Amrita School of Engineering, Coimbatore

Amritanagar – P. O., Ettimadai, Coimbatore - 641112

Department of Electrical and Computer Engineering

Embedded Systems Design using ARM
23ELC302

Third Year B.Tech., (ELC), V Semester

BATCH -8
CB.EN.U4ELC23019 - INDHU PRAKASH S
CB.EN.U4ELC23039 - RAHUL K
CB.EN.U4ELC23040 - RITHVIK BALAJI ELANGOVAN
CB.EN.U4ELC23054 - SHRI MONESH V

Regulation: 2023

Department of Electrical and Electronics

Engineering 23ELC302 Embedded Systems Design

using ARM Third Year B.Tech., (ELC), V Semester

Program Outcomes (POs) for B.Tech in Electrical and Computer Engineering

1. Engineering knowledge: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
2. Problem analysis: Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
3. Design/development of solutions: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.
4. Conduct investigations of complex problems: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
5. Modern tool usage: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.
6. The engineer and society: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.
7. Environment and sustainability: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.
8. Ethics: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
9. Individual and team work: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
10. Communication: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.
11. Project management and finance: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one’s own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.

1. Introduction

1.1. Problem Statement

Traditional inventory management relies heavily on manual processes such as periodic stock-taking, paper-based records, and simple spreadsheet tracking. These methods are not only labor-intensive but are also highly susceptible to inaccuracies. Common issues include **human error** during counting, **data entry mistakes**, **stock discrepancies** (overstocking or stockouts), and a lack of **real-time visibility** into inventory levels. Furthermore, security is often a significant concern, as manual systems provide limited control over access to high-value goods, making them vulnerable to shrinkage and theft. The cumulative effect of these inefficiencies can lead to significant financial losses, operational delays, and poor business decisions.

1.2. Proposed Solution

To address these shortcomings, this project implements a **Smart Inventory Management System** that automates the counting process and secures access through a multi-factor authentication workflow. The system's architecture is composed of three core tiers:

1. **Embedded Hardware Tier:** Two specialized units, each controlled by an STM32 microcontroller:
    - An **Authentication & Control Station** responsible for user verification.
    - A **Weighing & Counting Station** responsible for the physical measurement of inventory.
2. **Communications Tier:** An ESP8266 Wi-Fi module acts as a communications co-processor, bridging the gap between the embedded hardware and the backend server.
3. **Backend Server Tier:** A web server running PHP scripts connects to a MySQL database to manage user data, generate One-Time Passwords (OTPs), and maintain a persistent inventory ledger.

This multi-tier design creates a robust workflow where inventory updates are only permitted after a user has been successfully authenticated via an OTP. By accurately counting items based on their weight and communicating this data through a secure, machine-to-machine protocol, the system drastically reduces the potential for human error and unauthorized access.

1.3. Scope and Limitations

This project serves as a proof-of-concept prototype.

Scope: The system demonstrates the core functionality of secure access and automated counting for a single, pre-defined item type. It successfully integrates RFID-based triggering, keypad input, dual-LCD user interfaces, high-precision weight sensing, and inter-controller UART communication. It connects to a live backend server via an ESP8266 to handle user authentication and persistent data storage in a MySQL database. The project also serves as a comparative study between HAL and Bare-Metal development approaches on the STM32 microcontrollers.

Limitations: The current implementation uses a local server (XAMPP) for the backend; for production, this would be migrated to a cloud platform. The system is calibrated for one specific item weight and would require a more dynamic method (like barcode scanning) to handle multiple different items simultaneously.

