T.R

ESKISEHIR OSMANGAZI UNIVERSITY
DEPARTMENT OF ELECTRICAL-ELECTRONICS ENGINEERING

AND

DEPARTMENT OF COMPUTER ENGINEERING

INTRODUCTION TO MICROCOMPUTERS
TERM PROJECT

Smart Home System

Enes Buğra Damar 152120221068

Emir Cıvır 151220222102

Zeynep Pazarözyurt 151220232065

Nazlı Polat 151220212086

Ali Yılmaz 152120221108

January 2026

1. Introduction

This term project builds and simulates a two-board smart home automation system using the PIC16F877A microcontroller. The primary objective is to show how various embedded subsystems, constructed with low-level peripheral control in Assembly language, can cooperate to accomplish useful home automation tasks.

Two separate but communicating units make up the system. Board #1 serves as a module for controlling temperature. It uses an ADC, like an LM35 sensor, to read the surrounding temperature. A keypad allows the user to select the desired temperature. Based on this measured value, the system then regulates the heater or cooler. Board #2 functions as a module for controlling curtains. The second board acts as a curtain control unit by setting a target curtain position (e.g., utilizing an LDR and/or manual input) and controlling a stepper motor to achieve the required position. A UART-based communication protocol is devised and proven for linking these subsystems to outside control. A Python interface on the PC side is used for sending control commands (like setpoint or curtain target updates) and for monitoring real-time sensor values and system states. All features are tested in PICSimLab, which allows for witnessing the timing hierarchy (display refresh, motor stepping) and communication dependability. The report elaborates on the comprehensive architecture, the modules developed, the communication protocol, and the outcomes of the simulation tests.

2. Design

Board #1 is where the smart home system's temperature control and user interface module has been realized. It measures the ambient temperature using an LM35 sensor connected to the therefore it is converted and read through the PIC16F877A's ADC (RA0/AN0) and then compares this measured value with a user-defined setpoint which is entered through the 4×4 keypad. To avoid erroneous operation the firmware checks the entered setpoint and accepts only values within the range of 10.0–50.0 °C; all invalid inputs are rejected and the last setpoint is kept. Based on the result of the comparison, Board #1 controls the heating and cooling outputs (RC0 for heater, RC1 for cooler/fan), using a small hysteresis if necessary to prevent rapid switching. Meanwhile, a multiplexed 4-digit 7-segment display gives the real-time feedback by indicating important values like current temperature and setpoint, plus UART communication (RC6/RC7, 9600 bps) allows status reporting and command exchange with the other board and the external interface during PICSimLab simulation.

Temperature Control System module (LM35 + heater/cooler/fan block)

An LM35-type analog source is connected to RA0/AN0, and the 10-bit ADC of the PIC is employed for the transformation of the surrounding temperature. The control decision involves comparing the user-defined setpoint with the actual temperature. The digital control lines are employed to imitate the heater output (RC0) and cooler/fan output (RC1) of the actuator.

This block was beneficial to test the primary function of the thermostat and demonstrate the correct on/off control (optional with hysteresis).

4-digit 7-segment display (multiplex operation)

A multiplexed 7-segment module continuously shows key values (like temperature and setpoint). The selection of the digit lines (D1-D4) is done in a fast manner so that digit enabling can be performed while the segment patterns are fed by a common segment bus (usually PORTD RD0-RD7). In this project, the display confirmed correct timing and real-time behavior during the simulation.

4×4 keypad module (user input)

The keypad is used to enter the temperature setpoint. The firmware includes a basic debounce, scans the rows, and reads the columns (with pull-up logic), and then converts the key presses into numerical input. A validation system restricts input to setpoints in the range of 10.0–50.0 °C; erroneous inputs are discarded during the time the earlier valid target is active. This module is a reliable human-machine interaction in the embedded loop.

UART (serial communication)

The UART communication at 9600 baud (8N1 usual) is implemented using the hardware serial pins, RC6/TX and RC7/RX, of the PIC. UART, being the main channel of communication in the system, connects subsystems and/or an external interface and allows the interchange of commands and statuses which also helps in system debugging by monitoring sent values during simulation.

Module / Signal	PIC Pin (Port)	Direction	Description / Notes
Temperature Sensor (LM35)	RA0 / AN0	Input (ADC)	Ambient temperature input to ADC (10-bit).
Heater Control	RC0	Output	Heater ON/OFF control line (digital).
Cooler/Fan Control	RC1	Output	Cooler/Fan ON/OFF control line (digital).
Fan Tachometer (pulse)	RA4	Input	Tach/pulse feedback input (speed measurement if implemented).
UART TX	RC6 (typical)	Output	UART transmit to PC/UART module (e.g., 9600 bps).
UART RX	RC7 (typical)	Input	UART receive from PC/UART module.
7-Segment Segments (a–g, dp)	PORTD (RD0–RD7) (common)	Output	Segment lines driven by lookup table.
7-Segment Digit Select (D1–D4)	PORTA/PORTB pins (project-specific)	Output	Multiplex digit enable lines (4 digits).
Keypad Rows	PORTB pins (project-specific)	Output	Row scanning (one row active at a time).
Keypad Columns	PORTB pins (project-specific)	Input	Column read with pull-ups enabled.
Clock	4 MHz	—	Simulation clock setting (PICSimLab).
Actuator Supply	12 V (module)	—	Fan/actuator block supply shown in PICSimLab module.

Table 1 Board#1 Pin Assignments

The ADC temperature input, heater/cooler outputs, multiplexed 7-segment connections, keypad scan lines, and UART pins are all included in this table that highlights the pin mapping of Board #1 used in PICSimLab.
The mapping offers a clear reference for firmware initialization (TRIS/PORT configuration) and guarantees that there are no peripheral conflicts.

Test ID	Scenario	Steps	Expected Result
T1	Valid setpoint entry	Enter 25.0 and confirm (#)	Setpoint becomes 25.0
T2	Lower bound check	Try 9.9 and confirm	Rejected (invalid message)
T3	Upper bound check	Try 50.1 and confirm	Rejected (invalid message)
T4	Heater activation	Set 30.0; ambient < set	Heater ON, Cooler OFF
T5	Cooler activation	Set 20.0; ambient > set	Cooler ON, Heater OFF
T6	Stable region / hysteresis	Ambient near setpoint	No rapid toggling
T7	7-seg update	Observe display during run	No visible flicker, correct digits
T8	UART monitoring	Python requests temp/set	Correct bytes received

