SERIAL PORT DESIGN USING AN ARDUINO

GROUP 14:
PANKAYARAJ P (E/14/237)
SAMARASINGHE U.G.C.B (E/14/305)
SUMANASEKERA Y.D (E/14/337)

TABLE OF CONTENTS

1. INTRODUCTION

1.1. OVERVIEW

In the past, many PCs used to have at least one if not two, built-in serial ports. Throughout most of the history of personal computers, data was transferred through serial ports to devices such as modems, terminals, and various peripherals. Nowadays, built-in serial ports are rare. Most modern PCs that need an RS-232 serial port opt for either an add-in card, or USB-to-serial converters to allow compatibility with RS-232 serial devices.

However, serial ports are still widely used within the industrial space in applications such as industrial automation systems, scientific instruments, point of sale systems and some industrial/ consumer products. Serial ports are still used in these areas as they are simple, cheap and their console functions are highly standardized and widespread.

The objective behind this project is to build a serial port interfacing device using an Arduino which can be connected to any type of PC via USB (as illustrated in Figure 01). This interface will be shipped with a custom C library with standard commands to read and write into the serial port and a GUI program which can be used to configure and monitor the serial port.

This design can be extended to interface multiple Serial ports using a single USB port, which can be used for a larger industrial automation project.

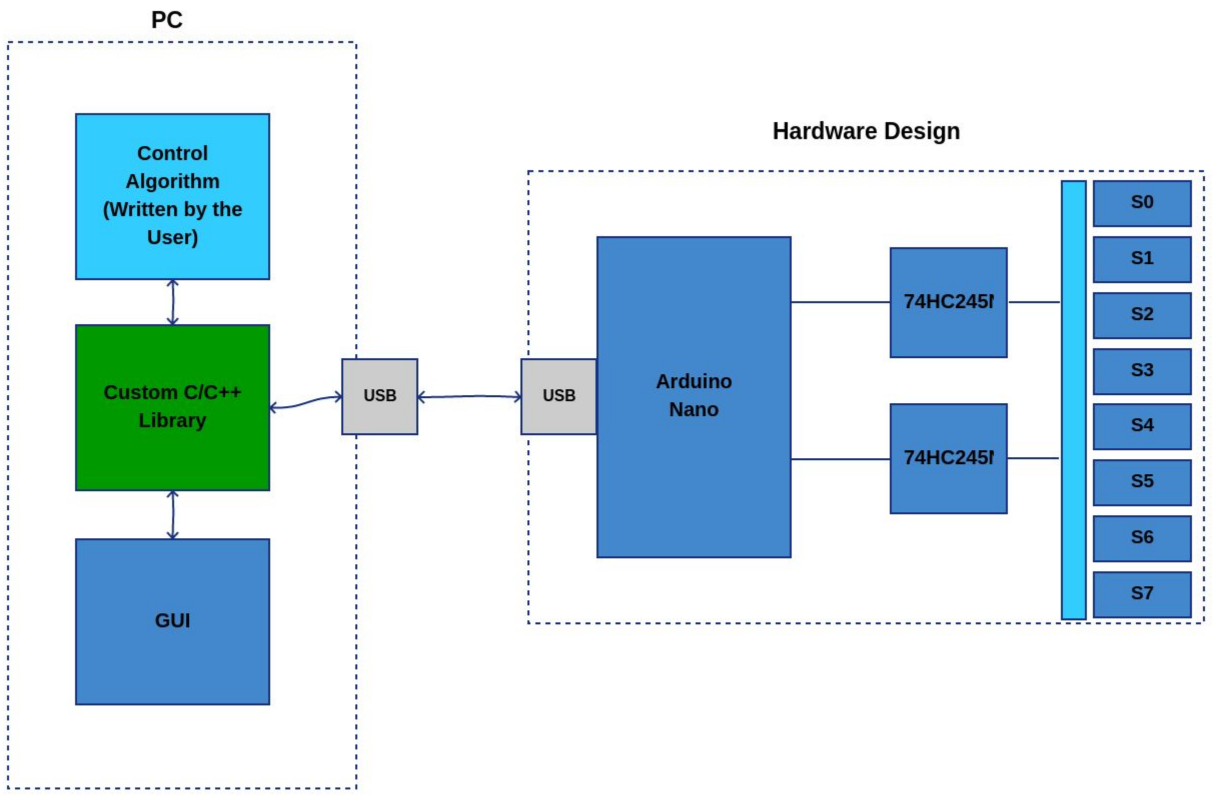


FIGURE 01: BLOCK DIAGRAM

In this design, the Arduino will be used for communication purposes only (as the USB interface board to communicate with the serial port design). The control algorithm will be implemented on the PC using C language with the provided library.

Thus the project deliverables can be summarized as follows.

- **Software:**
  - GUI application and a custom C Library for serial port communication.
- **Hardware:**
  - Arduino based Serial Port Interface which can be connect to any PC via USB.

1.2. WORKLOAD DISTRIBUTION

The overall workload was broken down into the following major tasks and sub-tasks.

(1) Design of the overall software and hardware architecture (abstract design)

- Designing a block diagram for the hardware and software
- Selecting a suitable microcontroller for PC-Device USB interfacing.
- Determining possible approaches to power the circuit components. That is, determining whether to design the hardware as USB bus-powered or self-powered.
- Determining the optimal usage of the electronic components (E.g. the number of serial ports).
- Determining user flexibility and the ease of using the product.

(2) Hardware design and fabrication

- Selecting the Arduino Nano for the USB interface.
- Selecting buffer ICs.
- Designing the schematic circuit diagram.
- Designing the single sided PCB.
- Fabrication of the PCB.
- Soldering components.
- Inspection for the continuity of routes. This included individually checking the continuity and functionality of the electronic components.
- Selecting IC bases and headers considering maintainability.

(3) Feasibility study to determine approaches to ensure efficient PC-Arduino communication

- Testing the serial communication using standard Arduino libraries.
- Writing and testing PC-side C/C++ software components.
- Developing Arduino-side USART communication library functions from the scratch as standard libraries were found to be inefficient.
- Developing PC-side serial communication software components using Qt libraries.

(4) GUI application development

- Identifying the device connection status (i.e. properly connected or not connected).
- Developing the capability to configure each serial interface on the device via the GUI.
- Developing a PC-based C/C++ library integrated with the GUI.
- Critically analysing the user friendliness of the GUI.

(5) High-level software protocol design

- Implementing the protocol in such a way that it is handled by the library (and not by the programmer).
- Ensuring that a programmer can easily invoke library functions to perform tasks.
- Determining the use of fixed-sized or variable-sized messages.
- Determining the types of messages to be implemented (E.g. token messages, data messages, configuration messages).
- Testing the functionality of the protocol.

(6) Design and development of a C/C++ library for the PC-side

- Testing several C/C++ software components following online resources.
- Developing PC-Arduino communication using serial communication classes and libraries offered by Qt toolkit libraries.
- Developing basic library functions to enable,
    - Configuration of serial port parameters:
        - Baud rate
        - Number of data bits, parity bits and stop bits
    - Writing of data to the serial port
    - Reading of data from the serial port
- Isolating the necessary library files from the Qt library and configuring a simple environment to build the program using the library.

(7) Design and development of an Arduino software

- Developing Arduino-side USART communication library functions from the scratch as standard libraries were found to be inefficient.
- Implementing and testing the designed protocol in Arduino.
- Determining approaches to develop Software Serial ports in Arduino as eight serial interfaces were required to be implemented in the device.
- Implementing the software serial functionality.
- Adding the functionality to ensure permanent saving of configurations using EEPROMs.

(8) Writing sample applications

- Developing a PC-side C/C++ software component to test serial communication (as the host)
- Developing an Arduino sketch to test serial communication (as the listener)

Thus, the workload was distributed among the group members in the following manner while some tasks were carried out collaboratively.

GROUP MEMBER	**TASK**
E/14/237	(1) , (3) , (8)
E/14/305	(1) , (2) , (4) , (6) ,(7)
E/14/337	(1) , (3) , (5) , (8)

TABLE 01: WORK DISTRIBUTION

2. PROJECT WORK

2. 1. Design of the Overall Software and Hardware Architecture (Abstract Design)

Initially, the expected final product and the scope were determined upon discussing with the lecturer-in-charge. Thereafter, the basic block diagram was designed paying individual attention to the following criterion.

- User friendliness and the ease of using the product.
- Simple to use C/C++ based library.