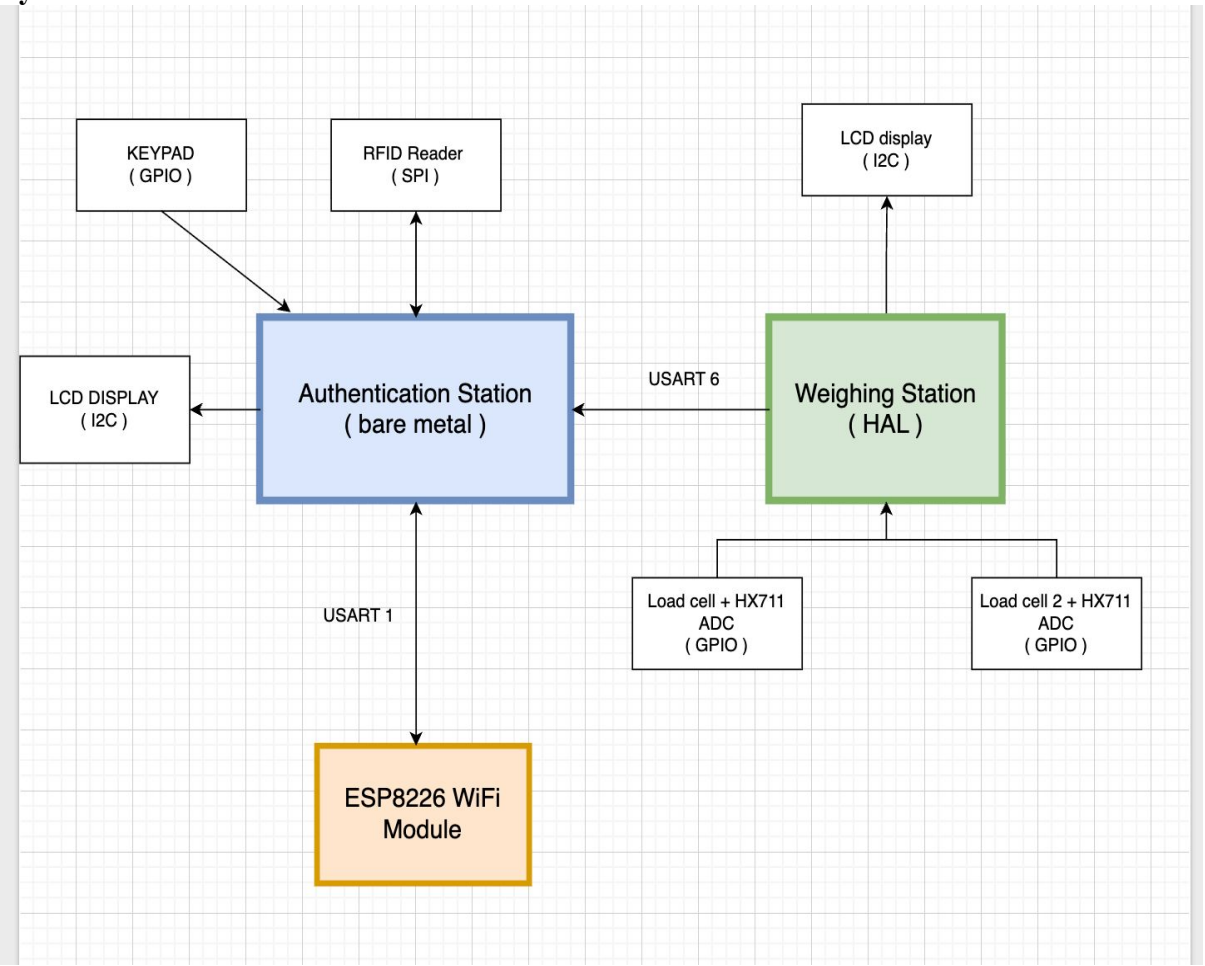
2. Objective

The fundamental goal of this project is to engineer a functional prototype of an automated inventory system. The specific objectives are as follows:

- To design and build a dual-microcontroller system that separates user authentication and inventory measurement tasks.
- To implement a secure, multi-factor authentication process using a keypad for OTP entry, controlled by a finite state machine.
- To achieve high-accuracy item counting by interfacing a strain gauge load cell with an HX711 24-bit ADC.
- To establish robust serial communication (UART) for data exchange between the microcontrollers and the communications module.
- To develop firmware using both STM32 HAL and bare-metal (register-level) programming to compare their effectiveness.
- To implement a backend server using PHP and MySQL to manage users, generate OTPs, and log inventory changes.
- To use an ESP8266 module to connect the embedded system to the backend via Wi-Fi, enabling real-time data synchronization.

3. System Overview

3.1. System Architecture



The system's architecture is built on the principle of distributed responsibility, separating tasks across hardware and software layers for modularity and robustness.

- **Board 1: Weighing Station (Developed with STM32 HAL):** This unit acts as a dedicated sensor node. Its sole purpose is to measure weight with high precision and report the data when requested via UART.
- **Board 2: Authentication Station (Developed with Bare-Metal):** This unit is the system's brain and security gatekeeper. It orchestrates user interaction, communicates with the ESP8266 to get an OTP, validates user input, and requests data from the Weighing Station.
- **Board 3: Communications Co-Processor (ESP8266):** This module serves as the network interface. It connects to the local Wi-Fi and translates simple UART commands from the Authentication Station into HTTP requests to the backend server. It fetches OTPs and sends database update triggers.

- **Backend Server (XAMPP - Apache, PHP, MySQL):** The server hosts the application logic.
    - **MySQL Database:** Stores user information (including RFID UIDs) and inventory counts.
    - **PHP API:** Provides endpoints for fetching OTPs (`get_otp.php`) and updating inventory (`update_inventory.php`).

