

Fall 2024

MEDDI-BUDDY

By Noor HADDAD - 202101165 and Wissam AL BOUSTANY - 201901136

CTE 558

Table of Contents

[1. Abstract 3](#_Toc187389600)

[2. Introduction 3](#_Toc187389601)

[3. Hardware description 3](#_Toc187389602)

[ ESP32 WROOM 32U Microcontroller 3](#_Toc187389603)

[ Real-Time Clock (RTC) Module 3](#_Toc187389604)

[ Servo Motors and LEDs 3](#_Toc187389605)

[ LCD Display (I2C) 3](#_Toc187389606)

[ Joystick (Analog) 4](#_Toc187389607)

[ Power Supply 4](#_Toc187389608)

[ Wiring and Connectors 4](#_Toc187389609)

[4. Software description 5](#_Toc187389610)

[ Programming Platform 5](#_Toc187389611)

[ Core Functionalities 5](#_Toc187389612)

[ User Modes 6](#_Toc187389613)

[ Scheduling Logic 7](#_Toc187389614)

[ Extensibility 7](#_Toc187389615)

[5. Problems encountered 8](#_Toc187389616)

[ Scheduler Code Issues 8](#_Toc187389617)

[ Hardware Constraints of Arduino Uno 8](#_Toc187389618)

[ Simulation Challenges in Proteus 8](#_Toc187389619)

[ Integration and Testing 8](#_Toc187389620)

[ Prototype Assembly and Bluetooth Integration 8](#_Toc187389621)

[ Joystick Malfunction 8](#_Toc187389622)

[6. Future expansions 9](#_Toc187389623)

[ IoT Integration and Alerts 9](#_Toc187389624)

[ Pill Counting and Verification 9](#_Toc187389625)

[ Remote Dosage Adjustment 9](#_Toc187389626)

[7. Appendix 9](#_Toc187389627)

List of figures

[Figure 1: Block diagram of complete system. 4](#_Toc187390595)

[Figure 2: MIT application inventor Blocks of code for Meddi-Buddy. 5](#_Toc187390596)

[Figure 3: User interface of the mobile app. 6](#_Toc187390597)

[Figure 4: Summary flowchart of the complete system. 7](#_Toc187390598)

# Abstract

This project “Meddi-Buddy” involves the development of an advanced medication dispenser. The system is designed to address challenges in medication adherence by automating the dispensing process, ensuring timely delivery, and providing user-friendly scheduling both, hardware through the buttons and joystick or via a Bluetooth-connected mobile application. Key hardware components include servo motors for dispensing pills, an RTC module for precise timing, a joystick for intuitive navigation, and an LCD display for real-time display and feedback. The dispenser supports customizable schedules for multiple medications, adjustable through manual input or the mobile app. The integration of the Bluetooth functionality enables seamless remote configuration and monitoring, making this system a reliable and efficient solution for individuals requiring strict medication management.

# Introduction

Medication adherence plays a crucial role in managing health conditions effectively, especially for individuals requiring routine doses, such as those with chronic illnesses or the elderly. Despite its importance, many patients struggle to adhere to prescribed schedules due to forgetfulness, complexity of regimens, or physical limitations. This project aims to address these challenges by designing a smart medication dispenser.

The proposed system combines hardware and software to automate and streamline the medication dispensing process. At its core is the ESP32 WROOM 32U microcontroller, which coordinates various components, including a real-time clock module for precise timing, servo motors for dispensing, and a joystick for interactive navigation. An LCD display provides real-time updates, while a Bluetooth-connected mobile application allows for remote scheduling and monitoring.

This innovative approach ensures that medications are dispensed accurately and on time, reducing the burden on patients and caregivers. By incorporating user-friendly features and expanding on future capabilities like Bluetooth connectivity, this project seeks to deliver a reliable solution to improve medication adherence and overall quality of life.

# Hardware description

The hardware components used in this project are selected to ensure efficient and accurate operation of the medication dispenser. Below is a detailed description of the key hardware elements:

## **ESP32 WROOM 32U Microcontroller**

* Acts as the central processing unit of the system.
* Manages the communication between components and executes the dispensing schedule.
* Provides Bluetooth connectivity for integration with a mobile application.

## **Real-Time Clock (RTC) Module**

* Ensures precise timing for medication dispensing.
* Keeps track of current date and time, even during power interruptions.

## **Servo Motors and LEDs**

* Responsible for the physical dispensing of pills from their compartments.
* Responsible for alerting the user that it is time to take the drugs.

## **LCD Display (I2C)**

* Displays real-time information, including current time, scheduled doses, and system status.
* Provides user feedback during interaction with the system.

