# Smart Home Automation

*Major Project Report*


**National Institute of Technology, Kurukshetra**

**Submitted By:**

**Armaan Dhillon**

Roll No: 12114058

Department of Electrical Engineering

**Submitted To:**

**Dr. Monika Mittal**

Department of Electrical Engineering

January 2025 - May 2025

**Electrical Engineering Department**

**National Institute of Technology Kurukshetra**

# CANDIDATE DECLARATION

---

I hereby declare that the work presented in this report, **"Smart Home Automation"**, is an original work that was completed under the supervision of **Dr. Monika Mittal** and submitted to the **Department of Electrical Engineering**.

I have not submitted the work described in this dissertation for the grant of any other degree from this or any other institute. I have respected intellectual property rights in every possible way and have assisted others in using them for academic purposes.

I shall be held responsible for any complete or partial copyright violation or intellectual property rights violation discovered at any time after the award of my degree, and not my Supervisor/-Head of the Department/Institute.

Armaan Dhillon

12114058

**Electrical Engineering Department**

**National Institute of Technology Kurukshetra**

# CERTIFICATE

---

Certified that the work described by Mr. **Armaan Dhillon** (Roll No. **12114058**), **"Smart Home Automation"**,was completed under my supervision during the **8th Semester**. According to our knowledge, this work has not been submitted for a degree elsewhere.

**Dr. Monika Mittal**

Associate Professor

NIT Kurukshetra

# ACKNOWLEDGMENTS

My heartfelt gratitude goes out to my supervisor, **Dr. Monika Mittal**, for her helpful assistance and unwavering support throughout this project. Her technical expertise, accurate recommendations, and timely guidance were invaluable. Her wisdom and insight provided me with a great deal of motivation.

Additionally, I would like to express my sincere gratitude to **Prof. B.V. Rama Reddy**, Director of NIT Kurukshetra, and **Prof. Jyoti Ohri**, Head of the Department, for creating a conducive environment and inspiring us to pursue constructive research efforts.

I also want to express my gratitude to all the instructors and staff at the NIT Kurukshetra Electrical Engineering Department for their invaluable assistance and support.

I would like to extend my sincere gratitude to my colleagues and seniors for their valuable suggestions and encouragement, which helped me complete my thesis work.

**Abstract**

Home automation utilizing Raspberry Pi has grown in popularity due to its low cost and capacity to improve convenience and energy efficiency. However, many existing home automation systems suffer severe issues, such as inadequate security, restricted interoperability, lengthy configuration processes, and insufficient real-time energy monitoring. This project overcomes these restrictions by creating a smart home automation system that combines Raspberry Pi, ThingsBoard, Docker, and real-time sensor data to deliver a safe, modular, and scalable solution.

The system collects energy usage data from sensors linked to a smart energy meter and delivers it to a cloud-based IoT platform using MQTT and REST APIs. ThingsBoard enables real-time visualization and control via customisable dashboards, while Docker ensures modular deployment and service separation. The use of encrypted communication protocols improves security, while the open-source, protocol-agnostic architecture allows for smooth integration of numerous devices. This project provides a realistic and efficient solution to modern home automation by simplifying system setup and enabling real-time monitoring and automation, resulting in improved energy management and user accessibility.

# Contents

# List of Figures

# Chapter 1

# Introduction

## 1.1 Introduction

Home automation using Raspberry Pi has gained popularity due to its numerous advantages and cost-effectiveness. These systems provide users with the ability to control household appliances through local networks or remote access, thereby enhancing convenience and energy efficiency[1]. Modern home automation technologies offer features such as automatic meter reading, real-time monitoring, and remote control of electrical connections without the need for personal involvement[2]. In addition to energy monitoring, home automation systems commonly include sensors, cameras, and web-based applications to increase security and enable comprehensive device control[3]. The MQTT protocol, known for its lightweight and reliable messaging, is frequently used in IoT-based systems to improve data quality and communication reliability.

Despite these developments, there are still certain limits in existing home automation options. Many systems continue to have insufficient encryption and access control, leaving them vulnerable to illegal access. Proprietary communication protocols and restricted integration possibilities further impede interoperability across devices from different manufacturers. Furthermore, extensive setup processes and non-intuitive user interfaces make it difficult for non-technical individuals to access and operate.

This project addresses these challenges by creating a smart home automation system using Raspberry Pi, Docker, ThingsBoard, and real-time sensor data. The use of encrypted MQTT assures safe communication, while Docker containers enable modular deployment and service isolation. ThingsBoard supports protocol-agnostic integration and user-friendly dashboards, easing system maintenance and customization. The system's real-time monitoring and automa-

tion aims to increase energy efficiency, user experience, and support for a scalable, cost-effective, and open-source approach to home automation.

## 1.2 Problem Statement

Smart home automation has received a lot of interest in recent years because of its ability to improve energy efficiency, increase convenience, and enable cognitive management over household appliances. Despite significant development in this area, current systems suffer a number of ongoing problems that restrict their effectiveness and widespread acceptance.

One of the primary considerations is security. Existing systems frequently use inadequate encryption standards and lack effective access control measures. This makes the system susceptible to unwanted access, data breaches, and manipulation of essential home systems. Furthermore, interoperability presents a substantial challenge. Many commercial smart home solutions use proprietary communication protocols and closed designs, making it impossible to integrate devices from other suppliers or expand the system without encountering compatibility concerns.

Another important factor to consider is user experience. Many smart home systems are difficult to set up and provide little customization, discouraging non-technical consumers from embracing and efficiently utilizing new technologies. Finally, energy efficiency remains a significant concern in many systems. Limited support for real-time monitoring and control of energy consumption frequently leads to wasteful power consumption and higher operational costs.

## 1.3 Proposed Solution

To overcome these issues, a smart home automation solution was created that combines Raspberry Pi, ThingsBoard, Docker, and real-time sensor data. This system seeks to be affordable, secure, flexible, and adaptable to a wide range of residential situations.

The Raspberry Pi is the primary hardware platform, operating as a local controller and managing edge computing duties. It supports sensor data gathering and communication with connected devices. ThingsBoard, an open-source IoT platform, is used for device administration, real-time data visualization, and building interactive dashboards. This platform supports several protocols and enables for simple customisation, addressing interoperability and user experience issues.

Docker uses containerization to isolate services, improve system security, and simplify deployment and upgrades. Each service operates in its own container, making the system more flexible and simple to manage or scale. Real-time sensor integration enables continuous monitoring of environmental conditions and energy consumption, resulting in automation and increased energy efficiency.

From a security standpoint, encrypted MQTT protocols are employed for data transmission, establishing a safe communication channel between devices and the cloud platform. Docker-based service isolation provides an additional degree of safety, reducing possible risks from system flaws.

In terms of interoperability, the system is intended to be protocol-agnostic and completely open source. This enables the smooth integration of several devices, independent of manufacturer, and encourages vendor independence. To improve the user experience, the platform includes visual dashboards with drag-and-drop widgets, allowing users to design and monitor their smart home environment with little technical knowledge.

Real-time sensor data enables dynamic decision-making, allowing the system to respond immediately to changes in ambient conditions. This improves energy efficiency by enabling context-aware automation, such as changing lighting or shutting off equipment when not in use.

## 1.4 Project Workflow

This section depicts the workflow of a smart home automation system that is coupled with blockchain for energy billing. The smart meter captures energy usage data, which is then transmitted to a cloud-based IoT platform for real-time monitoring[1][2]. Data is processed using MQTT and REST APIs to ensure reliability.

A Web3 application obtains this information and communicates with an Ethereum smart contract to generate energy bills in a secure and transparent manner[4][5]. By merging IoT and blockchain, the solution automates billing and payments, increasing efficiency and security.

### 1.4.1 System Architecture

The architecture consists of multiple interconnected components, including sensors, microcontrollers, cloud platforms, and blockchain technology. The data flow and interactions are illustrated in Figure 1.1.

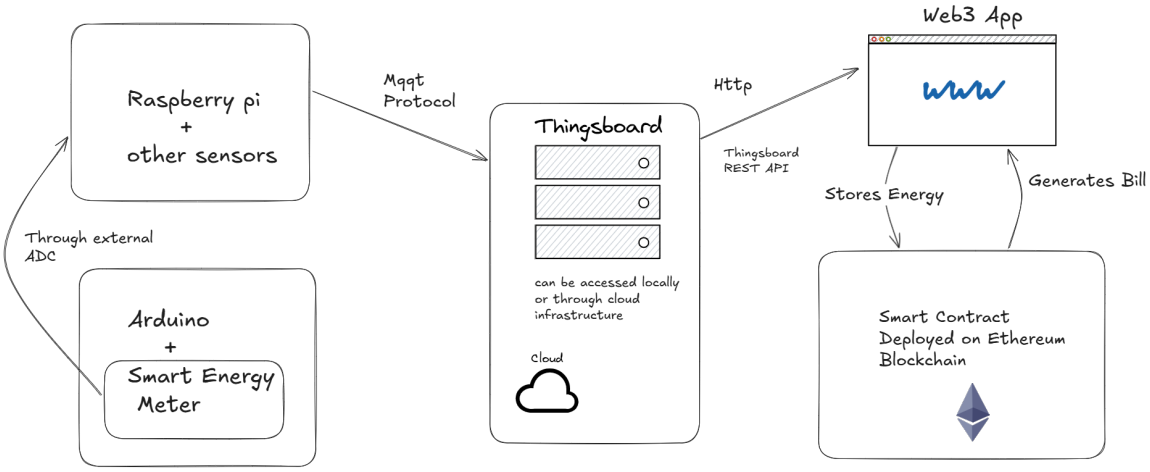


Figure 1.1: System Workflow

### 1.4.2 Components and Data Flow

**Energy Measurement and Data Acquisition:** The system starts with an Arduino connected to a smart energy meter, which measures power consumption. Since Arduino lacks a built-in analog-to-digital converter (ADC) with the required resolution, an external ADC is used for accurate readings. The measured data is then transmitted to a Raspberry Pi for further processing.