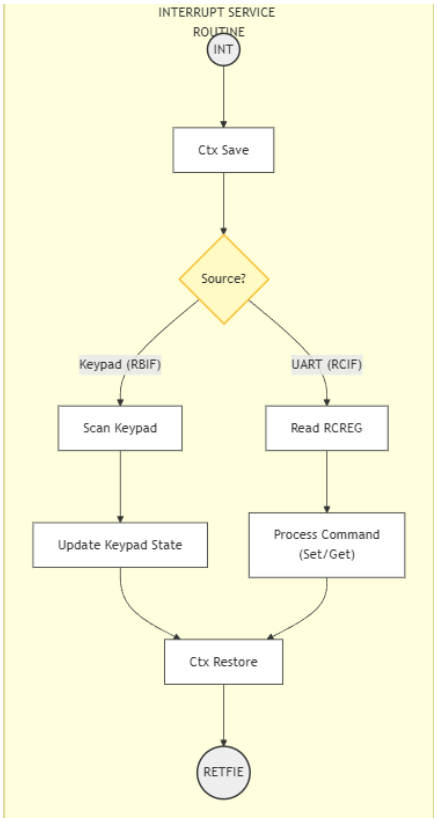
Table 2 Board#1 Test Cases

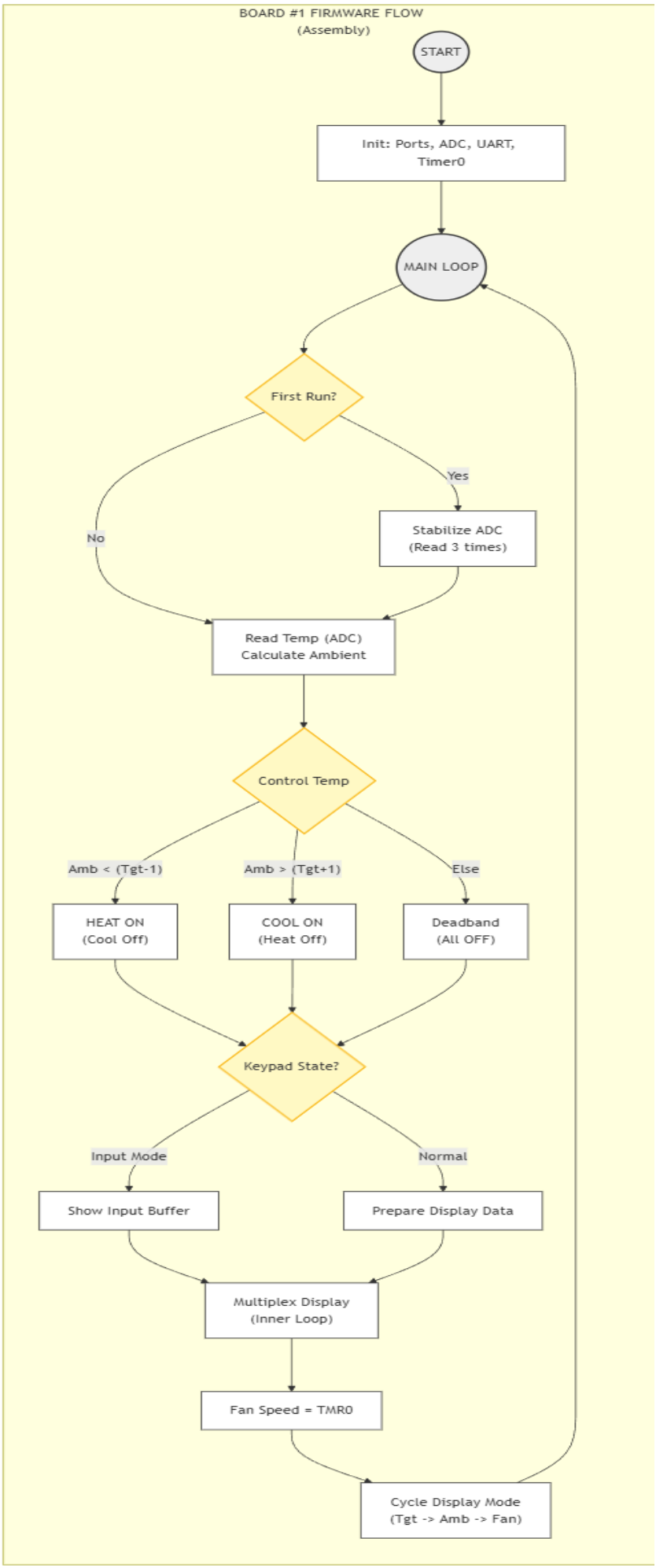
The verification scenarios carried out for Board #1 are shown in this table, which includes display updates, heater/cooler switching behavior, and valid/invalid setpoint entry (10–50°C).
To verify proper functionality, each test case is connected to simulation evidence (screenshots).

Command	Hex / Format	Function	Direction	Response
GET_TARGET_FRAC	0x01	Read Target Temp (Decimal Part)	PC -> PIC	1 Byte
GET_TARGET_INT	0x02	Read Target Temp (Integer Part)	PC -> PIC	1 Byte
GET_AMBIENT_FRAC	0x03	Read Ambient Temp (Decimal Part)	PC -> PIC	1 Byte
GET_AMBIENT_INT	0x04	Read Ambient Temp (Integer Part)	PC -> PIC	1 Byte
GET_FAN_SPEED	0x05	Read Fan Speed	PC -> PIC	1 Byte
SET_CMD (Flag)	Bit 7 = 1	Command to Set Temp (Internal Flag)	PC -> PIC	None
SET_TEMP_INT	11xxxxxx	Set Target Temp (Integer Part)	PC -> PIC	None
SET_TEMP_FRAC	10xxxxxx	Set Target Temp (Decimal Part)	PC -> PIC	None

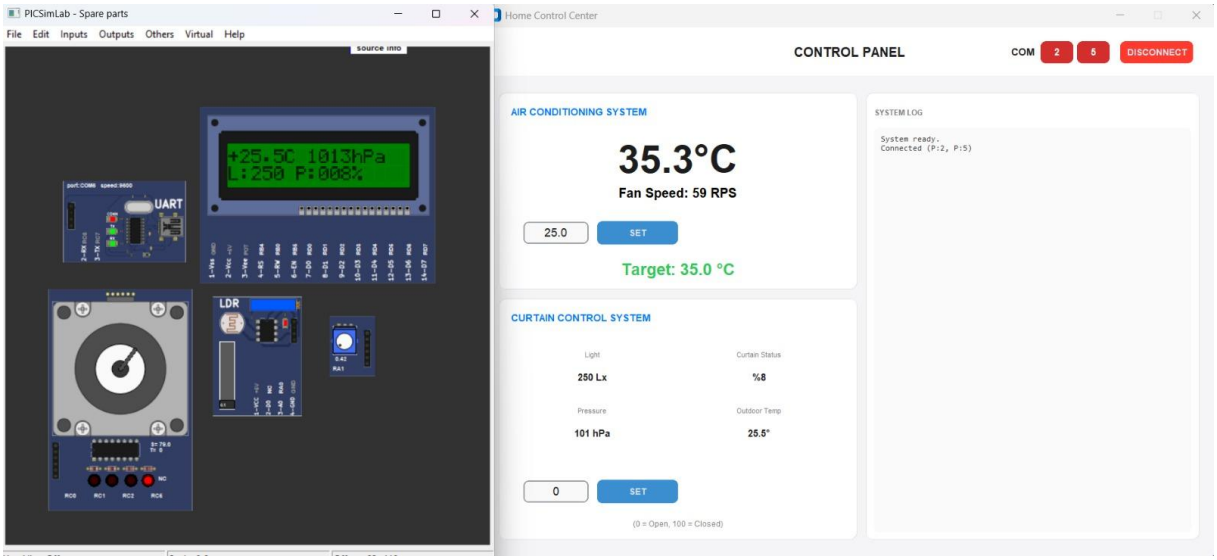
Table 3 Inter-Board UART Protocol (Template)