## **Joystick (Analog)**

* Facilitates user interaction for navigating and selecting menu options.
* Enables manual setup of dispensing schedules and system adjustments.
* **Push Buttons:**
* Used for mode selection, confirming or cancelling inputs, and toggling between settings.
* Provides a secondary input method alongside the joystick.

## **Power Supply**

* Provides stable and sufficient power to the entire system.
* Ensures uninterrupted operation of the ESP32 and other components.

## **Wiring and Connectors**

* Facilitates communication and power distribution among components.
* Ensures reliable connections for efficient system operation.

The hardware is organized into a cohesive system with the ESP32 serving as the central hub, coordinating all sensors and actuators operations. This setup ensures reliability and scalability, enabling future enhancements such as additional compartments or remote monitoring as shown in Figure 1.

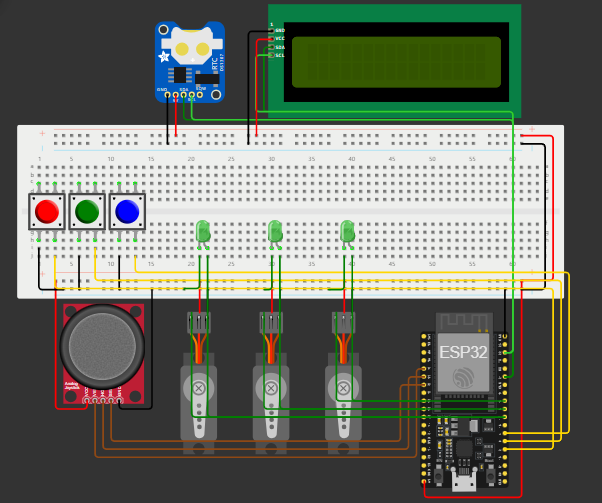


Figure : Block diagram of complete system.

# Software description

The software developed for this project ensures seamless operation by integrating hardware components and providing an intuitive interface for users. It is designed with modularity and scalability in mind to support comprehensive functionality. Below are the main features and components of the software:

## Programming Platform

* The software is written in C using the Arduino IDE.
* Libraries such as Wire.h, LiquidCrystal\_I2C.h, RTClib.h, and ESP32Servo.h are utilized for component-specific functionalities.

## Core Functionalities

* Real-Time Clock Integration: Interfaces with the RTC module to maintain accurate scheduling and timing for medication dispensing.
* Servo and LED Control: Controls the movement of servos and the lighting of LEDs for precise pill dispensing based on the set schedule.
* LCD Display Management: Updates the LCD with real-time information, including current time, scheduled doses, and user prompts.
* Joystick Navigation: Allows users to navigate through menus and set schedules using an analog joystick.
* Push Button Operations: Provides manual controls for mode selection and confirmation of settings.
* Bluetooth Integration: Enables a mobile application to configure schedules, monitor the dispensing process, and receive real-time notifications as shown in Figure 2 and Figure 3.

