

**RV COLLEGE OF ENGINEERING**

**Bengaluru-560 059 REPORT ON EXPERIENTIAL LEARNING ACY 2024-25**

**THEME:**

***Microcontroller Programming***

**Title of the Project:**

***Geyser Control with Dry Run Protection***

**Students Group**

|  |  |  |  |
| --- | --- | --- | --- |
| **SL. No.** | **USN** | **Name** | **Prog.** |
| **1** | 1RV23ET024 | Nikhilesh Rao | **ET** |
| **3** | 1RV23ET010 | Anwitha Vishwanath | **ET** |
| **4** | 1RV23ET019 | Harshit Saroha | **ET** |
| **5** | 1RV23ET016 | Chinmay Sharma | **ET** |

### Topics Covered:

Abstract

1. Introduction
   1. Problem Statement
   2. Objectives
2. Literature Survey
3. Design
   1. Methodology
   2. Tools and Techniques Used
4. Results and Discussion
5. Testing
6. Conclusion
   1. Summary
7. Visuals

# ABSTRACT

The rapid increase in domestic energy usage and equipment failure due to poor safety mechanisms has made intelligent appliance control an essential requirement. This project proposes an **automated geyser control system with dry run protection and overheat prevention**, designed using an **STM32 microcontroller**, **temperature sensor (DS18B20)**, and **water level sensor (P43)**. The goal is to ensure energy efficiency, user safety, and increased appliance lifespan by preventing hazardous conditions such as running the geyser without water (dry run) and overheating.

The water level sensor detects whether the minimum required water is present in the tank. If the water level is too low, the system prevents the geyser from turning ON, thereby avoiding a dry run condition which can damage the heating element. Simultaneously, the DS18B20 temperature sensor continuously monitors the water temperature. If the temperature exceeds a predefined threshold, the system automatically turns OFF the geyser to prevent overheating.

The **STM32 microcontroller** acts as the brain of the system, executing the control logic programmed in Embedded C. The ON/OFF state of the geyser is visually simulated using an LED. The system operates autonomously, requiring no manual intervention once powered.

To enhance user awareness and remote monitoring, the project includes a **web interface** that displays real-time warnings such as:

* **Dry Run Detected**
* **Overheating Warning**
* **Geyser ON/OFF Status**

This integration demonstrates how microcontroller-based safety mechanisms can be combined with IoT features for smart appliance management. The solution is low-cost, reliable, and scalable, suitable for household or small industrial applications.

# CHAPTER 1: INTRODUCTION

#### INTRODUCTION

Electric geysers are widely used in households and commercial establishments for instant hot water supply. However, conventional geysers lack intelligent control systems, making them vulnerable to two critical issues: **dry run operation**and **overheating**. A dry run occurs when the geyser is powered ON without sufficient water inside the tank, which can severely damage the heating element and pose a fire hazard. Similarly, continuous heating without temperature regulation may result in **overheating**, causing energy wastage, risk of burns, or system failure.

To address these safety and efficiency concerns, this project proposes a **microcontroller- based intelligent geyser control system** using **STM32**, a 32-bit ARM Cortex-M4 microcontroller. The system uses two primary sensors:

* A **water level sensor (P43)** to detect the presence of sufficient water,
* A **temperature sensor (DS18B20)** to monitor real-time water temperature.

The core objective is to **automatically prevent dry run conditions** and **turn off the geyser once the temperature threshold is exceeded**, without any manual intervention. This ensures user safety, energy savings, and improved appliance life.

For user interaction and monitoring, the system also includes a **web-based dashboard** that displays the current status of the geyser, water level, and temperature. Warnings like **"Dry Run Detected"** or **"Overheating"** are shown in real-time, making the system not just smart but also **IoT-enabled**.

By combining embedded systems and web technologies, the project represents a cost- effective, scalable solution for **safe, automated, and intelligent water heating control** in modern homes and buildings

#### PROBLEM STATEMENT

Conventional geysers lack safety mechanisms to detect low water levels and over-temperature conditions, leading to **dry run damage**, **overheating**, and **energy waste**. There is a need for an **automated system** that can **prevent geyser operation without sufficient water** and **shut it off when the temperature exceeds a safe limit**, with real-time status monitoring.

