ASSIGNMENT COVERSHEET FOR INDIVIDUAL

Faculty of Design and Creative Technologies

First Name	Jackson	Family Name	Green	Student ID No	20123871
Paper Name	Embedded Systems Design	Paper Code:	ENEL712	Assignment Due Date	09/06/2025
Lecturer:	Jack Li	Tutorial Day	Wednesday	Date Submitted	09/06/2025
Tutor:	Tom Marsden	Tutorial Time	1200-1400	No.Words/Pages	24 Pages

In order to ensure fair and honest assessment results for all students, it is a requirement that the work that you hand in for assessment is your own work. If you are uncertain about any of these matters then please discuss them with your lecturer.

Plagiarism and Dishonesty are methods of cheating for the purposes of General Academic Regulations (GAR) http://www.aut.ac.nz/calendar

Assignments will not be accepted if this section is not completed and signed.

Please read the following and **tick** ☑ to indicate your understanding:

1. I understand it is my responsibility to keep a copy of my assignment. ☑ **Yes** ☐ **No**
2. I have signed and read the **Student's Statement below**. ☑ **Yes** ☐ **No**
3. I understand that a software programme (Turnitin) that detects plagiarism and copying may be used on my assignment. ☑ **Yes** ☐ **No**

Student's Statement:

This assessment is entirely my own work and has not been submitted in any other course of study. I have submitted a copy of this assessment to Turnitin, if required.
In this assessment I have acknowledged, to the best of my ability:

- The source of direct quotes from the work of others.
- The ideas of others (includes work from private or professional services, past assessments, other students, books, journals, cut/paste from internet sites and/or other materials)
- The source of diagrams or visual images.

Student's Signature: **Date: 09/06/2025**

The information on this form is collected for the primary purpose of submitting your assignment for assessment. Other purposes of collection include receiving your acknowledgement of plagiarism polices and attending to administrative matters. If you choose not to complete all questions on this form, it may not be possible for the Faculty of Design and Creative Technologies to accept your assignment.

Auckland University of Technology

School of Engineering, Computer and Mathematical Science

ENEL712 Embedded Systems Design

Project Report

Submitted by: Jackson Green
Student ID: 20123871

09/06/2026

Contents

List of Figures

Project Report

Developing a GUI for use with an AT90USB1287 Applications Board and an SQL Database

Jackson Green

Xuejun Li, Tom Marsden

1. Introduction

This project is undertaken as part of the responsibilities of an Embedded System Engineer at Ideal IoT Incorporation. The objective is to design and implement a C#-based Graphical User Interface (GUI) that facilitates control of the AUT AT90USB1287 Applications Board (AT90USB) from a personal computer. The GUI will be organised into multiple tabs, each dedicated to managing specific functionalities of the applications board. Communication between the GUI application and the AT90USB board will be established through a USB serial interface. Additionally, the GUI will interact with a database server configured using XAMPP and phpMyAdmin. This connection will be established over the Internet to support data logging, remote monitoring, and related database operations. The integration of embedded hardware control and network-based database communication defines the technical scope of this project.

2. Aims and Objectives

2.1. Aims

To develop a fully functioning, efficient, and maintainable C# application with clean, readable code for controlling the AUT AT90USB Applications Board over serial port via a GUI made with Windows Forms in Visual Studio.

2.2. Objectives

Microcontroller (MCU) Objectives:

- Implement digital I/O functionality to read and write to specific ports.
- Configure the ADC to read voltage-related data from onboard sensors and potentiometers.
- Set up and utilise the clock to control PWM outputs for various peripherals such as a motor, lamp, and heater.
- Establish UART communication for sending measurements and receiving instructions from the GUI using a defined protocol with state-machine decoding.

Graphical User Interface (GUI) Objectives:

- Design a multi-tabbed interface to allow user interaction with the board's digital, analog, and control functionalities.
- Initialise and manage serial communication with user-defined settings (COM port, baud rate, etc.).
- Enable real-time data exchange through structured read/write operations over the serial port.
- Implement a PI control loop to manage temperature regulation based on user-defined gain and setpoint parameters with measured sensor data.

SQL Database Objectives:

- Establish a connection to a database server configured using XAMPP.
- Implement UserID and Password authentication for access to the database.
- Log temperature and timestamp in the database given a user turns on "data logging."
- Log custom remarks in the database through a manual entry form.

3. Methodology

The approach for creating the system was structured in a staged and modular manner. Initially, the MCU firmware was developed in C, focusing on handling UART-based serial communication. The MCU listens for interrupts triggered by UART input, validates incoming packets using the predefined start and stop bytes, and determines the appropriate action based on the received data type, subsequently returning the relevant response. Once the communication layer was functional, the GUI was constructed. The design phase began by laying out the interface visually, adding and naming all required components without embedding functionality. After establishing the visual structure, the navigation logic, such as page switching and component interaction was implemented in C#. In the final phase, the SQL database was set up, and the necessary queries and data-handling logic was integrated into the C# application. This staged approach ensured that each layer of the system was independently verified before full system integration and aligned with the instructions laid out by the project brief.

3.1. Design of MCU Code

The MCU architecture was designed with the knowledge of the order and type of information that would come over the serial port in mind. The entire structure would operate around interrupts on receiving data over UART (RX Interrupts) and handling that data appropriately - checking for start and stop bytes, checking whether the instruction was simply to log and send data or to pass a new value to a component like the heater or fan, and to execute those objectives appropriately with a confirmation byte returned.

The code was written to adhere to standard coding conventions surrounding modularity, readability and sustainability by breaking down key elements into their own function.

3.2. Implementation of MCU Code

The first element of the MCU code that was built was the Interrupt Service Routine (ISR) logic. The variable `receiveByte` is a pointer to the register UDR1 where the data is received over the serial port. This determines what occurs in a given state using the switch case `mode`.

The switch case `mode` starts in the `initialise` state where it awaits the start byte and resets any local variables. Upon receiving the start byte, it waits for the instruction byte within the `instruct` case. Depending on the value of this byte it will either skip straight to checking for a stop byte, or await the Least Significant Byte (LSB) and subsequently Most Significant Byte (MSB) before checking for the stop byte. The value of the instruction byte is then used to determine whether the MCU is reading back information across the serial port or also writing to a component on the application board, and the variable `readorwrite` is assigned based on this.

If the next byte is received, there is check to ensure it is the stop byte. If so, the `readorwrite` variable is evaluated and the instruction byte is passed to the relevant function. If the write function is called, the LSB and the combined LSB and MSB as a 16-bit integer is also passed. If a different byte is received, the system returns to the initialisation state.

The `read` function taking the instruction byte as an argument uses a switch case to check what data needs to be returned. This could be

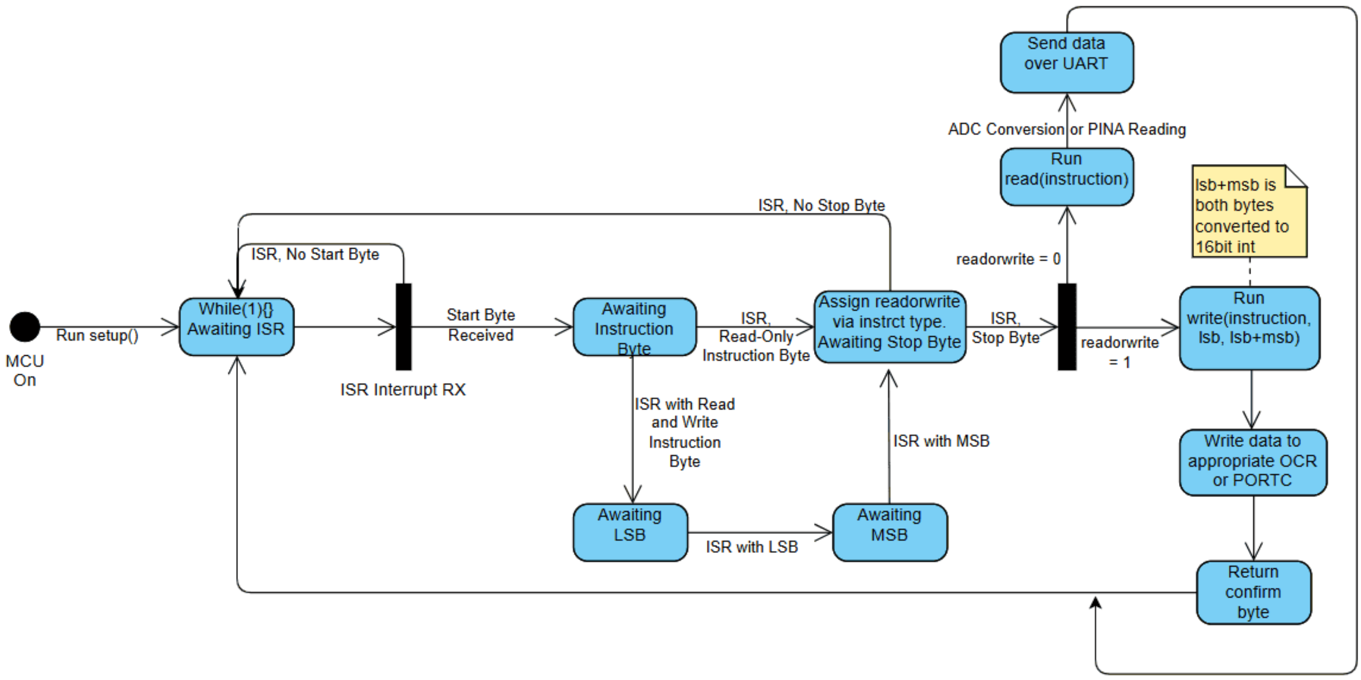


Figure 1. State diagram of the MCU code operation

a simple connection check, PINA status, potentiometer position, temperature reading or a light sensor reading. This then runs the relevant function (eg. `readPOT2` for potentiometer 2) and data is transmitted over UART. In the case of a component requiring an analogue to digital conversion (ADC), the ADC is set to single conversion with a division factor of 128. This means every time a reading is desired this function must be called, and switching channels is simple, and achieved by assigning the ADC Multiplexer appropriately.

The `write` function operates a switch case in the same way as the `read` function, but uses additional arguments to control components on the board. This is done with `sixteenbit`, the 16-bit integer combination of the LSB and MSB, in most cases. The `setHeater`, `setLight`, and `setMotor` functions pass this directly to the relevant OCR (eg. in `setHeater`, `OCR1C = sixteenbit`), and returns a confirmation byte to UDR1 dependant on the function (eg. 0x0B for `setHeater`). PORTC is an exception, and only requires the LSB. As such that variable is passed to the `write` function for use in the `setPORTC` function.

The OCR value controls the Pulse Width Modulation (PWM) so the intensity of the component's activity can be varied. The PWM timer was set up for Fast PWM, clear on compare at Top value set by ICR1 (399), with no pre-scaling.