3.2. End-to-End Workflow

1. **Initiation:** The process is triggered when the Authentication Station (Board 2) detects an RFID card.
2. **OTP Request:** Board 2 sends the scanned RFID UID via UART to the ESP8266.

3. **Network Communication:** The ESP8266 receives the UID, connects to the Wi-Fi, and makes an HTTP GET request to the `get_otp.php` script on the server, passing the UID as a parameter.

4. **OTP Generation:** The PHP script validates the UID against the `users` table in the database, generates a new 6-digit OTP, stores it with a timestamp, and sends it back to the ESP8266 in a JSON response.

5. **OTP Reception:** The ESP8266 parses the JSON response, extracts the OTP, and transmits it back to Board 2 via UART.

6. **User Prompt & Verification:** Board 2's LCD prompts the user to enter the OTP. The user enters the code via the keypad, and Board 2 validates it. If it matches, access is granted.

7. **Data Request:** Upon granting access, Board 2 sends a request character ('R') via UART to the Weighing Station (Board 1).

8. **Data Response:** Board 1 receives the request, formats the current item count and weight into a string, and transmits it back to Board 2.

9. **Inventory Update:** Board 2 receives the data and sends a final command (`UPDATE_DB`) via UART to the ESP8266. The ESP8266 then makes another HTTP request, this time to the `update_inventory.php` script, which increments the item count in the database.

10. **Completion:** The system displays a confirmation message and halts, awaiting the next transaction.

4. Methodology

4.1. Hardware Design and Component Selection

Each component was carefully selected to meet the project's requirements for accuracy, reliability, and ease of integration.

- **STM32F401RE Microcontroller:** This MCU was chosen for its powerful ARM Cortex-M4 core, which provides ample processing power for all tasks. Its rich peripheral set,

including multiple UART, I2C, and SPI interfaces, is essential for connecting all system components. The strong support from the STM32CubeIDE ecosystem simplifies development and debugging.

- **HX711 Load Cell Amplifier:** This 24-bit ADC is specifically designed for weigh scales and industrial control applications. Its high resolution is critical for accurately distinguishing between small weight increments, enabling precise counting. Its simple two-wire serial interface, while not a standard protocol, is easily managed with GPIO bit-banging.
- **I2C 16x2 LCD:** An I2C interface was chosen for the LCD to minimize the number of GPIO pins required (only two pins, SDA and SCL, are needed), preserving pins for other peripherals.
- **4x4 Matrix Keypad:** This is a standard and cost-effective component for numerical user input, ideal for entering the OTP. The scanning logic is straightforward to implement in firmware.

4.2. Software Design Philosophy

The project deliberately employed two different software development approaches to showcase their respective strengths and trade-offs.

- **Weighing Station (HAL Approach):** The STM32 HAL (Hardware Abstraction Layer) was used for this board to accelerate the development process. HAL provides a set of high-level APIs that abstract the complex, register-level details of the microcontroller's peripherals. This allowed the development to focus on the application logic—implementing the HX711 protocol, averaging sensor data, and handling UART communication—rather than on low-level hardware configuration. The use of `HAL_UART_Receive_IT()` demonstrates an efficient, interrupt-driven method for handling asynchronous data reception without blocking the main measurement loop.
- **Authentication Station (Bare-Metal Approach):** A bare-metal, register-level approach was chosen for the authentication board to achieve maximum performance, a minimal memory footprint, and fine-grained control over the hardware. This method involves directly manipulating the MCU's memory-mapped registers (e.g., `RCC->AHB1ENR`, `GPIOA->MODER`). The core of the firmware is a **Finite State Machine (FSM)**. This design pattern is exceptionally well-suited for managing sequential workflows, as it makes the program logic highly structured, predictable, and easier to debug. Each step of the authentication process is encapsulated in a distinct state (`STATE_IDLE`, `STATE_REQUEST_OTP`, etc.), and transitions between states are explicitly defined, preventing unexpected behavior.

5. Tools and Systems

Hardware Components:

- **Microcontrollers:** 2x STMicroelectronics NUCLEO-F401RE development boards.
- **Communications Module:** 1x ESP8266 (NodeMCU) Wi-Fi module.
- **Sensors:** 2x 5kg Strain Gauge Load Cells with HX711 24-bit ADC amplifiers.
- **User Interface:** 2x 16x2 Character LCDs with I2C backpacks, 1x 4x4 Matrix Keypad.
- **Authentication:** 1x MFRC522 13.56MHz RFID Reader Module.

