**ETHERNET BASED REMOTE TEMPERATURE**

**MONITORING SYSTEM**

Project submitted to the

SRM University – AP, Andhra Pradesh

for the partial fulfillment of the requirements to award the degree of

**Bachelor of Technology**

In

**Electronics and Communication Engineering**

**School of Engineering and Sciences**

Submitted by

**P. Sudeep Reddy(AP22110020104)**

**A picture containing text

Description automatically generated**

Under the Guidance of

**Dr. Ramakrishnan Maharajan**

**SRM University–AP**

**Neerukonda, Mangalagiri, Guntur**

**Andhra Pradesh – 522 240**

# Certificate

Date: 30-Jan-24

This is to certify that the work present in this Project entitled entitled “ethernet based remote temperature monitoring system” has been carried out by P. sudeep, P.jayanth , S. Likith under my/our supervision. The work is genuine, original, and suitable for submission to the SRM University – AP for the award of Bachelor of Technology in the **School of Engineering and Sciences**.

**Supervisor**

(Signature)

Prof. / Dr. Ramakrishna Maharajan

Associate Professor/ECE,

SRM University, AP, Amaravathi

# Acknowledgements

We would like to thank Dr. Ramakrishna Maharajan , for their support and guidance throughout this project, and for making us work efficiently on the research topic by providing a good path on the research topic Ethernet based remote temperature monitoring system**.** Without his help, we would not be able to proceed with the project for good results.

Table of Contents

[Certificate 1](#_Toc195719350)

[Acknowledgements 2](#_Toc195719351)

[Abstract 4](#_Toc195719352)

[Abbreviations 5](#_Toc195719353)

[List of Figures 6](#_Toc195719354)

[Introduction 1](#_Toc195719355)

[1. Methodology 5](#_Toc195719356)

[2. Discussion 13](#_Toc195719357)

[3. Concluding Remarks 14](#_Toc195719358)

[4. Future Work 15](#_Toc195719359)

[References 16](#_Toc195719360)

# Abstract

In modern automation systems, especially within industrial and home environments, real-time environmental monitoring is critical for ensuring system safety and efficiency. This project presents the development of an Ethernet-based remote temperature monitoring system using a PIC18F47Q10 microcontroller and ETH Click board for TCP/IP communication. The system collects temperature data via ADC, transmits it over Ethernet, and displays the results in real-time using a Python-based application. This approach ensures reliable, long-distance data communication, offering a scalable solution for applications such as industrial automation, smart homes, greenhouses, and server rooms.

# Abbreviations

| **Term** | **Description** |
| --- | --- |
| ADC | Analog to Digital Converter |
| TCP/IP | Transmission Control Protocol/Internet Protocol |
|  |  |
| GUI | Graphical User Interface |
|  |  |
| PIC | Peripheral Interface Controller |
| SPI | Serial Peripheral Interface |
| IoT | Internet of Things |
| ETH | Ethernet |
| GUI | Graphical User Interface |
| MCC | MPLAB Code Configurator |
| SCADA | Supervisory Control and Data Acquisition |

# List of Figures

Figure 1: The model of PIC18F and ETH board ……………………………………………2

Figure 2: console window setup……………………….…………………………………….4

Figure 3: temperature values in Hercules.………………………………….……………….13

# Introduction

The growing need for real-time temperature monitoring across various domains such as data centers, greenhouses, and industrial plants has led to the rise of Ethernet-enabled embedded systems. Traditional systems relying on wireless protocols or manual inspection face challenges in range, reliability, and scalability. To address this, the proposed project focuses on designing and implementing a reliable temperature monitoring system using Ethernet communication.

