**ENEL712 Embedded Systems Design—Project**

**Final Report Due Date: Friday, 6 June 2025, 4pm**

This project is part of the overall course assessment, and it will contribute 40% towards the final grade. The following general points apply to all work handed in for assessment:

* The assignment should be completed on a word processor such as Microsoft Word with suitably commented C/C++/C# code included.
* The AUT cover sheet should be signed and dated by you, and submitted together with the report in a single PDF file.
* Two files are required for online submission through Canvas:

1. PDF file for the project report with naming convention: ENEL712-project-report-ST19xxxxx8.pdf (replace the red number part with your student ID).
2. .ZIP file for the project code (you can use 7-zip to zip a folder that includes two subfolders--one for microcontroller code project, the other for GUI the visual studio project). The .zip file should be named as ENEL712-project-code-ST19xxxxx8.zip (replace the red number part with your student ID).

* You are allowed to discuss the problems with your colleagues or AI tools (e.g., ChatGPT, Deepseek, Grok, etc.), but you must hand in your own work.

Overview

Suppose you are working as an Embedded System Engineering at Ideal IoT Incorporation, and you are assigned to develop a C# Graphical User Interface (GUI) for controlling the functionality of the AUT AT90USB Applications Board from a PC. The GUI will be made up of several tabs, each of which will control different aspects of the applications board. The application will communicate with the applications board via a USB Serial port, and it will also communicate with a database server (set up by the software Xampp) through Internet connections.

Note: Software will have to be written for both the AT90USB MCU and the PC. It is suggested you start with the MCU code, and then move on to write the PC code.

GUI Specifications

As mentioned above, each tab contains different functionality. The subsections below describe the requirements of each tab:

**1. Part A**

Setup Tab [Part A]

The Setup tab allows the user to select the USB-Serial COM Port and Baud Rate via two combo boxes. The application should automatically populate the COM Port combo box with the COM ports available on the target PC (see [here](https://msdn.microsoft.com/en-us/library/system.io.ports.serialport.getportnames(v=vs.110).aspx)), while the Baud Rates should be fixed to the options [9600,19200,38400,57600,115200], defaulting to 38400.

As referred to Figure 1, when the user presses the Connect Button, the application should open the selected serial port within C#, ensure communication with the applications board is working, then turn the Serial Port Status LED from black to green (see LEDBulb below) if communication was successful. Details on the serial port and ensuring communication is working is presented later.

Similarly, when the user presses the Database Connect Button, the application should set up the database connection to database specified by the user with the provided log in details, such as database server name, username, and password (note that the password will be hidden as a user enters the password). Once the database connection is established, the Database Connection Status LED will be turned on from black to green (see LEDBulb below).

Graphical user interface, application

Description automatically generated Graphical user interface, application

Description automatically generated

Figure . Setup Tab Functionalities

Digital I/O Tab [Part A]

Graphical user interface, application, table

Description automatically generated Graphical user interface

Description automatically generated

Figure . Digital I/O Tab Functionalities

As referred to Figure 2, the Digital I/O tab allows your program to read each of the PORTA pins (switches on the applications board), and write to the LEDs on PORTC. To provide a more interesting UI, two new third-party UI controls are to be used:

LEDBulb.cs:  
<http://www.codeproject.com/Articles/114122/A-Simple-Vector-Based-LED-User-Control>

SevenSegment.cs:  
<http://www.codeproject.com/Articles/37800/Seven-segment-LED-Control-for-NET>

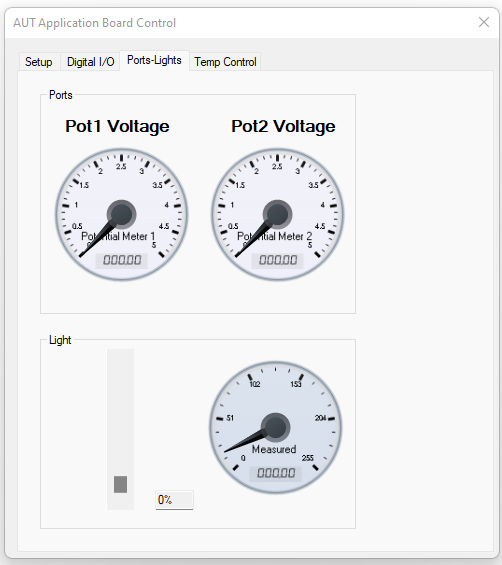
Both the above projects are available on Canvas. Simply add the .cs files to your project (Add->Existing File), rebuild the solution, and they will be available from the Toolbox to drop onto your Form.