**Data Transmission and Storage:** This section illustrates the workflow of a smart home automation system that is integrated with blockchain for energy billing. The system's goal is to collect energy consumption data, store it in a cloud-based IoT platform, and bill and pay using a Web3 application that interacts with an Ethereum blockchain smart contract.

**Web3 Integration and Smart Contracts:** A Web3 application fetches energy consumption data from ThingsBoard via HTTP requests. The retrieved energy data is then stored in a smart contract deployed on the Ethereum blockchain. The smart contract automatically generates an energy bill based on the stored consumption values.

**Billing and Payment System:** Using HTTP queries, a Web3 application retrieves energy usage data from ThingsBoard. A smart contract that is implemented on the Ethereum blockchain then stores the energy data that has been recovered. Using the saved consumption values, the smart contract automatically creates an energy bill.

## 1.5 Conclusion

This process combines blockchain, cloud computing, and the Internet of Things to produce an automated and decentralized energy billing system. An effective and transparent billing procedure is made possible by the modular architecture, which facilitates smooth data flow from energy monitoring to smart contract interactions.

# Chapter 2

# Literature Survey

## 2.1 Introduction

Smart home automation has become a vital part of modern living, offering benefits such as increased convenience, energy optimization, and enhanced security. With the advancement of Internet of Things (IoT) technologies, homes are now equipped with interconnected sensors, controllers, and cloud platforms that enable real-time automation and monitoring. However, as noted in numerous studies, the journey toward widespread adoption remains hindered by technical, financial, and usability-related challenges[6][7].

While these technologies show promise, real-world implementations frequently fall short due to constraints such as inadequate device interoperability, security risks, a lack of customisation, and exorbitant installation costs. Existing systems sometimes rely on closed ecosystems, further complicating integration and limiting user flexibility. This chapter explores major literature in the field, identifies the primary barriers to smart home automation, and compares those results to our system's implementation, which uses Raspberry Pi, ThingsBoard, Docker, and real-time sensor-cloud connection.

## 2.2 Overview of the Research Domain

The smart home ecosystem comprises a diverse set of devices and platforms that collaborate to automate household operations. Temperature control, lighting, security systems, and energy monitoring are all examples of IoT-enabled applications. As seen in recent studies, the integration of different sensors and platforms brings both functional potential and additional complications[8].

Security breaches, data privacy issues, device incompatibilities, and poor user engagement remain significant impediments to adoption. Although open-source systems and current edge computing devices such as the Raspberry Pi provide promising possibilities, much of the research focuses on systems that are either highly centralized or dispersed due to proprietary constraints. This assessment divides the main challenges into four categories: security, interoperability, user experience, and energy efficiency.

## 2.3 Review of Existing Work

### 2.3.1 Security Concerns

Security in smart home automation is a serious concern raised in various research. As the number of internet-connected gadgets grows, so does the attack surface for cyber attacks. According to multiple sources, many IoT devices lack strong encryption and authentication, leaving them vulnerable to unauthorized access[8][9]. Others raise worries about user data being collected or modified owing to inadequate or nonexistent privacy protections[10]. Furthermore, the lack of common security methods across different manufacturers increases these risks[11][12].

### 2.3.2 Interoperability Issues

Interoperability refers to the ability of equipment from various vendors to communicate seamlessly. According to research, many smart home systems fail to integrate effectively because they use proprietary communication protocols.Incompatibility between platforms frequently causes users to rely on vendor-specific hubs or apps, resulting in a disjointed user experience. This difficulty is exacerbated by the lack of globally accepted IoT standards[12].

### 2.3.3 User Experience

Despite their functional potential, smart home systems can provide a terrible user experience. According to studies, setup methods are typically technically complex, preventing non-expert users[13][14]. Other findings indicate that many systems fail to tailor interactions depending on user behavior, leaving automation seeming stiff and detached[15]. Furthermore, non-intuitive interfaces and a lack of transparency increase user irritation and slow adoption.

### 2.3.4 Energy Efficiency

Energy efficiency is usually highlighted as a significant motivator for smart home adoption, however research shows that many systems fall short of meeting this expectation. Several studies address how real-time energy monitoring and management are underutilized due to the high system complexity[13]. Others point out that balancing user comfort with energy-saving automation is a continuing challenge[15], and that consumers frequently lack awareness into their own consumption habits[16].

## 2.4 Research Gaps and Motivation

While the present literature thoroughly examines the theoretical and technological underpinnings of smart home automation, the majority of solutions are either too expensive or do not provide end-to-end capability. Systems that rely significantly on centralized cloud services are vulnerable to latency, data breaches, and platform dependencies. On the other hand, decentralized systems frequently lack the coordination and integration needed for comprehensive automation.

The proposed project was motivated by the need for a modular, cost-effective, and fully adaptable system that could be readily installed and maintained. The project takes a balanced approach, using a Raspberry Pi as an edge controller, ThingsBoard for visualization and device management, and Docker for scalable deployment, to bridge the gap between academic research and real-world applications.

## 2.5 Comparative Analysis with the Proposed Solution

The proposed smart home automation solution utilizes Raspberry Pi, ThingsBoard, Docker, and real-time sensor data integration to address limitations identified in existing literature. By leveraging open-source and containerized technologies, it aims to offer a low-cost, secure, and modular framework adaptable to varied use cases. Table 2.1 summarizes how this solution compares with the challenges found in previous research.

| Challenge | Literature Observations | Proposed Solution |
|---|---|---|
| Security | Weak encryption, poor access control | Encrypted MQTT, Docker-based isolation |
| Interoperability | Proprietary protocols, poor integration | Open-source, protocol-agnostic stack |
| User Experience | Complex setup, limited personalization | Visual dashboards, easy configuration |
| Energy Efficiency | Limited real-time monitoring | Live sensor data, extensible dashboards |

Table 2.1: Comparison of Literature Challenges and Proposed Solution

## 2.6 Conclusion

Smart home automation is always evolving, with substantial advances in device capabilities, cloud integration, and automation intelligence. However, difficulties such as security vulnerabilities, device compatibility, usability, and energy management continue to impede large-scale adoption. Existing literature identifies these difficulties, but often lacks practical remedies.
This project provides a comprehensive, technically competent response to these difficulties. It provides a realistic framework that corresponds with academic research concepts while boosting scalability, dependability, and user empowerment by integrating open-source platforms, containerized infrastructure, and real-time sensor-cloud communication.

# Chapter 3

# Proteus Schematic Design and Simulation

## 3.1 Introduction

Proteus is a powerful software program that is commonly used to develop and simulate embedded systems prior to their real implementation. It allows you to efficiently test circuits, detect design problems, and enhance performance without the need for actual hardware. This chapter focuses on the schematic design and modeling of a smart home automation system and a smart energy meter, which integrate sensors, actuators, and communication modules to provide real-time monitoring and control.

The Raspberry Pi 3 serves as the core processing unit for the smart home automation system, which communicates with a variety of sensors such as Light Dependent Resistors (LDR), Passive Infrared (PIR) sensors, and gas detectors to automate household appliances. The Raspberry Pi lacks built-in analog input capabilities, thus an external ADC (MCP3208) is used to interpret analog sensor data and enabling IoT-based automation[17].

The smart energy meter focuses on real-time power consumption monitoring, incorporating an Arduino Uno for data acquisition, ACS712 current sensors for current measurement, and voltage dividers for accurate voltage monitoring. Additionally, power factor measurement is achieved using LM358 operational amplifiers and XOR gates to analyze phase differences between voltage and current waveforms. These measurements are transmitted to the IoT platform, enabling remote energy management and optimization.

This chapter details the schematic design, component selection, and software implementation in Proteus. The proposed system provides not just automation and remote monitoring, but

also efficient energy utilization and billing. Proteus simulations evaluate the system's many features prior to real-world deployment, ensuring reliability and performance.

## 3.2 Schematic Design in Proteus

### 3.2.1 Smart Home Automation System

The home automation circuit includes:

**Raspberry Pi 3:** The Raspberry Pi 3 B+ was chosen for this home automation project because of its powerful capabilities and simplicity of integration. It is driven by a Broadcom BCM2837B0 quad-core Cortex-A53 (ARMv8) processor clocked at 1.4GHz, with 1GB LPDDR2 SDRAM and a Broadcom Videocore-IV GPU. Networking options include Gigabit Ethernet (by USB), dual-band Wi-Fi (2.4GHz and 5GHz 802.11b/g/n/ac), and Bluetooth 4.2 (BLE)[17].It includes a 40-pin GPIO header, HDMI, a 3.5mm audio connector, four USB 2.0 ports, Ethernet, CSI, and DSI. The Micro-SD storage format and compact dimensions (82mm × 56mm × 19.5mm, 50g) make it ideal for embedded applications[17].