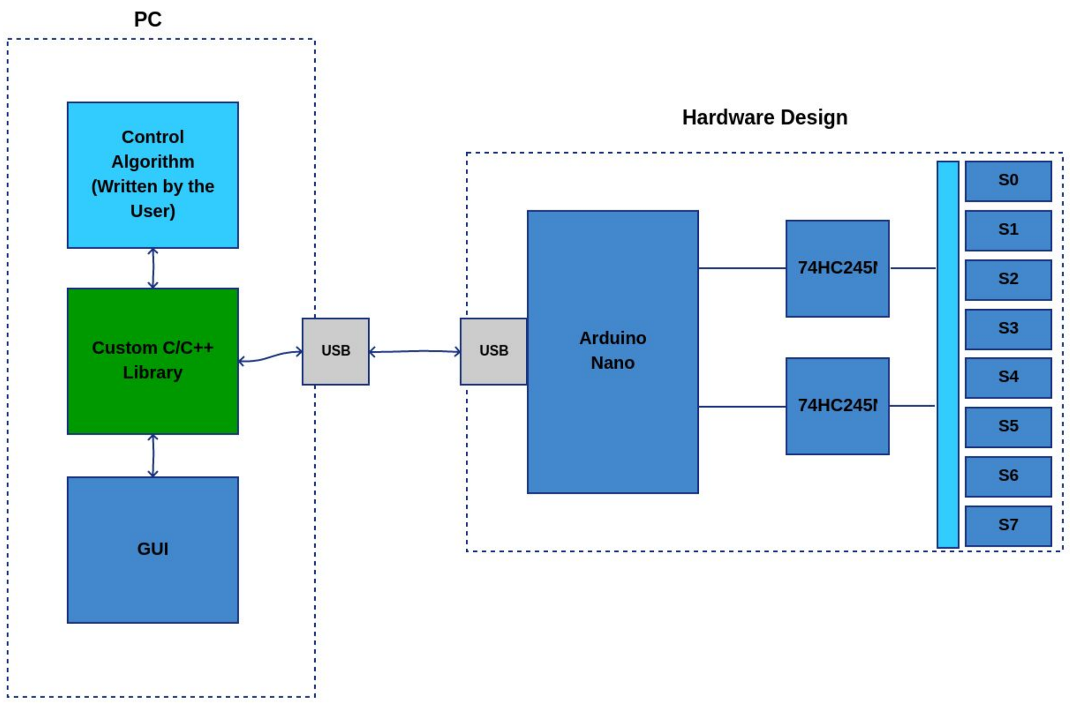


FIGURE 02: BLOCK DIAGRAM

Arduino Nano was selected as the PC-to-Device USB interfacing device (as shown in Figure 02), as it had the following characteristics which were best suited for the product.

- Mini USB support
- Compact physical design
- Sufficient number of GPIO pins
- Availability

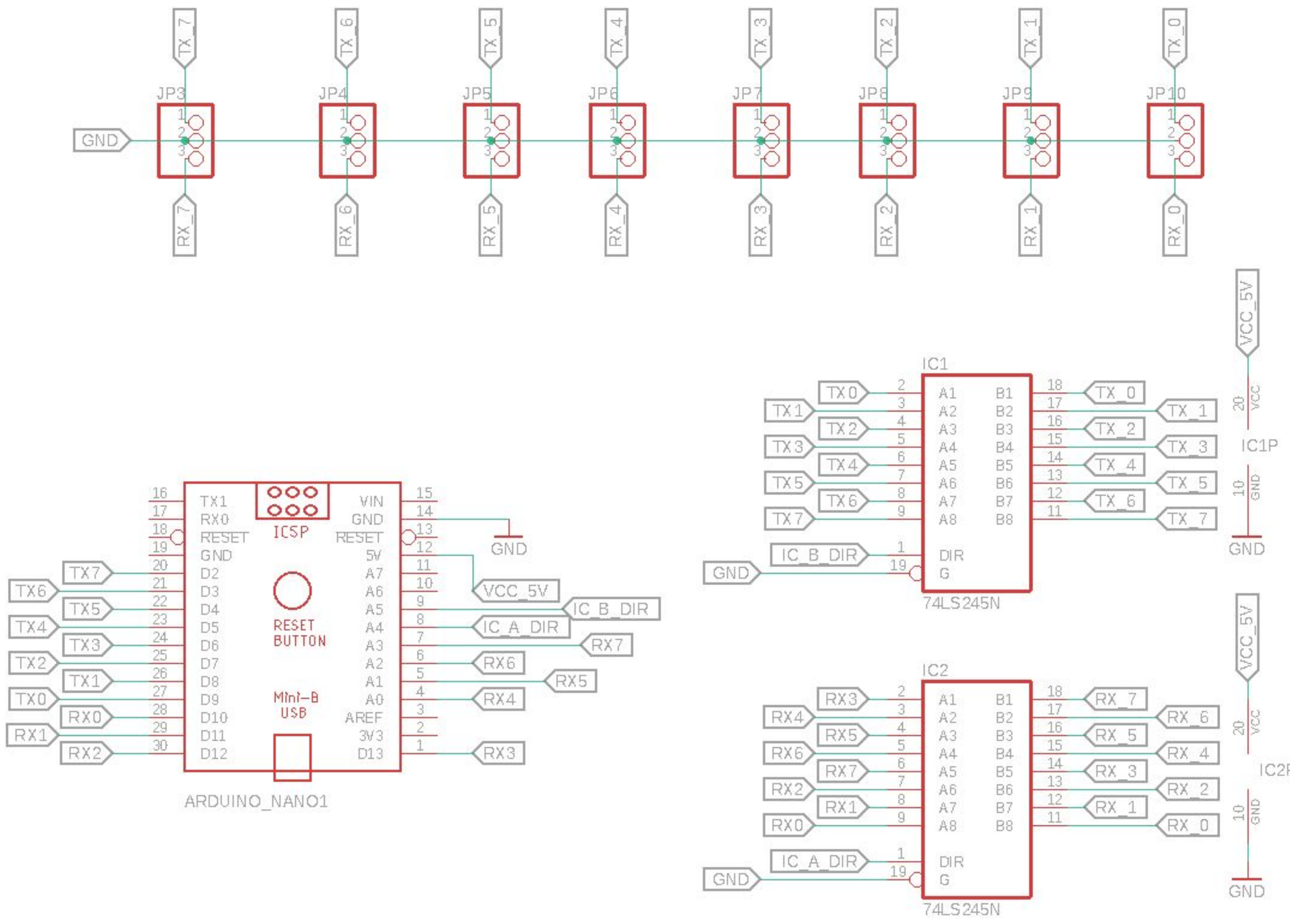
Upon considering the portability of the device and the power consumption of the components, the device was designed as a USB bus-powered device. The reasoning behind the selection of a bus-powered design as opposed to a self-powered design was that a self-powered design would require the addition of a rechargeable power source and thus would not be a convenient design.

At least one buffer IC is required in the circuit, in order to design even a single serial interface. On the other hand, the Arduino has a number of unused GPIO pins. Therefore, in order to ensure optimal usage of the hardware components, instead of designing a single serial interface, the hardware was designed with 8 serial interfaces by using two 8-bit buffer ICs.

The GUI application and a C/C++ library was designed determining user flexibility and the ease of using the product.

2.2. Hardware Design and Fabrication

Upon selecting the hardware components, the schematic diagram (Figure 03) was drawn using the Autodesk Eagle electronic design automation application.



<span style="text-decoration:underline">**FIGURE 03: SCHEMATIC CIRCUIT DIAGRAM**</span>

Thereafter the PCB was designed using the same software. When designing the PCB, the pin mappings were carefully done in order to effectively route all the connections without any overlapping on a single-sided PCB. Therefore, the connections were repeatedly changed, switching between the schematic and PCB designing views.