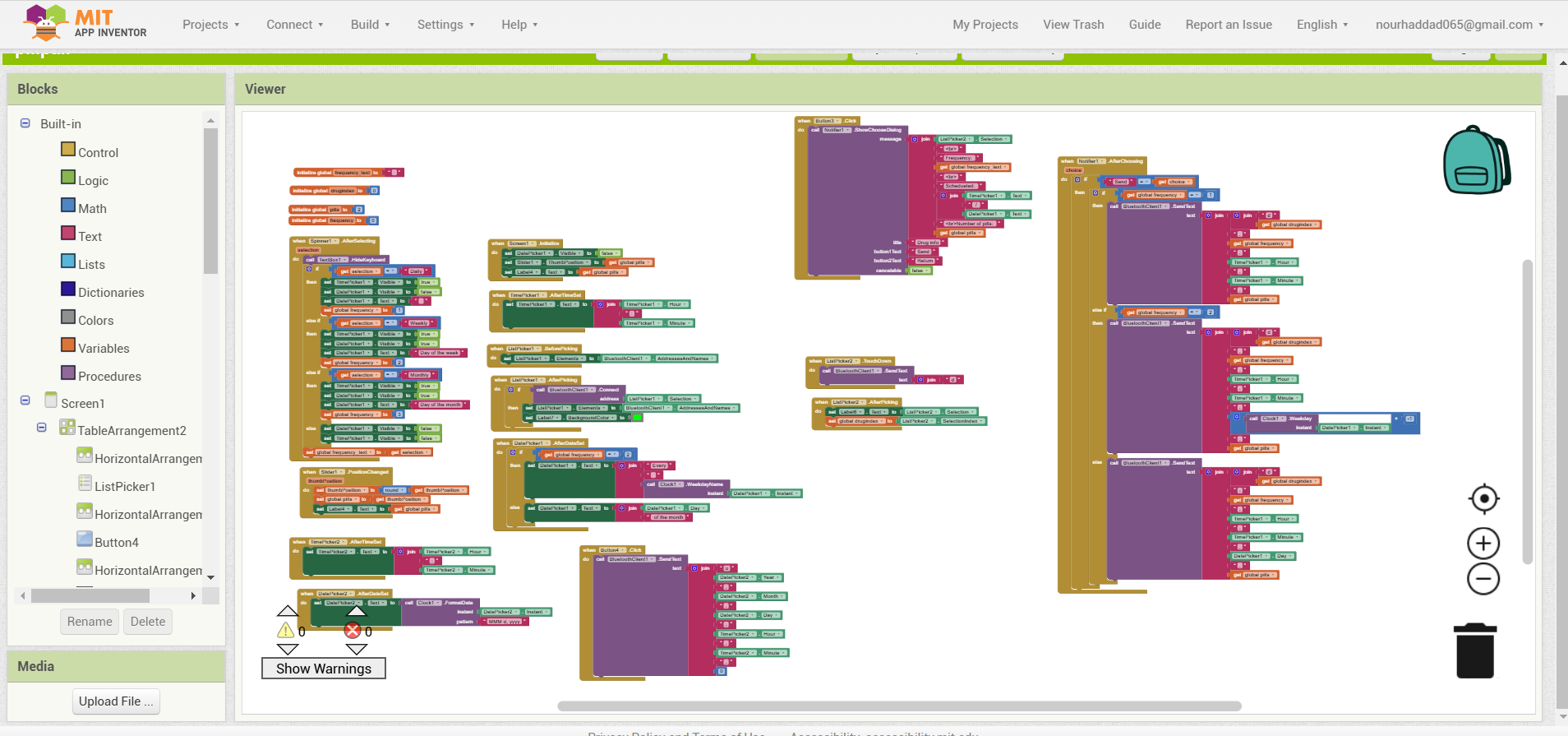


Figure : MIT application inventor Blocks of code for Meddi-Buddy.

