GUJARAT TECHNOLOGICAL UNIVERSITY

L.D. College of Engineering

A Report on:

Data-Logger
Under subject of DESIGN ENGINEERING 2B
B.E. Semester – 6
Electronics And Communication Engineering

Name of student	Enrolment No.
Himanshu Gautam Raj	220283111009
Parmar Kuldipsinh Hardevsinh	220283111023
Mayurkumar Shaileshkumar Bhagora	210280111115
Bhakti Jayeshbhai Meghani	220283111017

Prof. Kirit V. Patel	Dr. CHANDULAL VITHALANI
(Faculty Guide)	(Head of Department)

CANDIDATE’S DECLARATION

We hereby declare that the work presented in this project entitled **"Data-Logger"** Submitted towards completion of project in Semester 6th of B.E.(Electronic & Communication) is an authentic record of my original work carried out under the guidance of **Professor Kirit V. Patel** During The Academic Year 2023-24. that no part of these DE-2B reports has been directly copied from any students reports or taken from any other source, without providing the reference.

Semester: 6th

College: L.D.College of engineering, Ahmedabad.

Name of student	Enrolment No.
Himanshu Gautam Raj	220283111009
Parmar Kuldipsinh Hardevsinh	220283111023
Mayurkumar Shaileshkumar Bhagora	210280111115
Bhakti Jayeshbhai Meghani	220283111017

ACKNOWLEDGMENT

The success and final outcome of this project required a lot of guidance and assistance from many people and extremely privileged to have got this all along the completion my project.

We respect and thank colleagues for providing us an opportunity to do the project work and giving us all support and guidance which made us complete the project duty.

We owe our deep gratitude to our project guide **Professor Kirit V. Patel**, who took keen interest on our project work and guides us all along, till the completion of our project work by providing all the necessary information for developing a good system. We heartily thank our **HEAD OF DEPT. (Dr. CHANDULAL VITHALANI)** for his guidance and suggestions during this project work.

ABSTRACT

The Data Logger project introduces a sophisticated system for efficient acquisition, storage, and transmission of sensor data. Employing a NodeMCU microcontroller interfaced with sensors and an RTC DS1307 module, the system ensures precise timing and data integrity.

At the heart of the system lies the NodeMCU, which leverages interrupts triggered by the RTC DS1307 to timestamp data readings every second. Concurrently, sensor values are sampled and formatted into strings containing timestamp information and sensor readings.

The NodeMCU's file system (FS) facilitates seamless storage of these formatted strings in text files, offering efficient data management. Serial communication with a PC enables user-defined parameters such as sample rate, stop duration, and sensor selection via Python scripts.

Upon completion of data logging, a Python script converts the stored text file into CSV format for streamlined analysis and visualization. This modular and user-friendly system holds promise for diverse applications, empowering researchers and practitioners with valuable insights into their respective domains.

INDEX

1. INTRODUCTION

In modern engineering and scientific endeavors, the collection and analysis of data are fundamental processes driving innovation and informed decision-making. Data loggers play a pivotal role in this landscape by providing a robust means of capturing, storing, and transmitting data from various sensors and instruments.

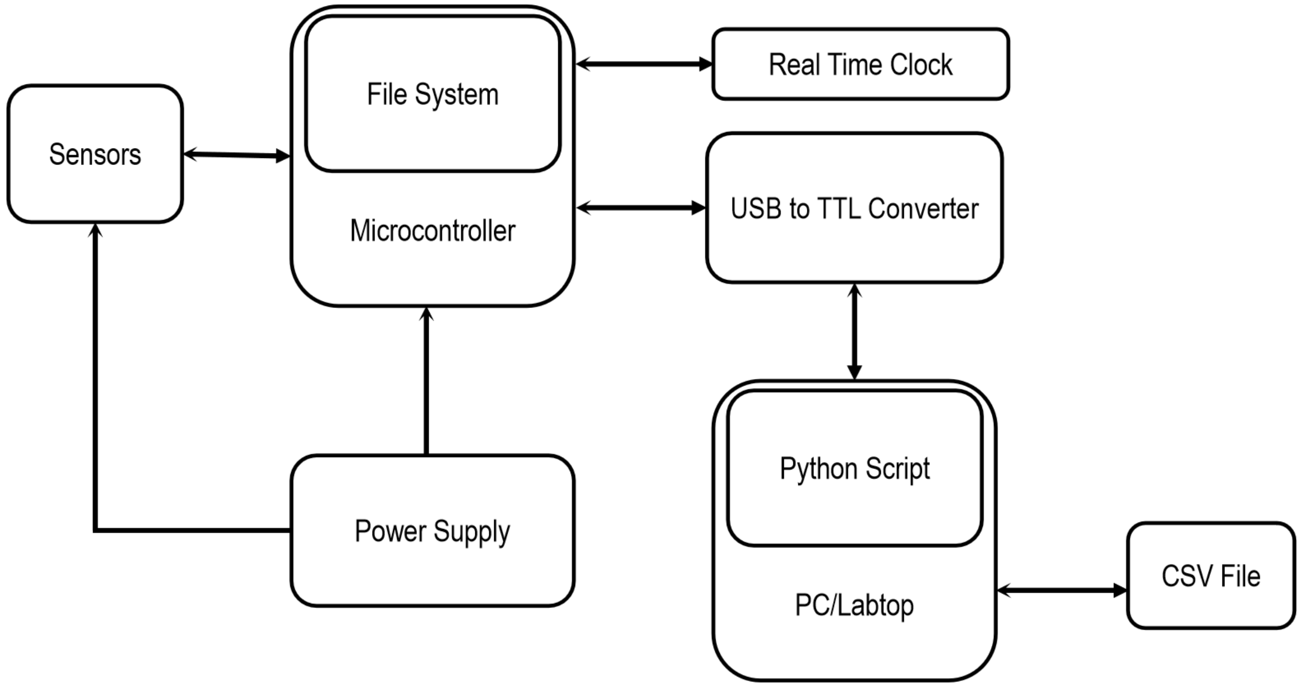
This project report focuses on the design, implementation, and utilization of a data logger system tailored to specific requirements, serving as a critical component in monitoring and analyzing environmental parameters, industrial processes, or experimental conditions. The data logger serves as the nexus between the physical world and digital analysis, enabling researchers, engineers, and practitioners to gather insights, detect trends, and make data-driven decisions.