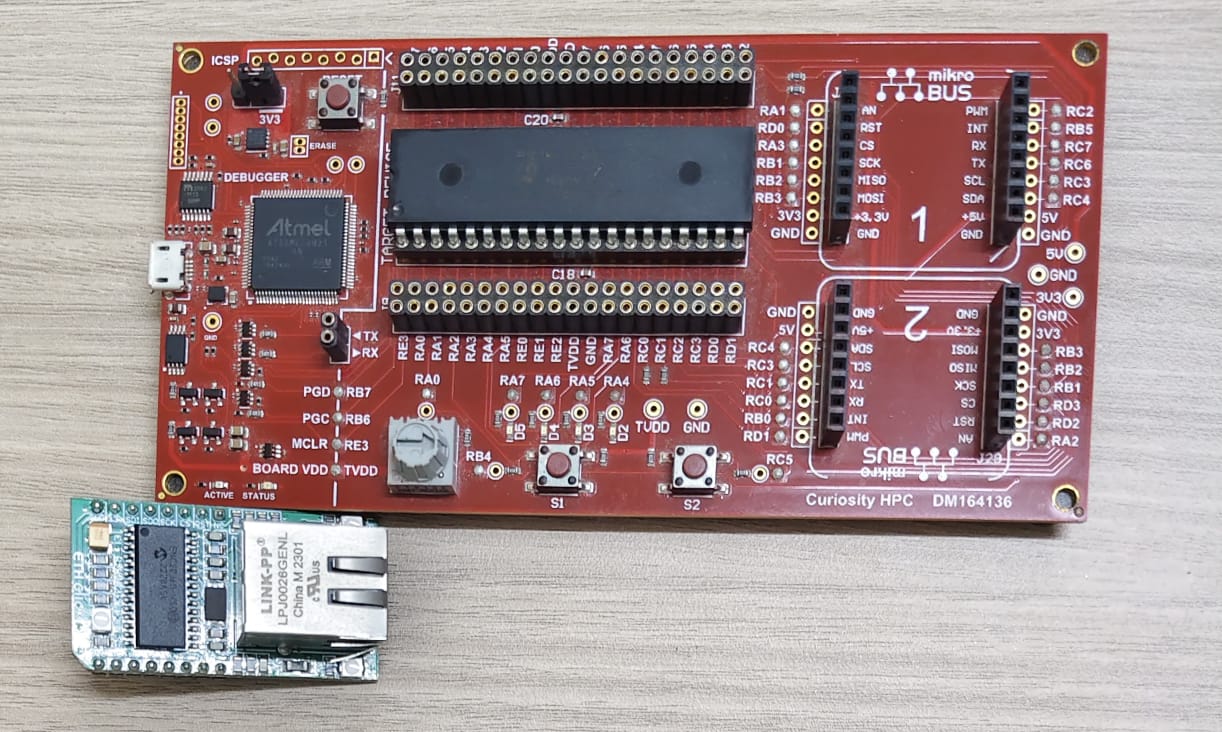
The system uses the PIC18F47Q10 microcontroller paired with an ETH Click board to achieve Ethernet connectivity. A temperature sensor connected to the microcontroller allows for accurate temperature measurement, and data is transmitted over a TCP/IP network to a remote system. The Python-based application on the PC end logs and displays real-time data.

*PAPER REVIEW*

Ethernet-Based Remote Temperature Monitoring system:

This project proposes a reliable and cost-effective method to remotely monitor and control temperature using Ethernet communication. It leverages a microcontroller (PIC18F47Q10) interfaced with an Ethernet module (ENC28J60 via ETH Click) to transmit real-time sensor data across a TCP/IP network. A Python-based application on a PC side acts as a TCP client to visualize the data.

Traditional temperature monitoring systems often rely on short-range wireless protocols like Wi-Fi or ZigBee, which suffer from limitations in reliability, scalability, and coverage area—especially in industrial or server room environments. This system, by contrast, makes use of Ethernet, which is known for stable and secure data transmission, low latency, and longer-range communication. The architecture and implementation presented in this work ensure accuracy and robustness, which are critical for applications like data center monitoring, greenhouses, and industrial automation.