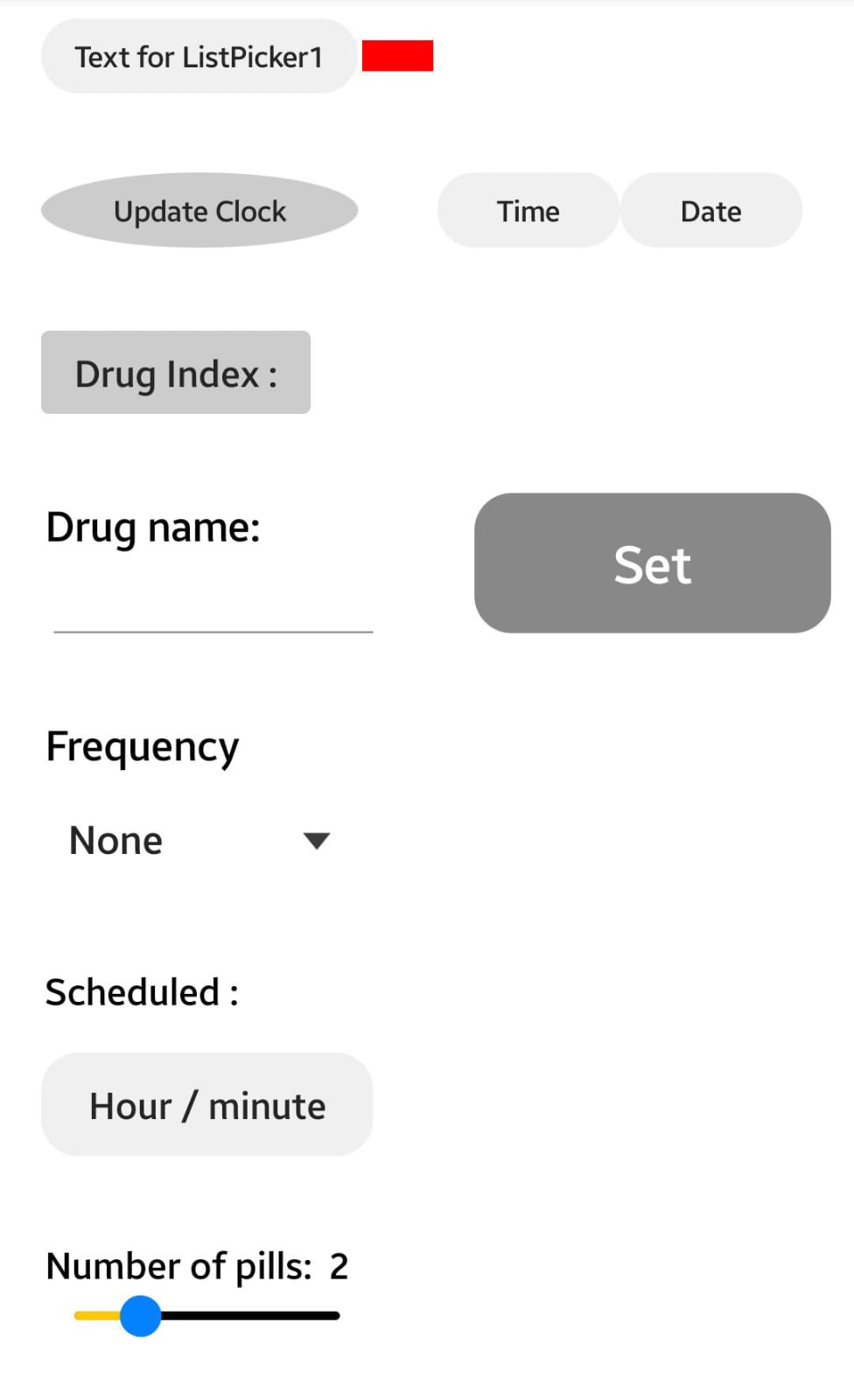


Figure : User interface of the mobile app.

## User Modes

* Normal Mode: Displays the current time and checks schedules for medication dispensing. Activates servos when the schedule matches the RTC.
* Edit Time Mode: Allows users to update the RTC time using the joystick.
* Scheduling Mode: Facilitates the creation or modification of medication schedules using joystick input and the LCD for feedback.
* Mobile Control Mode: Provides seamless remote control through the Bluetooth-connected mobile application for the user to edit time and schedule medication times without using the hardware components of the device.

The system flowchart as shown in Figure 4 showcases the process of the entire code.

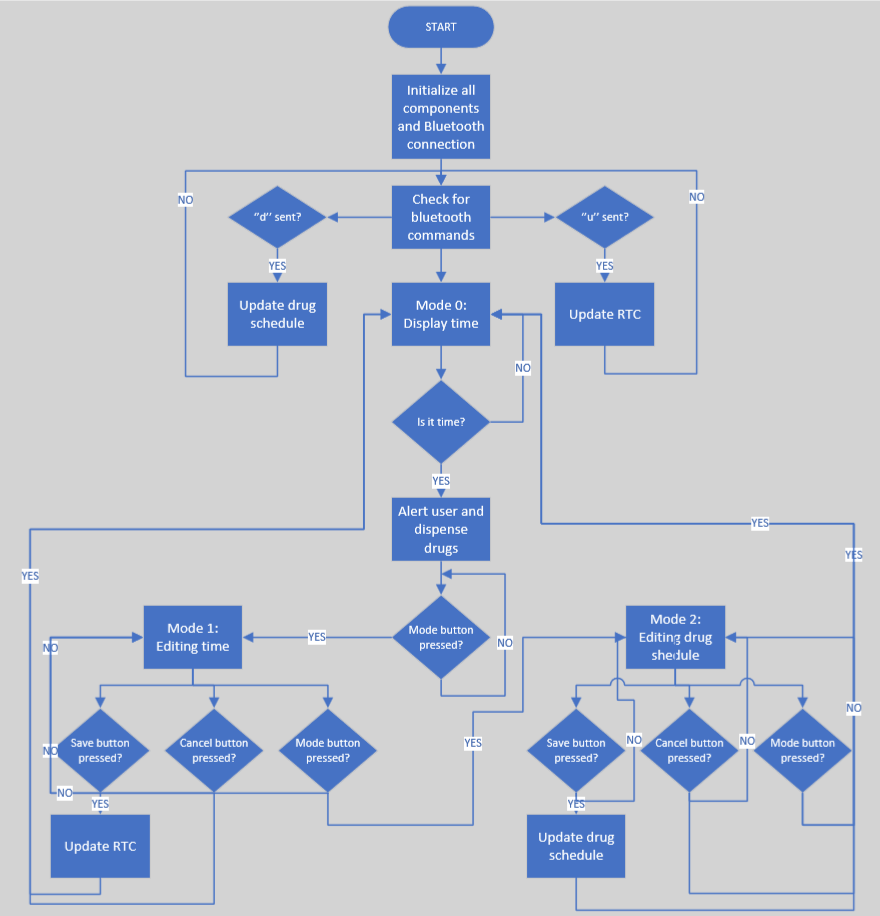


Figure : Summary flowchart of the complete system.