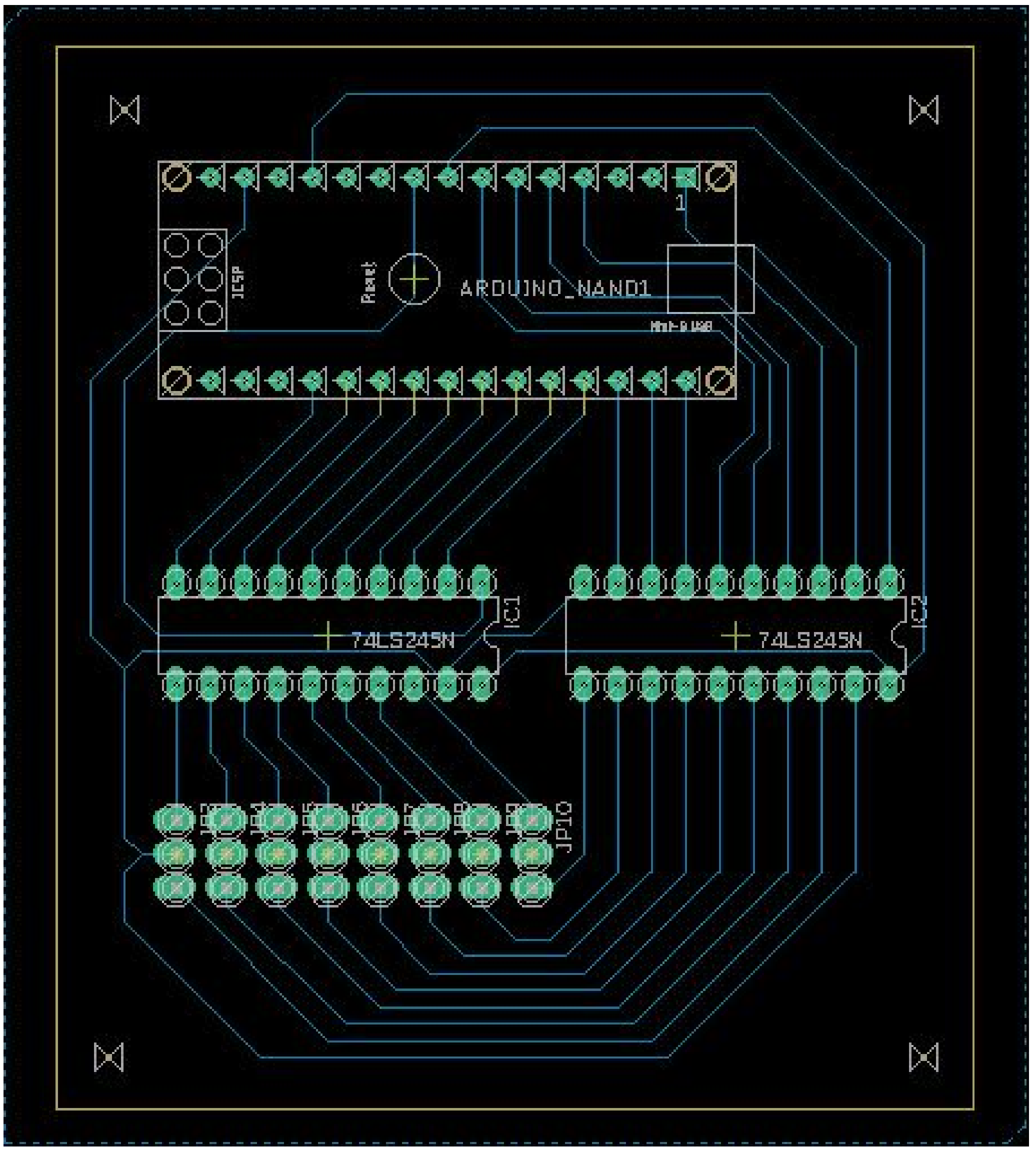
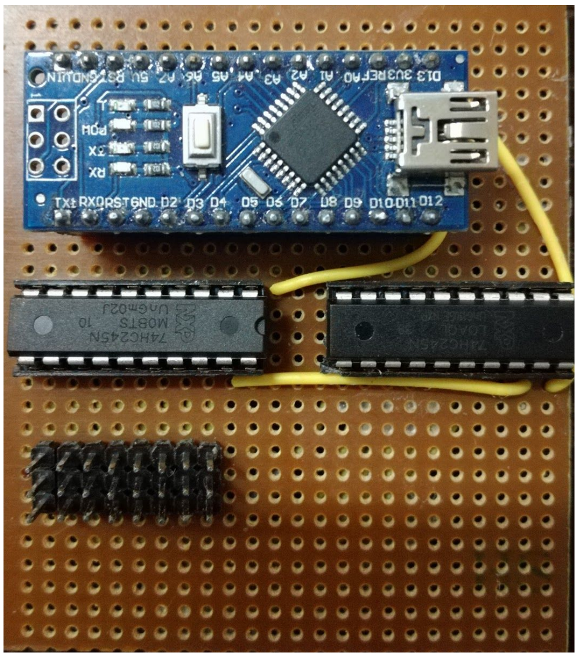


FIGURE 04: PCB DESIGN

Upon receiving the PCB, a thorough inspection was carried out to check the continuity of routes. However, the printed PCB was found to have a number of discontinuities and errors. This resulted in the implementation of the entire circuit using a dotted board. After soldering the components, the continuity and functionality of the electronic components were individually checked.

When soldering the ICs, IC bases and headers were used considering the maintainability.


TOP VIEW

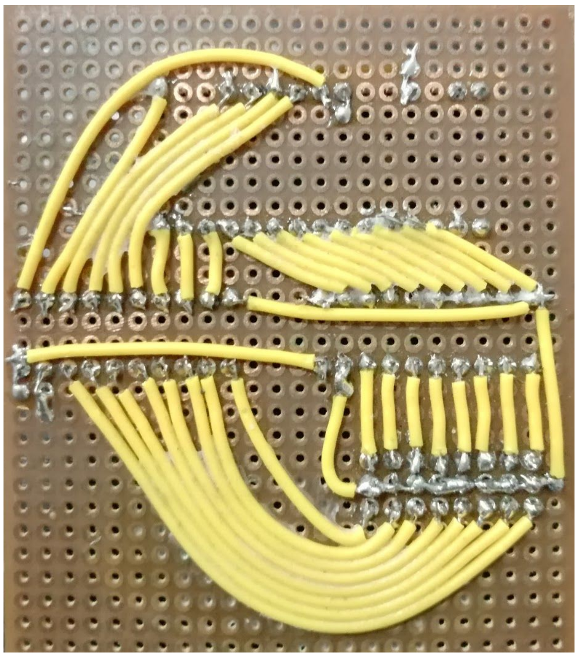

BOTTOM VIEW

FIGURE 05: SOLDERED CIRCUIT

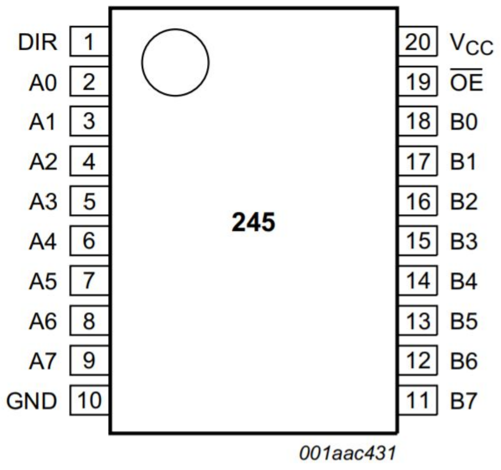
List Of Components:

COMPONENT	QUANTITY
74HC245 8-bit transceiver	2
Arduino Nano	1

TABLE 02: COMPONENTS LIST

- **74HC245:**

74HC245[1] is an 8-bit transceiver with non-inverting 3-state outputs. 74HC245 is designed for asynchronous transfer of data between different data buses. The data can be transferred in both directions depending upon the logic level on the DIRection control input (DIR). Out Enable pin (OE) is used to enable and disable the device depending upon the requirements of the task.



DIR	Direction control
A0, A1, A2, A3, A4, A5, A6, A7	Data input/output
GND	Ground (0 V)
B7, B6, B5, B4, B3, B2, B1, B0	Data input/output
$\overline{OE}$	Output enable input (active LOW)
V_{CC}	Supply voltage

FIGURE 05: 74HC245 PIN ARRANGEMENT AND DESCRIPTION

The recommended operating conditions of the 74HC245 IC are stated in the following table.

Symbol	Parameter	Conditions	74HC245			Unit
			Min	Typ	Max	
V_{CC}	supply voltage		2.0	5.0	6.0	V
V_I	input voltage		0	-	V_{CC}	V
V_O	output voltage		0	-	V_{CC}	V
$\Delta t/\Delta V$	input transition rise and fall rate	V_{CC} = 2.0 V	-	-	625	ns/V
		V_{CC} = 4.5 V	-	1.67	139	ns/V
		V_{CC} = 6.0 V	-	-	83	ns/V
T_{amb}	ambient temperature		–40	+25	+125	°C

TABLE 03: RECOMMENDED OPERATING CONDITIONS

The purpose of using buffer ICs is to safeguard the microcontroller. That is, in order to protect the microcontroller in the presence large currents where it may act as a current sink/source. However significantly large currents are not expected, as the current is drawn from the USB.

Arduino Nano:

Arduino Nano[2] is a small, complete, flexible and breadboard-friendly microcontroller board based on ATmega328P. Although the operating voltage is 5V, the input voltage can vary from 7V to 12V. Arduino Nano pinout contains 14 digital pins, 8 analog pins, 2 reset pins and 6 power pins.