# OBJECTIVES

1. **Dry Run Protection through Water Level Monitoring** Integrate a water level sensor (P43) to detect the presence of minimum water in the geyser tank. If the water level falls below a predefined threshold, the system ensures the geyser remains OFF to prevent dry run operation, thereby protecting the heating element from damage and ensuring safe usage.
2. **Automated Overheating Control** Utilize the DS18B20 temperature sensor to continuously monitor the water temperature. When the sensed temperature exceeds a predefined safety threshold, the system automatically turns OFF the geyser to avoid overheating, prevent energy wastage, and reduce the risk of accidents.
3. **Visual Geyser Status Indication using LED** Simulate the operational state of the geyser using an LED. The LED glows when the geyser is ON and turns OFF when the geyser is either switched off due to low water level or high temperature. This visual indicator helps in quickly understanding the system’s current status during testing and demonstration.
4. **Embedded Threshold-Based Control Logic** Implement intelligent logic on an STM32 microcontroller using Embedded C. The logic interprets real-time data from the sensors and makes decisions without any manual input. The control system ensures that geyser operation is automated and aligned with defined safety thresholds for water level and temperature.
5. **Web Interface for Real-Time Monitoring and Alerts** Develop a user-friendly website that displays the current status of the geyser system, including water level, temperature, and ON/OFF state. The website also provides real-time alerts such as “Dry Run Detected” or “Overheating Warning”, enabling remote supervision and enhancing user awareness and safety.

# CHAPTER 2: LITERATURE SURVEY

### Automated Water Heating Management with Internet of Things (2023)

#### Introduction

Household water heaters often operate unsafely or inefficiently due to manual controls. This study uses IoT to automate monitoring and control of water heaters, ensuring safe water temperature and efficient energy use.

#### Authors and Publication Details

Authors: R. Herasustanti, A. Abdi, H. Purnata, N. Mailahi, S. Rian Affiliation: Department of Electronics Engineering, State Polytechnic of Cilacap, Indonesia Publication: Journal of Telecommunication Network, 2023

#### Key Findings

* + - * System uses NodeMCU ESP8266 to collect temperature and flow sensor data.
      * Data is sent via Wi-Fi to Firebase and displayed in an Android app.
      * Achieved < 0.4 L measurement error and ~7.11 % flow accuracy.
      * Relay control operates with an average delay of 3.6 s.

#### Merits

* + - * Low-cost, real-time remote monitoring and control
      * High accuracy of sensor measurements
      * Seamless integration with Android via Firebase

#### Demerits

* + - * Relay delay may impact real-time safety
      * No explicit dry-run or overheat protections
      * Limited to basic sensor types (flow and temperature)

#### Conclusion

A practical proof-of-concept for IoT-based control of water heaters, demonstrating remote monitoring, accurate sensor readings, and basic automation. Expanding safety features would further strengthen the system’s utility.

### Title: Embedded Model Predictive Control of Tankless Gas Water Heaters to Enhance Users’ Comfort

#### Introduction

Tankless gas water heaters often experience temperature instability under varying water demand. This paper implements Model Predictive Control (MPC) on embedded hardware to enhance comfort and energy efficiency.

#### Authors and Publication Details

Authors: Duru Viegas et al.

Affiliation: Research-based study published via MDPI/ResearchGate Publication Year: 2022–2023

#### Key Findings

* MPC implemented using MATLAB/Simulink and auto-generated C code.
* Deployed on both 8-bit and 32-bit microcontrollers.
* 6.8 % improvement in user comfort index observed on 32-bit systems.

#### Merits

* Predictive control ensures stable water temperature
* Enhances comfort and minimizes energy waste
* Validated on real embedded systems via hardware-in-loop

#### Demerits

* Focused on gas-based systems, not electric geysers
* Computational overhead may limit low-end MCU usage
* No field tests on commercial appliances

#### Conclusion

Model Predictive Control improves user comfort and reduces energy consumption in tankless heaters. Though computationally heavier, embedding this control on microcontrollers shows strong potential for smart appliance applications.

# Title: Optimal Water Heater Control in Smart Home Environments (2016)

#### Introduction

Smart homes require efficient energy management for appliances like water heaters. This study proposes dynamic programming techniques to optimally schedule geyser operation based on demand, pricing, and solar generation.