When the refresh button is pushed:  
1) The current value of PORTC (read from the checkboxes) is to be displayed on the two GUI seven segment displays.  
2) The current value of PORTC is to be transmitted to the MCU board, to be displayed on the actual LEDs.  
3) The current value of PINA is to be read from the MCU board, and displayed on the GUI LEDs.

For initial set up, you may place a Refresh button on the Tab to trigger the data exchange. For the second submission, this must be done using a timer, presented later.

**2. Part B**

Pots Tab [Part B]

 Diagram

Description automatically generated

Figure . Ports-Light Tab Functionalities

The Pots (Potentiometer) tab allows the user to view the current *Voltage* selected by each of the pots on the applications board. The voltage is to be displayed using two third-party gauges:  
<http://www.codeproject.com/Articles/20341/Aqua-Gauge>

As before, the above project is available on Canvas. Add the two .cs files and .resx file to your project, rebuild, and you will have them available in the toolbox.

You may wish to have a Refresh button to update the voltage manually (1st submission only), or use a timer, described later on.

When the refresh button is pushed:  
1) The current value of the ADC connected to each Pot is read from the MCU board (as an 8 bit integer)  
2) Within C#, the reading is converted to a voltage  
3) Each voltage is displayed on its respective gauge.

The light tab allows the user to adjust the lamp brightness by dragging a vertical scroll bar up and down. In addition, the current light level is to be measured (using the ADC) and displayed on a gauge, similar to the ones used in the Pots tab.

You may wish to have a Refresh button to update the light level manually (initial setup), or use a timer, described later on.

When the refresh button is pushed:  
1) The scroll bar value is read and sent to the MCU, which then adjusts the lamp brightness accordingly.  
2) The current value of the ADC connected to the light sensor is read from the MCU board.  
3) The ADC value received is displayed on the gauge.

Temp Control Tab [Part B]

The final tab implements a temperature PI controller with real-time plotting and tuning. The user selects a setpoint temperature, PI tuning constants, and the heater is turned on to 100%. The motor speed is then varied to control the temperature to the desired value, and the measured data is plotted. Details of the PI controller are given later in this document. It allows the user to upload temperature reading (from the AUT Application Board) to the database server, either manually (using the Insert Data to Table button) or automatically (using the Enable Data Logging button). It also allows the user to stop data logging by clicking the Stop Data Logging button.

Chart, box and whisker chart

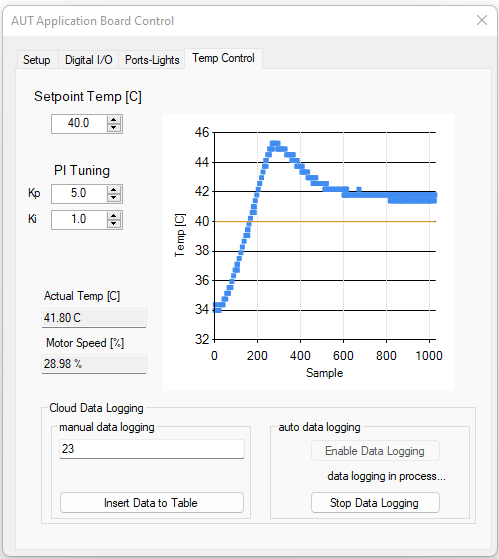
Description automatically generated 

Figure . Temperature Control Tab Functionalities

**3. Part C—Xampp configuration**

Database Server Configuration [Part C]

[Xampp](https://www.apachefriends.org/) is available for the PCs in WZ412 and WZ413. For the Database Server side, please use a website browser (Microsoft Edge, Google Chrome, Mozilla Firefox, etc.) and enter the URL: <http://localhost/phpmyadmin/>, log in using username **root** and the default password is blank. It allows you to configure the [SQL database server](https://www.youtube.com/watch?v=M6JJWrz2J9A). First, add a user account with username **ST123456**, leave the Host name as ‘Any Host’ like % (this will enable the user to use any computer to log in the server), and enter the respective password (alternatively, you can use the ‘generate password’ option to generate the password) for the user ST123456. Remember this password as later you will need it for the GUI programme part.