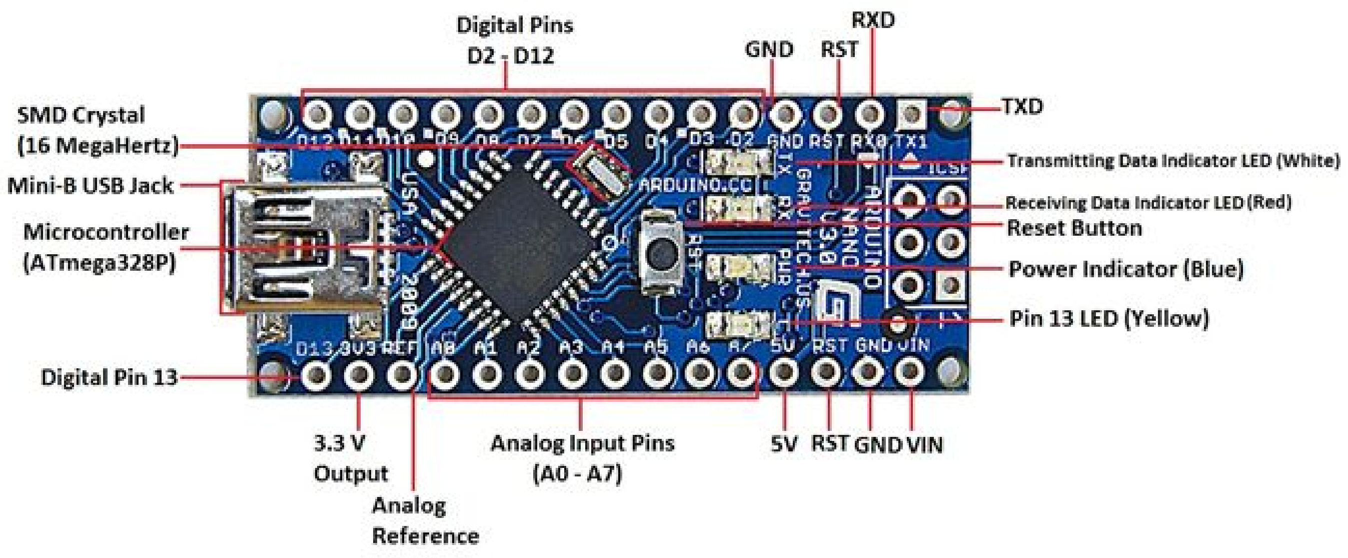

Digital Pins D2 - D12
GND
RST
RXD
TXD
SMD Crystal (16 MegaHertz)
Mini-B USB Jack
Microcontroller (ATmega328P)
Transmitting Data Indicator LED (White)
Receiving Data Indicator LED (Red)
Reset Button
Power Indicator (Blue)
Pin 13 LED (Yellow)
Digital Pin 13
3.3 V Output
Analog Reference
Analog Input Pins (A0 - A7)
5V
RST
GND
VIN

FIGURE 06: ARDUINO NANO PINOUT

(a) Communication

An FTDI FT232RL on the board channels the serial communication over USB and the FTDI drivers (included with the Arduino software) provide a virtual com port to software on the computer.

(b) Input and Output

Each of the 14 digital pins on the Nano can be used as an input or output. They operate at 5 volts. Each pin can provide or receive a maximum of 40 mA and has an internal pull-up resistor (disconnected by default).

A few of the technical specifications of the Arduino Nano are summarized in the following table.

Microcontroller	ATmega328
Architecture	AVR
Operating Voltage	5 V
Clock Speed	16 MHz
DC Current per I/O Pins	40 mA (I/O Pins)
Input Voltage	7-12 V
Power Consumption	19 mA

TABLE 04: TECHNICAL SPECIFICATIONS OF ARDUINO NANO

2.3. Feasibility Study to Determine Approaches to Ensure Efficient PC-Arduino Communication

An FTDI FT232RL on the Arduino board channels the serial communication over USB and the FTDI drivers provide a virtual com port. Therefore, an online research was carried out to determine suitable methods to connect to the Arduino via USB. Thus, it was found that almost all the methods involved communicating with the Arduino via serial communication methods.

Initially, the standard serial library was used in the Arduino sketch. However, the time taken to receive a reply from the Arduino for a transmitted message proved to be too large (nearly a second). Thus, as the standard libraries were found to be inefficient, it was decided to develop our own functions from scratch in the Arduino for USART communication by referring to available material on UART communication on AVR chips[3].

Furthermore, the PC-side serial communication software components were developed using Qt libraries. This library was found to be the most stable serial communication library thus far.

2.4. GUI Application Development

The Qt Creator[4] integrated development environment was used to create the GUI application using C++. As the the library was expected to be developed in C/C++ (i.e. as per project requirements), the GUI as also developed using C++ as well.

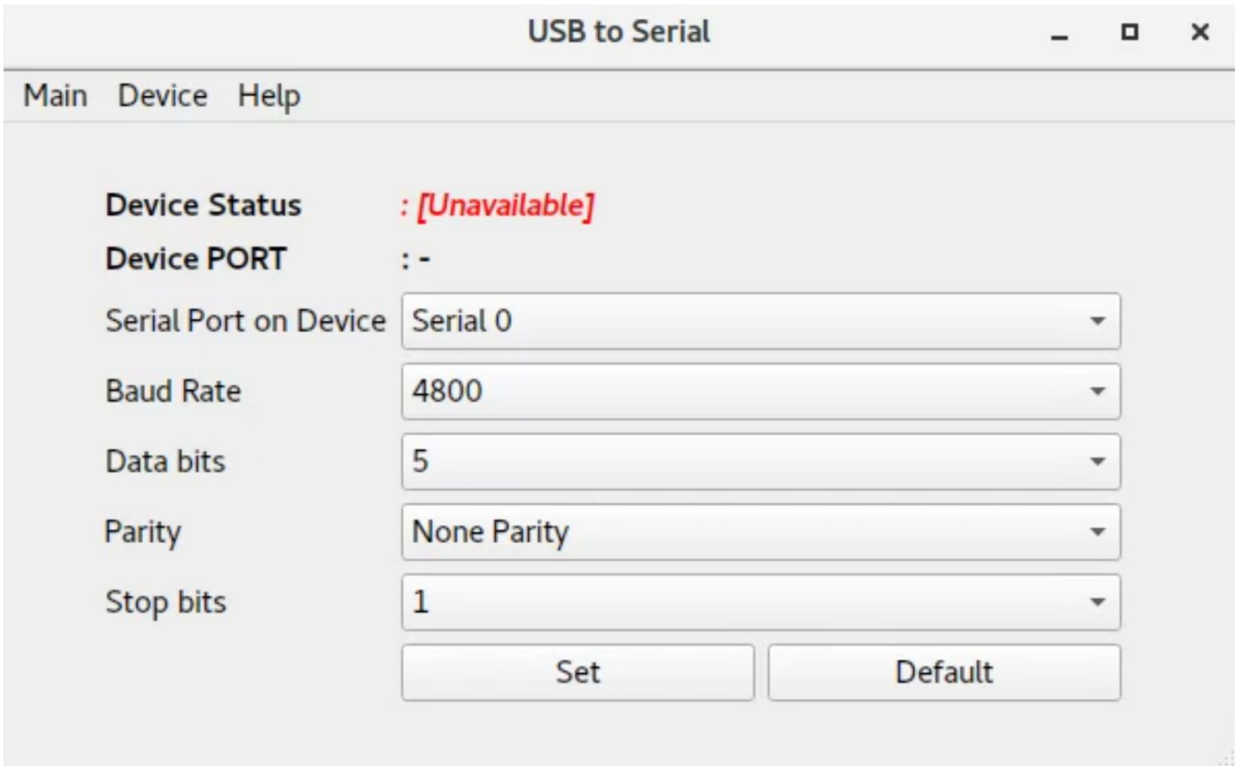
The GUI application was designed to provide the following functionalities.

- Identifying the device connection status (i.e. properly connected or not connected).
- Ability to configure each serial interface on the device.
- Automatically loads configuration information from the device and displays it in the GUI for a selected serial port

The PC-side C/C++ library was integrated in the GUI to achieve the above two functionalities.

Moreover, the GUI enables the user to configure the arduino port prior to initiating serial communication by setting the following parameters[5]. Setting these parameters help ensure robust and error-free data transfers.
(a) Data bits: The data bits indicate the amount of data in each packet which can be set to 5, 6, 7, or 8 bits.
(b) Baud rate: The baud rate refers to the number of times per second a serial communication signal changes states. In other words, it is the rate at which information is transferred through the communication channel.
(c) Parity bits: The parity bit is used to determine if the data character being transmitted is correctly received. Thus, it is the the simplest method of error detection.
(d) Stop bit: The stop bit acts as a synchronization bit. That is, it acts as a means of timing or synchronizing the data characters being transmitted. The number of stop bits is configurable to either one or two.