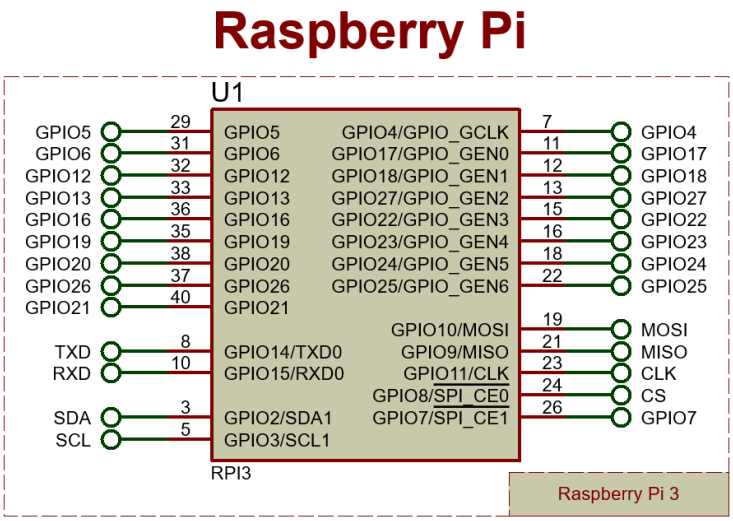


Figure 3.1: Raspberry Pi 3 Schematic

**ADC (MCP3208):** Analog-to-digital converters (ADCs) transform analog signals into digital data, allowing microcontrollers and computers to process real-world inputs.The MCP3208 is a 12-bit ADC with 8 input channels. It provides high accuracy (±1 LSB DNL, ±1-2 LSB INL) and low power consumption. It runs on 2.7V-5.5V, has an SPI interface, and is commonly used for sensor applications and data gathering.

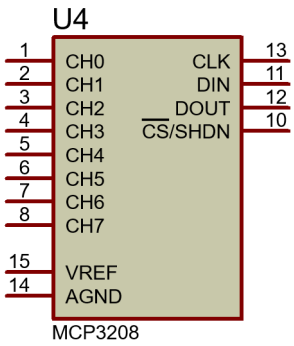


Figure 3.2: ADC (MCP3208)

Because Raspberry Pi lacks built-in analog inputs, an external ADC such as the MCP3208 is required to read analog sensor data. Interfacing via SPI allows for real-time monitoring in IoT applications and industrial automation[18].

**Other Sensors:** We used a variety of sensors and actuators to allow for sophisticated control and monitoring. The system uses an LDR (Light Dependent Resistor) to detect ambient light levels, allowing lights to be turned on or off based on the brightness of the surroundings. A PIR (Passive Infrared) sensor was used to detect motion and human presence. This sensor is essential for security and automated lighting because it activates lights or alerts when motion is detected. Furthermore, a MQ-2 gas sensor was used to monitor air quality and identify combustible gases such as LPG and CO, assuring safety by sending alarms in the event of a gas leak. The LM35 temperature sensor was included to properly monitor room temperature and allow for automatic cooling control.

### 3.2.2 Complete Schematic of Smart Home Automation System

The smart home automation circuit integrates multiple sensors and actuators to automate home appliances based on environmental conditions ans involves following connections.

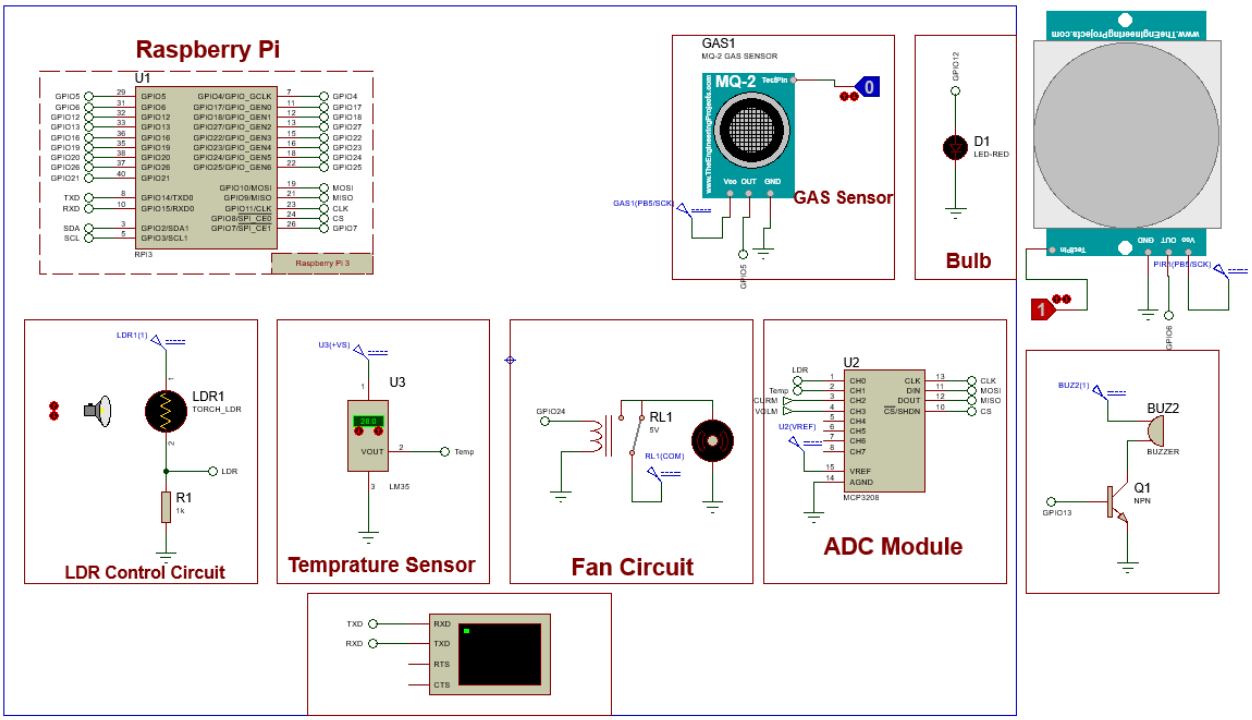


Figure 3.3: Complete Schematic

### 3.2.3 Smart Energy Meter and Power Factor Measurement

The smart energy meter circuit consists of:

**Arduino Uno:**The Arduino Uno, built around the AVR microcontroller, has developed as a versatile platform for teaching and implementing digital control systems.

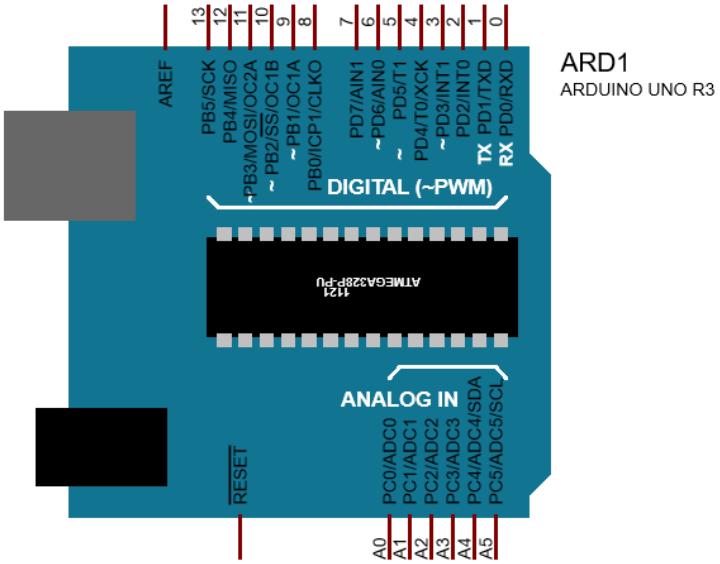


Figure 3.4: Arduino Uno

It has characteristics suited for power electronics applications, providing switching frequencies

of up to 100 kHz with extra libraries[19]. The Arduino Uno has been shown to be effective in controlling industrial-scale instruments, performing similarly to industrial-class controllers that use PID algorithms[20].

**Current Measurement**:In our smart home automation project, we employ the ACS712 Hall-effect-based current sensor to monitor AC/DC power levels. It has low resistance (1.2 mohm) and 2.1 kVRMS isolation for precise and safe current measuring. The 30A variant with 66mV/A sensitivity is integrated with an arduino uno, allowing for real-time power tracking and energy consumption analysis via ThingsBoard. Calibration improves accuracy and reduces measurement error, making it ideal for IoT-based billing and automation systems. Its rapid reaction (5 µs) and low noise improve performance for smart energy management.