#### Authors and Publication Details

Authors: Christoph Passenberg, Dominik Meyer, Johannes Feldmaier, Hao Shen

Affiliation: Technische Universität München, Germany

Published on: arXiv.org

#### Publication Date: July 2016 Key Findings

* Uses predicted hot water usage and solar PV output to control heater.
* Dynamic programming used to determine optimal ON/OFF schedules.
* Demonstrated reduced electricity cost without affecting comfort.

#### Merits

* Smart scheduling integrated with renewable energy usage
* Effective cost savings while maintaining hot water availability
* Algorithm adaptable to varying home conditions

#### Demerits

* Only tested in simulations
* Relies heavily on accurate forecasting
* High computation may not suit real-time embedded systems

#### Conclusion

Dynamic programming enables cost-effective and energy-efficient water heater operation in smart homes. While promising in simulation, real-world deployment requires simplification and practical testing for feasibility.

.

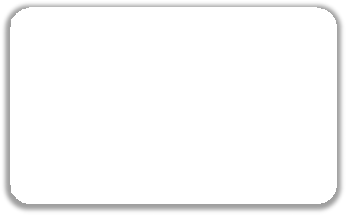
# CHAPTER 3: DESIGN

**METHODOLOGY**

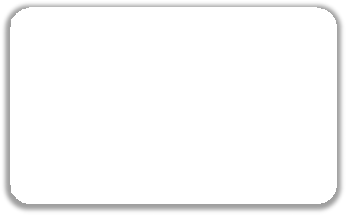
This project uses a structured, layered methodology for smart geyser control. It ensures safety by preventing dry run and overheating conditions.

Automation is achieved using STM32 with water level and temperature sensors.

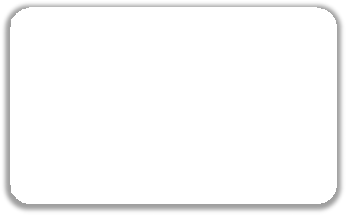
Real-time monitoring and alerts are provided through a connected web interface.



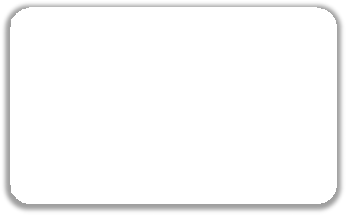
Define Control Objectives



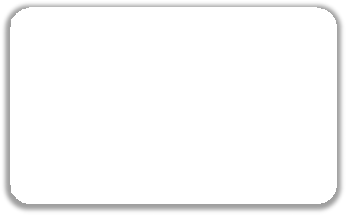
System Design and Setup



Signal Processing and Control Logic Implementation



Sensor and Hardware Configuration



Data Visualization and Alert System

#### Define Control Objectives

The first step involves identifying the core functionalities required for safe and efficient geyser operation. The main objectives are:

* + To **prevent dry run** situations where the geyser operates without sufficient water, potentially damaging the heating element.
  + To **avoid overheating** by turning off the geyser once the water temperature exceeds a

defined safety threshold.

* + To simulate these conditions using an LED for demonstration and testing purposes.
  + To incorporate real-time alerts through a web interface for remote monitoring.

#### System Design and Setup

A complete embedded system is designed using the **STM32 microcontroller**, chosen for its rich peripheral set and real-time processing capability. The two key sensors interfaced are:

* + **Water Level Sensor (P43)**: A float-type digital sensor that detects the presence or absence of

water.

* + **Temperature Sensor (DS18B20)**: A 1-Wire digital sensor providing accurate temperature readings in real-time.

The hardware is powered via a regulated 5V/3.3V supply, and the geyser is simulated using a **LED** to reflect the ON/OFF states of heating.

The system also includes optional interfacing with a **ESP32 (via UART)** for cloud/web

communication.

#### Sensor and Hardware Configuration

The STM32 microcontroller's **GPIO pins** are configured for digital input from the SEN18 sensor and digital communication with the DS18B20 temperature sensor. The LED is connected to a digital output pin to represent the geyser's status.

* + **Water level sensor input** is connected to a GPIO configured in input mode with pull- up/pull-down as required.
  + **DS18B20** is connected via 1-Wire protocol using a digital pin, and its temperature readings

are periodically polled.

* + **LED** is connected to a GPIO pin configured as output and is toggled according to logic conditions.