The primary objectives of this report are to elucidate the functionality and architecture of the data logger system, discuss its design considerations, highlight its key features and capabilities, and present practical applications or case studies demonstrating its efficacy in real-world scenarios. Additionally, this report aims to provide insights into the challenges encountered during the development and deployment phases, along with strategies for optimization and future enhancements.

Through a comprehensive exploration of the data logger system, readers will gain a deeper understanding of its role in contemporary data acquisition processes, its technical specifications, its integration with existing infrastructure, and its potential to revolutionize data-driven endeavors across various domains.

2. Hardware Description

2.1 Block Diagram:



Sensors : Sensor are transducer to provide the different types of physical quantities and environmental change to microcontroller.

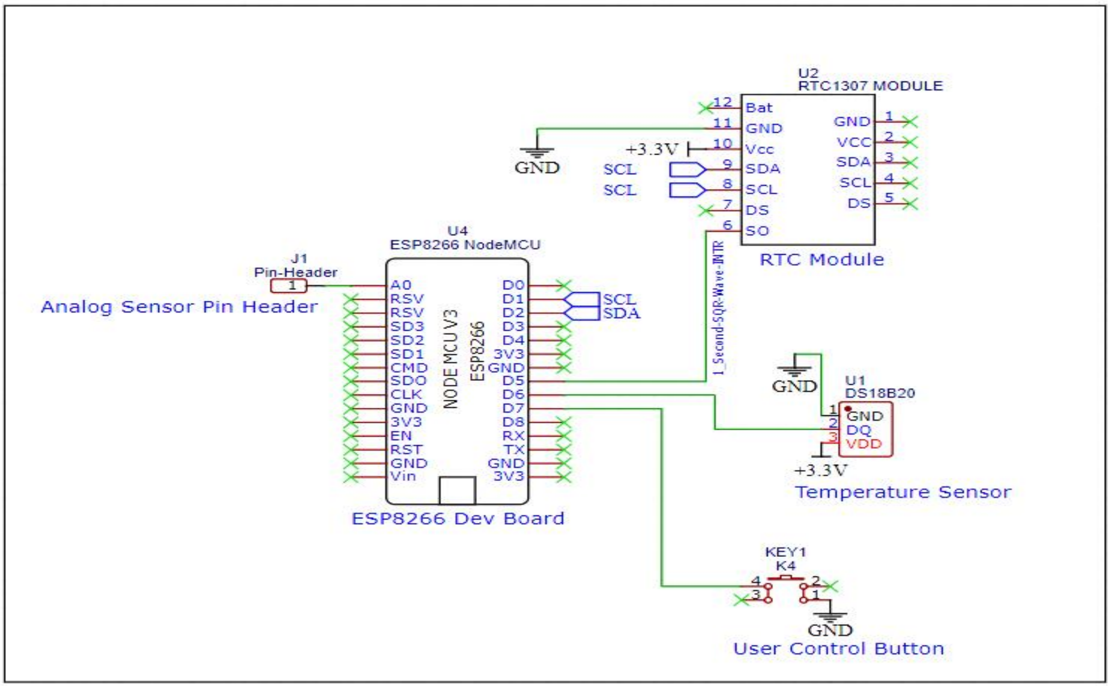
Microcontroller : microcontroller is central processing unit to process the sensor data and make usable as analysis. Here microcontroller read the data from sensor and store this data to inside the microcontroller in txt file format.

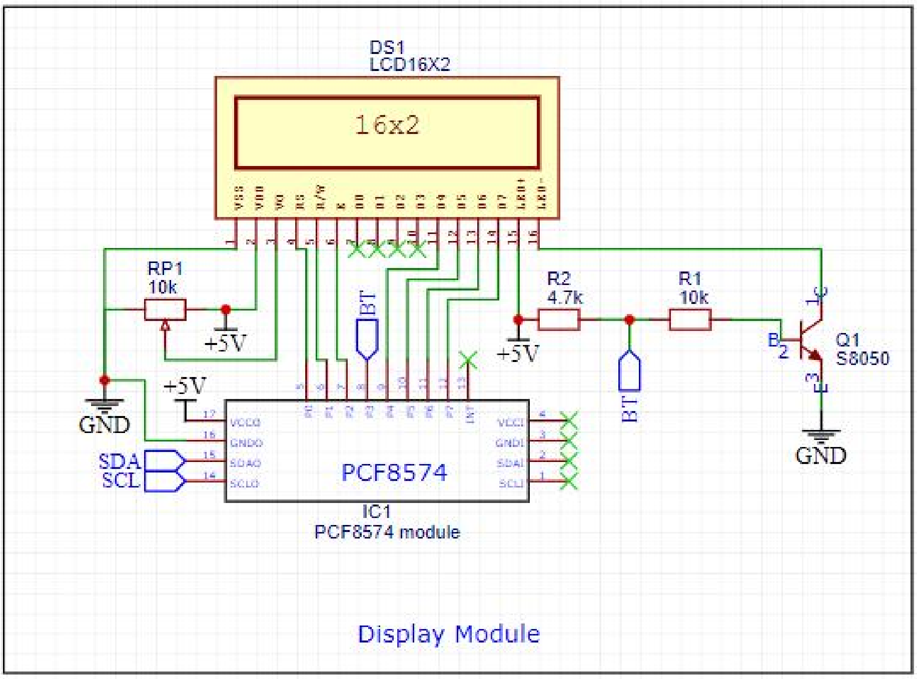
Real Time Clock : RTC provide the current time and date to microcontroller.