The following diagram illustrates the window used to setup the above mentioned parameters.


USB to Serial

Main Device Help

Device Status : [Unavailable]

Device PORT : -

Serial Port on Device: Serial 0

Baud Rate: 4800

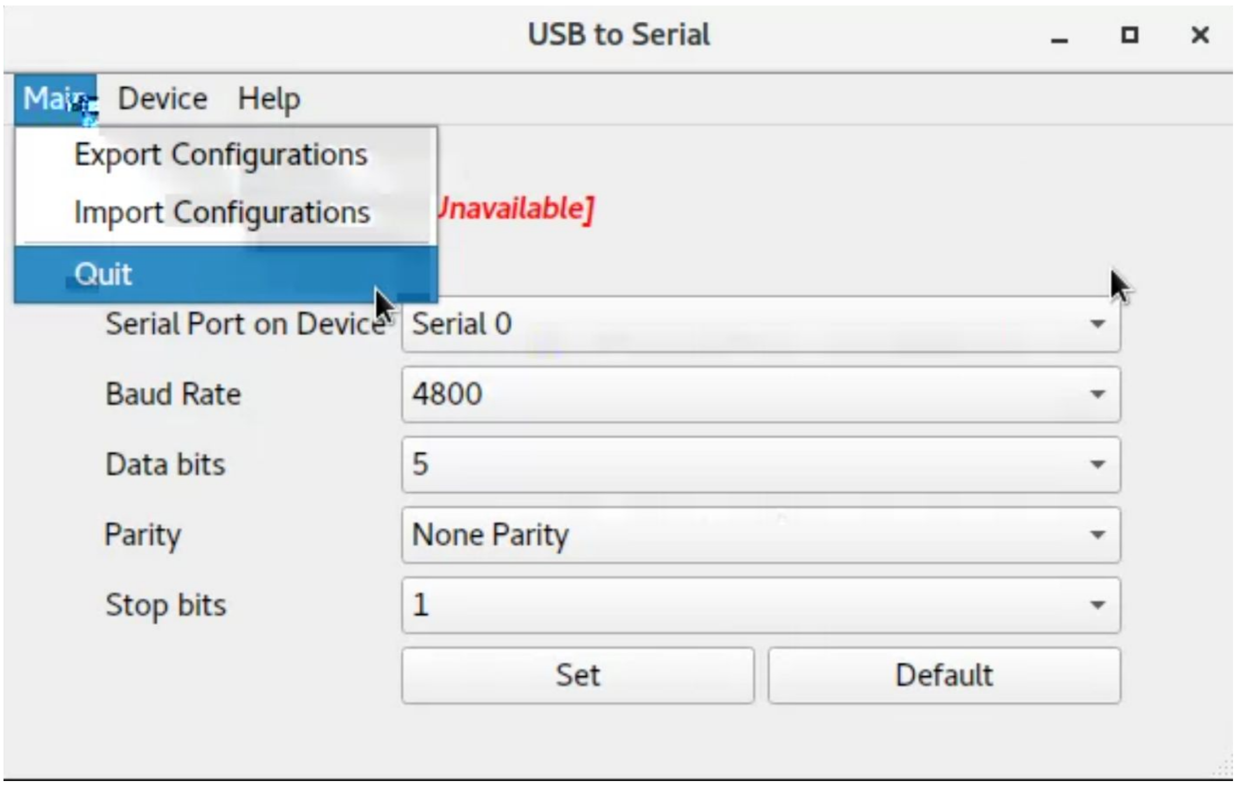
Data bits: 5

Parity: None Parity

Stop bits: 1

FIGURE 07: GUI FOR CONFIGURING PARAMETERS

The subsequent diagrams illustrate the additional functionalities of the GUI.