#### Signal Processing and Control Logic Implementation

The control logic is written in **Embedded C** and runs in the main loop or using timer-based

interrupts. It performs the following steps:

* + Continuously reads the state of the water level sensor.
  + Fetches real-time temperature from the DS18B20.
  + Checks whether the **water level is adequate** and **temperature is within safe limits**.
  + If **water is present** and **temperature is below threshold**, the LED (geyser) is turned ON.
  + If either the **water level is low** or **temperature exceeds the limit**, the LED is turned OFF to prevent dry run or overheating.

Thresholds are defined as:

* + **Minimum water level = 1 (High)** from the sensor.
  + **Maximum temperature = e.g., 60°C** (adjustable as needed).

Debouncing and fail-safe conditions are handled in software to prevent false triggers due to noise.

#### Data Visualization and Alert System

To enhance usability and provide real-time visibility into the geyser's status, a **web interface** is developed using:

* + **ESP32** to send data from STM32 to a cloud dashboard.
  + The interface shows:
    - Current water level status
    - Temperature in °C
    - Geyser ON/OFF state
    - Warnings like **"Dry Run Detected"** or **"Overheating Warning"**

This dashboard allows users to monitor their geyser status remotely and ensure it

operates under safe conditions.

**VISUAL STUDIO CODE**

#include "main.h"

/\* Private includes \*/

/\* USER CODE BEGIN Includes \*/

#include "main.h" #include "delay\_us.h" #include "ds18b20.h" #include <stdio.h> #include <string.h> #include "onewire.h"

/\* USER CODE END Includes \*/

/\* Private typedef \*/

/\* USER CODE BEGIN PTD \*/

/\* USER CODE END PTD \*/

/\* Private define \*/

/\* USER CODE BEGIN PD \*/

/\* USER CODE END PD \*/

/\* Private macro \*/

/\* USER CODE BEGIN PM \*/

/\* USER CODE END PM \*/

/\* Private variables \*/

UART\_HandleTypeDef huart2;

/\* USER CODE BEGIN PV \*/

char uart\_buf[50];

#define TEMP\_THRESHOLD 32.0

/\* USER CODE END PV \*/

/\* Private function prototypes \*/

void SystemClock\_Config(void); static void MX\_GPIO\_Init(void);

static void MX\_USART2\_UART\_Init(void);

/\* USER CODE BEGIN PFP \*/

##### /\* USER CODE END PFP \*/

/\* Private user code \*/

##### /\* USER CODE BEGIN 0 \*/

void SendToESP32(float temp, int water\_present) {

char buffer[64];

sprintf(buffer, "TEMP:%.2f,WATER:%d\n", temp, water\_present); HAL\_UART\_Transmit(&huart2, (uint8\_t \*)buffer, strlen(buffer), HAL\_MAX\_DELAY);

}

##### /\* USER CODE END 0 \*/

/\*\*

* @brief The application entry point.
* @retval int

\*/

int main(void)

{

##### /\* USER CODE BEGIN 1 \*/

/\* USER CODE END 1 \*/

/\* MCU Configuration \*/

/\* Reset of all peripherals, Initializes the Flash interface and the Systick. \*/ HAL\_Init();

/\* USER CODE BEGIN Init \*/

/\* USER CODE END Init \*/

/\* Configure the system clock \*/

SystemClock\_Config();

/\* USER CODE BEGIN SysInit \*/

/\* USER CODE END SysInit \*/

/\* Initialize all configured peripherals \*/ MX\_GPIO\_Init(); MX\_USART2\_UART\_Init();

##### /\* USER CODE BEGIN 2 \*/

Onewire\_Init();

Page | 13

##### /\* USER CODE END 2 \*/

/\* Infinite loop \*/

##### /\* USER CODE BEGIN WHILE \*/

while (1)

{

float temperature = DS18B20\_GetTemp();

int water\_present = HAL\_GPIO\_ReadPin(GPIOA, GPIO\_PIN\_0);

if (water\_present || temperature >= TEMP\_THRESHOLD) {

HAL\_GPIO\_WritePin(GPIOD, GPIO\_PIN\_12|GPIO\_PIN\_13|GPIO\_PIN\_14|GPIO\_PIN\_15, GPIO\_PIN\_RESET);

} else {

HAL\_GPIO\_WritePin(GPIOD, GPIO\_PIN\_12|GPIO\_PIN\_13|GPIO\_PIN\_14|GPIO\_PIN\_15, GPIO\_PIN\_SET);

}

SendToESP32(temperature, water\_present);

HAL\_Delay(1000);