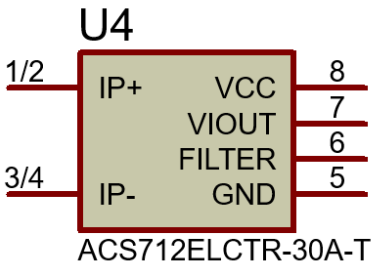


Figure 3.5: Current Sensor

**Voltage Measurement:**We monitor AC voltage in our smart home automation project with a step-down transformer, a voltage divider network, and a filtering capacitor. The step-down transformer (TR3) converts the high AC mains voltage to a lower, safer level suited for measurement. The voltage divider circuit (R9, R10, R11, and R12) reduces the voltage to a range compatible with the Arduino ADC, ensuring precise readings while protecting the microcontroller. A 500µF capacitor (C2) filters, reduces noise, and stabilizes the signal for accurate RMS voltage measurement. This configuration enables the Arduino to process voltage data and send it to ThingsBoard, allowing for real-time monitoring and effective energy management in our smart home system.

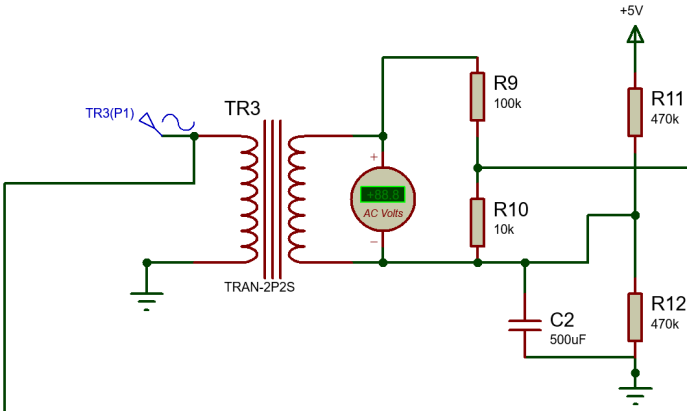


Figure 3.6: Voltage Measurement Circuit

**Power Factor Measurement:** we monitor power factor with LM358 operational amplifiers and an XOR gate. The LM358 op-amps act as zero-crossing detectors, transforming AC voltage and current waveforms to square waves. The first op-amp (U8:A) detects zero crossings in the voltage waveform, whereas the second op-amp (U8:B) detects current waveform. These processed signals are then routed through an XOR gate (U9), which generates a pulse-width output proportional to the phase difference between the voltage and current waves. The Arduino measures pulse width and calculates power factor as $\cos(\theta)$, where $\theta$ is the phase angle.

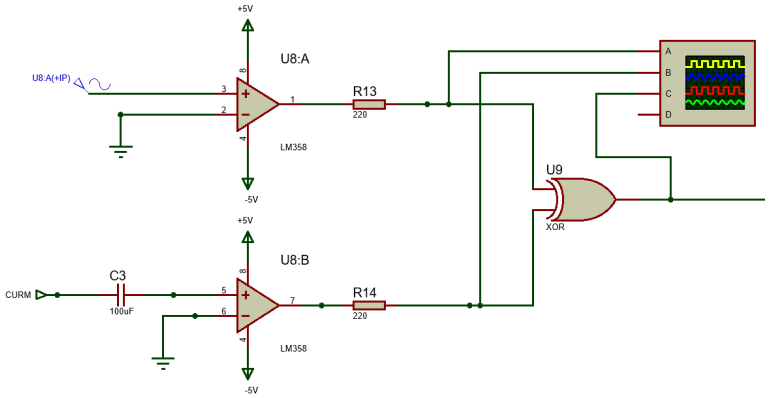


Figure 3.7: PowerFactor Measurement Circuit

**relay-based load control** electrical appliances are managed remotely using a relay control system. The circuit consists of electromagnetic relays (RL4, RL5) that are controlled by the microcontroller's GPIO pins (for example, Raspberry Pi or Arduino). When a GPIO pin is set to HIGH, the accompanying relay coil is activated, shutting the switch and enabling current to flow to the attached load (such as lights or motors).

### 3.2.4 Complete Schematic of Smart Energy Meter

The smart energy meter circuit is designed to measure and monitor power consumption while ensuring efficient energy usage.

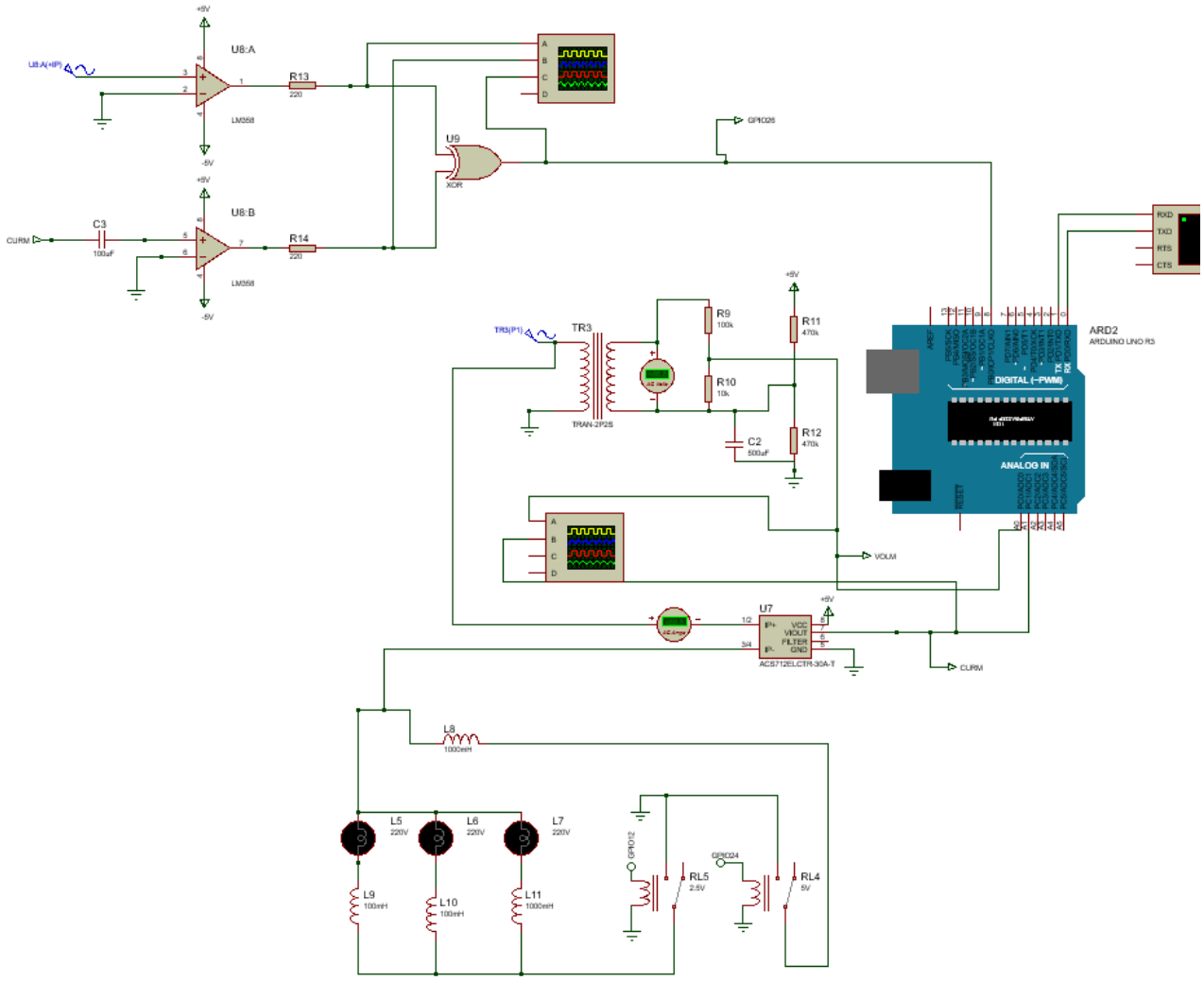


Figure 3.8: Complete Schematic of Smart Energy meter

## 3.3 Code Implementation

### 3.3.1 Arduino Code for Smart Energy Meter

The Arduino is responsible for measuring voltage, current, and calculating power consumption in real-time. The analog pins read voltage and current sensor outputs, which are then converted into meaningful electrical values. The calculated power is then transmitted over serial communication.

```
// Pin configuration
const int voltagePin = A0; // Voltage sensor input
const int currentPin = A1; // Current sensor input

// Constants for sensor calibration
const float voltageMultiplier = 230.0 / 1023.0;  // Adjust based on sensor
    specs
const float currentMultiplier = 10.0 / 1023.0;   // Adjust for CT sensor
    scaling

void setup() {
    Serial.begin(9600);  // Start serial communication
}

void loop() {
    // Read raw analog values from sensors
    int rawVoltage = analogRead(voltagePin);
    int rawCurrent = analogRead(currentPin);

    // Convert raw values to real-world measurements
    float voltage = rawVoltage * voltageMultiplier;
    float current = rawCurrent * currentMultiplier;
    float power = voltage * current; // Active power calculation

    // Print the measurements to serial monitor
    Serial.print("Voltage: "); Serial.print(voltage); Serial.print(" V, ");
    Serial.print("Current: "); Serial.print(current); Serial.print(" A, ");
    Serial.print("Power: "); Serial.print(power); Serial.println(" W");

    delay(1000);  // Wait 1 second before next reading
}
```