USB to TTL Converter : USB to TTL converter provide the interface between PC and Microcontroller.

PC/Laptop : here PC in send the data logging mode to microcontroller to run and completion of data logging we get the data from microcontroller using python script. Here python script store the data as CSV file format.

2.2 Circuit Diagram:





2.3 Component list:

Sr.No	Component Name	Pieces
1	NodeMCU (ESP8266-12E)	1
2	DS18B20 (temperature sensor)	1
3	LCD display (16x2)	1
4	Analog Sensor(LDR,RTD etc)	1
5	Real Time Clock (RTC1307)	1

3. Software Description

3.1 Microcontroller Code:

```
#include <Wire.h>
#include <RTClib.h>
#include <OneWire.h>
#include <DallasTemperature.h>
#include <LiquidCrystal_I2C.h>
#include <ArduinoJson.h>
#include "LittleFS.h"
#include "FS.h"
#include <Arduino.h>

RTC_DS1307 rtc;
DateTime C_time;

OneWire oneWire(D6);  // for the DS18B20 temperature sensor pin connect
DallasTemperature TemSensor(&oneWire);

LiquidCrystal_I2C lcd(0x27, 16, 2);  // lcd initialize
int8_t btnClickPin = D7;             // input pin for button click
int8_t RTCsqrPin = D5;                    // input pin for RTC Square wave pulse count (1Hz
frequency)

// Variable Define
JsonDocument OutgoingData ;
long duration;              // store the total duration in second received from pc
String msg;                // store the mode of microcontroller to perform, which received from pc
int sample;                // store the  time in second received from pc
int8_t modeSelect = -1;      // store the selected mode index
String CSV_Data;            // store the csv data string
long sampleCount = 0;        // store total number of remaining sample counts
volatile int16_t sampleFlag; /* store the total number of pulse get from RTC at every second
            if sampleFlag is equal to sample time, its trigger the data sample capturing */
bool btnFlag = false;         // button flag set during data log to exit data-log mode by user to
press button through button intrrupt
volatile unsigned long last_INTR_Trigger_sqr = 0;
volatile unsigned long last_INTR_Trigger_btn = 0;

// --------------------------- Functions Declaration Start -------------------------

bool CurrentTime(void);            // function for get current time
String ReadBuiltInTemSensor(void);   //function for read the temperature value from sensor
return the CSV string
```

```
String ReadAnalogSensor(void);
void DataReceiveACK_Print(void);    // print the data get received properly to lcd
void DataReceiveError_Print(void);  // print the data not get received properly to lcd
bool DataLogEndPrint(void);
bool dataWrite_File(String);        // function for write the data to txt file
void RTCsqr(void);

//---------------------------- Interrupt function define -------------------------

void IRAM_ATTR RTCsqrCount() {
  unsigned long INTR_trigger_sqr = millis();
  if (INTR_trigger_sqr - last_INTR_Trigger_sqr > 400) {
    sampleFlag += 1;
    // Serial.println("RTCsqr intrrupt");
  }
  last_INTR_Trigger_sqr = INTR_trigger_sqr;
}
void IRAM_ATTR btnFlagSet() {
  unsigned long INTR_trigger_btn = millis();
  if (INTR_trigger_btn - last_INTR_Trigger_btn > 400) {
    btnFlag = true;
    // Serial.println("btnFlagSet intrrupt");
  }
  last_INTR_Trigger_btn = INTR_trigger_btn;
}


// --------------------------------- Setup Start ----------------------------
void setup() {

  // Serial Communication start
  Serial.begin(115200);
  delay(2);

  pinMode(btnClickPin, INPUT_PULLUP);
  pinMode(RTCsqrPin, INPUT_PULLUP);
  pinMode(LED_BUILTIN, OUTPUT);
  digitalWrite(LED_BUILTIN, 1);

  // Serial.println("start.......");
  // LittleFS File system initialize
  if (!LittleFS.begin()) {
    Serial.println("An Error has occurred while mounting LittleFS");
  }
  // Temperature Sensor initialize
  TemSensor.begin();
  delay(2);
  lcd.init();       // lcd initialize
  lcd.backlight();  // Turn on the LCD backlight
  lcd.print("..DATA-LOGGER..");

  Wire.begin();
  RTCsqr();

#ifndef ESP8266
  while (!Serial);  // wait for serial port to connect. Needed for native USB
#endif

  if (!rtc.begin()) {
    Serial.println("Couldn't find RTC");
    Serial.flush();
    // abort();
  }

  if (!rtc.isrunning()) {
    Serial.println("RTC is NOT running, let's set the time!");
    rtc.adjust(DateTime(F(__DATE__), F(__TIME__)));
  }

  deserializeJson(OutgoingData, "{\"msg\":\"msg\"}");

  File csvFile = LittleFS.open("/SensorData.txt", "a");
  csvFile.close();

  File rstFile = LittleFS.open("/rst.txt", "a");
  rstFile.print(String(ESP.getResetReason()).c_str());
  rstFile.close();

  delay(2000);

// Serial.println("Reset reason: " + String(ESP.getResetReason()));

}

// ------------------------------- Loop Start -----------------------------

void loop() {

  // variable define here because reduce RAM overload

  if (!Serial.available()) {
    lcd.clear();
    lcd.setCursor(0, 0);
    lcd.print("   UART MODE");
    lcd.setCursor(0, 1);
    lcd.print("Buad-Rate:115200");
    delay(1000);

    lcd.clear();
    lcd.setCursor(0, 0);
    lcd.print("Waiting For Data");
    lcd.setCursor(0, 1);
    lcd.print("  ............");
    delay(1000);
  }

  else if (Serial.available()) {
  //  long j = millis();
    JsonDocument IncomingData;
    String jsonData = Serial.readString();
    // Serial.println(jsonData);
    DeserializationError error = deserializeJson(IncomingData, jsonData);
    if (error) {
      Serial.print("deserializeJson() failed: ");
      Serial.println(error.c_str());
    }

    msg = IncomingData["msg"].as<String>();
    sample = IncomingData["sample"];
    duration = IncomingData["duration"];

    if (msg == "File Transfer Mode") {
     // Write code for file transfer to pc
      lcd.clear();
      lcd.setCursor(0, 0);
      lcd.print("File Transfering");
      OutgoingData["msg"]="ACK";
      serializeJson(OutgoingData, Serial);
      while(!Serial.available()){ yield();}
      String jsonData = Serial.readString();
      DeserializationError error = deserializeJson(OutgoingData, jsonData);
       if (error) {
      Serial.print("deserializeJson() failed: ");
