

**DESIGN AND IMPLEMENTATION OF ENERGY USAGE
MONITORING AND CONTROL SYSTEMS USING
MODULAR IIOT FRAMEWORK**

by

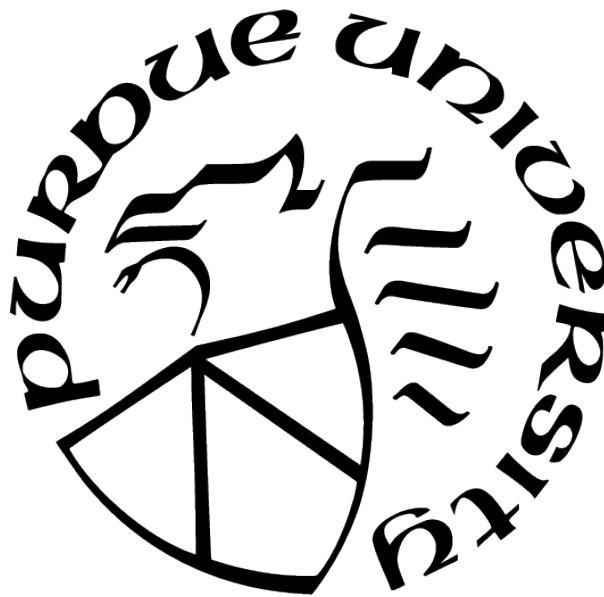
Monil Vallabhbhai Chheta

A Thesis

Submitted to the Faculty of Purdue University

In Partial Fulfillment of the Requirements for the degree of

Master of Science



Department of Electrical and Computer Engineering

Indianapolis, Indiana

May 2021

**THE PURDUE UNIVERSITY GRADUATE SCHOOL
STATEMENT OF COMMITTEE APPROVAL**

Dr. Stanley Yung-Ping Chien

Department of Electrical and Computer Engineering

Dr. Jie Chen

Department of Mechanical Engineering

Dr. Lingxi Li

Department of Electrical and Computer Engineering

Approved by:

Dr. Brian King, Chair

Dedicated To
My Parents: Heena and Vallabhbhai Chheta
Sister: Kruti Chheta
and
Dr. Paridhi Gupta
For their Love and constant support

ACKNOWLEDGMENTS

I would like to express my sincere gratitude to my advisor Dr. Stanley Yung-Ping Chien for his patience, motivation, and constant guidance throughout my masters education and research. Besides my advisor, I would like to thank Dr. Jie Chien and Dr. Ali Razban for constantly driving me and for valuable advice and stimulating sleepless discussion for the best outcome of the given opportunity. My sincere thanks to the Industrial Assessment Center and Department of Electrical and Computer Engineering, who provided access to the laboratory and research facilities. Special thanks to Sherrie and Dr. Brian King for their constant support and motivation throughout my time at IUPUI. I would like to thank and acknowledge, US Department of Energy for believing and supporting the IUPUI Center for continued research. Last but not the least, I would like to thank my sister Parita Radadiya, brother-in-law Dr. Pragneshkumar Radadiya, my friends Vidish Poojari, Saumya Upadhyay, Tanmay Zantye, Dewant Katore, Durvesh Pathak and Akash Gaikwad for supporting me throughout my journey.

TABLE OF CONTENTS

LIST OF TABLES	9
LIST OF FIGURES	10
ABBREVIATIONS	14
ABSTRACT	15
1 INTRODUCTION	16
1.1 Challenges	16
1.2 Problem Statement	18
1.3 Expected Contribution	20
1.4 Objectives	21
1.5 Document Structure	23
2 BACKGROUND AND LITERATURE SURVEY	24
2.1 Background	24
2.2 What are IOT and IIoT?	28
2.3 Sensors	29
2.4 Database	30
2.5 Web	31
2.6 Summary	32
3 SYSTEM ARCHITECTURE	33
3.1 System Components	33
3.1.1 Factory	34
3.1.1.1 Equipment	34
3.1.1.2 Sensors and Actuators	34
3.1.1.3 Sensor Box	34
3.1.1.3.1 Microcontroller	35
3.1.1.3.2 Wi-Fi Module	36
3.1.1.3.3 Power Supply	36
3.1.1.3.4 Sensor Interfacing Connectors	36
3.1.1.3.5 Relay Interface Connectors	36

3.1.2	Cloud	37
3.1.2.1.	Database	37
3.1.2.2	Webserver	38
3.1.3	User	38
3.2	Operational Capabilities of the System	38
3.2.1	Sensor Configuration	40
3.2.2	Data Acquisition	40
3.2.3	Sensor Failure Detection	41
3.2.4	Data Processing	43
3.2.5	Wi-Fi Connection	44
3.2.6	Internet Failsafe and Offline Logging	45
3.2.7	Store Data in Cloud	46
3.2.8	Retrieve Data to Matlab	47
3.2.9	Store Legacy Data	48
3.2.10	Privilege Based Login	49
3.2.11	View Real-Time and Legacy Data	50
3.2.12	Control Connected Equipment Remotely	51
4	SMART SENSOR DESIGN	53
4.1	Hardware	53
4.1.1	Sensor Interface Circuits	54
4.1.1.1	Signal Conversion Circuit	55
4.1.1.1-a	0-2.5V Conversion Circuit	55
4.1.1.1-b	0-5V Conversion Circuit	56
4.1.1.1-c	0-10V Conversion Circuit	57
4.1.1.1-d	0-20mA Conversion Circuit	57
4.1.1.2	Sensor Connector	58
4.1.1.3	Interface Selection Circuit	59
4.1.1.4	Sensor System	60
4.1.2	Actuator Circuit	61
4.1.3	Power Supply Circuit	62

4.1.3.1	Power Supply	63
4.1.4	Microcontroller Circuit	63
4.1.5	SD Card	64
4.1.6	Wi-Fi	65
4.1.7	PCB	66
4.2	Embedded Software	68
4.2.1	Configuration	69
4.2.2	Connection	71
4.2.3	Read Raw Data from Sensors in Real-time and Convert to Standard Units	72
4.2.4	Send Data to Webserver	73
4.2.5	Actuator	76
4.2.6	Backup to SD Card	78
5	DATABASE	84
5.1	Database Design	84
5.1.1	Information Tables	85
5.1.2	Operating Table	89
5.1.3	Application Function Tables	91
5.2	Database Usage	93
5.3	Implementation	96
6	WEB SERVER	98
6.1	Establishing and Closing Connection from the Microcontroller to Cloud . . .	100
6.1.1	Establishing a Connection from Microcontroller	101
6.1.2	Closing Connection from the Microcontroller	102
6.2	PHP	103
6.3	Sending Board Startup Specifications from Microcontroller to Cloud	103
6.3.1	Sending Board Startup Specifications From Microcontroller to Web Server	104
6.3.2	Storing Received Data From Webserver to Database	106
6.4	Sending Sensor Values from Microcontroller to Cloud	109

6.4.1	Send Sensor Values from Microcontroller to Web Server	110
6.4.2	Storing Received Data From Webserver to Database	112
6.5	Web Pages for Users	114
7	TESTING	117
7.1	Monitoring Test	117
7.2	Monitoring and Control Test	119
7.3	Demonstration of SD Data Storage during Network Failure and the Data Recovery after the Restoration of the Network Connection	123
8	CONCLUSION	129
9	FUTURE WORK	130
	REFERENCES	131
A	APPENDIX	132
A.1	Design and Implementation Requirements	132
A.2	Computation and Communication Components	134
A.3	Sensor List	134
A.4	Power Supply	135
A.5	Sensor-Board Interface Circuit	136
A.6	Sensor Connection	136
A.7	Actuator Circuit	137
A.8	Logging to Database	138
A.9	Computation and Communication Board	138
A.10	Data Storage	139
B	APPENDIX	141
B.1	Component List	141
C	APPENDIX	144
C.1	Accessing Database from Matlab Command Line	144
	Sample of the Entire Code	147
D	APPENDIX	148
D.1	Using MySQL Workbench	148

LIST OF TABLES

2.1	Comparison Table of Current Services and IAC Energy Box	26
3.1	Capabilities of the System	35
4.1	Sensor List	59
5.1	Sample Questions along with the Queries	93
A.1	System Requirements	132
A.2	Sensor List	134
A.3	Sensor Output List	135
B.1	Component List	141

LIST OF FIGURES

1.1	Power Outline of Energy Usage in United States in 2017	17
1.2	Overall Structure of the System	22
2.1	Energy Consumption in 2018 by Sectors	25
2.2	IoT Enterprise Functions	30
3.1	Operational Context Diagram	33
3.2	Components in the Cloud	37
3.3	Operational Capabilities	39
3.4	Configure Sensor Scenario	40
3.5	Data Acquisition Scenario	41
3.6	Sensor Failure Detection	42
3.7	Data Processing Scenario	44
3.8	Wi-Fi Capable Scenario	45
3.9	Internet Fail-safe Scenario	46
3.10	Send and Store Data in Cloud Scenario	47
3.11	Retrieve Data to Matlab Scenario	48
3.12	Store Legacy Data Scenario	49
3.13	Level Based Login Scenario	50
3.14	View Real-Time and Legacy Data Scenario	51
3.15	Control Remote Equipment Scenario	52
4.1	Hardware Diagram	54
4.2	Interface Board Block Diagram	55
4.3	0-5V Conversion Circuit	56
4.4	0-10V Conversion Circuit	57
4.5	0-20mA Conversion Circuit	58
4.6	Sensor Connector Pin Layout	58
4.7	Conversion Selection Block Diagram	60
4.8	Sensory System	61
4.9	Actuator Circuit	62

4.10	Power-Microcontroller Interface Circuit	63
4.11	Microcontroller Circuit	64
4.12	SD Card Connection	65
4.13	Wireless Connection	65
4.14	PCB Layout	67
4.15	Software Block Diagram	68
4.16	Cloud Connection and Data Upload Sequence Diagram	72
4.17	Client-Server Messaging	75
4.18	Microcontroller Code to Send Data to the Web Server	76
4.19	Actuator Request Handling	77
4.20	Server Response with Relay Status	78
4.21	Backup to SD Card Activity Diagram	79
4.22	Backup to SD Card	80
4.23	Send Backup Data to Server	81
4.24	Sendddta Function	82
4.25	Screen Shot of how Backup Data is Saved	82
4.26	Delete the Data Sent to Server	83
5.1	Information Table E-R Diagram	87
5.2	Database Table Containing Information Regarding Companies	87
5.3	Database Table Containing Contact Information	88
5.4	Database Table for Appliance	88
5.5	Database Table for Information on Appliance Port	88
5.6	Database Table Containing Information Regarding Board	88
5.7	Database Table Containing Information about Company Appliance	89
5.8	Database Table Containing Information about Sensors	89
5.9	Database Table Containing Information Regarding Zone	89
5.10	Operating Table E-R Diagram	90
5.11	Board Startup Specifications Table in the Database	90
5.12	Database Sensor Readings Table	91
5.13	Application Function Table E-R Diagram	92

5.14	Air Compressor Forecast Database Table	92
6.1	Edge-Server Communication	99
6.2	Front Rnd Web Pages Sequence Diagram	100
6.3	The getreply Function	101
6.4	TCP_Establish_Connection Function	102
6.5	Close TCP/IP Connection	102
6.6	Board Specification Communication	104
6.7	Board Specification Code Snippet	105
6.8	PHP Program Flow Chart for Board Startup Specifications	107
6.9	PHP Page Receiving Information from the Microcontroller	108
6.10	PHP Code Establishing a Connection to the Database	108
6.11	PHP Code Storing Values to the Database	109
6.12	Sensor Readings Communications	110
6.13	PHP Program Flow Chart for Sensor Values	111
6.14	Microcontroller Preparing Data Frame to Send to Webserver	112
6.15	Web Server PHP Program Receives Data from the Microcontroller	112
6.16	PHP Program Connecting to Database	113
6.17	PHP Page Storing Data to Database	113
6.18	PHP Program Acknowledging Microcontroller Request and Replying with Relay Status	114
6.19	Sensor Readings Displayed Real-time on the Web	115
6.20	Webpage to Control Relays	116
7.1	CTV-C Current Sensor	118
7.2	Current Usage of Supply and Return Fan Between February 14, 2019 12:00 AM – February 15, 2019 12:00 AM	118
7.3	Current Usage of Supply and Return Fan Between February 13, 2019 – Febru- ary 18, 2019	119
7.4	Demo Station	121
7.5	Front-end Application Demand Forecast	123
7.6	Motor Current Usage Based on User Control	124
7.7	Data from the System with Network Failure	125

7.8	Equipment Turned On	126
7.9	Backup Mode	126
7.10	Internet Turned On	127
7.11	Backup Data Sent	127
A.1	Power Outline	135
A.2	Sensor-Board Interface Outline	136
A.3	Sensor Connection Outline	137
A.4	Actuator Circuit Outline	137
A.5	Board Outline	139
A.6	Log Outline	140

ABBREVIATIONS

SEU	Significant Energy Usage
IoT	Internet of Things
IIOT	Industrial Internet of Things
IAC	Industrial Assessment Center
AHU	Air Handling Unit
Php	Hypertext Preprocessor
HTML	Hypertext Markup Language
CSS	Cascading Style Sheets
SQL	Structured Query Language
TCP/IP	Transmission Control and Internet Protocol

ABSTRACT

This project aims to develop a cloud-based platform that integrates sensors with business intelligence for real-time energy management at the plant level. It provides facility managers, an energy management platform that allows them to monitor equipment and plant-level energy consumption remotely, receive a warning, identify energy loss due to malfunction, present options with quantifiable effects for decision-making, and take actions, and assess the outcomes. The objectives consist of:

1. Developing a generic platform for the monitoring energy consumption of industrial equipment using sensors
2. Control the connected equipment using an actuator
3. Integrating hardware, cloud, and application algorithms into the platform
4. Validating the system using an Energy Consumption Forecast scenario

A Demo station was created for testing the system. The demo station consists of equipment such as air compressor, motor and light bulb. The current usage of these equipment is measured using current sensors. Apart from current sensors, temperature sensor, pressure sensor and CO2 sensor were also used. Current consumption of these equipment was measured over a couple of days. The control system was tested randomly by turning on equipment at random times. Turning on the equipment resulted in current consumption which ensured that the system is running. Thus, the system worked as expected and user could monitor and control the connected equipment remotely.

1. INTRODUCTION

Internet of things is gaining more and more popularity each day. Numerous companies are providing IoT solutions. As a part of this trend, the Industrial Assessment Center at IUPUI (Indiana University Purdue University Indianapolis) has taken up the challenge of coming up with a Smart Manufacturing platform. This platform integrates the existing manufacturing systems with an IoT device developed in-house. The Smart Manufacturing platform is a low-cost tool, requires low power to operate, and can be readily deployed. The edge IoT device integrates the system with existing sensors at minimal effort. The platform is created to monitor the parameters such as Temperature, Pressure, Current, etc., from these sensors in Real-Time. This platform also has actuation capabilities, which can turn on/off machinery manually or by an automated program. The data from these sensors can be used to analyze energy usage and predict trends. Based on the predictions, automated programs like peak shaving algorithms can be used. The platform has been made as generic as possible. Even though the platform is generic, it is highly customizable to tailor companies' requirements.

1.1 Challenges

Lawrence Livermore National Laboratory[3], a federal research facility funded by DOE (Department of Energy) and University of California, Berkley, prepares a Sankey diagram that shows the generated energy from different sources, utilized by various sectors and the energy wasted. Figure 1.1 is the Sankey diagram for the year 2017.

Figure 1.1 uses quads as unit of measurement. Each quad is equal to a quadrillion BTUs (British Thermal Unit). The units in the graph are equivalent as follows:

1. Gasoline - 8,007,000,000 gallons (US)
2. Electricity - 293,071,000,000 kilowatt-hours (kWh)
3. 36,000,000 tons of coal
4. Natural Gas - 970,434,000,000 cubic feet
5. Oil - 25,200,000 tons

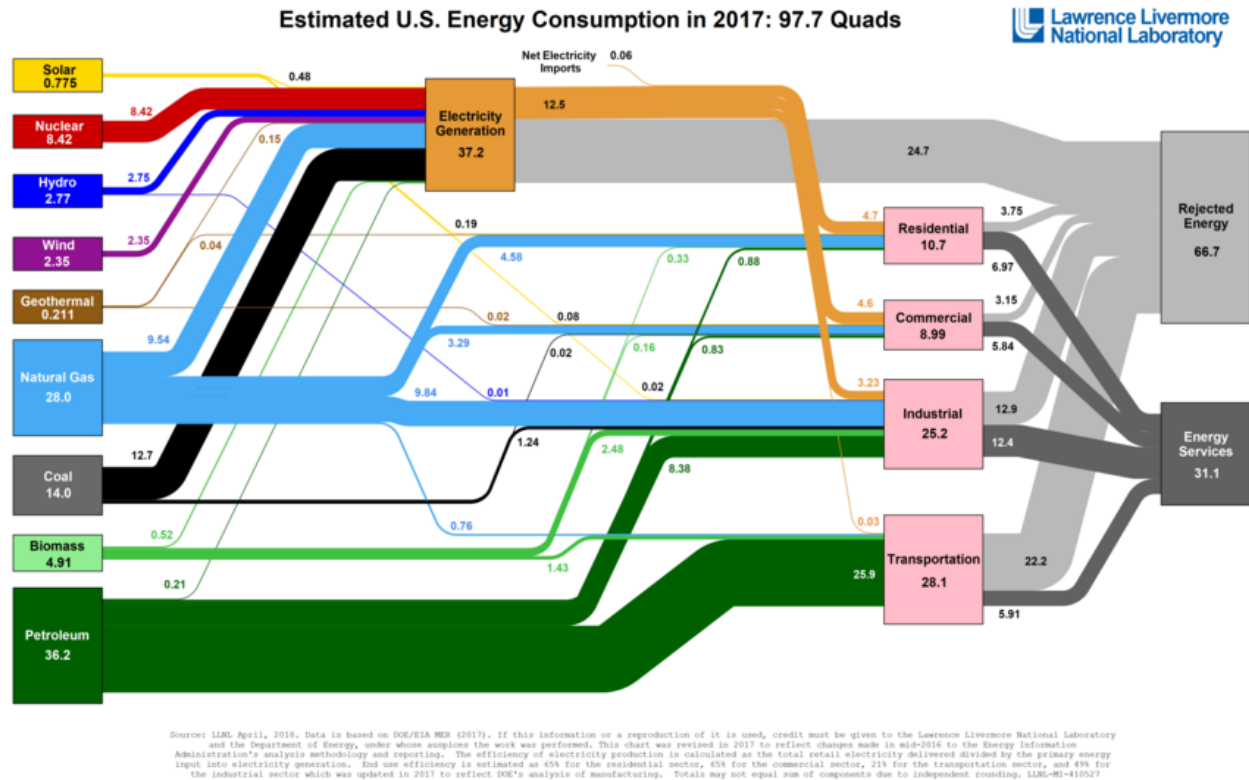


Figure 1.1. Power Outline of Energy Usage in United States in 2017

6. TNT - 252,000,000 tons

7. Uranium-235 - 13.3 tons

The total amount of energy used by the US was 97.7 quadrillion BTUs.

As we can see from Figure 1.1, two-thirds of the energy was wasted. This number is quite high. More than half is rejected energy. The challenge is to save energy at every level possible in order to make more efficient use of the resources.

Energy is an essential asset for the economic development of any country. Household requires energy for heating, cooling, lighting, and appliances. Transportation requires energy in fuel for cars, trucks, planes, trains, and ships. The industrial sector requires energy for factories, industry complexes, Research and Development divisions, trades, etc. Health education and welfare activities require energy for administration, water supply, sewage and trash disposal, military forces, schools, and hospitals. IAC focuses on helping the industrial sector save energy by auditing the manufacturing plants, analyzing the inefficient usage

of energy, and recommending improvements to save energy and, hence, lower energy bills. According to an article by Constellation[1], An Exelon Company, the following are six ways Manufacturers can reduce industrial costs

1. Develop an Energy Management Team
2. Conduct an Energy Audit
3. Strategically Schedule Machinery Use
4. Schedule Shut-Downs and Start-Ups
5. Optimize Air Compressors
6. Conduct an HVAC Audit

To fulfil the challenge of minimizing energy waste and saving energy, the steps mentioned above can be useful and fulfilled with a system having IoT services. To fulfil method 3, 4 and 5, data is required to analyze the machinery's usage pattern. To collect data, audits need to be performed. Using this system allows the auditor to install the sensors at the plant with ease, collect real-time data, and control the machinery.

This system's end goal is to use IoT as a tool to manage the energy consumption of a plant using algorithms like peak shaving, which would require a lot of legacy data.

Cloud computing, along with IoT is a contribution towards industry 4.0. It is slowly adding values to industries and becoming an essential part of industrial processes. More and more industries are adopting IoT. This has caused aggressive development efforts by the enterprise community.

1.2 Problem Statement

At IUPUI, Industrial Assessment Center [6] conducts energy usage audits at industries and manufacturing plants. The data collected from the audits are analyzed, and energy-saving solutions are proposed.

Audits have the following steps involved:

1. Client interaction – At this stage, the client interacts with the auditing organization. The organization acquires necessary details such as types of equipment used, power usage pattern and production requirements.
2. The auditing organization requests utility bills such as gas, electricity, etc. A minimum of one-year of utility bills is required.
3. The organization then does a background study of the plant. It determines specifications of the equipment used by the plant. They decide on what equipment and sensors they need to carry to the plant to measure the useful data.
4. The auditor and team visits the plant. They make preliminary observations and try to identify potential energy-saving opportunities.
5. Sensors are attached to the equipment where there is a potential to save energy. At this stage, if the audit level is ASHRAE level 3, the auditor requires a minimum of two months of heating data and two months of cooling data to study the usage pattern.
6. After acquiring data, the auditor analyzes the pattern.
7. Cost calculations are made for the identified energy-saving opportunities.
8. A report is handed out to the client organization, which includes recommendations for potential energy savings.

In this process, step 5, data collection, is critical. Currently, the method used at IAC is based on leaving the sensors at the plant. The data would be collected locally and not available until collection devices are connected to the computer and transferred. It will be challenging to determine if the data collection process is running correctly unless frequent visits to the plant are made to ensure this. If the logger is spoilt due to human or system error, data for the season would be lost, and the auditor will have to wait until next season to collect data, hence costing a lot of time. The system mentioned in this thesis will allow the auditing organization to ensure that the data collection process is running correctly. It is available in real-time remotely.

1.3 Expected Contribution

The project in this thesis is referred to as a smart manufacturing platform. The idea to save energy using an algorithm requires a platform to deploy to prove it is working. To build this platform, designing appropriate hardware that can work in the industry is required. Considering the information and sensors provided by IAC, the system was designed with capabilities like generic interface for different kinds of sensors, internet capabilities, data logging, equipment control, certain fail-safe features, etc. The system is generic yet highly configurable.

The following are the requirements:

1. The system shall have Hardware for data collections – sensors installed on SEU and data acquisition.
2. The system shall provide Data storage for collected sensor data.
3. The system shall provide Database services for storing and sharing converted data.
4. The system shall have a Data analytic module for extracting useful information from the data, which includes predicting energy demand peaks, recommendations with predicted effects using the current and legacy information, such as effects of varying pressure setting on the production, re-scheduling, etc.
5. The system shall provide a Knowledge base for storing the information as the legacy information – an energy-saving decision needs to be made based on multiple factors, including weather, production needs, shift, occupants, etc. The historical records of these factors are needed for model validation, accurate prediction, and assessment.
6. The system shall have a User Interface with a dashboard tailored for the operators, which will enable the operators to monitor plant performance, get warnings on malfunctions, present options with predicted benefits, and send commands to the controller.
7. The system shall have Controller, which will implement actions on the SEUs either automatically or manually from the mobile device.

8. The system shall have algorithms for fail-safe ensuring reliability.

1.4 Objectives

This project aims to develop a cloud-based platform[11] that integrates sensors with business intelligence for real-time energy management at the plant level. The platform provides facility managers an energy management tool to monitor equipment and plant level energy consumption remotely, receive a warning, identify energy loss due to malfunction, present them options with quantifiable effects for decision-making, take actions, and assess the outcomes. Specifically, the objectives of this project consist of:

1. Developing a generic platform for the system
2. Integrating sensing communication and control hardware, data models, and control algorithms into the platform
3. The ability to monitor the energy usage and demonstrate through web-based GUI
4. The ability to store the real-time data in a database
5. The ability to retrieve the data from the database
6. The ability to write to the database
7. The ability to control the machine remotely
8. The ability to deploy energy-efficient algorithms via Matlab and Web Applications
9. Validating the system using various energy-saving scenario

A previous group had worked on the project. Their work formed the starting point for this project. Several components were already selected, which consists of several subsystems modules comprising of: power supply, sensor output interface, actuator circuit, microcontroller, WiFi module, SD card, logging, sleep, filtering, and self-diagnostics. The description of each of the subsystem can be found in Appendix A. The result should be proven to satisfy the overall project requirements as previously specified. Specifically, the goal includes the

requirement of sensor selections, data acquisition interface to all sensors satisfy the sensor selection requirements, models of significant energy use (SEU) equipment, integrated energy consumption model at the plant level, data warehouse, data analytics module, knowledge base module, mobile communication, and controller for energy management actions. The data storage and computations will be handled in the Cloud, with encrypted data for cyber-security. The overall structure of the system is described in Figure 1.2.

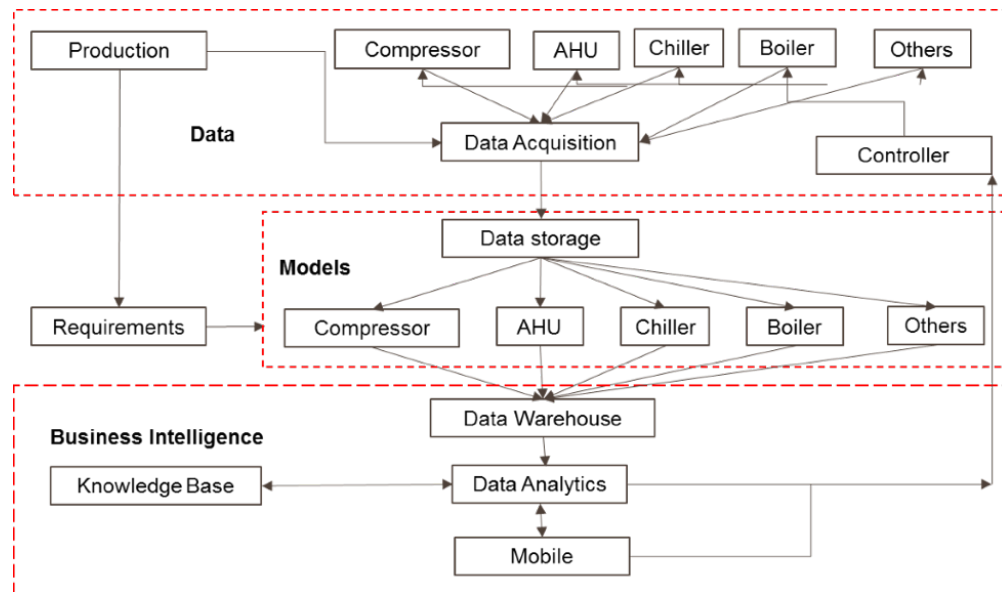


Figure 1.2. Overall Structure of the System

1.5 Document Structure

The rest of the thesis is organized as follows.

Chapter 2 consists of the literature survey regarding the topic that is concerned. Description regarding the components and technology used for overcoming the issue has been mentioned.

Chapter 3 describes the system architecture. This chapter shows the top-level view of the system with the components collaboration and the underlying technology used.

Chapter 4 provides the design and implementation of the Hardware System. A Hardware system consists of various components such as a microcontroller, Wi-Fi, sensor connection, Actuator. To run the system, microcontroller firmware is required to operate. The structure of the firmware is explained in this section.

Chapter 5 demonstrates the database design and the use of the database. The data acquired from sensors mentioned in Chapter 4 needs to be stored. Since there is a limit on data stored locally on the memory card, database service has been used. This way, the data is readily available to the user and can be used for further processing.

Chapter 6 describes data processing on the cloud end when the hardware sends it to the cloud. This section also provides information on how users can interact with the system. Web services are also required for front end users who can visualize, manipulate, and make decisions. This service also makes sure the system data and control are readily available from anywhere. Internet and browser are sufficient to access the system.

Chapter 7 consists of the implementation and testing of the system. The Demo board was built, and all software was developed. All the sub-systems mentioned in the earlier chapter were integrated, on which testing and validation are performed.

Chapter 8 is the discussion and conclusion from the efforts made to develop the system.

Chapter 9 suggests improvements and future work that can be done to make the system perform better.

2. BACKGROUND AND LITERATURE SURVEY

The leading IoT platform providers are Amazon AWS, Microsoft Azure, Mathworks Thingspeak[8], etc. These platforms provide a robust service and easy installation. However, the devices using these platforms need to be compatible with the service providers. They have a list of devices they support and also provide development packages for the same. In this project, the hardware and the software are developed by IAC. It allows interfacing almost all the industrial sensors found in the manufacturing plant, a part of design consideration and compatibility. Provisions have been made to make minimal changes to the manufacturing plant hardware. The platform also collects legacy data and predicts pattern that is later used to make a smart decision, for example, usage of lighting, the power consumption of heater, air compressor, and other energy consumption devices. Currently, an algorithm is under the testing phase to predict peak energy demands and make smart decisions to shave the peak. This kind of dedicated environment is currently not provided by the services mentioned above.

2.1 Background

Energy conservation is an important concern in today's world. Industry uses several kinds of energy sources like Electricity, Petroleum products, Natural Gas, etc. The naturally occurring resources like Petroleum products, natural gas, coal, etc., are converted to electric energy so that it can be used to drive the equipment. According to the Center for sustainable systems, the University of Michigan Energy Fact-sheet[10], the figure shown below, in 2018, the highest amount of energy utilized was by the industrial sector. If the given trend continues, fossil fuels, which are primary contributors to energy generation, will be exhausted in the next 20 years.

The energy monitoring system would play a vital role in the conservation of energy. This thesis focuses on building such a platform, which can monitor energy consumption, real-Time, using different kinds of sensors. The system is also compatible with controlling Significant Energy consumption units.

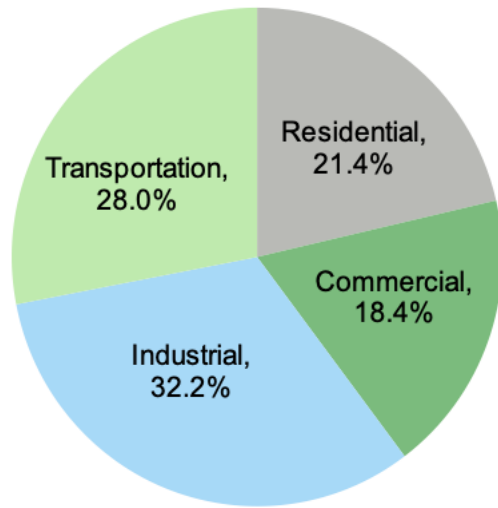


Figure 2.1. Energy Consumption in 2018 by Sectors

Table 2.1 provides a comparison of various commercial services[7]. At the end of the table, it can be seen that IAC Energy Box provides all-round support compared to the current services.

Table 2.1. Comparison Table of Current Services and IAC Energy Box

IoT Software Platform	Device management?	Integration	Protocols for data collection	Types of analytics	Visualizations?
IAC Energy Box	Yes	REST API	HTTP, TCP/IP, HTTPS	Real-Time Monitoring and prediction	Yes
2lemetry (AWS)	Yes	Salesforce, Heroku, ThingWorx APIs	MQTT, CoAP	Real-time analytics	No
Appcelerator	No	REST API	MQTT, HTTP	Real-time analytics	Yes
AWS IoT platform	Yes	REST API	MQTT, HTTP1.1	Real-time analytics	Yes
Bosch IoT Suite	Yes	REST API	MQTT, CoAP, AMQP, STOMP	*Unknown	Yes
Ericsson	Yes	REST API	CoAP	*Unknown	No

Table 2.1 continued from previous page

EVERYTHING	No	REST API	MQTT, CoAP, Web-Sockets	Real-time analytics (Rules Engine)	Yes
IBM IoT	Yes	REST and Real-time APIs	MQTT, HTTPS	Real-time analytics	Yes
ParStream	No	R, UDX API	MQTT	Real-time analytics	Yes
PLAT.ONE	Yes	REST API	MQTT, SNMP	*Unknown	Yes
ThingWorx	Yes	REST API	MQTT, AMQP, XMPP, CoAP, DDS, WebSockets	Predictive analytics, Real-time analytics	Yes
Xively	No	REST API	HTTP, HTTPS, Sockets/ Websocket, MQTT	*Unknown	Yes

* The cells marked with Unknown indicates that the relevant information could not be found from the available documentation.

2.2 What are IOT and IIoT?

According to Oracle[13], The Internet of Things (IoT) describes the network of physical objects—“things”—that are embedded with sensors, software, and other technologies to connect and exchange data with other devices and systems over the internet. These devices range from ordinary household objects to sophisticated industrial tools. With more than 7 billion connected IoT devices today, experts expect this number to grow to 10 billion by 2020 and 22 billion by 2025. Oracle defines Industrial IoT as, application of IoT technology in industrial settings, especially with respect to instrumentation and control of sensors and devices that engage cloud technologies. Recently, industries have used machine-to-machine communication (M2M) to achieve wireless automation and control. However, with the emergence of cloud and allied technologies (such as analytics and machine learning), industries can achieve a new automation layer and create new revenue and business models. IIoT is sometimes called the fourth wave of the industrial revolution, or Industry 4.0.