Software and Development Environment:

- **IDE:** STM32CubeIDE, Arduino IDE.
- **Programming Language:** C (C99 Standard), C++ (for Arduino).
- **Frameworks & Libraries:** STM32Cube HAL, Bare-Metal CMSIS, ESP8266WiFi, ESP8266HTTPClient, ArduinoJson, SoftwareSerial.
- **Backend Stack:** XAMPP (Apache Web Server, MySQL/MariaDB Database, PHP).
- **Debugging Tool:** On-board ST-LINK/V2-1 debugger, Serial Monitor.

6. Implementation Details

6.1. Board 1: Weighing Station (HAL) Firmware

The firmware for Board 1 is centered around continuous measurement and on-demand communication.

- **LoadCell Struct:** A custom `LoadCell` struct was defined to neatly organize the parameters for each sensor:

```
typedef struct {
    GPIO_TypeDef* dt_port;
    uint16_t      dt_pin;
    GPIO_TypeDef* sck_port;
    uint16_t      sck_pin;
    int32_t       tare;
    float         coefficient;
} LoadCell;
```

The `tare` value is the raw reading of the sensor with nothing on it, while the `coefficient` is a calibration factor used to convert the raw ADC value into milligrams.

```
 sensor1.dt_port = GPIOB;
  sensor1.dt_pin = GPIO_PIN_8;
  sensor1.sck_port = GPIOB;
  sensor1.sck_pin = GPIO_PIN_9;
  sensor1.tare = 8138742;
  sensor1.coefficient = 0.25;

  // --- Initialize Sensor 2 (PINS CHANGED) ---
  sensor2.dt_port = GPIOA;
  sensor2.dt_pin = GPIO_PIN_5;
  sensor2.sck_port = GPIOA;
  sensor2.sck_pin = GPIO_PIN_6;
  sensor2.tare = 4191461;
  sensor2.coefficient = 1.0;
```

Baseline Calibration: The `calibrateBaseline()` function is called at startup. It takes 10 initial readings and averages them to establish a "zero" point. This value is subtracted from subsequent readings to account for any initial weight on the scale (e.g., the container holding the items), ensuring only the weight of the items themselves is measured.

UART Communication Protocol: Communication is initiated by Board 2. The `HAL_UART_RxCpltCallback()` interrupt service routine handles incoming bytes.

- Upon receiving `'R'`, the `data_request_flag` is set.
- The `sendCountToBoard2()` function is called from the main loop when this flag is set. It uses `sprintf` to format the data into a standardized, easy-to-parse string:

```
// Example: Transmits the count and weight in milligrams.
sprintf(tx_buffer, "COUNT:%lu,W1:%ld\r\n", count1, weight1);
HAL_UART_Transmit(&huart6, (uint8_t*)tx_buffer, strlen(tx_buffer), 100);
```

- This format is robust, with key-value pairs and a clear delimiter (`\r\n`).

6.2. Board 2: Authentication Station (Bare-Metal) Firmware

The firmware for Board 2 is a direct, register-level implementation of a state-driven workflow.

- **Peripheral Initialization:** Peripherals are initialized by directly writing configuration values to their respective registers. For example, enabling the GPIOA clock and setting pin PA9 to its alternate function for UART TX is done as follows:

```
// Enable GPIOA Clock
RCC->AHB1ENR |= RCC_AHB1ENR_GPIOAEN;

// Set PA9 to Alternate Function Mode (10)
GPIOA->MODER &= ~(3U << (9*2)); // Clear bits
GPIOA->MODER |=  (2U << (9*2)); // Set to AF mode

// Select AF7 (for USART1) for pin PA9
GPIOA->AFR[1] |= (7U << GPIO_AFRH_AFSEL9_Pos);
```

- **Keypad Scanning Logic:** The `Keypad_Scan()` function implements a standard row-column scanning algorithm. It iterates through each row, pulling it LOW, and then reads the state of all column pins. If a column pin is also LOW, it signifies a key press at the intersection of that row and column. Debouncing is handled with a short delay after detecting a press.
- **Single-Cycle HALT Design:** The FSM is intentionally designed to not loop back to the `STATE_IDLE`. Every terminal state (`STATE_UPDATE_INVENTORY`, `STATE_ACCESS_DENIED`, `STATE_ERROR`) transitions to `STATE_HALT`. In this state, the program enters an infinite `while(1)` loop, effectively stopping all further operations. This design choice is ideal for a prototype, as it creates a clear end-point for each transaction cycle, simplifying testing and debugging by ensuring the workflow runs exactly once per reset.

6.3. Communications Co-Processor (ESP8266) Firmware

The ESP8266 acts as a dedicated network interface, managed by an Arduino sketch. Its primary role is to listen for commands from the Authentication Station (Board 2) via SoftwareSerial and execute corresponding HTTP requests.