##### /\* USER CODE END WHILE \*/

/\* USER CODE BEGIN 3 \*/

}

##### /\* USER CODE END 3 \*/

}

/\*\*

* @brief System Clock Configuration
* @retval None

\*/

void SystemClock\_Config(void)

{

RCC\_OscInitTypeDef RCC\_OscInitStruct = {0};

RCC\_ClkInitTypeDef RCC\_ClkInitStruct = {0};

/\*\* Configure the main internal regulator output voltage

\*/

##### HAL\_RCC\_PWR\_CLK\_ENABLE();

HAL\_PWR\_VOLTAGESCALING\_CONFIG(PWR\_REGULATOR\_VOLTAGE\_SCALE1);

/\*\* Initializes the RCC Oscillators according to the specified parameters

* in the RCC\_OscInitTypeDef structure.

Page | 14

RCC\_OscInitStruct.OscillatorType = RCC\_OSCILLATORTYPE\_HSI; RCC\_OscInitStruct.HSIState = RCC\_HSI\_ON; RCC\_OscInitStruct.HSICalibrationValue = RCC\_HSICALIBRATION\_DEFAULT; RCC\_OscInitStruct.PLL.PLLState = RCC\_PLL\_ON; RCC\_OscInitStruct.PLL.PLLSource = RCC\_PLLSOURCE\_HSI; RCC\_OscInitStruct.PLL.PLLM = 8;

RCC\_OscInitStruct.PLL.PLLN = 72; RCC\_OscInitStruct.PLL.PLLP = RCC\_PLLP\_DIV2; RCC\_OscInitStruct.PLL.PLLQ = 7;

if (HAL\_RCC\_OscConfig(&RCC\_OscInitStruct) != HAL\_OK)

{

Error\_Handler();

}

/\*\* Initializes the CPU, AHB and APB buses clocks

\*/

RCC\_ClkInitStruct.ClockType = RCC\_CLOCKTYPE\_HCLK|RCC\_CLOCKTYPE\_SYSCLK

##### |RCC\_CLOCKTYPE\_PCLK1|RCC\_CLOCKTYPE\_PCLK2;

RCC\_ClkInitStruct.SYSCLKSource = RCC\_SYSCLKSOURCE\_PLLCLK; RCC\_ClkInitStruct.AHBCLKDivider = RCC\_SYSCLK\_DIV1; RCC\_ClkInitStruct.APB1CLKDivider = RCC\_HCLK\_DIV2; RCC\_ClkInitStruct.APB2CLKDivider = RCC\_HCLK\_DIV2;

if (HAL\_RCC\_ClockConfig(&RCC\_ClkInitStruct, FLASH\_LATENCY\_2) != HAL\_OK)

{

Error\_Handler();

}

}

/\*\*

* @brief USART2 Initialization Function
* @param None
* @retval None

\*/

static void MX\_USART2\_UART\_Init(void)

{

/\* USER CODE BEGIN USART2\_Init 0 \*/

/\* USER CODE END USART2\_Init 0 \*/

/\* USER CODE BEGIN USART2\_Init 1 \*/

/\* USER CODE END USART2\_Init 1 \*/

huart2.Instance = USART2;

Page | 15

huart2.Init.BaudRate = 115200;

huart2.Init.WordLength = UART\_WORDLENGTH\_8B; huart2.Init.StopBits = UART\_STOPBITS\_1; huart2.Init.Parity = UART\_PARITY\_NONE; huart2.Init.Mode = UART\_MODE\_TX\_RX; huart2.Init.HwFlowCtl = UART\_HWCONTROL\_NONE; huart2.Init.OverSampling = UART\_OVERSAMPLING\_16; if (HAL\_UART\_Init(&huart2) != HAL\_OK)

{

Error\_Handler();

}

/\* USER CODE BEGIN USART2\_Init 2 \*/

/\* USER CODE END USART2\_Init 2 \*/

}

/\*\*

* @brief GPIO Initialization Function
* @param None
* @retval None

\*/

static void MX\_GPIO\_Init(void)

{

GPIO\_InitTypeDef GPIO\_InitStruct = {0};

/\* USER CODE BEGIN MX\_GPIO\_Init\_1 \*/

/\* USER CODE END MX\_GPIO\_Init\_1 \*/

/\* GPIO Ports Clock Enable \*/

HAL\_RCC\_GPIOH\_CLK\_ENABLE();