The following are some common uses for IIoT:

- Smart manufacturing
- Preventive and predictive maintenance
- Smart power grids
- Smart cities
- Connected and smart logistics
- Smart digital supply chains

Having technology like cloud computing, big data, low-cost computing, analytics, and mobile technology, enables physical objects (or things) to collect and transmit data without human interaction. With such a network of connected systems, it is possible to monitor, capture, and take action between things. Internet is widely available in today’s era, and the cost of connection is decreasing drastically. Almost every device we use today has internet connectivity, from our cell phones, laptops to TVs and refrigerators. Lately, modern sensors

are being developed which have Wi-Fi capabilities. Since the technology cost is going down, the use of such Wi-Fi enabled devices is sky-rocketing.

2.3 Sensors

Traditional sensors sense physical conditions or change in physical conditions such as temperature, pressure, occupancy, etc. and sends the signal to other electronic devices such as microcontrollers. The microcontroller then performs suitable actions if programmed to do so. On the other hand, smart sensors are Wi-Fi enabled. They are connected to the network and sends data wirelessly to the cloud or other computing devices. This enables the data to be available to humans. In this project, traditional sensors are connected to the microcontroller, and the microcontroller is Wi-Fi enabled. Therefore, the sensors' measurements are sent to the microcontroller, and the microcontroller sends the data to the cloud. Any suitable action required by the cloud program will be sent to the microcontroller, and it will actuate the requested device. The use of the traditional sensor is to make the system low cost. Industries already have established systems with sensors installed. The energy box needs to be installed and interfaced with existing sensors to make it smart, making installation an easy and low-cost process. Wi-Fi enabled sensors have been around before 2010. However, the term IoT was coined later. Nowadays, manufacturing companies are trying to enable Wi-Fi capabilities in anything possible. Hence, the IIoT sensors have come into existence. IIoT sensors are beneficial to the management for monitoring the production line, detecting faults, and scheduling maintenance. Researchers use this data to analyze and improve the efficiency of the production machinery. Sensors can provide many data like Temperature, Humidity, Current, Pressure, Power consumption, CO2 levels, occupancy, etc. Earlier in the days, the sensors were not wifi enabled. The standard industry sensors are used in this thesis project and patched to connect to a microcontroller, which is wifi enabled so the data can be sent to the cloud. This mechanism allows installing an IIoT device with minimum investment and replacement of sensors. Figure 2.1 below shows an enterprise View of the IoT. The essential function of sensors is to provide measurements and provide the enterprise data using wired/wireless protocol. The control commands are sent from the enterprise

to the controller using the same protocol. These sensors and controllers can be integrated into various domains like industrial, office, medical, etc. Figure 2.2, from ZDNet[4], defines various business functions that IoT can cater to. Several protocols can help build a network of systems such as Wi-Fi, Bluetooth, Zigbee, etc. Using these protocols, sensing of sensors and actuation of controllers such as light sensor, HVAC control, humidity sensor, etc., can be done wirelessly. There are various domains like industry, medical center, vehicle, etc., where IoT can be applied and used. In this project, the protocols used are WiFi and TCP/IP. Sensors used are temperature, pressure, CO2, Current, etc., targeted towards the industrial domain.

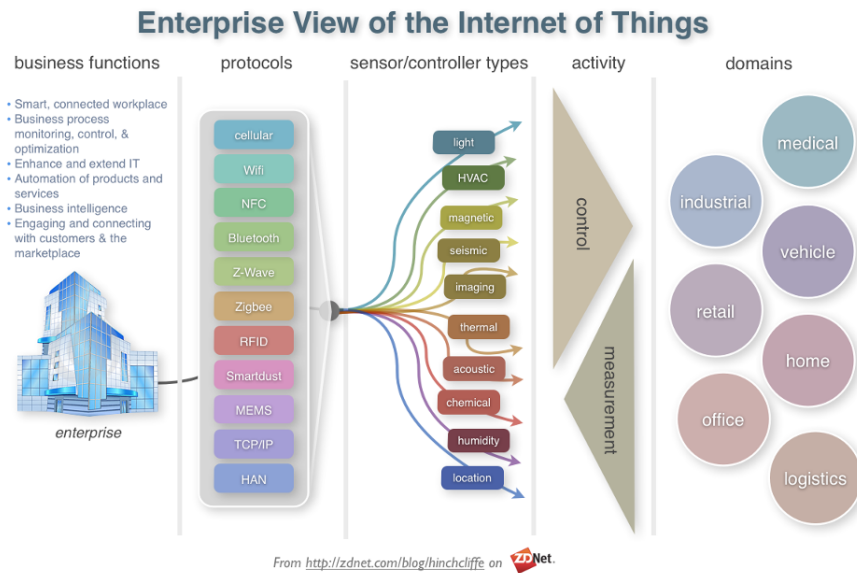


Figure 2.2. IoT Enterprise Functions

2.4 Database

A database is a structured collection of data. Several Database Management Systems (DBMS) are available, handling the data, transactions, and problems relating to the database. There are two major types of DBMS:

1. Relational DBMS - This uses a structured query language and is the traditional system

2. Non-Relational DBMS also known as NoSQL - This does not have a structured query.

SQL Database follows a relational data model in which the data is stored in tabular form in rows and columns. Some widely available relational database services are MySQL, MicrosoftSQL, Oracle, etc. NoSQL uses a non-relational data model. This does not require schema for the storage of data. It stores data in forms like documents, graphs, etc.

The three factors on which the database for this project was selected are

1. Scalability – SQL databases are vertically scalable, whereas NoSQL is horizontally scalable. Vertical scalability means that a single node's performance can be increased by adding resources like memory or processors to the same node. Horizontal Scaling means the number of nodes or servers can be increased to share the system load. Every development has a scope of future expansion. In this case, the best-suited option is SQL database because the system is expected to be expanded vertically.
2. Data Retrieval – Data retrieval is faster in NoSQL since the data is stored in objects containing all the related information. In SQL, the tables are linked together; therefore, JOIN statements are required, which are time-consuming.
3. System Maturity – SQL is a technology that has been around for quite some time. Most of the issues have been resolved. Several features are present in SQL, such as data confidentiality, integrity, and authentication. These features are yet to come to NoSQL. IoT system requires a robust database system; hence SQL is a suitable choice for this IIoT thesis project.

Since SQL is a suitable choice, MySQL DBMS has been selected for this project.

2.5 Web

The data from the sensors need to be processed. The business logic, database access, and the reply to the user require a service. Since the IoT device does not directly access the database service, it usually consists of data like sensor ID, reading value, etc. Using this data, it uses a web API to send the data to the database. This API collects the information

sent by the edge device and stores it in the database in the appropriate schema and table. As shown in the image above, the database's communication is done through a component called a server. Apart from storing the data from the edge device, Web services are also used for front end display and control of the devices. Database access should not be given to everyone due to security concerns. Therefore using the web services, we make certain information accessible and manipulatable by the user. Webservices are an essential part of the modern IoT devices.

2.6 Summary

In this chapter, the components used for development has been described at a high level. An IoT system has been developed for monitoring of industry energy usage and other parameters like CO2 levels, pressure, temperature etc. The edge device consists of a microcontroller device NXP FRDM K64f with Wi-Fi module ESP 8266. Various sensors mentioned in the table 1 have been integrated and tested. MySQL database is used for storing the data generated by the sensors. A web service, which hosts a website, has been used for handling the data from the sensors and pushing it to database. Additionally, this service is also used for displaying the data from the tables and it also handles requests for controlling equipment like air compressor, boiler, chiller etc. connected to the system.

3. SYSTEM ARCHITECTURE

The real-time energy monitoring system consists of a smart sensing and actuation component and a cloud-based computation component. The sensing and actuation component is placed in a factory or manufacturing plant with energy consumption equipment such as Air compressor, Motor, Lights, etc., for monitoring their energy consumptions. The smart sensing and actuation component contains various types of sensors and a microcontroller-based data acquisition system. This component allows the smart sensor to pre-process and log data and push the data to the cloud, where the data are stored systematically. The smart sensing and actuation component also enables controlling the equipment by making use of actuation relays. The cloud-based computation component comprises a web server, a database, data processing software, and user interface software. The data sent by the sensing and actuation component to the cloud are processed and stored in the cloud database. Section 3.1 describes the system components, and section 3.2 shows the Operational Capabilities of the System.

3.1 System Components

The Operational Context diagram of the system shown in Figure 3-1 depicts the operational entities and operational actors involved with the system. The system involved is a factory under which the sensing and actuation component and equipment are subsystems of the factory entity.

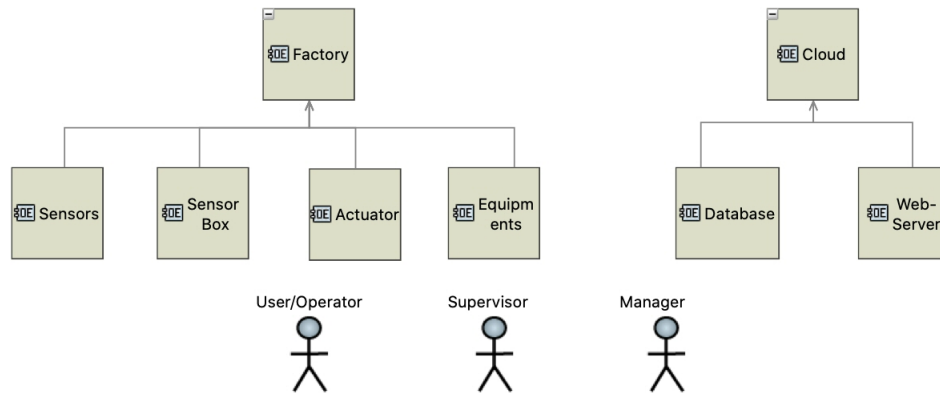


Figure 3.1. Operational Context Diagram

3.1.1 Factory

A factory is a collection of machines, which are used for manufacturing goods. These machines consume high amounts of energy that costs the company a significant amount of money. A factory contains several such machines that work in series or parallel to produce finished goods.

3.1.1.1 Equipment

A factory contains machines such as air compressors, motors, hydraulics, etc., called equipment. This equipment run for long hours and consume a significant amount of electricity. This thesis focuses on monitoring the energy usage of this equipment and controlling them to save energy.

3.1.1.2 Sensors and Actuators

Many types of sensors can be directly or indirectly used for energy usage monitoring. Table 1.2 lists sensors widely found in industries and have been made compatible with the system built. These sensors are used for testing and verification in this thesis work. For the actuator control, a power relay of CB1A-P 12V is used. The microcontroller and an intermediate circuit control this relay. As the user or the code sends a command, the appropriate action is taken to turn on/off the relay.

3.1.1.3 Sensor Box

The sensor box consists of a microcontroller device, Wi-Fi module, power jack, sensor interfacing connectors and relay interface connectors.

The operational capabilities identified from the requirements are shown in Table 3.1.

Table 3.1. Capabilities of the System

ID	Capability	Fulfilled by
1	Sensor Configuration	Sensors
2	Data Acquisition	Sensor Box
3	Sensor Failure Detection	Sensor Box
4	Data Processing	Sensor Box
5	Wi-Fi connection	Sensor Box
6	Offline Logging for Internet Failsafe	Sensor Box
7	Store Data in Cloud	Cloud
8	Retrieve Data to Matlab	Cloud
9	Store Legacy Data	Cloud
10	Level Based Login	User/Operator, Supervisor, Manager, Web-Server
11	View Real-Time and Legacy Data	Cloud, User/Operator, Supervisor, Manager
12	Control Connected Equipment Remotely	Supervisor, Manager, Cloud

3.1.1.3.1 Microcontroller

The microcontroller is the brain of the hardware system. It is responsible for Sensor configuration, Data Acquisition, Communication, preparing data to be pushed to the cloud, and handling errors. As shown in Table 3.1.1-3, the microcontroller is responsible for fulfilling almost half the entire system's capabilities.

3.1.1.3.2 Wi-Fi Module

The microcontroller acquires and processes data from the sensors. It then prepares data to be sent to the cloud. Wi-Fi module is responsible for the communication between the microcontroller and the cloud. It establishes a TCP/IP connection using Wi-Fi, sends data prepared by the microcontroller device, and returns the server's response to the microcontroller.

3.1.1.3.3 Power Supply

For the functioning of the microcontroller and sensors, it requires power. Since sensors require 24V DC to operate, a DC adapter providing 24 Volts is used for the system. The microcontroller requires 5V to operate; therefore, a voltage regulator is used to supply 5V to the microcontroller. Some sensors such as Temperature sensors, CO2 sensors require a 24 V DC supply to operate. This power jack supplies 24 volts of power to the sensors too.

3.1.1.3.4 Sensor Interfacing Connectors

There are various types of sensors available in the market, which have different output ranges and connectors. To interface different types of sensors and sensor connectors, the system uses a sensor interfacing connector. Patching of sensor wires is required to make it compatible with the system. The sensors are patched with an RJ-45 cable using the appropriate wires mentioned in later chapters. This allows the system to support a set of commonly used sensors for energy measurements.

3.1.1.3.5 Relay Interface Connectors

A power relay is used to control equipment remotely. The microcontroller in the sensor box controls this relay. Remotely control the relay allows the implementation of energy-efficient algorithms that can turn on or turn off equipment determined by the algorithm. Relay interfacing is done using a 3.5mm adapter attached to the sensor box. The relay box consists of power lines going through the relay, and the control terminals of the relay are

attached to a 3.5mm cable. Control of the relay is done through a microcontroller using the 3.5mm adapter.

3.1.2 Cloud

Cloud computing refers to the processing and storing of data in a remote computer known as a server. This is required since the smart sensors have memory and processing constraints. For example, the Sensor Box in this system has a microcontroller device with limited computation power and limited memory. Cloud services are used to make the system providing fast computation and last data storage. Cloud computing is also practically useful when the data or services need to be accessed by multiple users (shared access). A system shown in Figure 3.2 is a cloud under which the database and web-server are subsystems. The operational actors interacting with these systems are User/Operator, Supervisor, and Manager.



Figure 3.2. Components in the Cloud

3.1.2.1. Database

Database refers to a structured set of data held in a computer. In this system, we store the sensor readings in a structured manner in database tables. In this study, MySQL is the database service used for storing data. The database is also used to store legacy/historical data. Authorized users can access the database.

3.1.2.2 Webserver

The webserver is a cloud hosting server where several PHP and HTML scripts are stored. Each script is intended to perform a dedicated task. The webserver also helps restrict access to the database, allowing users to view or modify certain parts of data.

3.1.3 User

There can be several types of users in this system. As shown in Figure 3.1, the most common types of users are Operator, Supervisor, and Manager. The type of user determines the level of access privilege that the user has. Determining the user type is a critical component because giving unauthorized access to the system may result in the manufacturing process's failure. For example, an operator can only view data and not remotely turn on/off the equipment, whereas a supervisor and manager can. The supervisor may not have access to legacy data, but the manager may have to observe trends. Giving access to legacy data to unauthorized users can also hurt the business.

3.2 Operational Capabilities of the System

The Operational capabilities (OC) of the system is shown in Figure 3.3. Each capability is fulfilled by a system, subsystem, actors or combinations. It also shows which actors are involved with the capabilities.

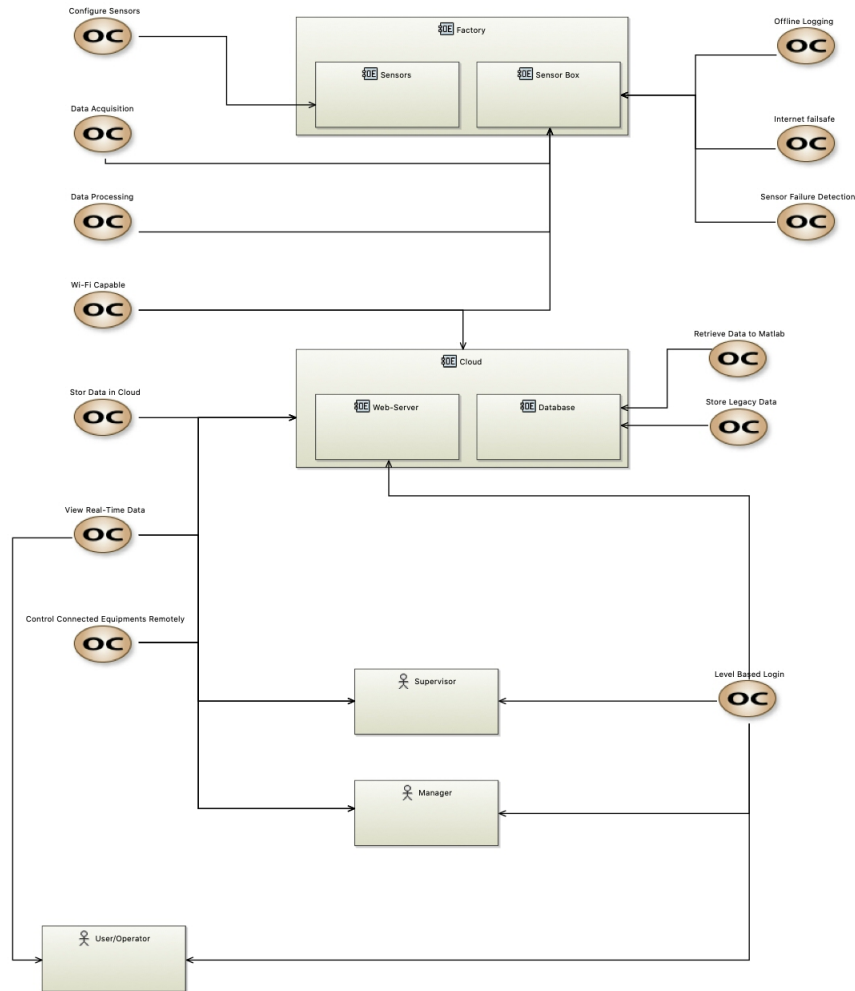


Figure 3.3. Operational Capabilities

Capabilities are described in detail by Scenarios. These scenarios determine the steps or instructions to follow to fulfill the capability. The following diagrams show how capabilities are fulfilled.

3.2.1 Sensor Configuration

Sensor configuration is a process (Figure 3.4) which includes the following steps:

1. Find all connected sensors
2. Determine electrical signal type and range of the sensor output
3. Switch the corresponding signal selector depending on the signal type
4. Determine the unit conversion if the sensor does not exist in the pre-defined list
5. Multiplication factors, Wi-Fi credentials and sensor information stored in a look up table in the firmware

These steps are done by a technician or engineer familiar with the system, and this process falls under the installation part of the system.

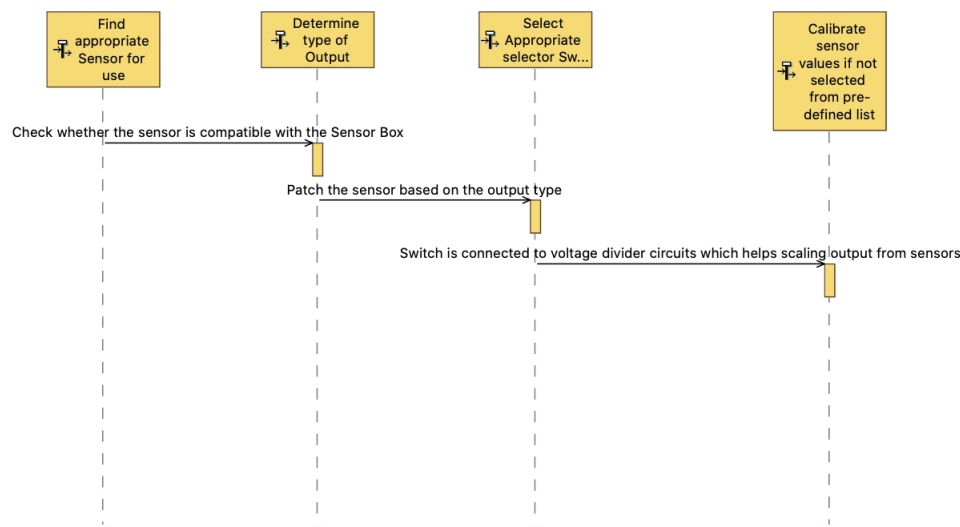


Figure 3.4. Configure Sensor Scenario

3.2.2 Data Acquisition

Figure 3.5 describes the data acquisition process from sensors by the microcontroller.

- Initialize Microcontroller ports - This step configures microcontroller ports as an input mode, allowing the microcontroller to read signals from the sensors.

- The read function allows the microcontroller to read the sensor's current value/status using the port defined and activated in the previous step.
- According to the configuration, the data can either be sent as is read or averaged for 5 seconds and then sent to the cloud. The sampling time is currently hard coded. However, provisions are made to make it remotely configurable.

This process is programmed in the microcontroller firmware, and the second step is executed periodically without human interaction. The frequency of reading is determined by the sampling time selected by the developer as per requirement. For example, if the user requires a 5 seconds interval, the microcontroller will keep reading the data for 5 seconds and then send it. Since the microcontroller reads very fast, around 25,000 samples are read in 5 seconds and then averaged before sending.

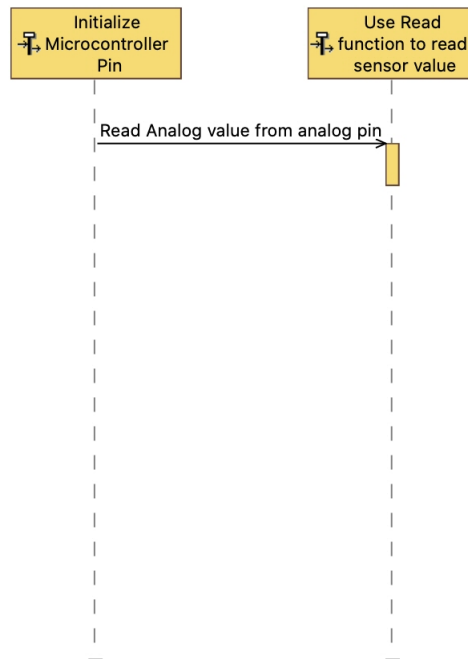


Figure 3.5. Data Acquisition Scenario

3.2.3 Sensor Failure Detection

Figure 3.6 defines the process for detection of sensor failure. This process is not implemented in this version of the firmware. However, a different group working on this project

had previously implemented this part and can be merged in future update of the firmware. If the microcontroller reads ambiguous values that are not in the sensor's range, it will warn the monitoring data user. Since the range of sensors can be defined in prior and stored in the system, it becomes easier for the code developer to recognize such ambiguity. The steps include are as follows:

1. Check if the sensor data value is in the predefined range stored in the lookup table in firmware. The predefined ranges are determined by the developer when developing the system. The sensors provided to integrate are used for determining these ranges and store in a look-up table. If a new sensor is to be added, the developer will need to calibrate and store new values in the lookup table,
2. Display warning on debugging terminal to the user if the values are not in range. A Debug terminal is an application on a computer that connects to the microcontroller and debugging the firmware.

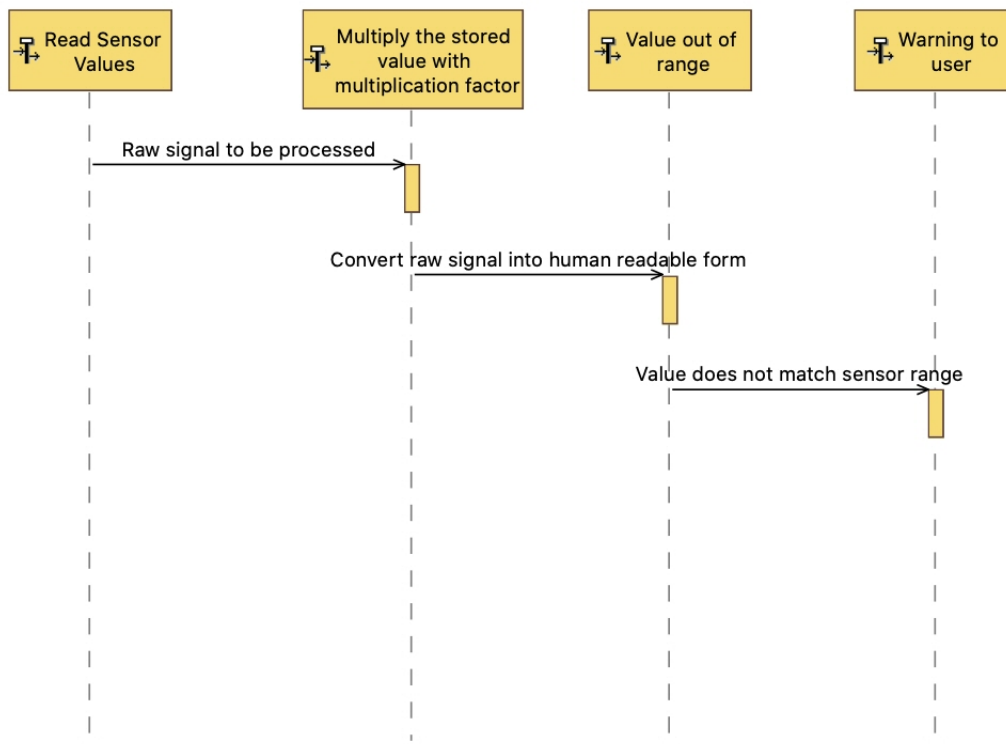


Figure 3.6. Sensor Failure Detection

3.2.4 Data Processing

The signals from the sensor to the microcontroller are analog and converted to digital values between 0-1. These are not in the standard unit and need to be converted by multiplying them to multiplication factors. These multiplication factors are obtained by calibrating by testing. The read values can either be sent as they are read or be averaged for 5 seconds based on requirements. The current version of the firmware reads raw sensor values, converts to standard units (psig, Ampere, ppm etc.) and sends it to the server. Since microcontroller has ten sensor ports, the delay between reading each port is set to 1.5 seconds using the “wait” command. Therefore, the values for all sensors in use, are uploaded to the server in approximately 15-16 seconds. To optimize the required delay, future development may make use of an RTOS timer. Note. The wait command is only executed if there is no backup data (see section 4.2.6, explanation for Figure 4.23). If backup data exists, the wait command will not be executed, and one reading of backup data is sent to the server.

Figure 3.7 shows the data processing scenario which includes the following steps:

1. The microcontroller firmware converts each input port’s raw value to pre-determined units based on the type of sensor connected to the port (defined in the sensor configuration part).
2. Firmware applies filter if it is selected (such as averaging) to the multiplied value.
3. The firmware then prepares TCP/IP packets with the data appended to the payload to be sent to the cloud to store in the database.

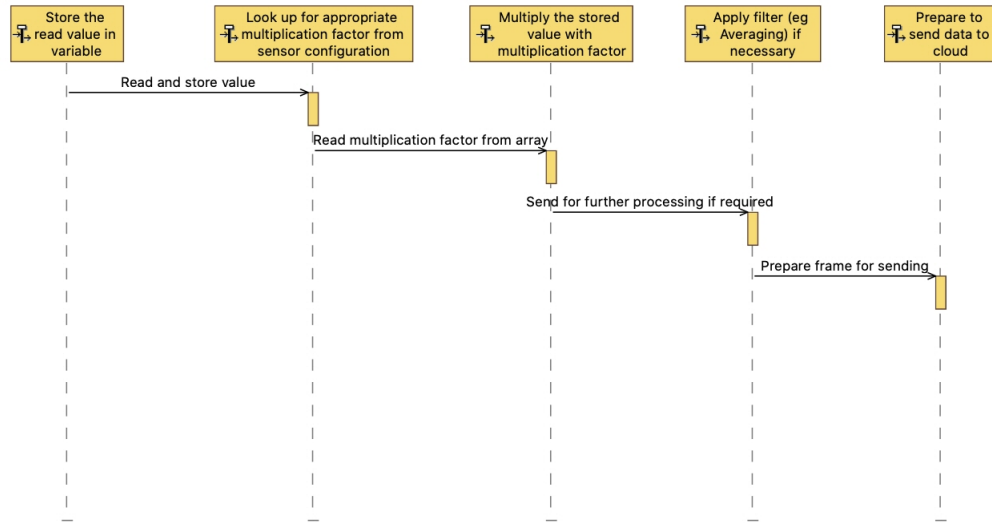


Figure 3.7. Data Processing Scenario

3.2.5 Wi-Fi Connection

The internet is required to send data to the cloud. For this reason, the Wi-Fi module is integrated with a microcontroller to enable internet connectivity. Figure 3.8 shows that the system is Wi-Fi capable. The steps include as follows:

1. Initialize the sensor box and microcontroller.
2. It reads configuration file from SD card and generates a table containing multiplication factors, and also sets Wi-Fi credentials.
3. Once the connection is established, it starts reading sensor values.
4. It converts raw data and prepares to send the data to the cloud by preparing a TCP/IP packet.
5. It then sends the packet to the webserver using a Wi-Fi internet connection, and the webserver later stores it in the database.

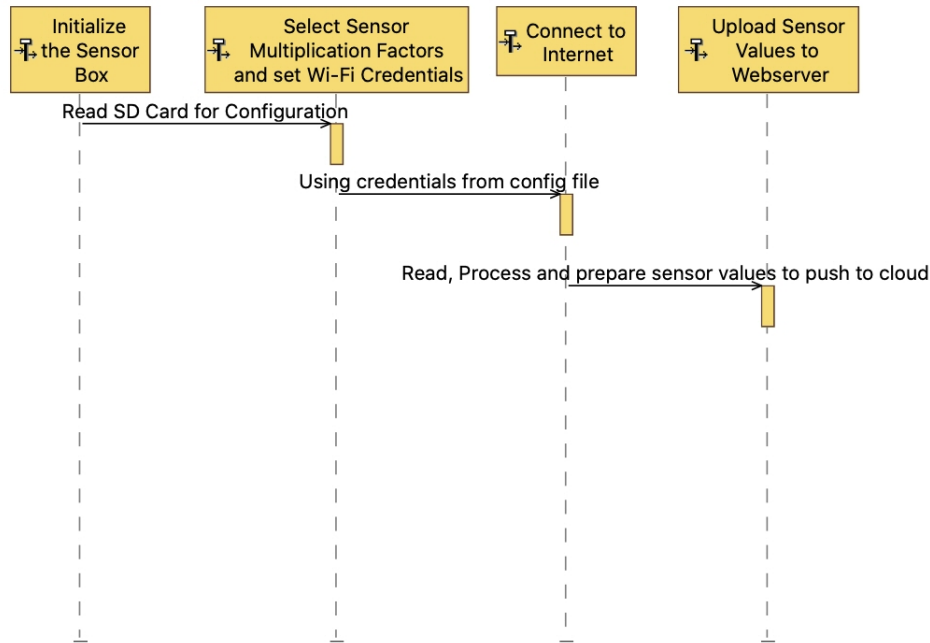


Figure 3.8. Wi-Fi Capable Scenario

3.2.6 Internet Failsafe and Offline Logging

Suppose the system is unable to connect to the cloud due to internet failure or any other anomaly. In that case, the fail-safe mechanism allows the system to temporarily store the sensor data into an SD card attached to the microcontroller (Figure 3.9). Several errors listed below were recognized during system testing and error handling mechanisms defined within the firmware to keep the system robust.

1. Server busy
2. Server not responding
3. Unable to connect
4. Hardware failure (like Wi-Fi module wire broken, etc.)

If the system failed to send data to the cloud, it will temporarily store the data and attempt to reconnect to the internet and reconfigure itself. This process belongs to the sensor box component of the system. For detailed explanation, see Section 4.2.6.

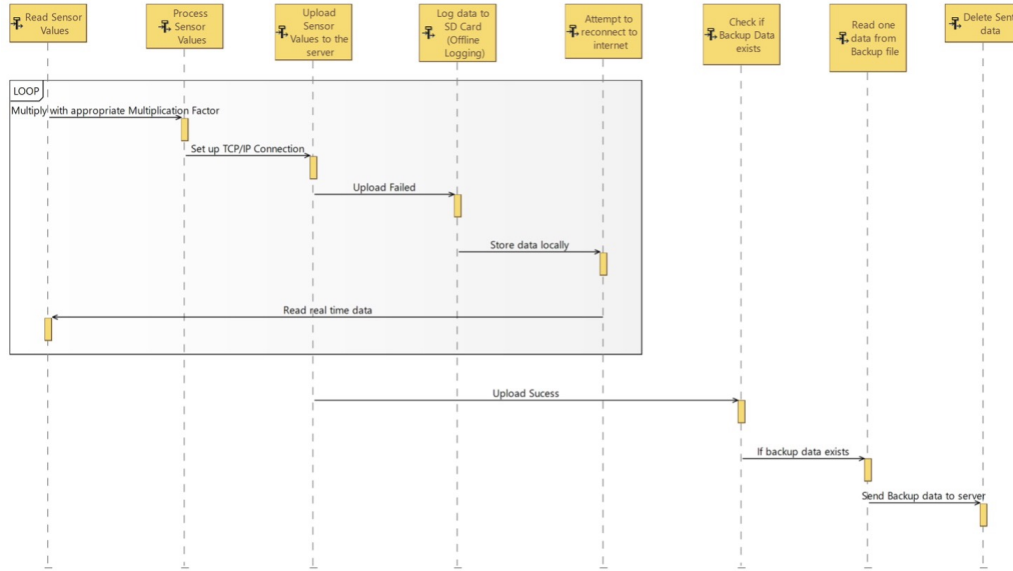


Figure 3.9. Internet Fail-safe Scenario

3.2.7 Store Data in Cloud

Figure 3.10 shows the process for storing sensor data from hardware to the cloud. It includes the following steps:

1. The microcontroller prepares and sends the TCP/IP packet to webserver. Each packet contains one sample of data read from the sensor. The packet also contains the timestamp when the value was read.
2. The web server part of the cloud receives the TCP/IP packet from the microcontroller. Since sending takes more time than reading, this system's fastest data sending time is 1 second per packet
3. It arranges and attaches a timestamp to each packet of TCP/IP data when it was received. Then it generates a query to store the data along with a time stamp. The system has two timestamps. One is appended by a PHP page that uses internet time, the other being the Real-time clock implemented in the microcontroller.
4. Once the query is executed, it stores the data in appropriate table and query returns success.