Code 3.1: arduino code for energy calculations

### 3.3.2 Raspberry Pi Code for Home Automation

The Raspberry Pi is responsible for reading sensor data, controlling home appliances based on environmental conditions, and sending data to an IoT platform (ThingsBoard) using MQTT.

```
import RPi.GPIO as GPIO
import time
import json
import spidev
import math
from paho.mqtt.client import Client

# GPIO setup
GPIO.setmode(GPIO.BCM)
GPIO.setup(17, GPIO.IN)  # LDR sensor for light detection
GPIO.setup(27, GPIO.OUT) # Relay control for light switching

# SPI setup for ADC communication (e.g., MCP3008)
spi = spidev.SpiDev()
spi.open(0, 0)
spi.max_speed_hz = 1350000

# MQTT setup
THINGSBOARD_HOST = "localhost"
ACCESS_TOKEN = "your_access_token"
client = Client()
client.username_pw_set(ACCESS_TOKEN)
client.connect(THINGSBOARD_HOST, 1883, 60)

# Read ADC channel for sensor values
def read_channel(channel):
    adc = spi.xfer2([1, (8 + channel) << 4, 0])
    return ((adc[1] & 3) << 8) + adc[2]

# Calculate RMS voltage from multiple samples
def calculate_rms_voltage():
    sum_squares = sum((read_channel(3) * 3.3 / 1023.0) ** 2 for _ in range
    (200))
    return math.sqrt(sum_squares / 200) * 11  # Voltage divider correction

# Publish sensor data to MQTT
def publish_sensor_data():
    while True:
        light_level = read_channel(0)  # Read LDR sensor value
        GPIO.output(27, light_level < 100)  # Turn on relay if dark
```

```
40
41          payload = json.dumps({
42              "light": light_level,
43              "voltage": calculate_rms_voltage()
44          })
45          client.publish("v1/devices/me/telemetry", payload)
46          time.sleep(0.5)  # Wait before next update
47
48  publish_sensor_data()
```

Code 3.2: Raspberry Pi Code Collects Sensor Data and Sends it over using MQTT

## 3.4 Simulation and Testing

### 3.4.1 Home Automation Testing

Several tests were run in the Proteus simulation environment to validate the smart home automation system. The LDR-based light control circuit was tested under various lighting situations. The simulation confirmed that when the light intensity fell below a particular threshold, the relay was activated, which turned on the linked bulb. When the light intensity increased, the relay deactivated and the bulb turned off. This behavior provided automatic lighting management based on ambient light levels.

The PIR sensor was tested by mimicking motion within its detecting range. When motion was detected, the buzzer sounded an alert. When there was no motion, the buzzer remained off. This proved that the motion detection system was working properly, making it appropriate for security applications.

The MQTT communication between the Raspberry Pi and ThingsBoard Cloud was also tested. Sensor data, including light intensity and relay state, was successfully transmitted and displayed on the ThingsBoard dashboard in real time. This validated the proper integration of IoT communication in the system.

### 3.4.2 Smart Meter Testing

The smart energy meter system was evaluated for accuracy and dependability in measuring electrical characteristics. The Arduino successfully obtained voltage and current readings from the sensors. These readings were compared to expected values, and the findings revealed an

acceptable margin of error, indicating measurement accuracy.

The power consumption was calculated using the formula $P = V \times I$, and the computed values were displayed on the serial monitor. The readings matched theoretical expectations, demonstrating the correct implementation of power measurement.

Furthermore, the power factor correction circuit was simulated by adding different loads. As reactive power varied, the circuit modified to enhance the power factor. This enabled the system to dynamically optimize power consumption, decreasing energy waste.

## 3.5 Conclusion

The Proteus simulation offered a dependable testing environment for both the smart home automation and smart energy metering systems. The home automation system operated as planned, automatically regulating lighting based on ambient conditions and detecting motion to trigger security warnings. The smart meter measured electrical characteristics precisely and estimated power usage with excellent reliability.

Both systems' successful simulation testing shows that they are ready for real-world implementation. With the right hardware, these systems can improve energy efficiency and automation in smart home situations.

# Chapter 4

# ThingsBoard Integration

## 4.1 Introduction to ThingsBoard

Rapid breakthroughs in semiconductor technology and wireless communication have resulted in the development of low-cost sensor-based devices, which serve as the foundation for the Internet of Things (IoT) ecosystem. These sensors produce massive volumes of data, demanding effective collection, processing, and management frameworks. IoT platforms play an important role in managing this data by offering connectivity, security, data visualization, and analytics capabilities[21]. Among these platforms, ThingsBoard has emerged as an effective open-source solution for IoT data collecting and management.

ThingsBoard is a Java 8-based IoT platform that acts as a gateway for devices that communicate using MQTT[22] and HTTP. These protocols allow for lightweight communication between resource-constrained IoT devices and cloud services. MQTT is a publish/subscribe protocol for small, low-power devices that enables efficient message exchange via a broker with varying Quality of Service (QoS) levels[22]. In contrast, CoAP is an UDP-based protocol designed for limited contexts, with lower overhead but lesser dependability than MQTT.One of ThingsBoard's key features is the ability to build rules and plugins for message processing. Rules contain data filters, metadata enrichment processors, and action triggers that change messages into new formats before sending them to plugins. This rule-based system supports basic data processing, including threshold-based notifications. However, the technology does not automatically support complex data aggregation over time or across several devices[21].

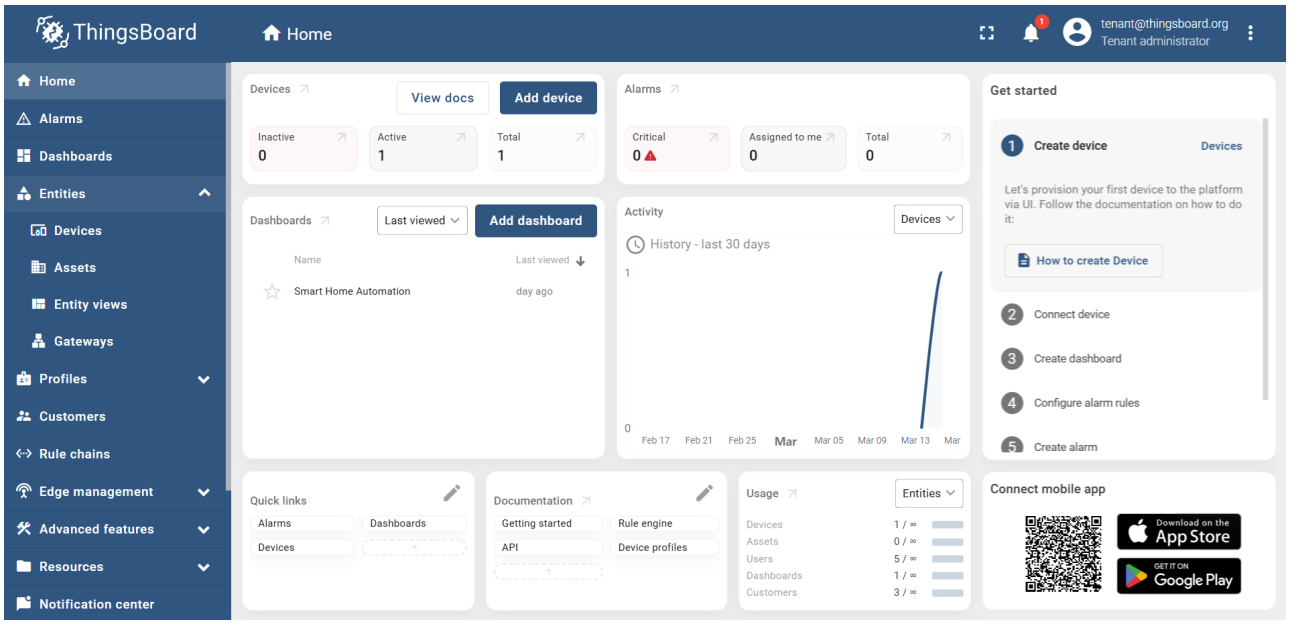


Figure 4.1: ThingsBoard's Home

## 4.2 Prerequisites for ThingsBoard MQTT Integration

Before integrating ThingsBoard with MQTT, ensure that the required software components are installed and configured correctly. The ThingsBoard platform must be running on either a local system or a cloud server. Although ThingsBoard has a built-in MQTT broker, an external broker such as Mosquitto can also be used if needed.Each device must first be created in ThingsBoard before it can communicate via MQTT. To create a new device, navigate to the ThingsBoard dashboard, go to the **Devices** section, and click on **Add New Device**. Assign a meaningful name and select the appropriate device type. Once the device is created, an **Access Token** is generated, which will be required for authentication in MQTT communication. A visual representation of the device creation process is shown in Figure 4.2. Additionally, MQTT communication requires appropriate network settings. Ensure that port **1883** is open for unencrypted communication. Each device must authenticate with ThingsBoard using an **Access Token**, which is assigned when the device is created in ThingsBoard.