SYSTEM MODEL

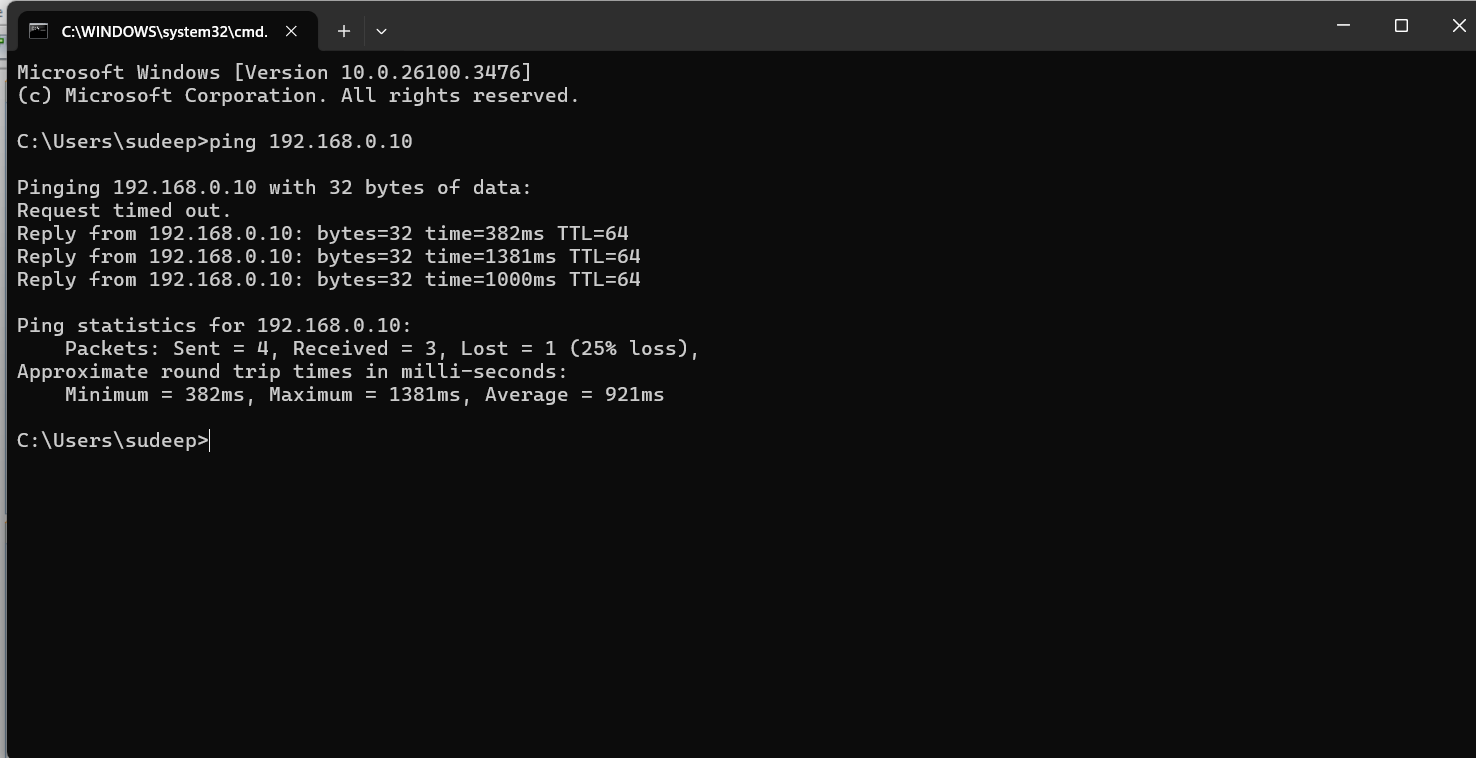
The system consists of the following key components:

* A **temperature sensing unit** (e.g., LM35 analog sensor),
* A **PIC18F47Q10 microcontroller** that reads the analog value via ADC,
* An **Ethernet controller (ENC28J60)** that transmits the processed temperature data to a remote terminal via TCP,
* A **PC client**, which acts as a server to receive and display the data in real-time.

The problem addressed is the need for long-range, stable temperature monitoring and control without relying on wireless communication, which may be unstable or insecure in certain environments.

The system uses the SPI protocol to establish communication between the PIC microcontroller and the Ethernet module. The transmitted data follows the TCP/IP stack protocol, ensuring packet reliability and error-checking mechanisms during data transmission.