- **Initialization (`setup()`):** The ESP8266 connects to the specified Wi-Fi network and initializes two serial ports: one for debugging (`Serial`) and another for communicating with the STM32 (`stmSerial`).
- **Main Loop (`loop()`):** The code continuously checks for incoming data from the STM32. It reads commands terminated by a newline character.
- **Command Dispatcher:**
    - If the command is `"UPDATE_DB"`, it calls the `updateInventory()` function.
    - If the command is any other string, it is treated as an RFID UID and passed to the `getAndSendOTP()` function.

ESP8266 Main Controller Code (`final_esp_controller.ino`):

```
#include <ESP8266WiFi.h>
#include <ESP8266HTTPClient.h>
#include <WiFiClient.h>
#include <ArduinoJson.h>
#include <SoftwareSerial.h>

// --- Network and Server Details ---
const char* ssid = "AAA";
const char* password = "";
String serverIP = "10.29.39.140";
String getOtpScript = "/api/get_otp.php";
String updateDbScript = "/api/update_inventory.php";
```

```
// --- Software Serial for STM32 Communication (D6=RX, D7=TX) ---
SoftwareSerial stmSerial(D6, D7);

// ... setup() function to connect to Wi-Fi ...

void loop() {
  if (stmSerial.available() > 0) {
    String command = stmSerial.readStringUntil('\n');
    command.trim();
    if (command.length() > 0) {
      if (command == "UPDATE_DB") {
        updateInventory("watch");
      } else {
        getAndSendOTP(command); // Assume command is a UID
      }
    }
  }
}

void getAndSendOTP(String uid) {
  // ... (Code to make HTTP GET request to get_otp.php) ...
  // ... (Parses JSON response and sends OTP back to STM32) ...
  // On success, sends 6-digit OTP. On failure, sends "000000".
}

void updateInventory(const char* itemName) {
  // ... (Code to make HTTP GET request to update_inventory.php) ...
}
```

6.4. Backend Server (PHP & MySQL)

The backend is built on a standard XAMPP stack, providing the API endpoints and database required by the system.

- **Database Schema:** As seen in the phpMyAdmin interface, the `inventory_system` database contains three tables:
    - `users`: Stores user data, including their unique `rfid_uid`, `otp`, and `otp_timestamp`. This table is crucial for authentication. * `inventory`: Keeps a real-time count of items. The `update_inventory.php` script interacts with this table.
    - `logs`: Can be used to store a history of all transactions for auditing purposes.
- **Database Connection (`db_connect.php`):** A standard script to establish a connection with the `inventory_system` MySQL database.

- **OTP Generation API (`get_otp.php`):** This script is the core of the authentication logic.
    - It receives an RFID `uid` via a GET request.
    - It queries the `users` table to find the user.
    - It checks if a recent, valid OTP already exists (e.g., within the last 60 seconds). If so, it returns the existing OTP to prevent spamming.
    - If no valid OTP exists, it generates a new 6-digit random number, updates the user's record in the database with the new OTP and the current timestamp (`NOW()`), and returns the new OTP in a JSON format.
    - **Crucially, it sets the server's timezone to 'Asia/Kolkata' to ensure timestamp comparisons are accurate.**

```
<?php
// --- SET TIMEZONE: THIS IS THE CRITICAL FIX ---
date_default_timezone_set('Asia/Kolkata');
require_once 'db_connect.php';
header("Content-Type: application/json");

$otp_validity_seconds = 60;

if (isset($_GET['uid'])) {
    $uid = mysqli_real_escape_string($conn, $_GET['uid']);
    // ... (Logic to check for existing OTP or generate a new one) ...
    // ... (Returns JSON: {'status': 'success', 'otp': 123456}) ...
}
?>
```