HAL\_RCC\_GPIOA\_CLK\_ENABLE();

HAL\_RCC\_GPIOD\_CLK\_ENABLE();

/\*Configure GPIO pin Output Level \*/ HAL\_GPIO\_WritePin(GPIOA, GPIO\_PIN\_1, GPIO\_PIN\_RESET);

/\*Configure GPIO pin Output Level \*/

HAL\_GPIO\_WritePin(GPIOD, GPIO\_PIN\_12|GPIO\_PIN\_13|GPIO\_PIN\_14|GPIO\_PIN\_15, GPIO\_PIN\_RESET);

/\*Configure GPIO pin : PA0 \*/ GPIO\_InitStruct.Pin = GPIO\_PIN\_0; GPIO\_InitStruct.Mode = GPIO\_MODE\_INPUT; GPIO\_InitStruct.Pull = GPIO\_PULLUP; HAL\_GPIO\_Init(GPIOA, &GPIO\_InitStruct);

/\*Configure GPIO pin : PA1 \*/ GPIO\_InitStruct.Pin = GPIO\_PIN\_1;

GPIO\_InitStruct.Mode = GPIO\_MODE\_OUTPUT\_PP; GPIO\_InitStruct.Pull = GPIO\_NOPULL; GPIO\_InitStruct.Speed = GPIO\_SPEED\_FREQ\_LOW; HAL\_GPIO\_Init(GPIOA, &GPIO\_InitStruct);

Page | 16

/\*Configure GPIO pins : PD12 PD13 PD14 PD15 \*/

GPIO\_InitStruct.Pin = GPIO\_PIN\_12|GPIO\_PIN\_13|GPIO\_PIN\_14|GPIO\_PIN\_15; GPIO\_InitStruct.Mode = GPIO\_MODE\_OUTPUT\_PP;

GPIO\_InitStruct.Pull = GPIO\_NOPULL; GPIO\_InitStruct.Speed = GPIO\_SPEED\_FREQ\_LOW; HAL\_GPIO\_Init(GPIOD, &GPIO\_InitStruct);

/\* USER CODE BEGIN MX\_GPIO\_Init\_2 \*/

/\* USER CODE END MX\_GPIO\_Init\_2 \*/

}

/\* USER CODE BEGIN 4 \*/

/\* USER CODE END 4 \*/

/\*\*

* @brief This function is executed in case of error occurrence.
* @retval None

\*/

void Error\_Handler(void)

{

/\* USER CODE BEGIN Error\_Handler\_Debug \*/

/\* User can add his own implementation to report the HAL error return state \*/

disable\_irq(); while (1)

{

}

/\* USER CODE END Error\_Handler\_Debug \*/

}

#ifdef USE\_FULL\_ASSERT

/\*\*

* @brief Reports the name of the source file and the source line number
* where the assert\_param error has occurred.
* @param file: pointer to the source file name
* @param line: assert\_param error line source number
* @retval None

\*/

void assert\_failed(uint8\_t \*file, uint32\_t line)

{

/\* USER CODE BEGIN 6 \*/

/\* User can add his own implementation to report the file name and line number,

ex: printf("Wrong parameters value: file %s on line %d\r\n", file, line) \*/

/\* USER CODE END 6 \*/

}

#endif /\* USE\_FULL\_ASSERT \*/

Page | 17

# TOOLS & TECHNIQUES USED

### STM32 Microcontroller (STM32F4 Series):

Executes control logic and interfaces with sensors.

### Water Level Sensor (SEN18) & Temperature Sensor (DS18B20):

Used to detect dry run and overheating conditions.

### Visual Studio Code with ARM GCC Toolchain:

Embedded C code developed and compiled in VS Code.

### LED Simulation:

Indicates geyser ON/OFF state during testing.

## ESP32 :

Sends real-time data to a web interface via Wi-Fi.

### Custom Web Interface(Firebase):

Displays geyser status and alerts (Dry Run / Overheating) remotely.

# CHAPTER 4: RESULTS & DISCUSSIONS

The prototype of the geyser control system was successfully implemented using STM32, SEN18 water level sensor, and DS18B20 temperature sensor. The system was designed to autonomously manage the geyser’s operation by monitoring two critical safety parameters: **water availability** and **temperature threshold**. The following key observations and outcomes were noted during testing:

#### Dry Run Protection Functionality

The SEN18 water level sensor effectively detected the presence or absence of water in the simulated tank.

