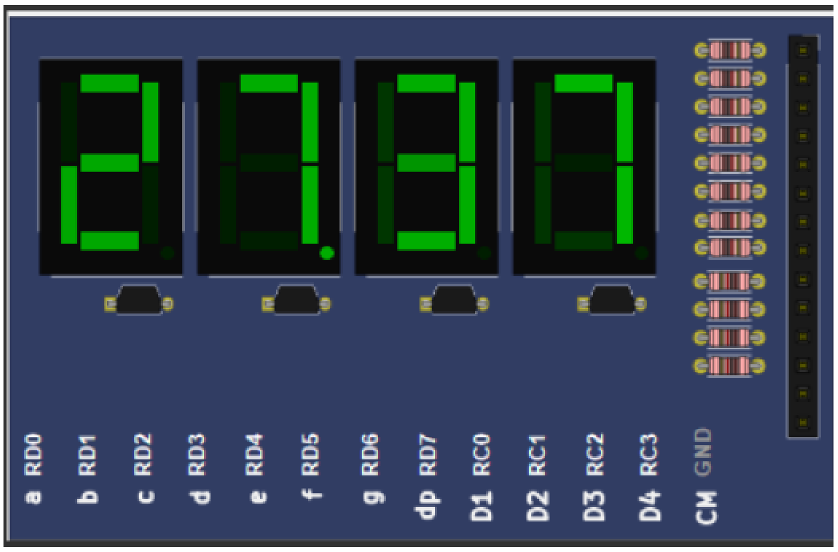
2.3.2 Keypad Interface

A 4x4 matrix keypad is used for user input. The system utilizes the External Interrupt (RB0) to detect when a key is pressed. The scanning algorithm activates columns (RB4-RB7) sequentially and reads the rows (RB0-RB3) to identify the specific button. The user enters a value followed by '#' to confirm. The system validates that the temperature is between 10.0°C and 50.0°C before saving it to memory.



2.3.3 Display Multiplexing

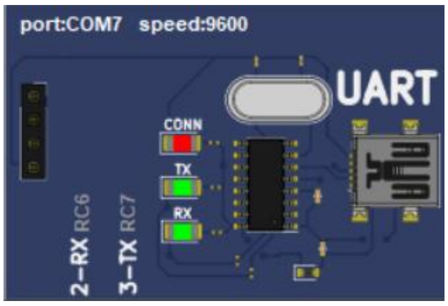
To drive the 4-digit 7-segment display using limited pins, a multiplexing technique is employed. The segments (PORTD) are set for a specific number, and the corresponding digit enable pin (RC0-RC3) is activated for a few milliseconds. This process is repeated rapidly for all four digits, creating the illusion of a continuous display. The screen cycles between showing Ambient Temperature, Desired Temperature, and Fan Speed.

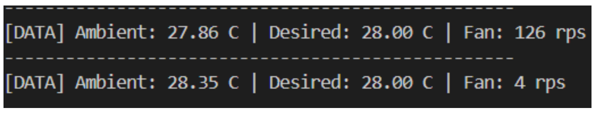


2.3.4. Communication Protocol (UART)

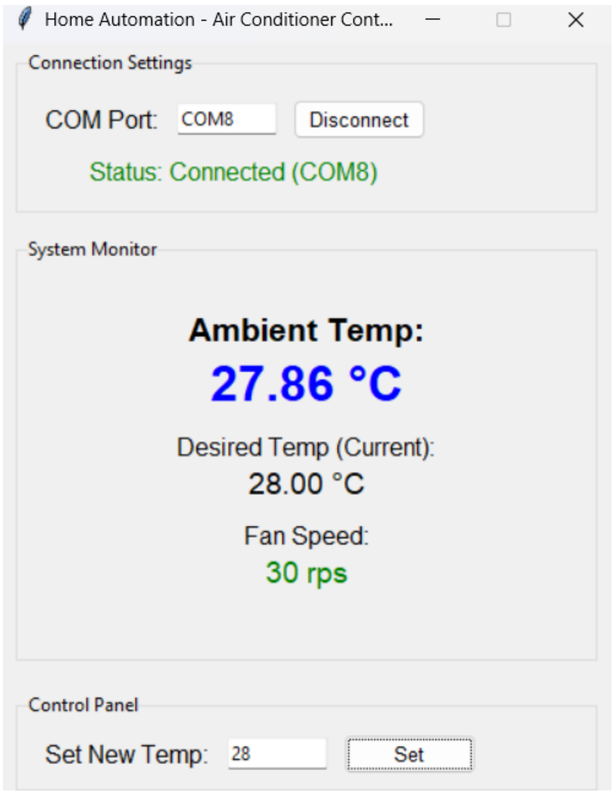
The system communicates with the PC Application using the UART protocol at 9600 baud rate. The communication is interrupt-driven to ensure no data is lost. The system responds to the command set defined in the requirements:

- 0x01 - 0x05 (GET Commands): The system sends the requested byte (Temperature Integer/Fraction or Fan Speed) back to the PC.
- 10xxxxxx / 11xxxxxx (SET Commands): The system parses the received byte to update the Desired Temperature (Fractional or Integer part) in the microcontroller's memory.





```
---------------------------------------------------
[DATA] Ambient: 27.86 C | Desired: 28.00 C | Fan: 126 rps
---------------------------------------------------
[DATA] Ambient: 28.35 C | Desired: 28.00 C | Fan: 4 rps
```



Home Automation - Air Conditioner Cont...

Connection Settings

COM Port: COM8 Disconnect

Status: Connected (COM8)

System Monitor

Ambient Temp:
27.86 °C

Desired Temp (Current):
28.00 °C

Fan Speed:
30 rps

Control Panel

Set New Temp: 28 Set

2.4 Board #2: Smart Curtain Control System

The objective of this part is to design a "Smart Curtain System" that moves automatically based on ambient light levels and user input using the PIC16F877A microcontroller. The system processes sensor data to drive a stepper motor, displays real-time status information on an LCD screen, and simultaneously transmits this data to a Personal Computer (PC) via the UART protocol.

2.4.1 Hardware and Connection Architecture

The components used in the system and their connections to the PIC16F877A microcontroller are as follows:

- Microcontroller: PIC16F877A (Operating with a 4MHz Crystal Oscillator).
- Input Units:
    1. LDR : Connected to channel AN0 (Pin 2). Measures ambient light intensity.
    2. Potentiometer: Connected to channel AN1 (Pin 3). Allows manual curtain adjustment by the user.
- Output Units:
    1. Stepper Motor: Unipolar driver circuit connected to PORTB (RB0-RB3) pins.
    2. LCD Display : Driven in 4-bit mode via PORTD.
    3. UART : Serial communication is established via RC6 (TX) and RC7 (RX) pins.

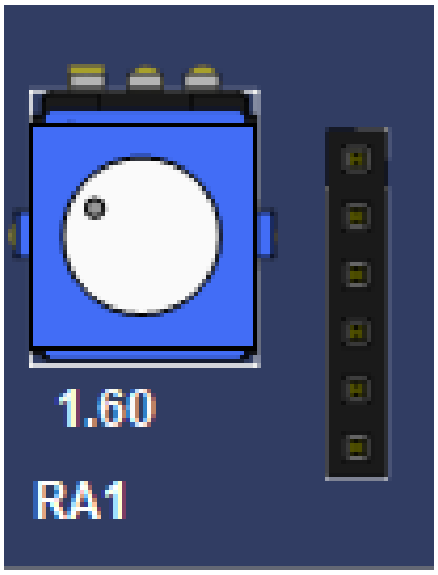
2.4.2 Software Algotrithm and Operational Logic

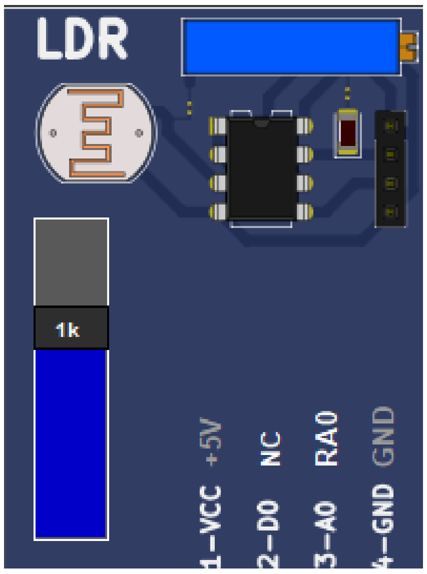
The system software is developed in Assembly language and executes the following steps within an infinite main loop:

2.4.3 Analog Reading

The system sequentially scans the AN0 and AN1 channels to obtain 8-bit digital data.

- LDR Value: Determines the ambient lighting condition .
- Potentiometer Value: Determines the target curtain position during manual mode.