Following the steps in Figures 5-9 to configure the database server.

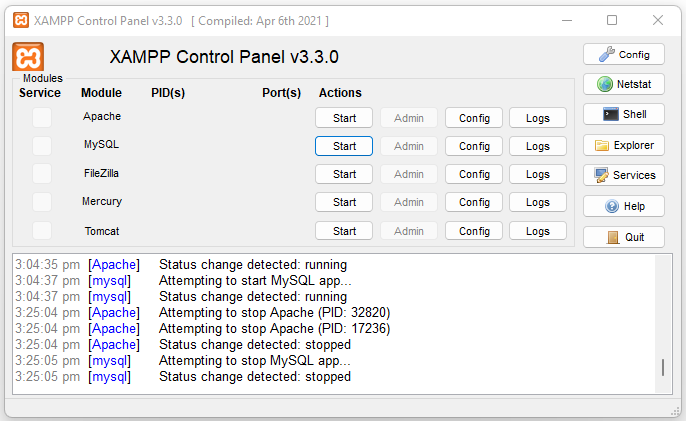


Figure . Xampp Control Panel

Graphical user interface, application

Description automatically generated

Figure . Xampp Control Panel with Apache and MySQL servers started

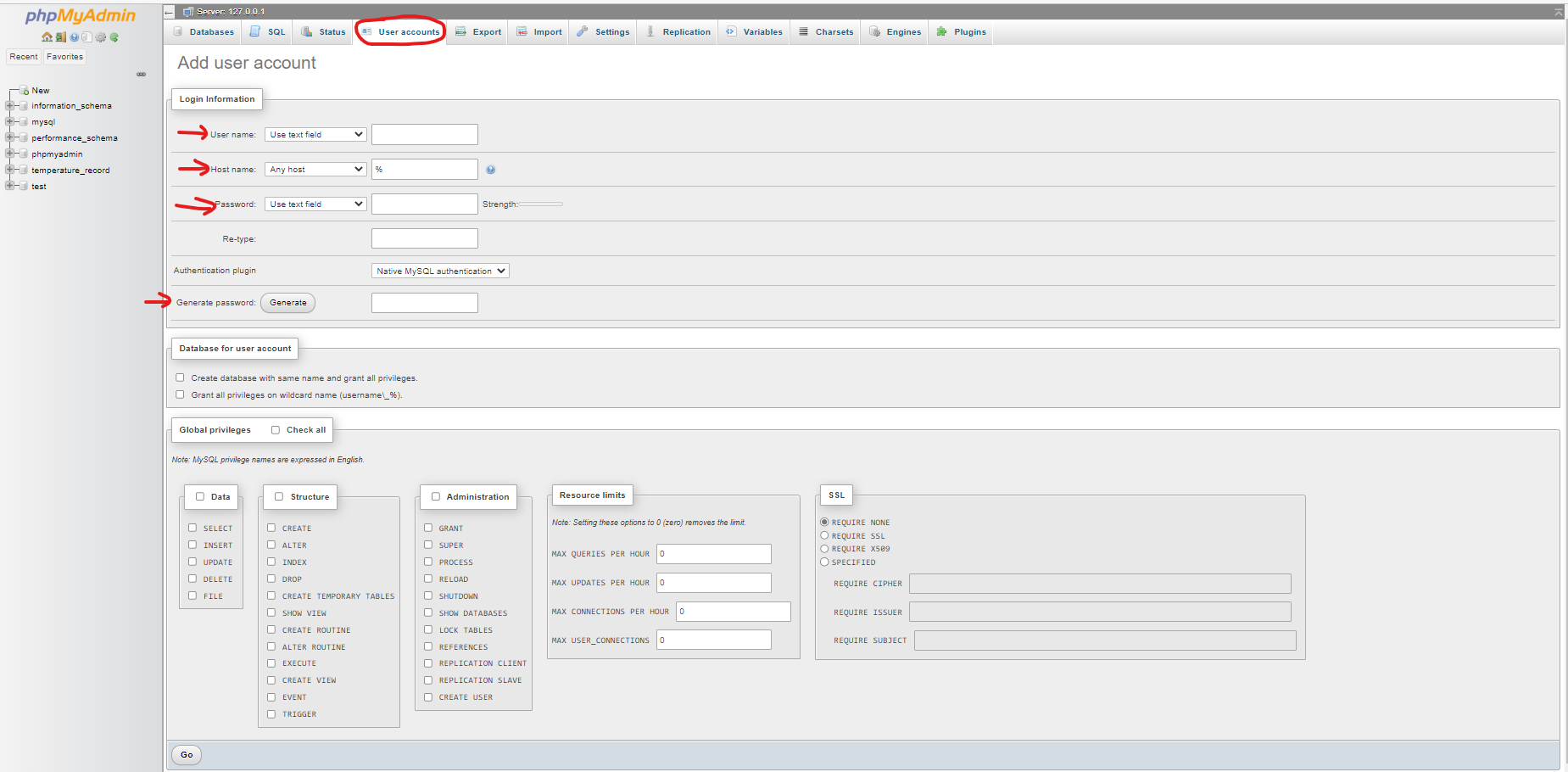


Figure . phpMyAdmin Panel—adding user accounts

Then, through phpMyAdmin, go to the database tab, and create a database with a name ‘**temperature\_record**’. On the left-hand panel of phpMyAdmin, click on the created database **temperature\_record**, you can create a new table with table name ‘**temperature**’, which should have three columns, namely *timeStamp*, *temperature* and *remark*. The data types of these three columns are *varchar(50)*, *varchar(20)* and *text*.