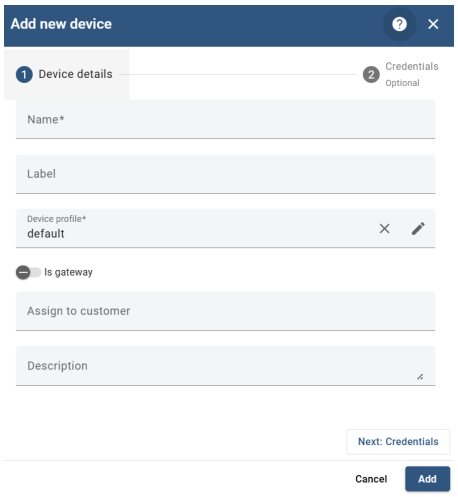


Figure 4.2: Device creation process in ThingsBoard

## 4.3 Connecting ThingsBoard with MQTT

To send telemetry data to ThingsBoard using MQTT, follow these steps:

### 4.3.1 Step 1: Define Connection Parameters

The first step is to configure the MQTT connection by specifying the ThingsBoard host, access token, and MQTT port. Replace the `ACCESS_TOKEN` with the token obtained from the ThingsBoard device.

```
THINGSBOARD_HOST = "localhost"
ACCESS_TOKEN = "dU6S0YIAPX5WwfmB3wUi"  # Replace with your actual token
MQTT_PORT = 1883

```

Code 4.1: Connection Parameters as Environment Variables

### 4.3.2 Step 2: Install and Import MQTT Library

After setting up the connection parameters, ensure that the `paho-mqtt` library is installed. This library is required to establish an MQTT connection. Once installed, import the necessary modules in the Python script.

```
import paho.mqtt.client as mqtt
import json

```

Code 4.2: Importing MQTT using Python

### 4.3.3 Step 3: Create MQTT Client and Connect

The next step is to create an MQTT client instance and authenticate using the access token. The client must then connect to the ThingsBoard host on the specified port.

```
client = mqtt.Client()
client.username_pw_set(ACCESS_TOKEN)  # Use Access Token for authentication
client.connect(THINGSBOARD_HOST, MQTT_PORT, 60)

```

Code 4.3: Connecting MQTT to ThingsBoard using Device Token

### 4.3.4 Step 4: Publish Sensor Data

Once connected, sensor data must be prepared in JSON format and published to the ThingsBoard MQTT topic. The following code snippet demonstrates how to send temperature and humidity data.

```
telemetry_data = {"current": 14.919, "fan_status": 0,"gas_detected":0,"
  light":61,"motion":1,"power":3493.48,"power_factor":0.98,"temperature
  ":27.85}
client.publish("v1/devices/me/telemetry", json.dumps(telemetry_data))
print("Data sent successfully!")
client.disconnect()

```

Code 4.4: Sending Telemetry Response in JSON

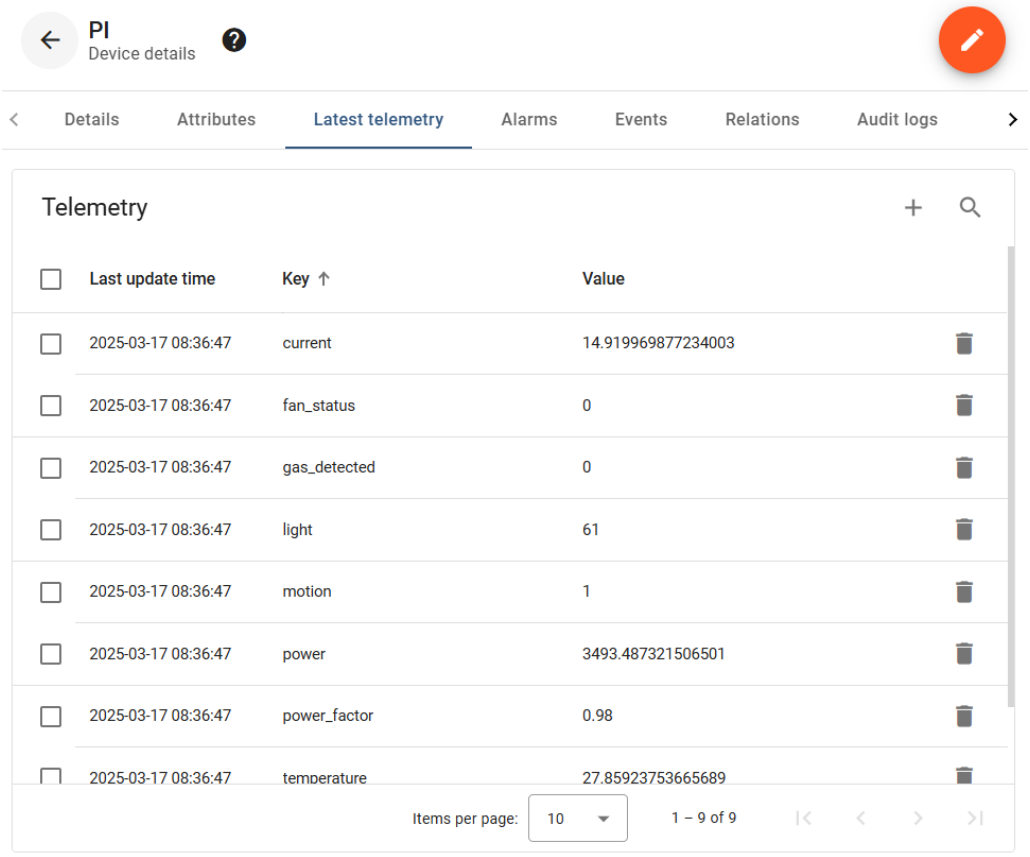


Figure 4.3: Latest Telemetry after Publishing Data

### 4.3.5 Step 5: Verify Data in ThingsBoard

After publishing the data, it is essential to verify whether ThingsBoard has received it. This can be done by navigating to the ThingsBoard UI and checking the **Latest Telemetry** section of the configured device.

### 4.3.6 Step 6:Creation of Dashboard in ThingsBoard

After creating a device in ThingsBoard, the next step is to create a dashboard for monitoring and visualization. To create a new dashboard, navigate to the **Dashboards** section and click on **Create New Dashboard**. Provide a meaningful name and description for easy identification. Once the dashboard is created, widgets can be added to visualize device telemetry data. Click on **Edit Dashboard**, then use the **Add New Widget** option to select an appropriate widget type, such as charts, gauges, or tables. Link the widget to the corresponding device and telemetry keys. The dashboard creation process is illustrated in Figure 4.4.

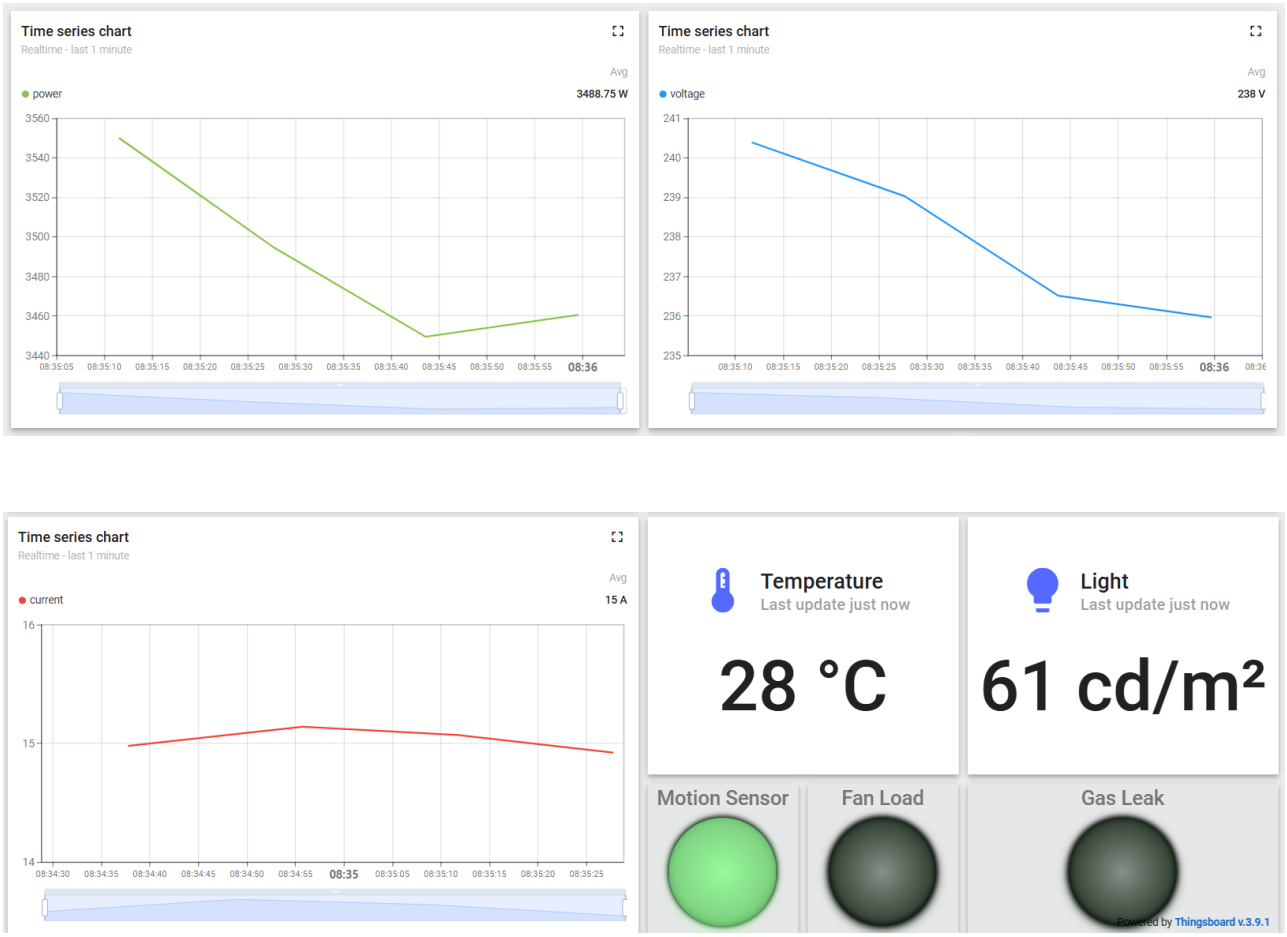


Figure 4.4: Dashboard Creation with ThingsBoard Widgets

## 4.4 Prerequisites for ThingsBoard API Integration

Before integrating ThingsBoard with the API, ensure that the required software and authentication mechanisms are set up properly. The ThingsBoard platform must be installed on a local system or a cloud server to facilitate communication. A REST client such as Postman or the `fetch` API in JavaScript is necessary for interacting with ThingsBoard's API. Additionally, proper network access should be configured to allow API requests.