Preparing and uploading data to the cloud is done at the microcontroller end, whereas pushing values to the database and storing it is done at the web server's cloud end. This process will be described in Chapter 4 in detail.

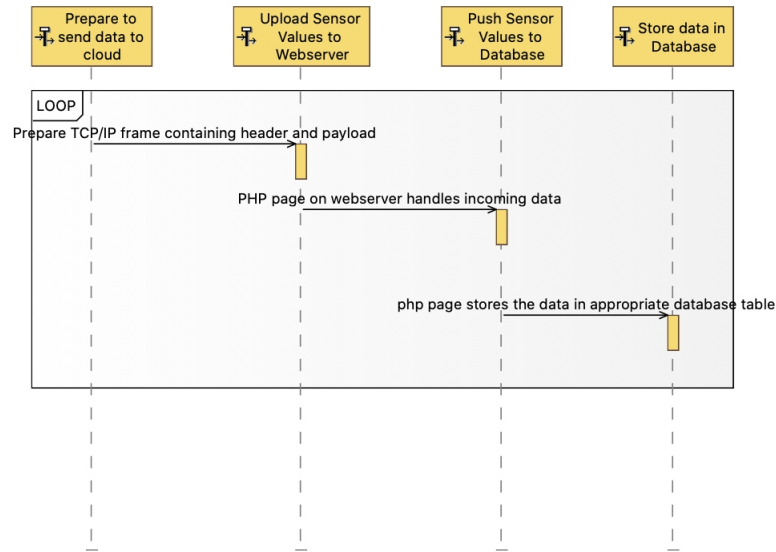


Figure 3.10. Send and Store Data in Cloud Scenario

3.2.8 Retrieve Data to Matlab

Figure 3.11 describes the steps to access the database from a Matlab function. This feature is available to the authorized user, such as a data analyst, manager, or developer. The user needs to download a Database explorer add-on for Matlab and download a JDBC connector from the MySQL website. After installing the required tools, the user can see the example code shown in Appendix C, which shows the commands to establish a connection to the database. Once the connection is established, the user can either use the MySQL queries or inbuilt functions to retrieve data and store in Matlab variables and later be used for analysis. Using these tools, users can also perform operations other than retrieving data, like alter table, delete table, etc. The details can be found in Appendix C

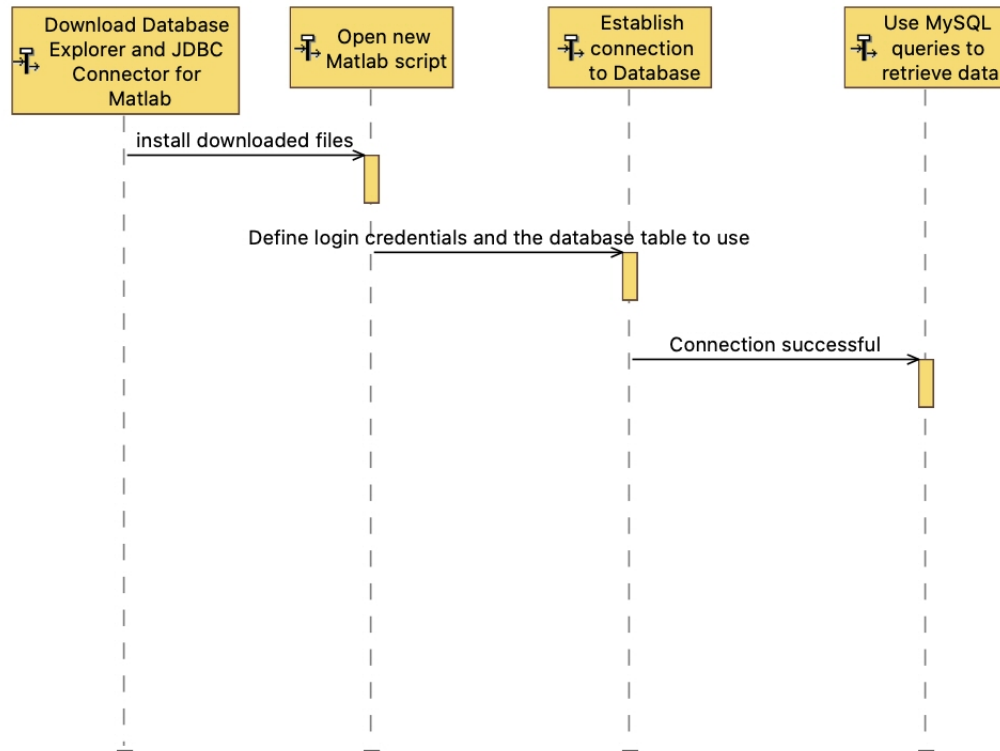


Figure 3.11. Retrieve Data to Matlab Scenario

3.2.9 Store Legacy Data

Legacy data is the historical data generated by the cloud system over a long period. A large amount of data is required To understand the trend of energy consumption of a factory. The energy trend differs from season to season as well. This system keeps collecting data for a long time and generating legacy data for analysts who can help save energy. Figure 3.12 is the process that runs in a loop from the time system is started. Since it keeps collecting data in real-time and uploading it to the server, this data is jammed historic data over a long time.

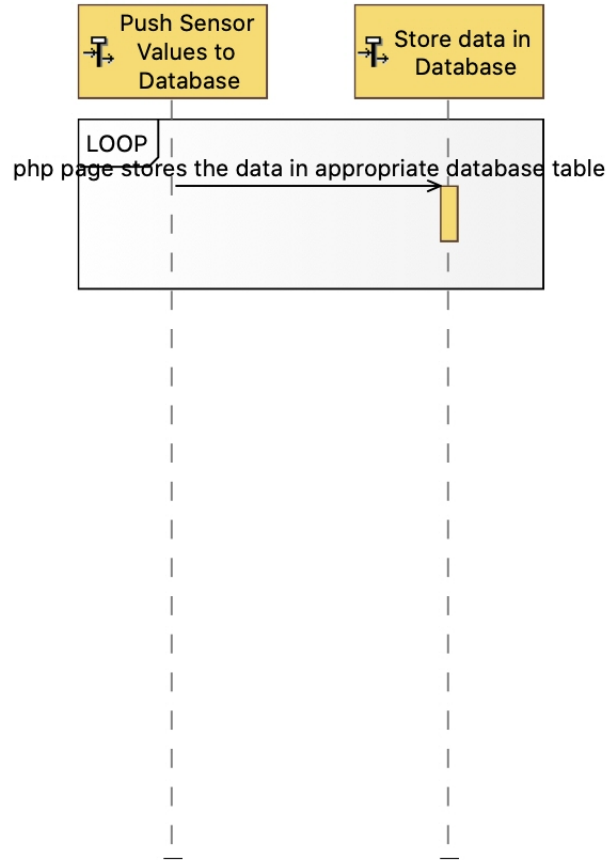


Figure 3.12. Store Legacy Data Scenario

3.2.10 Privilege Based Login

The level-based login feature is proposed in Figure 3.13. Whenever a user is given access to the system, their roles are defined. Based on the role and management hierarchy, the access privilege of the user can be controlled. If a manager or a team leader accesses the system, they will have permission to view the current data, legacy data, and equipment control. If a data analyst accesses the system, they might have access to the legacy data and may not control equipment connected to the system.

In the diagram below, two of the scenarios are described in the alternate loop, which shows that the user may access either one or both of the system's features. The feature is not yet implemented on the front end web, but provisions have been made in the database for level based login. This capability must be implemented by front-end developer.

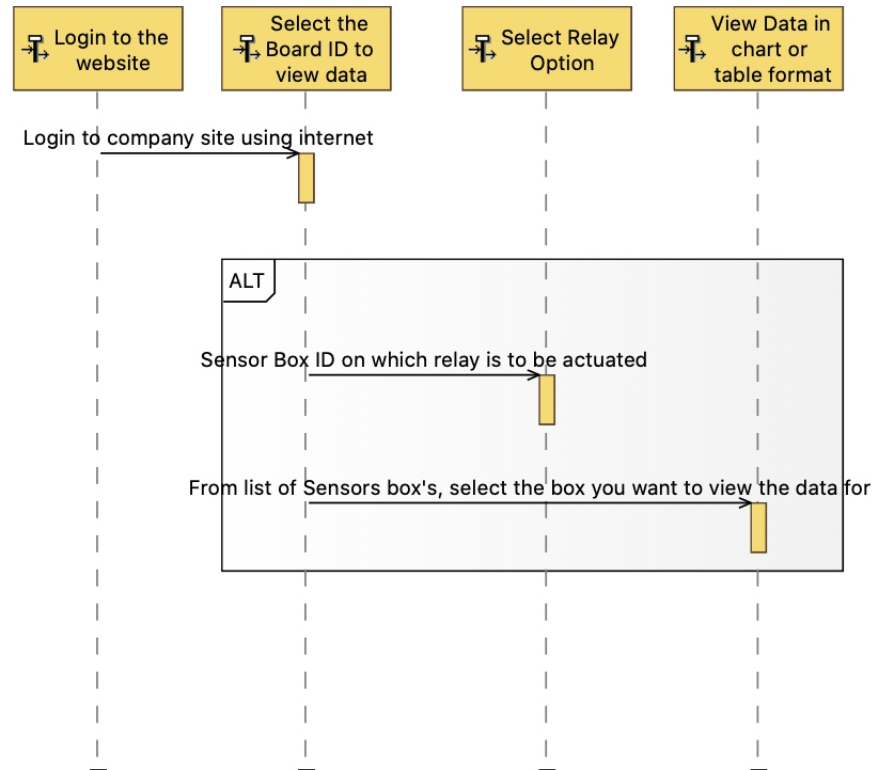


Figure 3.13. Level Based Login Scenario

3.2.11 View Real-Time and Legacy Data

The process in Figure 3.14 shows the steps to view real-time data. The system must have a web-based user interface, which allows the authorized company personnel such as operator/manager/developer to log in to the system and monitor real-time data. Steps include logging in to the database, selecting the sensor box system to view data in chart or table form. This feature should be implemented by a web developer.

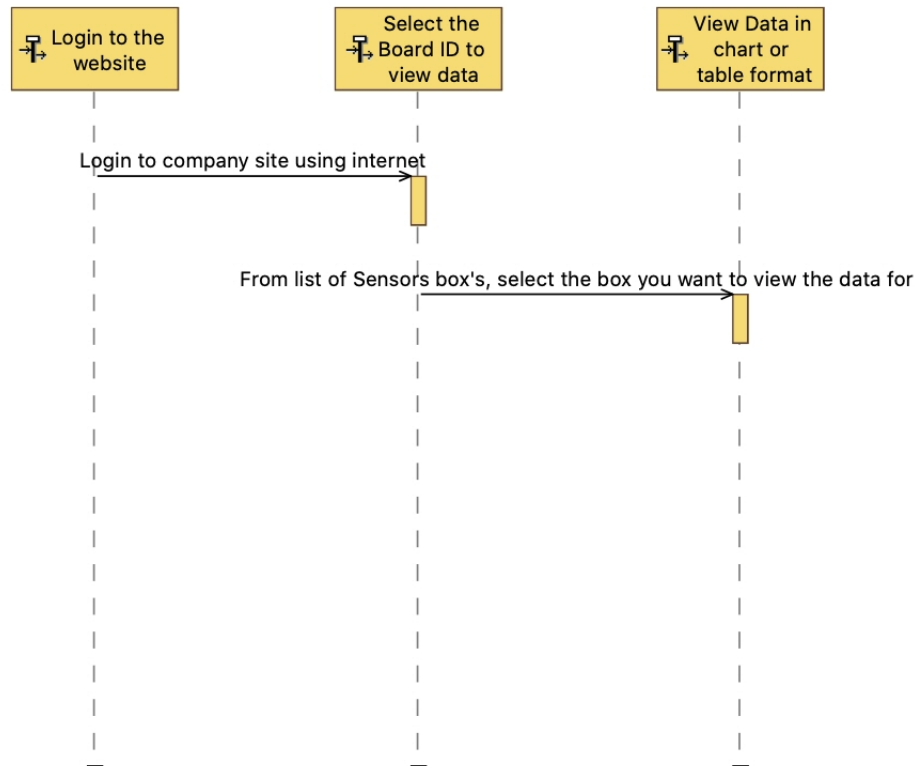


Figure 3.14. View Real-Time and Legacy Data Scenario

3.2.12 Control Connected Equipment Remotely

Figure 3.15 shows the steps for remotely controlling the equipment connected to the system. This is a combination of the sensing component and cloud processing component. The system will only show boxes belonging to the particular company the employee belongs to. Following are the steps involved to control connected equipment remotely:

1. The user with the proper privilege first needs to login to the system website
2. The user selects the specific sensor box that he has the privilege to see.
3. Out of the four available relays (controllers), the user can select to turn on/off any one or more relays and click on submit. The equipment attached to the relay can be found in another table in the database called equipment info. All relays need not be used—the information regarding which equipment connected to what relay is stored in the database.

4. The state of the respective relay is changed in the database system.
5. Sensor box firmware receives the command to change the status.
6. The microcontroller takes appropriate action as requested by the user and turns on/off the equipment.

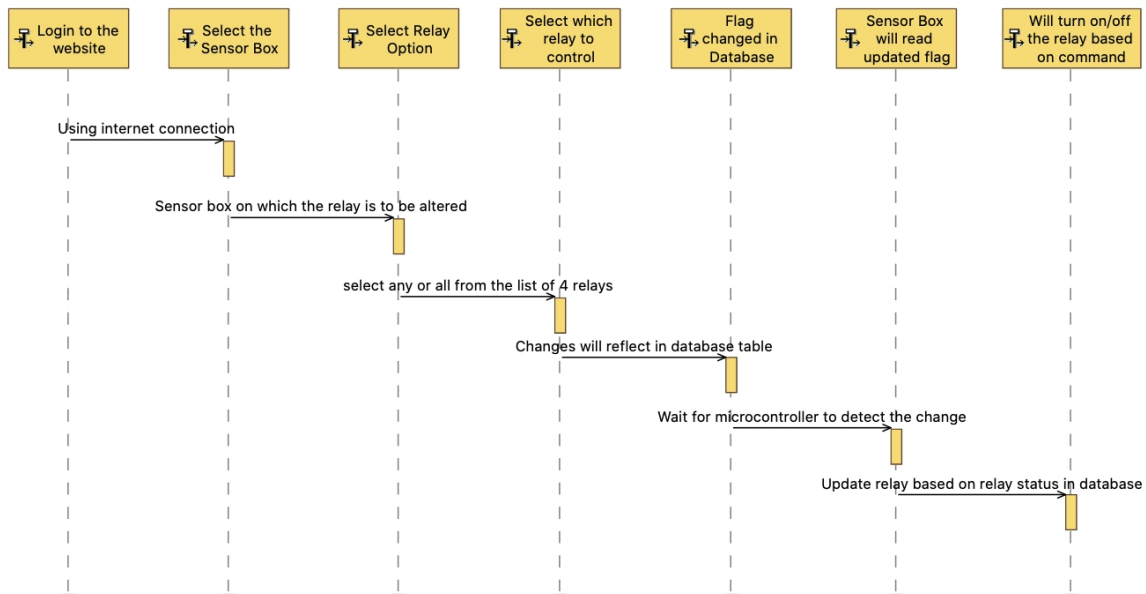


Figure 3.15. Control Remote Equipment Scenario

4. SMART SENSOR DESIGN

Hardware is a very important part of the system. Hardware is responsible for interfacing various sensors to collect data such as temperature, pressure and current from the concerning environment, for sending configuration information and sensor data to cloud where it is stored into database table, and for interfacing various actuators that are used for turning the appliances on or off.

4.1 Hardware

Figure 4.1 is the top-level block diagram of the hardware. The hardware consists of various components as listed below:

1. The sensor interface circuit (Conversion Circuit and Patch connector)
2. Actuator circuit
3. Power supply circuit (24V Power Supply and Regulator)
4. Microcontroller circuit
5. SD Card
6. Wi-Fi module
7. PCB

Each of these components has been explained in detail in the following subsections.

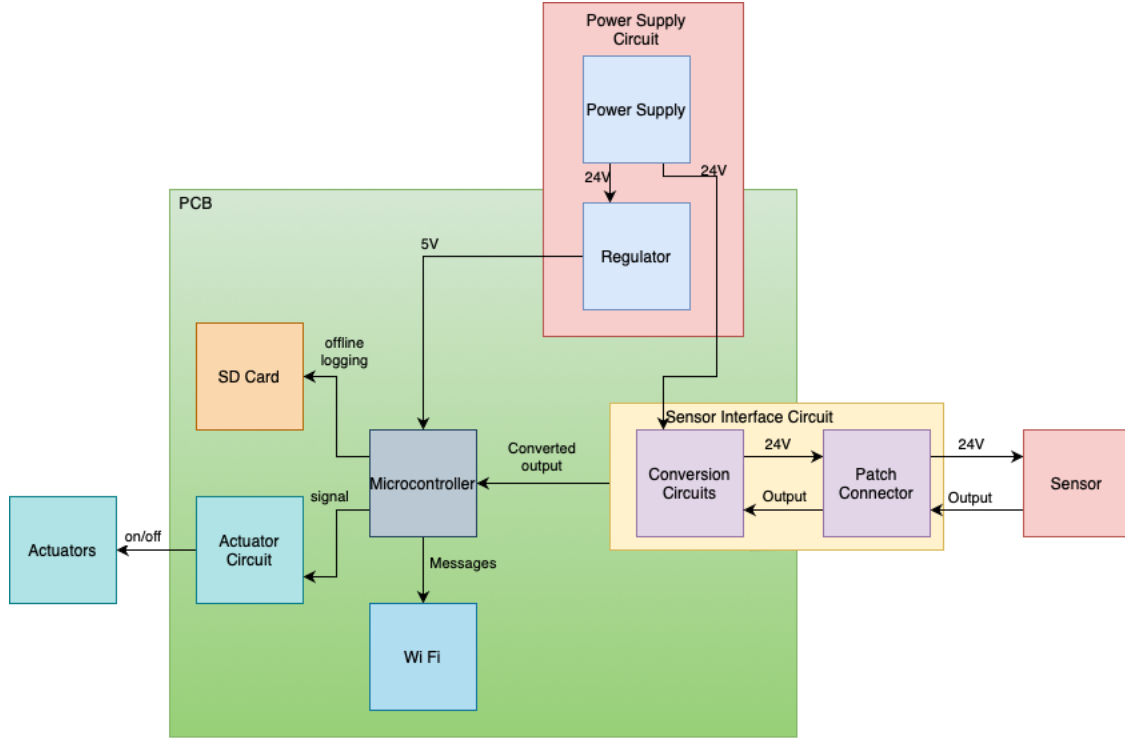


Figure 4.1. Hardware Diagram

4.1.1 Sensor Interface Circuits

The sensor interface circuits consist of conversion circuits and patch connectors. The sensor circuits are designed to enable the data acquisition system's connection to various types of commonly available sensors. The system is designed to interface 4 types of sensor output. The sensors have a binary output of 0-5v or 0-12v or analog output in various ranges. This study only considers 4 types of commonly used sensor analog outputs in industrial equipment: 0-2.5V, 0-5V, 0-10V, 0-20mA.

The microcontroller can take input of 0-3.3V. Any input above that would cause damage to the microcontroller. Therefore, interface circuits are required to covert the sensor output to 0-3.3 digital microcontroller input. A special sensor computer interface circuit was developed to make the system easily connect to any of the sensors. Figure 4.2 shows the prototype interface board containing 4 interface circuits for each microcontroller input port.

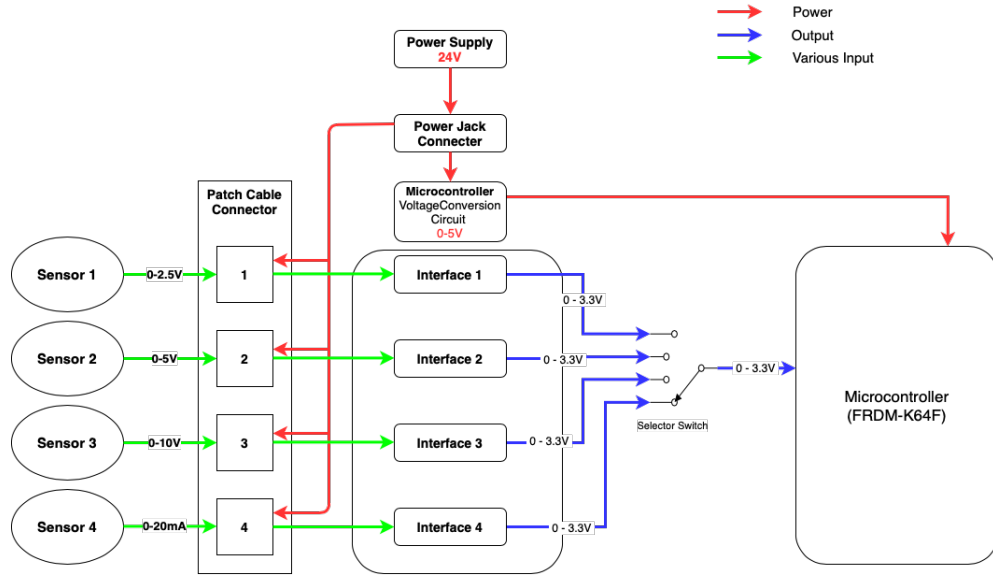


Figure 4.2. Interface Board Block Diagram

The circuit has the following components:

1. Signal Conversion circuit
 - (a) 0-2.5V Conversion Circuit
 - (b) 0-5V Conversion Circuit
 - (c) 0-10V Conversion Circuit
 - (d) 0-20mA Conversion Circuit
2. Connector
3. Interface Selection Circuit
4. Sensor ports

4.1.1.1 Signal Conversion Circuit

4.1.1.1-a 0-2.5V Conversion Circuit

Since the microcontroller can take input signal up to 3.3 Volts, no circuit is required to step down the voltage. Therefore, this kind of input is directly connected to the microcontroller device.

4.1.1.1-b 0-5V Conversion Circuit

The design uses a voltage divider to drop the voltage from 0-5V to 0-3.3V (see Figure 4.3). These values of resistors are obtained using the voltage divider formula, as shown in Equation 4.1.

$$V_{out} = \frac{V_s \times R_2}{R_1 + R_2} \quad (4.1)$$

Since the microcontroller can only take 3.3V input, the output from the sensor will not exceed 5V. The resistors used for input of 5V are 1.7k ohms and 3.3k ohms.

V_s is the source voltage, measured in volts (V),

R_1 is the resistance of the 1st resistor, measured in Ohms (Ω).

R_2 is the resistance of the 2nd resistor, measured in Ohms (Ω).

V_{out} is the output voltage, measured in volts (V),

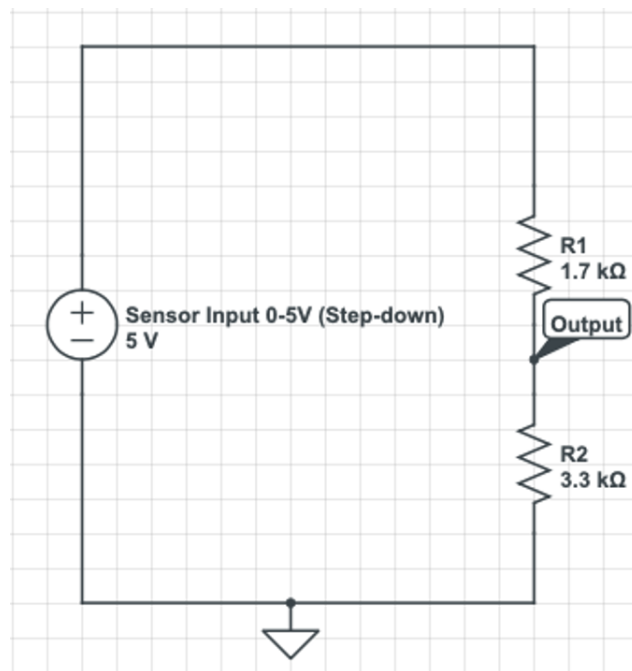


Figure 4.3. 0-5V Conversion Circuit

4.1.1.1-c 0-10V Conversion Circuit

The design uses a voltage divider to drop the voltage from 0-10V to 0-3.3V (see Figure 4.4). For this calculation, the value of V_{out} is taken as 3.3V. The resistors used for input of 10V are 2k ohms and 985 ohms in Equation 4.1.

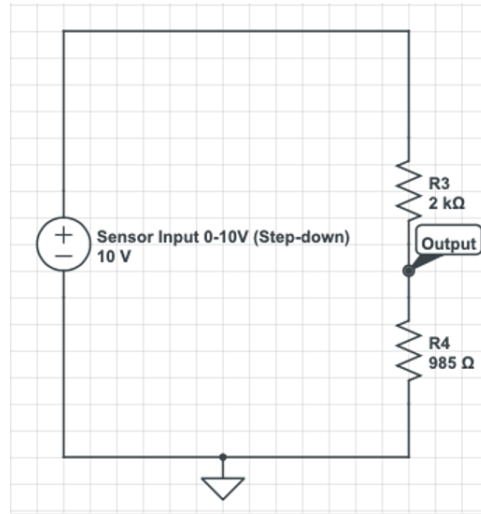


Figure 4.4. 0-10V Conversion Circuit

4.1.1.1-d 0-20mA Conversion Circuit

The 0-20mA conversion circuit is shown in Figure 4.5. As V is taken as 3.3 and I is taken as 20 mA, R is 165 Ohm according to Equation 4.2.

$$V = I \times R \quad (4.2)$$

- V is a voltage signal that goes to the microcontroller
- I is the current from the current source
- R is the resistance value required to obtain the desired V value

As V is taken as 3.3 and I is taken as 20 mA, R is 165 Ohm according to Equation 4.2.

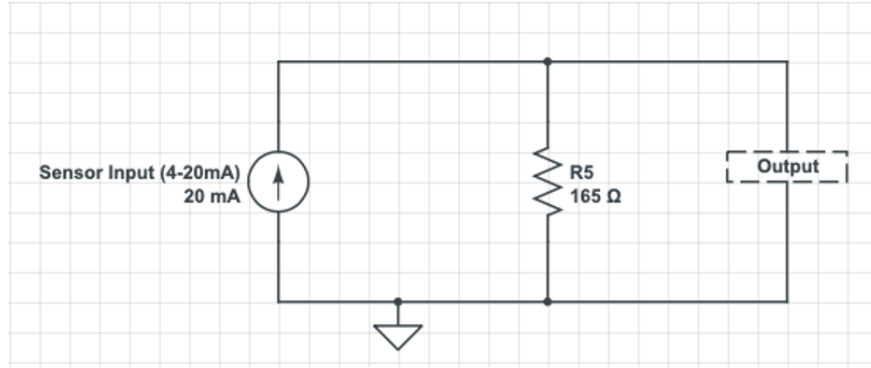


Figure 4.5. 0-20mA Conversion Circuit

4.1.1.2 Sensor Connector

An 8 line patch cable was chosen to provide a standard connection to the system. The patch cable initially used for the network provides 8 lines in a standard form factor. The patch standard chosen was T568A. The pin layout is used to select the conversion circuit in an interface and provide the sensor's power/ground. Figure 4.6 shows the diagram, which defines the pins to be connected for the chosen sensor. For example, if the sensor chosen produces a maximum of 5 volts, the sensor's output will be connected to pin 2. Also, if the sensor requires power to operate, pin 7 and pin 6 will provide 24 volts and ground, respectively. Note, Only pin 6 should be used as ground for power supply. Pin 3 is sensor ground.

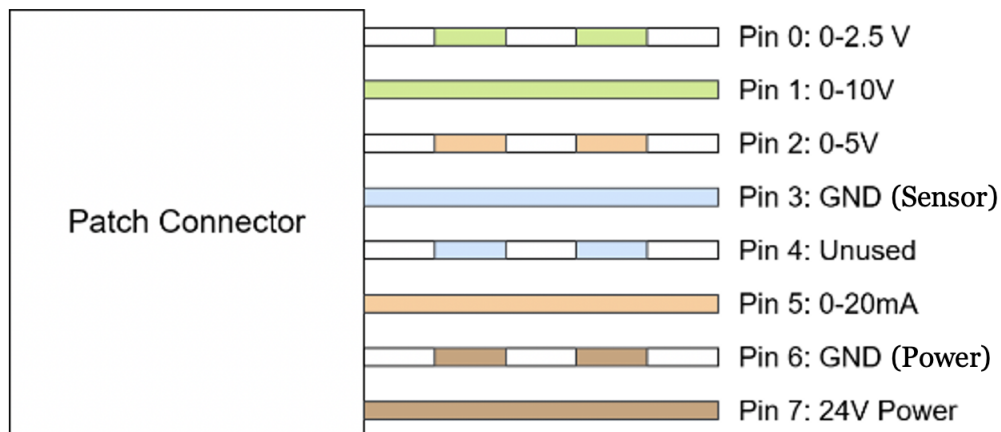


Figure 4.6. Sensor Connector Pin Layout

Table 4.1 lists the pins to be connected for 4 types of sensor output.

Table 4.1. Sensor List

Pin	Wire Color	Input	Example Sensor
0	Green/White	0-2.5V	CTV-C, CTV-E
1	Green	0-10V	C7232A1008
2	Orange/White	0-5V	TOAV22, T-ASH-G1-200, HE-67S3-0N0BT
3	Blue	Sensor GND	
4	Blue/White	unused	unused
5	Orange	0-20mA	CDI-5200
6	Brown/White	24V GND	
7	Brown	24V Power Supply	

4.1.1.3 Interface Selection Circuit

The patch pin switch is needed to select the correct conversion circuit. An sp4t switch is used to select the input type to demultiplex the circuit to a single connection. This completes the path to the K64F port. The block diagram in Figure 4.7 shows that the output will be directed to the 0-5V conversion circuit by switching to pin 0. The patch connection also can provide the sensor with 24V power and ground. Note that pin 3 ground is the sensor signal ground, and pin 6 is the AC power ground. They are not connected directly.

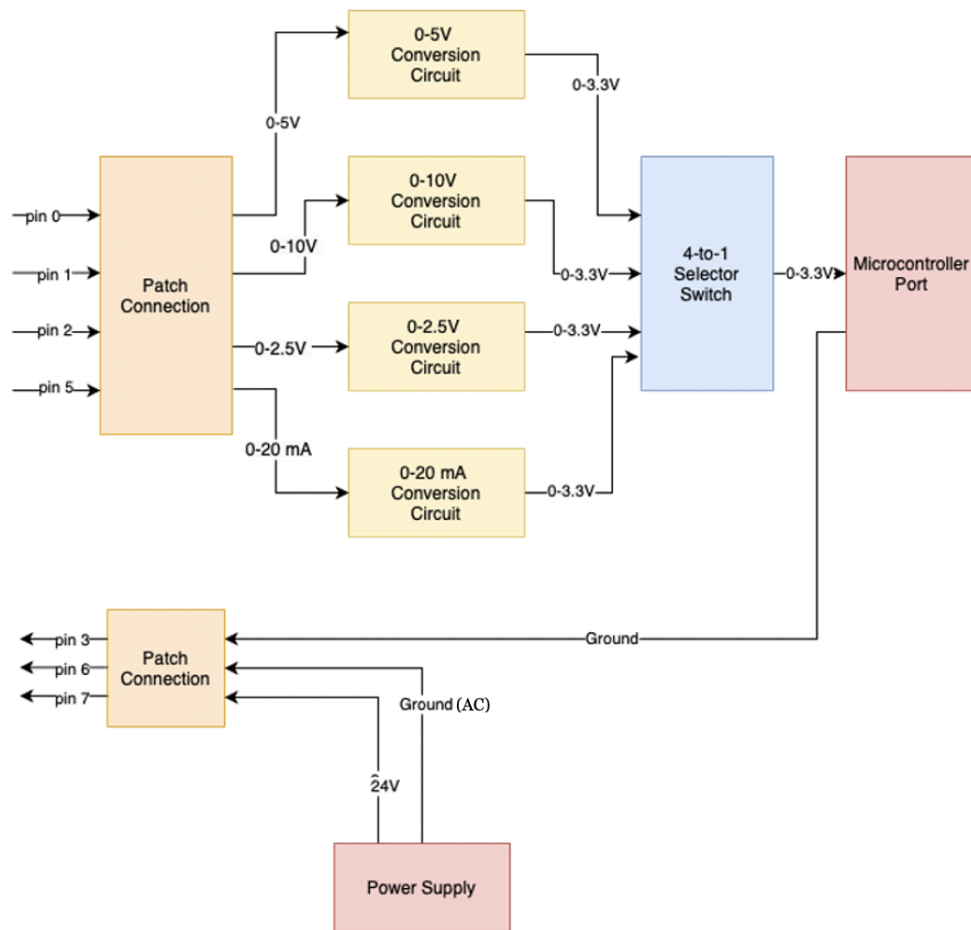


Figure 4.7. Conversion Selection Block Diagram

4.1.1.4 Sensor System

Figure 4.8 shows the prototype sensor system design with 10 ports.