Serial Port on Device: Serial 0

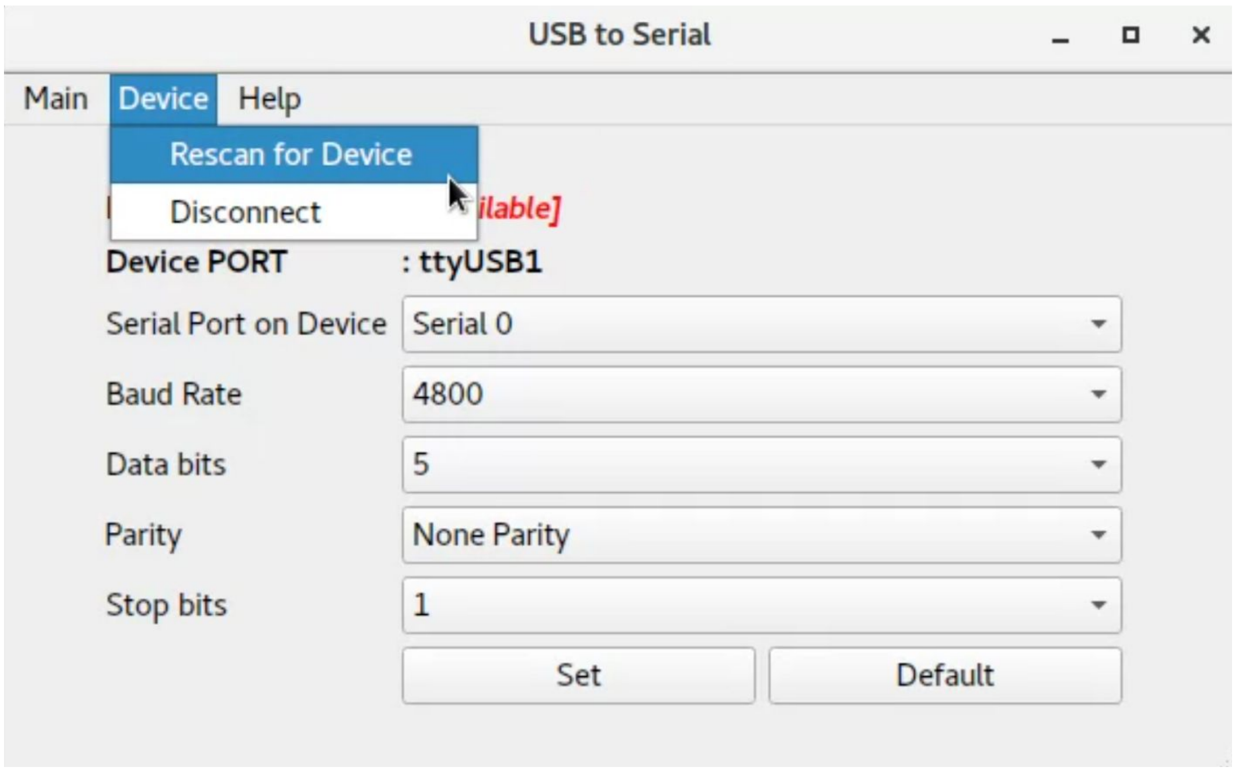
Baud Rate: 4800

Data bits: 5

Parity: None Parity

Stop bits: 1

FIGURE 08: MAIN MENU

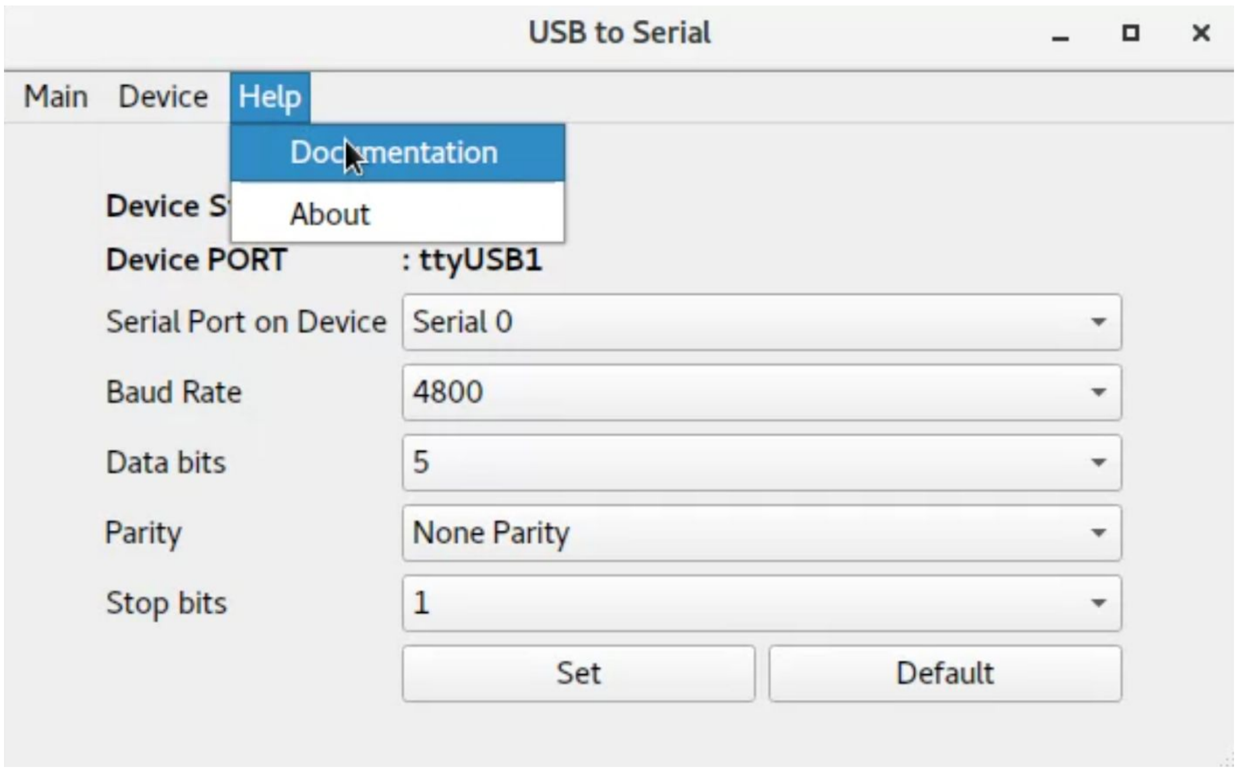


ilable]

Device PORT : ttyUSB1

Serial Port on Device	Serial 0
Baud Rate	4800
Data bits	5
Parity	None Parity
Stop bits	1

FIGURE 09: DEVICE SCAN



Device S

Device PORT : ttyUSB1

Serial Port on Device	Serial 0
Baud Rate	4800
Data bits	5
Parity	None Parity
Stop bits	1

FIGURE 10: HELP MENU

2.5. High-level Software Protocol Design

A high level software protocol was designed to communicate with the device since the device is required to have a mechanism to distinguish one message from another to perform various tasks such as,

- Reading data from a particular serial port
- Writing data to a particular serial port
- Configuring a particular serial port
- Reading the configuration parameters of a particular serial port

The implementation was such that, the protocol related communication is entirely handled by the library. Thus, this eliminates the need for programmer intervention in such a situation. Moreover, focus was put on ensuring that a programmer can easily invoke library functions to perform tasks.

The designed protocol has the following characteristics.

- Use of fixed-sized (1 byte) messages.
- Three types of messages are considered. Namely,
    - Token messages
    - Configuration messages
    - Data messages

The three mentioned message types are of the following format.

- **Token Message:**

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
SERIAL PORT ID			TOKEN FLAG				

FIGURE 11: FORMAT OF A TOKEN MESSAGE

Here the **Serial Port ID** is 3-bit number used to uniquely identify each of the eight serial ports in the device (0-7). The **Token Flag** is a 5-bit number which specifies the type of the message. The following table states the different types of token messages available.

DESCRIPTION	**BIT PATTERN**	**FLAG**
Write serial port configuration	00001	`M_CONF_WRITE`
Read serial port configuration	00010	`M_CONF_READ`
Read data from serial port	00100	`M_DATA_READ`
Write data to serial port	01000	`M_DATA_WRITE`
ACK from device after writing a configuration	10001	`M_FROM_DEVICE_CONF_WRITE_ACK`

Configuration read response message	10010	`M_FROM_DEVICE_CONF_READ`
Data read response message	10100	`M_FROM_DEVICE_DATA_READ`
ACK from device after writing data to a serial port	11000	`M_FROM_DEVICE_DATA_WRITE_ACK`
Reserved	11111	`M_FROM_DEVICE_DATA_SERIAL`

TABLE 05: TYPES OF TOKEN MESSAGES

- **Configuration Message:**

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
BAUD RATE			DATA BITS		PARITY BITS		STOP BIT

FIGURE 12: FORMAT OF A CONFIGURATION MESSAGE

Here the **Baud Rate** is a 3-bit number used to select one of the baud rates as shown in the following table.

BAUD RATE	BIT PATTERN
4800	000
9600	001
14400	010
19200	011
38400	100
57600	101
115200	110
Unset baud rate	111

TABLE 06: BAUD RATE SETTINGS

The **Data bits** represent a 2-bit number used to specify the amount of data bits (i.e. the data size).

DATA SIZE	BIT PATTERN
5	00
6	01
7	10
8	11

TABLE 07: DATA BIT SETTINGS

The **Parity Bits** is a 2-bit number used to set the parity mode.

PARITY MODE	BIT PATTERN
None	00
Odd	01
Even	10

TABLE 07: PARITY SETTINGS

The **Stop Bit** is a 1-bit number used to select the number of stop bits.

STOP BIT(S)	BIT PATTERN
1	0
2	1

TABLE 08: STOP BIT SETTINGS

- **Data Message:**

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
DATA							

FIGURE 13: FORMAT OF A DATA MESSAGE

ARRANGEMENT IN MESSAGES

✱ ***Notation:*** PC , DEVICE

- Setting a configuration

  [Token Message] → [Configuration] → [Token - ACK]

- Read a configuration

  [Token Message] → [Data]

- Write data to serial

  [Token Message] → [Data] → [Token - ACK]

- Read data from serial

  [Token Message] → [Data]

The protocol was implemented in both the device and the PC-side libraries and was tested successfully.

2.6. Design and Development Of a C/C++ Library for the PC-side

The C/C++ library was developed to perform the following main functionalities with the device according to the high-level protocol explained in Section 2.5.

- Configuration of serial port parameters:
  - Baud rate
  - Number of data bits, parity bits and stop bits
- Reading the configuration parameters of a given serial port
- Writing of data to the serial port
- Reading of data from the serial port

IMPLEMENTED LIBRARY FUNCTIONS

- `bool` **`device_identify()`**: Use this function to identify whether the device is physically connected or not. Returns a boolean value representing whether the device is properly identified by the system or not.
- `bool` **`device_open()`**: Use this function to open and configure the device for communication. Returns a boolean value representing whether the device is properly opened or not.
- `void` **`device_close()`**: Use this function to disconnect from the device. It will disconnect, if already connected.

- `void` **`set_config`**`(unsigned char serial_selected,int baud_rate,int serial_conf)`: Use this function to write the configuration parameters to a given serial port.
- `void` **`read_config`**`(unsigned char serial_selected,int *baud_rate,int *serial_conf)`: Use this function to read configuration information about the given serial port.

- `void serial_write(unsigned char serial_port, char data)`: Use this function to write data to a given serial port
- `char serial_read(unsigned char serial_port)`: Use this function to read data from a given serial port.

PARAMETER CONSTRAINTS

The **Baud Rate** is an integer value selected from the following range.

BAUD RATE
4800
9600
14400
19200
38400
57600
115200

TABLE 09: BAUD RATES

Configuration is a single integer value representing the number of data bits, parity bits and stop bits information.
The programmer can use predefined flags defined in the library for this purpose.
The format of the `FLAG` is as follows.

`SERIAL_<Number_Of_Data_Bits><Parity><Number_Of_Stop_Bits>`

- The integer range of `Number_Of_Data_Bits` is 5 to 8.
- `Parity` can take the following values.

PARITY	VALUE
None	N
Odd	O
Even	E

TABLE 10: PARITY VALUES

- The range of `Number_Of_Stop_Bits` is either 1 or 2.

Example: Configuration flag for 8 data bits, no parity and 1 stop bit is as follows.

`SERIAL_8N1`

BUILDING THE ENVIRONMENT

Serial communication classes offered by Qt toolkit libraries were used in developing our C/C++ library. One of the key things which was noticed, was that in order to use the library, an end user is required to install the libraries offered by the Qt toolkit on their PC. This includes nearly 900 MB of library related files and total of 4.8 GB with the GUI Development IDE.

As this is neither flexible nor convenient from the perspective of an end user, a lightweight (of only 97 MB) environment was offered to build the program using the library, by carefully isolating only the necessary C++ library files from the Qt toolkit libraries.

File Structure of the building environment:

```
\build
      \gcc_64 (the isolated library files from the Qt toolkit)
      \MakeFile
\source
      \USB2SerialLib (the C++ library developed)
      \project.pro
      \main.cpp (end users program/control algorithm)
```

Thus the end user is only required to modify their control program in `main.cpp` file and using `MakeFile`, the said program can easily be compiled into an executable file.

Moreover, developers who wish to improve the library functionalities, can make the necessary changes to the C++ files within the `USB2SerialLib` directory.

2.7. Design and Development Of an Arduino Software

The implemented Arduino software includes the following key elements.