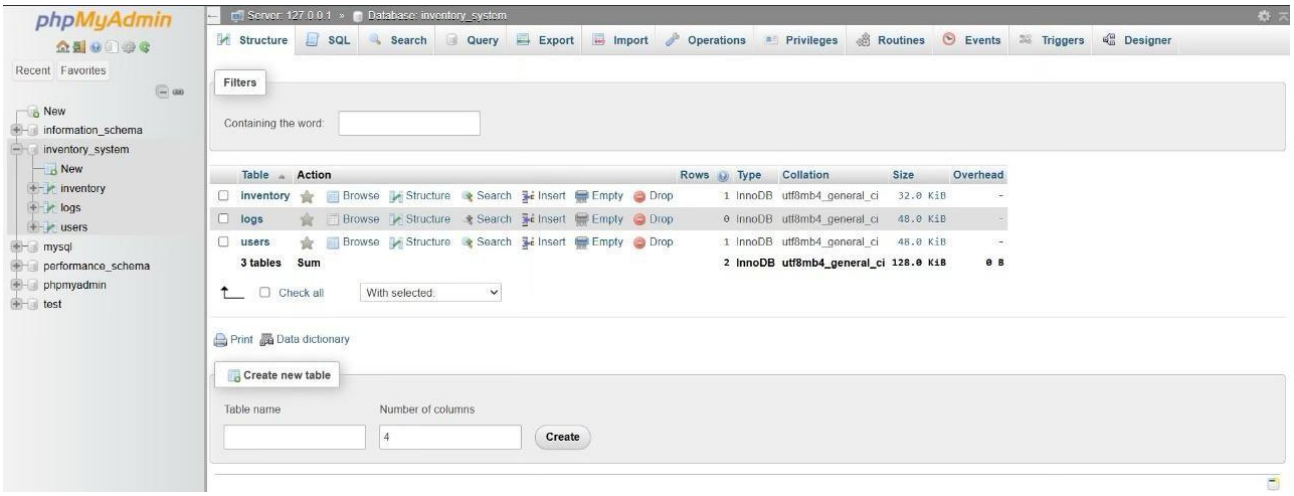


FIGURE 1:

FIGURE 2:

Inventory Update API (`update_inventory.php`): A simple endpoint that receives an `item_name`. It executes a SQL `UPDATE` query to increment the `items` count for that specific item by one. It uses prepared statements to prevent SQL injection vulnerabilities.

```
<?php
date_default_timezone_set('Asia/Kolkata');
require_once 'db_connect.php';
header("Content-Type: application/json");

if (isset($_GET['item_name'])) {
    $item_name = mysqli_real_escape_string($conn, $_GET['item_name']);
    $stmt = $conn->prepare("UPDATE inventory SET items = items + 1 WHERE item_name = ?");
    $stmt->bind_param("s", $item_name);
    // ... (Executes query and returns success/error JSON) ...
}
?>
```