* + When the sensor indicated **low water level**, the system **disabled the geyser** by turning OFF the LED.
  + This behavior remained consistent across multiple trials, confirming the **reliability of the dry run protection logic**.
  + This feature is crucial in real-world applications where dry run conditions can **damage the heating coil**, thus saving maintenance costs and enhancing safety.

#### Overheating Detection and Control

The DS18B20 sensor provided **real-time temperature data** to the STM32.

* + Once the water temperature crossed the predefined threshold (e.g., 60°C), the system **automatically turned OFF the geyser** (LED OFF).
  + The sensor readings were stable and accurate, and the decision logic responded

instantly to temperature changes.

* + This ensures that **energy wastage and overheating risks are minimized**, making the system safer and more efficient.

#### Geyser Status Indication via LED

An LED was used to represent the geyser’s operational state.

* + When both water level was adequate and temperature was below the threshold, the **LED remained ON**, simulating geyser heating.
  + In any unsafe condition (dry run or overheating), the **LED turned OFF**, indicating

the geyser was shut down.

#### Web-Based Monitoring and Alerts

The integration with ESP32 enabled real-time transmission of sensor data to a web interface.

* + The dashboard displayed water level status, current temperature, and geyser

ON/OFF status.

* + Alerts such as **“Dry Run Detected”** or **“Overheating Warning”** were clearly shown to the user.
  + This component transforms the system into a **smart IoT-enabled appliance**,

allowing users to monitor geyser status remotely via a phone or browser.

#### Overall System Performance

* + The system **operated reliably under all tested conditions**, showing prompt and correct responses.
  + The **control logic was stable**, and sensor feedback loops ensured real-time action.
  + Web interface interaction had minimal latency and enhanced user experience.

#### Discussion

The project demonstrates how **simple sensors**, combined with **real-time embedded control and IoT interfaces**, can dramatically enhance the safety and efficiency of conventional appliances like geysers. The **use of STM32** provided low-latency and robust processing, while **coding in VS Code** allowed flexible firmware development.

Although the current setup is suitable for prototyping and small-scale use, future improvements can include:

* + Integration with **mobile notifications or SMS alerts**,

#### Adding relay control for actual geyser power lines

* + **web-based dashboard** for real-time monitoring and user alerts

## CHAPTER 5: TESTING

Comprehensive testing was conducted to validate the functional accuracy, reliability, and responsiveness of the geyser control system under various simulated conditions. The aim was to ensure that the system accurately detects unsafe conditions (dry run and overheating) and responds appropriately to maintain safety and efficiency.

#### Water Level Sensor Testing (Dry Run Detection)

**Objective:** To verify whether the system correctly identifies low water level conditions and prevents geyser activation.

Method:

* + The water level sensor (SEN18) was manually triggered to simulate both adequate and inadequate water levels.
  + The LED (representing the geyser) was observed for ON/OFF behavior based on sensor input.

#### Observations:

* + When the water level was adequate, the system allowed the LED (geyser) to turn ON.
  + When the water level dropped below the threshold, the LED was immediately turned OFF, simulating geyser shutdown.
  + Multiple tests were conducted to verify consistent response, and in all cases, the dry run logic functioned correctly.

#### Temperature Sensor Testing (Overheating Protection)

**Objective:** To validate the temperature monitoring system and automatic shutdown at high temperatures.

Method:

* + The DS18B20 temperature sensor was placed in environments with gradually

increasing temperature (e.g., heated water).

* + The system was monitored to observe the LED state as temperature increased past the threshold (e.g., 60°C).

#### Observations:

* + - The system accurately tracked temperature in real time.
    - Upon exceeding the defined threshold, the LED turned OFF, simulating the geyser

being shut off to prevent overheating.

* + - The response time was quick and consistent during repeated tests.

#### Combined Sensor Logic Testing

**Objective:** To ensure the control logic correctly handles combined water level and temperature conditions.

Method:

* + Several combinations were tested:
  + Adequate water + Low temperature
  + Adequate water + High temperature
  + Low water + Low temperature
  + Low water + High temperature

#### Observations:

* + The geyser (LED) only turned ON when both conditions were safe.
  + If either water level was low or temperature exceeded the limit, the system prevented geyser activation.