## Scheduling Logic

* Supports daily, weekly, and monthly medication schedules.
* Users can define the frequency, day, hour, minute, and number of pills for each medication slot.
* Schedule synchronization between the app and the dispenser ensures accurate operation.

## Extensibility

* The software’s modular architecture allows for future additions such as voice alerts, cloud connectivity, and advanced health monitoring features.
* The software has been rigorously tested under various conditions to ensure robust and reliable performance, making the medication dispenser an efficient and user-friendly solution for enhancing medication adherence.

# Problems encountered

During the development of Meddi-Buddy, several challenges were encountered and resolved, as outlined below:

## Scheduler Code Issues

Initially, the project followed a scheduler-based code to manage medication timings. However, after three full days of debugging and troubleshooting numerous errors, we were unable to achieve a functional implementation. However, we managed to implement the system using a task-dispatcher ‘FreeRTOS’ which is embedded in all ESP chips (will be shown in the presentation and is uploaded on GitHub).

## Hardware Constraints of Arduino Uno

While transitioning to a simpler coding approach, we found that the Arduino Uno did not have enough pins to support all required components, including a Bluetooth module for the mobile app. This limitation led us to switch to the ESP32 WROOM 32U, which provided built-in Bluetooth functionality and sufficient pins for the project.

## Simulation Challenges in Proteus

The Proteus simulation environment lacked a compatible library for the ESP32 WROOM 32U and was excessively slow. As a workaround, we switched to Wokwi, an online simulation platform. Although Wokwi did not support the RTC DS3231 module directly, it included the DS1307 module. To bridge this gap, we used the RTClib.h library, which is compatible with both RTC modules and successfully integrated the hardware and simulation environments.

## Integration and Testing

Each hardware component and software function were tested individually to ensure proper operation. The entire system was then assembled and tested in stages, resolving issues incrementally.

## Prototype Assembly and Bluetooth Integration

Once the simulation worked seamlessly, we assembled the hardware and added the Bluetooth functionality. The mobile app was developed using MIT App Inventor to interact with the ESP32, enabling remote scheduling and monitoring.

## Joystick Malfunction

The joystick used for navigation proved to be slightly faulty during testing. Although the system is fully functional, replacing the joystick in future iterations is recommended.

Through iterative testing and problem-solving, we successfully created a reliable and efficient medication dispenser prototype, overcoming significant technical challenges during development.

# Future expansions

The current design of the medication dispenser provides a solid foundation for functionality and reliability. Several future enhancements could further improve the usability, efficiency, and scope of the project:

## IoT Integration and Alerts

* Connect the dispenser to Wi-Fi to enable real-time data synchronization with cloud platforms.
* Send notifications to caregivers or guardians if the user misses a scheduled dose.
* Allow healthcare providers to monitor medication adherence remotely.

## Pill Counting and Verification

* Integrate an IR sensor or similar mechanism to detect when a pill is successfully dispensed and taken.
* Use this data to log medication usage and identify missed doses.

## Remote Dosage Adjustment

* Develop features that enable doctors to adjust medication schedules or dosages remotely through a secure cloud interface, improving flexibility and care personalization.

# Appendix

Scheduler code:  
main.ino:  
#include "hardware.h"

#include "tasks.h"

void setup() {

  // 1) Initialize hardware (LCD, RTC, servos, pins, etc.)

  hardwareInit();

  // 2) Create tasks

  //    - We'll pin the tasks as in the previous example, or you can omit the last param for "auto" core assignment.

  xTaskCreatePinnedToCore(TaskCheckButtons, "CheckButtons", 2048, NULL, 2, NULL, 0);

  // Mode tasks

  xTaskCreatePinnedToCore(TaskMode0, "Mode0", 4096, NULL, 1, &handleMode0, 1);

  xTaskCreatePinnedToCore(TaskMode1, "Mode1", 4096, NULL, 1, &handleMode1, 1);

  xTaskCreatePinnedToCore(TaskMode2, "Mode2", 4096, NULL, 1, &handleMode2, 1);

  // mode=0 initially, so we want only TaskMode0 running.

  // Suspend TaskMode1 & TaskMode2

  vTaskSuspend(handleMode1);

  vTaskSuspend(handleMode2);

  Serial.println("Setup complete. Mode0 task running, others suspended.");

}

void loop() {

  // Minimal or empty. FreeRTOS tasks do the heavy lifting.

  vTaskDelay(1);

}

Tasks.h:  
#ifndef TASKS\_H

#define TASKS\_H

#include <Arduino.h>

// We need to declare TaskHandle\_t for each mode

extern TaskHandle\_t handleMode0;

extern TaskHandle\_t handleMode1;

extern TaskHandle\_t handleMode2;

// Mode tasks

void TaskCheckButtons(void \*pvParams);

void TaskMode0       (void \*pvParams);

void TaskMode1       (void \*pvParams);

void TaskMode2       (void \*pvParams);

#endif

Tasks.cpp:  
#include "tasks.h"

#include "hardware.h"  // So we can use 'mode', the LCD, etc.