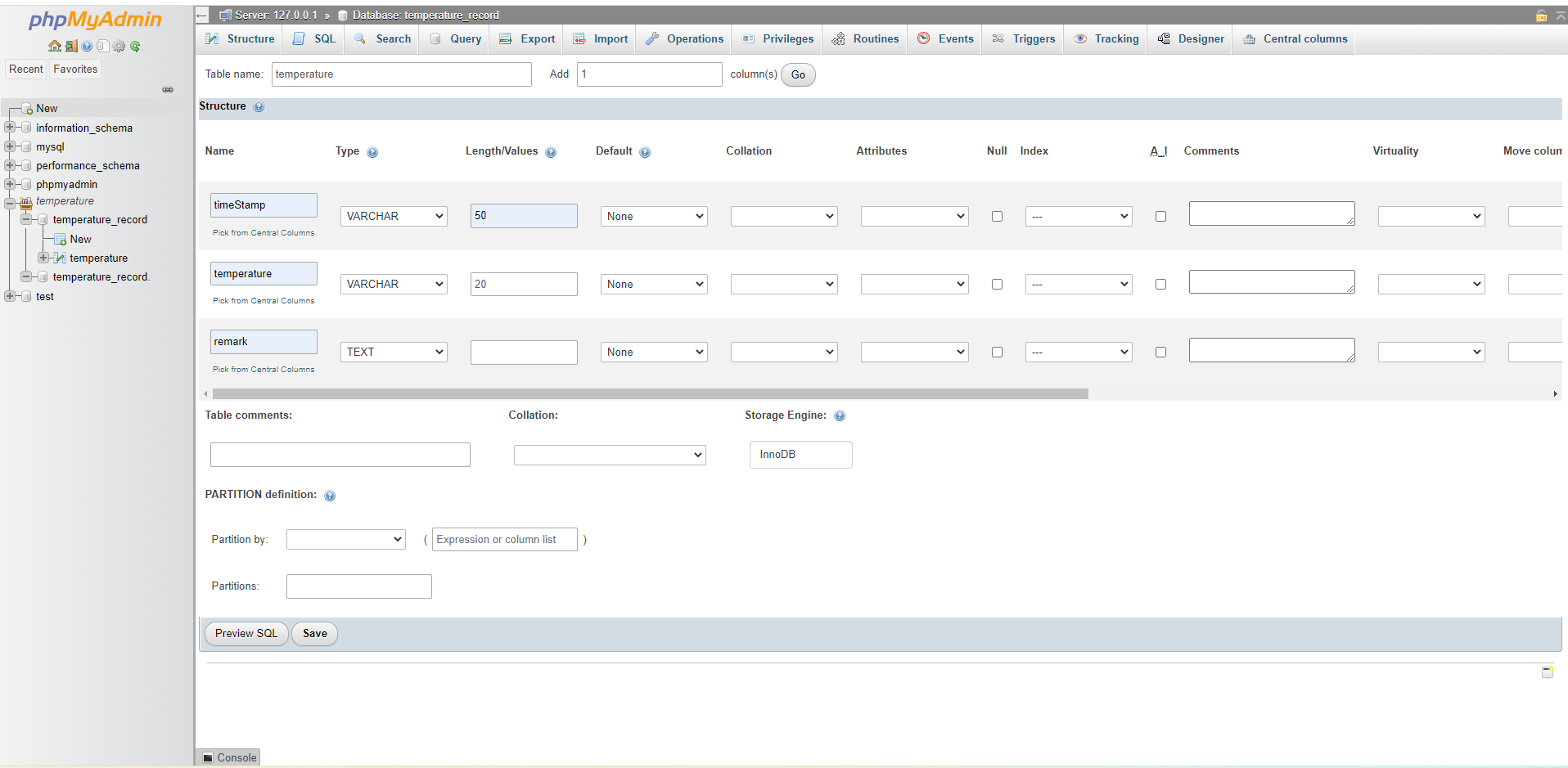


Figure . phpMyAdmin Panel—adding table

You can grant user ST12346 with the corresponding database access privilege here (user accounts🡪edit privileges🡪select database🡪change the access privilege for a database).

Graphical user interface, application

Description automatically generated

Figure . phpMyAdmin Panel—setting database access privilege

Figure 10 illustrates the table with uploaded temperature data from the GUI developed in this Project.

Graphical user interface, text, application, email

Description automatically generated

Figure . phpMyAdmin Panel—database showing uploaded temperature reading data

**4. Specifications and Guidelines**

MCU Specifications

The program on the application board microcontroller will be written in C using Atmel Studio. It has the following responsibilities/tasks:

Digital I/O

* Reading the value of PINA
* Writing the value of PORTC

Analogue I/O

* Using the ADC to take voltage measurements of the following sensors:
  + Light sensor
  + Temperature sensor (50mV/°C)
  + Pot1 wiper (AVCC = 5V)
  + Pot2 wiper (AVCC = 5V)
* Each measurement will return the 8 MSB of the ADC.
* Store the measurement as an unsigned char, the conversion to voltage will be done in C#.

Timer

* Using Timer 1 to control the PWM duty cycle of the following peripherals:
  + Motor (OCR1A)
  + Lamp (OCR1B)
  + Heater (OCR1C)
* Timer 1 should be set up to use Fast PWM with a TOP value of ICR1, with ICR1 set to 399. This will provide a PWM frequency of 20kHz, when a prescaler of 1 is used. The OCR values accepted will therefore range from 0-399, thus a 16 bit integer will be required to set them.

Serial Port

* UART1 should be used to receive instructions and data from the C# GUI, and transmit measurements back to the C# GUI.
* Use an interrupt with state machine to decode the information being received over the serial port. See the communication protocol below for details on the state machine.
* The serial port should be setup with a Baudrate of 38400 and 8N1.

The MCU application will continuously poll to see whether an instruction has been received by the serial port, and if so, respond accordingly, before waiting for the next instruction.