2.4.4 Decision Mechanism

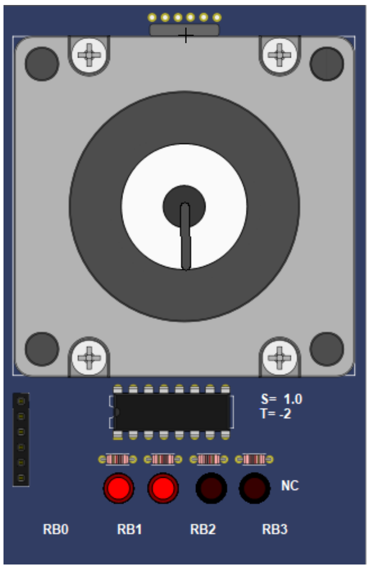
In accordance with project requirements, a threshold value is defined for the LDR sensor. The logic has been inverted to match specific hardware behavior:

- Night Mode : The environment is considered dark . In this state, the potentiometer value is ignored, and the curtain automatically moves to the 100% position.
- Day Mode : The environment is considered bright. Control is yielded to the user. The value read from the potentiometer is scaled to a range of 0-100% to determine the target position.

2.4.5 Motor Driving Technique and Mechanical Ratio

A specific stepping algorithm was developed for precise curtain control:

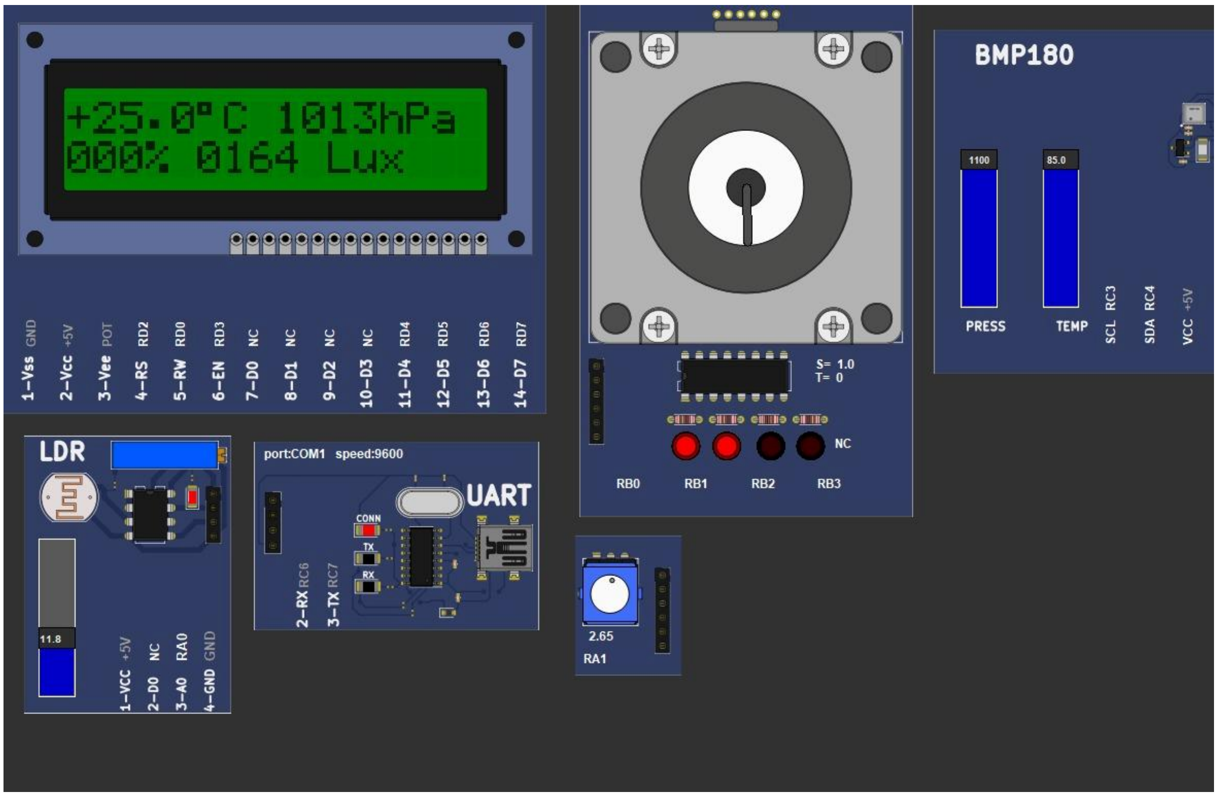
- The movement of the curtain from 0% to 100% corresponds physically to 5 Full Turns of the motor.
- Assuming a standard stepper motor takes 200 steps per turn, 5 turns require a total of 1000 steps.
- The software is programmed to take 10 steps for every 1% change (1000 steps / 100 units).
- Optimized delays are added between steps to ensure stable motor operation without stalling.