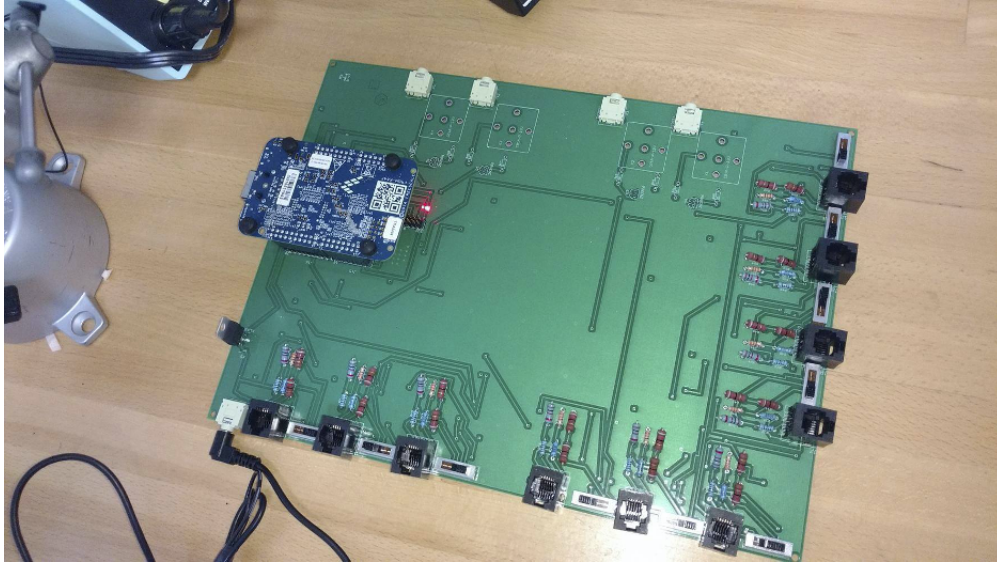


Figure 4.8. Sensory System

4.1.2 Actuator Circuit

The actuator circuit shown in Figure 4.9 turns off a 5V system when supplied a 3.3 voltage signal. The circuit uses a relay and a bipolar junction transistor. The actuator circuit was tested off the board but implemented on the PCB design. A signal needs to be output from a port to test the actuator circuit. The relay used in this system is CB1A-P-12V.

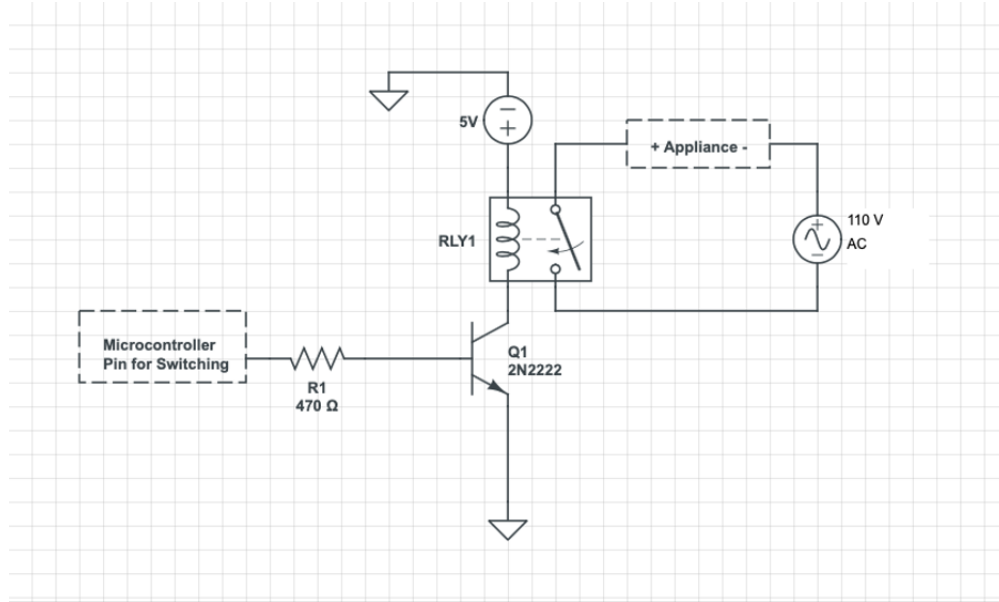


Figure 4.9. Actuator Circuit

4.1.3 Power Supply Circuit

The conversion from the 24V power supply to the 5V microcontroller is done with a linear regulator shown in Figure 4.10. This resistive type regulator heats during sustained load and requires a heatsink. The implementation uses a KA7805AETU 5V regulator and a TO-220/TO-202 heatsink. The microcontroller should have consistent power from the circuit. The heatsink should also be able to dissipate the heat generated quickly enough to prevent overheating.

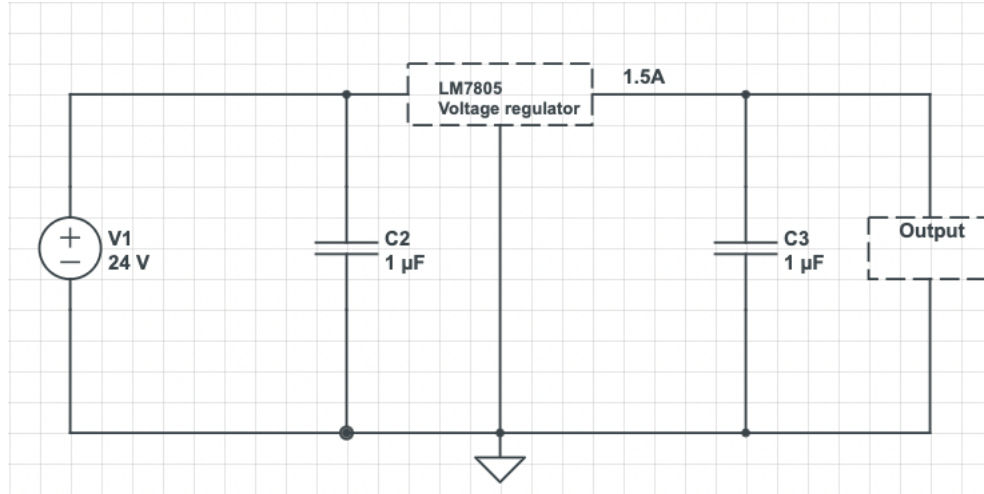


Figure 4.10. Power-Microcontroller Interface Circuit

4.1.3.1 Power Supply

The power supply supplies 24V to the board. The connection is done with a DC power jack connector plug. For the implementation, a Wsdcam DC Power Jack Adapter Connector Plug pair was used.

4.1.4 Microcontroller Circuit

Figure 4.11 shows the microcontroller block diagram. 10 ports are connected to the Analog input pins of the microcontroller. The microcontroller converts these analog signals to digital signals using an Analog to Digital Converter (ADC). 4 relays are also connected to the microcontroller's digital output pins. When the output pin is set to 0, the relay turns off (open); when it is set to 1, the relay turns on (close). Microcontroller also has a Wi-Fi module connected to it. A 5 volts power is supplied to the microcontroller.

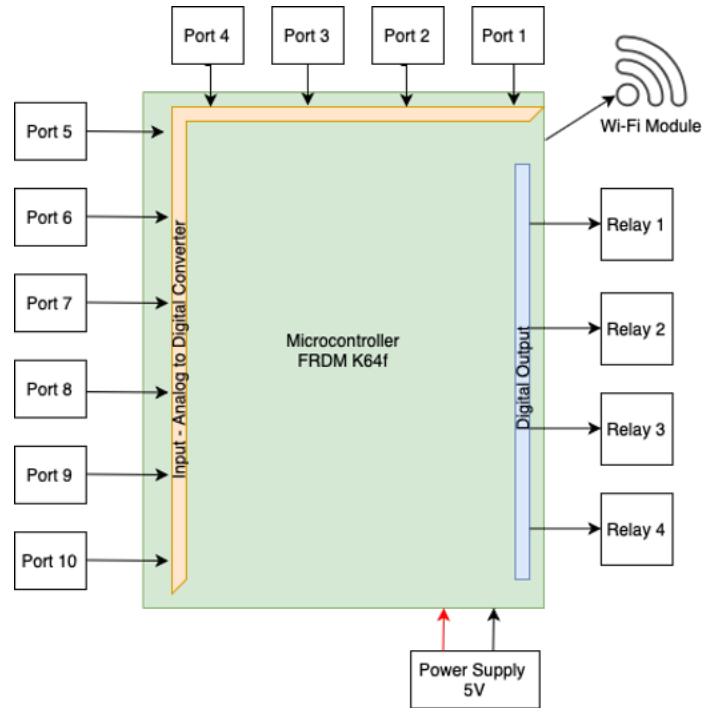


Figure 4.11. Microcontroller Circuit

4.1.5 SD Card

The system can log at least 48 hours of data, which would add up to around 10 MB. Any SD card with more than 12 MB should work. The model used in this development is an 8GB microSDHC UHS-I by Sandisk. It connects to the MicroSD slot on the K64F board. Figure 4.12 shows the SD card used to mount on the microcontroller device and the adapter to connect the SD card to the computer.

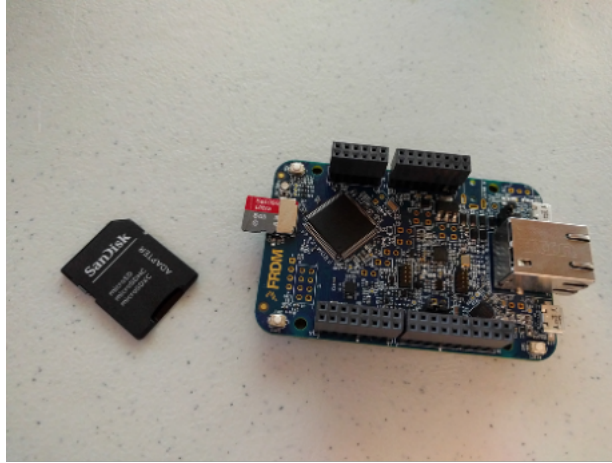


Figure 4.12. SD Card Connection

4.1.6 Wi-Fi

Wi-Fi module (ESP 8266) is connected to the microcontroller using 4 wires, Power (3.3V), Ground, Transmit, and Receive, as shown in Figure 4.13. Transmit wire is used to transmit data from the microcontroller to the module. Receive wire is used to receive a response from the cloud and convey the same to the microcontroller. Transmit and Receive pins use UART (Universal Asynchronous Receiver Transmitter) protocol. The module connects to the internet and is used to send data to the Cloud.

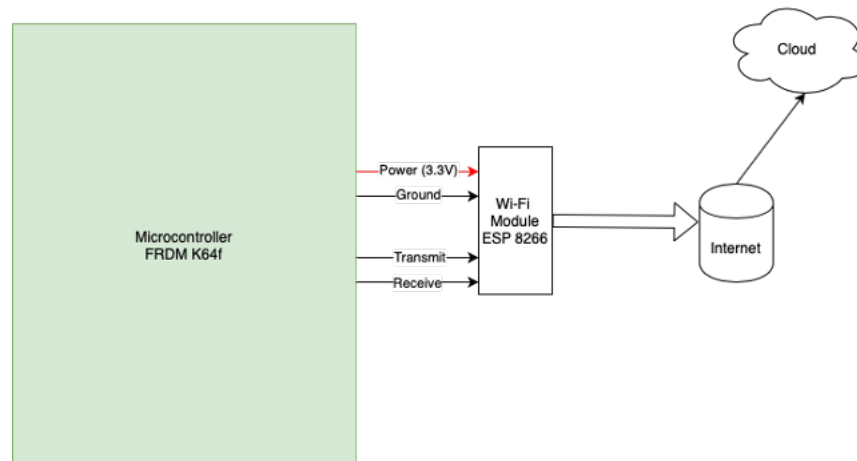


Figure 4.13. Wireless Connection

4.1.7 PCB

The PCB design shown in Figure 4.14 was designed with AutoDesk Eagle, version 8.1.1. The design had 10 ports and 4 actuator circuits. The PCB design has been designed to have FRDM K64F microcontroller and ESP8266 boards be plugged directly into the board. Using this design, five PCBs were manufactured for testing. The component list is in Appendix B.

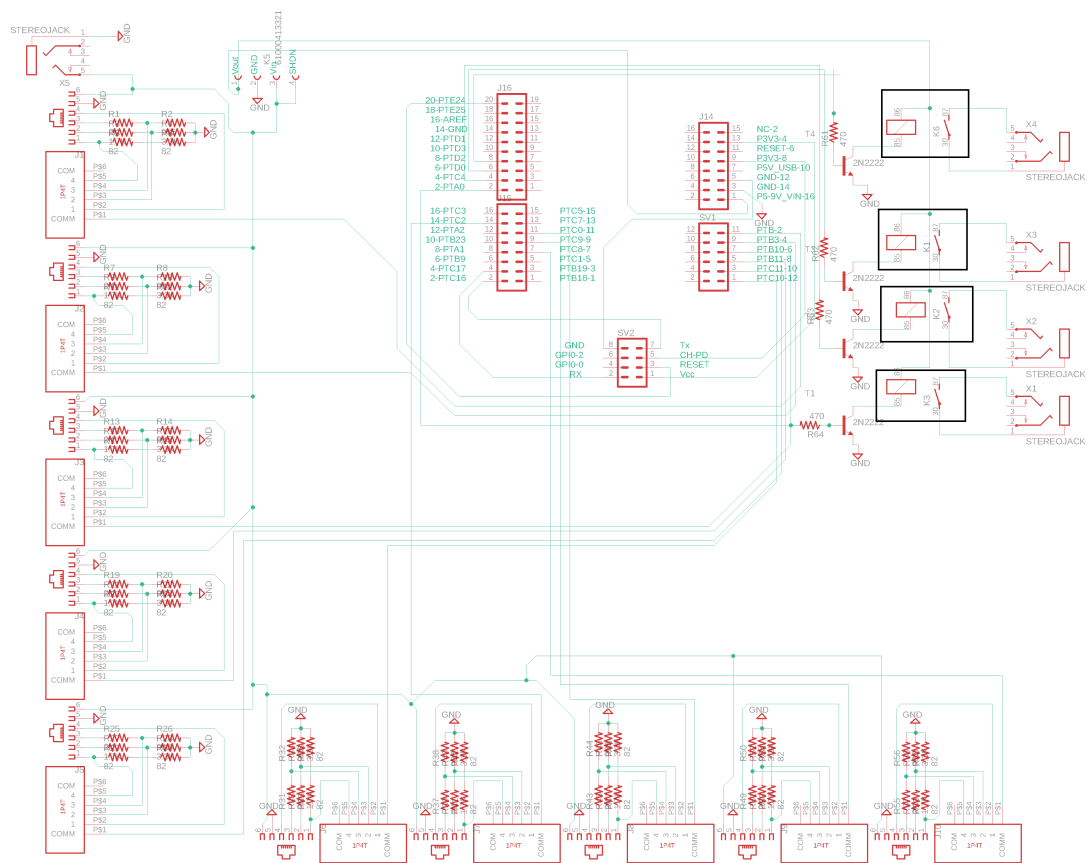


Figure 4.14. PCB Layout

4.2 Embedded Software

Software is needed to instruct the hardware to perform the desired tasks. The software includes acquiring data, pushing data to the cloud, actuating the equipment, performing fail-safe mechanisms, etc. Figure 4.15 shows the overall block diagram of the software. It consists of various functions, which will be explained individually in the following subsections.

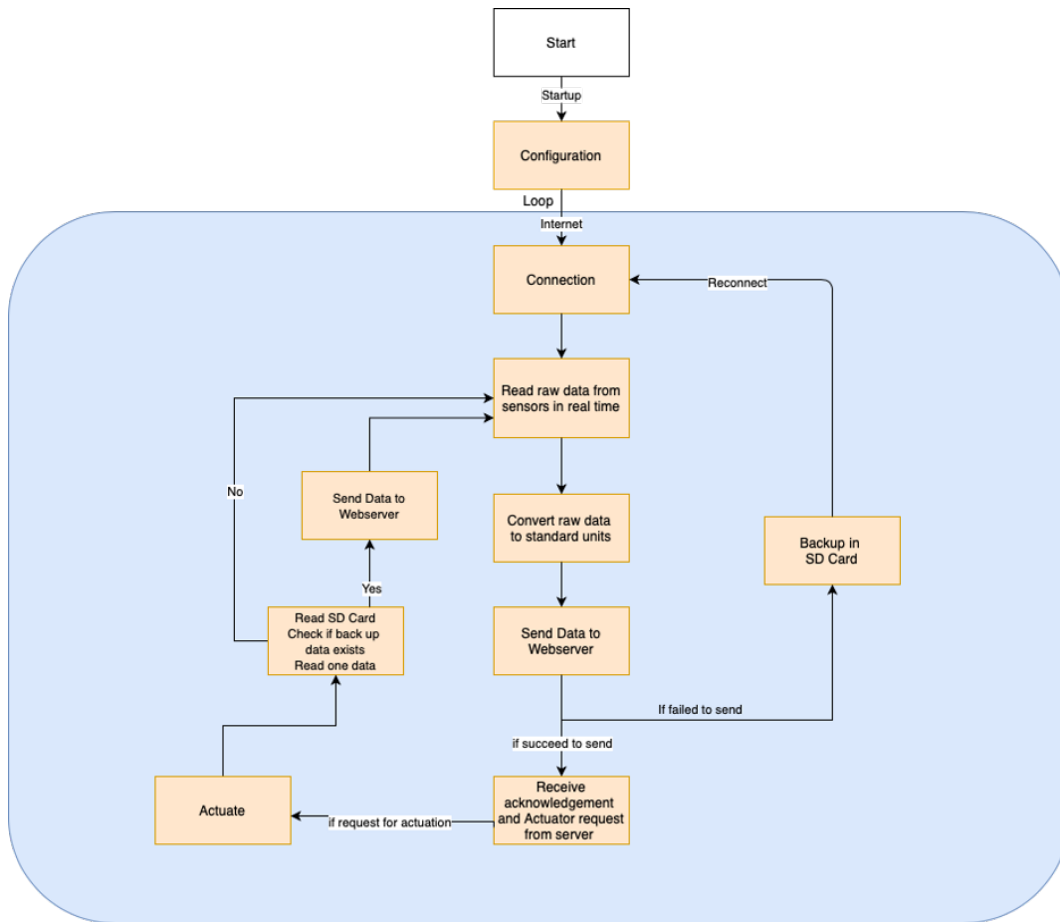


Figure 4.15. Software Block Diagram

The software consists of the following functions

1. Configuration
2. Connection
3. Read raw data from sensors in real-time and convert to standard units
4. Send Data to Webserver
5. Actuation
6. Backup to SD Card

4.2.1 Configuration

On startup, the software will go into setup mode. In this mode, it first parses the configuration file from the SD card named “IAC Config.txt” and then enters into configuration mode. The configuration file is in a csv format. The configuration file is divided into two sections. The first section is the system information. System info contains the information to connect to the Wi-Fi and Board ID. The lines starting with the ‘#’ symbol indicates a comment. These lines are not executed by the firmware and are for information purpose only.

```
# system info.  
# Board Number:Wifi-SSID:Wifi-Password  
Board-3:Monil's iPhone:lebolebo
```

The second section is the sensor connection information. This section contains the set of streams that the system will run. It contains the port ID with the associated sensor ID and switch positions. This helps the microcontroller determine which sensor is attached to what port number, and hence appropriately convert raw values using the appropriate multiplication factor. The multiplication factors are hardcoded in microcontroller code and is the value used for multiplying the raw data from the sensor (ADC value) to convert into standard units. If the sensor to be used is selected from the list in configuration file, microcontroller code does not need to be changed. However, if new sensor is to be added

which is not present in the list, microcontroller code will need to be changed accordingly. The multiplication factors have been calibrated manually. For example, a current sensor was attached to a heat gun. At the same time a digital multimeter was also attached to the heat gun. The current sensor logged analog values in the microcontroller. Concurrently the reading from the digital multimeter was read and compare to the analog output of the current sensor. Based on several such readings, a multiplication factor is determined which would give the correct reading in standard unit (Amperes). The configuration file also contains information regarding which Relay controls what equipment. This version of firmware does not send the relay information to the server. However, future update of the firmware may support sending the relay information to the server. Note: For attaching the current sensor, this system uses a line splitter which has 2 ports which are 1x and 10x. 1x gives actual reading where as 10x gives ten times the actual reading. Current sensor may be attached to any of the two ports. The multiplication factors are calibrated to give actual readings (1x). However, if 1x port does not work (out of lower range), 10x port can be used and the current value can be divided by 10 on the front end (web) program. Microcontroller code need not be changed.

```

# Sensor info

# Sensor-id , type, start-range, end-range, conversion circuit, Multiplication factor, Output unit.

SensorID = 0, No Sensor, 0, 0, none, no multiplication factor, no unit.
SensorID = 1, CO2, 0, 2000, 10 Vdc, 2000, ppm.
SensorID = 2, Current, 0, 100, 2.5 Vdc, 140, Ampere.
SensorID = 3, Current, 0, 20, 2.5 Vdc, 27.85, Ampere.
SensorID = 4, Temperature, 0, 50, 5 Vdc, 50, Degree Celcius.
SensorID = 5, Temperature, 0, 50, 10 Vdc, 50, Degree Celcius.
SensorID = 6, Pressure, 0, 200, 5 Vdc, 240, Psig.
*SensorID = 7, Humidity, 0, 100, 5Vdc, 100, Percentage.
*SensorID = 8, Compressed Air Flow, 0, 100, 20mA, 100, Percentage.

# Slide Switch Info

# Position : Conversion Circuit
Swt_Pos = 1 : 0 - 5 Vdc
Swt_Pos = 2 : 0 - 2.5 Vdc
Swt_Pos = 3 : 0 - 20 mA
Swt_Pos = 4 : 0 - 10 Vdc

# Port info.

# Port : SensorID : Swt_Pos

Port_1 : 0 : 0
Port_2 : 5 : 4
Port_3 : 3 : 2
Port_4 : 4 : 1
Port_5 : 1 : 4
Port_6 : 6 : 1
Port_7 : 1 : 4
Port_8 : 0 : 0
Port_9 : 2 : 2
Port_10 : 0 : 0

Relay_1: Air Compressor
Relay_2: Light Bulb
Relay_3: Valve Actuator
Relay_4: Motor

```

In this mode, the firmware extracts all information from the configuration file to get sensor information, port information, Board ID, and Wi-Fi credentials and store them into appropriate variables. The microcontroller is rebooted every time when the internet fails or when sensors are changed. When the microcontroller reboots, it sends data from the configuration file to the webserver. The web server stores the information in the database. It is advised to turn off the microcontroller before changing any sensors or configurations. However, if the microcontroller reboots often, such anomaly can be detected in the database table. Once the setup is complete, it proceeds to the next function, Connection.

4.2.2 Connection

Once the configuration process starts, it uses the config file information to connect to the mentioned Wi-Fi connection using the credentials mentioned in the config file. In this step, as shown in Figure 4.16, the firmware establishes the internet connection using the credentials

obtained from the configuration file and using the Wi-Fi module. Once the connection is established, it moves to the next functions, Data Read. Best upload speed for this system is 1 second per sensor reading.

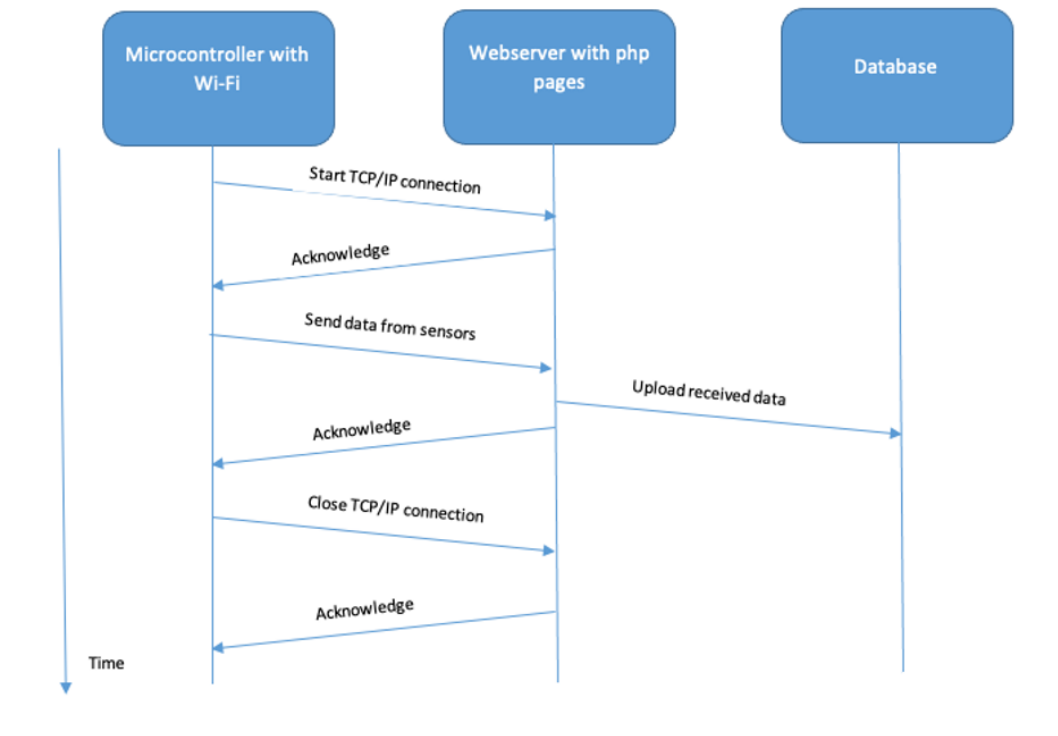


Figure 4.16. Cloud Connection and Data Upload Sequence Diagram

4.2.3 Read Raw Data from Sensors in Real-time and Convert to Standard Units

This function reads real-time data from all sensors. The configuration file has information regarding which of the ports have sensors attached and which sensors are attached. Using that information, the firmware creates a lookup table in the microcontroller memory itself. The function then reads values from the ports with sensors connected. It then converts the raw values to standard units by multiplying with the multiplication factors stored in a lookup table. This function reads raw sensor values and converts to standard units. “Wait” command is used to obtain desired delay. Wait command is executed when backup data does not exist. If backup data exists, wait command will not be executed and that time will be used to send one reading of backup data to the server. Once the data is converted to

standard units, the program moves to the next function, which sends the data to the cloud. The standard units are defined in the configuration file as follows:

1. Celsius for Temperature
2. PPM for CO2
3. PSIG for pressure
4. Ampere for current

Also, to make the system fail-safe, the following errors were observed to handle by the firmware:

1. ERROR
2. Busy
3. 404 Not Found

If the firmware detects these errors as a server response, it will attempt to reconnect to the internet. This ensures internet failures and server errors can be handled.

4.2.4 Send Data to Webserver

This function is responsible for sending the read data to the cloud. This function prepares a TCP/IP frame that contains the server address, packet size, board ID, port ID and the sensor value read in the Data Read function. As shown in Figure 4.17, it then sends the frame to the server and waits for the response. The detailed description of the frame is explained in Chapter 6. Microcontroller can get one of the two responses,

1. Relay status: This response indicates that the server has received the data. It sends the on/off status of the four relays attached to that board from which data was sent. If any change in the relay status is detected, the program goes to the Actuator function. The information regarding which relay controls which appliance can be found in an information table in the database called Appliance. The last 4 lines in the config file will define equipment connected to the relay.

2. Error: This response indicates that the data has not been sent to the cloud successfully. In this case, the program goes to the Offline logging function and then attempts to reconnect to the internet using the Connection function.

Figure 4.17 shows the interaction diagram between the client/Microcontroller and the server. The microcontroller prepares a packet and requests to open a TCP/IP connection using the ESP8266 Wi-Fi module. The Wi-Fi module sends out the request, and the server responds back to the module. Then the microcontroller sends the data size and then the data. Once the one packet data is sent, the connection is closed.

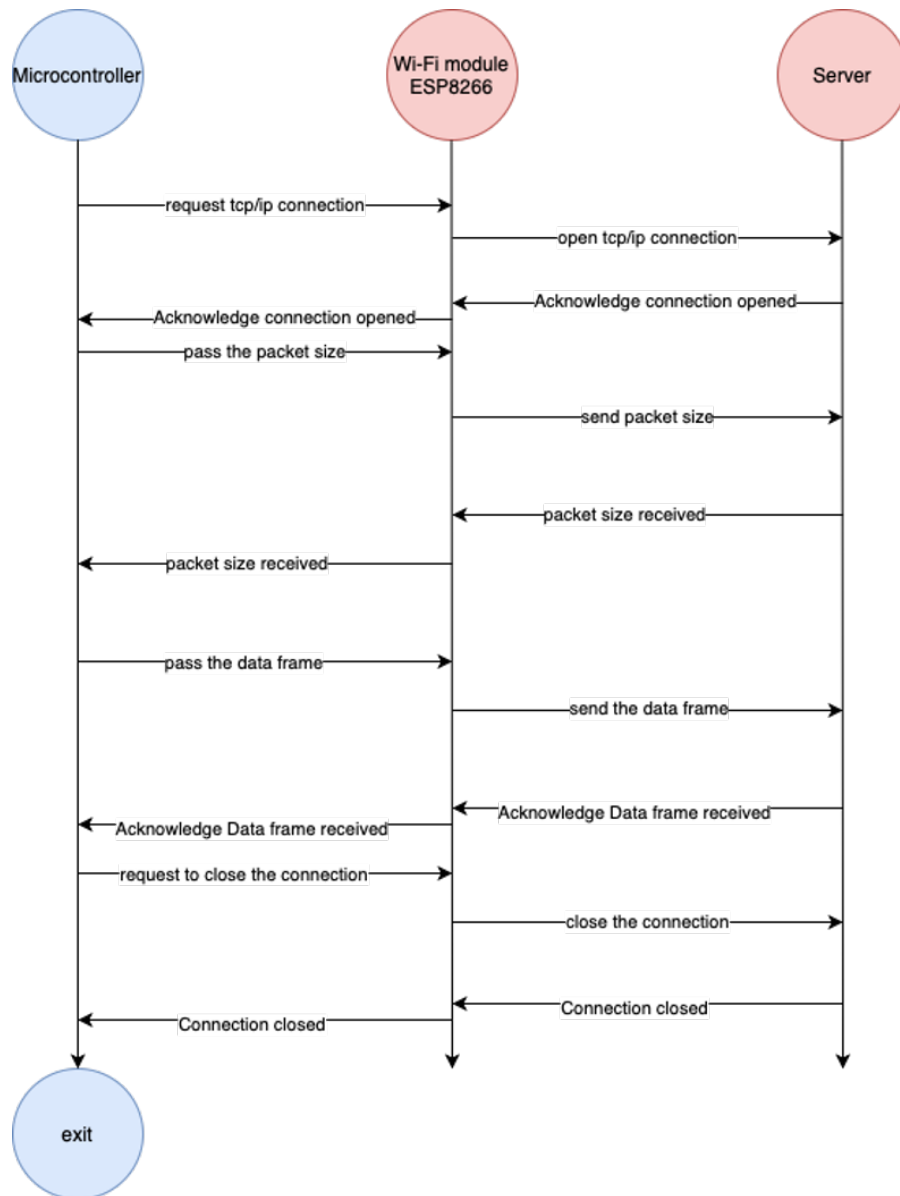


Figure 4.17. Client-Server Messaging

Figure 4.18 shows the microcontroller code's snippet, which is used to send data to the server. The variable "sz1" uses the function "strlen" to compute the length of the frame. The command "SendCMD" sends the size of the frame containing the values to the webserver. The command "getreply" receives the response from the server. The variable "text" stores the frame containing data to be sent. The command "SendCMD" is used again to send the actual frame containing data.

```

void TCP_DataSend(){
    pc.printf("\n----- Start TCP_IP Connection with WAMP ----- \r\n");
    strcpy(snd, "AT+CIPSTART=4,\"TCP\", \"134.68.70.220\", 80\r\n");
    SendCMD();
    timeout=0.1;
    getreply();
    pc.printf(buf);
    pc.printf("\n----- Set TCP Data frame ----- \r\n");
    memset(text, '\0', sizeof(text));

    char buffer[32];
    sprintf(text, "GET /Sensor_readings.php?Board_ID=%s&Port_ID=%s&Value=%0.2f&MCTime=%u\r\n", Board_specs[4], port[port_number], port_value, (unsigned int)seconds);
    int sz1 = strlen(text);
    pc.printf("\n\n\r\n", sz1);
    sprintf(snd, "AT+CIPSEND=4,%d\r\n", sz1);
    SendCMD();
    pc.printf("\r\nGetting Reply\r\n\r\n");
    timeout=0.1;
    getreply();
    pc.printf(buf);
    if(strstr(buf, "ERROR")!=0 || strstr(buf, "busy")!=0 || strstr(buf, "404 Not Found")!=0)
    {
        ESPconfig();
    }
    pc.printf("\n----- Send Data frame ----- \r\n");
    strcpy(snd, text);
    pc.printf(snd);
    pc.printf("\n\n Getting Reply\r\n\r\n");
    SendCMD();
    timeout=0.1;
    getreply();
    pc.printf(buf);
}

```

Figure 4.18. Microcontroller Code to Send Data to the Web Server

4.2.5 Actuator

This function is used to actuate relays on the boards. Once the data frame is sent by the microcontroller to the web-server, as shown in Figure 4.19, the command “getreply” in the microcontroller code is used to get a response from the webserver. The web server responds with the on/off status of each relay. The snippet in Figure 4.19 is the microcontroller code, which checks the status of each relay received by the microcontroller from the server.


```

pc.printf("\n----- Send Data frame ----- \r\n");
strcpy(snd, text);
pc.printf(snd);
pc.printf("\n\n Getting Reply \r\n\r\n");
SendCMD();
timeout=0.1;
getreply();
pc.printf(buf);
if(strstr(buf, "Relay 1=on")!=0)
    Relay1=1;
else if(strstr(buf, "Relay 1=off")!=0)
    Relay1=0;

if(strstr(buf, "Relay 2=on")!=0)
    Relay2=1;
else if(strstr(buf, "Relay 2=off")!=0)
    Relay2=0;

if(strstr(buf, "Relay 3=on")!=0)
    Relay3=1;
else if(strstr(buf, "Relay 3=off")!=0)
    Relay3=0;

if(strstr(buf, "Relay 4=on")!=0)
    Relay4=1;
else if(strstr(buf, "Relay 4=off")!=0)
    Relay4=0;

```

Figure 4.19. Actuator Request Handling

The microcontroller receives the command to the relay from the cloud upon acknowledging the successful reception of the sensor data by the server, as shown in Figure 4.20. The response of the status is such as “Relay 1 = on”. If there is a change from the actuator’s current state, the microcontroller performs the requested actuation. After the actuation, the program goes back to the Data read function.

```

----- Start TCP_IP Connection with WAMP -----
AT+CIPSTART=4,"TCP","134.68.70.220",80
4,CONNECT

OK

----- Set TCP Data frame -----

91

Getting Reply

91

OK
>
----- Send Data frame -----
GET /Sensor_readings.php?Board_ID=IAC_Board1&Port_ID=Port_9&Value=13.90&MCTime=1604877633

Getting Reply

Recv 91 bytes

SEND OK

+IPD,4,68: Hello, World!
Relay 1=off
Relay 2=off
Relay 3=off
Relay 4=off
4,CLOSED

----- Close TCP/IP Connection -----
AT+CIPCLOSE=5

OK

```

Figure 4.20. Server Response with Relay Status

4.2.6 Backup to SD Card

Figure 4.21 shows how the data is backed up into an SD card. This function is triggered when the Data Send function gets an error while sending the data to the cloud. If data fails to send, the firmware stores the converted to standard units data into an SD card on the microcontroller. It then attempts to reconnect to the internet using the connect function. Once the connection is established, the Data Send function sends the data logged into the SD card while the system was offline. The data is stored in the format as Timestamp, Board ID, Port ID, and Value. This stored data are sent to the cloud. This helps in the quick upload of data to the server once the connection is established.