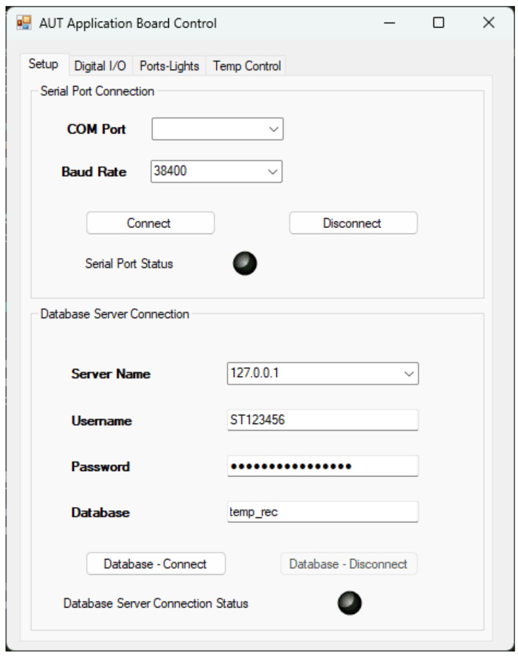


Figure 2. Initial state of the GUI on startup

3.3. Design of the GUI

Design of the GUI was informed by the project descriptor regarding the components, component interaction and visual layout. The first step was to lay out the visual user interface elements and appropriately name the components to be called intuitively later when coding. Attention was paid to the relative positions of the components so the GUI was tidy and organised, and initial states were assigned where appropriate.

The flow of the program was designed with Defensive Design principles in mind. Should the user try to change tabs before connecting to the MCU there is an error message, and the program returns to the Setup page.

On startup the program launches the setup page, as well as defaulting to a baud rate 38400 and pulling the available COM Ports to populate the COM Port dropdown. The database information is determined by the SQL database that was set up with small changes to the project specification due to an existing database on the local machine. All status lights are off until connections is established.

The second page that the user interacts with is the Digital I/O page. This page contains checkboxes to assign the LEDs on the board (PORTC), and displays the value of the switches on the seven-segment display on the board and GUI, alongside digital LEDs on the GUI. Every 100ms there is a communication with the MCU requesting the values of the switches and assigning the values of the LEDs and seven-segment display.

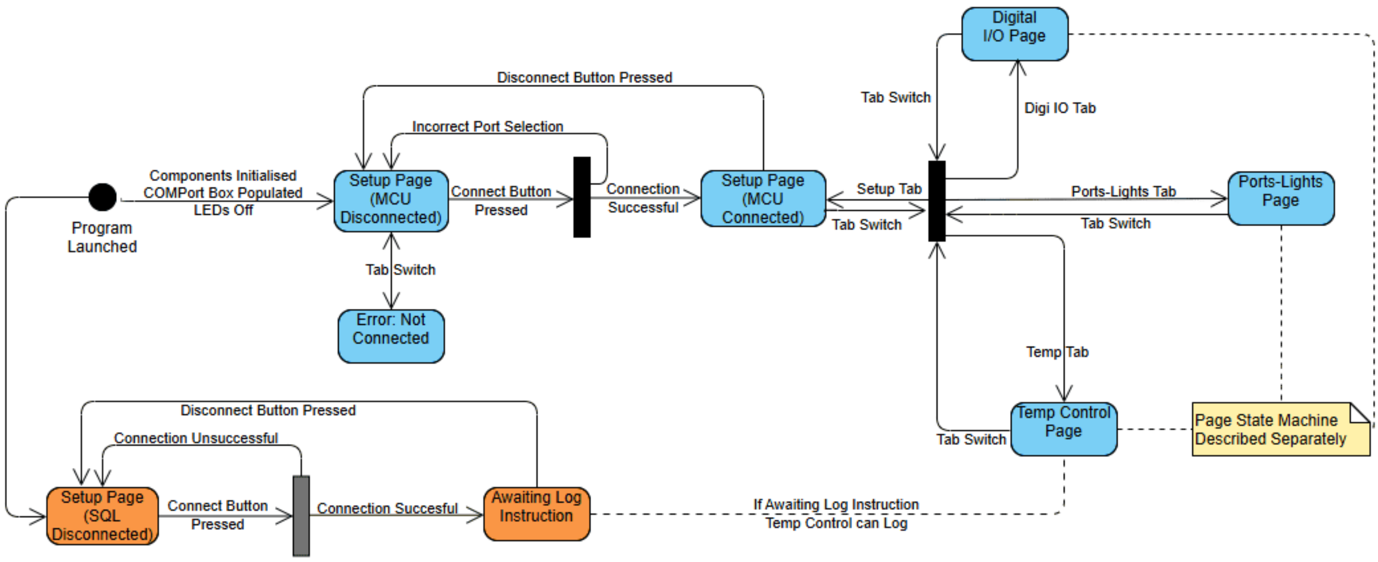


Figure 3. State diagram of the GUI code architecture

The third user-interaction page is "Ports-Lights". On this page the program reads the values of the potentiometers and converts the 0-255 bit value to voltage sequentially. These values are displayed on a pair of gauges from A. Thirugnanam[1]. It then retrieves the value of a user adjustable scroll wheel and sends it to the on-board lamp PWM to adjust intensity. Finally the light sensor next to the lamp is read and displayed on a third gauge.

The final page is the Temperature Control page. The GUI program sends an instruction to turn on a heater, and a temperature control system based on a user defined PI controller (setpoint temperature, proportional gain and integral gain) is initiated. The current setpoint temperature and the temperature since the user has been on this tab are shown in a graph, along with the fan speed and temperature in text boxes.

At the bottom of the page there is a section for interacting with the SQL database. If the database is connected the user can manually enter data from a text field, or setup an autologging system that logs the temperature and date/time data every tick.

When the user switches to any other tab, the heater is turned off.

All elements were aligned and equally spaced using the built-in tools in Visual Studio for aesthetic and readability purposes, although they broadly follow the layout given by the brief.

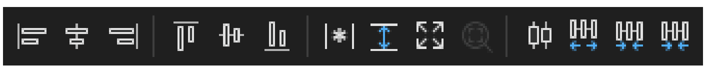


Figure 4. Visual studio GUI tool groups from left to right: X alignment, Y alignment, sizing tools, spacing tools

3.4. Implementation of C# Code for the GUI

3.4.1. AppBoard Class

To implement this design a class was made called `AppBoard`, which handles interactions with the MCU. This class includes functions such as `Connect`, `Disconnect`, `readPINA`, `writePORTC`, etc. The AppBoard class functions similarly to the MCU C code in that it is used to run functions in the main program, but contains very little logic in terms of user interaction.

The function `Connect` presented challenges to implement when accounting for edge cases. As code can execute faster than a connection is established, the method uses the ThreadPool to perform the connection logic in a background thread. This prevents the main UI thread from continuing (and calling `TxCheck`) before the microcontroller is ready. The background thread attempts to open the serial port and includes a loop that waits up to 3 seconds for the connection to be established, acting as a timeout mechanism. `serialPort.ReadTimeout` was set to 1 second as a backup.

This code is executed when the user presses the Connect button on the setup page, which passes the COM Port and Baud Rate from the dropdowns. Once the connection attempt completes, a callback (onConnected) is invoked. As it runs on a background thread, any UI updates are marshalled back to the main thread using Invoke.

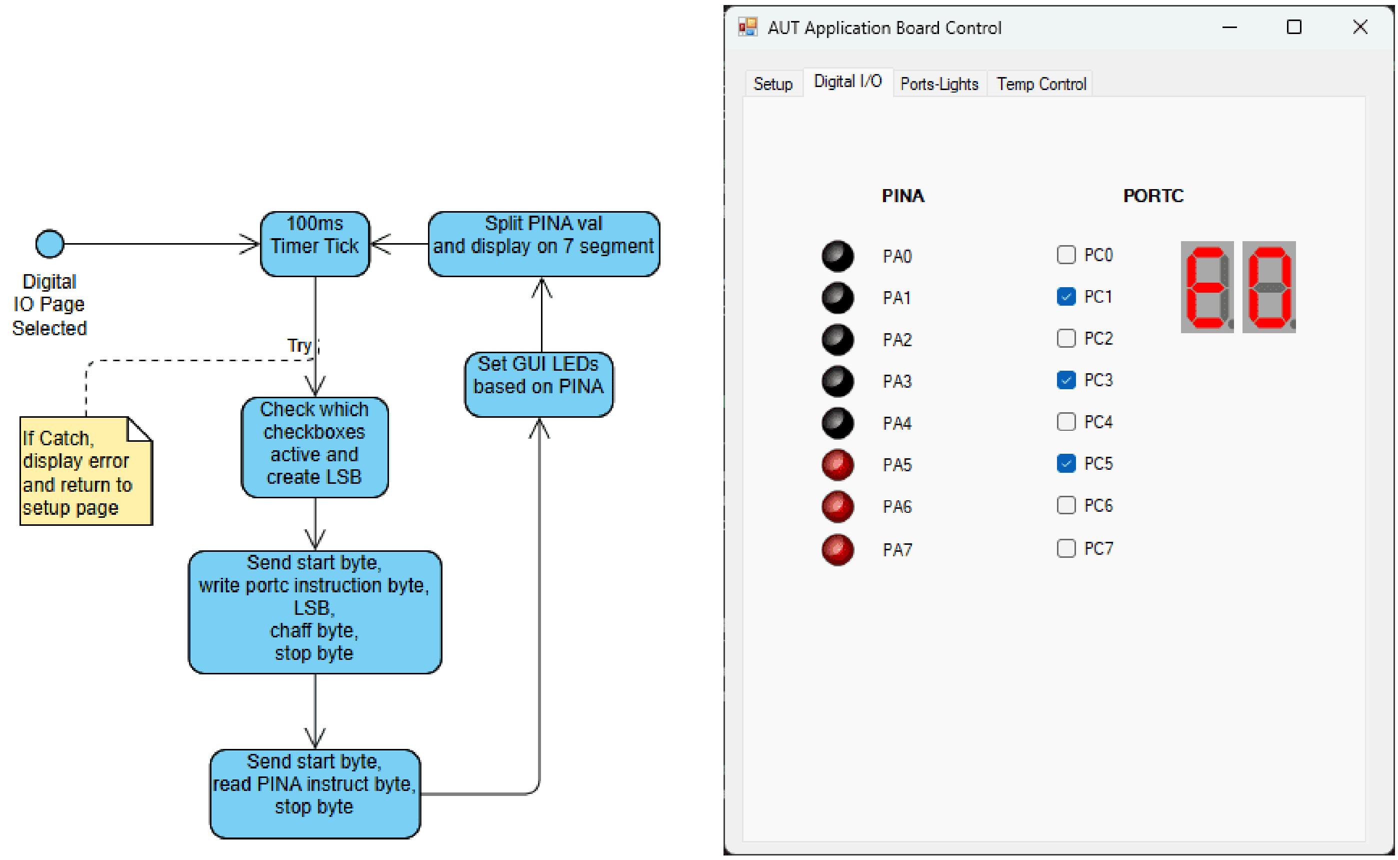
Finally there is a connection check using `TxCheck`, and if that is successful the Connect button is disabled and Disconnect button is enabled.