// Task handles

TaskHandle\_t handleMode0 = NULL;

TaskHandle\_t handleMode1 = NULL;

TaskHandle\_t handleMode2 = NULL;

/\*\*

 \* TaskCheckButtons:

 \*   Continuously reads BUTTON\_PIN\_1 to cycle through modes (0->1->2->0).

 \*   Suspends the old mode task and resumes the new one.

 \*/

void TaskCheckButtons(void \*pvParams) {

  static int lastMode = 0;

  for (;;) {

    if (digitalRead(BUTTON\_PIN\_1) == LOW) {

      vTaskDelay(200 / portTICK\_PERIOD\_MS); // debounce

      mode = (mode + 1) % 3;

      Serial.printf("Mode changed to: %d\n", mode);

      // Suspend the old mode's task

      if (lastMode == 0 && handleMode0) vTaskSuspend(handleMode0);

      if (lastMode == 1 && handleMode1) vTaskSuspend(handleMode1);

      if (lastMode == 2 && handleMode2) vTaskSuspend(handleMode2);

      // Resume the new mode's task

      if (mode == 0 && handleMode0) vTaskResume(handleMode0);

      if (mode == 1 && handleMode1) vTaskResume(handleMode1);

      if (mode == 2 && handleMode2) vTaskResume(handleMode2);

      lastMode = mode;

    }

    vTaskDelay(50 / portTICK\_PERIOD\_MS);

  }

}

/\*\*

 \* TaskMode0 (Normal Dispensing & RTC Display):

 \*   Runs in a loop, but only if this task is not suspended.

 \*/

void TaskMode0(void \*pvParams) {

  for (;;) {

    // Clear & display time

    lcd.clear();

    displayRTCDateTime();

    DateTime now = rtc.now();

    // Automatic dispensing checks

    for (int i = 0; i < 3; i++) {

      bool executeServo = false;

      if (frequencyChoices[i] == 1) {

        // Daily

        if (now.hour() == drugHours[i] && now.minute() == drugMinutes[i]) {

          executeServo = true;

        }

      }

      else if (frequencyChoices[i] == 2) {

        // Weekly

        if (now.dayOfTheWeek() == drugDays[i] &&

            now.hour()         == drugHours[i] &&

            now.minute()       == drugMinutes[i]) {

          executeServo = true;

        }

      }

      else if (frequencyChoices[i] == 3) {

        // Monthly

        if (now.day()    == drugDays[i] &&

            now.hour()   == drugHours[i] &&

            now.minute() == drugMinutes[i]) {

          executeServo = true;

        }

      }

      if (executeServo && !servoActionExecuted[i]) {

        dispenseDrug(i);

      }

    }

    // Reset servo flags at the start of each hour

    if (now.minute() == 0 && now.second() == 0) {

      for (int i = 0; i < 3; i++) {

        servoActionExecuted[i] = false;

      }

    }

    // Update once per second, or as you prefer

    vTaskDelay(1000 / portTICK\_PERIOD\_MS);

  }

}

/\*\*

 \* TaskMode1 (Time-Setting via Joystick):

 \*   Only runs when mode=1 is active (not suspended).

 \*/

void TaskMode1(void \*pvParams) {

  for (;;) {

    lcd.clear();

    lcd.setCursor(0, 0);

    lcd.print("Mode 1: Time");

    lcd.setCursor(0, 1);

    lcd.print("Setting...");

    vTaskDelay(1000 / portTICK\_PERIOD\_MS);

    // The blocking time-set function

    setTimeUsingJoystick();  // sets mode=0 when done

    // small pause

    vTaskDelay(500 / portTICK\_PERIOD\_MS);

  }

}

/\*\*

 \* TaskMode2 (Drug Scheduling):

 \*   Only runs when mode=2 is active.