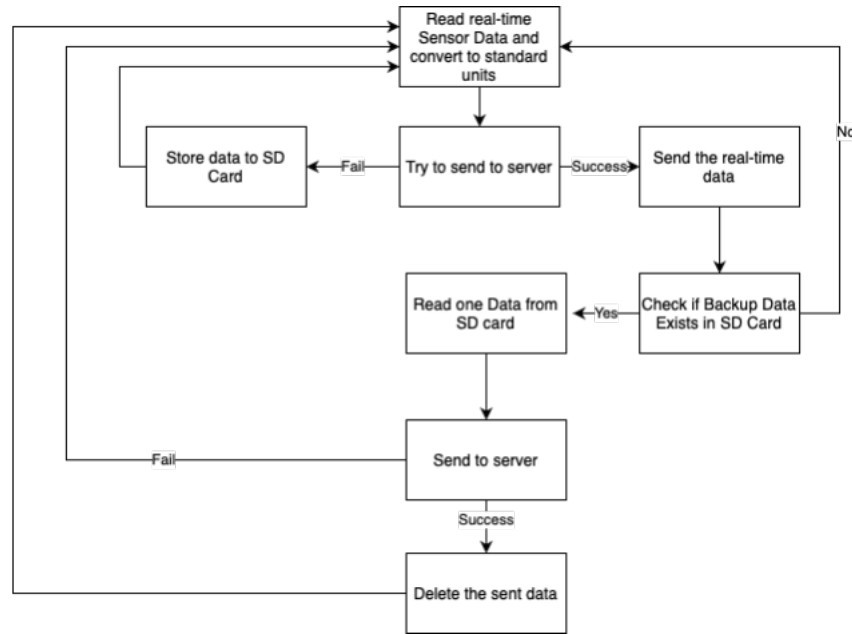


Figure 4.21. Backup to SD Card Activity Diagram

The function shown in Figure 4.22 stores the data into SD card in a file named Backup.txt. When sending data to server fails, the data in character array format is passed to this function.

- The first line is function name and argument. The function receives a character array as an argument called “text”.
- The second line of the function ensures that the file is closed using “fclose” command.
- Using “if” statement in the fourth line, the program then attempts to open the file using “fopen” command in append mode “a”. If the file cannot be opened, the pointer returns “NULL”.
 - If the pointer returns NULL, the program will close the file using “fclose” command in the 6th line.
 - In the 7th line, the program will attempt to create a “Backup.txt” file in SD card using “fopen” command in write “w” mode.
 - After creating file in write mode, the file is closed using “fclose” command in 8th line.

- Again, the file is opened in append mode outside if block on 11th line.
- The data received by the function in character array “text” is appended to this file using fprintf command on 12th line.
- Once the data is appended in the “Backup.txt” file, the file is closed on 14th line using fclose command.

Note: When file is opened in write mode “w”, the program will start writing content from the beginning of the file. However, when a file is opened in append mode “a”, the data will be appended to the existing file content.

```

1 void Backup(char text[])
2 {
3     fclose(fp);
4     pc.printf("Appending data to data file \r\n");
5     if((fp=fopen("/sd/Backup.txt","a")) == NULL){
6         pc.printf("Could not open file");
7         fclose(fp);
8         fp=fopen("/sd/Backup.txt","w");
9         fclose(fp);
10    }
11    fclose(fp);
12    fp = fopen("/sd/Backup.txt", "a");
13    fprintf(fp, "%s", text);
14    pc.printf("\n\n%s\n\n", text);
15    fclose(fp);
16 }

```

Figure 4.22. Backup to SD Card

When the microcontroller is able to send data to the server, the Function shown in Figure 4.23 will check if backup data exists.

- In the 4th line, the function will open file using “fopen” command in read “r” mode.
- In the 5th line, the function then reads data from the file using “fgets” command and checks if the length of the data is less than 10 characters. It can be any number greater than 1 and less than 90. Actual length of the one data frame is around 90 characters (bytes) but this function has not been tested using 1. The program checks the length of the read string to ensure that Backup data exists. If the length is less than 10 characters, it means that backup data does not exist.

- If the backup data does not exist, the program will close the file using `fclose` command on 7th line.
- Wait command is executed on line 8. This ensures that 1.5 second of delay is obtained before exiting this function and reading next sensor value.
- If backup data exists the program in else block will be executed from 10th line.
 - The program opens the Backup.txt file using `fopen` command on line 13.
 - Using the `fgets` command on line 14, the program reads one line from the Backup.txt file.
 - The program will then send the read data from the file to the server using “Senddta” function on line 17. Note, each line has one reading, therefore this function will read the first line and send it to the server.
 - Once the data is passed to Senddta function, the file is closed on line 18 using `fclose` command.

```

1 void SendBackupData()
2 {
3     char bckp[1024];
4     fp = fopen("/sd/Backup.txt", "r");
5     if(strlen(fgets(bckp, sizeof(bckp), fp)) < 10){
6         pc.printf("No Backup Data");
7         fclose(fp);
8         wait(1.5);
9     }
10    else
11    {
12        fclose(fp);
13        fp = fopen("/sd/Backup.txt", "r");
14        fgets(bckp, sizeof(bckp), fp);
15        pc.printf("%s", bckp);
16        fclose(fp);
17        Senddta(bckp);
18        fclose(fp);

```

Figure 4.23. Send Backup Data to Server

The “Senddta” function gets character array as an argument which is the data to be sent to the server. The function establishes connection to the server, sends the data (from character array) to the server, and closes the connection. The function is shown in Figure

4.24. Note: In this function, sending failures are not handled and may be implemented in future development.

```
void Senddata(char bckp[])
{
    if(strlen(bckp)>1){
        pc.printf("\n----- Start TCP_IP Connection with WAMP -----<div data-bbox="358 500 636 516" data-label="Caption">


Figure 4.24. Senddata Function


```

Figure 4.25 shows the sample of the “Backup.txt” file. My approach stores the data in format which is used to send to the server(see Section 6.4.2 for detailed information). Saving the data in this format reduces the time required to prepare the frame to send to the server. For example, the first line stores data for IAC_Board1, Port_2 whose value is 10.87. The MCTime is the Unix/epoch time format used in computer engineering field.

```
GET /Sensor_readings.php?Board_ID=IAC_Board1&Port_ID=Port_2&Value=10.87&MCTime=1611192876
GET /Sensor_readings.php?Board_ID=IAC_Board1&Port_ID=Port_5&Value=320.81&MCTime=1611192878
GET /Sensor_readings.php?Board_ID=IAC_Board1&Port_ID=Port_2&Value=17.46&MCTime=1611192879
GET /Sensor_readings.php?Board_ID=IAC_Board1&Port_ID=Port_3&Value=283.76&MCTime=1611192880
GET /Sensor_readings.php?Board_ID=IAC_Board1&Port_ID=Port_4&Value=6.27&MCTime=1611192882
GET /Sensor_readings.php?Board_ID=IAC_Board1&Port_ID=Port_2&Value=12.91&MCTime=1611192885
```

Figure 4.25. Screen Shot of how Backup Data is Saved

The code shown in Figure 4.26 shows the process of deleting the backup data which is sent to the server. The data sent from the backup file is then deleted as follows:

1. Open “Backup.txt” file and another empty “temp.txt” file.
2. Read data from “Backup.txt” file line by line and store in “temp.txt” except for the first line (since we already sent that data).
3. Once all data, except first line, is copied to “temp.txt”, delete “Backup.txt” file.
4. Rename “temp.txt” file as “Backup.txt”. (This is done because we want to delete sent data. “Temp.txt” file contains all data of “Backup.txt” file except the one that is already sent).

```

fclose(fp);
int ctr = 0;
memset(bckp, '\0', sizeof(bckp));
//char c = fgetc(fp);
char copy[100];
//wait(3);
fp=fopen("/sd/Backup.txt", "r");
fp2 = fopen("/sd/temp.txt", "w");
while ((fgets(copy, sizeof(copy), fp)) != NULL)
{
    ctr++;
    if(ctr==1)
        pc.printf("Skip first line\r\n");
    else
        fputs(copy, fp2);
}
//sprintf(copy, "\0");
//sprintf(bckp, "\0");
//wait(2);

fclose(fp);
fclose(fp2);
remove("/sd/Backup.txt");
file_rename("/sd/temp.txt", "/sd/Backup.txt");
memset(copy, '\0', sizeof(copy));
}

```

Figure 4.26. Delete the Data Sent to Server

5. DATABASE

The primary goal of energy management is to utilize energy efficiently. This means reducing wasted energy. To save energy, one needs to analyze the usage. As a result of this analysis, one can determine the usage pattern and hence provide suggestions on how to save energy. Energy analysis requires a lot of historical data to build a hypothesis on usage patterns. The database is used to structure and organize such data to use that data to realize the energy usage trend and predict future energy usage. Collaborating IoT with a database helps to collect and store data into a database from the sensors. The purpose of the Internet of things is to collect and exchange data to understand a system better. This is accomplished by using network-connected devices like sensors that can measure and collect relevant data. An incredible amount of data is generated over time. To organize the collected data, the selection and design of the database are critical. The database service should handle and process large amounts of data efficiently and provide high availability and excellent performance.

5.1 Database Design

As mentioned earlier, to understand the energy trend, a huge amount of data is required. For example, several years of energy consumption data is needed for analyzing energy usage over different seasons. This system makes use of IoT to collect this data regularly. IoT helps collect data at specified intervals (e.g., Every 5 seconds) and stores it into a database. Using IoT makes sure that data is saved in the cloud and does not run out of local memory like data loggers. Database design is critical for a successful IoT system. It is the element, which stores data

1. Information Tables - These tables store information about the company, such as the board ID's the company is using, sensors connected, etc. Most of the information in these tables is static and will be required to change if there is a technical change in the board or companies.
2. Operating Tables - In these tables, the data gathered from the microcontroller and sensors is stored, and the relay status helps to control onboard equipment remotely (turn on/off).
3. Application Function Tables - These tables are created based on the application function supported. Using the data from information and operating tables, the application function developer might want to produce meaningful results and store them in the database.

5.1.1 Information Tables

Figure 5-1 shows all tables designed for this project and the Entity-Relationship (E-R) diagrams, which show how the tables are linked to one another to extract useful information required by the company. As shown in diagram 5-1, in each table, the first row represents the key. The arrows connecting different table is cardinality. Cardinality in this context means the entity occurs in more than one table. For example, company id occurs in company table, contact table, board table, zone table and company appliance table. Using company id, information can be extracted from any of these tables.

The tables in the static information part of the database are as follows:

1. Appliance – This table contains information regarding appliances or equipment that use a significant amount of energy. The key for this table is appliance ID
2. Company – This table contains necessary information like company ID as a key, company name, and company address.
3. Contact – This table contains information about the company, such as contact ID as a key, company name, contact person, contact number, email, company ID, etc.

4. Zone – Manufacturing plant may be divided into several zones. This table contains information about the zones, such as zone ID as a key, company ID, and address.
5. Company Appliance – This relation table contains company ID as a key, Zone ID, and appliance ID. This helps to determine which appliance belongs to a particular company and which zone it belongs to.
6. Sensor – This table provides information regarding the sensors used to monitor appliance energy consumption. This table lists the sensor ID as key, name, model, sensor output range and electric signal ranges.
7. Board – Sensor box is the hardware part of the system, which needs to be attached to appliances. There can be several sensor boxes in the manufacturing plant. Therefore, it needs identification. This table provides information like board ID as a key, company ID, which shows which company the sensor box is installed, and zone ID, which provides information regarding which zone of the plant is sensor box installed in.
8. Appliance Port – Each sensor box has ten ports. This table provides information regarding which sensor is connected to which board (sensor box) and what appliance. The key for this table is Port ID.

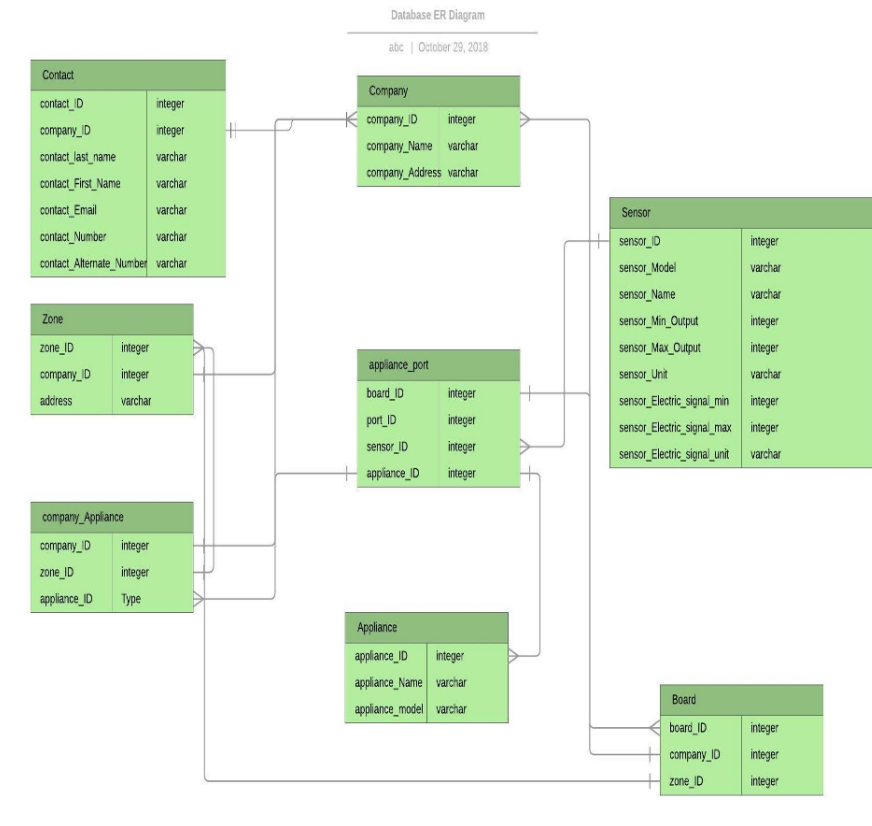


Figure 5.1. Information Table E-R Diagram

The information tables hold all the essential information as the design depicted in the E-R diagram. Figure 5.2 is the Company table, which holds information such as Company name and Address.

company_ID	company_Name	company_Address
1	IAC	799 W Michigan St IN 46202
2	IUPUI	723 W Michigan St IN 46202

Figure 5.2. Database Table Containing Information Regarding Companies

Figure 5.3 is the Contact table consisting of information about the company contact person, contact number, email, etc.

contact_ID	comapany_ID	contact_last_name	contact_first_na...	contact_email	contact_numb...	contact_alt_numb...
1	1	Chen	Jie	jchen3@iupui.edu	317-274-4567	NULL
2	2	Chien	Stanley	schien@iupui.edu	317-272-6789	NULL
3	1	Wu	Allen	wudach@iupui.edu	317-274-1221	NULL

Figure 5.3. Database Table Containing Contact Information

Figure 5.4 is the Appliance table, which holds information regarding the appliances used at the company. Various companies can have similar appliances.

appliance_ID	appliance_Name	appliance_model
1	Air Compressor	DeWalt
2	Motor	Bosch
3	Light Bulb	Phillips
4	Actuator Valve	Bosch

Figure 5.4. Database Table for Appliance

Figure 5.5 is the Appliance Port table, which holds information regarding sensors connected to the particular appliance. This table also provides information on which sensor is connected to which board and which appliance it is connected to.

port_ID	board_ID	sensor_ID	appliance_ID
9	IAC_Board3 2	1	
4	IAC_Board3 4	3	
6	IAC_Board3 6	1	

Figure 5.5. Database Table for Information on Appliance Port

Figure 5.6 is the Board table, which contains information on which board belongs to which company and in which Zone of the company it is installed in.

Board_ID	company_ID	zone_ID
1	1	317
2	1	313
3	2	163
4	1	317

Figure 5.6. Database Table Containing Information Regarding Board

Figure 5.7 is the Company Appliance table, which provides information about a particular appliance installed in a specific company zone.

appliance_ID	company_ID	zone_ID
1	2	163
2	2	163
3	1	313

Figure 5.7. Database Table Containing Information about Company Appliance

The database table in Figure 5.8 is the Sensors table, which holds information on different kinds of sensors compatible with the system. It also provides specifications like units, electrical signal, model, etc.

sensor_ID	sensor_Model	sensor_name	sensor_min_output	sensor_max_output	sensor_Unit	sensor_Electric_signal_min	sensor_Electric_signal_max	sensor_Electric_signal_u...
1	C7232A1008	CO2	0	2000	ppm	0	10	Volts
2	CTV-C	Current	0	100	Ampere	0	2.5	Volts
3	CTV-E	Current	0	600	Ampere	0	2.5	Volts
4	TOAV22	Temperature	0	50	Celsius	0	5	Volts
5	TOAV22	Temperature	0	50	Celsius	0	10	Volts
6	T-ASH-G1-200	Pressure	0	200	psig	0	5	Volts
7	HE-67S3-0N...	Humidity	0	100	Percentage	0	5	Volts
8	CDI-5200	Comprese...	0	100	Percentage	4	20	milli Amperes

Figure 5.8. Database Table Containing Information about Sensors

The database table in Figure 5.9 is the Zone table, which provides addresses for each zone of a company since a large company may have more than one zone and address.

zone_ID	company_ID	Address
163	2	723 W Michigan St IN 46202
313	1	799 W Michigan St IN 46202
317	1	799 W Michigan St IN 46202

Figure 5.9. Database Table Containing Information Regarding Zone

5.1.2 Operating Table

Figure 5.10 shows the E-R Diagram for operating tables, which are Board Startup Specifications and Sensor Readings. Here, the Board Startup Specifications contains information regarding the Board ID, the boot time of the microcontroller, and the sensors connected to each board's port. The sensor readings table stores the sensor readings from the microcontroller. Each row contains a timestamp, Board ID, Port ID, and value. Timestamp signifies precisely at what date and time the value was read. The Board ID provides information on which Board is the sensor reading coming from. Port ID provides information on which port of the board is the value coming from. the Value is the actual sensor Value. Using board

ID from Sensor Readings table, it can be determined from the Board Startup Specifications table as to which port is connected to which sensor. The diagram also shows the relation of the operating tables to information tables. The board ID enables one to find which company and zone the board is installed. Conversely, using a given company's board ID makes it easy to find which sensors are connected to that board.

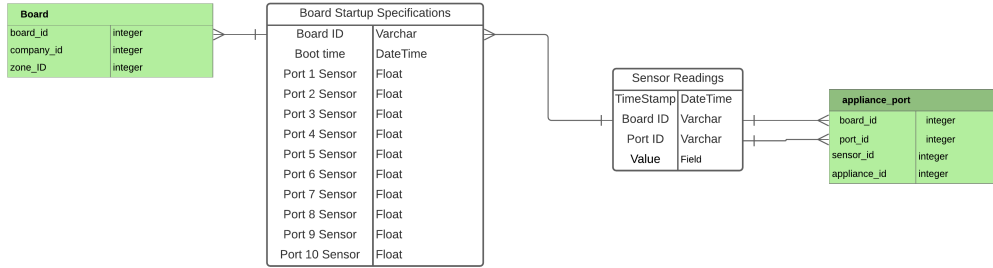


Figure 5.10. Operating Table E-R Diagram

The database table in Figure 5.11 is an implementation of the operating table called Board startup specifications. It logs the microcontroller boot time and the information about the kind of sensor attached to each port. This table is updated in case of microcontroller reboot when sensors are changed, and data from the new configuration file is uploaded to the server.

Board-ID	BootTime	WiFi-IP	Port1-Sensor	Port2-Sensor	Port3-Sensor	Port4-Sensor	Port5-Sensor	Port6-Sensor	Port7-Sensor	Port8-Sensor	Port9-Sensor	Port10-Sensor
Board-3	2019-04-05 17:48:00	192.168.1.4	Null	Temperature(0-50)Degree-Celsius	Current(0-600)Amps	Temperature(0-50)Degree-Celsius	CarbonDioxide(ppm)	Pressure(PSIG)	CarbonDioxide(ppm)	Null	Current(0-100)Amps	Null
Board-4	2019-02-21 13:24:40	192.168.1.8	Current(0-100)Amps	Current(0-100)Amps	Current(0-600)Amps	Current(0-600)Amps	Null	Null	Null	Null	Null	Null
Board-3	2019-02-12 10:49:01	192.168.1.5	Null	Temperature(0-50)Degree-Celsius	Current(0-600)Amps	Temperature(0-50)Degree-Celsius	CarbonDioxide(ppm)	Pressure(PSIG)	CarbonDioxide(ppm)	Null	Current(0-100)Amps	Null
Board-2	2019-01-24 15:28:19	192.168.1.5	Null	Temperature(0-50)Degree-Celsius	Current(0-600)Amps	Temperature(0-50)Degree-Celsius	CarbonDioxide(ppm)	Pressure(PSIG)	CarbonDioxide(ppm)	Pressure(PSIG)	Current(0-100)Amps	Null
Board-3	2019-01-16 17:13:15	192.168.1.6	Null	Temperature(0-50)Degree-Celsius	Current(0-600)Amps	Temperature(0-50)Degree-Celsius	CarbonDioxide(ppm)	Pressure(PSIG)	CarbonDioxide(ppm)	Null	Current(0-100)Amps	Null
Board-4	2019-01-11 11:45:14	192.168.1.6	Null	Temperature(0-50)Degree-Celsius	Current(0-600)Amps	Temperature(0-50)Degree-Celsius	CarbonDioxide(ppm)	Pressure(PSIG)	CarbonDioxide(ppm)	Null	Current(0-100)Amps	Null
Board-3	2018-11-28 14:07:12	192.168.1.6	Null	Temperature(0-50)Degree-Celsius	Current(0-600)Amps	Temperature(0-50)Degree-Celsius	CarbonDioxide(ppm)	Pressure(PSIG)	CarbonDioxide(ppm)	Null	Current(0-100)Amps	Null
Board-4	2018-11-28 14:38:30	192.168.1.6	Null	Temperature(0-50)Degree-Celsius	Current(0-600)Amps	Temperature(0-50)Degree-Celsius	CarbonDioxide(ppm)	Pressure(PSIG)	CarbonDioxide(ppm)	Null	Current(0-100)Amps	Null
Board-3	2018-11-19 14:41:50	192.168.1.6	Null	Temperature(0-50)Degree-Celsius	Current(0-600)Amps	Temperature(0-50)Degree-Celsius	CarbonDioxide(ppm)	Pressure(PSIG)	CarbonDioxide(ppm)	Null	Current(0-100)Amps	Null
Board-2	2018-04-09 22:35:22	192.168.4...	Null	Temperature(0-50)Degree-Celsius	Current(0-600)Amps	Temperature(0-50)Degree-Celsius	CarbonDioxide(ppm)	Pressure(PSIG)	CarbonDioxide(ppm)	Null	Current(0-100)Amps	Null
Board-2	2018-03-04 17:45:05	192.168.4...	Null	Temperature(0-50)Degree-Celsius	Current(0-600)Amps	Temperature(0-50)Degree-Celsius	CarbonDioxide(ppm)	Pressure(PSIG)	CarbonDioxide(ppm)	Null	Current(0-100)Amps	Null
Board-2	2018-02-26 07:07:09	192.168.4...	Null	Temperature(0-50)Degree-Celsius	Current(0-600)Amps	Temperature(0-50)Degree-Celsius	CarbonDioxide(ppm)	Pressure(PSIG)	CarbonDioxide(ppm)	Null	Current(0-100)Amps	Null

Figure 5.11. Board Startup Specifications Table in the Database

The database table in Figure 5.12 is one of the most important tables of the system. This table stores the actual sensor data from the hardware. It has 4 fields, Timestamp, Board ID, Port ID, Value. Timestamp displays information regarding when the data was sent by the microcontroller and stored in the database. THE board ID column provides information regarding from which board does the data belong. Port ID provides information regarding

which Port (or Sensor) of the given Board ID the data comes from. Value is the actual sensor value. This table continuously logs data from several Boards installed. Details regarding how the microcontroller sends data to the server and how data is stored in the database are explained in Chapter 6.

Timestamp	Board_ID	Port_ID	Value
2020-03-19 04:50:08	IAC_Board3	Port_7	0.96
2020-03-19 04:50:08	IAC_Board3	Port_9	0.25
2020-03-19 04:50:07	IAC_Board3	Port_5	395.61
2020-03-19 04:50:07	IAC_Board3	Port_6	0
2020-03-19 04:50:06	IAC_Board3	Port_2	22.85
2020-03-19 04:50:06	IAC_Board3	Port_3	0.04
2020-03-19 04:50:06	IAC_Board3	Port_4	0
2020-03-19 04:49:53	IAC_Board3	Port_9	0.25
2020-03-19 04:49:52	IAC_Board3	Port_6	0
2020-03-19 04:49:52	IAC_Board3	Port_7	0.96
2020-03-19 04:49:51	IAC_Board3	Port_4	0
2020-03-19 04:49:51	IAC_Board3	Port_5	395.55
2020-03-19 04:49:50	IAC_Board3	Port_2	22.85
2020-03-19 04:49:50	IAC_Board3	Port_3	0.04
2020-03-19 04:49:37	IAC_Board3	Port_9	0.25
2020-03-19 04:49:36	IAC_Board3	Port_6	0
2020-03-19 04:49:36	IAC_Board3	Port_7	0.96
2020-03-19 04:49:35	IAC_Board3	Port_4	0
2020-03-19 04:49:35	IAC_Board3	Port_5	395.55
2020-03-19 04:49:34	IAC_Board3	Port_2	22.85
2020-03-19 04:49:34	IAC_Board3	Port_3	0.04
2020-03-19 04:49:21	IAC_Board3	Port_9	0.25
2020-03-19 04:49:20	IAC_Board3	Port_5	395.64
2020-03-19 04:49:20	IAC_Board3	Port_6	0
2020-03-19 04:49:20	IAC_Board3	Port_7	0.96
2020-03-19 04:49:19	IAC_Board3	Port_3	0.04
2020-03-19 04:49:19	IAC_Board3	Port_4	0
2020-03-19 04:49:18	IAC_Board3	Port_2	22.85

Figure 5.12. Database Sensor Readings Table

5.1.3 Application Function Tables

Figure 5.13 is the Entity-Relationship Diagram for an Application Function Table, which is the Forecast Data table. Using values from a particular Board ID and Port ID, the application function developer can generate meaningful results and store them in the Forecast Data table. The Forecast Data table contains columns such as TimeStamp, Board ID, Port ID, Appliance ID, Company ID, Zone ID, and kW Value. Timestamp stores the date and time the kilowatt (kW) value is predicted for. The diagram also shows the Application Function table's relation to the information table and operating table-Sensor Readings. Board ID and Port ID provides information on which board and which port of the board is the value related to. Appliance ID provides the information regarding which appliance the prediction is made for. Company ID and Zone ID provide information on which company and which company's zone is the predicted value for.

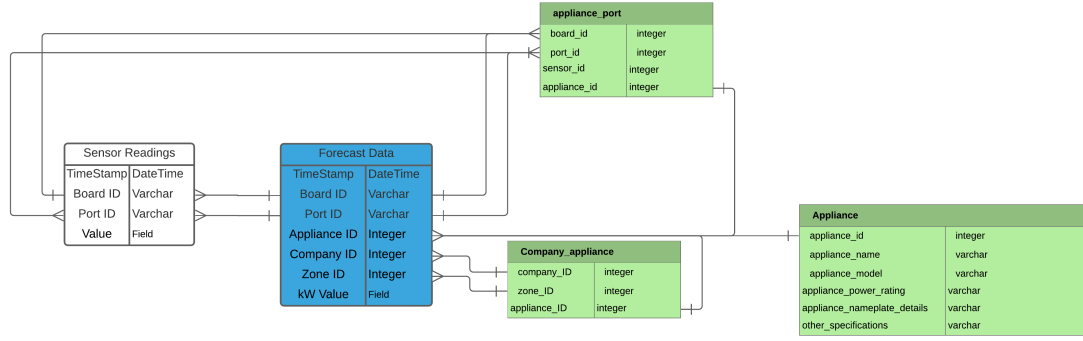


Figure 5.13. Application Function Table E-R Diagram

For example, the application function developer used data from Sensor Readings and values corresponding to Board 3 and Port 9. Port 9 is the current sensor attached to Air Compressor, which is Appliance ID =1. Using the current readings, the application function developer calculated the kilowatt (kW) usage by Air Compressor. Using a prediction algorithm, the application function developer generated forecast data that predicts future usage by the Air Compressor and stored into a database table shown in Figure 5.14. Appendix C shows the procedure to connect Matlab to the database. Connecting to the database allows the application function developer to execute MySQL queries and retrieve Matlab's data for processing. The developer can also store forecast and prediction data into the database tables.

Timestamp	Appliance_ID	Company_ID	Zone_ID	Board_ID	Port_ID	Value_kW
2020-05-12 15:05:56	1	1	317	IAC_Bo...	Port_9	0.051266716
2020-05-12 15:06:11	1	1	317	IAC_Bo...	Port_9	0.051005443
2020-05-12 15:06:26	1	1	317	IAC_Bo...	Port_9	0.050448827
2020-05-12 15:06:41	1	1	317	IAC_Bo...	Port_9	0.049895985
2020-05-12 15:06:56	1	1	317	IAC_Bo...	Port_9	0.049826825
2020-05-12 15:07:11	1	1	317	IAC_Bo...	Port_9	0.050082187
2020-05-12 15:07:26	1	1	317	IAC_Bo...	Port_9	0.04996928
2020-05-12 15:07:41	1	1	317	IAC_Bo...	Port_9	0.049972044
2020-05-12 15:07:56	1	1	317	IAC_Bo...	Port_9	0.050426091
2020-05-12 15:08:11	1	1	317	IAC_Bo...	Port_9	0.050761259
2020-05-12 15:08:26	1	1	317	IAC_Bo...	Port_9	0.050657494
2020-05-12 15:08:41	1	1	317	IAC_Bo...	Port_9	0.050371262
2020-05-12 15:08:56	1	1	317	IAC_Bo...	Port_9	0.050303307
2020-05-12 15:09:11	1	1	317	IAC_Bo...	Port_9	0.050726446
2020-05-12 15:09:26	1	1	317	IAC_Bo...	Port_9	0.050697612
2020-05-12 15:09:41	1	1	317	IAC_Bo...	Port_9	0.050746444
2020-05-12 15:09:56	1	1	317	IAC_Bo...	Port_9	0.050299222

Figure 5.14. Air Compressor Forecast Database Table

5.2 Database Usage

Table 5.1 gives examples of what query the design can support. During the development phase, the developer tries to understand what kind of queries the user might have. While working with the users and supervisors of this project, the system's questions can be answered by the given queries. This does not cover all possible questions, but only example questions that may be asked. Queries can be modified for different requirements.