The `Disconnect` method simply checks if the serial port is open, and if so closes it with `serialPort.Close` which is used when the Disconnect button is pressed. On loading the form in the main program, the `getCOMs` method returns the port names.

The read and write methods operate more or less the same across the class. The read methods create an array of bytes containing the start byte, the instruction byte and the stop byte, which is then sent to the MCU using `serialPort.Write`. The response is returned in whichever data type is relevant for the program, e.g. double for Temperature or a byte for the Light Sensor.

The write methods take an argument in the form of a byte or array of bytes. If it receives an array, the first index in the byte array is assigned to the LSB and the second is assigned to the MSB. Similar to the read function, an array is created with the start byte, instruction byte, LSB, MSB, and stop byte, before being passed to `serialPort.Write`. It then listens for a response byte which it takes as an integer, and checks this against the expected response. If the wrong response is received, an error message shows.

Where this differs are the `readPotV(int pot)` and `writeHeat(string value)` methods. For reading the potentiometers the method is passed a zero or one depending on the potentiometer that is to be read and adjusts the instruction byte accordingly. When writing to the heater the the LSB and MSB are zero unless the string is “on”.



(a) State diagram of the Digital IO page

(b) Digital IO page

Figure 5. Overview of the Digital IO interface and its state transitions

3.4.2. Main C# Program

The main program file relating to the .NET form interaction, contains the code in the partial class Form1. On startup it initialises some variables that are required to be globally declared, such as the bool `shouldLogData`, the double `integral`, and the EventHandler `ev`. It creates a new Appboard object, and runs `loadForm`, which calls the `getCOMs` method previously discussed, sets LEDs to off, and sets some GUI elements to their default state. The Connect/Disconnect button behave as previously discussed.

The user may switch tabs on the setup page. When they do so, the event handler `appTabs_SelectedIndexChanged` checks if there is a connection. If not, there is an error is displayed before "kicking" the user back to the setup page if a check determines they are not already there.

If there is a connection, there is a switch case based on the selected tab. Switching tabs to a page that is not setup initiates `UpdateTimerEvent` where a method is passed and called on every tick, specified in the Form Designer properties as 100ms. All pages other than setup contain a try-catch that stops the timer, displays an error message and switches to the setup page.

Tab Response:

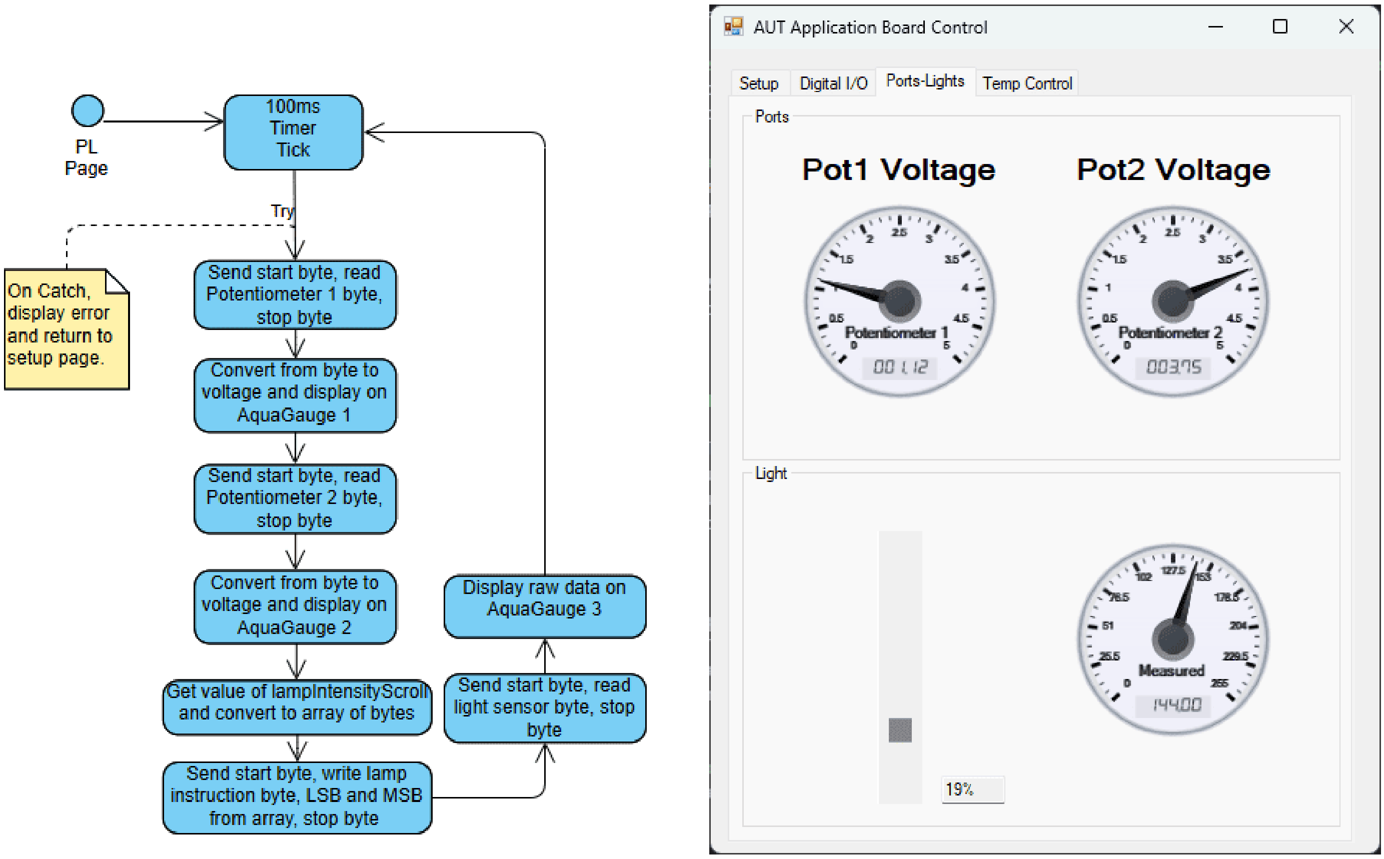
- Setup Page: Timer is turned off, and `appBoard.tempTabClosed` is called to turn off the heater
- Digital IO Page: Runs `digIO_Tick` in the `UpdateTimerEvent` method and `appBoard.tempTabClosed` is called to turn off the heater
- Ports-Lights Page: Runs `pot_tick` in the `UpdateTimerEvent` method and `appBoard.tempTabClosed` is called to turn off the heater
- Temperature Control Page: Sets `x`, `integral` and `prev_error` to 0, runs `appBoard.WriteHeat("on")` to turn on the heater and runs `temp_tick` in the `UpdateTimerEvent` method

`digIO_Tick` calls `getPORTCValue()` to retrieve the boxes that have been checked. This is done by creating a byte of value 0x00 and creates the required byte using `if (tickPC0.Checked) lsb |= 0b00000001`, where each tickPCn checked shifts where the 1 is in the byte and this is repeated for each position in the byte and the corresponding checkbox. The `appBoard.WritePORTC` method is then used to send that value to the MCU. `displayByteInHex` is called with `getPORTCValue` as the argument. This function splits the byte into a low and high "nibble". These nibbles are converted into a high and low `char`s with `hexToChar`. If the nibble is between 0 and 9, `hexToChar` returns the character '0' through '9' by adding the nibble to the ASCII value of '0'. If the nibble is between 10 and 15, it returns 'A' through 'F' by adding the result of `nibble - 10` to the ASCII value of 'A'. These are then converted to a string while being passed to `sevenSegment.Value`, a type that can be read by the sevenSegment plugin[2].

`ReadPINA` is then called and assigned to the variable `PINAVal`. The LEDs on the GUI are then set based on this variable with `setPINALEDs`.

The Ports-Lights page runs three functions sequentially - `potentiometres`, `lampScroll`, and `lightSensor`. It also updates the text box next to the scroll bar on update of the scroll bar to display the value from 0-100 followed by a percent symbol.

`potentiometres` calls `appBoard.ReadPotV(n)` for each potentiometer converted to a float, divides the result by 255 and multiplies by 5 to obtain the voltage of the potentiometer from 0-5V. This is saved to `pot[n+1]DisplayVal` and passed to `pot[n+1]VoltageDisplay.Value`, setting the gauge needle. These values are then assigned to `pot[n+1]VoltageDisplay.RecommendedValue` to assign the



(a) State diagram of the Ports-Lights page

(b) Ports-Lights page

Figure 6. Overview of the Ports-Lights Tab and its state transitions

readout under the needle.

`lampScroll()` converts the value of the scroll wheel to a 0-255 range and converts it to a byte. This is then passed to the lamp using `appBoard.WriteLamp`.

Finally, `lightSensor` reads from the light sensor using `appBoard.ReadLight` and applies the raw 0-255 value to the `lightDisplay.Value` and `.RecommendedValue`.

The Temperature Control page is the most complex. `temp_tick` pulls the setpoint value, Kp and Ki from the GUI and passes them to `piLogic`. There is an `if` statement that checks whether the user has enabled data logging that will be discussed in a subsequent section of this report.

`piLogic` assigns the sample time `Ts`, sets the double `currentTemp` to `appBoard.ReadTemp`, and converts it from a voltage to degrees C. Changes were made to the provided PI equations which is discussed after this subsection. The output `u` is converted into an array of bytes and passed to the MCU with `appBoard.writeFan`.

The measured temperature and setpoint temperature are then added to the `tempChart`, which is dynamically scaled to account for either temperature.