 \*/

void TaskMode2(void \*pvParams) {

  for (;;) {

    lcd.clear();

    lcd.setCursor(0, 0);

    lcd.print("Mode 2: Sched");

    lcd.setCursor(0, 1);

    lcd.print("Drug Config...");

    vTaskDelay(1000 / portTICK\_PERIOD\_MS);

    // The blocking scheduling function

    Scheduling();  // sets mode=0 when done

    vTaskDelay(500 / portTICK\_PERIOD\_MS);

  }

}

Hardware.h:  
#ifndef HARDWARE\_H

#define HARDWARE\_H

#include <Arduino.h>

#include <Wire.h>

#include <LiquidCrystal\_I2C.h>

#include <RTClib.h>

#include <ESP32Servo.h>

// ------------------ Pin Definitions ------------------

extern const int JOY\_VERT\_PIN;

extern const int JOY\_HORZ\_PIN;

extern const int DRUG\_BUTTON;

extern const int BUTTON\_PIN\_1;

extern const int BUTTON\_PIN\_2;

extern const int BUTTON\_PIN\_3;

extern const int LCD\_SDA\_PIN;

extern const int LCD\_SCL\_PIN;

extern const int LCD\_ADDRESS;

extern const int SERVO\_PINS[3];

// ------------------ Hardware Objects ------------------

extern LiquidCrystal\_I2C lcd;   // The LCD

extern RTC\_DS3231 rtc;          // The RTC

extern Servo servos[3];

extern bool servoActionExecuted[3];

// ------------------ Drug Data ------------------

extern String drugNames[3];

extern int frequencyChoices[3];

extern int pillCounts[3];

extern int drugDays[3];

extern int drugHours[3];

extern int drugMinutes[3];

// For scheduling, etc.

extern int tempFrequency;

extern int tempDay;

extern int tempHour;

extern int tempMinute;

extern int tempPillCount;

extern int currentIndex;

extern int currentField;

extern String fieldNames[5];

// Shared mode variable

extern volatile int mode;

// ------------------ Functions ------------------

void hardwareInit();

void displayRTCDateTime();

void dispenseDrug(int index);

// If you want them here, you can also put the joystick & scheduling helpers here:

void handleJoystick(int xValue, int yValue);

void loadTempValues();

void saveDrugDetails();

void updateLCD2();

void changeDrugIndex();

void setTimeUsingJoystick();

void Scheduling();

hardware.cpp:  
#include "hardware.h"

// ------------------ Pin Definitions ------------------

const int JOY\_VERT\_PIN  = 34;

const int JOY\_HORZ\_PIN  = 35;

const int DRUG\_BUTTON   = 32;

const int BUTTON\_PIN\_1  = 2;

const int BUTTON\_PIN\_2  = 0;

const int BUTTON\_PIN\_3  = 4;

const int LCD\_SDA\_PIN   = 21;

const int LCD\_SCL\_PIN   = 22;

const int LCD\_ADDRESS   = 0x27;

const int SERVO\_PINS[3] = {18, 5, 17};

// ------------------ Hardware Objects ------------------

LiquidCrystal\_I2C lcd(LCD\_ADDRESS, 16, 2);

RTC\_DS3231 rtc;

Servo servos[3];

bool servoActionExecuted[3] = {false, false, false};

// ------------------ Drug Data ------------------

String drugNames[3]       = {"Drug1", "Drug2", "Drug3"};

int frequencyChoices[3]    = {0, 0, 0};

int pillCounts[3]          = {0, 0, 0};

int drugDays[3]            = {0, 0, 0};

int drugHours[3]           = {0, 0, 0};

int drugMinutes[3]         = {0, 0, 0};

int tempFrequency          = 1;

int tempDay                = 0;

int tempHour               = 0;

int tempMinute             = 0;

int tempPillCount          = 1;

int currentIndex           = 0;

int currentField           = 0;

String fieldNames[5]       = {"Freq", "Day", "Hour", "Min", "Pills"};

volatile int mode          = 0; // 0=normal,1=time-set,2=scheduling

// -------------------------------------------------------

//            Implementation of Hardware Functions

// -------------------------------------------------------

void hardwareInit()

{

  // Serial for debug

  Serial.begin(115200);

  // Initialize I2C for LCD & RTC

  Wire.begin(LCD\_SDA\_PIN, LCD\_SCL\_PIN);

  // LCD

  lcd.init();

  lcd.backlight();

  if (!rtc.begin()) {

    Serial.println("RTC initialization failed!");

    lcd.print("RTC Error");

    // could block here if needed

    while (1) { delay(1000); }

  }

  // Servos

  for (int i = 0; i < 3; i++) {

    servos[i].attach(SERVO\_PINS[i]);

    servos[i].write(90); // initial position

  }

  // Pins

  pinMode(JOY\_VERT\_PIN,  INPUT);

  pinMode(JOY\_HORZ\_PIN,  INPUT);

  pinMode(DRUG\_BUTTON,   INPUT\_PULLUP);

  pinMode(BUTTON\_PIN\_1,  INPUT\_PULLUP);

  pinMode(BUTTON\_PIN\_2,  INPUT\_PULLUP);

  pinMode(BUTTON\_PIN\_3,  INPUT\_PULLUP);