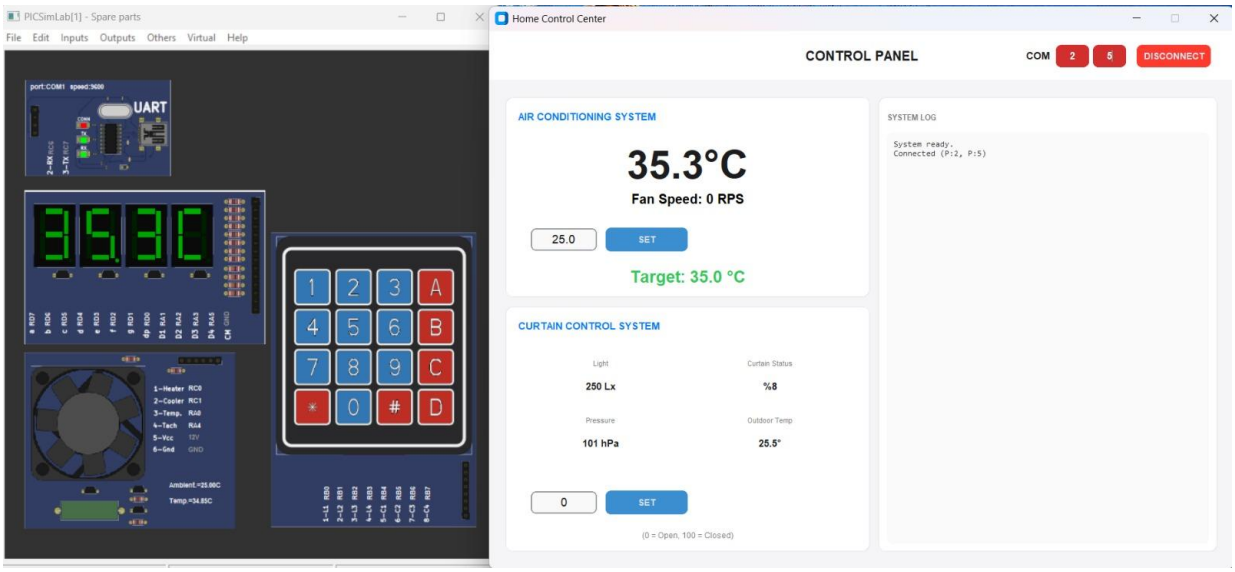
The UART instructions and anticipated answers used for Board #1 communication at 9600 bps are listed in this table.
In order to facilitate reproducible testing and debugging during simulation, it specifies the transmis sion of temperature/setpoint data and control updates.

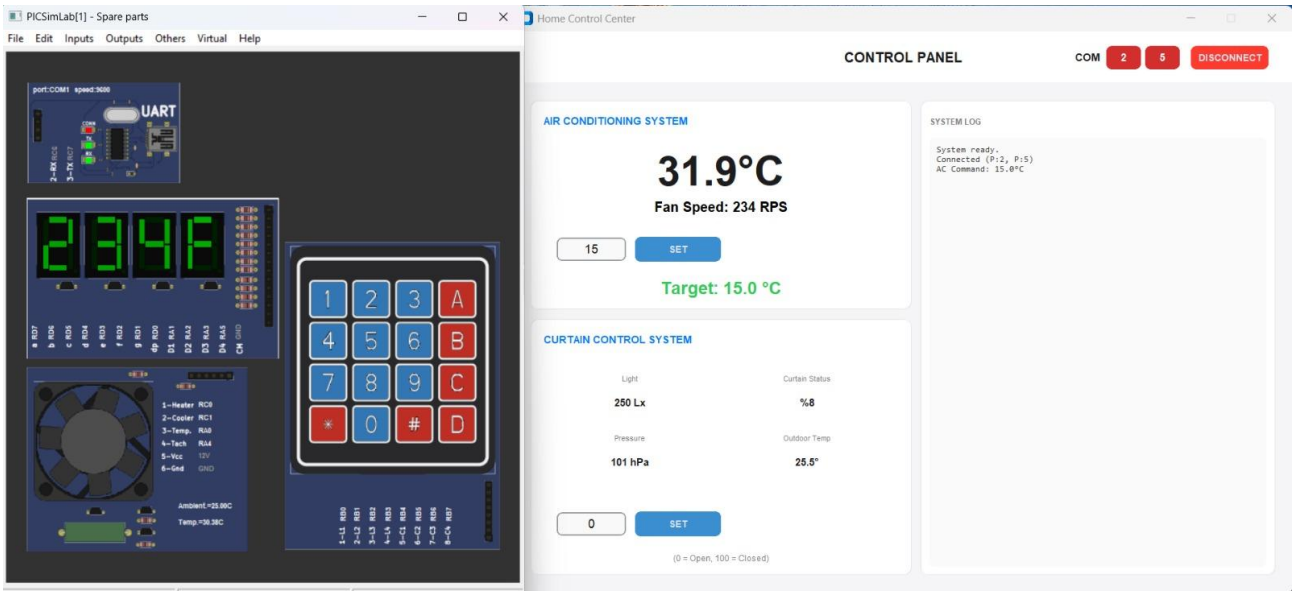



BOARD #1 FIRMWARE FLOW
(Assembly)
START
Init: Ports, ADC, UART, Timer0
MAIN LOOP
First Run?
Yes
No
Stabilize ADC (Read 3 times)
Read Temp (ADC) Calculate Ambient
Control Temp
Amb < (Tgt-1)
Amb > (Tgt+1)
Else
HEAT ON (Cool Off)
COOL ON (Heat Off)
Deadband (All OFF)
Keypad State?
Input Mode
Normal
Show Input Buffer
Prepare Display Data
Multiplex Display (Inner Loop)
Fan Speed = TMR0
Cycle Display Mode (Tgt -> Amb -> Fan)

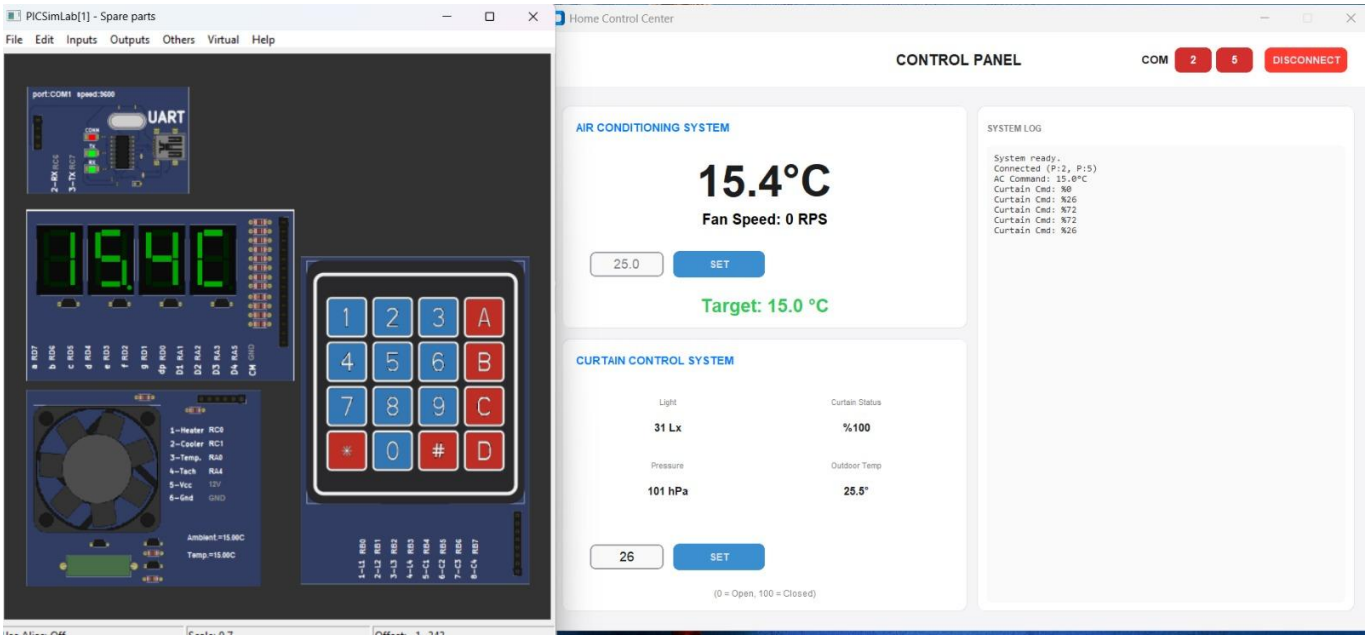
35 degrees entered from the user panel;