Table 5.1. Sample Questions along with the Queries

S. No.	Question	Query	User
1	Which company is board 3 installed in?	Select Company_name from board_info where Board_ID='IAC_Board3'	User
2	What is the address of a company?	Select Address from com- pany_info where Com- pany_Name='Industrial Audit Centre'	User
3	Which room is the board installed in?	Select Room from board_info where Board_ID='IAC_Board2'	User
4	Who is the contact person of a company?	Select User_name from user_info WHERE com- pany_id='1'	User
5	What is the email address of a person?	Select Email from user_info WHERE (company_id='2' and Position='Advisor')	User
6	What is the contact email of a company?	Select Email from user_info WHERE company_id='2'	User

Table 5.1 continued from previous page

7	What is the contact number of a person?	Select Contact_Number from user_info WHERE user_name='Dr.Jie Chen'	User
8	What is the contact number of a company?	Select Contact_Number from user_info WHERE (Com- pany_ID='1' and position='Di- rector')	User
9	Which sensor is connected to port2 in board 1?	Select Sensor_Type from port_info where Port_ID='Port_2B1'	User
10	What is the output type of temperature sensor?	Select Distinct Sensor_Out- put_Type from port_info where Sensor_Type='Tempera- ture'	User
11	What is the range of Current sensor?	Select Distinct Sensor_Range from port_info where Sen- sor_Type='Current'	User
12	Which sensors are connected Boiler on Board 1?	Select Sensor_Type from port_info where (Ap- pliance='Boiler' AND Board_ID='IAC_Board1')	User
13	Which sensors are connected Chiller on Board 2?	Select Sensor_Type from port_info where (Ap- pliance='Chiller' AND Board_ID='IAC_Board2')	User
14	How many boards are installed in a company?	Select count(*) from board_info where Company_Name='In- dustrial Audit Centre'	User

Table 5.1 continued from previous page

15	Join tables company info and board info using company ID and display together	Select company_info. Company_Name, company_info. Address, board_info. Board_ID,board_info. Room From company_info Inner join board_info on company_info. Company_ID=board_info. Company_ID	Developer
16	The average value of 240 latest samples of port 4 of board 2	"SELECT Avg(Value) as avg FROM Sensor_Readings WHERE Board_ID='IAC_Board2' AND Port_ID='Port_4' ORDER BY TIMESTAMP desc LIMIT 240"	Developer
17	Inserts user-defined data into sensor config table	INSERT INTO sensor_configuration (Board, Port, Category, Name, System_Requirement) VALUES ('\$value1', '\$value2', '\$value3', '\$value4', '\$value5')	Developer
18	Delete the user-defined name in the sensor config table	DELETE FROM sensor_configuration WHERE Board='\$value1' AND Port='\$value2'	Developer
19	Select certain value from sensor config	SELECT Port as p FROM sensor_configuration WHERE Name='{\$_GET['sensor']}'}	Developer

Table 5.1 continued from previous page

20	Update user-defined values	UPDATE Demand_Control SET Value='{ \$value}' WHERE Sensor='{ \$relay}'	Developer
21	Find the sum of the last 60 entries for ports 3, 9 and 10	SELECT SUM(VALUE) AS value FROM (SELECT TIMESTAMP, VALUE FROM sensor_readings WHERE (Port_ID = 'Port_9' OR Port_ID = 'Port_10' OR Port_ID = 'Port_3') and Board_ID = '{ \$board}')	Developer
		sensor_readings GROUP BY TIMESTAMP ORDER BY TIMESTAMP DESC LIMIT 60	

5.3 Implementation

The database design has been implemented in the MySQL database service hosted in a virtual machine server provided by IUPUI. Several factors were taken into consideration for choosing this service.

MySQL is one of the most preferred and popular database services. It is used in a wide variety of commercial as well as scientific applications. The database server has been implemented into a Linux Red Hat server hosted and managed by Indiana University. The server consists of 16 GigaBytes of RAM, 2 Terabytes of hard disk, and an Intel Xeon Gold processor with 2 cores. This is a virtual machine. The address to this server is in-engr-iac.engr.iupui.edu. The webserver is hosted on the same server as the database server.

The advantages of using MySQL and IU supported server are as follows:

1. Free to use Open Source
2. Data Security
3. On-Demand Scalability
4. High Performance
5. Round-the-clock Uptime
6. Comprehensive Transactional Support
7. Complete Workflow Control
8. Reduced Total Cost of Ownership
9. The flexibility of Open Source

6. WEB SERVER

Monitoring and controlling industry equipment in real-time is the fundamental goal of this project. A web server can serve the Graphical User Interface (GUI) on edge devices. It can implement REST API services, a significant factor of the system designed in this project. REST APIs enable the development of web applications to perform operations on data like create, retrieve, update, and delete using methods such as GET, POST, PUT, DELETE, and PATCH. Asynchronous WebSocket communication and telemetry data exchange are also essential for the system to be successful. The web server also allows remote access to authorized users. Hence a web server is required to provide a user interface to the users. Since directly accessing the database can breach database security, some accesses to the database should not be given to all users. Therefore, preset queries with user input flexibility are essential, so an unauthorized person cannot alter that database. Figure 6.1 demonstrates the ideal working of the system. It shows the communication between the microcontroller edge board, server, and database.

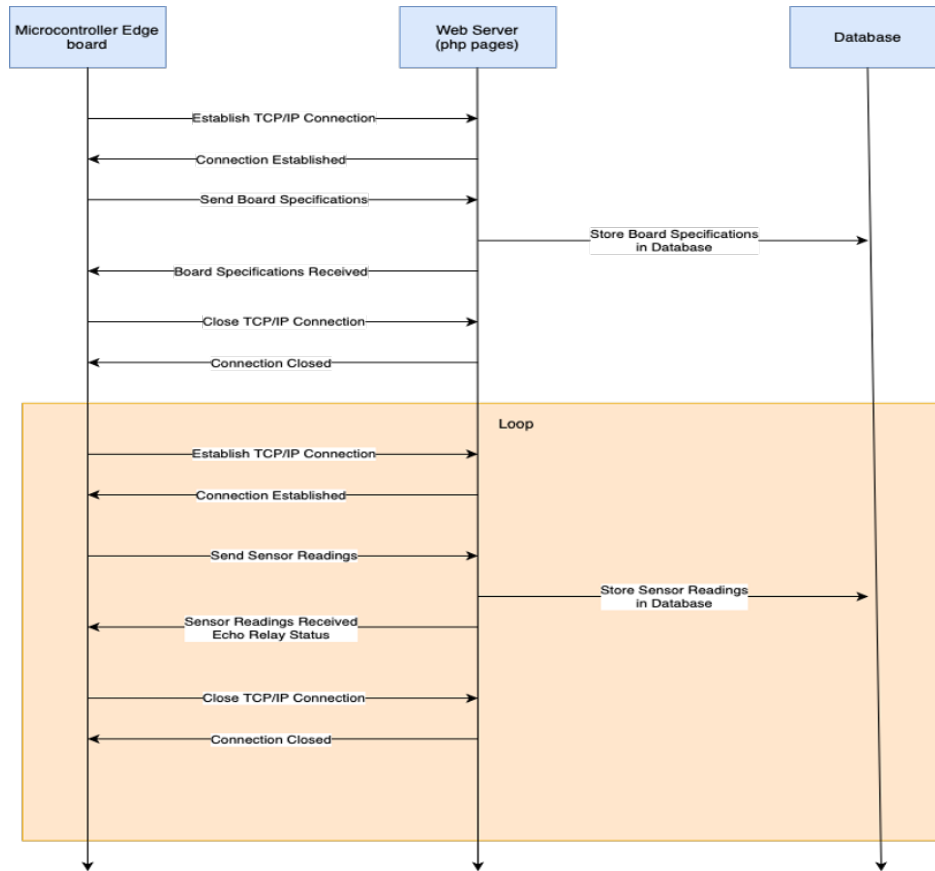


Figure 6.1. Edge-Server Communication

Figure 6.2 shows the sequence diagram[2] when a user is browsing the web pages in the webserver. The user uses a client browser such as Microsoft Edge, Google Chrome, or Safari, to access a website. The web server receives the client browser's request and requests the scripts and application to generate a page to show to the user in the client browser. Suppose the user's request is a dynamic page, such as view data for a specified time period or view data for a particular appliance. The scripts will need to execute SQL queries to get data from the database. Once the script has the user's data via the web server, the script will generate a dynamic page containing the data.

The web server will then respond to the client browser with the dynamically generated page, and the user will be able to view this page in the browser. This sequence diagram for front-end web pages is used when the user wants to view the sensors' data or want to perform actuation to the appliances.

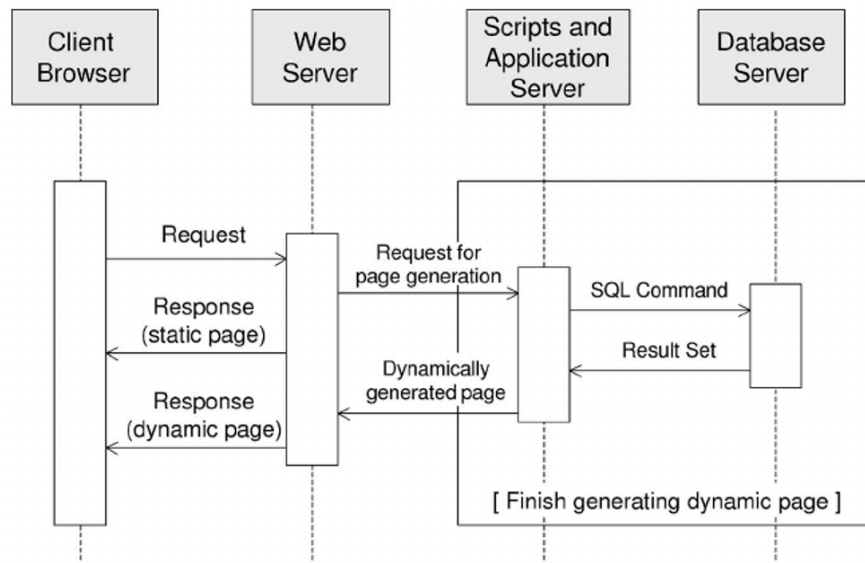


Figure 6.2. Front Rnd Web Pages Sequence Diagram

6.1 Establishing and Closing Connection from the Microcontroller to Cloud

Web servers are designed to handle client requests. Here, the microcontroller is the client that initiates the connection. If the webserver is ready to handle the microcontroller requests, it acknowledges the microcontroller request, and the connection is established. If the webserver is unable to handle requests, it responds as busy. If the connection is successful, the microcontroller sends some data to the webserver to handle. Once the microcontroller sends its request, the connection needs to be closed so other clients/microcontroller can use the webserver to process its requests. One of the main functions is the getreply function. This function is used to get a response from the webserver to the microcontroller and check if the connection is established between the microcontroller and webserver or whether the microcontroller received acknowledgment from the webserver. The code snippet in Figure 6.3 shows this function and error checking. Function “getreply” also stores the server’s response

in a buffer variable called “buf”. The getreply function then checks if the response contains any of the errors such as:

1. ERROR
2. Busy
3. 404 Not Found

If one of these errors appears in the response, it means the connection failed, or acknowledgment was not received. Then the getreply function tries to reconnect to the internet using the function “ESP_Config”.

```
void getreply()
{
    memset(buf, '\0', sizeof(buf));
    t.start();
    ended=0;
    count=0;
    while(!ended) {
        if(esp.readable()) {
            buf[count] = esp.getc();
            count++;
        }
        if(t.read() > timeout) {
            ended = 1;
            t.stop();
            t.reset();
        }
    }
    if(strstr(buf, "ERROR") != 0 || strstr(buf, "busy") != 0 || strstr(buf, "404 Not Found") != 0)
    {
        ESPconfig();
    }
}
```

Figure 6.3. The getreply Function

6.1.1 Establishing a Connection from Microcontroller

To upload data from the microcontroller to the webserver and store it in the database, the microcontroller needs to establish a connection to the webserver. TCP Establish Connection commands shown in Figure 6.4 is defined in the microcontroller firmware to establish the connection to the server. Figure 6.4 is the microcontroller firmware code snippet, which using AT commands, defines the type of connection (TCP) followed by the IP of the webserver followed by the webserver port number we are trying to access. Firmware stores the command to start the connection in a string variable called “snd.” The firmware then sends this

command out to the webserver using the command “SendCMD” and waits for the web server’s response whether the connection has been established. Getreply function is used to get a response from the webserver to the microcontroller, and this function also checks for errors. If there are errors in the server response to the microcontroller, the microcontroller will reconnect to the internet. Further commands will be executed after a connection is established. Firmware has a defined timeout so that microcontroller does not wait for infinite time to get a response from the server. If the connection is not successful, the firmware exits this function and tries to reconnect to the internet. The function TCP_DataSend will be executed again by the main function once the connection is established.

```
char buffer[32];
pc.printf("\n----- Start TCP_IP Connection with WAMP -----< r\n");
strcpy(snd, "AT+CIPSTART=4,\"TCP\", \"134.68.70.220\", 80< r\n");
SendCMD();
timeout=0.1;
getreply();
pc.printf(buf);
pc.printf("\n----- Set TCP Data frame -----< r\n");
memset(text, '\0', sizeof(text));
```

Figure 6.4. TCP_Establish_Connection Function

6.1.2 Closing Connection from the Microcontroller

Closing the Established TCP/IP connection is important so that the webserver can serve requests from other sensor boxes. The microcontroller initiates the closing procedure. Once the data is sent and the microcontroller firmware has received an acknowledgment, the TCP/IP connection is closed by the microcontroller firmware using the commands shown in Figure 6.5. The web server acknowledges the request to close the connection by the microcontroller.

```
pc.printf("\n----- Close TCP/IP Connection -----< r\n");
strcpy(snd, "AT+CIPCLOSE=5< r\n");
SendCMD();
timeout=0.1;
//getreply();
pc.printf(buf);
```

Figure 6.5. Close TCP/IP Connection

6.2 PHP

According to php.net[12], Hypertext Preprocessor (php) is a general-purpose scripting language widely used for web development. Php scripts are executed on the server-side, and results can be returned to the web browser. Php supports text, HTML, CSS, JavaScript too. Php is used to

- Generate static or dynamic page content
- Open, read, write, create or delete files on the server
- Collect data filled on an online form
- Perform database operations like add, delete or modify
- Encrypt data
- Control user access
- Output files like images, pdf or flash movies

6.3 Sending Board Startup Specifications from Microcontroller to Cloud

To store data into the database, the microcontroller must establish a TCP/IP connection with the server. It is also important to store information regarding which sensors are connected to the board. To do so, the microcontroller establishes a TCP/IP connection with the webserver (see section 6.1). Once the connection is established, it prepares the frame containing information regarding the connected sensors. It then sends the prepared frame's packet size to the server, so the server knows how much data is to be received. Once the webserver receives the packet size, the webserver will send an acknowledgment to the microcontroller. The microcontroller then sends the board specification packet to the server. The web server receives the packet, extracts information from the packet, and stores it into a database table. After storing the data in the database, the webserver sends an acknowledgment to the microcontroller. The Block diagram in Figure 6.6 demonstrates the communication process between the microcontroller, web server, and the database.

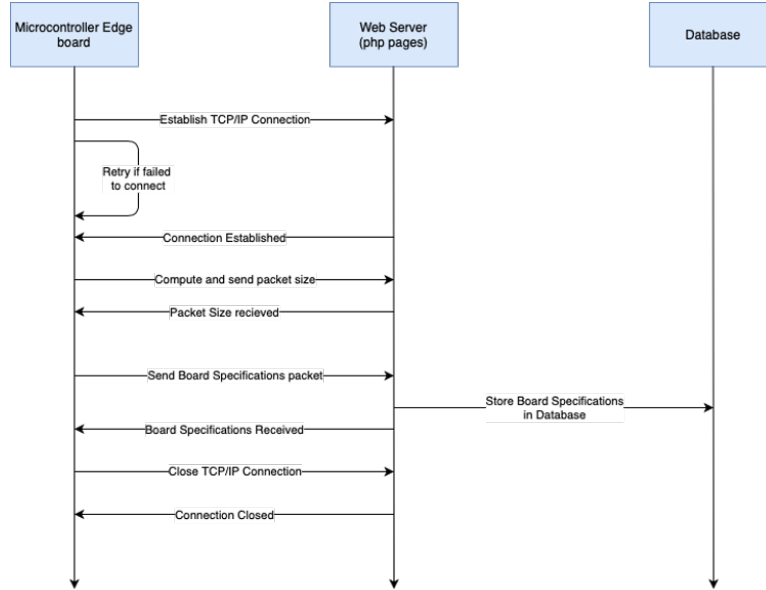


Figure 6.6. Board Specification Communication

6.3.1 Sending Board Startup Specifications From Microcontroller to Web Server

After establishing the connection (see section 6.1), microcontroller firmware prepares the data frame to be sent to the webserver. The microcontroller firmware then prepares to send the board configuration details to the webserver. The board configuration details include:

1. Board ID
2. Global IP of the router the microcontroller is connected to.
3. Local IP of the Microcontroller
4. The sensors connected to each port of the board.

Once the above data is prepared, the size of the data is calculated. The data's size is sent to the webserver first so that the webserver knows how much data is to be received. Figure 6.7 shows the code's snippet, which prepares the data frame to be sent and calculates the prepared frame's size. The firmware then sends the size of the frame to the webserver. The first word 'GET' instructs the php program on server to extract information from the sent frame. The next field `"/Meuser/IACBoardSpecs _DataLoad.php"` is the path of the php

program on the web server. The question mark “?” is a query string for the php program indicating that the next field is the data which the php program needs to extract and store into the database. In the program shown in Figure 6.7, the third line computes the string to be sent to the webserver. The ‘GET’ command is used, then the path of the php program is specified. Then the query string is appended after ‘?’. The query string consists of the Board ID and Port 1 to 10 Sensor information. Port 1 to 10 sensor information consists information regarding the sensor connected to each port for example,

1. Port_1: Temperature(0-50)Degree-Celsius
2. Port_2: Current(0-50)Amps
3. Port_3: Carbondioxide(ppm)
4. Port_4: Pressure(PSIG)
5. Port_7: Null
6. Port_8: Current(0-20)Amps
7. Port_9: Null
8. Port_10: Temperature(0-50)Degree-Celsius

Here, the parameters define the sensor type, range, and unit. If no sensor is attached, the firmware sends Null. The information regarding the the sensor type, range and units is derived from the configuration file. For example here, Port 2 has current sensor attached to it and measures current ranging from 0-50 Ampere (See Figure 5.11 for reference). This information is sent over to webserver using TCP/IP protocol.

```
pc.printf("\n----- Set TCP Data frame -----\\n");
memset(text, '\\0', sizeof(text));
sprintf(text, "GET /MEuser/IAC_BoardSpecs_DataLoad.php?boardId=%s&wifiIP=%s&Port1Sensor=%s&Port2Sensor=%s&Port3Sensor=%s
&Port4Sensor=%s&Port5Sensor=%s&Port6Sensor=%s&Port7Sensor=%s&Port8Sensor=%s&Port9Sensor=%s&Port10Sensor=%s\\n",
Board_specs[1],wifiIP,Port_Description[1],Port_Description[2],Port_Description[3],Port_Description[4],Port_Description[5],
Port_Description[6],Port_Description[7],Port_Description[8],Port_Description[9],Port_Description[10]);
int sz = strlen(text);
pc.printf("\\n\\n%d\\n",sz);
sprintf(snd, "AT+CIPSEND=4,%d\\n",sz);
//strcpy(snd, "AT+CIPSTART=4,\\\"TCP\\\",\\\"184.106.153.149\\\",80\\n");
SendCMD();
timeout=5;
getreply();
pc.printf(buf);
```

Figure 6.7. Board Specification Code Snippet

Once the firmware receives the server's response using the "getreply" command, the frame containing board and port information is sent. If there is an error in receiving the response, the getreply function will attempt to reconnect.

Command - "GET /MEuser/IAC_Board_Specs.php?" is used to access the web server's PHP page. Everything after "?" is handled by the PHP code shown in Figure 6.9 on the server. In this case, the frame has Board ID, Global IP, Local IP, Port information. PHP handles all this data. It will read in the values passed by the microcontroller and store it in the appropriate database table. If the webserver fails to store data into the database, the webserver will respond to the microcontroller with the error.

6.3.2 Storing Received Data From Webserver to Database

Figure 6.8 shows the flow chart of the PHP program handling microcontroller re-quest. The web server receives the data from the microcontroller. The program then extracts data and stores it in a variable. It then establishes a connection with the database. Once the connection is established, the PHP program generates an SQL query. Upon successfully executing the query, the data is stored in the database, and the PHP program replies to the microcontroller with an acknowledgment. If, for any reason, the connection to the database fails or the PHP program is unable to store data to the database, the PHP program will respond to the microcontroller with the error. The microcontroller will store data in the SD card and attempt to reconnect to the internet, as mentioned in chapter 4.

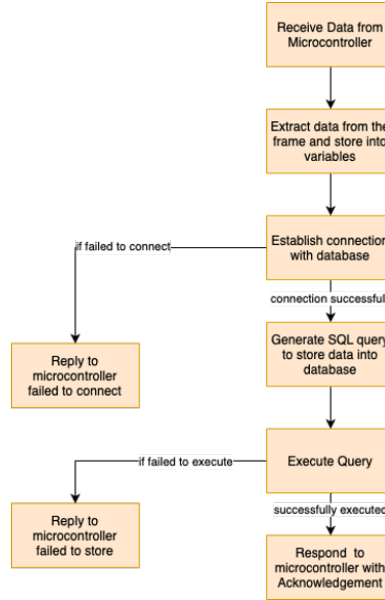


Figure 6.8. PHP Program Flow Chart for Board Startup Specifications

The following PHP page developed on the webserver is triggered by the microcontroller request. As shown in the code, all data after “?” is taken care of by this page. E.g., board, wifiIP, Port1Sensor, etc., If the microcontroller cannot establish a connection, the PHP code will not be triggered. In this case, the microcontroller will log the sensor readings in the SD Card (see Chapter 4). Figure 6.9 is the PHP code executed at the webserver end. Once the code in Figure 6.9 reads the microcontroller’s data using the command “\$_GET”, it establishes a connection to the database server. It will ensure that it is connected to the database where the above-received data will be stored.

```

k?php
$now = new DateTime("now",new DateTimeZone('America/Indiana/Indianapolis'));

//$field = $_GET['field'];
$Port1Sensor = $_GET['Port1Sensor'];
$Port2Sensor = $_GET['Port2Sensor'];
$Port3Sensor = $_GET['Port3Sensor'];
$Port4Sensor = $_GET['Port4Sensor'];
$Port5Sensor = $_GET['Port5Sensor'];
$Port6Sensor = $_GET['Port6Sensor'];
$Port7Sensor = $_GET['Port7Sensor'];
$Port8Sensor = $_GET['Port8Sensor'];
$Port9Sensor = $_GET['Port9Sensor'];
$Port10Sensor = $_GET['Port10Sensor'];
$boardId = $_GET['boardId'];
$wifiIP = $_GET['wifiIP'];

```

Figure 6.9. PHP Page Receiving Information from the Microcontroller

Command “Mysqli,” shown in Figure 6.10 runs at the web server end, and the command establishes a connection to the database. The parameters passed to that function are HOST IP (with Port Number), Username, Password, and database name.

```

$mysqli = new mysqli("in-engr-iac.engr.iupui.edu", "iac_user", "iacPasswd_IUPUI_2019", "iac");
connect to MYSQL database sever
//Check if connection was successfull
if (mysqli_connect_errno())
{
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}

```

Figure 6.10. PHP Code Establishing a Connection to the Database

The code in Figure 6.11 also runs at the webserver end and it stores the received data along with the time stamp in the IAC_Board_Specs table on the database.


```

$datenow = $now->format("Y-m-d H:i:s");
// $value = $value;

$sql2 = "INSERT INTO `IAC_Board_Specs` ('Board-ID', 'Time-Change', 'Wifi-IP', 'Port1-Sensor', 'Port2-Sensor', 'Port3-Sensor', 'Port4-Sensor', 'Port5-Sensor',
Port6-Sensor', 'Port7-Sensor', 'Port8-Sensor', 'Port9-Sensor', 'Port10-Sensor') VALUES ('$boardId', '$datenow', '$wifiIP', '$Port1Sensor', '$
Port2Sensor', '$Port3Sensor', '$Port4Sensor', '$Port5Sensor', '$Port6Sensor', '$Port7Sensor', '$Port8Sensor', '$Port9Sensor', '$Port10Sensor
')";
$result2 = $mysqli->query($sql2);

$sql="SELECT value from Testing where ID=1"; //MYSQL query language
$result= $mysqli->query($sql); //using PHP library to send this query to MYSQL database

while($rs1 = $result->fetch_assoc()) //Read the data one by one if your query get multiple value
{
    echo $rs1['value']."<br>"; // print each data line by line
}
//echo " Hello, World!"
}
}

```

Figure 6.11. PHP Code Storing Values to the Database

6.4 Sending Sensor Values from Microcontroller to Cloud

Section 6.2 sends the information regarding the sensors attached to the microcontroller. In this section, the actual sensor readings are sent from the microcontroller to the web server, and from the web server, it is stored in the database. One of the main features of this system is to store sensor readings into a database. To do so, the microcontroller establishes a TCP/IP connection with the webserver. Once the connection is established, it prepares the frame containing sensor readings. It then sends the prepared frame's packet size to the server, so it knows how much data is to be received. Once the web server receives packet size, the microcontroller sends the sensor readings packet to the server. The web server receives the packet, extracts information from the packet, and stores the readings into the database. The block diagram in Figure 6.12 demonstrates the communication process between the microcontroller, web server, and database. If the microcontroller fails to send data to the server, the microcontroller will receive an error and will store the backup data to SD card and send it later.

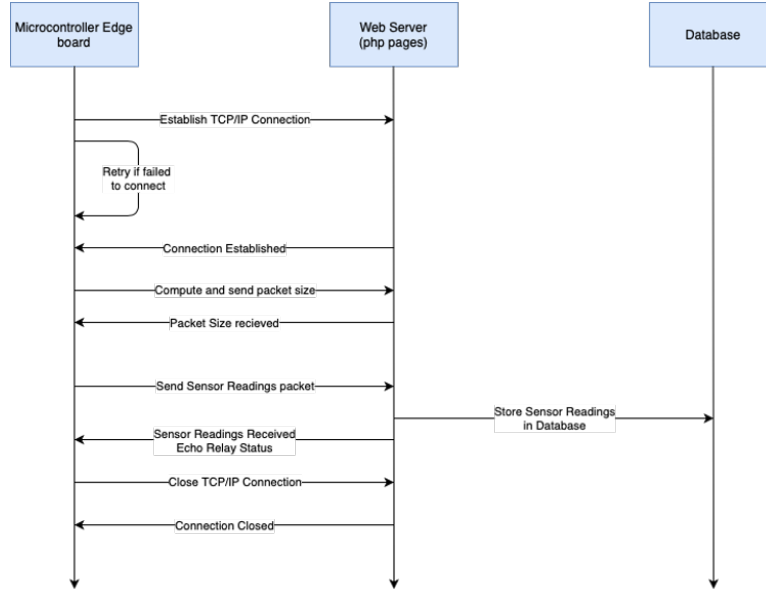


Figure 6.12. Sensor Readings Communications

6.4.1 Send Sensor Values from Microcontroller to Web Server

Figure 6.13 shows the flow chart of the PHP program handling microcontroller request. The web server receives the data from the microcontroller. The program then extracts data and stores it in a variable. It then establishes a connection with the database. Once the connection is established, the PHP program generates an SQL query. Upon successful execution of the query, the data is stored in a database. The PHP program then fetches the relay status from the database and replies to the microcontroller with an acknowledgment and the fetched relay status. If, for any reason, the connection to the database fails or the PHP program is unable to store data to the database, the PHP program will respond to the microcontroller with the error.

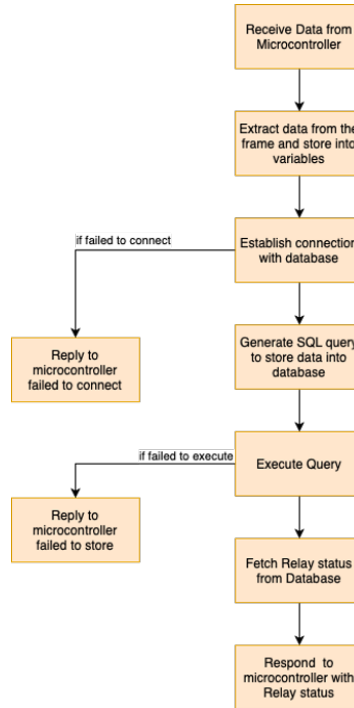


Figure 6.13. PHP Program Flow Chart for Sensor Values

Once the connection is established between the microcontroller and the webserver (see Section 6.1), the microcontroller firmware receives an acknowledgment, it prepares the frame (Payload). The frame consists of the value to be sent for each port. Once the data is prepared, the size of the data is calculated. The data's size is sent first to the server by the firmware so that the server knows how much data is to be received. Once the response is received from the server, the firmware sends the frame containing board and port information. The code snippet for this is shown in Figure 6.14.

```

pc.printf("\n----- Set TCP Data frame -----r\n");
memset(text, '\0', sizeof(text));

char buffer[32];
//strftime(buffer, 32,"%Y%m%d%H%M%S\n", localtime(&seconds));
//strftime(buffer, 32,"%s\n", localtime(&seconds));
//printf("Time as a custom formatted string = %s", buffer);
sprintf(text, "GET /Sensor_readings.php?Board_ID=%s&Port_ID=%s&Value=%0.2f&MCTime=%u\r\n",
Board_specs[4], port[port_number], port_value, (unsigned int)seconds);
int sz1 = strlen(text);
pc.printf("\n\n%d\r\n", sz1);
sprintf(snd, "AT+CIPSEND=4,%d\r\n", sz1);
SendCMD();
pc.printf("\r\n\nGetting Reply\r\n\r\n");
timeout=0.1;
getreply();
pc.printf(buf);
if(strstr(buf, "ERROR") !=0 || strstr(buf, "busy") !=0 || strstr(buf, "404 Not Found") !=0)
{
    ESPconfig();
}

```

Figure 6.14. Microcontroller Preparing Data Frame to Send to Webserver

6.4.2 Storing Received Data From Webserver to Database

Command “GET /Sensor_readings.php?” is used to access the PHP page on the IAC database. Everything after “?” is handled by the PHP code on the webserver. In this case we have Board ID, Port ID, and Sensor Values. All this data is handled by PHP. The PHP program will read in the values passed by the microcontroller and store it in the appropriate database table. Each board is given a unique board ID so that the data can be identified from the board it came from. For testing purpose board 2 was used to send the data to cloud. Its SD card has a configuration file where its Board ID is specified. The PHP command in Figure 6.15 is hit by the microcontroller. As shown in the code, all data after “?” is taken care by this page. Eg. Value for Sensor, Board ID, Port ID etc.

```

<?php
$now = new DateTime("now", new DateTimeZone('America/Indiana/Indianapolis'));

$Board_ID = $_GET['Board_ID'];
$Port_ID = $_GET['Port_ID'];
$Value = $_GET['Value'];
$MCTime = $_GET['MCTime'];

```

Figure 6.15. Web Server PHP Program Receives Data from the Microcontroller

Once the PHP page in Figure 6.15 reads the microcontroller’s data, it establishes a connection to the database server, as shown in Figure 6.16. It will ensure that it is connected

to the database where the above-received data will be stored. Mysqli command establishes the connection. The parameters passed to that function are HOST IP (with Port Number), Username, Password, and database name.

```
$host="localhost";
$port=3306;
$socket="";
$user="iac";
$password="iacPasswd_IUPUI_2019";
$dbname="iac";

$con = new mysqli($host, $user, $password, $dbname);
if ($con->connect_error) {
    die("Connection failed: " . $con->connect_error);
}
```

Figure 6.16. PHP Program Connecting to Database

The web server code shown in Figure 6.17 stores the received data and the time stamp to the database boardID table.

```
$datenow = $now->format("Y-m-d H:i:s");
// $hvalue = $value;
// echo gmdate("Y-m-d H:i:s", $MCTime);
$MCTime = gmdate("Y-m-d H:i:s", $MCTime);
$sql2 = "INSERT INTO iac.sensor_readings(Timestamp, Board_ID, Port_ID, Value,MCTime) VALUES ('$datenow', '$Board_ID', '$Port_ID', '$Value', '$MCTime')";
$result2 = $con->query($sql2) or die("Bad Query: $sql2");
```

Figure 6.17. PHP Page Storing Data to Database

Every time a value is sent to the web server, the server responds back to the microcontroller with that board's relay status, as shown in Figure 6.18. Users can use an automated program or web UI to control the equipment connected to the board. The user needs to toggle the relay status on or off. Once the request is sent to the database, the relay status in the database will change. Once the microcontroller sends any value to the database, the PHP code will read the status change return the acknowledgment changes. Then the microcontroller acts.

```

$sql="SELECT * from iac.backward_communication WHERE BoardID='$Board_ID'"; //MYSQL query language
$result = $con->query($sql) or die("Bad Query: $sql");
// $result= $mysqli->query($sql); //using PHP library to send this query to MYSQL database
while($rs1 = $result->fetch_assoc()) //Read the data one by one if your query get multiple value
{
    echo "\r\n";
    echo "Relay 1=".$rs1["Relay1"]."\r\n";
    echo "Relay 2=".$rs1["Relay2"]."\r\n";
    echo "Relay 3=".$rs1["Relay3"]."\r\n";
    echo "Relay 4=".$rs1["Relay4"]."\r\n";
}
?>

```

Figure 6.18. PHP Program Acknowledging Microcontroller Request and Replying with Relay Status

6.5 Web Pages for Users

Some prototype web pages where users can monitor and control the sensor box is shown in Figure 6.19. The page[9] in Figure 6.19 auto-refreshes every 5 seconds since the database receives a reading every 5 seconds. It shows the real-time readings from the sensor box. This is a prototype to demonstrate that it is possible to develop a GUI for the system and can show real-time data remotely from anywhere in the world. Handheld or Computer with internet can be used to access this web-site.