3.4.3. Changes to Provided PI Logic

The implemented PI controller deviates from the original assignment specification to address issues related to integral windup and numerical instability due to the chosen integration method. The standard PI control law is defined as:

$$u(t) = K_p e(t) + K_i \int_0^t e(\tau)\, d\tau$$

where $e(t)$ is the error between the measured temperature and the desired temperature. In the original brief, the integral term was computed using the backward Euler method:

$$\text{integral} = \text{integral} + e(t) \cdot T_s$$

However, this method exhibited significant oscillatory behaviour in practice, as seen in Figure 14a. To mitigate this, the integration method was replaced with the bilinear (trapezoidal) rule, which offers improved numerical stability:

$$\text{integral} = \text{integral} + \frac{T_s}{2}(e(t) + e(t - T_s))$$

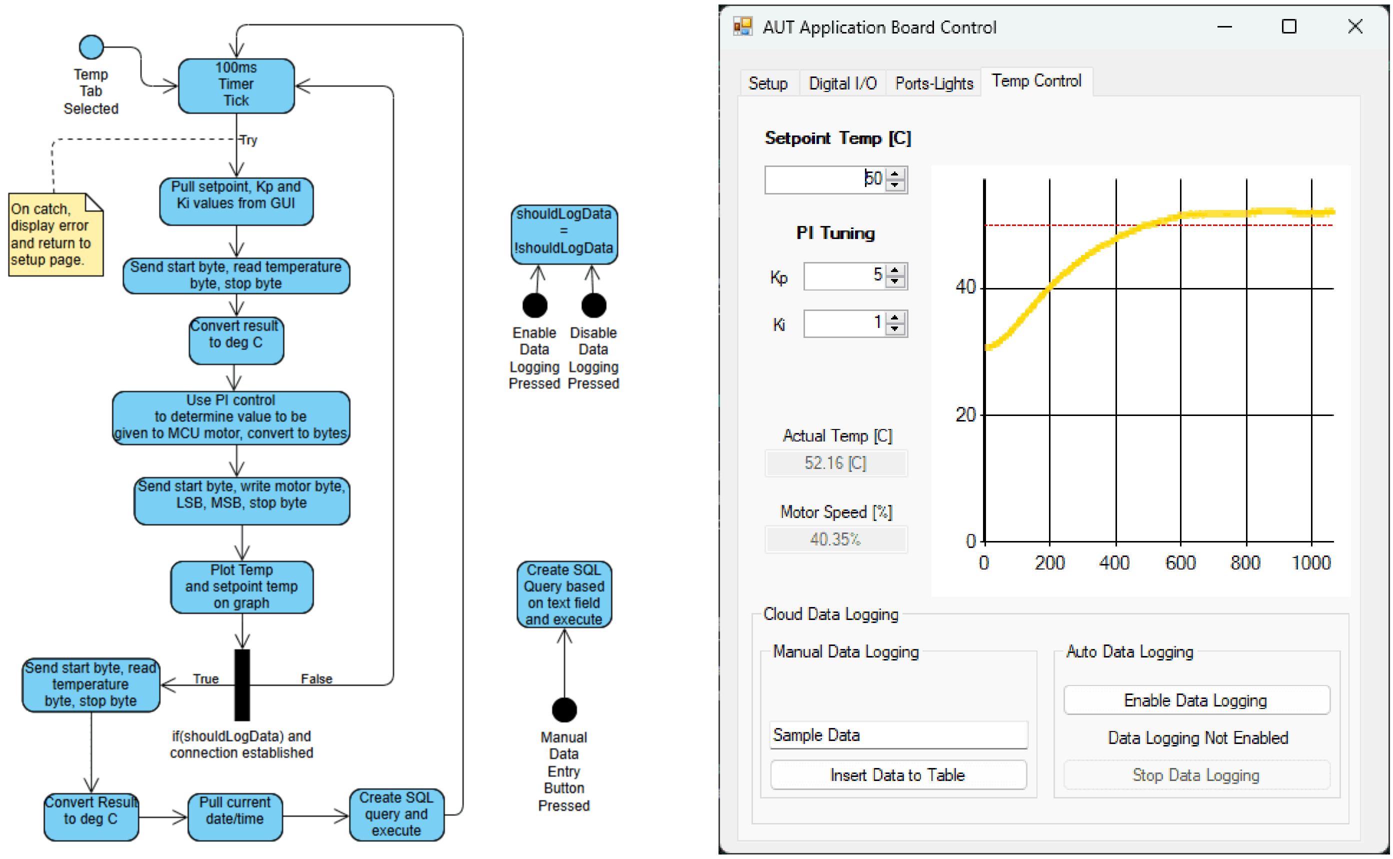
In addition, integral windup was addressed by implementing a conditional integration strategy. The integral term is only updated if the control signal u is within the virtual actuator bounds $[-20, 180]$, or if the control effort is out of bounds but the error would drive it back towards the valid range. This was done using the general approach outlined by Wilson[4], however there include some small changes to account for this particular case. This is expressed as:

$$\text{if } (u > -20 \wedge u < 180) \vee (u < -20 \wedge e < 0) \vee (u > 180 \wedge e > 0)$$

If this is satisfied the new integral is computed, the control signal is recalculated and then clamped to the range $[-20, 180]$ using a clamp function added to the code. It is scaled to fit the PWM signal range $[0, 399]$ by normalising to a 0 to 1 range and multiplying by the max value:

$$u = \left(\frac{\text{Clamp}(u, -20, 180) + 20}{200} \right) \cdot 399$$

These changes improved the response by mitigating integral windup and using a more accurate integration method, giving far less oscillatory behaviour.



(a) State diagram of the Temperature Control page

(b) Temperature Control page

Figure 7. Overview of the Temperature Control Tab and its state transitions

Note: The clamp function simply takes a value, a maximum and a minimum, and compares the value. If it is above the maximum or below the minimum it is clamped to the max/min appropriately and returned. Otherwise it is returned with no change.

3.5. Implementation of C# Code for the Database

The database code was based on the instructional video provided in the brief[3]. If the Database Connect button on the setup page is pressed, a new MySQLConnection is created and opened with the connection string set by `string connString = $"SERVER={address};PORT=3306; DATABASE={database};UID={uid};PASSWORD={pass}"`. On the Temp Control tab there are two buttons that interact with the SQL database - "Insert Data to Table" and "Enable Data Logging". In both cases once a query was established a new SQL command was created with the appropriate parameters and executed, e.g.:

```
string query = "INSERT INTO table (column)...
VALUES (@entry)";

using (MySqlCommand cmd =...
new MySqlCommand(query, conn))
    {
        cmd.Parameters.AddWithValue("@entry", column);
        cmd.ExecuteNonQuery();
    }
```

3.5.1. Manual Data Logging

If the user presses this button, a query is created to insert the value of the text box `manualData` into the "remark" column of the "temperature" table with `string query = "INSERT INTO temperature (remark) VALUES (@data)"`. There is error handling to ensure a connection existed before executing this query and a message box upon successful addition to the database.

3.5.2. Auto Data Logging

To enable automatic data logging there needed to be a way to update the `temp_tick` function so data could be sent every time the temperature was checked. If the "Enable Data Logging" button is pressed', a global boolean `shouldLogData` is flipped to true, and an if statement in the `temp_tick` function passes the current temperature, converted from voltage to degrees C, the system date/time as a string, and the username to `addDataToDatabase`. This function acted similarly to the manual data approach, but included error handling that turned off automatic data logging should it be unsuccessful, and used the query `"INSERT INTO temperature (timeStamp, temp, remark)... VALUES (@timedata, @tempdata, @userdata)"`.

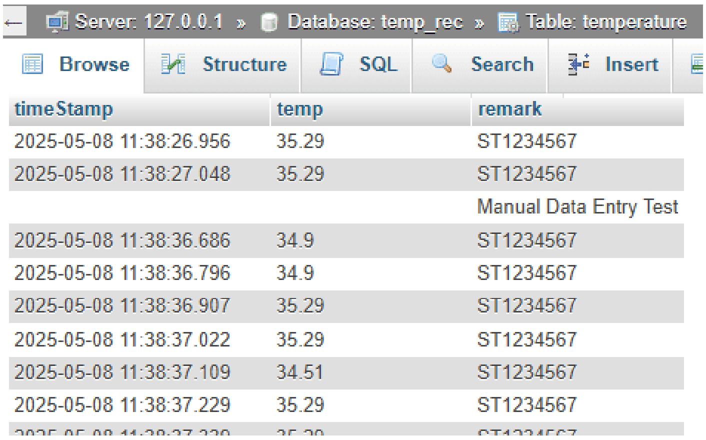
Disabling automatic data logging simply requires the user to push the "Disable Data Logging" button, which flips the `shouldLogData` boolean to false and the if statement in `temp_tick` is not satisfied.

3.6. Database Setup

Setting up the database was done with small changes to the report specification as the database had already been established on the PC used for this task by another student. XAMPP Control Panel was started and Apache and MySQL was started. In phpMyAdmin a user account with the username "ST1234567" (note the added 7) was added, as well as a randomly generated password, which was saved as the default value in the database password box in Visual Studio. The user was given all data and structure privileges.

A new databse was then created and named "temp_rec" (rather than "temperature_record" to avoid conflicts with the existing database), with a new table named "temperature" containing three columns: "timeStamp", "temperature", and "remark". The data types for these columns were set to varchar(50), varchar(20), and text, respectively.

A test of the database was done using the C# code described above and found both the manual data entry and automatic data logging both operated as expected.



timeStamp	temp	remark
2025-05-08 11:38:26.956	35.29	ST1234567
2025-05-08 11:38:27.048	35.29	ST1234567
		Manual Data Entry Test
2025-05-08 11:38:36.686	34.9	ST1234567
2025-05-08 11:38:36.796	34.9	ST1234567
2025-05-08 11:38:36.907	35.29	ST1234567
2025-05-08 11:38:37.022	35.29	ST1234567
2025-05-08 11:38:37.109	34.51	ST1234567
2025-05-08 11:38:37.229	35.29	ST1234567

Figure 8. The 'temperature' table in the 'temp_rec' database at http://localhost/phpmyadmin/

4. Results

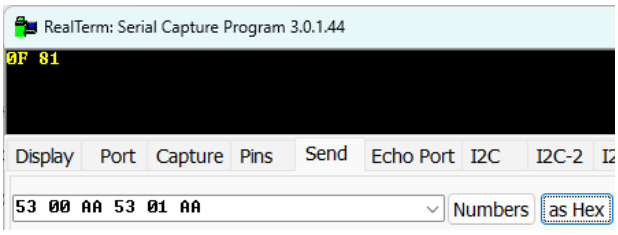
Following is the simulated and actual results from interactions with the MCU by the C# based GUI. The error handling of the system was also tested.

4.1. Simulation Results

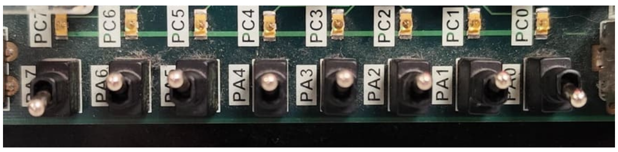
Simulating the response of the MCU was performed using RealTerm. RealTerm allows a user to connect to devices through a serial port and send commands in a variety of forms, including binary and hexadecimal.

4.1.1. Read

The "read" functions were tested using RealTerm, with successful responses. Included are the responses of a Tx Check and PINA read request. The PINA read request was sent with the PA0 and PA7 switches oriented to 1, and PA1 to PA6 switches were oriented to 0. This should return the hex value 81 in the RealTerm terminal. As per Figure 9a the appropriate data was received.



(a) Sending the start byte, Tx check instruction, and stop byte receives the correct response, followed by sending the start byte, PINA check instruction, and stop byte. This receives the correct response back from the MCU.