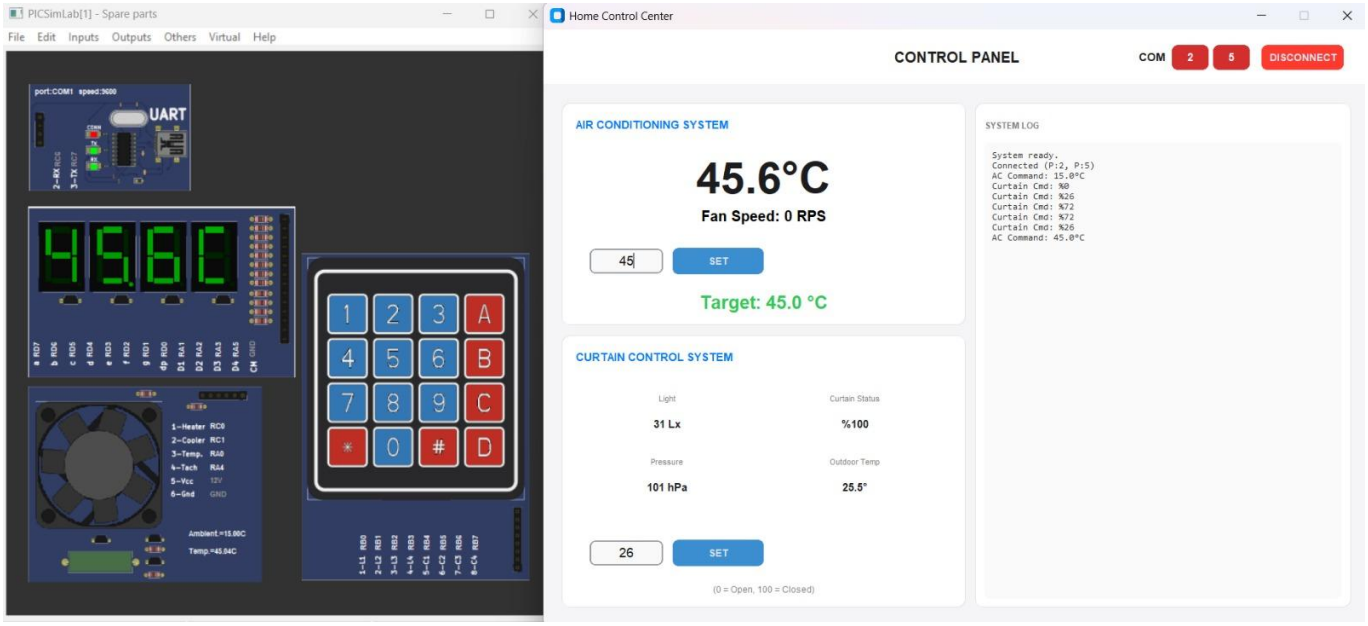




15 degrees entered from the user panel;



45 degrees entered from the user panel;



Curtain Control System Design

The second board acts as the controller for the curtains in the smart home system. It measures the light level of the environment with an LDR sensor and sets the amount of curtain opening either automatically (depending on the light intensity) or manually (based on user/PC commands). The microcontroller drives the motor in the correct phase sequence to position the curtain according to the opening value that has been converted into a number of steps for the stepper motor (for example, 0–100%). An LCD shows the most important data like the current light level, mode of operation (manual/auto), and current/target curtain position. Furthermore, during the simulation in PICSimLab, Board #2 is capable of exchanging commands and status data with Board #1 and the PC-side Python interface through UART at a speed of 9600 baud.

Module / Signal	PIC Pin (Port)	Direction	Description / Notes
LDR Sensor (Light)	ANx (e.g., RA0/AN0)	Input (ADC)	Reads light level (0–1023).
Stepper Coil A	RC0 (example)	Output	Phase control line 1 (depends on your wiring).
Stepper Coil B	RC1 (example)	Output	Phase control line 2.
Stepper Coil C	RC2 (example)	Output	Phase control line 3.
Stepper Coil D	RC5 (example)	Output	Phase control line 4.
LCD Data D4-D7	PORTB pins (project-specific)	Output	4-bit LCD interface.
LCD RS / EN	PORTB pins (project-specific)	Output	LCD control lines.
UART TX	RC6 (typical)	Output	UART transmit (e.g., 9600 bps).
UART RX	RC7 (typical)	Input	UART receive.

Table 4 Board#2 Pin Assignments

The pin allocation for Board #2 is displayed in this table, together with the LDR ADC input, UART communication lines, LCD interface pins, and stepper motor phase outputs.
The mapping serves as the foundation for low-level I/O configuration in Assembly as well as hardware simulation wiring.

Test ID	Scenario	Steps	Expected Result
B2-T1	Manual set 0%	Send target=0%	Curtain closes to 0%, position updates
B2-T2	Manual set 100%	Send target=100%	Curtain opens to 100%
B2-T3	Mid position	Send target=50%	Moves correct direction, stops near 50
B2-T4	Auto mode trigger	Change LDR value above/below threshold	Curtain reacts according to rule
B2-T5	Night mode	Hold LDR below threshold	Curtain closes to preset
B2-T6	LCD update	Run while moving	LCD shows mode + position correctly
B2-T7	UART status	Request status via Python	Correct bytes received

Table 5 Board#2 Test Cases

The functional tests for Board #2, including switching to 0/50/100% targets, responding to LDR thr eshold changes, LCD output accuracy, and UART command handling, are listed in this table.Throu gh anticipated results and screenshots, the table offers demonstrable proof for every requirement.

Stepper motor control (curtain actuator)

The movement of the curtain is indicated by a stepper motor. The firmware generates a phase sequence on four GPIO lines (for instance, RC0/RC1/RC2/RC5 according to the project wiring) to turn the motor in the right direction. The system operates the motor until the required position is reached during the process of changing a curtain "opening" request (e.g., 0–100%) into step counts. This section demonstrates phase sequencing, accurate positioning, and time-sensitive control of the actuator.

LDR light sensor module (automatic behavior input)

An LDR block gives a corresponding analog signal of the surrounding light intensity. The signal is then read via the ADC by the automation logic, and it is characterized in terms of percentage or level for a useful purpose. For example, the system can automatically close the curtain to a defined position when the light drops down to a defined threshold (for example, Night Mode). This module facilitates the "environment-driven" automation feature of the project.16×2 LCD module (system feedback for Board #2)

The measured light level, curtain status (current/target position), and current operation mode (manual/auto/night) are all shown on the LCD.

The system features an automatic Night Mode with the highest priority logic. The LDR sensor value (read via ADC Channel 0) is compared against a threshold (LDR_THRESHOLD = 80).

- If LDR < 80 (Darkness): The system clears the Override_Flag (disabling PC control) and forces Desired_Percent = 100 to close the curtains automatically.
- If LDR >= 80 (Daylight): The system checks for PC commands (Override_Flag). If no command is active, it defaults to Manual Mode, reading the Potentiometer value (ADC Channel 1) to set the curtain position.

UART (serial communication)

Board #2 has UART at 9600 baud to communicate status (e.g., current position, light level, m ode) and receive commands (e.g., target curtain %, mode changes).

This makes it possible to integrate with Board #1 and validate subsystem coordination and co mmunication at the system level.