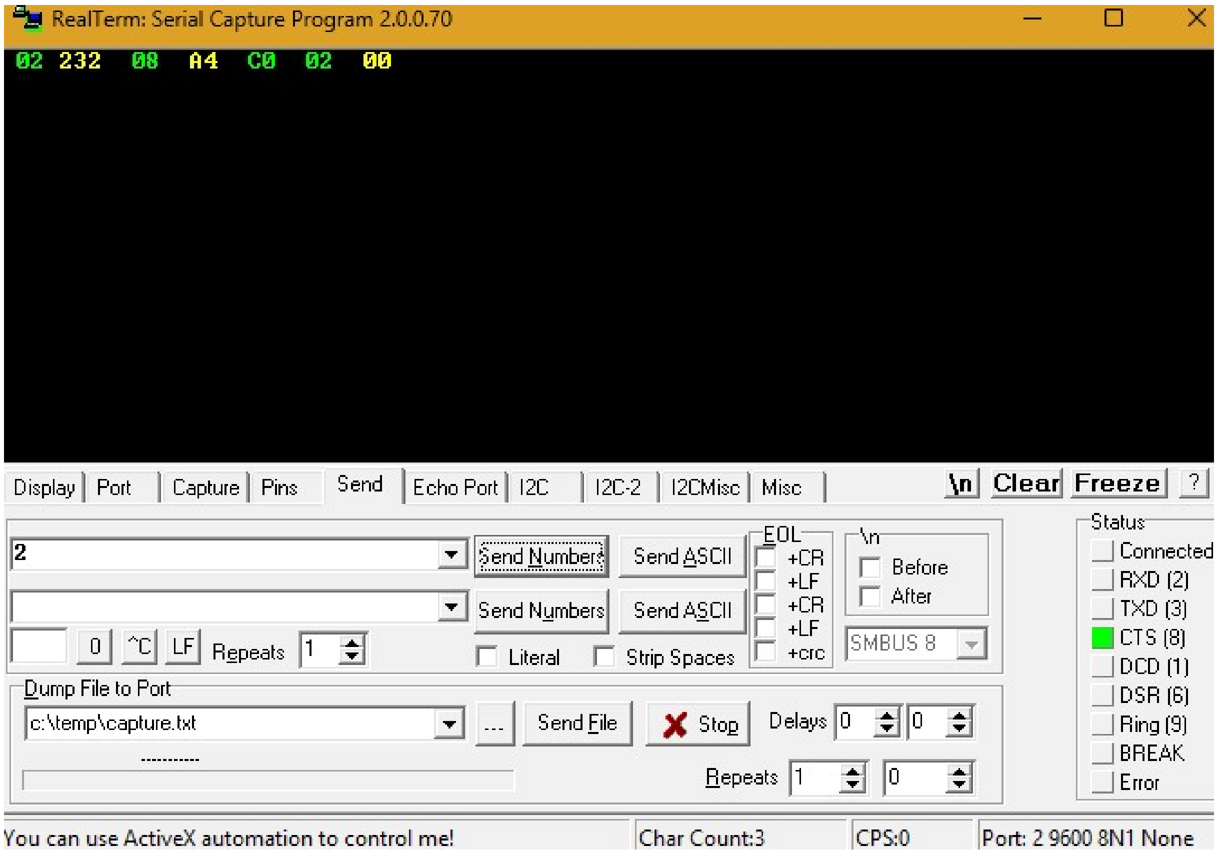
2.4.6 Communication and Display

The system status is presented to the user via two different interfaces:

1. LCD Screen:
    - Line 1: LDR: [Value]
    - Line 2: PERDE: %[Percentage]
2. UART (PC Interface):
    - Data is transmitted at a 9600 Baud Rate.
    - In every loop, the System OK status and current sensor data are transmitted to the PC terminal via UART and successfully observed using a serial monitoring application.



```
+25.0°C 1013hPa
000% 0164 Lux
```