The Python application running on the PC receives the temperature data string and presents it to the user through a console or GUI. This makes the system highly accessible, low-cost, and easy to deploy in real-world scenarios.



PROBLEM FORMULATION:

The problem addressed is the need for long-range, stable temperature monitoring and control without relying on wireless communication, which may be unstable or insecure in certain environments.

The system uses the SPI protocol to establish communication between the PIC microcontroller and the Ethernet module. The transmitted data follows the TCP/IP stack protocol, ensuring packet reliability and error-checking mechanisms during data transmission.

The Python application running on the PC receives the temperature data string and presents it to the user through a console or GUI. This makes the system highly accessible, low-cost, and easy to deploy in real-world scenarios.

**IMPROVEMENT USING ETHERNET COMMUNICATION**

In the design and implementation of the Ethernet-Based Remote Temperature Monitoring and Controlling System, the decision to adopt **Ethernet** as the primary mode of communication addresses several limitations found in conventional monitoring approaches.

Previous systems for remote temperature monitoring commonly relied on:

* **Manual inspection**, which is labor-intensive and prone to human error,
* **Wireless protocols** such as Wi-Fi or ZigBee, which suffer from:
  + Limited range,
  + High susceptibility to interference,
  + Unreliable performance in industrial environments.

These drawbacks are particularly significant in large facilities, server rooms, or industrial automation setups, where **reliable, real-time, and continuous monitoring** is critical.

# Methodology

#### **1. Hardware Implementation**

* **Microcontroller Unit**: PIC18F47Q10 configured using MPLAB X and MCC.
* **Temperature Sensor**: Analog sensor (e.g., LM35) connected to the microcontroller's ADC pin.
* **ETH Click Module**: ENC28J60 Ethernet module interfaced with the PIC via SPI protocol.

#### **2. Firmware Development**

* Initialization of ADC for continuous temperature data acquisition.
* SPI communication setup for ETH Click.
* Integration of TCP/IP stack to facilitate data transmission over Ethernet.

#### **3. Software and Communication**

* **Python Application**: Uses socket programming to receive temperature data.
* **Hercules Utility**: Used for debugging and acting as a TCP server/client during development.
* GUI or console output displays temperature values in real time.

#### **4. Data Flow**

1. Temperature sensor senses data.
2. Data digitized by ADC.
3. Microcontroller sends formatted data via TCP/IP.
4. Python application on PC receives and logs the data.

**ADC CODE:**

#include "mcc\_generated\_files/mcc.h"

#include "mcc\_generated\_files/TCPIPLibrary/tcpv4.h"

#include "mcc\_generated\_files/TCPIPLibrary/ipv4.h"

#include "mcc\_generated\_files/TCPIPLibrary/tcpip\_config.h"

#include <stdio.h>

#include <string.h>

// Global variables

static tcpTCB\_t port7TCB;

static uint8\_t rxdataPort7[32];

static uint8\_t txdataPort7[64];

void ADC\_Init(void)

{

TRISAbits.TRISA0 = 1; // RA0 as input

ANSELAbits.ANSELA0 = 1; // RA0 analog

ADPCH = 0x00; // AN0 channel

ADREFbits.ADPREF = 0; // Vref+ = VDD

ADREFbits.ADNREF = 0; // Vref- = VSS

ADCLK = 0x3F; // Fosc/64

ADCON0bits.ADCONT = 0; // Single conversion

ADCON0bits.ADON = 1; // Turn on ADC

}

uint16\_t ADC\_Read(void)

{

ADCON0bits.GO = 1;

while (ADCON0bits.GO);

return ((ADRESH << 4) | (ADRESL >> 4)); // 12-bit result

}

void TCP\_Temp\_Server(void)

{

uint16\_t adc\_val = 0;

float voltage = 0;

uint16\_t temp = 0;

socketState\_t socket\_state = TCP\_SocketPoll(&port7TCB);

switch (socket\_state)

{

case NOT\_A\_SOCKET:

TCP\_SocketInit(&port7TCB);

break;

case SOCKET\_CLOSED:

TCP\_Bind(&port7TCB, 7);

TCP\_InsertRxBuffer(&port7TCB, rxdataPort7, sizeof(rxdataPort7));

TCP\_Listen(&port7TCB);

break;

case SOCKET\_CONNECTED:

if (TCP\_SendDone(&port7TCB))

{

adc\_val = ADC\_Read();

voltage = adc\_val \* 0.8056; // assuming 0.8056mV/step (Vref = 3.3V / 4096 steps)

temp = voltage / 10; // simulate °C from voltage

sprintf((char\*)txdataPort7, "ADC: %u, Temp: %u C\r\n", adc\_val, temp);

TCP\_Send(&port7TCB, txdataPort7, strlen((char\*)txdataPort7));

}

break;

case SOCKET\_CLOSING:

TCP\_SocketRemove(&port7TCB);

break;

default:

break;

}

}

void main(void)

{

SYSTEM\_Initialize();

ADC\_Init();

INTERRUPT\_GlobalInterruptEnable();

INTERRUPT\_PeripheralInterruptEnable();

while (1)

{

Network\_Manage();

TCP\_Temp\_Server();

\_\_delay\_ms(1000); // send every 1 second

}

}

**TCP WITH ADC CODE :**

#include "mcc\_generated\_files/mcc.h"

#include "mcc\_generated\_files/TCPIPLibrary/tcpv4.h"

#include "mcc\_generated\_files/TCPIPLibrary/ipv4.h"

#include "mcc\_generated\_files/TCPIPLibrary/tcpip\_config.h"

#include <stdio.h>

#include <string.h>

// Global variables

static tcpTCB\_t port7TCB;

static uint8\_t rxdataPort7[32];

static uint8\_t txdataPort7[64];

void ADC\_Init(void)

{

TRISAbits.TRISA0 = 1; // RA0 as input

ANSELAbits.ANSELA0 = 1; // RA0 analog

ADPCH = 0x00; // AN0 channel

ADREFbits.ADPREF = 0; // Vref+ = VDD

ADREFbits.ADNREF = 0; // Vref- = VSS

ADCLK = 0x3F; // Fosc/64

ADCON0bits.ADCONT = 0; // Single conversion

ADCON0bits.ADON = 1; // Turn on ADC

}

uint16\_t ADC\_Read(void)

{

ADCON0bits.GO = 1;

while (ADCON0bits.GO);

return ((ADRESH << 4) | (ADRESL >> 4)); // 12-bit result

}

void TCP\_Temp\_Server(void)

{

uint16\_t adc\_val = 0;

float voltage = 0;

uint16\_t temp = 0;

socketState\_t socket\_state = TCP\_SocketPoll(&port7TCB);

switch (socket\_state)

{

case NOT\_A\_SOCKET:

TCP\_SocketInit(&port7TCB);

break;

case SOCKET\_CLOSED:

TCP\_Bind(&port7TCB, 7);

TCP\_InsertRxBuffer(&port7TCB, rxdataPort7, sizeof(rxdataPort7));

TCP\_Listen(&port7TCB);

break;

case SOCKET\_CONNECTED:

if (TCP\_SendDone(&port7TCB))

{

adc\_val = ADC\_Read();

voltage = adc\_val \* 0.8056; // assuming 0.8056mV/step (Vref = 3.3V / 4096 steps)

temp = voltage / 10; // simulate °C from voltage

sprintf((char\*)txdataPort7, "ADC: %u, Temp: %u C\r\n", adc\_val, temp);

TCP\_Send(&port7TCB, txdataPort7, strlen((char\*)txdataPort7));

}

break;

case SOCKET\_CLOSING:

TCP\_SocketRemove(&port7TCB);

break;

default:

break;

}

}

void main(void)

{

SYSTEM\_Initialize();

ADC\_Init();

INTERRUPT\_GlobalInterruptEnable();

INTERRUPT\_PeripheralInterruptEnable();

while (1)

{

Network\_Manage();