Command	Hex / Format	Function	Direction	Response
GET_DESIRED_LOW	0x01	Read Target % (Low Byte: 0)	PC -> PIC	1 Byte
GET_DESIRED_HIGH	0x02	Read Target % (High Byte: Value)	PC -> PIC	1 Byte
GET_TEMP_LOW	0x03	Read Sim Temp (Low Byte: 0xFF)	PC -> PIC	1 Byte
GET_TEMP_HIGH	0x04	Read Sim Temp (High Byte: 0x00)	PC -> PIC	1 Byte
GET_PRESS_LOW	0x05	Read Sim Pressure (Low Byte: 3 -> 0.3)	PC -> PIC	1 Byte
GET_PRESS_HIGH	0x06	Read Sim Pressure (High Byte: 101)	PC -> PIC	1 Byte
GET_LDR_LOW	0x07	Read LDR (Low Byte: 0)	PC -> PIC	1 Byte
GET_LDR_HIGH	0x08	Read LDR (High Byte: Value)	PC -> PIC	1 Byte
SET_POSITION	1xxxxxxx (Bit 7 = 1)	Set Target Position % (Overrides Auto Mode)	PC -> PIC	None

Communication Protocol (Board #2)

Mode	Condition	Action	Notes
Manual mode	Target % is set by user/PC	Move to target position	UART command or local input
Auto mode (LDR)	Light level compared to threshold	Open/close to keep desired light	Threshold e.g., 40%
Night mode	Light < threshold for N samples	Close curtain to 0% (or preset)	Debounce/filter recommended
Safety/limit (optional)	Max step count reached	Stop motor, clamp position	Prevents overshoot

Table 6 Modes & Threshold Logic

The curtain system's three operating modes—manual, auto, and night—as well as the circumstances that activate each mode are compiled in this table. To make the automation behavior clear and testable, threshold-based decisions (such LDR %) are defined.

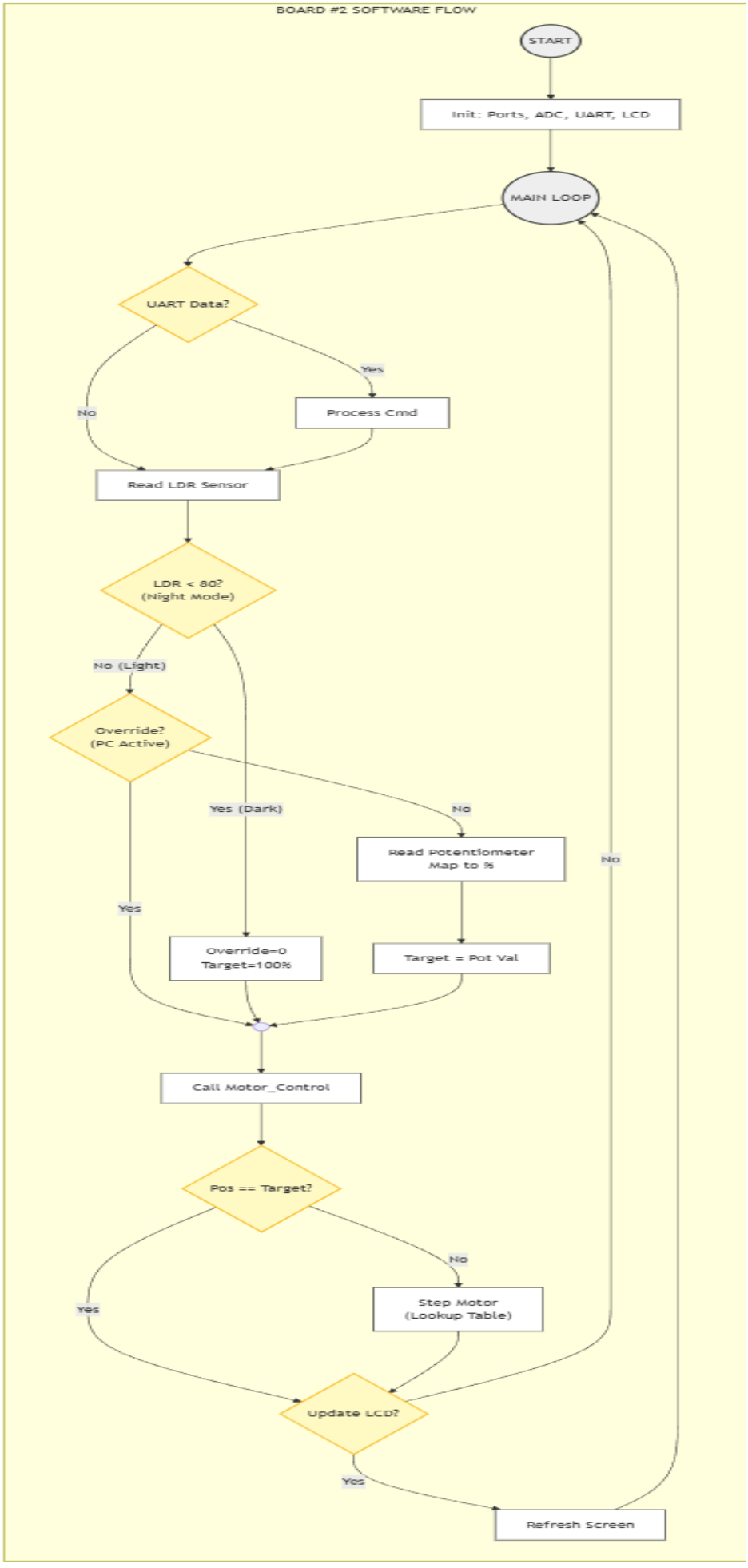
Item	Symbol / Equation	How it is obtained	Notes / Example
Reference (0%)	cur_steps = 0	Define "fully closed" as home position	Set at startup or after manual calibration
Full travel steps	STEPS_FULL	Count steps from 0% → 100% (one-time measurement)	Example: STEPS_FULL = 1000 steps
Steps per 1%	STEPS_PER_% ≈ STEPS_FULL / 100	Integer rounding (use division)	Example: 1000/100 = 10 steps/%
Target input	target_percent (0–100)	From UART/user command	Clamp to 0–100
Convert target	target_steps = (target_percent × STEPS_FULL) / 100	Compute desired absolute position in steps	Clamp 0…STEPS_FULL
Current position	cur_steps (cur_percent optional)	Track after each motor step	cur_percent = (cur_steps×100)/STEPS_FULL
Delta movement	delta = target_steps − cur_steps	Compute required movement	steps_to_move = |delta|
Direction	if delta > 0 → OPEN else → CLOSE	Select phase order	Reverse sequence to change direction
Update rule	after each step: cur_steps += dir	Stop when cur_steps == target_steps	Prevents overshoot
Timing estimate	T_total ≈ steps_to_move × t_step	t_step = phase delay (ms)	Example: 500×4 ms ≈ 2 s