Communication Protocol

To communicate between the PC and the MCU, we will use a custom protocol on top of the Serial Port. It has two forms:

Sending an Instruction from the PC to the MCU (e.g. read PINA):

Data

0xAA

INS

0x53

Start Byte Instruction Byte Stop Byte Received Measurement Byte from MCU

PC Rx – MCU Tx

PC Tx - MCU Rx

Sending an Instruction and a 16bit Integer from the PC to the MCU (e.g. write Motor Speed):

INS

0xAA

Data[1]

Data[0]

0x53

INS

Start Byte Instruction Byte LS Byte MS Byte Stop Byte Instruction sent Back from MCU to Ack.

PC Rx – MCU Tx

PC Tx - MCU Rx

Typical communication scenarios will include:

Reading PINA (viewed from the PC)

1. 0x53 is sent (start byte)
2. 0x01 is sent (instruction to read PINA, see table below)
3. 0xAA is sent (stop byte)
4. PINA data is received as one byte from the MCU

Reading PINA (viewed from the MCU)

1. A byte is received by the UART, if its 0x53, move to the instruction state
2. A byte is received by the UART, if it is a measurement instruction, save it and move to the stop state
3. A byte is received by the UART, if it is 0xAA, indicate to the while(1) that an instruction is ready for processing, and return to the start state
4. The while(1) detects an instruction is ready, processes the instruction, and sends back the required information (e.g. UDR1 = PINA, in this case).

Writing the Motor Speed (viewed from the PC)

1. 0x53 is sent (start byte)
2. 0x0D is sent (instruction to write the motor speed, see below)
3. LS Byte of 16 bit integer containing speed is sent
4. MS Byte of 16 bit integer containing speed is sent
5. 0xAA is sent (stop byte)
6. 0x0D (the same instruction as sent) is received as one byte from the MCU, indicating it was successfully received and understood

Writing the Motor Speed (viewed from the MCU)

1. A byte is received by the UART, if its 0x53, move to the instruction state
2. A byte is received by the UART, if it is a write instruction, save it, and move to the LSB state
3. A byte is received by the UART, shift appropriately into integer storage, move to MSB state
4. A byte is received by the UART, shift appropriately into integer storage, move to stop state
5. A byte is received by the UART, if its 0xAA, indicate to the while(1) that an instruction is ready for processing, and return to the start state
6. The while(1) detects an instruction is ready, processes the instruction (in this case writes the integer to OCR1A), then sends back the received instruction (in this case UDR1 = 0x0D).

In all cases, if the start byte is not received by the MCU ISR while in the start state, then the data received is thrown away. The same applies if the stop byte is not received at the end of a packet of information – if not received, all data and instructions are discarded and no action is taken.

The table below shows the instructions and desired actions between the MCU and PC:

|  |  |  |
| --- | --- | --- |
| Instruction | Code | MCU Action |
| TXCHECK | 0x00 | Return 0x0F to indicate the communications is working |
| READ\_PINA | 0x01 | Return the contents of PINA |
| READ\_POT1 | 0x02 | Read ADC Channel 2 and return ADCH |
| READ\_POT2 | 0x03 | Read ADC Channel 1 and return ADCH |
| READ\_TEMP | 0x04 | Read ADC Channel 3 and return ADCH |
| READ\_LIGHT | 0x05 | Read ADC Channel 0 and return ADCH |
|  |  |  |
| SET\_PORTC | 0x0A | Write the 8 LSB of the received integer to PORTC, return 0x0A |
| SET\_HEATER | 0x0B | Write the received integer to OCR1C, return 0x0B |
| SET\_LIGHT | 0x0C | Write the received integer to OCR1B, return 0x0C |
| SET\_MOTOR | 0x0D | Write the received integer to OCR1A, return 0x0D |

Note that all write instructions are 0x0A or greater. You can use this feature to determine whether your ISR state machine expects the data bytes next, or the stop byte. Also be aware that start and stop bytes are not required for the return information, and that only 1 byte is ever returned.