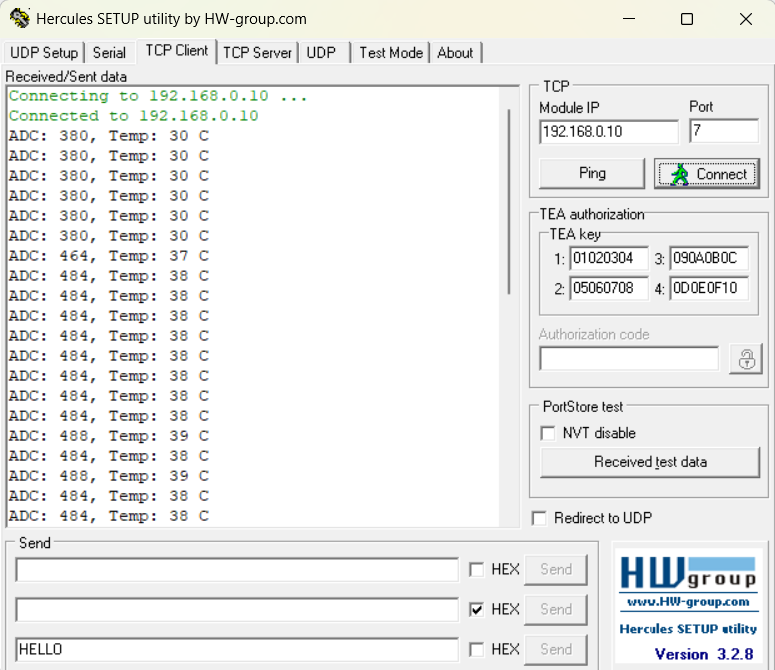
TCP\_Temp\_Server();

\_\_delay\_ms(1000); // send every 1 second

}

}

**Results:**



# Discussion

The implementation successfully demonstrates the practical application of Ethernet in real-time embedded temperature monitoring. Unlike Wi-Fi-based systems, Ethernet ensures lower latency and higher reliability, especially over long distances or in interference-prone industrial environments. The PIC18F47Q10 microcontroller, with its integrated ADC and support for SPI communication, offers a cost-effective solution for this application.

Using Python for the PC-side application allowed for quick development and testing, with the added benefit of potential GUI enhancements and database integration in the future. The system is flexible, allowing for extensions such as cloud connectivity, alert mechanisms, or temperature-based automated control (e.g., fan or AC activation).

# 

# Concluding Remarks

The Ethernet-based remote temperature monitoring system presents a reliable, scalable, and cost-effective solution for industrial and smart environment applications. By leveraging the robustness of Ethernet and the simplicity of embedded hardware, the system meets the demand for accurate and long-distance environmental monitoring. Future developments may include cloud-based data storage, integration with mobile platforms, and multi-sensor support for a comprehensive smart sensing network.

1. **Future Work**

Future enhancements could include integrating the system with cloud services for remote access and storage. A mobile application can be developed for on-the-go monitoring. The system could be extended to support multiple sensors for broader coverage. Additionally, data logging and alert-based automation can be implemented. Security can also be improved by adding encryption to data transmission. Further, the firmware can be optimized for power efficiency and extended with support for OTA (Over-The-Air) updates. Integration with environmental sensing (e.g., humidity, CO2) and edge AI for predictive analysis can also be considered.

# References

1. [1] Zhao, X., & Wang, J. (2020). Design and Implementation of a Remote Temperature Monitoring System Based on IoT and Ethernet. International Journal of Computer Applications, 175(7), 45-50.
2. Chien, C. F., & Lin, C. L. (2018). Development of a Smart Temperature Control System for Industrial Applications Using Ethernet Communication. Journal of Automation and Control Engineering, 6(4), 157-162.
3. Kumar, R., & Gupta, P\*. (2019). Energy-efficient Temperature Control Systems in Data Centers: A Review. Energy and Buildings, 192, 25-39.
4. Seth, A., & Kaur, P. (2021). IoT-based Remote Monitoring and Control Systems: A Case Study on Temperature Control. IEEE Access, 9, 12345-12353.