      Serial.println(error.c_str());}
      String data=OutgoingData["msg"].as<String>();
      if (data == "R") {
      File csvFile = LittleFS.open("/SensorData.txt", "r");
          while(1){
                String fileContent = csvFile.readStringUntil('\n');
                OutgoingData["msg"]=fileContent;
```

```
        serializeJson(OutgoingData, Serial);
        Serial.write("\r\n");
        Serial.flush();
          if (fileContent=="END"){break;}
        yield();
     }
     csvFile.close();
       csvFile = LittleFS.open("/SensorData.txt", "a");
      if (csvFile) {
       csvFile.print("START\n");
       csvFile.print("DATA DOESN'T EXIT..\n");
       csvFile.print("END\n");
       csvFile.close();
                }
     lcd.setCursor(0, 1);
     lcd.print(" ..Completed...");
     delay(10000);
   }
   else {
    OutgoingData["msg"]="NACK";
    serializeJson(OutgoingData, Serial);
    DataReceiveError_Print();
   }
   modeSelect = -1;
  }

  else if (msg == "Temperature Data-Log") {
// Write code for Temperature data log
   if (sample && duration) {
    Serial.write("ACK\n");
    DataReceiveACK_Print();
    delay(10);
    modeSelect = 1;

    lcd.clear();
    lcd.setCursor(0, 0);
    lcd.print("Temperature Log");
    delay(3000);
    lcd.clear();
    lcd.setCursor(0, 0);
    lcd.print("Sample T:");
    lcd.setCursor(9, 0);
    lcd.print(sample);
    lcd.setCursor(0, 1);

    if(duration >= 60 ){
     lcd.print("Duration(M):");
```

```
        int i = int(duration / 60);
        lcd.setCursor(12, 1);
        lcd.print(i);}

      else{
        lcd.print("Duration(S):");
        lcd.setCursor(12, 1);
        lcd.print(duration);
      }

      LittleFS.remove("/SensorData.txt");
      delay(3000);
    }

    else {
      Serial.write("NACK\n");
      DataReceiveError_Print();
    }
  }

  else if (msg == "Analog Input Data-Log") {
    if (sample && duration) {
      Serial.write("ACK\n");
      // Write code for Analog data log
      DataReceiveACK_Print();
      delay(10);
      modeSelect = 2;

      lcd.clear();
      lcd.setCursor(0, 0);
      lcd.print("Analog Input Log");
      delay(3000);
      lcd.clear();
      lcd.setCursor(0, 0);
      lcd.print("Sample T:");
      lcd.setCursor(9, 0);
      lcd.print(sample);
      lcd.setCursor(0, 1);

       if(duration >= 60 ){
        lcd.print("Duration(M):");
        int i = int(duration / 60);
        lcd.setCursor(12, 1);
        lcd.print(i);}

      else{
        lcd.print("Duration(S):");
```

```
         lcd.setCursor(12, 1);
         lcd.print(duration);
       }

       LittleFS.remove("/SensorData.txt");
       delay(3000);

     } else {
       Serial.write("NACK\n");
       DataReceiveError_Print();
     }
   }

   else {
     Serial.write("NACK\n");
     DataReceiveError_Print();
   }
 }


 // --------------------- code start to run selected mode ----------------------------

 if (!(modeSelect == -1)) {

   sampleCount = int(duration / sample);
   // Serial.println(sampleCount);
   sampleFlag = 0;
   btnFlag = false;

   //code for asking to user for start the data log
   lcd.clear();
   lcd.setCursor(0, 0);
   lcd.print("Press Button");
   lcd.setCursor(0, 1);
   lcd.print("To Start");

    digitalWrite(LED_BUILTIN, 0);
    delay(100);
   digitalWrite(LED_BUILTIN, 1);
   // Serial.println("waiting btn press");
   while (!(digitalRead(btnClickPin) == 0)) {
     yield();
     continue;
   }
   delay(1000);
   // Serial.println("--- btn press");
```

```
    // --------code for check the selected mode and start the data logging after user press button
------

    // temperature data log
    if (modeSelect == 1) {
      // write code for Temperature data log
      // Serial.println("temperature mode");
      lcd.clear();
      lcd.setCursor(0, 0);
      lcd.print("Temperature Log");
      lcd.setCursor(0, 1);
      lcd.print("Count:");
      lcd.setCursor(6, 1);
      lcd.print(sampleCount);

       File csvFile = LittleFS.open("/SensorData.txt", "a");
          if (csvFile) {
             csvFile.print("START\n");
            //  Serial.println("In the START Write");

             csvFile.close();
                 }

      attachInterrupt(digitalPinToInterrupt(btnClickPin), btnFlagSet, FALLING);
      attachInterrupt(digitalPinToInterrupt(RTCsqrPin), RTCsqrCount, FALLING);
      // Serial.println("Intrrupt attach");
      while (1) {
        // continuously store the data log in file until total sample completed
        if ((sampleFlag == sample) && !(sampleCount <= 0)) {
          digitalWrite(LED_BUILTIN, 0);
          lcd.clear();
          lcd.setCursor(0, 0);
          lcd.print("Temperature Log");
          lcd.setCursor(0, 1);
          lcd.print("Count:");
          sampleCount -= 1;
          lcd.setCursor(6, 1);
          lcd.print(sampleCount);
          // Serial.println("1 condition");
          CurrentTime();
          // Serial.println("step 2");
          CSV_Data = ReadBuiltInTemSensor();
          // Serial.println(CSV_Data);
          // Serial.println("step 3");
          dataWrite_File(CSV_Data);
          // Serial.println("step 4");
          sampleFlag = 0;
          // Serial.println("step 5");
          yield();
        }
        digitalWrite(LED_BUILTIN, 1);
        // condition for check to total sample completed it exit the data log mode
        if (sampleCount <= 0) {
          // Serial.println("2 condition");
          File csvFile = LittleFS.open("/SensorData.txt", "a");
          if (csvFile) {
             csvFile.print("END\n");
             csvFile.close();
                 }
          DataLogEndPrint();
          sampleFlag = 0;
          break;
        }
        // condition for check the user want exit or not
        if (btnFlag == true) {
          // Serial.println("3 condition");
           detachInterrupt(digitalPinToInterrupt(RTCsqrPin));   // Get off the RTC Square pulse
intrrupt avoid error in code
          btnFlag = false;
          lcd.clear();
          lcd.setCursor(0, 0);
          lcd.print("You want To Exit");
          lcd.setCursor(0, 1);
          lcd.print(" [Press Button] ");