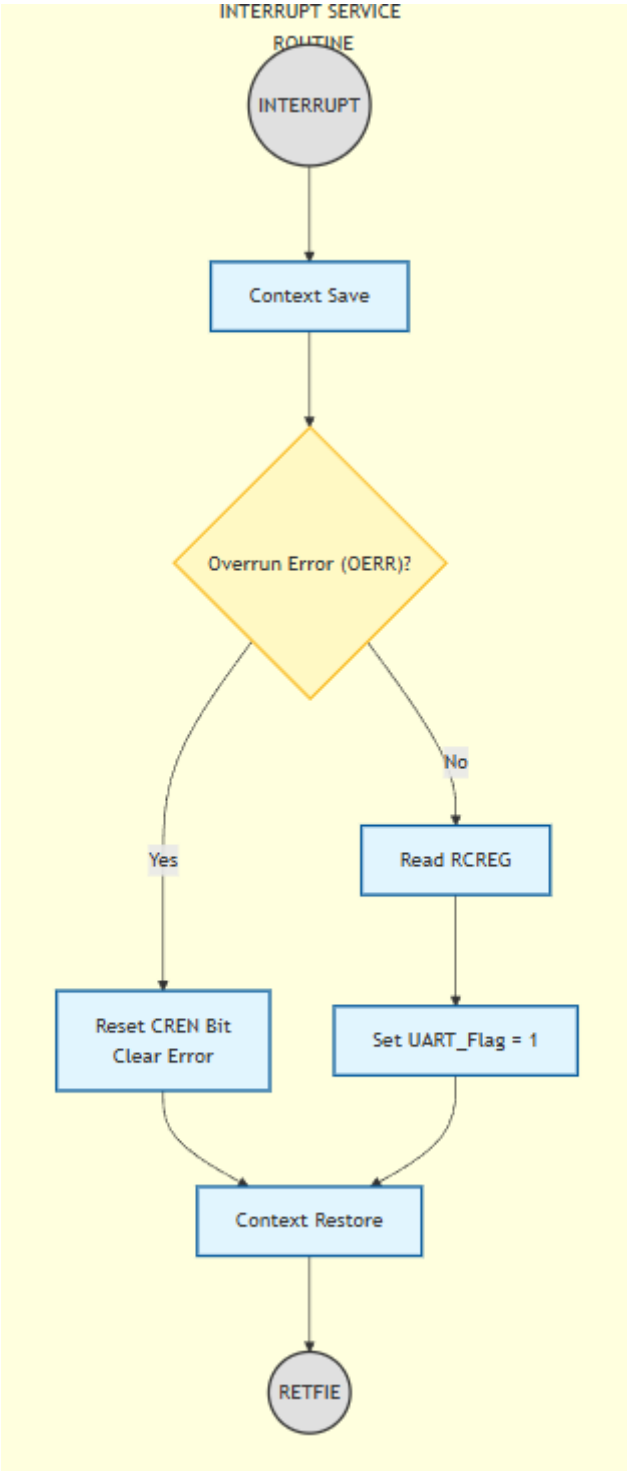
Table 7 Stepper Position Calibration (Board #2)

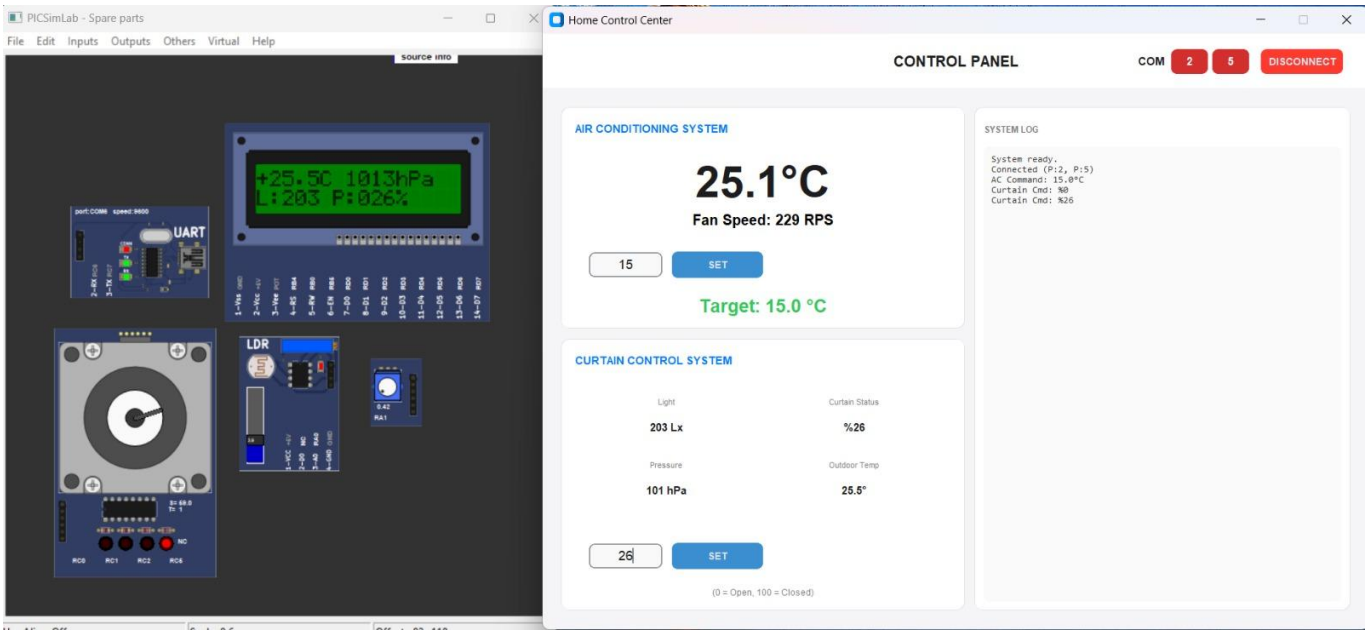
The conversion of curtain opening values (0-100%) into stepper motor steps is summarized in this table.

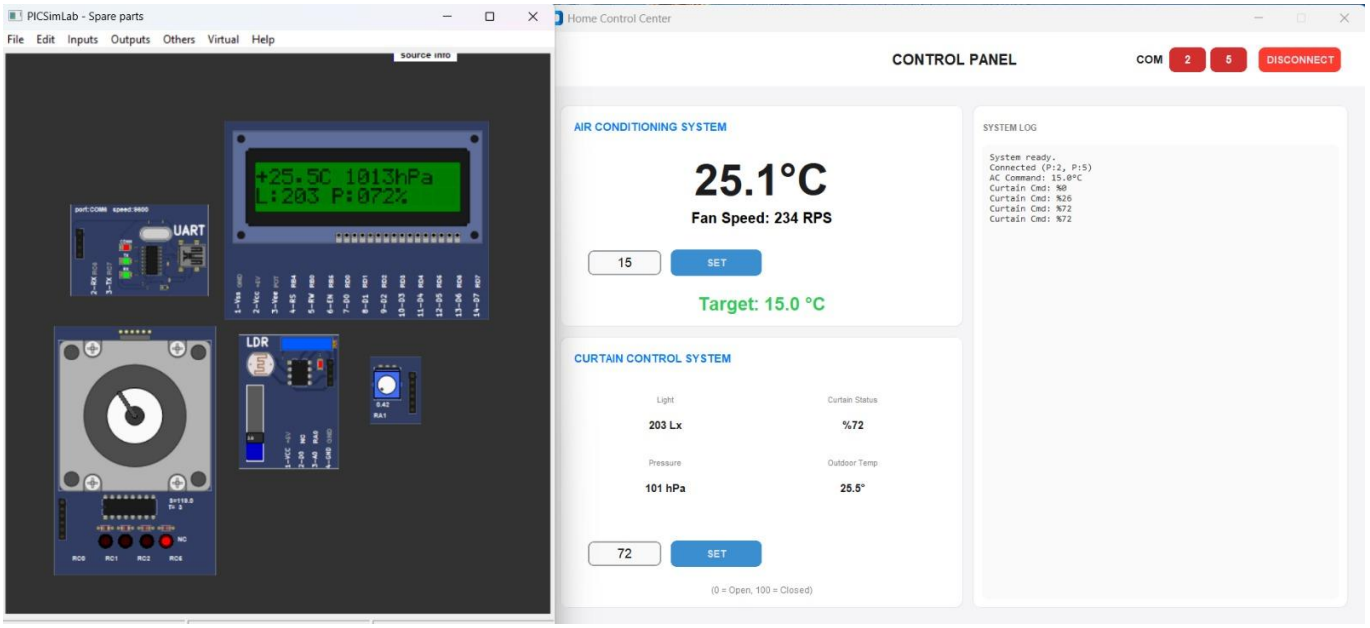
The target location is calculated as target_steps = (target_percent × STEPS_FULL) / 100 once a full-travel step count (STEPS_FULL) is established.