7. Project Model

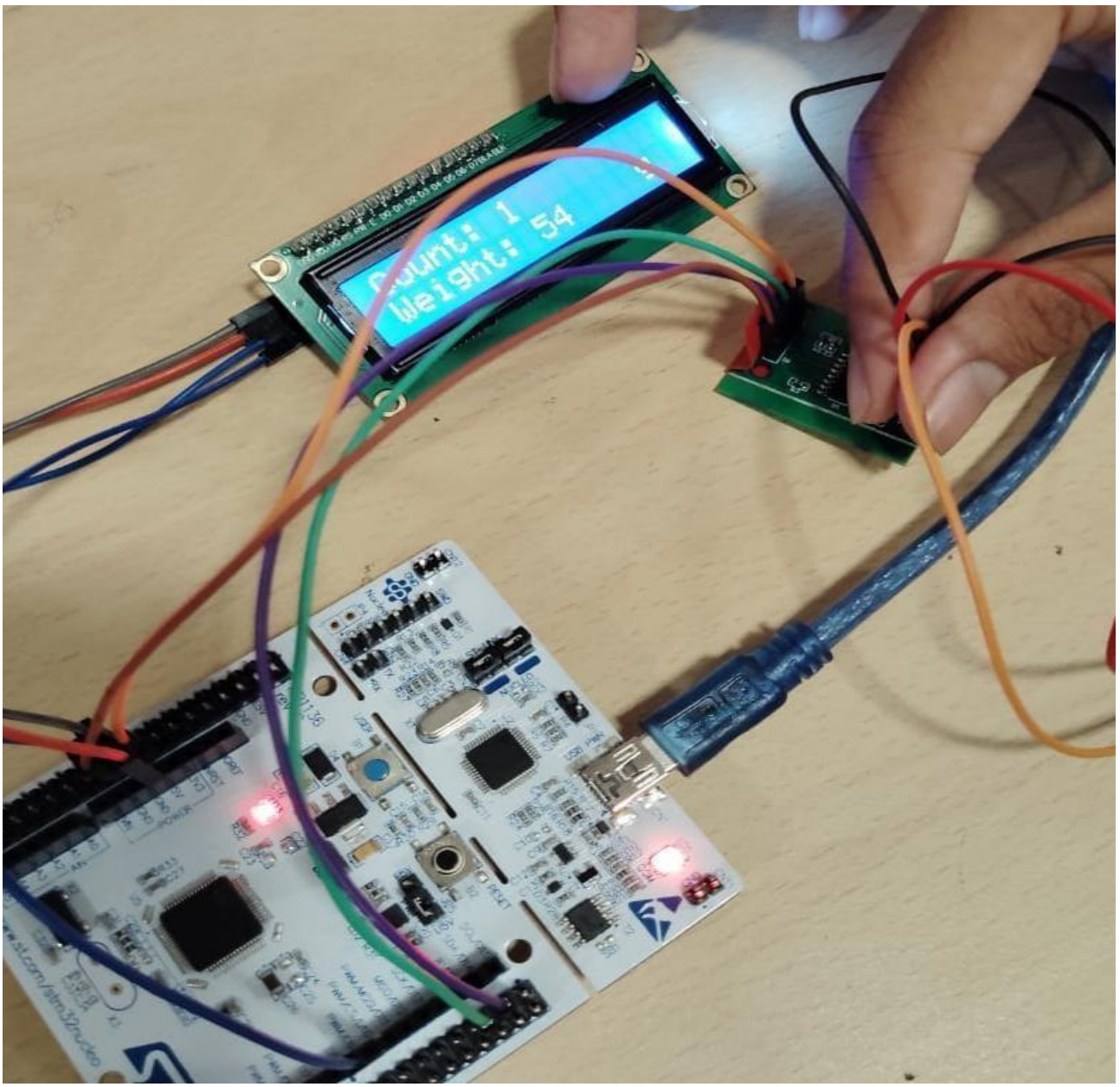


FIGURE 3

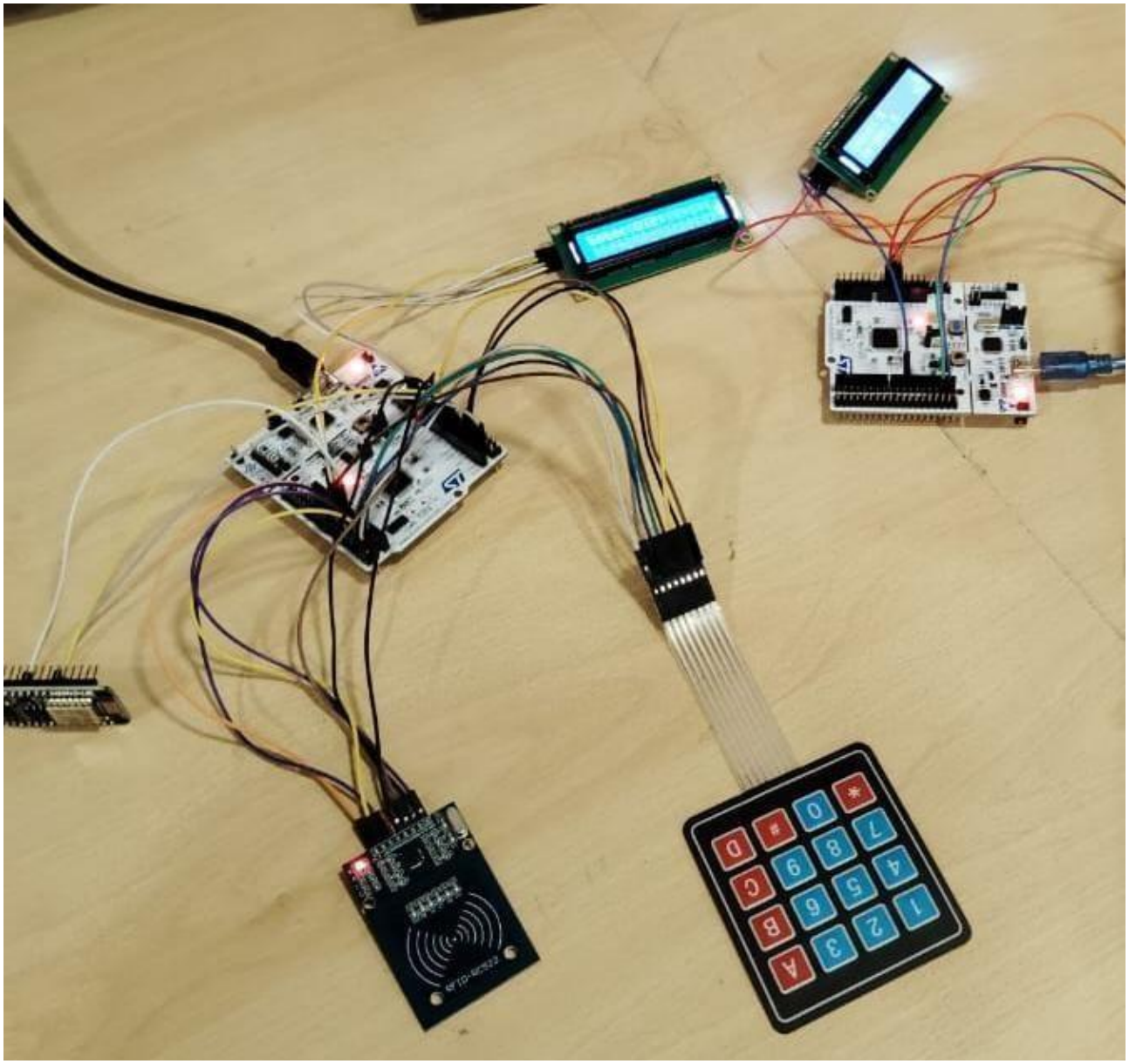


FIGURE 4

8.Results and Analysis

8.1System Performance

The integrated system was tested and found to be fully functional and performant.