C# Serial Port Use

In this assignment you will learn how to use C# to communicate over the serial port. It is actually remarkably easy, requiring only a few lines of code. A couple documents which may help you are listed below:

<https://msdn.microsoft.com/en-us/library/system.io.ports.serialport%28v=vs.110%29.aspx>  
<http://www.codeproject.com/Articles/678025/Serial-Comms-in-Csharp-for-Beginners>  
Your program should:

1. Create a new SerialPort object
2. Configure it based on the user settings (COM Port, Baudrate, 8N1, 500ms timeouts)
3. Open the SerialPort once connect is clicked
4. Use Write() and ReadByte() to communicate with the application board
5. Close the SerialPort when the program closes

Remember, as per the communication protocol above, every write should send the start and stop bytes as well.

Hint: To send a 16 bit integer, use BitConverter.GetBytes() to break up the integer into its respective bytes, then send it in the normal array order (e.g. dbytes[0] first, then dbytes[1]).

PI Controller

For the temperature control tab you are to implement a PI (not PID) controller to control the measured temperature to a desired setpoint. Wikipedia has a reasonable article on the PID controller:  
<http://en.wikipedia.org/wiki/PID_controller>

At the bottom of the above article is some pseudocode for implementing a PID controller. You are to modify this to only implement the Proportional and Integral components, and also flip the error calculation so it is measured\_value - setpoint. You will need to implement a timer running at 10-20Hz to obtain regular sampling and control of the temperature and motor.

The operation of the PI controller will be as follows:

1. A timer ‘tick’ occurs
2. The heater is set to 100% on
3. The current temperature (in °C) is read from the applications board
4. The difference between the measured temperature (3) and setpoint temperature is calculated (the error)
5. The integral error is calculated as the running sum plus error\*dt, where dt is the sampling interval (0.05 for 20Hz)
6. The controller output is calculated from the PI control law Kp\*error + Ki\*int\_error, where Kp and Ki are positive constants chosen by the user.
7. The controller output is saturated to be 0->100% (it must not exceed these values)
8. The controller output is written to the motor on the applications board
9. The event ends until the next timer tick, after which it repeats the above

C# Timer

In order to avoid using a Refresh button, you can use a C# Timer Object to automate the reading and writing of data from the applications board. Simply drag and drop a timer from the toolbox onto your Form and it will sit in the designer tray at the bottom of the Designer. You can programmatically configure the timer to then call one of your functions every n milliseconds, allowing you to automate the Refresh button event.

The best way to implement the timer is to change the Tick Event Handler based on which tab is open. For example if the Digital I/O tab is open, then the digIO\_Tick Event should be called (reading and writing the digital I/O), and the events associated with the pots, light and temperature can be ignored. Create an event handler associated with the “SelectedIndexChanged” in the tab control, and use the following code:

private void appTabs\_SelectedIndexChanged(object sender, EventArgs e)

{

switch(appTabs.SelectedIndex)

{

case 0: //Setup Tab

appTimer.Stop();

break;

case 1: //Digital I/O Tab

UpdateTimerEvent(digIO\_Tick);

break;

Where digIO\_Tick is an event handler you have written to perform the digital I/O tasks required:

//Digital I/O Callback to Automate LEDs

void digIO\_Tick(object sender, EventArgs e)

{

//Read PORTC GUI boxes and convert to byte

byte PORTC = …

The UpdateTimerEvent function is a method you can use verbatim below:

//Removes existing tick events and adds the new one

private void UpdateTimerEvent(EventHandler newev)

{

appTimer.Stop(); //Stop the Timer

if (ev != null) //If we have a previous event (ev is a data member of  
 // type EventHandler)

appTimer.Tick -= ev; //Remove the previous tick event

appTimer.Tick += newev; //Add our new one

ev = newev; //Save the new one

appTimer.Start(); //Restart the timer

}

C# Classes

The group member writing the PC Application is required to write two classes:

Form Class

The form class will be automatically created for you when you create your Windows Form Project. However you will need to add the following methods to it:

* ConnectClick [Event Handler that is run when the Connect Button is clicked]
* RefreshClick [Event Handler that is run when the Refresh Button is clicked (optional)]
* appTabs\_SelectedIndexChanged [Event Handler that is run when a Tab is selected]
* UpdateTimerEvent [As per the C# timer section]
* digIO\_Tick [Tick Event to control Digital I/O Functionality]
* Pot\_Tick [Tick Event to control Potentiometer Functionality]
* Lamp\_Tick [Tick Event to control Lamp Functionality]
* Temp\_Tick [Tick Event to control Temperature Control Functionality]
* FormClosing [Event Handler that is run when the form is closing, to reset all MCU peripherals]

AppBoard Class

You will need to create the AppBoard class by right clicking on your project and selecting Add -> New Class. The AppBoard class will contain the following methods:

* Connect [Connects the Serial Port to the Application Board]
* Disconnect [Disconnects the Serial Port from the Application Board]
* getCOMPorts [Returns the available COM Ports]
* setCOMPort [Sets the user selected COM port]
* setBaudrate [Sets the user selected Baudrate]
* checkTx
* readPINA
* readPotV
* readLight
* readTemp
* writePORTC
* writeLight
* writeHeater
* writeMotor
* readUInt8 [Using the Serial Port, sends an instruction and receives a UInt8 back]
* writeUInt16 [Using the Serial Port, sends ends an instruction and 16 bit integer and receives a UInt8 back]

You do not have to follow the above list exactly, but it is a recommended architecture.

Getting Started

Due to the complexity of this assignment, a series of example steps are presented below to assist you completing the software component of the assignment.

MCU

1. Begin by modifying your ISR from Assignment 2 to now look for 0x53 as the start character, and 0xAA as the end character. You will not need a buffer, only an unsigned char to store the instruction, and an unsigned integer to store the data.
2. Use the MATLAB appBoard class on Canvas to test your ISR – it will send information in the correct format so you can test if your program is correctly interpreting the packets. You may like to light PORTC based on the instruction received (for example), to test it is all working.
3. Write your while(1) as a state machine to decode the instructions coming in. Each state is an instruction, simply follow the communication table to automate each of the tasks. Remember to send back the required byte for every instruction.
4. Confirm the program is working with the MATLAB appBoard class, this way you will not need to rely on your team mate!

PC Application

1. Begin by creating the GUI, including all tabs, gauges and LEDs. Remember to name them appropriately.
2. Get the SerialPort object working. Once you can reliably send and receive bytes, the rest of the program is simple. You may like to write an echo program on your MCU so you can confirm the correct bytes are being sent.
3. Write your UART class to handle communications with the MCU board.
4. Implement functionality required for the Digital I/O tab, including communications with the MCU board via the Refresh button event handler.
5. Add a timer to your project and connect the event handlers. Most of the code is already provided, it is just a matter of finishing off each of the functions. This will make the GUI much more impressive.
6. For database connection through C# form, please refer to this link: <https://www.youtube.com/watch?v=qAOQ3PJHGko>
7. Continue completing the functionality for the remainder of the tabs.

**5. Marking Schedule**

IMPORTANT: **Individual work only and one report per student**. Discussion with fellow colleagues is allowed, but it should be mentioned in the Acknowledgement section of the report. Sharing of code or report is strongly discouraged. Turnitin will be used to check for plagiarism.

It will contribute 40% towards the final grade. It requires a format report in PDF format and a .zip file of the source code. **Please submit it through Canvas**. The total marks are 40, where 15 marks come from the demonstration of working solution, and 25 marks come from the format project report. Here is a breakdown of the marks.

|  |  |  |
| --- | --- | --- |
| Items | Description | Marks |
| 1. | Correct GUI design—correct layout and naming of components | 2 |
| 2. | Correct C# code—without compilation or logical errors | 3 |
| 3. | Correct C/C++ code for microcontroller—without compilation or logical errors | 5 |
| 4. | Working Solution—satisfies all the requirement | 5 |
| 4. | Report Contents, including introduction, objective, methodology, design and implementation, results, discussion, conclusion and proper references, and code in tables in the appendices. | 12 |
| 5. | Report Presentation, including proper figures, tables, equations and codes. | 10 |
| 6. | Correct file format—PDF report with signed AUT Assignment Coverage Sheet, and codes in .ZIP file | 3 |
|  | Total | **40** |