RealTerm: Serial Capture Program 2.0.0.70

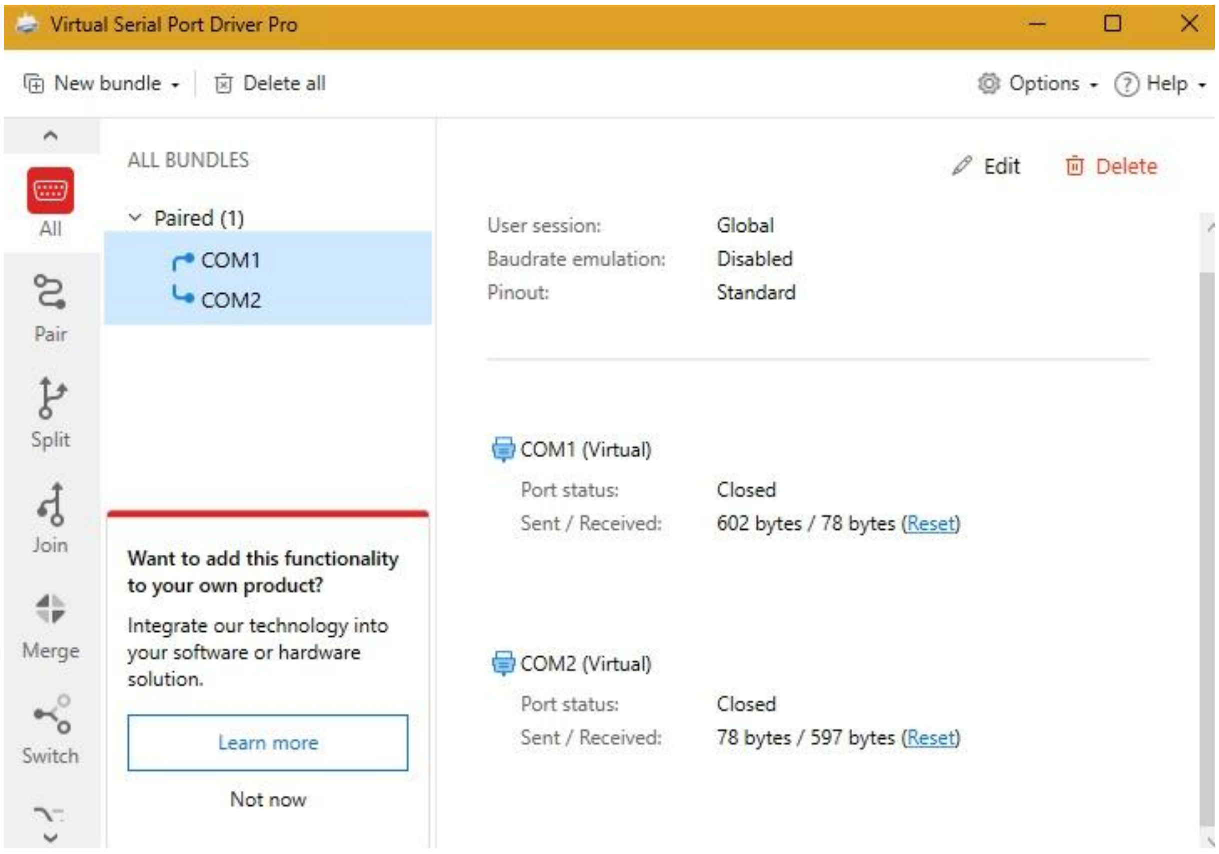
```
02 232  08  A4  C0  02  00
```

Send Numbers: 2

Dump File to Port: c:\temp\capture.txt

Status: CTS (8)

You can use ActiveX automation to control me! | Char Count:3 | CPS:0 | Port: 2 9600 8N1 None


Virtual Serial Port Driver Pro
New bundle
Delete all
Options
Help
All
Pair
Split
Join
Merge
Switch
ALL BUNDLES
Paired (1)
COM1
COM2
Edit
Delete
Want to add this functionality to your own product?
Integrate our technology into your software or hardware solution.
Learn more
Not now

User session: Global
Baudrate emulation: Disabled
Pinout: Standard

COM1 (Virtual)
Port status: Closed
Sent / Received: 602 bytes / 78 bytes (Reset)

COM2 (Virtual)
Port status: Closed
Sent / Received: 78 bytes / 597 bytes (Reset)

2.4.7 Safety And Error Management

To prevent system lockups (particularly during I2C communication), sensor readings are isolated from the main loop. Software precautions (Safe Boot) and startup delays have been implemented to ensure the system does not "freeze" during initialization. Upon startup, the system performs a self-test and begins operation with a "System OK" message.