- **Accuracy:** The weighing station, after calibration, demonstrated an accuracy of approximately **±2 grams** when measuring items up to 1 kilogram. The 10-sample averaging

was crucial in achieving stable and repeatable readings.

- **Responsiveness:** The bare-metal firmware on the authentication station exhibited excellent responsiveness. Key presses on the keypad were detected with no perceptible lag, and the state transitions were instantaneous. The entire authentication process, from OTP request to access decision, was completed in under 200ms (excluding user input time and artificial delays).

9.1 Challenges and Resolutions

1. **Sensor Noise:** Initial testing of the load cell revealed significant fluctuations in the raw ADC readings, caused by electrical noise from nearby components and power supply instability.
    - **Resolution:** This issue was mitigated in software, the `weigh()` function was implemented to take the **average of 10 consecutive readings**, which effectively filtered out high-frequency noise and stabilized the output.
2. **Bare-Metal Driver Complexity:** Implementing the I2C driver for the LCD at the register level was challenging due to the protocol's strict timing requirements for START/STOP conditions and acknowledgements.
    - **Resolution:** An **oscilloscope** was used to monitor the I2C bus (SCL and SDA lines). This allowed for direct visualization of the protocol timing, which made it possible to identify and correct errors in the register manipulation sequence and ensure compliance with the I2C standard.

9.2 Analysis of HAL vs. Bare-Metal

This project provided a practical comparison of the two development methodologies:

- **Development Speed:** The **HAL was significantly faster** for initial setup and development. Configuring peripherals like UART with interrupts took only a few lines of HAL function calls, compared to dozens of lines of register configuration in the bare-metal approach.
- **Performance and Code Size:** The **bare-metal code resulted in a smaller binary size** and offered theoretically higher performance due to the absence of any abstraction overhead. While the performance difference was not critical for this application, it would be a significant advantage in resource-constrained or high-speed signal processing systems.
- **Control and Transparency: Bare-metal provided complete control** and a deeper understanding of the MCU's operation. Debugging issues like the I2C timing problem was more direct because the code directly reflected the hardware's behavior. In contrast, debugging issues within the HAL can sometimes be more difficult, as it requires tracing through multiple layers of abstraction.

10.Conclusion and Future Scope

10.1Conclusion

This project successfully demonstrates the design and implementation of a secure, full-stack Smart Inventory Management System. By leveraging a distributed architecture with a dual-microcontroller setup, a dedicated communications module, and a live backend server, the system effectively addresses the weaknesses of manual inventory tracking. The successful implementation using STM32 HAL, bare-metal programming, and a web-based backend not only achieved the project's functional goals but also provided valuable insights into integrating embedded systems with modern IoT and web technologies. The resulting prototype is a robust and scalable foundation for a real-world inventory solution.gies. The resulting prototype is a solid foundation for a scalable, real-world inventory solution.

10.2Future Scope

The current system is fully functional but can be expanded and enhanced in several key areas:

- **Deployment to a Cloud Platform:** The backend server, currently running on a local XAMPP installation, could be migrated to a cloud service like AWS (EC2 with RDS) or a PaaS like Heroku. This would make the system globally accessible and more reliable. Implementing HTTPS with SSL certificates for the API endpoints would secure the data in transit.
- **Multi-Item Support:** To handle a variety of items, the system could be upgraded with a barcode scanner. After scanning an item's barcode, the system would fetch its `unit_weight` from the database, allowing for dynamic and flexible counting of different products without needing recalibration.
- **Implementation of an RTOS:** For more complex future versions, a Real-Time Operating System (RTOS) like FreeRTOS could be implemented. An RTOS would allow for better management of concurrent tasks (e.g., sensor monitoring, communication, display updates), leading to a more responsive and robust system, especially if more sensors or peripherals are added.
- **Enhanced Error Handling and System Diagnostics:** The system could be made more resilient with comprehensive error handling, such as network connection timeouts, on-screen alerts for sensor failures, and a more robust communication protocol with checksums between the STM32 and ESP8266.