          delay(9000);  // wait 9 second to second click of usser for exit
          yield();
          if (btnFlag == true) {
            btnFlag = false;
            File csvFile = LittleFS.open("/SensorData.txt", "a");
             if (csvFile) {
             csvFile.print("END\n");
             csvFile.close();
                 }
            break;
          }

          lcd.clear();
          lcd.setCursor(0, 0);
          lcd.print("Temperature Log");
          lcd.setCursor(0, 1);
          lcd.print("Count:");
            attachInterrupt(digitalPinToInterrupt(RTCsqrPin), RTCsqrCount, FALLING);  // Get on
the RTC Square pulse intrrupt to run again normally
        }
```

```
    yield();
    }
    detachInterrupt(digitalPinToInterrupt(btnClickPin));
    detachInterrupt(digitalPinToInterrupt(RTCsqrPin));
    modeSelect = -1;
  }
  // analog data log
  else if (modeSelect == 2) {
    // write code for Analog data log
    // Serial.println("Analog data mode");
    lcd.clear();
    lcd.setCursor(0, 0);
    lcd.print("Analog Value Log");
    lcd.setCursor(0, 1);
    lcd.print("Count:");
    lcd.setCursor(6, 1);
    lcd.print(sampleCount);

    File csvFile = LittleFS.open("/SensorData.txt", "a");
        if (csvFile) {
          csvFile.print("START\n");
          csvFile.close();
            }

    attachInterrupt(digitalPinToInterrupt(btnClickPin), btnFlagSet, FALLING);
    attachInterrupt(digitalPinToInterrupt(RTCsqrPin), RTCsqrCount, FALLING);
    // Serial.println("Intrrupt attach");
    while (1) {
      // continuously store the data log in file until total sample completed
      if ((sampleFlag == sample) && !(sampleCount <= 0)) {
        digitalWrite(LED_BUILTIN, 0);
        lcd.clear();
        lcd.setCursor(0, 0);
        lcd.print("Analog Value Log");
        lcd.setCursor(0, 1);
        lcd.print("Count:");
        sampleCount -= 1;
        lcd.setCursor(6, 1);
        lcd.print(sampleCount);
        // Serial.println("1 condition");
        CurrentTime();
        // Serial.println("step 2");
        CSV_Data = ReadAnalogSensor();
        // Serial.println("step 3");
        dataWrite_File(CSV_Data);
        // Serial.println("step 4");
        sampleFlag = 0;
        // Serial.println("step 5");
        yield();
      }
      digitalWrite(LED_BUILTIN, 1);
      // condition for check to total sample completed it exit the data log mode
      if (sampleCount <= 0) {
        // Serial.println("2 condition");
        DataLogEndPrint();
        sampleFlag = 0;
        File csvFile = LittleFS.open("/SensorData.txt", "a");
        if (csvFile) {
          csvFile.print("END\n");
          csvFile.close();
            }
        break;
      }
      // condition for check the user want exit or not
      if (btnFlag == true) {
        // Serial.println("3 condition");
            detachInterrupt(digitalPinToInterrupt(RTCsqrPin));  // Get off the RTC Square pulse intrrupt avoid error in code
        btnFlag = false;
        lcd.clear();
        lcd.setCursor(0, 0);
        lcd.print("You want To Exit");
        lcd.setCursor(0, 1);
        lcd.print(" [Press Button] ");
        delay(9000);  // wait 9 second to second click of usser for exit
        yield();
        if (btnFlag == true) {
          btnFlag = false;
        File csvFile = LittleFS.open("/SensorData.txt", "a");
        if (csvFile) {
          csvFile.print("END\n");
          csvFile.close();
            }
          break;
        }

        lcd.clear();
        lcd.setCursor(0, 0);
        lcd.print("Analog Value Log");
        lcd.setCursor(0, 1);
        lcd.print("Count:");
          attachInterrupt(digitalPinToInterrupt(RTCsqrPin), RTCsqrCount, FALLING);  // Get on the RTC Square pulse intrrupt to run again normally
      }
```

```
        yield();
      }
      detachInterrupt(digitalPinToInterrupt(btnClickPin));
      detachInterrupt(digitalPinToInterrupt(RTCsqrPin));
      modeSelect = -1;
    }

    else {
      modeSelect = -1;
      DataReceiveError_Print();
    }
  }
}


// --------------- Functions Start ---------------

bool CurrentTime(void) {
  C_time = rtc.now();
  return true;
}

void DataReceiveACK_Print(void) {
  lcd.clear();
  lcd.setCursor(0, 0);
  lcd.print(".Data Received..");
  delay(2000);
  lcd.clear();
  lcd.setCursor(0, 0);
  lcd.print("Mode Initialize");
  for (int i = 0; i < 16; i++) {
    yield();
    lcd.setCursor(i, 1);
    lcd.print(".");
    delay(300);
  }
}

void DataReceiveError_Print(void) {
  for (int i = 0; i < 3; i++) {
    yield();
    lcd.clear();
    lcd.setCursor(0, 0);
    lcd.print("..Data Error....");
    lcd.setCursor(0, 1);
    lcd.print("..Send Again....");
    delay(2000);
  }
}

bool DataLogEndPrint(void) {
  detachInterrupt(digitalPinToInterrupt(btnClickPin));
  lcd.clear();
  lcd.setCursor(0, 0);
  lcd.print(".. Completed ...");
  lcd.setCursor(0, 1);
  lcd.print("Press OK To EXIT");
  while (1) {
    yield();
    if (digitalRead(btnClickPin) == 0) {
      break;
    }
  }
  delay(300);
  attachInterrupt(digitalPinToInterrupt(btnClickPin), btnFlagSet, FALLING);
  return true;

}

String ReadBuiltInTemSensor(void) {
  TemSensor.requestTemperatures();
  delay(1);
  float value = TemSensor.getTempCByIndex(0);

  return (String(C_time.timestamp(DateTime::TIMESTAMP_DATE)) + "," +
String(C_time.timestamp(DateTime::TIMESTAMP_TIME)) + "," + String(value) + "\n");
}

String ReadAnalogSensor(void) {

  float value = analogRead(2);

          return      (String(C_time.timestamp(DateTime::TIMESTAMP_DATE))      +      ","      +
String(C_time.timestamp(DateTime::TIMESTAMP_TIME)) + "," + String(value) + "\n");
}

bool dataWrite_File(String data) {
  //  write code for write data to file
  File csvFile = LittleFS.open("/SensorData.txt", "a");
  if (csvFile) {
    csvFile.print(data);
    // Serial.println("Data write in file");
    csvFile.close();
    }
    else{
      return false;
    }
  // Serial.println(data);
  return true;
}

void RTCsqr(void) {
  Wire.beginTransmission(0x68);
  Wire.write(7);
  Wire.write(0x10);
  Wire.endTransmission();
}
```