<div> ← → × Not secure in-engr-iac.engr.iupui.edu/login/Sensor_Readings_Display.php </div>			
Timestamp	Board_ID	Port_ID	Value
2020-03-20 16:37:52	IAC_Board3	Port_9	0.25
2020-03-20 16:37:52	IAC_Board3	Port_7	0.96
2020-03-20 16:37:51	IAC_Board3	Port_6	0
2020-03-20 16:37:51	IAC_Board3	Port_5	395.57
2020-03-20 16:37:50	IAC_Board3	Port_3	0.03
2020-03-20 16:37:50	IAC_Board3	Port_4	0
2020-03-20 16:37:49	IAC_Board3	Port_2	22.53
2020-03-20 16:37:36	IAC_Board3	Port_9	0.25
2020-03-20 16:37:36	IAC_Board3	Port_6	0
2020-03-20 16:37:36	IAC_Board3	Port_7	0.96
2020-03-20 16:37:35	IAC_Board3	Port_4	0
2020-03-20 16:37:35	IAC_Board3	Port_5	395.54
2020-03-20 16:37:34	IAC_Board3	Port_3	0.03
2020-03-20 16:37:34	IAC_Board3	Port_2	22.53
2020-03-20 16:37:21	IAC_Board3	Port_9	0.25
2020-03-20 16:37:20	IAC_Board3	Port_7	0.96
2020-03-20 16:37:20	IAC_Board3	Port_6	0
2020-03-20 16:37:19	IAC_Board3	Port_4	0
2020-03-20 16:37:19	IAC_Board3	Port_5	395.54
2020-03-20 16:37:18	IAC_Board3	Port_2	22.53
2020-03-20 16:37:18	IAC_Board3	Port_3	0.03
2020-03-20 16:37:05	IAC_Board3	Port_9	0.25
2020-03-20 16:37:04	IAC_Board3	Port_7	0.96
2020-03-20 16:37:04	IAC_Board3	Port_6	0

Figure 6.19. Sensor Readings Displayed Real-time on the Web

Figure 6.20 is a snapshot of the web page[5], which allows you to control relays remotely from the web. As mentioned earlier, upon selecting the relays to alter, once the submit button is clicked, it changes to the database table holding relay status. The microcontroller will read these changes and toggle appropriate relays hence controlling the equipment. The best capable latency for this application was determined to be 1 second.

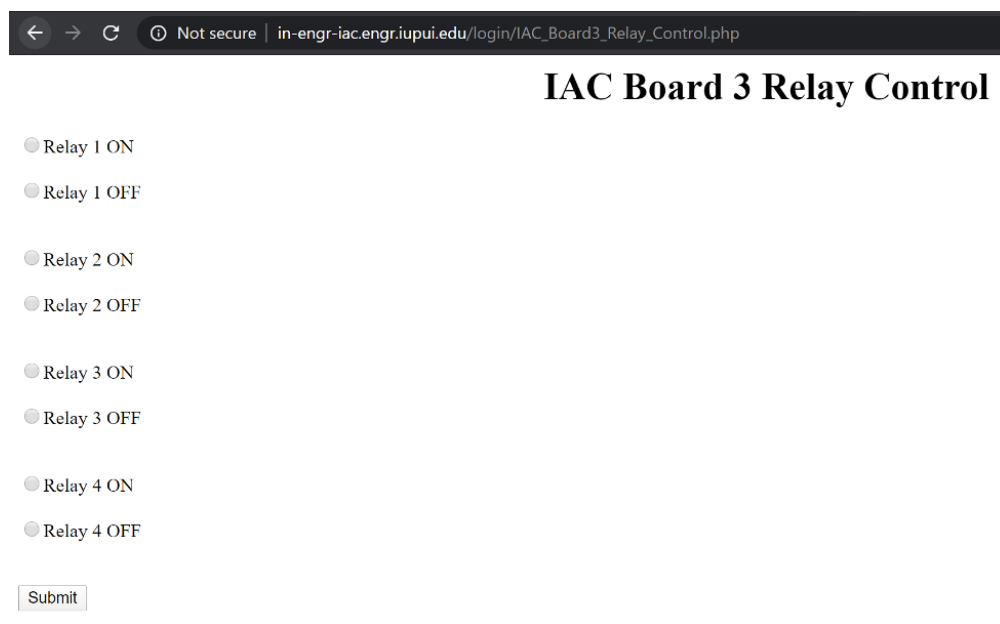


Figure 6.20. Webpage to Control Relays

7. TESTING

For any system to be deployed, testing is essential. There can be many factors that can play in, which can cause a system failure. The implemented system must be tested for an extended period.

7.1 Monitoring Test

A test was conducted in the Emerson building at IUPUI campus for monitoring the energy consumption of the HVAC system in that building. HVAC stands for heating, ventilation, and air conditioning. As the name suggests, the HVAC system is responsible for the heating, ventilation, and cooling of apartments or buildings. The HVAC system has various components. Two components of the HVAC system, called supply fan and return fan, were used to monitor this test's electrical current usage. The supply fan is responsible for supplying fresh air into the rooms. Return fan is responsible for extracting exhaust air out of the room. Two electric current sensors (shown in Figure 7.1) were attached to the supply and return fans' power supply of the HVAC system during the end of winter 2019 to observe a change in energy consumption during the transition from winter to summer. These sensors are clamped around the supply and return fans' power wire and connected to two ports of a sensor box. A portable Wi-Fi device was connected with the Sensor Box so the system can have connectivity to upload data to the cloud. Once the sensors are connected and Wi-Fi is set up, the sensor box is switched on, and data is logged into the cloud. The purpose of this experiment is to test the monitoring of current usage. This test proves that system is able to use the provided sensors, read real-time data from sensors and store it into database using cloud.



Figure 7.1. CTV-C Current Sensor

Figure 7.2 is the graph for current consumption (in Amperes) of supply and return fan for 1 day starting from February 14, 2019, 12:00 AM to February 15, 2019, 12:00 AM. The orange line in the graph represents current consumption in Amperes for the Supply fan, and the blue line corresponds to the current consumption in Amperes for the return fan. As shown in Figure 7.2, the supply and return fans are turned on and use electricity. After 7:15 PM, the supply and return fans are turned off, so the current consumption goes down to 0. At around 11 PM, the fans are turned on again, and the electrical current usage spikes up.

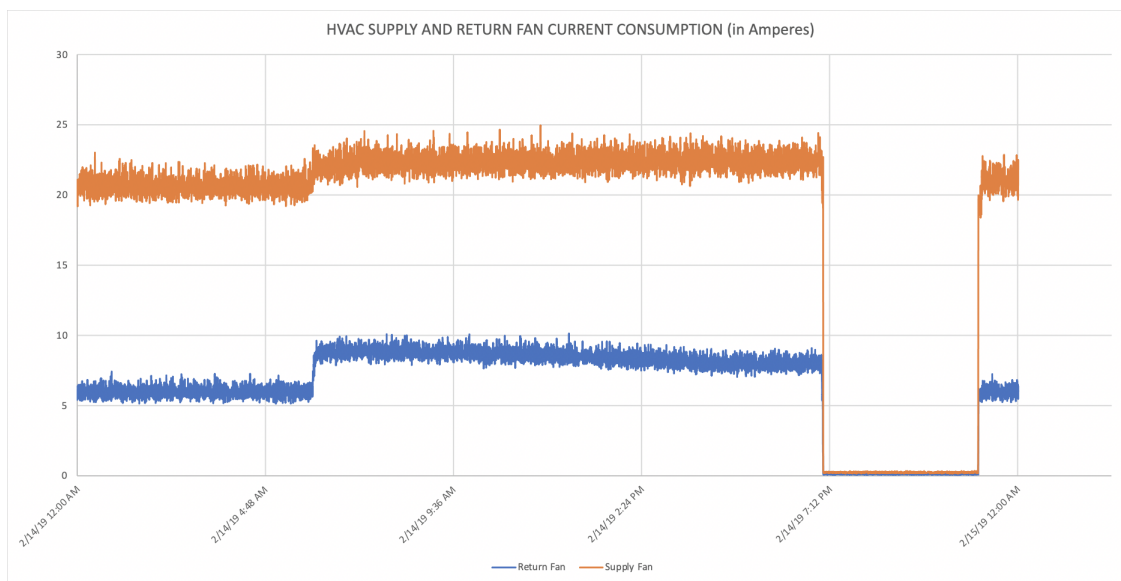


Figure 7.2. Current Usage of Supply and Return Fan Between February 14, 2019 12:00 AM – February 15, 2019 12:00 AM

While Figure 7.2 is the graph for current consumption (in Amperes) by supply and return fans for 1 day, Figure 7.3 is the graph for current consumption (in Amperes) by supply and return fans for 6 days. Figure 7.3 is the graph for supply and return fan's energy consumption for February 13, 2019 – February 18, 2019. The orange line in the graph represents current consumption in Amperes for Supply fan, and the blue line corresponds to the current consumption in Amperes for return fan. The spikes represent the turn off/on of the fans.

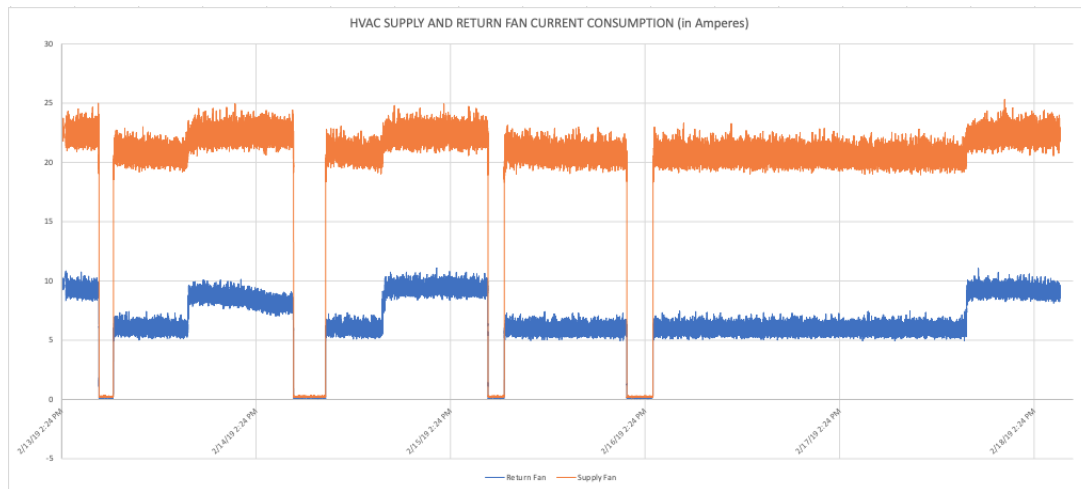


Figure 7.3. Current Usage of Supply and Return Fan Between February 13, 2019 – February 18, 2019

7.2 Monitoring and Control Test

The demo station showed in Figure 7.4 was developed to mimic a manufacturing facility. This demo board can demonstrate applications like the future forecast of energy consumption by equipment and implementation of (energy efficient algorithm) peak shaving algorithms using Matlab. An analyst has implemented energy prediction based on the data logged by this demo station. The peak shaving algorithm is still under development. This board is tested to run for 3 months without human interruption. Using the data logged by the demo station, an analyst can perform analysis on collected data and make predication (manually or using a program) on future energy usage by the connected equipment. It has various components such as an air compressor, a light bulb which acts as a heater, a CO2 sensor for

occupancy, pressure relief valve. Current is monitored for the Air Compressor, Light bulb (heater), and a Motor to carry out tests. Air compressor, motor, light bulb, and pressure relief valve can be turned on/off using the actuating mechanism remotely. Each sensor uses one port, and the sensors can be connected to any port of the board. The purpose of creating the demo station is to monitor energy, control connected equipment, retrieve data from database to matlab and apply energy efficient algorithm based on real-time and historic data.

The Demo Station consists of the following components (as labeled in Figure 7.4):

1. Microcontroller PCB
2. Monitor to display data
3. Compressor
4. Air tank
5. Air tank Release Valve
6. Light bulb enclosed in a case
7. Current sensor for Air compressor (with Actuator)
8. Air tank pressure sensor
9. Current sensor for the light bulb (with Actuator)
10. Heater Temperature sensor
11. CO2 sensor
12. Air Release actuator

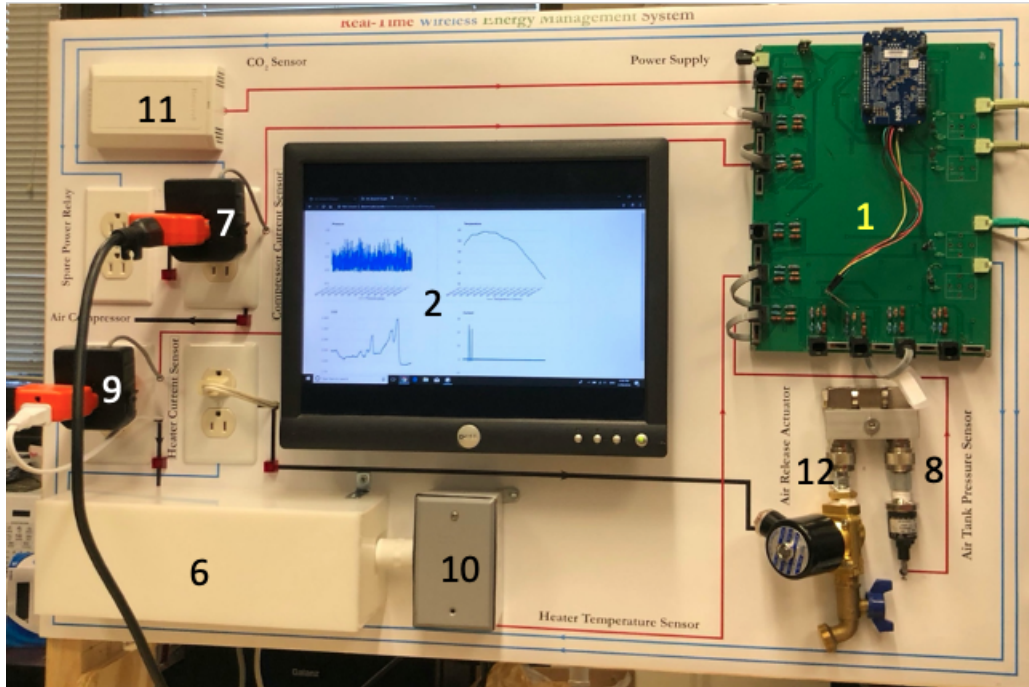


Figure 7.4. Demo Station

Figure 7.5 shows the front-end application that displays the energy consumption of an Air Compressor, a Light bulb, and a Motor connected to the dashboard. Energy consumption is calculated by using the data from the Sensor Readings in the database operating table.

The sensor readings table provides the current consumption in Amperes. The equipment is connected to a 120 Volt AC supply. Based on the current value and voltage value, power (in Watts) can be computed using the formula

$$P = V \times I$$

This gives the power in wattage. The watt-hour usage is computed by multiplying the power by 24 since the data is for 24 hours. The watt-hour usage is then converted to kilowatt-hour. The formula for kilowatt-hour is

$$Usage\ in\ kWh = \frac{(Watts \times Hour)}{1000}$$

Using data from the Sensor Readings operating table, an analyst makes a prediction [14] (may calculate manually or using a program) on future energy usage by the equipment (as shown by the Light blue line) and stores it in the Application Function Table called Forecast. The front-end developer uses the operating and Application function table to generate the graph shown in Figure 7.6. The page also displays the current status of equipment, whether it is on or off, and gives the user the ability to turn it on or off. As seen in the graph of Figure 7.5, the peaks are repetitive. This pattern is called the duty cycle, where the machines are turned on at a particular time and turned off once the machine is done. The peaks in the graph depict that the equipment is turned on and consuming energy. The red line shows total energy consumption by all connected equipment combined. The page in Figure 7.5 shows that using the existing hardware, database and webserver, such front-end applications can be developed and using the data from the sensors, applications such as demand forecast or prediction algorithm can be implemented.

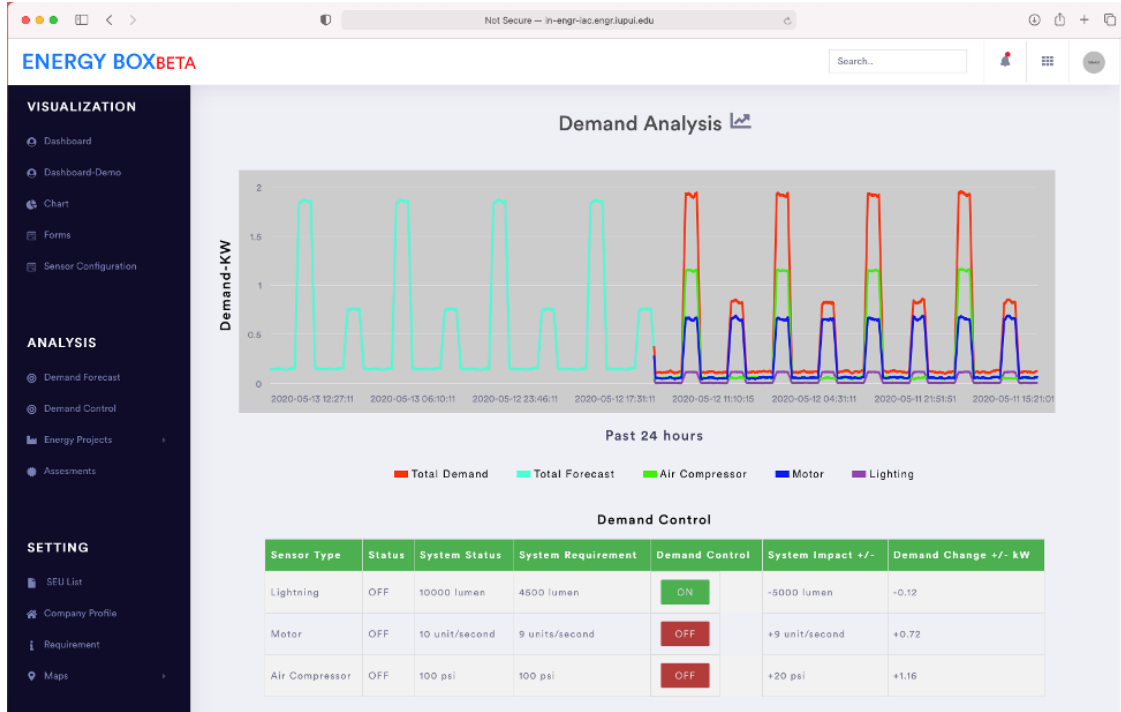


Figure 7.5. Front-end Application Demand Forecast

7.3 Demonstration of SD Data Storage during Network Failure and the Data Recovery after the Restoration of the Network Connection

This scenario is used to test the sensor monitoring module, equipment control module and network failure module. For this testing scenario, the demo station mentioned in Section 7.2 is used. In this setup, the system is started, and the data is being read by the sensor box and sent to the cloud. From a website, shown in Figure 6.20, the user sends a command to turn on the motor. The current usage goes up as shown in Figure 7.6. The user again commands to stop the motor for few minutes and again turns it on. The current usage shown in Figure 7.6 shows that the control module works, and the current usage goes up only when the motor is turned on.

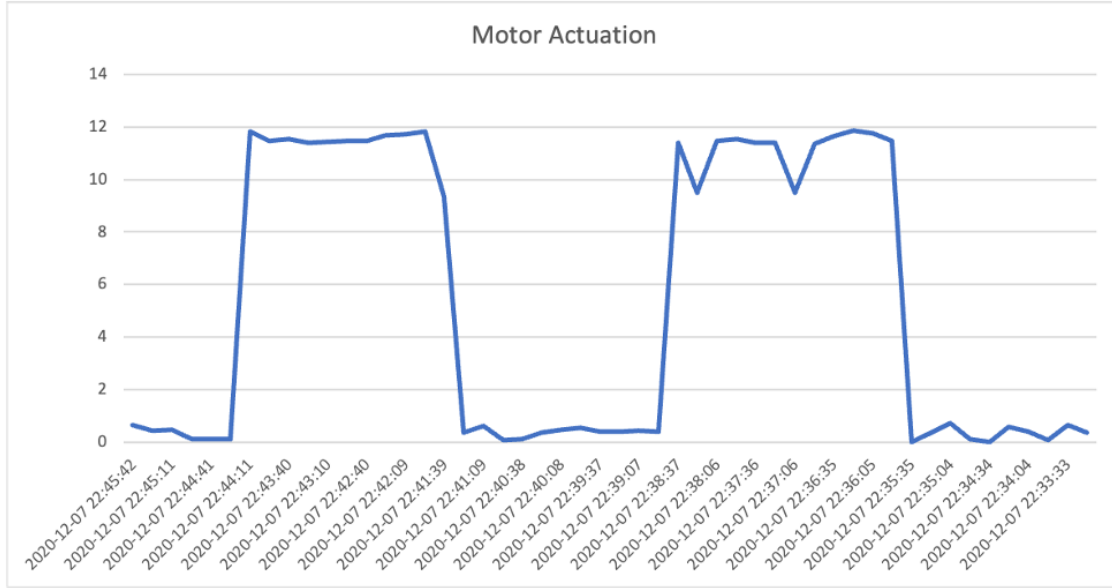


Figure 7.6. Motor Current Usage Based on User Control

This experiment proves that the monitoring and control module works as desired. Testing the network failure module includes disconnecting the internet connection to which the sensor box is connected. The sensor box should attempt to reconnect and also store data into SD card which can be sent to the server later when connection is reestablished. Figure 7.7 is the screen shot of the website where user can see incoming data. The column Timestamp indicates the time when data is received by the server and the column MCTime indicates the time at which the microcontroller read the data. As seen in the red box in Timestamp column, the network was cut off and reconnected after a few minutes. During this time, microcontroller senses that it is not able to send the data to cloud, so it stores this data into the SD card and attempts to reconnect to the internet. Once the connection is established and the microcontroller is able to send the data to cloud, it will check the SD card for back up data. If there is backup data, it will send it to the server and continue reading and sending real time data. The blue boxes in MCTime column show that the data was actually read earlier compared to the time the server received it.

This data proves that in case of network failure, the sensor box is able to back up data to SD card and reconnect to the server. Once it reconnects to the server, it will send the back up data to the server.

Timestamp	Board_ID	Port_ID	Value	MCTime
2020-12-17 15:33:09	IAC_Board1	Port_9	19.01	2020-12-17 15:33:07
2020-12-17 15:33:06	IAC_Board1	Port_9	13.96	2020-12-17 15:17:28
2020-12-17 15:32:50	IAC_Board1	Port_9	7.41	2020-12-17 15:32:49
2020-12-17 15:32:47	IAC_Board1	Port_9	11.57	2020-12-17 15:17:12
2020-12-17 15:32:32	IAC_Board1	Port_9	11.19	2020-12-17 15:32:30
2020-12-17 15:32:28	IAC_Board1	Port_9	14.56	2020-12-17 15:16:57
2020-12-17 15:32:13	IAC_Board1	Port_9	18.92	2020-12-17 15:32:11
2020-12-17 15:32:09	IAC_Board1	Port_9	16.75	2020-12-17 15:16:42
2020-12-17 15:31:53	IAC_Board1	Port_9	26.71	2020-12-17 15:31:52
2020-12-17 15:31:49	IAC_Board1	Port_9	19.01	2020-12-17 15:16:27
2020-12-17 15:31:34	IAC_Board1	Port_9	11.15	2020-12-17 15:31:32
2020-12-17 15:31:30	IAC_Board1	Port_9	18.45	2020-12-17 15:16:11
2020-12-17 15:15:58	IAC_Board1	Port_9	17.46	2020-12-17 15:15:56
2020-12-17 15:15:49	IAC_Board1	Port_9	27	2020-12-17 15:15:47
2020-12-17 15:15:41	IAC_Board1	Port_9	17.48	2020-12-17 15:15:39
2020-12-17 15:15:33	IAC_Board1	Port_9	22.82	2020-12-17 15:15:31
2020-12-17 15:15:25	IAC_Board1	Port_9	7.96	2020-12-17 15:15:22
2020-12-17 15:15:16	IAC_Board1	Port_9	11.11	2020-12-17 15:15:14
2020-12-17 15:15:08	IAC_Board1	Port_9	4.1	2020-12-17 15:15:06
2020-12-17 15:13:03	IAC_Board1	Port_9	22.96	2020-12-17 14:55:20
2020-12-17 15:12:56	IAC_Board1	Port_9	16.53	2020-12-17 15:12:54
2020-12-17 15:12:44	IAC_Board1	Port_9	16.97	2020-12-17 14:55:04
2020-12-17 15:12:38	IAC_Board1	Port_9	11.12	2020-12-17 15:12:36
2020-12-17 15:12:27	IAC_Board1	Port_9	17.32	2020-12-17 14:54:48
2020-12-17 15:11:19	IAC_Board1	Port_9	10.85	2020-12-17 15:11:17
2020-12-17 15:11:08	IAC_Board1	Port_9	11.08	2020-12-17 14:54:31
2020-12-17 15:11:00	IAC_Board1	Port_9	20.64	2020-12-17 15:10:58
2020-12-17 15:10:49	IAC_Board1	Port_9	23.45	2020-12-17 14:54:15
2020-12-17 15:10:41	IAC_Board1	Port_9	22.83	2020-12-17 15:10:39

Figure 7.7. Data from the System with Network Failure

Figure 7.8 - Figure 7.11 show the graphical representation of back up feature. The purple line indicates usage of a 75 watt light bulb. The brown line shows the usage of a 750 watt heater fan. The red line shows usage of the air compressor. and the blue line indicates the total usage by all the equipment. For this experiment, light bulb and heater fan is used. The light bulb and the fan were turned on. After 2-3 minutes, the internet was disconnected to prevent the microcontroller from sending the data to the server. Figure 7.8 shows the light bulb and the fan turned on.

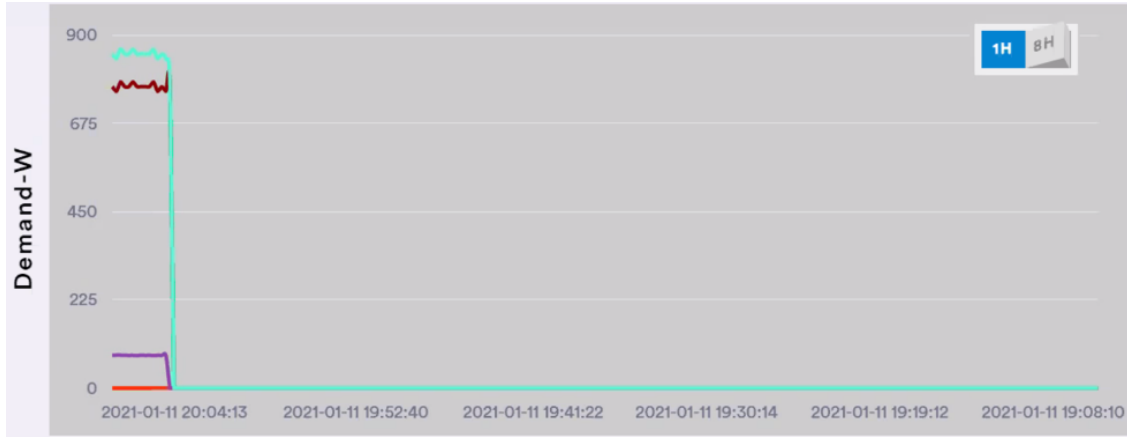


Figure 7.8. Equipment Turned On

After 2-3 minutes of turning on the equipment, the internet is disconnected. Therefore, no new data will be pushed into the database. The graph detects no new data hence, as shown in Figure 7.9, the graph will keep moving but no new data will be plotted. The orange circle in Figure 7.9 shows that no new data has been received by the server hence, no data is plotted.

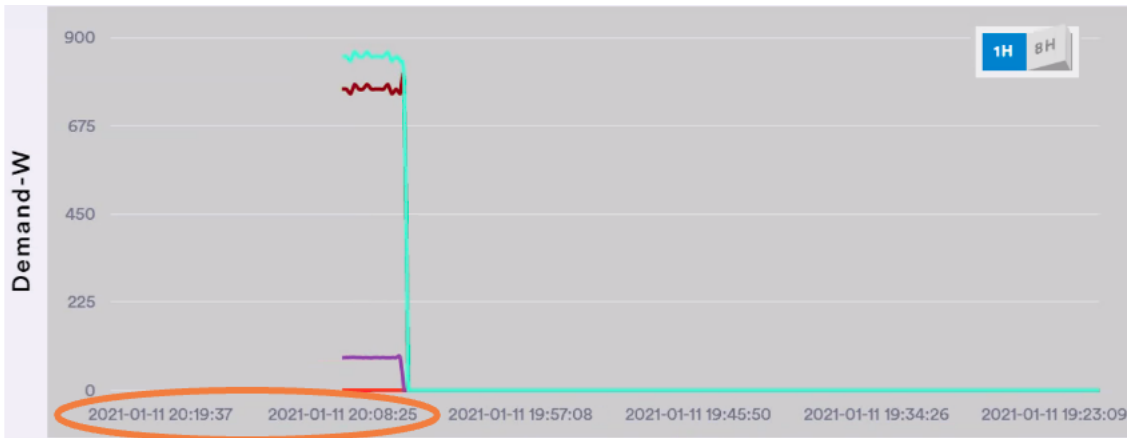


Figure 7.9. Backup Mode

The internet cut-off is detected by the microcontroller and it starts storing the data to the SD card. After around 11 minutes, the internet is turned on again and the light bulb and fan are turned off. The graph detects latest data and starts plotting again. This time, the equipment are turned off, so the usage drops to 0 as shown in Figure 7.10. However, the

actual use time of the equipment is shown in Figure 7.11 after the backup data is received by the server.

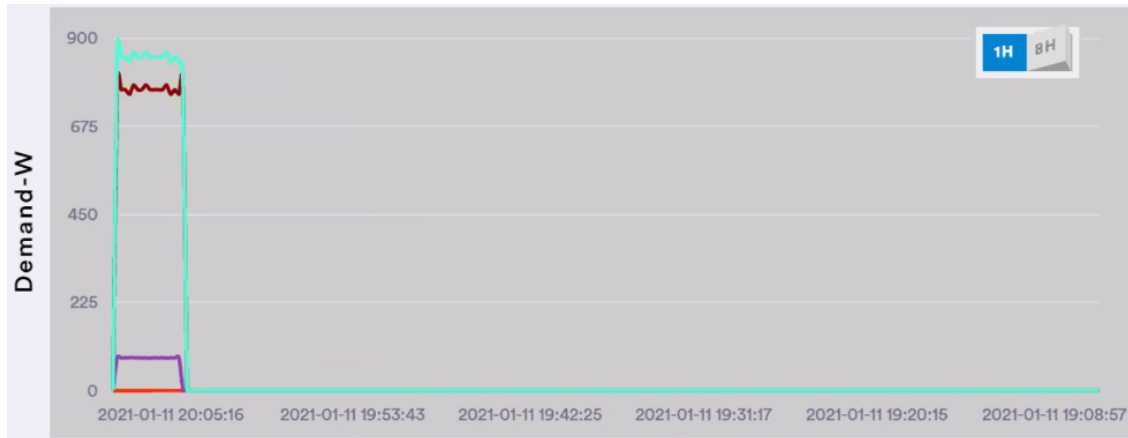


Figure 7.10. Internet Turned On

When the microcontroller is able to send the data to the server, it will start sending backup data too. Once, all the backup data is sent, the actual equipment usage is shown in the graph as shown in Figure 7.11. The orange circle in Figure 7.11 indicates the backup data received by the server.

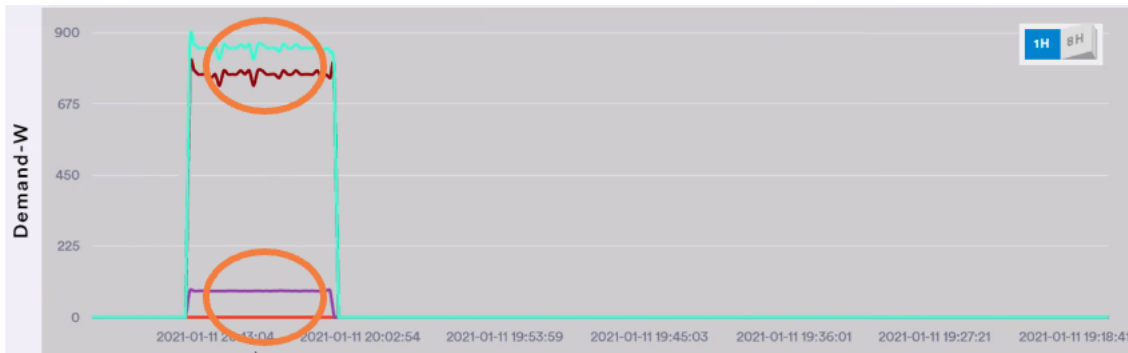


Figure 7.11. Backup Data Sent

The purpose of this experiment is to demonstrate the backup feature of the system. When microcontroller fails to send data to server, the data is stored in SD card as backup data and once it reestablishes connection to the server, it will send real-time data and backup data so there is no loss of data.