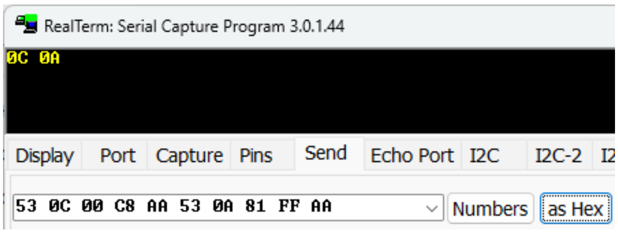
(b) PINA orientation for testing.

Figure 9. Read function response and testing setup.

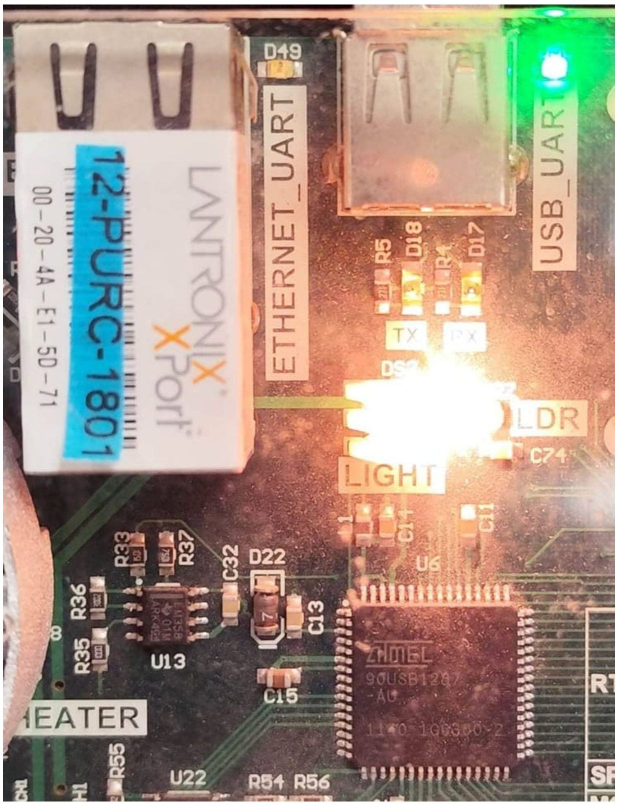
Each of the functions were simulated sequentially, and all returned data consistent with the expected output, allowing for the development to progress to integration with the GUI.

4.1.2. Write

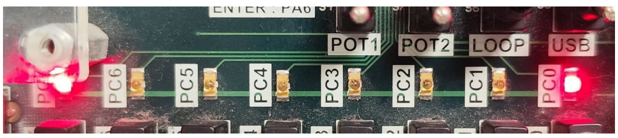
Simulating the "write" functions also achieved successful results as per Figure 10. Included is the test of the lamp, set to half brightness (200, or 00 C8 in hexadecimal), and PORTC 0 and 7 LEDs set to on.



(a) Sending the start byte, Write Light instruction, LSB, MSB, and stop byte. Then sending the start byte, Write LED instruction, LSB, chaff byte, and stop byte. This receives the correct response back from the MCU.



(b) Lamp response to RealTerm Instructions.



(c) LED response to the RealTerm Instructions.

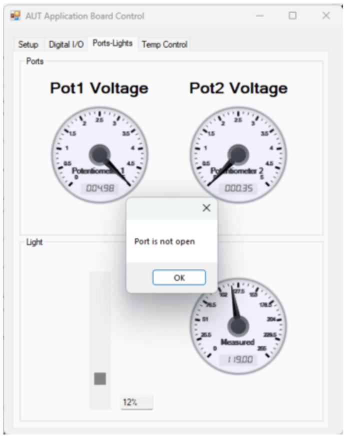
Figure 10. Overview of the board response.

These were correctly set and returned the appropriate return byte.

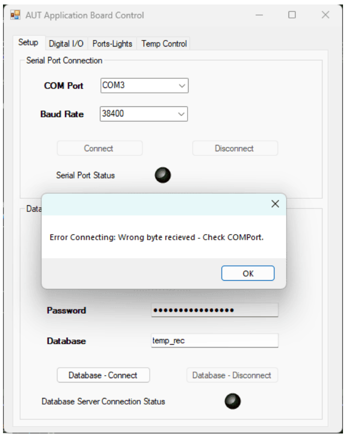
4.2. Experimental Results

Once the program was complete a series of tests were performed. These were designed to ensure the GUI functioned to the desired specification, to ensure user error was handled correctly, and to probe for potential bugs.

Checks to ensure the GUI would handle issues surrounding user connection to the MCU correctly were performed. This included switching tabs before establishing a connection and attempting a connection with the wrong COM Port set.



(a) Attempting to switch tabs without a connection.



(b) Attempting to connect with the wrong COMPort.

Figure 11. Overview of the GUI response to user MCU connection errors.

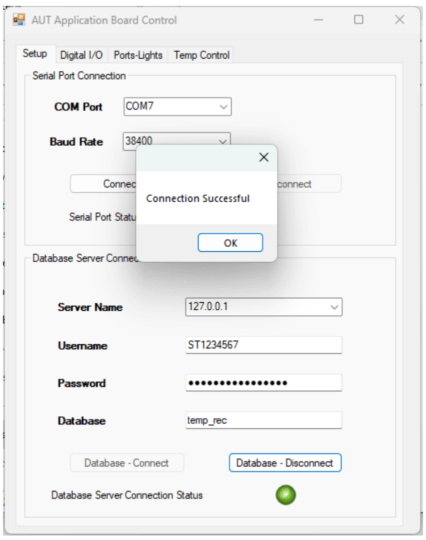
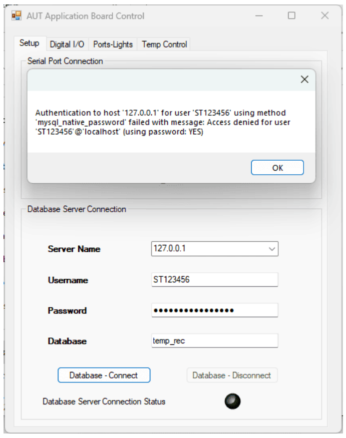
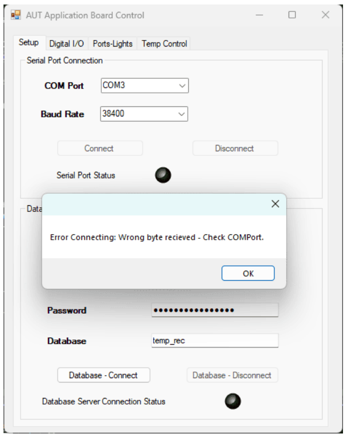


Figure 12. Successful connection to the database

Similar checks were performed for the database connection. This involved entering incorrect details into the server details, and attempting to log data without a connection.



(a) Attempting to connect with incorrect details.



(b) Attempting to log temperature without database connection.

Figure 13. Overview of the GUI response to user database connection errors.

Testing the Digital IO page consisted of checking PORTC tickboxes and flipping the PINA switches on the application board to ensure the response was correct, which behaved as designed as seen in Figure 15.

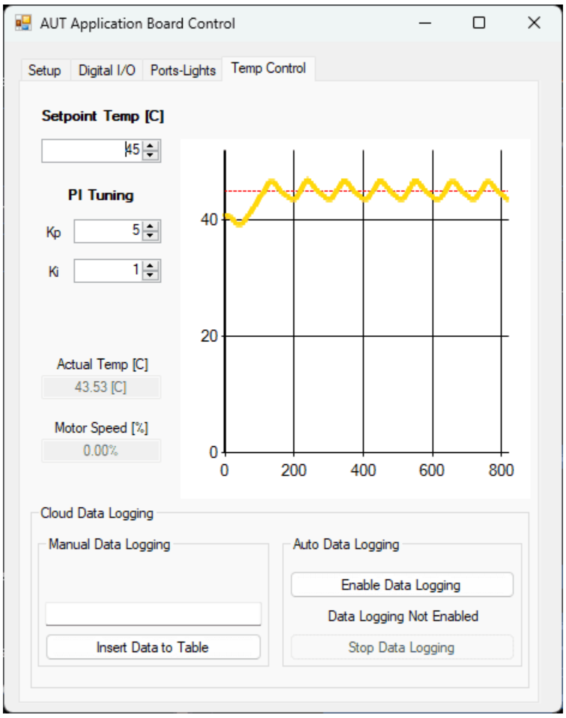
The Ports-Lights page was tested with two lamp intensities, as seen in Figure 17b and Figure 16b, so the light sensor could be properly tested. It responded with a very high degree of sensitivity to the upper end of the lamp intensity scale, giving little usable data beyond 20%.

The potentiometers were set to their respective high and low limits, with potentiometer one set to high and potentiometer set to low. These responded as expected.

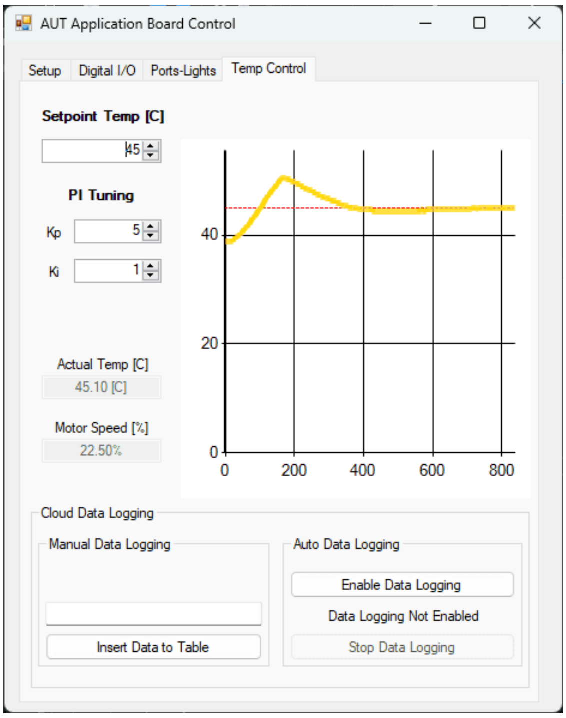
The temperature control page correctly turned on the heater to full power and controlled the fan motor only when the Temp Control tab was selected, and turn off both when the user was on a different page. The data logging and manual data entry both performed accurately as seen in Figure 8, and the temperature was maintained at the desired setpoint as seen in Figure 14b.

5. Discussion

There were many elements of the development of this program that contributed to validation and successful deployment. Object-oriented programming (OOP) in C# significantly contributed to the clarity, scalability, and maintainability of the GUI. The AppBoard class abstracted hardware interaction, encapsulating serial communication logic and command transmission into a single interface. This separation of concerns meant that the GUI logic within Form1 did not



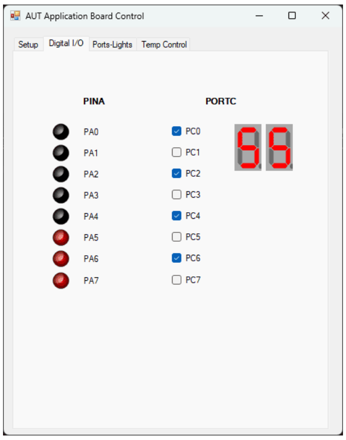
(a) Temperature Response with original PI Logic



(b) Temperature Response with modified PI Logic

Figure 14. Overview of the Temperature Control Response

require detailed knowledge of the communication protocol or byte-level encoding, leading to cleaner, more maintainable code.