3.2 Python Script:

3.1.1 Mode Initializing Script:

```
import tkinter as tk
from tkinter import ttk
import re  # For regular expressions
import serial
from serial.tools import list_ports
import json
import time

port=str()
baudR=str()
sample=str()
duration=str()

# Function to send data to the microcontroller
def send_data():
    # Collect data from the entry widgets
    global sample
    global duration
    global baudR
    sample = sample_entry.get()
    duration = duration_entry.get()
    msg=msg_combobox.get()

    # Validate input
    try:
        sample=int(sample)
        if not (sample>0 and sample<=3600):
            alert0.config(text="Invalid Sample Rate")
            return
        else:
            alert0.config(text=" ")
    except ValueError:
        alert0.config(text="Invalid Sample Rate")
        return

    if not re.match(r"^\d{2}:\d{2}:\d{2}$", duration):
        alert1.config(text="Invalid Duration")
        return

    try:
        baudR=int(baudR_combobox.get())
        if(baudR == 0):
            baudRmsg.config(text=ValueError)
            return
    except ValueError:
        baudRmsg.config(text=ValueError)
        return

    duration=duration.split(":")

    durationInSec=(int(duration[0])*3600) + (int(duration[1])*60) + int(duration[2])

    # Convert the data to a string format suitable for serial transmission
    serial_data =json.dumps( {"msg":msg,"sample":sample,"duration":durationInSec})
    print(serial_data)
    # Set up the serial connection (adjust the port and baudrate as needed)
    ser = serial.Serial(port,baudrate=baudR , timeout=1)
    ser.write(serial_data.encode())  # Send data as bytes
    ser.flush()
    time.sleep(1.11)
    data = ser.readline()
    print(data)


# {"sensor":"gps","time":1351824120,"data":[48.756080,2.302038]}
# {"msg":"Temperature Data-Log","sample":1,"duration":30}

# Create the main window
root = tk.Tk()
root.geometry("350x500")
root.configure(bg='aliceblue')
root.resizable(False, False)
root.title("Data Logger Mode transfer")

# Create a custom style for the button
style = ttk.Style()
style.configure("Custom.TButton", background="#3b445c",font=("Arial
Bold",10),foreground="#030303")

# function defination

def validate_sample():
    global sample
    sample=sample_entry.get()
    try:
        sample=int(sample)
        if not (sample>0 and sample<=3600):
            alert0.config(text="Invalid Sample Rate")
            return True
        else:
            alert0.config(text=" ")
            return True
    except ValueError:
        alert0.config(text="Invalid Sample Rate")
        return True

def validate_duration():
    P=duration_entry.get()
    if not re.match(r"^\d{2}:\d{2}:\d{2}$", P):
        alert1.config(text="Invalid Duration")
        return True
    else:
        alert1.config(text=" ")
        return True

def serial_ports():
    return serial.tools.list_ports.comports()

def com_select(event):
    global port
    port =""
    for i in com_combobox.get():
        if i==" ":
            break
        port += i

def baudR_select(event):
    global baudR
    try:
        baudR=int(baudR_combobox.get())
        baudRmsg.config(text=" ")

    except ValueError:
        baudRmsg.config(text=ValueError)



# Dropdown menu for mode selection
msg_var = tk.StringVar()
msg_var.set("Temperature Data-Log")  # Default value
msg_options = ["Temperature Data-Log", "Analog Input Data-Log"]

msg_label = ttk.Label(root, text="Select Mode",font=("Arial
Bold",10),foreground="#030303",background="aliceblue")
msg_label.pack(padx=10, pady=5)

msg_combobox = ttk.Combobox(root, textvariable=msg_var,
values=msg_options,state="readonly",width=30)
msg_combobox.pack(padx=10, pady=5)

# Dropdown menu for baud rate selection
baudR_var = tk.StringVar()
baudR_var.set("")  # Default value
baudR_options = ["2400", "9600","19200","38400","57600","115200"]
baudR_label = ttk.Label(root, text="Select Baud Rate",font=("Arial
Bold",10),foreground="#030303",background="aliceblue")
baudR_label.pack(padx=10, pady=5)

baudR_combobox = ttk.Combobox(root, textvariable=baudR_var,
values=baudR_options,state="readonly",width=30)
baudR_combobox.pack(padx=10, pady=5)
baudR_combobox.bind("<<ComboboxSelected>>", baudR_select)

baudRmsg=ttk.Label(root,background='aliceblue',foreground='#880808')
baudRmsg.pack(padx=10,pady=5)

# Dropdown menu for com ort selection
com_label = ttk.Label(root, text="Select COM Port",font=("Arial
Bold",10),foreground="#030303",background="aliceblue")
com_label.pack(padx=10, pady=5)

com_combobox = ttk.Combobox(root, values=serial_ports(),state="readonly",width=30)
com_combobox.pack(padx=10, pady=5)
com_combobox.bind("<<ComboboxSelected>>", com_select)

# Create labels and entry widgets for each data field
```

```
sample_label = ttk.Label(root, text="Sampling Time (1 to 3600S)",font=("Arial
Bold",10),foreground="#030303",background="aliceblue")
sample_label.pack(padx=10, pady=5)
sample_entry =
  ttk.Entry(root,width=30,validate="focusout",validatecommand=validate_sample)
sample_entry.pack(padx=10, pady=5)

alert0=ttk.Label(root,background='aliceblue',foreground='#880808')
alert0.pack(padx=10,pady=5)

duration_label = ttk.Label(root, text="Data Logging Duration (HH:MM:SS)",font=("Arial
Bold",10),foreground="#030303",background="aliceblue")
duration_label.pack(padx=10, pady=5)
duration_entry =
ttk.Entry(root,validate="focusout",width=30,validatecommand=validate_duration)
duration_entry.pack(padx=10, pady=5)

alert1=ttk.Label(root,background='aliceblue',foreground='#880808')
alert1.pack(padx=10,pady=5)

# Create a send button
send_button = ttk.Button(root, text="Send",style="Custom.TButton", command=send_data)
send_button.pack(padx=10, pady=30)

# Run the application
root.mainloop()
```