#### Web Dashboard and Alert Testing

**Objective:** To test real-time status monitoring and alert functionality via the web interface.

Method:

* + The system was connected to an ESP32 module to transmit data to a custom web

dashboard.

* + Sensor conditions were varied, and the corresponding display and alert messages were checked.

#### Observations:

* + Water level, temperature, and geyser status were correctly displayed.
  + Alerts such as **"Dry Run Detected"** and **"Overheating Warning"** appeared

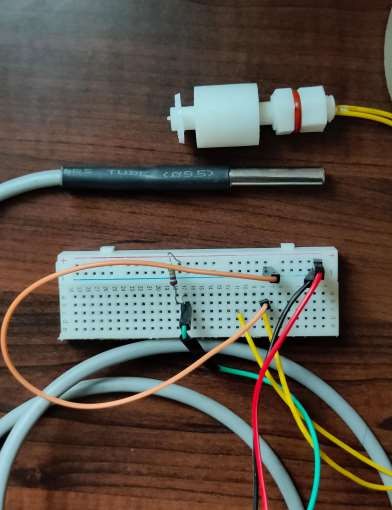
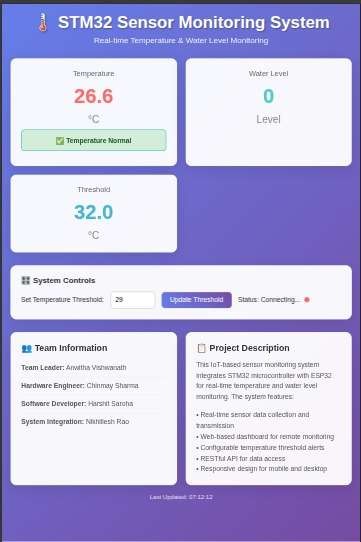
promptly.

* + The dashboard refreshed in near real-time and remained stable over multiple trials.

# CHAPTER 6 CONCLUSION

The implementation of an automated geyser control system with dry run protection and overheating prevention successfully demonstrates the integration of embedded systems and IoT for real-world safety applications. By using a water level sensor (SEN18) and a temperature sensor (DS18B20), the system effectively monitors critical parameters to ensure that the geyser operates only under safe conditions. The STM32 microcontroller executes real-time control logic, turning the geyser ON or OFF based on sensor inputs, and simulates this behavior using an LED. In addition, the integration of a web-based dashboard allows users to monitor the system remotely and receive immediate alerts such as "Dry Run Detected" and "Overheating Warning", thereby enhancing user safety and awareness. The system has been thoroughly tested and shown to be reliable, responsive, and energy-efficient. This project not only prevents potential hardware damage and reduces power consumption but also introduces a scalable model for smart appliance control. With further enhancements like power relay control or mobile integration, this system can be deployed in households or commercial buildings, offering a modern, safe, and intelligent alternative to conventional geyser operation.

**CHAPTER 7 VISUALS**



WebPage using Firebase

Basic Circuit desgin

Page | 24

24

## REFERENCES

1. *R. Herasustanti, A. Abdi, H. Purnata, N. Mailahi, and S. Rian, “Automated Water Heating Management with Internet of Things,” Journal of Telecommunication Network, vol. 11, no. 3, pp. 45–52, 2023.*
2. *D. Viegas, M. Nunes, P. Gonçalves, and R. Ribeiro, “Embedded Model Predictive Control of Tankless Gas Water Heaters to Enhance Users’ Comfort,” MDPI Electronics, vol. 12, no. 5, pp. 233–242, 2023.*
3. *C. Passenberg, D. Meyer, J. Feldmaier, and H. Shen, “Optimal Water Heater Control in Smart Home Environments,” arXiv preprint arXiv:1607.05129, 2016.*
4. *STMicroelectronics, STM32F4 Series Microcontroller Reference Manual, RM0090, Rev 19, 2023.*
5. *Maxim Integrated, DS18B20 Programmable Resolution 1-Wire Digital Thermometer Datasheet, Rev 5, 2021.*
6. *Seed Studio, SEN18 Float Water Level Sensor Technical Manual, Ver. 2.1, 2020.*
7. *Microsoft Corporation, Visual Studio Code User Guide, Version 1.78, 2023.*
8. *Espressif Systems, ESP32 Technical Reference Manual, Version 4.2, 2022.*

Page | 25