- Proper signaling to configure the direction of buffer ICs and IO pin mapping.
- Implementation of fast USART functions from the scratch which will be used for the PC-Device communication.
  - Function to initialize the USART with custom configurations
  - Function to read
  - Function to write
- EEPROM functions to store and retrieve the configuration information of each serial port.
- Software serial implementation for serial ports.
- Implementing controller algorithm following the high-level protocol discussed in Section 2.5.

- **Implementation of USART from the scratch**

The initial problem faced during the designing of the Arduino sketch was the significantly slow serial communication speed experienced when using the standard serial communication library for Arduino. Therefore, in order to improve the efficiency, material regarding USART in AVR chips were read and subsequently our own functions were implemented from the scratch as discussed in Section 2.3. The following functions were implemented for this purpose.

```
- void init_usart()
- unsigned char read_usart()
- void write_usart(unsigned char data)
```

- **Software Serial**

Another problem which was encountered was to find a proper way to allow serial communication through the serial ports.

To accommodate the need for more serial ports, the Software serial libraries have been developed to allow serial communication on other digital pins of the Arduino, using software to replicate the functionality of the hardwired RX and TX lines.

Upon referring to resources from the official Arduino website, it was found that the standard software library does not provide means of configuring data bits, parity, error bits etc. Therefore, prior to implementing a software serial library from the scratch, an online research was carried out on custom software serial libraries in order to determine the availability of third party implementations best suited to our requirements.

LIBRARY	REMARKS
Arduino Standard Software Serial [6]	● Only configurable parameters are the RX and TX pins and the baud rate. Inflexibility to set custom values to the parity bit, stop bits and the number of data bits. ● Upon using multiple software serial ports, only one can receive data at a time. ● For some Arduino boards, not all pins support change interrupts. Therefore, only a certain range of pins can be used for RX.
AltSoftSerial [7]	● Only configurable parameters are the RX and TX pins and the baud rate. Inflexibility to set custom values to the parity bit, stop bits and the number of data bits. ● Supports simultaneously transmit and receive operations.
Custom Software Serial [8]	● Allow the flexibility to configure RX and TX pins, the baud rate as well as to set custom parity bit, stop bits and number of data bits.

TABLE 11: COMPARISON OF SOFTWARE SERIAL LIBRARIES

Therefore, by considering the pros and cons of each library, the Custom Software Serial library was selected for the implementation of the serial ports.

- **EEPROM**

The standard EEPROM library[9] was used to implement the functions to store and retrieve the configuration information of each serial port.

- **Implementation of the Controller algorithm**

The following flowchart illustrates the logic used in the controller algorithm.

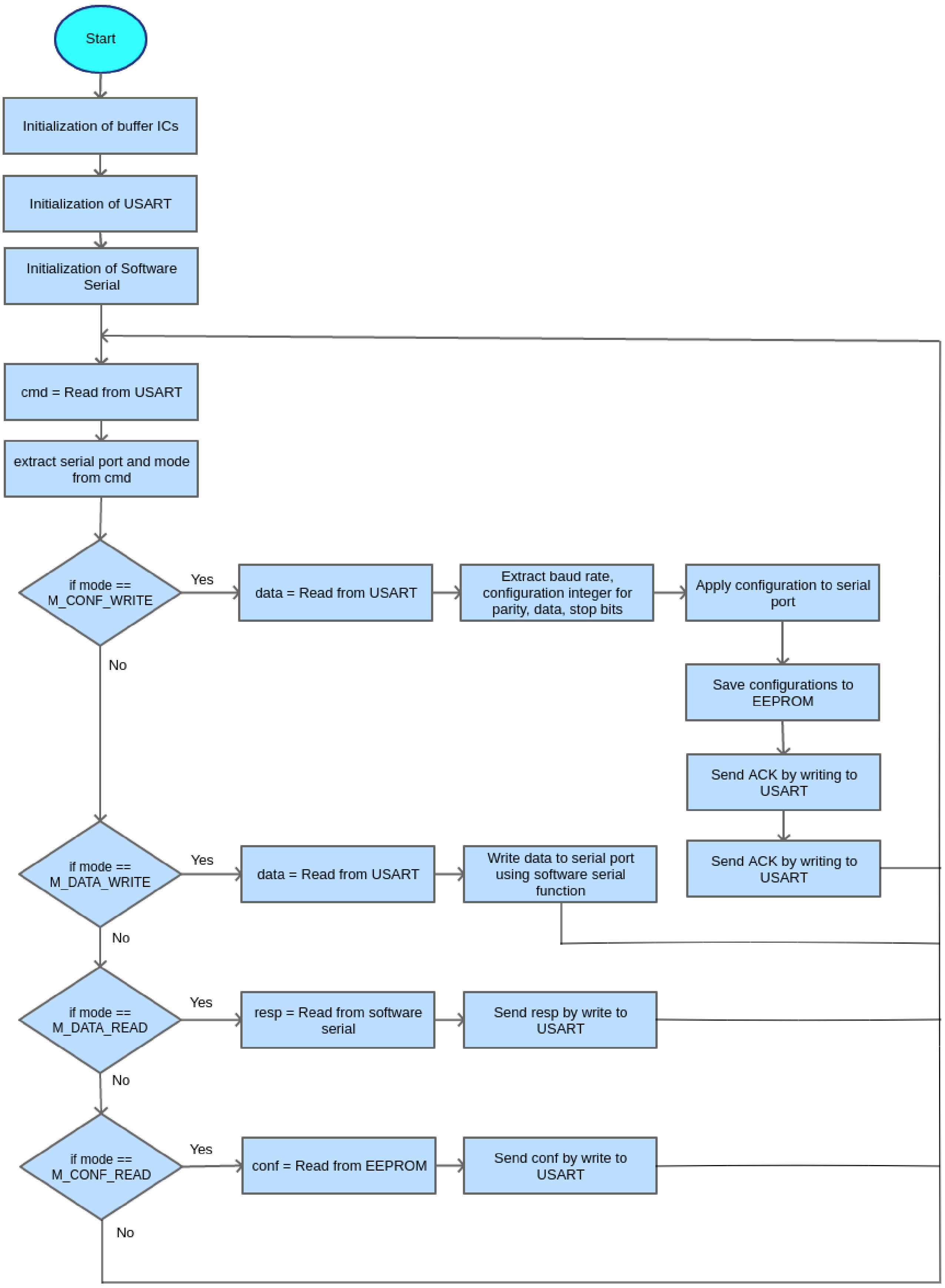

Start
Initialization of buffer ICs
Initialization of USART
Initialization of Software Serial
cmd = Read from USART
extract serial port and mode from cmd
if mode == M_CONF_WRITE
Yes
No
data = Read from USART
Extract baud rate, configuration integer for parity, data, stop bits
Apply configuration to serial port
Save configurations to EEPROM
Send ACK by writing to USART
Send ACK by writing to USART
if mode == M_DATA_WRITE
Yes
No
data = Read from USART
Write data to serial port using software serial function
if mode == M_DATA_READ
Yes
No
resp = Read from software serial
Send resp by write to USART
if mode == M_CONF_READ
Yes
No
conf = Read from EEPROM
Send conf by write to USART

FIGURE 14: FLOW CHART OF THE ALGORITHM

-

-

2.8. Writing Sample Applications

Sample software was written using the library to demonstrate how to include and use the implemented library functions.

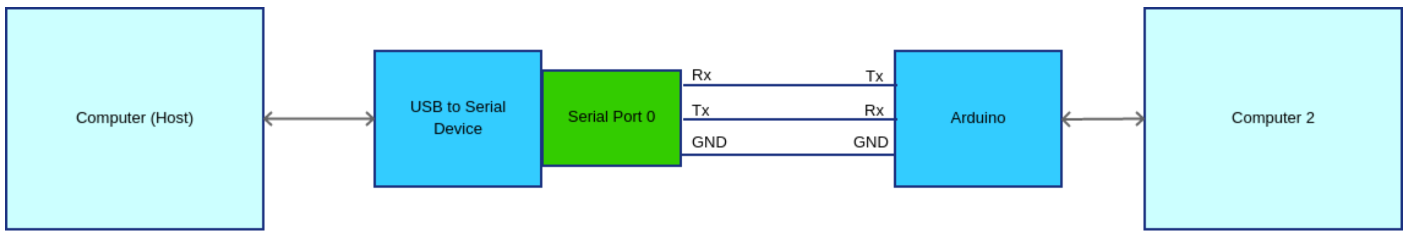


FIGURE 15: BLOCK DIAGRAM

Here, softwares were implement to send messages from the computer (host) to the Arduino device which is connected to another computer using the USB-to-Serial Device which was built.