The motor is stepped until the target is attained after the system tracks cur_steps and chooses a direction using delta = target_steps − cur_steps.


BOARD #2 SOFTWARE FLOW
START
Init: Ports, ADC, UART, LCD
MAIN LOOP
UART Data?
Yes
No
Process Cmd
Read LDR Sensor
LDR < 80?
(Night Mode)
No (Light)
Override?
(PC Active)
Yes (Dark)
No
Read Potentiometer
Map to %
Yes
Override=0
Target=100%
Target = Pot Val
Call Motor_Control
Pos == Target?
No
Yes
Step Motor
(Lookup Table)
Update LCD?
No
Yes
Refresh Screen


INTERRUPT SERVICE ROUTINE
INTERRUPT
Context Save
Overrun Error (OERR)?
Yes
No
Reset CREN Bit
Clear Error
Read RCREG
Set UART_Flag = 1
Context Restore
RETFIE


PICSimLab - Spare parts
File Edit Inputs Outputs Others Virtual Help
+25.5C 1013hPa
L:203 P:026%
UART
LDR
Home Control Center
CONTROL PANEL
COM 2 5
DISCONNECT
AIR CONDITIONING SYSTEM
25.1°C
Fan Speed: 229 RPS
15
SET
Target: 15.0 °C
CURTAIN CONTROL SYSTEM
Light
203 Lx
Curtain Status
%26
Pressure
101 hPa
Outdoor Temp
25.5°
26
SET
(0 = Open, 100 = Closed)
SYSTEM LOG
System ready.
Connected (P:2, P:5)
AC Command: 15.0°C
Curtain Cmd: %0
Curtain Cmd: %26


PICSimLab - Spare parts
File Edit Inputs Outputs Others Virtual Help
+25.5C 1013hPa
L:203 P:072%
UART
LDR
Home Control Center
CONTROL PANEL
COM 2 5
DISCONNECT
AIR CONDITIONING SYSTEM
25.1°C
Fan Speed: 234 RPS
15
SET
Target: 15.0 °C
CURTAIN CONTROL SYSTEM
Light
203 Lx
Curtain Status
%72
Pressure
101 hPa
Outdoor Temp
25.5°
72
SET
(0 = Open, 100 = Closed)
SYSTEM LOG
System ready.
Connected (P:2, P:5)
AC Command: 15.0°C
Curtain Cmd: %0
Curtain Cmd: %26
Curtain Cmd: %72
Curtain Cmd: %72


PICSimLab - Spare parts
File Edit Inputs Outputs Others Virtual Help
source info
+25.5C 1013hPa
L:031 P:100%
UART
LDR
RC0 RC1 RC2 RC6


PICSimLab - Spare parts
File Edit Inputs Outputs Others Virtual Help
source info
+25.5C 1013hPa
L:203 P:026%
UART
LDR
RC0 RC1 RC2 RC6

API Purpose and Functionality (PC Interface)

The system communicates via a specially designed PC application C# (Windows Forms) which acts as both the central control and the monitoring interface. The application programming interface (API) simplifies the complicated low-level UART exchanges and transforms them into intuitive controls which together with the Curtain Control System (Board #2) can be monitored live.

The main objective of the API is to serve as the "Master" node in the communication topology. It carries out three important jobs:

• Monitoring: Keeps on visualizing sensor data (LDR levels, Potentiometer inputs) and actuator status (Curtain Position %).

• Remote Control: The user has the freedom to overpower the automatic logic and to set a specific curtain position manually via a slider or input box.

• Debugging: Shows raw telemetry data that are extremely important for checking the system's timing and logic during the simulation.

The application is utilizing the System.IO.Ports.SerialPort library for communication with the PIC16F877A at 9600 baud rate. The operation is further split into two concurrent tasks:

• Periodic Polling (Data Acquisition): To guarantee the dashboard gets the live data, the API is using a software timer (set to ~900ms intervals) to transfer GET commands (0x01 to 0x08) one by one.

Step 1: Requesting Target Position (Low/High Byte).

Step 2: Requesting Simulated Temperature & Pressure.

Step 3: Requesting LDR Sensor Value. The bytes that were received are reassembled (e.g., HighByte * 256 + LowByte) and the results are shown on the UI labels.

• Command Injection (User Override): If the user does something on the UI (e.g., changes the "Curtain Position" slider), the API will trigger an event that will lead to the sending of a SET command.

The slider value (0–100) is being converted to the protocol format (1xxxxxxx).

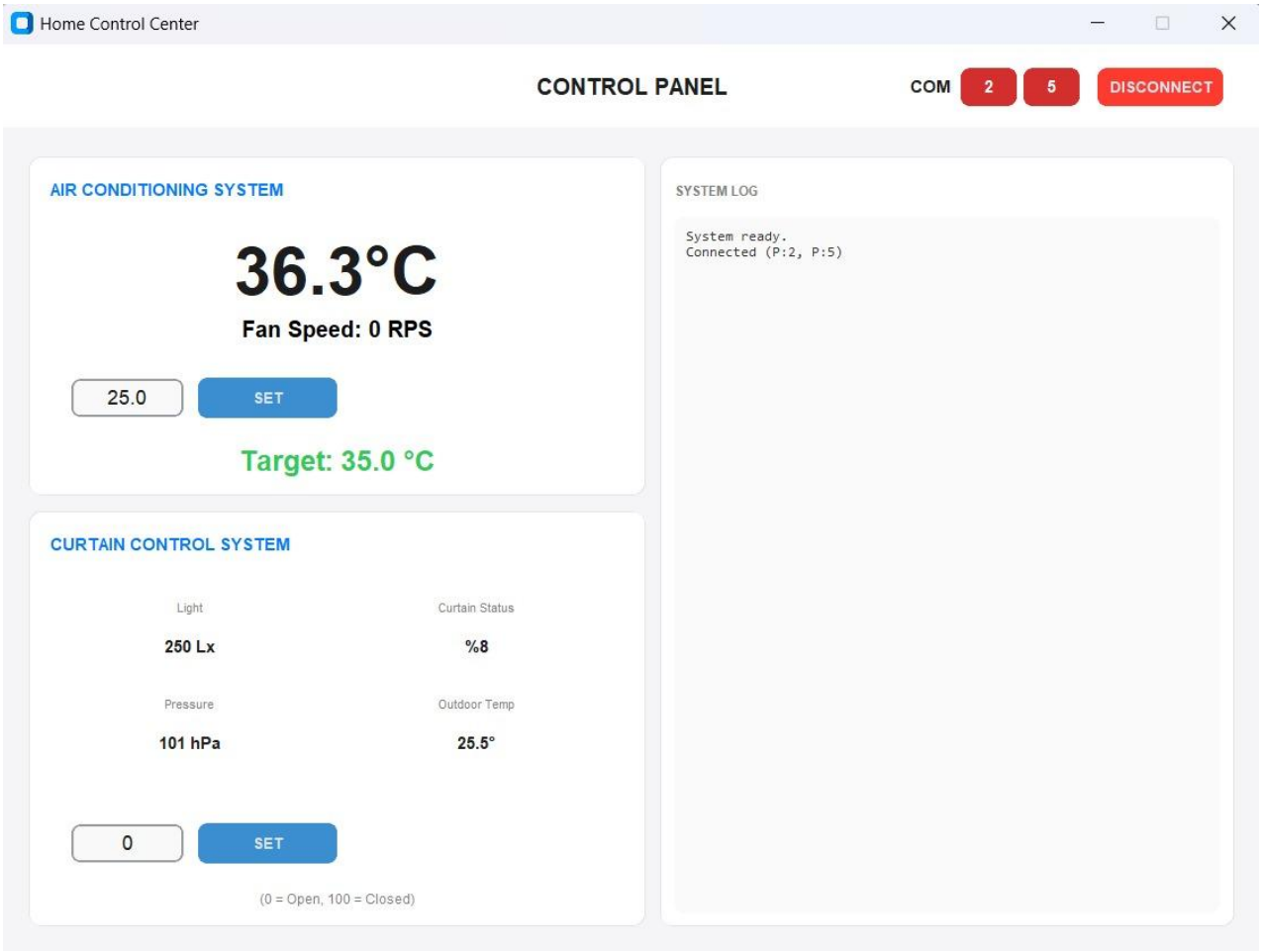
Bit 7 is being set to 1 (e.g., 50 becomes 0xB2 or 10110010) to suggest a "Set Position" command.