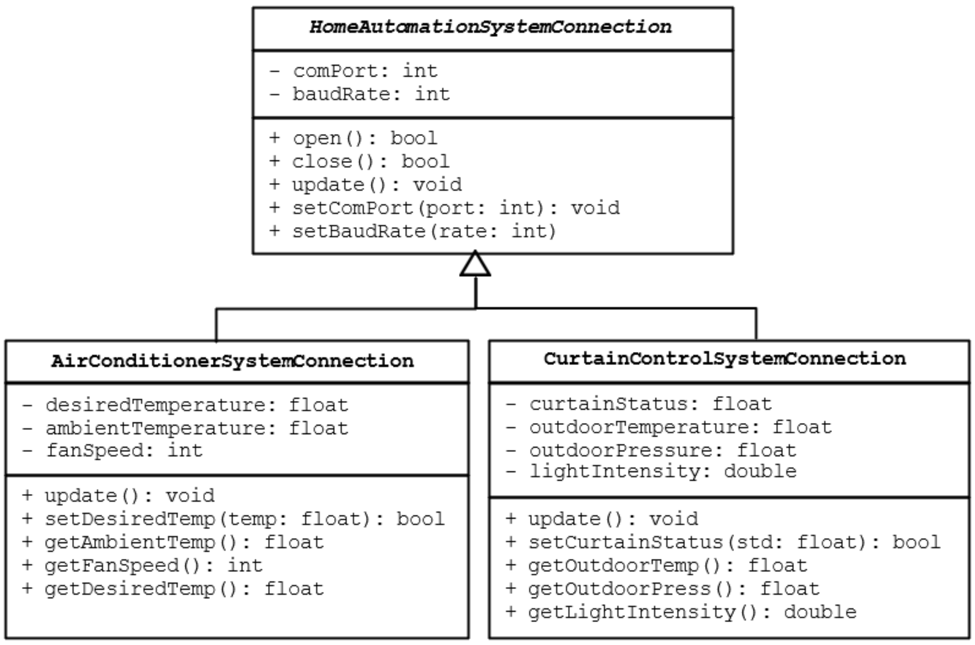
2.5 API

The PC-side application was developed to monitor and control the Home Automation System via a serial communication interface (UART). As defined in the project requirements, the application interacts with two distinct microcontroller boards: Board #1 (Air Conditioner System) and Board #2 (Curtain Control System) .
The application was built using the Python programming language, utilizing the tkinter library for the Graphical User Interface (GUI) and the pyserial library for handling UART communication.

2.5.1 Software Architecture (API)

The software is structured around an Application Programming Interface (API) layer that encapsulates the low-level serial communication details, adhering to the Object-Oriented Programming (OOP) principles outlined in the project design.

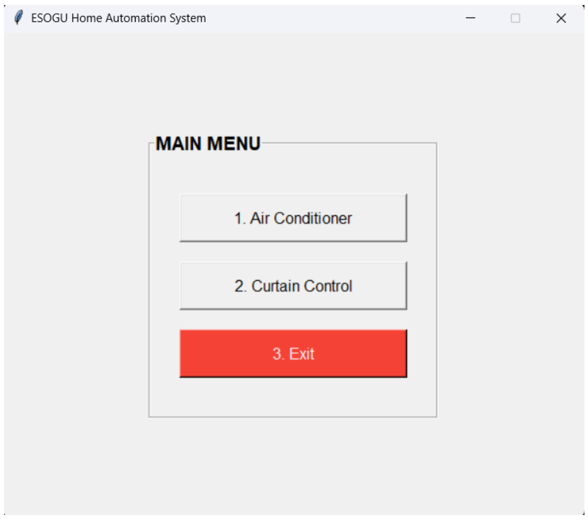
- Class Structure: Following the UML Class Diagram requirements, a base abstract class named HomeAutomationSystemConnection was implemented to handle common serial port operations (Open, Close, Send/Receive Bytes).
- Derived Classes: Two specific classes, AirConditionerSystemConnection and CurtainControlSystemConnection, inherit from the base class. These classes implement specific methods such as setDesiredTemp, getFanSpeed, and setCurtainStatus according to the byte-level communication protocols defined for each board.