To ensure security, ThingsBoard requires authentication via a JSON Web Token (JWT). The authentication process involves sending a username and password to ThingsBoard's login API. Upon successful authentication, a token is generated, which must be included in all subsequent API requests. Without this token, ThingsBoard will deny access to its resources.

## 4.5 Connecting to ThingsBoard API

To fetch telemetry data from ThingsBoard, follow these steps:

### 4.5.1 Step 1: Generate Access Token

The first step is to authenticate with ThingsBoard and obtain an access token. This is done by sending a POST request to the authentication API with valid credentials. The request should be formatted in JSON, containing a username and password.

```
POST http://localhost:9090/api/auth/login
Content-Type: application/json

{
    "username": "tenant@thingsboard.org",
    "password": "tenant"
}

```

Code 4.5: Content for Body of POST Request

### 4.5.2 Step 2: Extract JWT Token

Once the authentication request is processed, the API will return a response containing a JWT token. This token is essential for making authorized requests to ThingsBoard. The response will be in JSON format, and the token can be extracted from the `token` field.

```
{
    "token": "eyJhbGciOiJIUzI1NiIsInR5cCI...",
    "refreshToken": "eyJhbGciOiJIUzI1NiIsIn..."
}

```

Code 4.6: Response of POST Request

### 4.5.3 Step 3: Fetch Telemetry Data

After obtaining the authentication token, the next step is to retrieve telemetry data from a specific device. The device ID must be specified in the request URL, and the JWT token must be included in the request headers under the `X-Authorization` field. The following JavaScript code demonstrates how to fetch power telemetry data from a ThingsBoard device.

```
const response = await fetch(
    `http://localhost:9090/api/plugins/telemetry/DEVICE/0eb8ae80-ffe6-11ef-
  b36a-71656502eb9c/values/timeseries?keys=power`,
```

```
    {
        headers: { "X-Authorization": `Bearer ${thingsboardToken}` },
    }
);
const data = await response.json();
console.log(data);

```

Code 4.7: Fetching Data from ThingsBoard using Token Authentication method

### 4.5.4 Step 4: Process Response Data

Once the telemetry data is retrieved, it will be returned in JSON format. The response will contain key-value pairs representing the requested telemetry values. The received data can then be processed, displayed, or stored as needed, depending on the application requirements.

## 4.6 Conclusion

Integrating ThingsBoard with MQTT allows for efficient and secure real-time data transmission for IoT devices. By following the methods mentioned, devices can authenticate, publish telemetry data, and take advantage of ThingsBoard's monitoring and automation capabilities. This lightweight and scalable strategy improves IoT installations by providing consistent connectivity while also allowing for future enhancements such as encryption and improved data handling.

# Chapter 5

# Blockchain and Smart Contracts

## 5.1 Introduction

Blockchain is a decentralized, transparent, and secure technology for managing digital transactions. It combines cryptography, peer-to-peer networks, and consensus mechanisms to ensure that data is immutable and fraud-resistant[23].Smart contracts, introduced by Ethereum in 2013, are self-executing contracts with code-defined terms. These remove the need for intermediaries and allow for efficient and secure automation of processes in industries such as finance, energy, and logistics[24].

## 5.2 Advantages

Blockchain enables decentralization, immutability, and transparency. Its distributed architecture eliminates single points of failure and enables tamper-proof, cryptographically secure data. Smart contracts increase efficiency by enforcing logic directly on-chain, decreasing dependency on third parties.

## 5.3 Smart Contract Deployed in the Project

The following Solidity contract was deployed to implement a decentralized energy billing system:

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

contract EnergyBilling {
```

```
    address public owner;
    mapping(address => uint256) public userEnergy;
    mapping(address => uint256) public userBills;

    uint256 public constant RATE_PER_KWH = 2;

    event EnergyStored(address indexed user, uint256 energy);
    event BillPaid(address indexed user, uint256 amount);

    modifier onlyOwner() {
        require(msg.sender == owner, "Only owner can call this function");
        _;
    }

    constructor() {
        owner = msg.sender;
    }

    function storeEnergy(uint256 _totalEnergy) public {
        require(_totalEnergy > 0, "Energy must be greater than zero");

        userEnergy[msg.sender] += _totalEnergy;
        userBills[msg.sender] = userEnergy[msg.sender] * RATE_PER_KWH;

        emit EnergyStored(msg.sender, _totalEnergy);
    }

    function getBill() public view returns (uint256) {
        return userBills[msg.sender];
    }

    function payBill() public payable {
        uint256 billAmount = userBills[msg.sender];
        require(msg.value == billAmount, "Incorrect payment amount");

        userBills[msg.sender] = 0;

        emit BillPaid(msg.sender, msg.value);
    }
}
```

```
45
```

## 5.4 Web3 Application Setup

The frontend for interacting with this contract was built using React and Web3.js. The initial setup was done using Vite:

```
npm create vite@latest web3-app --template react
cd web3-app
npm install
npm install web3 dotenv

```

## 5.5 Blockchain Connection and Data Interaction

A Web3 connection to the smart contract was established using the following code snippet:

```
import Web3 from "web3";

const web3 = new Web3(import.meta.env.VITE_GANACHE_RPC);
const contractAddress = import.meta.env.VITE_CONTRACT_ADDRESS;
const privateKey = import.meta.env.VITE_PRIVATE_KEY;
const account = import.meta.env.VITE_ACCOUNT;

```

Energy data is fetched from ThingsBoard:

```
async function fetchEnergyData() {
    const response = await fetch(
        `http://localhost:9090/api/plugins/telemetry/DEVICE/${DEVICE_ID}/
  values/timeseries?keys=power`,
        { headers: { "X-Authorization": `Bearer ${thingsboardToken}` } }
    );
    const data = await response.json();
    const power = data.power[0].value;
    setEnergy(Math.floor(power));
}

```

## 5.6 Smart Contract Interaction

### 5.6.1 Store Energy

```
async function sendEnergyData() {
    const tx = contract.methods.storeEnergy(energy);
    const gas = await tx.estimateGas({ from: account });
    const gasPrice = await web3.eth.getGasPrice();
    const data = tx.encodeABI();
    const nonce = await web3.eth.getTransactionCount(account, "latest");

    const signedTx = await web3.eth.accounts.signTransaction(
        { to: contractAddress, data, gas, gasPrice, nonce },
        privateKey
    );

    await web3.eth.sendSignedTransaction(signedTx.rawTransaction);
}

```

### 5.6.2 Fetch and Pay Bill

```
async function fetchBill() {
    const billAmount = await contract.methods.getBill().call({ from:
  account });
    setBill(billAmount);
}

async function payBill() {
    const billAmount = await contract.methods.getBill().call({ from:
  account });
    const tx = contract.methods.payBill();
    const gas = await tx.estimateGas({ from: account, value: billAmount });
    const gasPrice = await web3.eth.getGasPrice();
    const data = tx.encodeABI();
    const nonce = await web3.eth.getTransactionCount(account, "latest");

    const signedTx = await web3.eth.accounts.signTransaction(
```

```
        { to: contractAddress, data, gas, gasPrice, nonce, value:
    billAmount },
        privateKey
      );