(a) Using the GUI to set the PORTC LEDs and seven segment value, and reading PINA values for the GUI LEDs.

(b) Application board response.

Figure 15. Overview of the board response to GUI control on Digi IO page.

OOP features such as encapsulation helped prevent unintended modifications to shared state by restricting access to internal variables. Inheritance and polymorphism were employed minimally, as the project did not involve complex type hierarchies, but modular class structure still enabled code reuse and ease of testing. For instance, tab-specific functions for updating UI elements and handling periodic tasks were compartmentalised using the UpdateTimerEvent, which allowed consistent logic to be executed across multiple tabs without code duplication.

Defensive programming techniques were employed throughout the development to ensure reliability, particularly at the interface between the GUI and microcontroller. In the C# application, input validation, exception handling, and timeout logic were used to guard against unexpected user input and communication failures. For example, the `Connect` method of the AppBoard class used background threading via the ThreadPool to avoid blocking the UI. A retry loop with a timeout mechanism ensured connection robustness, and UI updates were marshalled safely back to the main thread. Try-catch blocks across all tabbed interfaces caught runtime exceptions and returned users to a known safe state, namely the setup page.

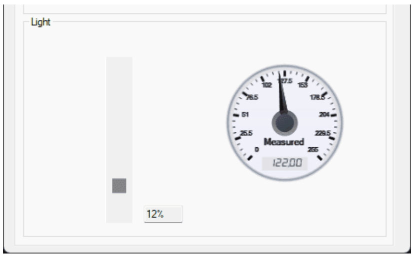
GitHub was used throughout the development process to manage version control across the C# GUI and embedded C microcontroller firmware. Git enabled version control, helping to fix bugs and test modules independently before merging into the main branch. Commit history allowed for traceable changes, while tags were used to mark key milestones in the development cycle. This was especially important given the need to synchronise protocol definitions and data structures between the C# GUI and the embedded C firmware, reducing integration issues.

This also enabled the creation of branches to test multiple versions of the program. The testing of different PI implementations was possible with branches using different methods of integration, before the selected option was merged to the main branch. This could be worked on in parallel with other parts of the code without impacting or overwriting it due to how GitHub handles merging files based on changes.

Evaluating the results of the PI controller, the implemented system exceeds the performance requirement of the original project objectives by removing the "all or nothing" oscillating response that was initially produced with a simple integration sum. A drawback to this implementation is there is now marginally more overshoot, approximately 5 degrees Celsius when initialising at 40 degrees with a 45 degree setpoint temperature. This overshoot may be mitigated by reducing the contribution of the integral term, either by lowering the integral gain K_i or by further constraining the conditions under which integration occurs.

6. Conclusion

The development of a GUI for use with an AT90USB1287 Applications Board and an SQL Database, undertaken as part of the responsibilities of an Embedded System Engineer at Ideal IoT Incorporation, was a success. A C#-based Graphical User Interface (GUI) that facilitates control of the AUT AT90USB1287 Applications Board (AT90USB) from a personal computer was designed and implemented per the report specification, with changes made to either improve functionality or handle small, unforeseen issues. The changes to the PI controller unquestionably improved temperature control performance and stability response. Further improvements may be made that integrate the heating element into the PI response rather than being at full power at all times, or to further dampen the system, but that is out of the scope of this task.



(a) Using the GUI to set the lamp to 12% where the sensor will not be "blown out".

(b) Application board response - Lamp.

Figure 16. Overview of the board response to GUI control on Ports-Lights page with the light sensor reading low light.

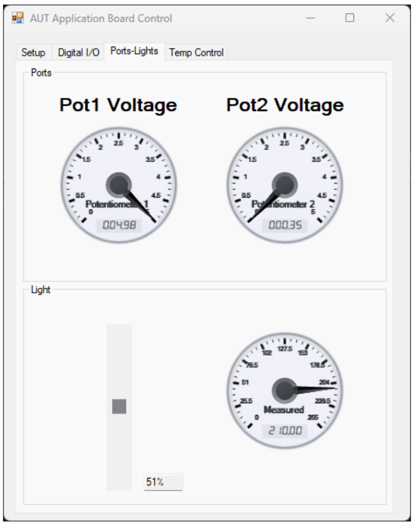
References

[1] A. Thirugnanam. "Aqua gauge". Accessed: 2025-05-21, The Code Project. (Sep. 4, 2007).

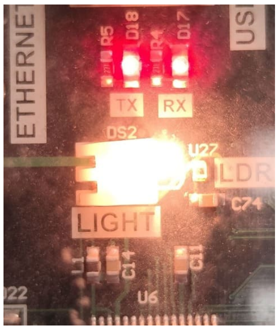
[2] D. Brant. "Seven-segment led control for .net". Accessed: 2025-05-21, The Code Project. (Jul. 1, 2009).

[3] Tech & Code Environment. "How to connect c# application to (mysql) a remote database on cpanel or godaddy". Accessed: 2025-05-07, YouTube. (May 2021).

[4] D. I. Wilson, *Advanced Control Using MATLAB: Or Stabilising the Unstabilisable*. New Zealand: Inverse Problem Ltd, 2025, January 31, 2025.



(a) Using the GUI to set the lamp to 51% and read potentiometers one and two, which are turned all the way clockwise and anticlockwise respectively.



(b) Application board response - Lamp.

(c) Potentiometer settings.

Figure 17. Overview of the board response to GUI control on Ports-Lights page.