8. CONCLUSION

Given the project's requirements, the system seems to have fulfilled most, if not all, the requirements. The system designed can read data from sensors, convert to standard units, and upload to a web server. The web server stores the data into database tables. The microcontroller is also capable of logging offline in case of internet failure. The system can turn the connected equipment on or off based on user request or an automated program. After rigorous testing and improvements, it can be concluded that the system is reliable and can be ushered to the next stage of development. The framework and architecture foundation is laid strong, and many of the probable deficiencies have been taken care of. It can be deployed into actual factories to monitor the manufacturing process. However, the GUI is in the preliminary stage; therefore, it could take a little effort to understand the features.

9. FUTURE WORK

The system has a great scope for expansion. Though the framework is present, a lot of effort is required to gather legacy data. The end goal of the system is to make smart decisions based on legacy data. The architecture, too, has a lot of scope for improvements. Below listed are some of the improvements that are suggested for better functioning of the system:

1. Real-Time Operating System

Using Real-Time Operating System will add more time-sensitive functionalities. There are areas in the code where it has to wait for a certain instructions to be executed in the current modular programming. However, using RTOS, the waiting period can be utilized to perform other waiting tasks.

2. GUI

The current User Interface is good for demonstration purposes. However, a lot of front end work needs to be done to make the system commercial and convenient.

3. PCB redesign

The current PCB design uses a large area and also has many circuit errors. For the prototype, jump wires were used to correct the connections. However, a much better design and smaller form factor can make the sensor box look neat and professional.

4. Remote Configuration

The current model uses an SD card to read the configuration file and set up the microcontroller sensors to read. However, a remote configuration system can be made where the sensors can be input or changed by the user using a UI on the web. Also, remotely configurable sampling time can be added.

REFERENCES

- [1] *6 Ways Manufacturers Can Reduce Industrial Energy Costs*. Nov. 2020. URL: <https://blogs.constellation.com/energy-management/6-ways-to-reduce-industrial-energy-costs/>.
- [2] S. Elbaum et al. “Leveraging user-session data to support Web application testing”. In: *IEEE Transactions on Software Engineering* 31.3 (Mar. 2005), pp. 187–202. DOI: [10.1109/tse.2005.36](https://doi.org/10.1109/tse.2005.36).
- [3] *Energy Flow Charts*. URL: <https://flowcharts.llnl.gov/>.
- [4] Dion Hinchcliffe. *Is the Internet of Things strategic to the enterprise?* May 2014. URL: <https://www.zdnet.com/article/is-the-internet-of-things-strategic-to-the-enterprise/>.
- [5] *IAC Board 3 Relay Control*. URL: http://in-engr-iac.engr.iupui.edu/login/IAC_Board3_Relay_Control.php.
- [6] *IAC: Indiana University-Purdue University*. URL: <https://iac.university/center/IP>.
- [7] Dayarathna Miyuru. *Comparing 11 IoT Development Platforms - DZone IoT*. July 2019. URL: <https://dzone.com/articles/iot-software-platform-comparison>.
- [8] Amirfardad Salami and Alireza Yari. “A framework for comparing quantitative and qualitative criteria of IoT platforms”. In: *2018 4th International Conference on Web Research (ICWR)* (2018), pp. 34–39. DOI: [10.1109/icwr.2018.8387234](https://doi.org/10.1109/icwr.2018.8387234).
- [9] *Sensor Readings Display*. URL: http://in-engr-iac.engr.iupui.edu/login/Sensor_Readings_Display.php.
- [10] *U.S. Energy System Factsheet*. URL: <http://css.sites.uofmhosting.net/factsheets/us-energy-system-factsheet>.
- [11] Alex Vakaloudis and Christian O’leary. “A framework for rapid integration of IoT Systems with industrial environments”. In: *2019 IEEE 5th World Forum on Internet of Things (WF-IoT)* (2019), pp. 601–605. DOI: [10.1109/wf-iot.2019.8767224](https://doi.org/10.1109/wf-iot.2019.8767224).
- [12] *What is PHP?* URL: <https://www.php.net/manual/en/intro-what-is.php>.
- [13] *What is the Internet of Things (IoT)?* URL: <https://www.oracle.com/internet-of-things/what-is-iot/>.
- [14] Da-Chun Wu et al. “Air compressor load forecasting using artificial neural network”. In: *Expert Systems with Applications* 168 (2021), p. 114209. DOI: [10.1016/j.eswa.2020.114209](https://doi.org/10.1016/j.eswa.2020.114209).

A. APPENDIX

A.1 Design and Implementation Requirements

The IAC provided following requirements for the smart manufacturing platform.

Table A.1. System Requirements

ID	Requirement	Type	Level
1	The subsystem shall provide power to the board.	Function	Threshold
2	The subsystem shall provide power to any connected sensor(s).	Function	Threshold
3	The subsystem shall connect to a 240-volt wall outlet.	Function	Threshold
4	The subsystem shall have an conversion circuit.	Function	Threshold
5	The subsystem shall have a power management circuit.	Function	Threshold
6	The selected board is the FRDM-K64F	Constraint	None
7	The sensor list must be accounted for.	Constraint	None
8	The subsystem shall connect to the board A to D headers.	Physical	Threshold
9	The subsystem shall connect to the sensors.	Physical	Threshold
10	The subsystem shall convert sensor output to ADC input.	Function	Threshold
11	The subsystem shall allow any sensor to connect to any port.	Physical	Threshold
12	The subsystem shall have a common interface.	Physical	Threshold
13	The subsystem shall connect to board control output.	Physical	Threshold
14	The subsystem shall connect to systems.	Physical	Threshold
15	The subsystem shall turn on the system when it receives a cmd signal.	Function	Threshold
16	The subsystem shall turn off the system when it receives a cmd signal.	Function	Threshold

Table A.1 continued from previous page

17	May need to boost signal.	Constraint	None
18	May need to isolate circuits.	Constraint	None
19	The system shall connect to internet	Constraint	None
20	The system shall have webservices	Constraint	None
21	The system shall have database tables to store data	Constraint	None
22	The system shall reconnect in case of internet failure	Function	Threshold
23	The subsystem shall hold the MCU.	Physical	Threshold
24	The subsystem shall hold the Wi-Fi module.	Physical	Threshold
25	The subsystem shall hold the SD card.	Physical	Threshold
26	The subsystem shall hold the power supply circuit.	Physical	Threshold
27	The subsystem shall hold the actuator circuit.	Physical	Threshold
28	The subsystem shall hold the conversion circuit.	Physical	Threshold
29	The subsystem will allow sensors to be connected.	Physical	Threshold
30	The Wi-Fi module is the ESP8266.	Constraint	None
31	The subsystem shall be connected to the MCU.	Physical	Threshold
32	The SD Card shall be readable and write-able.	Functional	Threshold
33	The subsystem shall save the sensor data to the SD card in case it loses connection to internet.	Functional	Threshold
34	The subsystem shall provide write capabilities to an attached SD card.	Functional	Threshold
35	The subsystem shall provide read capabilities to an at- tached SD card.	Functional	Threshold
36	The subsystem shall remove old data from SD Card when SD Card is full.	Functional	Threshold

A.2 Computation and Communication Components

The hardware components were a FRDM-K64F Evaluation Board and an ESP8266 WiFi Module . A C/C++ software for running these hardware had been written using mbed OS2 SDK.

A.3 Sensor List

The hardware of the proposed platform consists of sensors for each type of SEU (Significant Energy Use) equipment. The interest is energy consumption, electricity demand, cost and outputs associated with operator requirements. Thus, the data that affect these factors need to be collected from the SEUs. The typical sensors needed are electrical current for energy consumption, pressure, temperature, CO2, flow rate etc. The sensors need to monitor the SEU's operating conditions such as air tank capacity of compressors, occupancies energy consumptions etc. Such sensors are generally used by auditors at IAC to log data of manufacturing plants. Since the sensors are tried and tested by IAC for several audits, the sensors listed in Table A.2 were selected to ensure industry standards are maintained for development and testing. The board has a reference voltage of 0-3.3V. These sensors need to interface with the evaluation board.

Table A.2. Sensor List

Name	Type	Range	Input	Output
CTV-C	AC Current	0-100 amps	No input	0-2.5 VDC
CTV-E	AC Current	0-600 amps	No input	0-2.5 VDC
TOAV22	Temperature	0-50C	12-30 VDC	0-5, 0-10 VDC
CDI-5200	Compressed Air-flow	2-200 scfm	24 VDC	4-20 mA
T-ASH-G1-200	Pressure Transducer	0-200 psig	9-36VDC	1-5 VDC
C7232A1008	CO2	0-2000 ppm	24 VDC	0-10 VDC,

Table A.2 continued from previous page

HE-67S3-0N0BT	Humidity	0-100%	14-30 VDC	0-5, 0-10 VDC
---------------	----------	--------	-----------	---------------

The selected sensors can be powered by 24V so they can use the same power supply. All selected sensors can generate one of the 4 common output types.

Table A.3. Sensor Output List

Type	Sensor
0-5 Volt	TOAV22, T-ASH-G1-200
0-10 Volt	C7232A1008, HE-67S3-0N0BT
0-20mA	C7232A1008, CDI-5200
0-2.5V	CTV-C, CTV-E

A.4 Power Supply

The power supply provides power to the microcontroller and sensors as shown in Figure A.1.

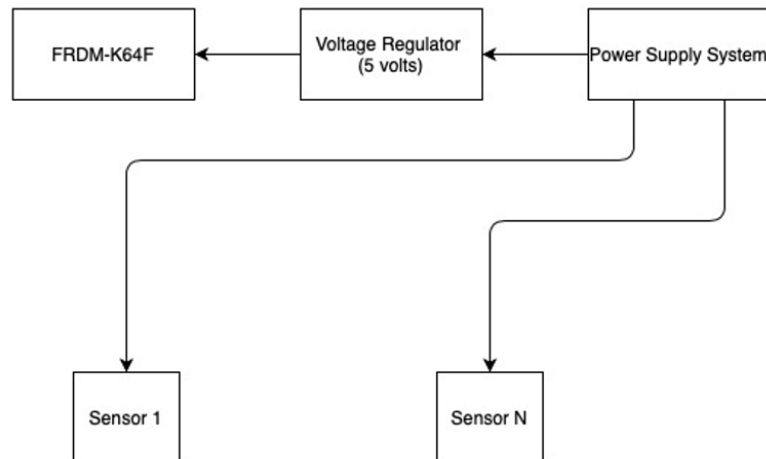


Figure A.1. Power Outline

A.5 Sensor-Board Interface Circuit

The sensor-board interface converts the various sensor outputs to the microcontroller 0-3.3V reference voltage as shown in Figure A.2.

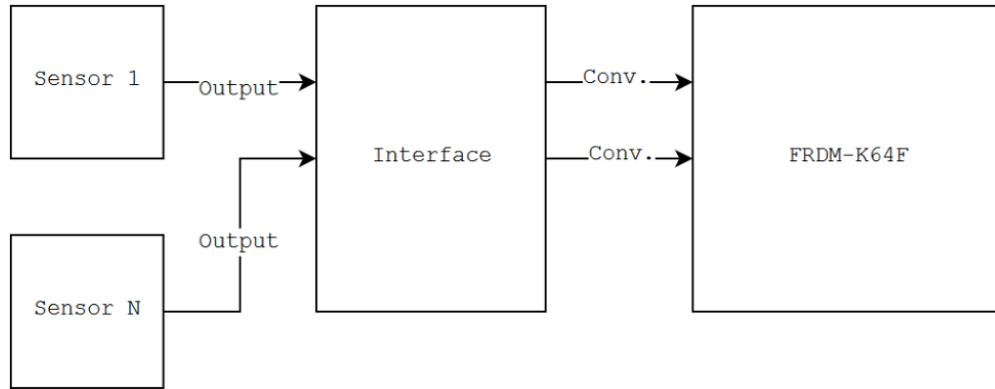


Figure A.2. Sensor-Board Interface Outline

A.6 Sensor Connection

The system should have a common port connection. Any type of sensor should be able to use and directly connected to any port. Figure A.3 shows the outline for connection of the sensor to microcontroller port.

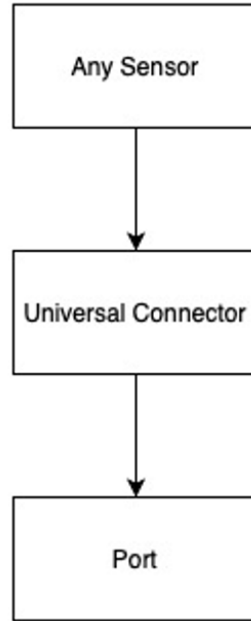


Figure A.3. Sensor Connection Outline

A.7 Actuator Circuit

The actuator circuit takes a command signal from the microcontroller and turns an attached 5V system on or off. Figure A.4 shows the outline of actuator circuit connected to the system and microcontroller and the command from microcontroller can turn on/off the system using actuator circuit.

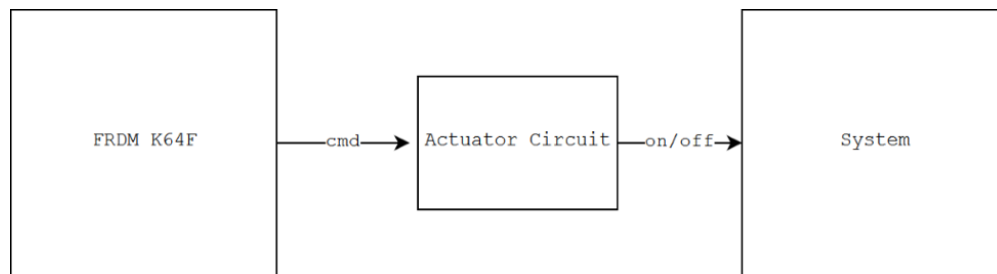


Figure A.4. Actuator Circuit Outline

A.8 Logging to Database

The system consists of wifi module, which connects to internet and sends the sensor readings to cloud to store in database.

A.9 Computation and Communication Board

The hardware component is an essential part of the system since it acquires data from sensors connected to physical equipment in the factory monitoring essential data and transmitting it to the server component to store in database. Figure A.5 consists of the modules, which make up the hardware of the system.

1. FRDM-K64F – The microcontroller chosen for this system is NXP FRDM-K64F. It can be considered as the brain of the hardware since it integrates sensors to the system, collects data and sends it to cloud via ESP8266 wifi module. It also contains features like fail-safe, self-diagnosis etc.
2. Power Circuit – The system is powered by a 24 Volt DC adapter. This provides power to sensors via the interface circuit and it also provides power to the microcontroller through a voltage regulator circuit since microcontroller works on 5 V DC.
3. SD Card – The microcontroller has a slot for SD card. In this SD card, a configuration file is stored which provides information to the microcontroller regarding the sensors connected to the system. This SD card is also used to temporarily store data in case there is an internet failure.
4. Actuator Circuit – The system has the capability of controlling equipment connected to it. It can turn on/off the equipment using a relay circuit and a power outlet. This controlling signals are received and processed by the microcontroller.
5. ESP 8266 – This is a wifi module that allows the microcontroller to connect to the internet and send the sensor data. It also receives messages and passes it on to the microcontroller, signals for actuation.

6. Interface Circuit – This is a universal connector that allows the sensors to connect to the system. It is basically a patch cable with color codes which needs to be interfaced to the sensor in order for it to work with the system.
7. Connection – This is the socket that allows the sensors to plug its wire into. It is an RJ – 45 connector.

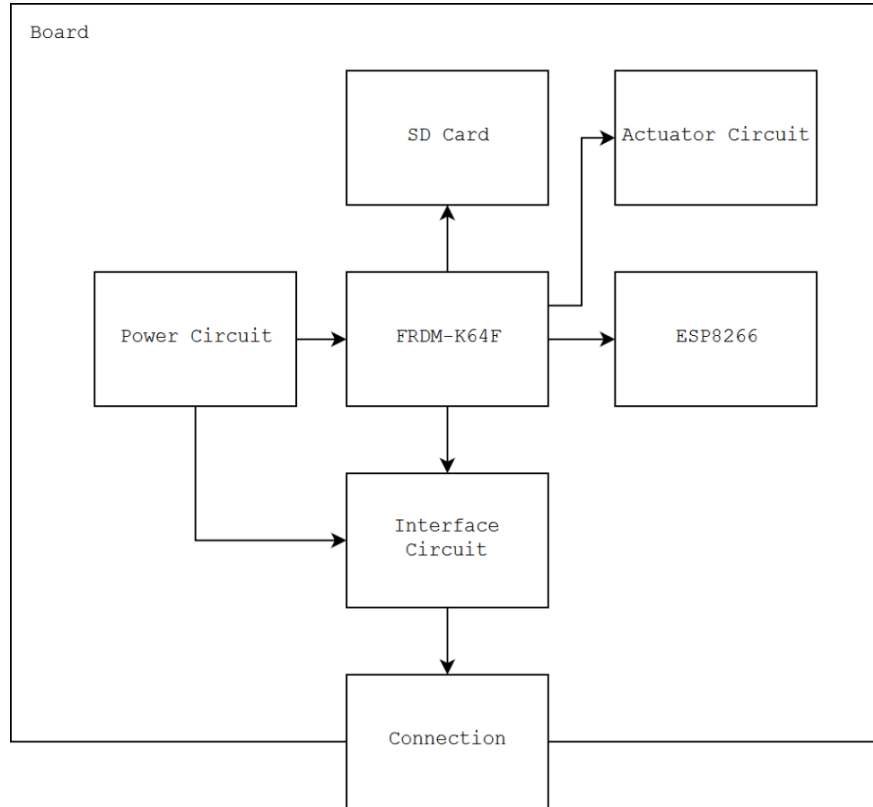


Figure A.5. Board Outline

A.10 Data Storage

The SD card is used to store the backup information of the sensor reads. The system logs backup data to an SD card in case of internet failure. Figure A.6 represents the outline for the back up to SD card. If data fails to send from microcontroller to Web server, the microcontroller logs data to SD card. If data is sent successfully to the web server, the microcontroller will check if SD card has any backup data to be sent. If data exists in SD Card, microcontroller will send it to web server.

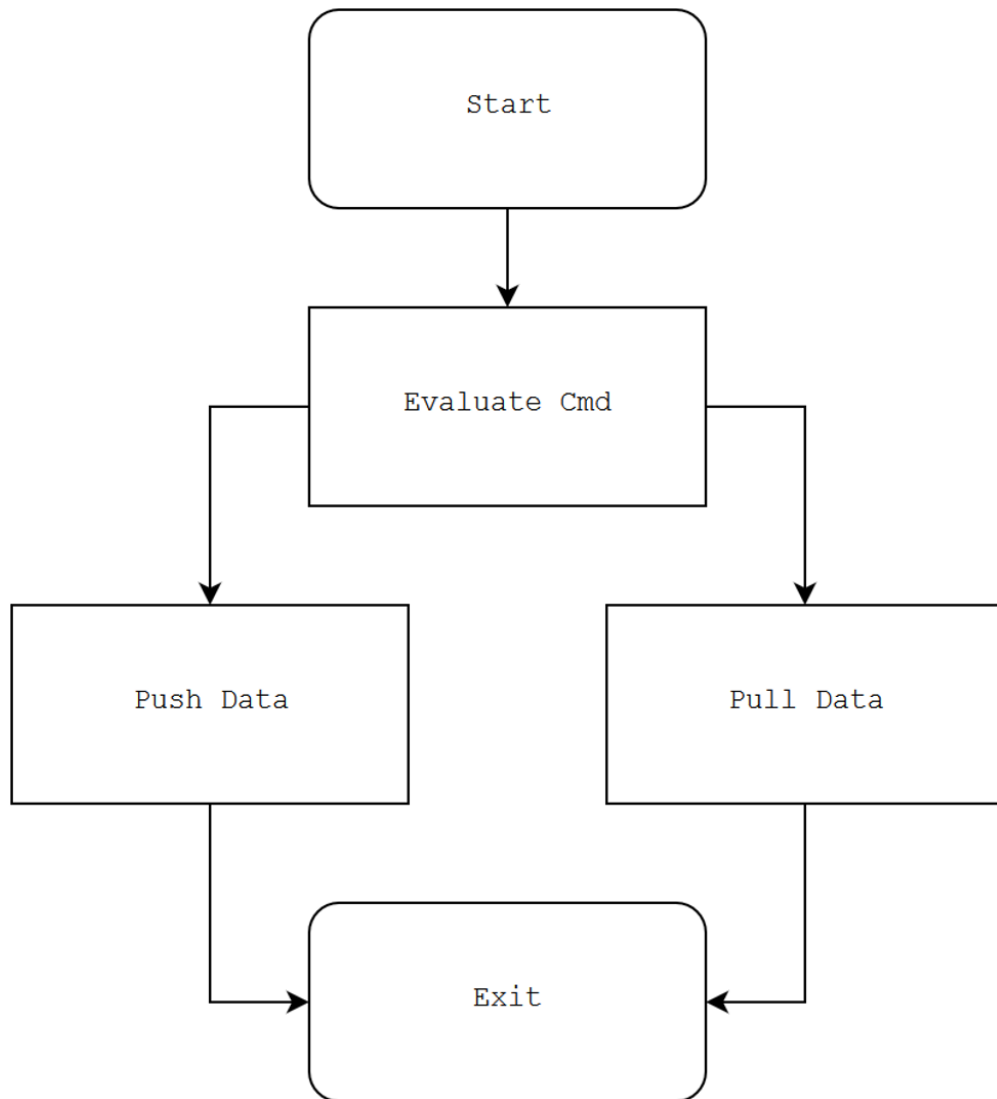


Figure A.6. Log Outline

B. APPENDIX

B.1 Component List

Table B.1. Component List

Qty	Value	Package	Manufacturer	Manufacturer Code	Description
10	CB1A-P-12V	520243-3	FARNELL	MHRJJ66NFV	MH CONNECTORS - MHRJJ66NFV - RJ12 MODULAR, JACK, 6 POSITION, 1PORT
2		MA08-2	MOLEX	10-89-7161	MOLEX - 10-89-7161 - BOARD-BOARD CONNECTOR, HEADER, 16 POSITION, 2ROW
1		MA10-2		2213S-20G	MULTICOMP - 2213S-20G - HEADER, THROUGH-HOLE, VERTICAL, 2.54MM, 20 POSITION
4		CB1	ARO-MAT/ MAT-SUSHITA	CB1A-P-12V	PANASONIC ELECTRIC WORKS - CB1A-P-12V - RELAY, AUTOMOTIVE, SPST-NO, 14VDC, 40A
1		61000-413321	TYCO ELECTRONICS	2-640497-4	TE CONNECTIVITY - 2-640497-4 - PLUG & SOCKET CONNECTOR, HEADER, 2 POSITION, 4.2MM
4		2N3906		2N3906	MULTICOMP - 2N3906 - BIPOLAR TRANSISTOR, PNP, -40V TO-92

Table B.1 continued from previous page

14	2K	0204/7		MFR4-2K0FI	WELWYN - MFR4-2K0FI - METAL FILM RESISTOR, 2KOHM, 500mW, 1%
10	985	0204/7		HW210	NTE ELECTRONICS - HW210 - RESISTOR, METAL OXIDE, 1KOHM, 500mW, 2%
10	1.7K	0204/7		MFR4-1K5FI	WELWYN - MFR4-1K5FI - METAL FILM RESISTOR, 1.5KOHM, 500mW, 1%
10	3.3K	0204/7	FARNELL	MCF 0.25W 3K3	MULTICOMP - MCF 0.25W 3K3 - CARBON FILM RESISTOR, 3.3KOHM, 250mW, 5%
20	82	0204/7		PR0200020 8209JR500	VISHAY - PR02000208209JR500 - METAL FILM RESISTOR, 82 OHM, 2 W, 5%
4	50	0204/7	Multicomp Passives	MC14709	MULTICOMP - MC14709 - WIRE-WOUND RESISTOR, 50 OHM, 5W, 1%
1		MA06-2		2213S-06G	MULTICOMP - 2213S-06G - HEADER, 2 ROW, VERTICAL, 6 POSITION

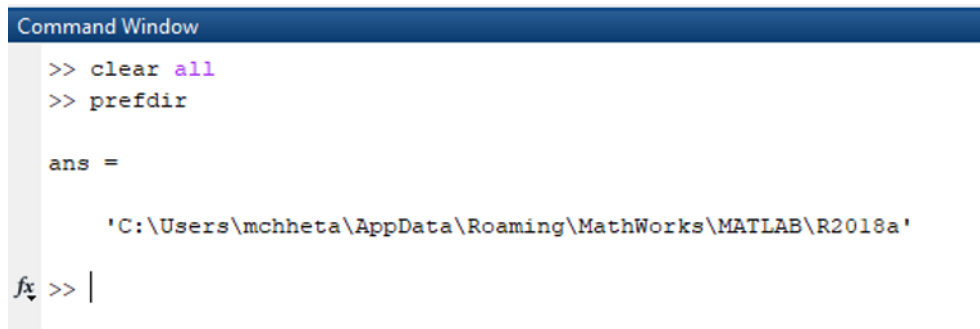
Table B.1 continued from previous page

1		MA04-2		2213S-08G	MULTICOMP - 2213S-08G - HEADER, THROUGH-HOLE, VERTICAL, 2.54MM, 8 POSITION
10		SLIDE SWITCH		STS1400PC04	ALCOSWITCH - TE CONNECTIVITY - STS1400PC04 - SLIDE SWITCH, SP4T, 0.25A, 125V, THD
5	STE- REO JACK	STX3100		FC68133	CLIFF ELECTRONIC COMPONENTS - FC68133 - STEREO JACK, 3.5MM, 5POS, PCB
1	step- down	D24V6F5	POLOLU	D24V6F5	https://www.pololu.com/product/2107/specs
5		ESP8266	SparkFun Electronics	1568-WRL- 17146-ND	WIFI MODULE - ESP8266
5		FRDM- K64F	NXP USA Inc.	FRDM-K64F- ND	K24, K63, K64, mbed-Enabled De- velopment Freedom Kinetis ARM® Cor- tex®-M4 MCU 32-Bit Embedded Evaluation Board

C. APPENDIX

C.1 Accessing Database from Matlab Command Line

1. Have matlab installed.
2. Download Database Explorer toolbox from matlab add-ons.
3. Download the JDBC connector file from the given link
<https://iu.box.com/s/liall8m6nppde6fa310cbb078bvsqlvp>
4. Extract the file and save it in a safe place where you will not delete it.
(I saved it in downloads).
5. Open Matlab and run the command `prefdir`.



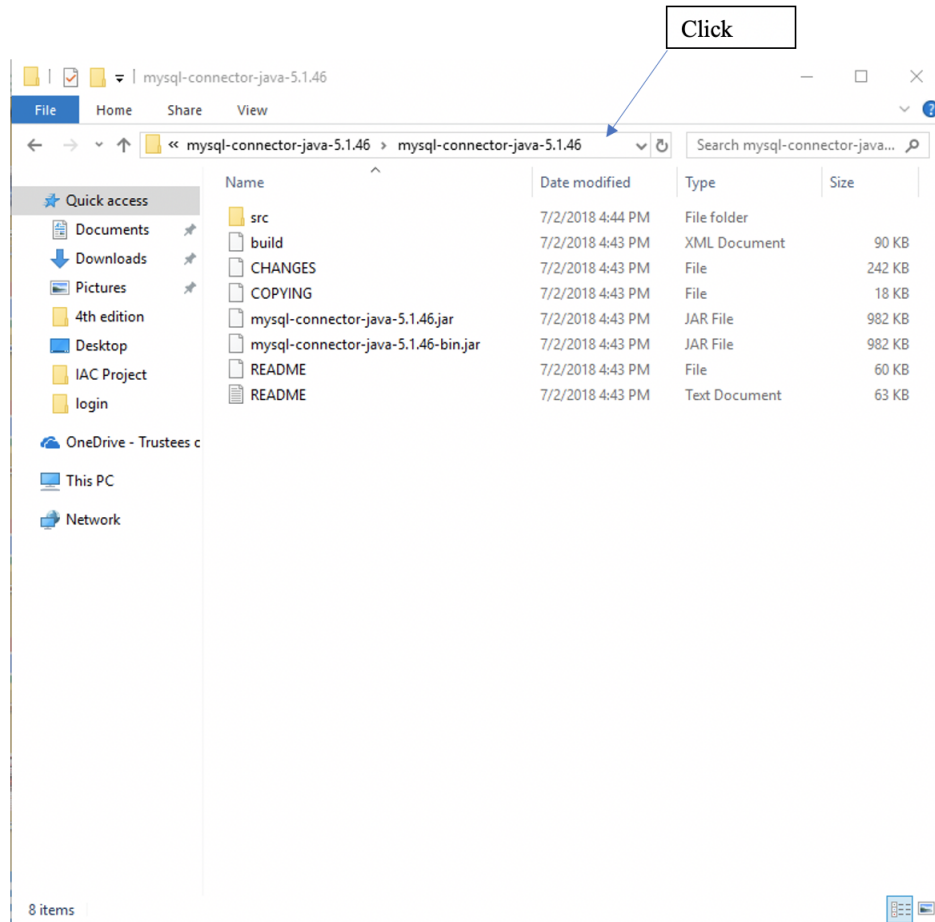
```
Command Window
>> clear all
>> prefdir

ans =

'C:\Users\mchheta\AppData\Roaming\MathWorks\MATLAB\R2018a'

fx >> |
```

6. Go to the path shown in the result and create a text file named `javaclasspath` and save it.
7. Navigate to the place where you stored the downloaded mysql connector file.



You should be able to see a path address. Copy that. In my case it is:

C: \Users \mchheta \Downloads \mysql-connector-java-5.1.46 \mysql-connector-java-5.1.46

Edit the in javaclasspath text file in

C: \Users \<username> \AppData \Roaming \Mathworks \Matlab \R2018a.

Copy the highlighted part and paste it in javaclasspath file twice. At the end of first line add '\ ' and 'mysql-connector-java-5.1.46.jar'. (without the quotes) At the end of second line add '\ ' and 'mysql-connector-java-5.1.46-bin.jar' Now the content of the javaclasspath file should look like this



8. Save the file and close it.
9. To establish connection via command line enter the following commands in the command window.

```
prefs = setdbprefs('DataReturnFormat');
setdbprefs('DataReturnFormat','table')
conn=database('iumri_ME','iumri_MEuser','123456abcd','Vendor','MYSQL','Server','sasrdsm01.uits.iu.edu','PortNumber',3306);
```

10. Now the connection to the database is established and we can start fetching data.
11. For example here I am fetching data of the users who have registered and storing it in a variable called data in workspace.

```
curs = exec(conn,['SELECT * ' 'FROM iumri_me.users']);
curs = fetch(curs);
data = curs.Data;
close(curs)
```

12. The fetched data will be visible in the workspace variable “data” created in step 11.

	1	2	3	4	5	6
	id	username	password	created_at		
1	1	'monil.chheta'	'\$2y\$10\$LLq...	'2018-04-26 0...		
2	2	'tanmay.zant...	'\$2y\$10\$4xy...	'2018-04-26 0...		
3	3	'tanmayzant...	'\$2y\$10\$my...	'2018-04-26 1...		
4	4	'IAC'	'\$2y\$10\$olt...	'2018-05-03 1...		
5						
6						
7						
8						

13. Be sure to close the connection once you are done using the command window. The Command is

```
%% Close connection to database
close(conn)

%% Restore preferences
setdbprefs('DataReturnFormat',prefs)

%% Clear variables
clear prefs conn curs
```

Sample of the Entire Code

Using a “.m” file, the whole code explained above, can be executed in a single “.m” file instead of typing individually in the command window.

```
%% Set preferences
prefs = setdbprefs('DataReturnFormat');
setdbprefs('DataReturnFormat','table')

%% Make connection to database
conn =
database('iumri_ME','iumri_MEuser','123456abcd','Vendor','MYSQL','Server','sa
srdsmp01.uits.iu.edu','PortNumber',3306);

%% Execute query and fetch results
curs = exec(conn,['SELECT * ' 'FROM iumri_me.users']);
curs = fetch(curs);
data = curs.Data;
close(curs)

%% Close connection to database
close(conn)

%% Restore preferences
setdbprefs('DataReturnFormat',prefs)

%% Clear variables
clear prefs conn curs
```

D. APPENDIX

D.1 Using MySQL Workbench

MySQL workbench software is used for database development and implementation.

In order to login, developer needs to enter information below:

1. Hostname
2. Port
3. User name
4. Password

Having this information, developer can login by filling out the details as shown in the figure below.

The screenshot shows the 'Setup New Connection' window in MySQL Workbench. It features a 'Parameters' tab with the following fields and values: 'Connection Name' (empty), 'Connection Method' (Standard (TCP/IP)), 'Hostname' (127.0.0.1), 'Port' (3306), 'Username' (root), 'Password' (empty with 'Store in Keychain ...' and 'Clear' buttons), and 'Default Schema' (empty). Each field has a tooltip explaining its purpose. The bottom of the window contains buttons for 'Configure Server Management...', 'Test Connection', 'Cancel', and 'OK'.

1. Create table Using MySQL queries, developer can easily create tables into the database. Example query to create a table is shown in Figure below.


```

1  CREATE TABLE `iac`.`sensor_Readings` (
2      `TimeStamp` DATETIME NOT NULL,
3      `Board_ID` VARCHAR(45) NULL,
4      `Port_ID` VARCHAR(45) NULL,
5      `Value` VARCHAR(45) NULL,
6      PRIMARY KEY (`TimeStamp`));
7



```

2. View table values Using MySQL queries, developer can see the data in the database.
As shown in figure below, the query displays the information from Company Table.

```
1 • | SELECT * FROM newschema.Company;
```

100% 1:1

Result Grid

  Filter Rows:

company_ID	company_Name	company_Address
1	IAC	799 W Michigan St IN 46202
2	IUPUI	723 W Michigan St IN 46202

Similarly, user can Update, delete or drop tables with the correct knowledge of database queries.