      await web3.eth.sendSignedTransaction(signedTx.rawTransaction);
  }

```

## 5.7 Frontend Interface

The React frontend allows users to interact with the contract through a simple UI:

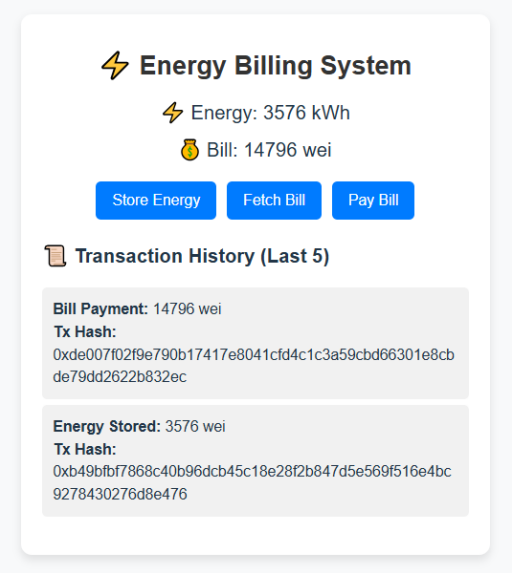


Figure 5.1: User interface of the Energy Billing System

## 5.8 Transaction Logs

Each smart contract interaction is recorded on the blockchain. The frontend logs these transactions for verification, as shown in Figure 5.2.

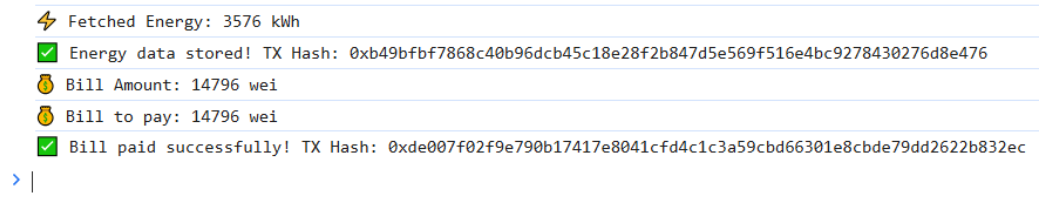


Figure 5.2: Transaction log from frontend interactions with the smart contract

## 5.9 Conclusion

This chapter demonstrated how blockchain and smart contracts can automate energy billing with transparency and security. By integrating Web3.js, React, and ThingsBoard, a decentralized billing system was developed that showcases the capabilities of Web3 in real-world applications.

# Chapter 6

# Conclusion and Future Scope

## 6.1 Conclusion

This study highlights the potential of integrating IoT and smart energy management within home automation systems to transform energy use, monitoring, and billing. By leveraging real-time data and automation, the proposed system ensures transparency, efficiency, and accuracy, leading to more streamlined and cost-effective energy management. The system's ability to monitor and manage energy usage dynamically paves the way for optimized billing and enhanced user experience in smart home environments.

## 6.2 Future Scope

Building on the current study, there are several opportunities to enhance the proposed system and further push the boundaries of smart home automation for energy management:

- **Smart Grid Integration**: Future work could explore the integration of smart grid capabilities, enabling households to share excess solar energy within a local network. This peer-to-peer energy sharing model could encourage sustainable living and improve overall energy efficiency.
- **Demand-Response Systems**: The system could be expanded with demand-response capabilities, which would dynamically adjust energy usage based on real-time grid conditions. This would improve load control and reduce energy waste, benefiting both consumers and the broader energy grid.
- **Solar Energy and Smart Water Pumping**: Incorporating solar energy generation

monitoring and intelligent water pumping systems could create a more self-sufficient household ecosystem. This would optimize energy distribution to appliances and regulate water consumption based on available energy, promoting greater efficiency.

- **Hardware Upgrades**: Moving to more powerful microcontrollers, such as the ESP32 or other advanced IoT-enabled devices, would significantly improve the system's performance. These devices would offer greater computational power, reduced energy consumption, and enhanced connectivity, making the system more scalable and suited for real-time applications.

These future enhancements could significantly improve the system's functionality, making smart home automation even more efficient, sustainable, and scalable.

# References


[1] S. Jain, A. Vaibhav, and L. Goyal, "Raspberry pi based interactive home automation system through e-mail," in *2014 International Conference on Reliability Optimization and Information Technology (ICROIT)*. IEEE, 2014, pp. 277–280.

[2] S. Chaudhari, P. Rathod, A. Shaikh, D. Vora, and J. Ahir, "Smart energy meter using arduino and gsm," in *2017 International Conference on Trends in Electronics and Informatics (ICEI)*. IEEE, 2017, pp. 598–601.

[3] V. Patchava, H. B. Kandala, and P. R. Babu, "A smart home automation technique with raspberry pi using iot," in *2015 International conference on smart sensors and systems (IC-SSS)*. IEEE, 2015, pp. 1–4.

[4] M. Abdelhamid and G. Hassan, "Blockchain and smart contracts," in *Proceedings of the 8th International Conference on Software and Information Engineering*, ser. ICSIE '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 91–95. [Online]. Available: https://doi.org/10.1145/3328833.3328857

[5] Y. Hu, M. Liyanage, A. Mansoor, K. Thilakarathna, G. Jourjon, and A. P. Seneviratne, "Blockchain-based smart contracts - applications and challenges," *arXiv: Computers and Society*, 2018. [Online]. Available: https://api.semanticscholar.org/CorpusID:182952419

[6] B. Zhang and Y. Zhang, "The current situation and future challenges of smart home," 2022.

[7] D. A. and, "Anthropomorphizing artificial intelligence: towards a user-centered approach for addressing the challenges of over-automation and design understandability in smart homes," *Intelligent Buildings International*, vol. 13, no. 4, pp. 227–240, 2021. [Online]. Available: https://doi.org/10.1080/17508975.2020.1795612

[8] V. A, "Iot-based smart home automation systems: Enhancing energy efficiency and security," *International Journal for Research in Applied Science and Engineering Technology*, vol. 12, no. 12, p. 2243–2253, Dec 2024.

[9] A. Mazid, S. Kirmani, and Manaullah, "Iot enabled framework for smart home automation using artificial intelligence and blockchain technology," in *International Conference on Artificial Intelligence of Things*. Springer, 2023, pp. 357–367.

[10] Y. Li, A. M. Mandalari, and I. Straw, "Who let the smart toaster hack the house? an investigation into the security vulnerabilities of consumer iot devices," *arXiv preprint arXiv:2306.09017*, 2023.

[11] C. C. Sobin, "A survey on architecture, protocols and challenges in iot," *Wireless Personal Communications*, vol. 112, no. 3, p. 1383–1429, Jan 2020.

[12] I. Holguin and S. Errapotu, "Smart home iot communication protocols and advances in their security and interoperability," *Proceedings of the International Conference on Cyber Security and Networking (CSNet)*, pp. 208–211, October 2023.

[13] I. V. Sita and B. David, "Development and implementation of a comprehensive smart home automation system using home assistant and iot devices," in *2024 4th International Conference on Electrical, Computer, Communications and Mechatronics Engineering (ICECCME)*. IEEE, 2024, pp. 1–7.

[14] X. Bai, "Research on smart home system design and user experience improvement strategies," *Advances in Computer, Signals and Systems*, vol. 8, pp. 34–39, 2024. [Online]. Available: https://www.clausiuspress.com/article/13197.html

[15] Y. A. Elmi, "Interoperable iot devices and systems for smart homes: A data analytics approach to enhance user experience and energy efficiency," *Journal of Digitainability, Realism & Mastery (DREAM)*, vol. 2, no. 10, pp. 51–66, October 2023. [Online]. Available: https://dreamjournal.my/index.php/DREAM/article/view/195

[16] G. Kaur, A. Kaur, N. Sharma, M. Singh, and K. Mittal, "Evolution of ai in smart home systems: A comprehensive review," *CGC International Journal of Contemporary Technology*, vol. 6, no. 2, pp. 388–395, October 2024. [Online]. Available: https://cgcijctr.com/download.php?file=C20240401_Evolution+of+AI+in+Smart+Home+Systems+A+Comprehensive+Review_Final.pdf

[17] N. Valov and I. Valova, "Home automation system with raspberry pi," in *2020 7th International Conference on Energy Efficiency and Agricultural Engineering (EE&AE)*. IEEE, 2020, pp. 1–5.

[18] G. Flurry, *An Analog-to-Digital Converter*. Berkeley, CA: Apress, 2021, pp. 397–412. [Online]. Available: https://doi.org/10.1007/978-1-4842-7264-0_12

[19] L. Müller, M. Mohammed, and J. W. Kimball, "Using the arduino uno to teach digital control of power electronics," in *2015 IEEE 16th Workshop on Control and Modeling for Power Electronics (COMPEL)*. IEEE, 2015, pp. 1–8.

[20] A. J. Taufiq, I. H. Kurniawan, and T. A. Y. Nugraha, "Analysis of arduino uno application on control system based on industrial scale," *IOP Conference Series: Materials Science and Engineering*, vol. 771, no. 1, p. 012015, mar 2020. [Online]. Available: https://dx.doi.org/10.1088/1757-899X/771/1/012015

[21] B. Hammi, R. Khatoun, S. Zeadally, A. Fayad, and L. Khoukhi, "Iot technologies for smart cities," *IET networks*, vol. 7, no. 1, pp. 1–13, 2018.

[22] Z. Kegenbekov and A. Saparova, "Using the mqtt protocol to transmit vehicle telemetry data," *Transportation Research Procedia*, vol. 61, pp. 410–417, 2022.

[23] S. Tern, "Survey of smart contract technology and application based on blockchain," *Open Journal of Applied Sciences*, vol. 11, no. 10, pp. 1135–1148, 2021.

[24] R. Johari, K. Gupta, and A. S. Parihar, "Smart contracts in smart cities: Application of blockchain technology," in *Innovations in Information and Communication Technologies (IICT-2020) Proceedings of International Conference on ICRIHE-2020, Delhi, India: IICT-2020*. Springer, 2021, pp. 359–367.