7. Appendix

7.1. Form1.cs File

```
using System;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Windows.Forms;
using System.Windows.Forms.DataVisualization.Charting;
using MySql.Data.MySqlClient;

namespace gui
{
    public partial class Form1 : Form
    {
        private AppBoard appBoard;
        private EventHandler ev;
        private double integral;
        private double prev_error;
        private int x;
        MySqlConnection conn;
        private bool shouldLogData = false;

        public Form1()
        {
            appBoard = new AppBoard();
            InitializeComponent();
            loadForm();
            setupCombos();
        }
                // INITIALISATION //
       public void loadForm()
        {
            try
            {
                string[] comPorts = appBoard.getCOMs(); // Get the list of COM ports
                serialPortStatusLED.On = false;// connection light off
                datBasStatLED.On = false;//6 weeks later my naming conventions are less verbose
                disconnectButton.Enabled = false;
                comPortDropdown.Items.Clear(); // clear any existing items in the ComboBox

                // Add each COM port to the ComboBox
                foreach (string port in comPorts)
                {
                    comPortDropdown.Items.Add(port);
                }
                lampIntensityScroll.Value = 0;
                lampPercentDisplay.Text = "0%";
            }
            catch (Exception ex)
            {
                MessageBox.Show($"Error: {ex.Message}");
            }
        }

        private void UpdateTimerEvent(EventHandler newev)
        {
            appTimer.Stop();
            if (ev != null)
                appTimer.Tick -= ev;
            appTimer.Tick += newev;
            ev = newev;
            appTimer.Start();
        }

        public void isMCUConnected()
        {
            if (!appBoard.serialPort.IsOpen) { MessageBox.Show("Port is not Open");tabControl1.SelectTab(0); }
        }

        private void appTabs_SelectedIndexChanged(object sender, EventArgs e) // TAB CONTROL
        {
            if (appBoard.serialPort.IsOpen)
            {
                switch (tabControl1.SelectedIndex)
                {
                case 0: // Setup Tab
                    appTimer.Stop();
                    appBoard.tempTabClosed();
```

```
                    break;

                case 1: // Digital I/O Tab
                    UpdateTimerEvent(digIO_Tick);
                    appBoard.tempTabClosed();
                    break;

                case 2: // Pot Tab
                    UpdateTimerEvent(pot_tick);
                    appBoard.tempTabClosed();
                    break;

                case 3: // Temp Tab
                    x = 0;
                    integral = 0;
                    prev_error = 0;
                    appBoard.WriteHeat("on");
                    UpdateTimerEvent(temp_tick);
                    break;
                }
            }
            else
            {
                if(tabControl1.SelectedIndex != 0)
                {
                    MessageBox.Show("Port is not open");
                    tabControl1.SelectTab(0);
                }
            }
        }

        // SETUP PAGE //

        private void setupCombos() // sets default baud rate to 38400
        {
            baudRateDropdown.SelectedIndex = 2;
        }

        private void connectButton_Click(object sender, EventArgs e) //connection button
        {
            try
            {
                int baudint = 0;
                Int32.TryParse(baudRateDropdown.Text, out baudint);

                //disable UI to avoid double-clicking while connecting
                connectButton.Enabled = false;

                //trigger async connect with a callback
                appBoard.Connect(comPortDropdown.Text, baudint, () =>
                {
                    //called from background thread, so invoke UI-safe changes
                    if (InvokeRequired)
                    {
                        Invoke(new Action(UIConnectResponse));
                    }
                    else
                    {
                        UIConnectResponse();
                    }
                });
            }
            catch (Exception ex)
            {
                MessageBox.Show($"Error Connecting: {ex.Message}");
                connectButton.Enabled = true;
            }
        }

        private void UIConnectResponse()
        {
            if(appBoard.checkTx()== 0x0F)
            {
                disconnectButton.Enabled = true;
                connectButton.Enabled = false;
                serialPortStatusLED.On = true;
            }
            else
            {
                MessageBox.Show($"Error Connecting: Wrong byte recieved - Check COMPort.");
                connectButton.Enabled = true;//turn back on if txcheck fails
```

```
                disconnectButton.Enabled = false;
            }
        }

        private void disconnectButton_Click(object sender, EventArgs e)
        {
            appBoard.Disconnect(); //use AppBoard to disconnect
            //turn stuff on/off accordingly
            disconnectButton.Enabled = false;
            connectButton.Enabled = true;
            serialPortStatusLED.Enabled = false;
            serialPortStatusLED.On = false;
        }

        //    SQL STUFF     //

        private void datBCon_Click(object sender, EventArgs e)
        {
            string uid = usernameBox.Text;
            string pass = passwordBox.Text;
            string address = addressBox.Text;
            string database = datBasBox.Text;

            string connString = $"SERVER={address};PORT=3306;DATABASE={database};UID={uid};PASSWORD={pass};";
            try
            {
                conn = new MySqlConnection();
                conn.ConnectionString = connString;
                conn.Open();
                if (conn.State == ConnectionState.Open)
                {
                    datBasStatLED.On = true;
                    datBDis.Enabled = true;
                    datBCon.Enabled = false;
                    MessageBox.Show("Connection Successful");
                }
            }
            catch(MySqlException ex)
            {
                MessageBox.Show(ex.Message);
            }

        }

        private void datBDis_Click(object sender, EventArgs e)
        {
            if (conn != null)
            {
                try
                {
                    conn.Close();
                    if (conn.State == ConnectionState.Closed)
                    {
                        datBasStatLED.On = false;
                        datBDis.Enabled = false;
                        datBCon.Enabled = true;
                    }
                }

                catch (MySqlException ex)
                {
                    MessageBox.Show(ex.Message);
                }
            }
        }

        // DIGI IO PAGE //

        public byte getPORTCValue()
        {
            byte lsb = 0x00;//initialise least significant byte to send

            //check which boxes are ticked and assign accordingly
            if (tickPC0.Checked) lsb |= 0b00000001;
            if (tickPC1.Checked) lsb |= 0b00000010;
            if (tickPC2.Checked) lsb |= 0b00000100;
            if (tickPC3.Checked) lsb |= 0b00001000;
            if (tickPC4.Checked) lsb |= 0b00010000;
            if (tickPC5.Checked) lsb |= 0b00100000;
            if (tickPC6.Checked) lsb |= 0b01000000;
            if (tickPC7.Checked) lsb |= 0b10000000;
```

```
            return lsb; //return lsb
        }

        public void setPINALEDs(byte PINAVal)
        {
            //set leds according to switches flipped on board
            pa0LED.Color = (PINAVal & (1 << 0)) != 0 ? Color.Red : Color.Black;
            pa1LED.Color = (PINAVal & (1 << 1)) != 0 ? Color.Red : Color.Black;
            pa2LED.Color = (PINAVal & (1 << 2)) != 0 ? Color.Red : Color.Black;
            pa3LED.Color = (PINAVal & (1 << 3)) != 0 ? Color.Red : Color.Black;
            pa4LED.Color = (PINAVal & (1 << 4)) != 0 ? Color.Red : Color.Black;
            pa5LED.Color = (PINAVal & (1 << 5)) != 0 ? Color.Red : Color.Black;
            pa6LED.Color = (PINAVal & (1 << 6)) != 0 ? Color.Red : Color.Black;
            pa7LED.Color = (PINAVal & (1 << 7)) != 0 ? Color.Red : Color.Black;
        }

        public void displayByteInHex(byte value)
        {
            // split the byte into two 4-bit nibbles
            byte highNibble = (byte)(value >> 4);  // most significant nibble
            byte lowNibble = (byte)(value & 0x0F); // Least significant nibble

            // convert each nibble to hex
            char highChar = hexToChar(highNibble);
            char lowChar = hexToChar(lowNibble);

            // sevenSegment to display the hex digits
            sevenSegment1.Value = highChar.ToString();
            sevenSegment2.Value = lowChar.ToString();
        }

        private char hexToChar(byte nibble)
        {
            if (nibble >= 0 && nibble <= 9)
                return (char)('0' + nibble); // 0-9 are directly mapped
            else
                return (char)('A' + (nibble - 10)); // A-F for nibbles 10-15
        }

        void digIO_Tick(object sender, EventArgs e)
        {
            try
            {
                appBoard.WritePORTC(getPORTCValue());// button checks sent to board to display on portc
                displayByteInHex(getPORTCValue());//display on seven segment
                byte PINAVal = appBoard.ReadPINA(); // get which switches are flipped
                setPINALEDs(PINAVal);//set appropriately
            }

            catch (Exception ex)
            {
                appTimer.Stop();
                tabControl1.SelectTab(0);
                MessageBox.Show($"Error: {ex.Message}");
            }
        }

                // POT TAB //

        void potentiometres()
        {
            float pot1DispVal = (5 * (float)appBoard.ReadPotV(0)) / 255;
            pot1VoltageDisplay.Value = pot1DispVal;
            float pot2DispVal = (5 * (float)appBoard.ReadPotV(1)) / 255;
            pot2VoltageDisplay.Value = pot2DispVal;
            pot2VoltageDisplay.RecommendedValue = pot2DispVal;
            pot1VoltageDisplay.RecommendedValue = pot1DispVal;
        }

        private void lampIntensityScroll_Scroll(object sender, ScrollEventArgs e)
        {
            lampPercentDisplay.Text = (-lampIntensityScroll.Value).ToString() + "%";
        }

        void lightSensor()
        {
            int lightDispVal = appBoard.ReadLight();
            lightDisplay.Value = lightDispVal;
            lightDisplay.RecommendedValue = lightDispVal;
        }
```

```
        void lampScroll()
        {
            byte[] value = BitConverter.GetBytes(-399*(lampIntensityScroll.Value)/100);
            appBoard.WriteLamp(value);
        }

        void pot_tick(object sender, EventArgs e)
        {
            try
            {
                potentiometres();
                lampScroll();
                lightSensor();
            }
            catch (Exception ex)
            {
                appTimer.Stop();
                tabControl1.SelectTab(0);
                MessageBox.Show($"Error: {ex.Message}");
            }
        }

                // TEMP TAB //

        double Clamp(double value, double min, double max)
        {
            if (value < min) return min;
            if (value > max) return max;
            return value;
        }

        void piLogic(double desiredTemp, int kp, int ki)
        {
            double Ts = 0.1;

            double currentTemp = appBoard.ReadTemp();

            //need to convert from bytes to actual degrees
            currentTemp = (currentTemp / 255)*5;
            currentTemp = currentTemp / 0.05;

            double error = currentTemp - desiredTemp;

            double u = kp * error + ki * integral;

            if ((u > -20 && u < 180) || ((u < -20) && error < 0) || ((u > 180) && error > 0)) //integral windup handling
      , -20 to 180 virtual clamp. implemeneted due to oscillation
            { integral += (Ts / 2) * (error + prev_error); } //bilinear rather than backwards euler. I changed this from
       the design spec as it was highly oscillatory.

            //recalculate, clamp again, normalise to 0 to 1 and then multiply by 399 to get pwm range
            u = kp * error + ki * integral;
            u = ((Clamp(u, -20, 180) + 20) / 200) * 399;


            //send to MCU
            ushort pwmValue = (ushort)u;
            byte[] pwmBytes = { (byte)(pwmValue & 0xFF), (byte)((pwmValue >> 8) & 0xFF) };
            appBoard.WriteFan(pwmBytes);

            double displayValue = (u / 399) * 100;//normalise for percentage ouput
            motorSpeedDisplay.Text = $"{displayValue:F2}%";
            actualTempDisplay.Text = $"{currentTemp:F2} [C]";

            prev_error = error;

            //GRAPH STUFF
            tempChart.Series[0].Points.AddXY(x++, currentTemp);

            var yStripLine = new StripLine();//line to show desired temp
            yStripLine.Interval = 0;
            yStripLine.StripWidth = 0;
            yStripLine.BackColor = Color.Red;
            yStripLine.BorderWidth = 1;
            yStripLine.BorderColor = Color.Red;
            yStripLine.BorderDashStyle = ChartDashStyle.Dash;
            yStripLine.IntervalOffset = desiredTemp;

            //clear previous striplines on tick, faster than check if there's a change in temp_desired
            tempChart.ChartAreas[0].AxisY.StripLines.Clear();
```

```
            tempChart.ChartAreas[0].AxisY.StripLines.Add(yStripLine);

            //find the max value between series and desiredTemp to set graph limits
            double maxSeriesValue = tempChart.Series[0].Points.Count > 0
                ? tempChart.Series[0].Points.Max(p => p.YValues[0])
                : 0;

            double yMax = Math.Max(maxSeriesValue, desiredTemp) + 5;
            tempChart.ChartAreas[0].AxisY.Maximum = yMax;
        }

        void temp_tick(object sender, EventArgs e)
        {
            try
            {
                double desiredTemp = (double)setpointTemp.Value;
                int kp = (int)kpTuning.Value;
                int ki = (int)kiTuning.Value;
                piLogic(desiredTemp, kp, ki);

                if (shouldLogData)
                {
                    double currentTemp = appBoard.ReadTemp();
                    currentTemp = (currentTemp / 255) * 5;
                    currentTemp = currentTemp / 0.05;
                    currentTemp = Math.Round(currentTemp, 2);
                    DateTime currentTime = DateTime.Now;
                    string formattedTime = currentTime.ToString("yyyy-MM-dd HH:mm:ss.fff");
                    string username = usernameBox.Text;

                    // Insert data to the database
                    addDataToDatabase(currentTemp, formattedTime, username);
                }
            }
            catch (Exception ex) {
                appTimer.Stop();
                tabControl1.SelectTab(0);
                MessageBox.Show($"Error: {ex.Message}");
            }
        }

        //   DATA LOGGING   //

        private void insDat_Click(object sender, EventArgs e)
        {
            string query = "INSERT INTO temperature (remark) VALUES (@data)";
            string data2Send = manualData.Text;
            try
            {
                if(this.conn != null)
                {
                    using (MySqlCommand cmd = new MySqlCommand(query, conn))
                    {
                        cmd.Parameters.AddWithValue("@data", data2Send);
                        cmd.ExecuteNonQuery();
                        MessageBox.Show("Remark Added");
                    }
                }
                else
                {
                    MessageBox.Show("Database not Connected");
                }
            }
            catch(MySqlException ex)
            {
                MessageBox.Show(ex.Message);
            }
        }

        private void enbDatLog_Click(object sender, EventArgs e)
        {
            if (this.conn != null)
            {
                shouldLogData = !shouldLogData;
                enbDatLog.Enabled = false;
                disDatLog.Enabled = true;
            }
            else
            {
                MessageBox.Show("Database not Connected");
            }
```

```
        }

        private void disDatLog_Click(object sender, EventArgs e)
        {
            shouldLogData = !shouldLogData;
            enbDatLog.Enabled = true;
            disDatLog.Enabled = false;
        }

        public void addDataToDatabase(double currentTemp, string currentTime, string username)
        {
            string query = "INSERT INTO temperature (timeStamp, temp, remark) VALUES (@timedata, @tempdata, @userdata)";
            try
            {
                if (this.conn != null)
                {
                    using (MySqlCommand cmd = new MySqlCommand(query, conn))
                    {
                        cmd.Parameters.AddWithValue("@timedata", currentTime);
                        cmd.Parameters.AddWithValue("@tempdata", currentTemp);
                        cmd.Parameters.AddWithValue("@userdata", username);
                        cmd.ExecuteNonQuery();
                    }
                }
                else
                {
                    shouldLogData = !shouldLogData;
                    enbDatLog.Enabled = true;
                    disDatLog.Enabled = false;
                    MessageBox.Show("Database not Connected");
                }
            }
            catch (MySqlException ex)
            {
                shouldLogData = !shouldLogData;
                enbDatLog.Enabled = true;
                disDatLog.Enabled = false;
                MessageBox.Show("MySQL Error: " + ex.Message);
            }
            catch (Exception ex)
            {
                shouldLogData = !shouldLogData;
                enbDatLog.Enabled = true;
                disDatLog.Enabled = false;
                MessageBox.Show("Error: " + ex.Message);
            }
        }
    }
}
```