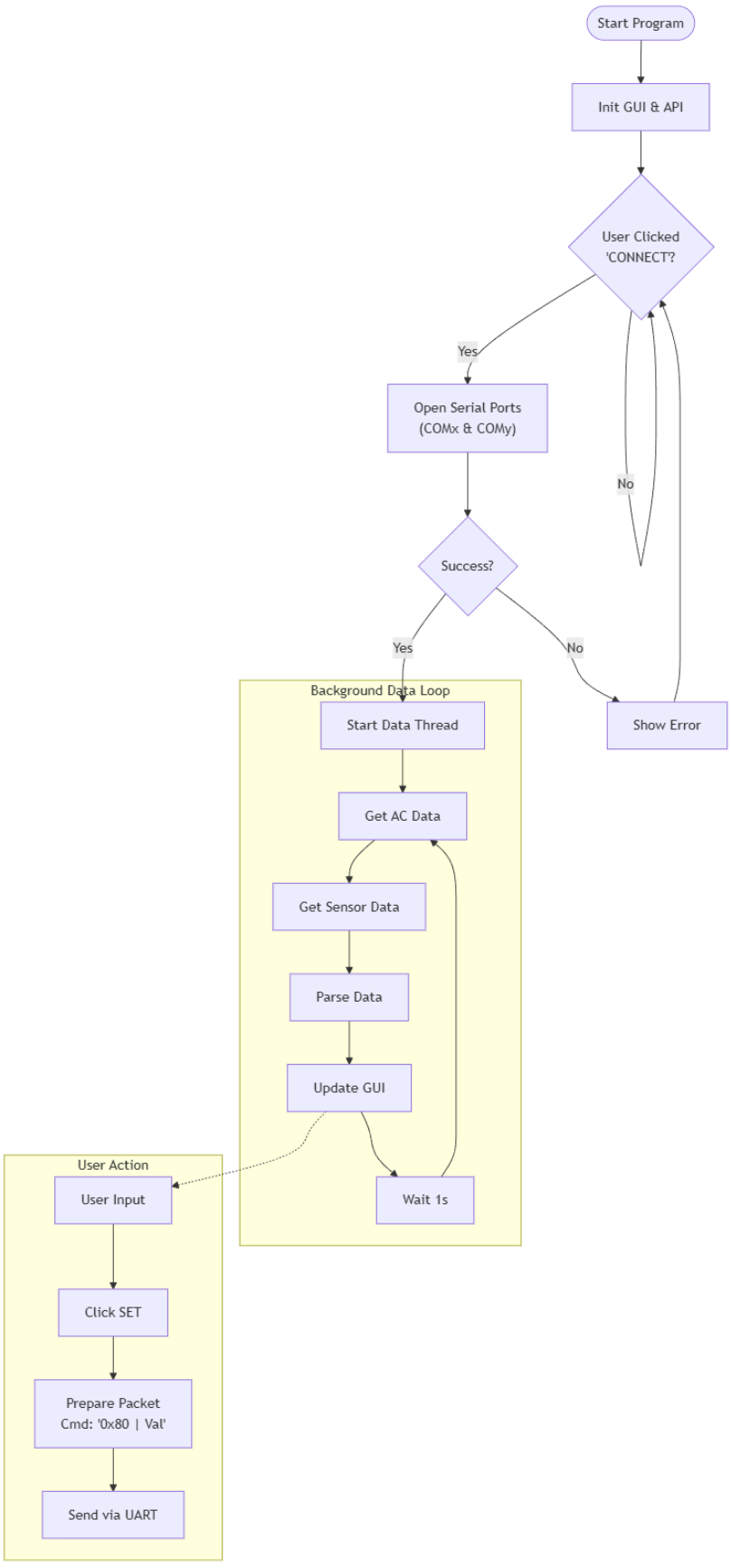
This command will require Board #2 to halt its automatic operation (resetting the internal Override_Flag) and relocate the stepper motor to the asked position straight away.

The API has incorporated primary robustness features to ensure synchronization with the hardware:

• Buffer Management: Before important data transfer operations, the input/output buffers are cleared to avoid any data overlap.

• Data Validation: The user inputs are validated to the allowed range (0–100%) before being sent to the hardware.




Start Program
Init GUI & API
User Clicked 'CONNECT?
Yes
No
Open Serial Ports (COMx & COMy)
Success?
Yes
No
Show Error
Background Data Loop
Start Data Thread
Get AC Data
Get Sensor Data
Parse Data
Update GUI
Wait 1s
User Action
User Input
Click SET
Prepare Packet Cmd: '0x80 | Val'
Send via UART

3.Task Assignment

- Enes Buğra Damar 152120221068, API,GUI Coding and Design UART coding
- Emir Cıvır 151220222102, Construction and coding Board #1
- Ali Yılmaz 152120221108, Construction and coding Board #2, coding LCD
- Zeynep Pazarözyurt 151220232065, Board #2, coding UART and Potentiometer modules, writing report
- Nazlı Polat 151220212086,Construction Board #2, coding Step Motor and LDR modules, writing report

4.Conclusion

The home automation platform development and testing, which used two boards and the PIC16F877A microcontroller, have successfully achieved the primary goal of the project named "Smart Home System." The application design was easier to understand and testing was simpler when it was partitioned into two cooperating units: User interface and indoor temperature control were managed by Board #1, whereas curtain placement and light-triggered automation were attended to by Board #2. The temperature control module of Board #1 was implemented successfully using an ADC-based sensor reading (LM35) along with a keypad-driven setpoint mechanism. Depending on the input validation for a range suitable for operating (10.0–50.0 °C), the program regulates the heater/cooler output. The LDR-based automation (night/low-light behavior) illustrated how the surrounding could detect the need for a tripped meaningful activity, and the curtain control module of Board #2 was capable of operating both automatically and manually. The communication infrastructure was the main achievement of the project, which managed to connect the two boards and the PC interface with a 9600 baud UART-based protocol and thus allowed a Python program to perform remote monitoring and command updates. Timer-based scheduling, straightforward filtering for sensor readings, stepper speed ramping for smoother positioning, and a more reliable UART protocol with timeout/retry management, checksums and acknowledgments, are just a few of the future improvements that are being considered.

Project Source Code: https://github.com/emircvr26/IntroToMicrocomputerTermProject