- **PC-side C/C++ Software**

```
// USB2Serial Library Headers
#include "USB2SerialLib/testlib_usb_to_serial.h"
#include "USB2SerialLib/testlib_usb_to_serial.cpp"
#include <stdio.h>
#include <stdlib.h>

int error(){
     printf("exiting the program.\n");
     return 1;
}

int main(){

     // creating an object
     TestLib_usb_to_serial usb_2_serial;

     printf("[Program started. identifying the device]\n");
     //cout <<"program started."<<endl;
     if(usb_2_serial.device_identify()){
     printf("[device is available at PORT %s]\n",
usb_2_serial.get_device_port());

     //connect to the device
     if(usb_2_serial.device_open()){
          printf("[device is opened and configured.]\n");

          //configure serial 0
          usb_2_serial.set_config(
               0, //serial port 0
               9600, // baud rate
               SERIAL_8N1//serial config 8N1
          );
```

```
        printf("[serial port 0 was configured 9600 Baud, 8N1 ]\n");

        char msg[] = {'h','e','l','l','o','\n'}; //6 chars

        printf("[serial transmission start.]\n");
        //send characters to serial 0
        for(int k=0; k<5; k++){
            for(int i=0; i<6; i++){
                usb_2_serial.serial_write(
                0, //serial port
                msg[i] //data
                );
                printf("%c", msg[i]);
            }
            //delay 1000ms

        }

        printf("[close device connection.]\n");
        //closing device
        usb_2_serial.device_close();

        printf("[exiting the program.]\n");
        return 1;

    }else{
        printf("[can't opened and configure the device.]\n");
        error();
    }

    }else{
    printf("[device is not available]\n");
    error();
    }

    return 0;
}
```

- **Arduino Sketch**

```
#include <CustomSoftwareSerial.h>

CustomSoftwareSerial* customSerial;              // Declare serial

// Init value
void setup() {
  Serial.begin(9600);
  while (!Serial) {
     ; // wait for serial port to connect. Needed for Leonardo only
  }
  Serial.write("Hello World");

  customSerial = new CustomSoftwareSerial(9, 10); // rx, tx
  customSerial->begin(9600, CSERIAL_8N1);       //    Baud    rate:    9600,
configuration: CSERIAL_8N1

}

void loop() {
  if (customSerial->available())
     Serial.write(customSerial->read());

}
```

3. RESULTS AND DISCUSSION

During the initial phase of the project, a feasibility study was carried out on the product in order to determine whether the product could be implemented within the limited timeframe. Subsequently, the project was divided into 8 major tasks (as discussed in Section 2) and further divided into sub-tasks so as to cover all the necessary requirements of the project. Due to the limited time constraint, wide scope and significant workload of the project, the major tasks were divided among the group members depending on their expertise and experience.

Task (1): Design of the overall software and hardware architecture was successfully achieved by designing the overall abstract design of the project upon discussing the intended project requirements with the lecturer-in-charge.

During the hardware design phase, the electronic components required for the hardware design (as described in Section 2.2) were selected, paying careful attention to factors such as power consumption, number of pins, physical arrangement (size) and optimal utilization of the selected hardware. As none of the group members had any prior experience with PCB designing using circuit design software, this stage provided an interesting learning point. Upon familiarizing ourselves with the Autodesk Eagle software, the schematic and PCB designs were successfully designed. Unfortunately, after a thorough inspection, it was observed that the fabricated PCB was not of suitable condition to be used. This resulted in finding an alternative approach. Given the limited time constraints, it was decided to fabricate the circuit using the dotted board. Thereafter, an inspection was carried out to determine the continuity of routes. This included individually checking the continuity and functionality of the electronic components. Upon satisfactory results, we proceeded with the other tasks. Thus, **Task (2): Hardware design and fabrication** was successfully completed as well.

Thereafter, one group member started working on developing the GUI, while another member proceeded with **Task (3): Feasibility study to determine approaches to ensure efficient PC-Arduino communication.** The completion of this task was critical at this stage, as it directly affected the ability to proceed with developing the library and carrying out further developments of the GUI application as well. Therefore, as described in Section 2.3, this task was completed as well.

Prior to continuing with the development of the library and including functionality to the GUI, a high-level protocol was designed to allow messages to be transferred between the PC and device. In Section 2.5, the complete protocol design was described in-depth. This led to the completion of **Task (5): High-level software protocol design** and subsequently it was successfully implemented in both the library and hardware as well.

The GUI application was designed using Qt Creator IDE and C++. Once again, none of the group members were familiar with GUI application design using C++. However upon referring to supporting material, the GUI was designed and developed and all the functionalities described in Section 2.4 were added to that. For the completeness of the GUI, additional Menu bar commands were added, whose functionalities can be implemented as further improvements to the GUI. Therefore, **Task (4): GUI application development** was completed.

The next phase involved developing the C/C++ library for the PC-side. First, a simple set of functions were selected to be included in the library as described in Section 2.6 and subsequently these functions were implemented according to the high-level messaging protocol which was designed in Task (5). The separation of some classes and library files from the original Qt toolkit was necessary to reduce the size of the library. Furthermore, a simple build environment was configured, considering the ease of usage for a potential user. Therefore, **Task (6): Design and development of a C/C++ library for the PC-side** was completed, paying careful attention to the usability of the library.

Task (7): Design and development of an Arduino software was started in parallel with Task (6) and Task (4) and had a number of sub tasks to be covered within that. USART communication library functions were developed from the scratch and the messaging protocol in Arduino's main algorithm was implemented (refer to the flowchart (Figure 14) in Section 2.7). Moreover, software serial interfaces were implemented and functionality to store serial port configurations using EEPROMs were also added, thus completing this task.

Finally, after completing all the requirements of the project, a sample software was included to demonstrate the usage of the library (i.e. how to include and use the implemented library functions). Thus, the final task, **Task (8): Writing sample applications** was completed as well.

Communication between the device and the PC was implemented via serial communication methods. However, instead of using this approach , it would have been better if USB protocol messages were directly used to establish the communication between PC and the device. The reasoning being, having higher level protocols tend to make the communication process slower and limits the usage to the functionalities/ features given by those libraries.

For the hardware design, bi-directional buffer ICs (74HC245) were used as these buffers were readily available. However, it should be noted that these could be replaced with unidirectional buffer ICs as well, as the direction of data flow is not changed within a given IC in the hardware design.

As a team, initially, a shared cloud space (Google Drive) was setup to save files and material necessary for the project. Separate documents were created for discussion, references and allocated tasks. Upon division of the project into tasks, discussions were carried out via the google sheet. This provided an effective platform to brainstorm ideas, and to leave a digital footprint of the ideas, problems and solutions which were discussed. Upon acquiring sufficient knowledge about the project, the tasks were assigned to each member, and subsequently a github repository was created and maintained.

4. FUTURE WORK

Due to the limited timeframe, the proposed solution was designed as a host-centric protocol. Therefore, if a slave device was to send information to a particular serial port, the host cannot access the information unless the host explicitly issues a reads command. Thus, the project could be extended to resolve this issue by using a non-host centric protocol.

In this design, only a single serial port can be used at a time. Thus, this capability can be further improved. Moreover, configuration saving functionalities to the PC can be added into the GUI.

Since, multiple serial ports can be controlled from a single USB port, by extending this functionality with the above mentioned improvements, this design can then be used for larger industrial automation projects.

5. REFERENCES

[1] "74HC245; 74HCT245 Product Datasheet" [Online]. Available: https://assets.nexperia.com/documents/data-sheet/74HC_HCT245.pdf [Accessed: January 1, 2018]

[2] "Arduino Nano" [Online]. Available: https://store.arduino.cc/usa/arduino-nano [Accessed: January 1, 2018]

[3] "The USART of the AVR" [Online]. Available: http://maxembedded.com/2013/09/the-usart-of-the-avr/ [Accessed: January 18, 2018]

[4] "Qt Documentation" [Online]. Available: https://doc.qt.io/qtcreator/index.html [Accessed: January 10, 2018]

[5] "Serial Communication" [Online]. Available: https://learn.sparkfun.com/tutorials/serial-communication/all [Accessed: January 25, 2018]

[6] "SoftwareSerial Library" [Online]. Available: https://www.pjrc.com/teensy/td_libs_SoftwareSerial.html [Accessed: January 22, 2018]

[7] "AltSoftSerial Library" [Online]. Available: https://github.com/PaulStoffregen/AltSoftSerial [Accessed: January 22, 2018]

[8] "CustomSoftwareSerial" [Online]. Available:https://github.com/ledongthuc/CustomSoftwareSerial [Accessed: January 25, 2018]

[9] "EEPROM Library" [Online]. Available: https://www.arduino.cc/en/Reference/EEPROM [Accessed: February 1, 2018]