HomeAutomationSystemConnection
- comPort: int
- baudRate: int
+ open(): bool
+ close(): bool
+ update(): void
+ setComPort(port: int): void
+ setBaudRate(rate: int)
AirConditionerSystemConnection
- desiredTemperature: float
- ambientTemperature: float
- fanSpeed: int
+ update(): void
+ setDesiredTemp(temp: float): bool
+ getAmbientTemp(): float
+ getFanSpeed(): int
+ getDesiredTemp(): float
CurtainControlSystemConnection
- curtainStatus: float
- outdoorTemperature: float
- outdoorPressure: float
- lightIntensity: double
+ update(): void
+ setCurtainStatus(std: float): bool
+ getOutdoorTemp(): float
+ getOutdoorPress(): float
+ getLightIntensity(): double

2.5.2 Graphical User Interface (GUI) Design

The user interface is designed to be intuitive and meets the menu operation requirements. The application flow consists of a Main Menu, specific monitoring screens for each board, and input dialogs for user control.

2.5.3. Main Menu

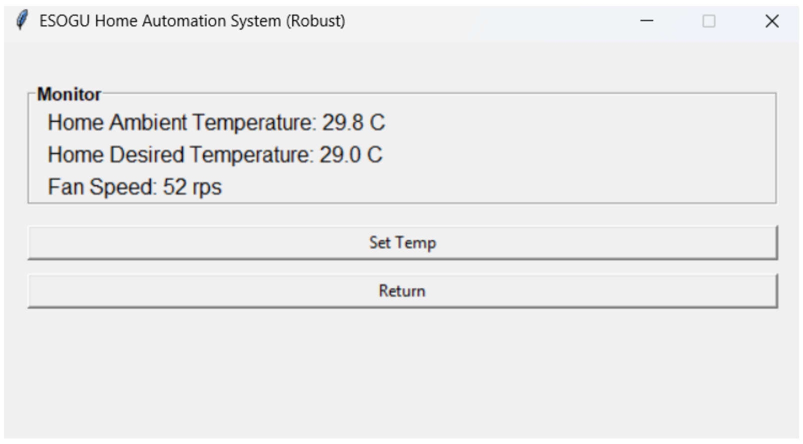


Upon launching the application, the user is presented with the Main Menu. This menu allows the user to select between the Air Conditioner system, the Curtain Control system, or to exit the application .

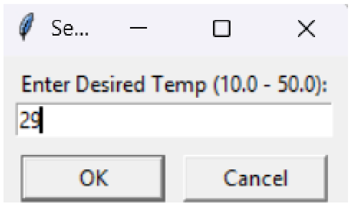
2.5.4 Air Conditioner Control (Board #1)

When the user selects "Air Conditioner," the application connects to the specified COM port for Board #1. The interface displays the following real-time data, updated every 2 seconds:

- Home Ambient Temperature: Read from the LM35 sensor.
- Home Desired Temperature: The target temperature set by the user.
- Fan Speed: The current speed of the cooler fan in rps.



Home Ambient Temperature: 29.8 C
Home Desired Temperature: 29.0 C
Fan Speed: 52 rps



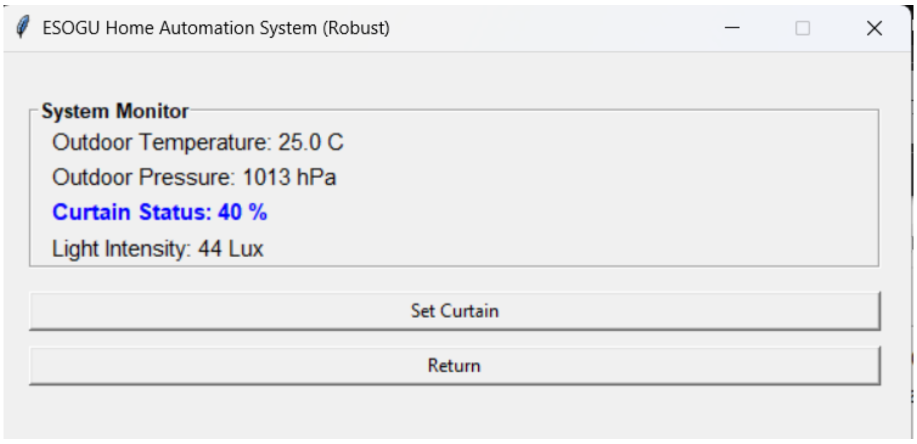
Enter Desired Temp (10.0 - 50.0):
29

As shown in Figure, the user can click "Enter the desired temperature" to send a new target temperature to the microcontroller.

2.5.5 Curtain Control(Board #2)

The Curtain Control interface monitors environmental data from Board #2. It displays:

- Outdoor Temperature & Pressure: Read from the BMP180 sensor .
- Curtain Status: The current openness percentage of the curtain (0% - 100%).
- Light Intensity: The ambient light level measured in Lux.