3.2.2 MCU to PC file transfer Script:

```
import tkinter as tk
from tkinter import ttk
import serial
from serial.tools import list_ports
import json
import time
from tkinter.filedialog import asksaveasfile
import csv

port=str()
baudR=str()
filePath=str()
msg=dict()

# Function to send data to the microcontroller
def send_data():
    global baudR
    global msg
    global port
    global filePath

    # Validate input
    try:
        baudR=int(baudR_combobox.get())
        if(baudR == 0):
            baudRmsg.config(text=ValueError)
            return
    except ValueError:
        baudRmsg.config(text=ValueError)
        return
# {"msg":"File Transfer Mode"}
    # Convert the data to a string format suitable for serial transmission
    msg =json.dumps( {"msg":"File Transfer Mode"})
    print(msg)
    # Set up the serial connection (adjust the port and baudrate as needed)
    ser = serial.Serial(port,baudrate=baudR , timeout=1)
    ser.write(msg.encode())  # Send data as bytes
    time.sleep(1.5)

    msg  = ser.readline()
    print(msg)
    try:
         msg = json.loads(msg)
    except json.JSONDecodeError:
                print("Waiting for ACK")
                return
    except KeyError:
                print("Key 'msg' not found in JSON")
                return

    if msg["msg"] == "ACK":
                # print("ACK condition")
                msg = json.dumps({"msg": "R"})
                ser.write(msg.encode())
    elif msg["msg"] == "NACK":
                alert.config(text="Data error. Send Request Again", foreground='#880808')
                return
    else:
                alert.config(text="Data error. Send Request Again", foreground='#880808')
                return

    try:
        file = open(filePath, "w", newline='')  # Open file in write mode
        csvFile = csv.writer(file)
        csvFile.writerow(["Date(DD-MM-YYYY)", "Time(HH:MM:SS)", "Sensor Value"])

        while True:
            time.sleep(0.1)
            msg = ser.readline().strip()  # Read a line and remove leading/trailing whitespace
            try:
                msg = json.loads(msg)
            except json.JSONDecodeError:
                print("JSON decoding error:", msg)
                continue  # Skip to the next iteration of the loop if decoding fails

            print("Received:", msg)

            if msg.get("msg") == "START":
                alert.config(text="Data Transfer started..", foreground='#020812')
                while True:
                    time.sleep(0.02)
                    msg = ser.readline().strip()
                    try:
                        msg = json.loads(msg)
                    except json.JSONDecodeError:
                        print("JSON decoding error:", msg)
                        continue  # Skip to the next iteration of the loop if decoding fails

                    print("Received:", msg)

                    if msg.get("msg") == "END":
                        alert.config(text="Data Received Successfully", foreground='#016113')
                        break  # Exit the inner loop when "END" message is received

                    data = str(msg.get("msg")).split(",")
                    print("Data:", data)
                    csvFile.writerow(data)

                break  # Exit the outer loop when "END" message is received

    except Exception as e:
       print("An error occurred:", e)
    finally:
        file.close()  # Close the file regardless of whether an error occurred

# {"sensor":"gps","time":1351824120,"data":[48.756080,2.302038]}
# {"msg":"Temperature Data-Log","sample":1,"duration":120}

# Create the main window
root = tk.Tk()
```

```
root.geometry("360x500")
root.configure(bg='aliceblue')
root.resizable(False, False)
root.title("Data Logger File Transfer Mode")

# Create a custom style for the shart button
style = ttk.Style()
style.configure("Custom.TButton", background="#3b445c",font=("Arial
Bold",11),foreground="#030303")

# function defination
def select_path(event=None):
    file = asksaveasfile(initialfile = 'Untitled.csv',defaultextension='.csv')
    name=str(file)
    name=name.split("'")
    global filePath
    filePath = name[1]
    if file:
        path_entry.delete(0, tk.END)  # Clear any existing text
        path_entry.insert(0, file)  # Insert the selected file path

def serial_ports():
    return serial.tools.list_ports.comports()

def com_select(event):
    global port
    port =""
    for i in com_combobox.get():
        if i==" ":
            break
        port += i

def baudR_select(event):
    global baudR
    try:
        baudR=int(baudR_combobox.get())
        baudRmsg.config(text=" ")

    except ValueError:
        baudRmsg.config(text=ValueError)



# File save path
File_label = ttk.Label(root, text="Select the File Path",font=("Arial
Bold",10),foreground="#030303",background="aliceblue")
File_label.grid(row=0, column=0,columnspan=2, padx=5, pady=5)
```

```
path_entry = ttk.Entry(root, width=40)
path_entry.grid(row=1, column=0, padx=10, pady=5)

browse_button = ttk.Button(root, text='Browse', command=select_path)
browse_button.grid(row=1, column=1, pady=5,padx=5)

# Dropdown menu for baud rate selection
baudR_var = tk.StringVar()
baudR_var.set("")  # Default value
baudR_options = ["2400", "9600","19200","38400","57600","115200"]
baudR_label = ttk.Label(root, text="Select Baud Rate",font=("Arial
Bold",10),foreground="#030303",background="aliceblue")
baudR_label.grid(row=2, column=0, columnspan=2,padx=10,pady=5)

baudR_combobox = ttk.Combobox(root, textvariable=baudR_var,
values=baudR_options,state="readonly",width=30)
baudR_combobox.grid(row=3, column=0,columnspan=2, padx=10,pady=5)
baudR_combobox.bind("<<ComboboxSelected>>", baudR_select)

baudRmsg=ttk.Label(root,background='aliceblue',foreground='#880808')
baudRmsg.grid(row=4, column=0,columnspan=2,padx=10,pady=5)

# Dropdown menu for com ort selection
com_label = ttk.Label(root, text="Select COM Port",font=("Arial
Bold",10),foreground="#030303",background="aliceblue")
com_label.grid(row=5, column=0,columnspan=2,padx=10,pady=5)

com_combobox = ttk.Combobox(root, values=serial_ports(),state="readonly",width=30)
com_combobox.grid(row=6, column=0,columnspan=2,padx=10,pady=5)
com_combobox.bind("<<ComboboxSelected>>", com_select)

# Create a send button
send_button = ttk.Button(root, text="Start",style="Custom.TButton", command=send_data)
send_button.grid(row=7, column=0,columnspan=2,padx=10,pady=20,ipadx=5,ipady=5)

alert=ttk.Label(root,text="",font=("Arial Bold",10),background='aliceblue',foreground='#016113')
alert.grid(row=8, column=0,columnspan=2,padx=10,pady=40)

# Run the application
root.mainloop()
```