7.2. AppBoard.cs File

```
using System;
using System.IO.Ports;
using System.Threading;
using System.Windows.Forms;


namespace gui
{
    internal class AppBoard
    {
        public SerialPort serialPort;
        byte startByte = 0x53;
        byte stopByte = 0xAA;

        public AppBoard()
        {
            serialPort = new SerialPort();
        }

        public string[] getCOMs()
        {
            return SerialPort.GetPortNames();
        }

        public void Connect(string portName, int baudRate, Action onConnected)
        {
            // Create a background thread to open the serial port to stop jumping to next task before connection is made
```

```
            ThreadPool.QueueUserWorkItem(state =>
            {
                try
                {
                    serialPort.Parity = Parity.None;
                    serialPort.DataBits = 8;
                    serialPort.StopBits = StopBits.One;
                    serialPort.ReadTimeout = 1000;
                    serialPort.PortName = portName;
                    serialPort.BaudRate = baudRate;
                    serialPort.Open();  // Try to open the port
                    int timeoutMs = 3000; // Wait up to 3 seconds
                    int intervalMs = 100;
                    int waited = 0;

                    while (!serialPort.IsOpen && waited < timeoutMs)
                    {
                        Thread.Sleep(intervalMs);
                        waited += intervalMs;
                    }

                    if (serialPort.IsOpen)
                    {
                        onConnected?.Invoke();
                    }
                    else
                    {
                        MessageBox.Show($"Failed to open {portName} within timeout.", "Connection Timeout", MessageBoxButtons.OK, MessageBoxIcon.Warning);
                    }
                }
                catch (UnauthorizedAccessException ex)
                {
                    Disconnect();
                    MessageBox.Show($"Error opening {portName}: {ex.Message}", "Access Denied", MessageBoxButtons.OK, MessageBoxIcon.Error);
                }
                catch (Exception ex)
                {
                    Disconnect();
                    MessageBox.Show($"Error opening {portName}: {ex.Message}", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
                }
            });
        }

        public void Disconnect()
        {
            if (serialPort.IsOpen)
            {
                serialPort.Close();
            }
        }

        public int checkTx()
        {
            byte[] checkConnect = { startByte, 0x00, stopByte };
            try
            {
                serialPort.Write(checkConnect, 0, checkConnect.Length);
                return serialPort.ReadByte();
            }
            catch (TimeoutException)
            {
                Disconnect();
                return -1;
            }
            catch(Exception)
            {
                Disconnect() ;
                return -2;
            }
        }

        public byte ReadPINA()
        {
                byte[] ReadPINA = { startByte, 0x01, stopByte };
                serialPort.Write(ReadPINA, 0, ReadPINA.Length);
                int response = serialPort.ReadByte();
                return (byte)response;
        }
```

```
        public void WritePORTC(byte lsb)
        {
                byte chaff = 0xff;
                byte[] WritePORTC = { startByte, 0x0A, lsb, chaff, stopByte };
                serialPort.Write(WritePORTC, 0, WritePORTC.Length);
                int response = serialPort.ReadByte();
        }

        public double ReadPotV(int pot)
        {
            byte[] readPOT = null;
            switch (pot)
            {
                case 0:
                    readPOT = new byte[] { startByte, 0x02, stopByte };
                    break;
                case 1:
                    readPOT = new byte[] { startByte, 0x03, stopByte };
                    break;
            }


            serialPort.Write(readPOT,0,readPOT.Length);
            int response = serialPort.ReadByte();

            return (double)response;
        }

        public int ReadLight()
        {
            byte[] readLight = { startByte, 0x05, stopByte };
            serialPort.Write(readLight, 0, readLight.Length);
            return serialPort.ReadByte();
        }

        public void WriteLamp(byte[] value)
        {
            byte lsb = value[0];
            byte msb = value[1];
            byte[] WriteLamp = { startByte, 0x0C, lsb, msb, stopByte };
            serialPort.Write(WriteLamp, 0, WriteLamp.Length);
            int response = serialPort.ReadByte();
        }

        public void tempTabClosed()
        {
            //turn off temp/fan stuff
            byte[] pwmBytesOff = { (byte)(0 & 0xFF), (byte)((0 >> 8) & 0xFF) };
            WriteHeat("off");
            WriteFan(pwmBytesOff);
        }

        public double ReadTemp()
        {
            byte[] readTemp = { startByte, 0x04, stopByte };
            serialPort.Write(readTemp, 0, readTemp.Length);
            return (double)serialPort.ReadByte();
        }
        public void WriteFan(byte[] value)
        {
            byte lsb = value[0];
            byte msb = value[1];
            byte[] Writefan = { startByte, 0x0D, lsb, msb, stopByte};
            serialPort.Write(Writefan, 0, Writefan.Length);
            int response = serialPort.ReadByte();
            if (response != 0x0D)
            {
                MessageBox.Show("Wrong return (Fan)");
            }
        }

        public void WriteHeat(string value)
        {
            byte lsb = 0;
            byte msb = 0;
            if (value == "on")
            {
                lsb = 255;
                msb = 255;
            }
```

```
            byte[] Writeheat = { startByte, 0x0B, lsb, msb, stopByte };
            serialPort.Write(Writeheat, 0, Writeheat.Length);
            int response = serialPort.ReadByte();
            if(response!= 0x0B)
            {
                MessageBox.Show("Wrong return (Heat)");
            }
        }
    }
}
```

7.3. MCU Code

```
#define F_CPU 8000000UL
#include <avr/io.h>
#include <util/delay.h>
#include <avr/interrupt.h>

#define startConversion ADCSRA |= (1<<6) //initiates conversion
#define conversionRunning ADCSRA & (1<<6) //bit is cleared on conversion complete

char *receiveByte;

//cases for isr
#define initialise 0
#define instruct 1
#define logLSB 2
#define logMSB 3
#define checkstop 6
unsigned char mode;
unsigned char readorwrite;
char startbyte = 0x53;
char stopbyte = 0xAA;

//data information
unsigned char instruction;
unsigned char lsb = 0b00;
unsigned char msb = 0b00;
int sixteenbit;

//define channels
#define p02 0b01100001
#define p01 0b01100010
#define thermometre 0b01100011
#define lightsensor 0b01100000


//enable reading inputs from selected channel and outputs completed conversion
char readADC(char channel)
{
  ADMUX = channel;
  startConversion;

  while(conversionRunning);

  return(ADCH);
}

/////////READ FUNCTIONS BEGIN //////////

void read(unsigned instruction){//read functions
  switch(instruction){
    case 0x00:
    txCheck();
    break;
    case 0x01:
    readPINA();
    break;
    case 0x02:
    readPOT1();
    break;
    case 0x03:
    readPOT2();
    break;
    case 0x04:
    readTemp();
    break;
    case 0x05:
    readLight();
```

```
      break;
  }
}

void txCheck(){
  UDR1 = 0x0F;
}

void readPINA(){
  UDR1=~PINA;
}

void readPOT1(){
  UDR1 = readADC(p01);
}

void readPOT2(){
  UDR1 = readADC(p02);
}

void readTemp(){
  UDR1 = readADC(thermometre);
}

void readLight(){
  UDR1 = readADC(lightsensor);
}


/////////WRITE FUNCTIONS BEGIN //////////

void write(unsigned char instruction, unsigned char lsb, int sixteenbit){ //write functions
  switch(instruction){
    case 0x0A:
    setPORTC(lsb);
    break;
    case 0x0B:
    setHeater(sixteenbit);
    break;
    case 0x0C:
    setLight(sixteenbit);
    break;
    case 0x0D:
    setMotor(sixteenbit);
    break;
  }
}

void setPORTC(unsigned char lsb){
  //write the 8 LSB to PORTC, return 0x0A over udr1
  PORTC = lsb;
  UDR1 = 0x0A;
}

void setHeater(int sixteenbit){
  OCR1C = sixteenbit;
  UDR1 = 0x0B;
}

void setLight(int sixteenbit){
  OCR1B = sixteenbit;
  UDR1 = 0x0C;
}

void setMotor(int sixteenbit){
  OCR1A = sixteenbit;
  UDR1 = 0x0D;
}

////////INTERRUPT//////////

ISR(USART1_RX_vect){ //receive interrupt
  *receiveByte = UDR1;//pointer to current byte

  switch(mode){

    case initialise:
      //reset variables
      instruction = 0;
      readorwrite = 2;
      lsb = 0;
```

```
      msb = 0;

      if(*receiveByte == startbyte){ //if the received byte is 0x53
        mode =  instruct; //standby for instruction byte
      }
    break;

    case instruct:
      instruction = *receiveByte; //save as instruction
      if(*receiveByte<0x0A){ //if received byte is read instruction

        readorwrite = 0; //select the right function
        mode = checkstop; //check if stop byte received
      }

      else{
        mode = logLSB; //otherwise jump to logging least significant bit for writing
      }

      break;

    case logLSB:// log lsb
      lsb = *receiveByte;
      mode = logMSB;
      break;

    case logMSB://log msb and check stop byte received
      msb = *receiveByte;
      sixteenbit = lsb+(msb<<8);//fixed was msb+lsb
      readorwrite = 1;
      mode = checkstop;
      break;

    case checkstop:
      if(*receiveByte == stopbyte){// if the stop byte is received continue
        switch(readorwrite){
        case 0:
          read(instruction);//pass instruction to function
          mode = initialise;//return to waiting state
          break;

        case 1:
          write(instruction, lsb, sixteenbit);//pass instruction, lsb and msb to function
          mode = initialise;//return to waiting state
          break;

        default://error handling
          mode = initialise;
          break;
        }
      }

      else{//if no stop byte received, reset
        mode = initialise;
      }
      break;
  }
}

//////////SETUP//////////

void setup(){
  DDRC = 0xFF;//all leds as output
  DDRB = 0b11100000;//fan/lamp/heater as output

  //start off
  PORTC = 0x00;
  PORTB = 0x00;

  //Setup board to use switches
  DDRE = 3;
  PORTE = 0b00000000;
  DDRA = 0;
  //conversion
  ADCSRA = 0b10000111;

  //pwm settings
  TCCR1A = 0b10101010;
  TCCR1B = 0b00011001;
  ICR1 = 399;

```

```
  cli();
  sei();

  UCSR1B = 0b10011000; //usart tx/rx on, enable receive complete interrupt
  UCSR1C = 0b00000110;//async no parity bit, 1 stop bit, 8 bit char size
  UBRR1L = 12; //38400 buad

  //start with certain conditions
  mode = initialise;
  readorwrite = 2;
}

/////////MAIN/////////

int main(void)
{
  setup();
    while (1){}
}
```