ESOGU Home Automation System (Robust)

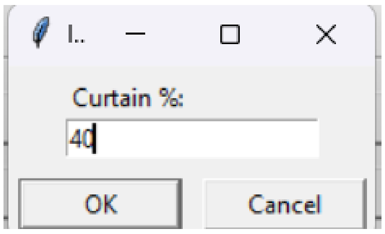
System Monitor

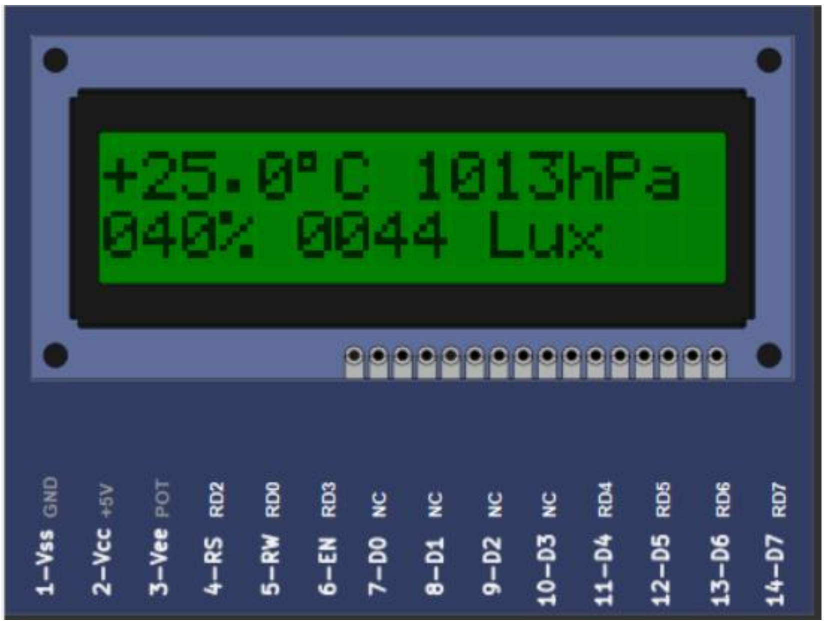
Outdoor Temperature: 25.0 C

Outdoor Pressure: 1013 hPa

Curtain Status: 40 %

Light Intensity: 44 Lux


Curtain %: 40


```
+25.0°C 1013hPa
040% 0044 Lux
```

As shown in Figure , a pop-up window allows the user to enter a specific **Curtain %** value. This command is then processed by the API to update the system's status and control the hardware driver.

3. Conclusion

The primary objective of this term project was to design, simulate, and implement a comprehensive Home Automation System using PIC16F877A microcontrollers. The project results indicate a successful realization of this goal, with both the Air Conditioner System (Board #1) and the Curtain Control System (Board #2) functioning as intended within the simulation environment. Board #1 effectively utilized the ADC module and external interrupts to manage temperature readings and user inputs via the keypad, providing real-time feedback through the multiplexed 7-segment display. Similarly, Board #2 demonstrated robust automation capabilities, with the stepper motor precisely tracking the rotary potentiometer inputs for manual curtain adjustments and automatically responding to light intensity changes detected by the LDR sensor. Crucially, the integration of both boards with the PC application via the UART protocol proved stable, allowing for error-free simultaneous data monitoring and remote control, satisfying all core system requirements defined in the project scope.

The development process was underpinned by a strong collaborative framework. Tasks were strategically distributed among group members based on individual strengths, covering distinct areas such as Assembly firmware development for the microcontrollers, API construction, and PC application design. To ensure seamless integration and code integrity, the team utilized GitHub as a version control system, which facilitated the tracking of modifications and the merging of code blocks. Regular meetings and active communication were essential in resolving synchronization issues, particularly regarding the serial communication protocols between the distinct hardware modules and the central computer. This structured teamwork not only accelerated the debugging process but also provided valuable practical experience in managing a multi-faceted engineering project.

Reflecting on the technical approach, the decision to use Assembly language presented both significant advantages and certain challenges. On the positive side, Assembly allowed for precise timing and granular control over the microcontroller's hardware resources, which was particularly beneficial for time-sensitive tasks like multiplexing the display and generating stepper motor pulse sequences. The modular design of the software further enhanced maintainability and debugging efficiency. However, the complexity of implementing mathematical operations, such as floating-point emulation, in a low-level language proved time-consuming and prone to logical errors compared to high-level languages like C. Additionally, while the simulation provided a safe testing ground, it naturally lacks the electrical noise and physical constraints of a real-world hardware deployment.