4. Advantages

- **Continuous Monitoring:** Data loggers enable uninterrupted data collection over extended periods, ensuring comprehensive and accurate data capture without the need for constant human supervision.

- **Versatility:** They can be deployed in diverse environments and applications, from remote field sites to industrial settings, providing flexibility and adaptability to different monitoring needs.

- **Data Integrity:** Data loggers maintain data integrity by securely storing information locally, reducing the risk of data loss or corruption compared to manual recording methods.

- **Efficiency:** Automation of data collection processes minimizes human error, streamlining workflows and improving efficiency in data management and analysis.

- **Cost-Effective:** With reusable hardware and scalable software solutions, data loggers offer cost-effective monitoring solutions over their operational lifespan, making them economically viable for long-term projects.

5. Conclusion

In conclusion, the data logger system presented in this project report serves as a pivotal tool in modern engineering and scientific endeavors. Through its robust design, continuous monitoring capabilities, and versatility, the data logger offers significant advantages in data acquisition and analysis across diverse applications.

By providing a means for precise, real-time data collection and remote access, the data logger enhances efficiency, reliability, and cost-effectiveness in monitoring processes. Its integration with electronic systems and software platforms further amplifies its utility, enabling seamless data sharing, analysis, and integration into existing workflows.

As demonstrated through practical applications or case studies, the data logger system proves its efficacy in addressing complex monitoring challenges, driving innovation, and facilitating data-driven decision-making.

6. Reference

- Introduction of data logger
  https://en.wikipedia.org/wiki/Data_logger
  https://youtu.be/GM5w6o6z4rg?feature=shared

- NodeMCU
  https://randomnerdtutorials.com/esp8266-pinout-reference-gpios/
  https://randomnerdtutorials.com/interrupts-timers-esp8266-arduino-ide-nodemcu/
  https://randomnerdtutorials.com/esp32-write-data-littlefs-arduino/

- RTC DS1307
  https://randomnerdtutorials.com/guide-for-real-time-clock-rtc-module-with-arduino-ds1307-and-ds3231/

- Tkinter
  https://docs.python.org/3/library/tkinter.html