  Serial.println("hardwareInit: done.");

}

void displayRTCDateTime() {

  DateTime now = rtc.now();

  lcd.setCursor(0, 0);

  lcd.printf("%02d-%02d-%04d", now.day(), now.month(), now.year());

  lcd.setCursor(0, 1);

  lcd.printf("%02d:%02d:%02d", now.hour(), now.minute(), now.second());

}

void dispenseDrug(int index) {

  if (index < 0 || index >= 3) {

    Serial.println("Invalid drug index.");

    return;

  }

  lcd.clear();

  lcd.setCursor(0, 0);

  lcd.print("Dispensing drug:");

  lcd.setCursor(0, 1);

  lcd.print(drugNames[index]);

  vTaskDelay(2000 / portTICK\_PERIOD\_MS);

  for (int i = 0; i < pillCounts[index]; i++) {

    servos[index].write(0);

    vTaskDelay(500 / portTICK\_PERIOD\_MS);

    servos[index].write(90);

    vTaskDelay(500 / portTICK\_PERIOD\_MS);

  }

  servoActionExecuted[index] = true;

  lcd.clear();

}

// -------------------------------------------------------

//               Other "Helper" Functions

// -------------------------------------------------------

void handleJoystick(int xValue, int yValue) {

  // Horizontal for field navigation

  if (xValue == 4095) {

    currentField = (currentField - 1 + 5) % 5;

  } else if (xValue == 0) {

    currentField = (currentField + 1) % 5;

  }

  // Vertical for increment/decrement

  if (yValue == 0) {

    // Decrement

    switch (currentField) {

      case 0: tempFrequency = constrain(tempFrequency - 1, 1, 3); break;

      case 1: tempDay       = constrain(tempDay - 1, 0, 31);      break;

      case 2: tempHour      = constrain(tempHour - 1, 0, 23);     break;

      case 3: tempMinute    = constrain(tempMinute - 1, 0, 59);   break;

      case 4: tempPillCount = constrain(tempPillCount - 1, 1, 10);break;

    }

  } else if (yValue == 4095) {

    // Increment

    switch (currentField) {

      case 0: tempFrequency = constrain(tempFrequency + 1, 1, 3); break;

      case 1: tempDay       = constrain(tempDay + 1, 0, 31);      break;

      case 2: tempHour      = constrain(tempHour + 1, 0, 23);     break;

      case 3: tempMinute    = constrain(tempMinute + 1, 0, 59);   break;

      case 4: tempPillCount = constrain(tempPillCount + 1, 1, 10);break;

    }

  }

}

void loadTempValues() {

  tempFrequency = frequencyChoices[currentIndex];

  tempDay       = drugDays[currentIndex];

  tempHour      = drugHours[currentIndex];

  tempMinute    = drugMinutes[currentIndex];

  tempPillCount = pillCounts[currentIndex];

}

void saveDrugDetails() {

  frequencyChoices[currentIndex] = tempFrequency;

  drugDays[currentIndex]        = tempDay;

  drugHours[currentIndex]       = tempHour;

  drugMinutes[currentIndex]     = tempMinute;

  pillCounts[currentIndex]      = tempPillCount;

  Serial.print("Saved => Drug #");

  Serial.print(currentIndex + 1);

  Serial.print(": Freq=");

  Serial.print(tempFrequency);

  Serial.print(", Day=");

  Serial.print(tempDay);

  Serial.print(", Hour=");

  Serial.print(tempHour);

  Serial.print(", Min=");

  Serial.print(tempMinute);

  Serial.print(", Pills=");

  Serial.println(tempPillCount);

}

void updateLCD2() {

  lcd.clear();

  lcd.setCursor(0, 0);

  lcd.print("Drug ");

  lcd.print(currentIndex + 1);

  lcd.setCursor(0, 1);

  lcd.print(fieldNames[currentField]);

  lcd.print(": ");

  switch (currentField) {

    case 0: lcd.print(tempFrequency); break;

    case 1: lcd.print(tempDay);       break;

    case 2: lcd.print(tempHour);      break;

    case 3: lcd.print(tempMinute);    break;

    case 4: lcd.print(tempPillCount); break;

  }

}

void changeDrugIndex() {

  currentIndex = (currentIndex + 1) % 3;

  loadTempValues();

  Serial.print("Switched to Drug ");

  Serial.println(currentIndex + 1);

}

// The "blocking" time-set function

void setTimeUsingJoystick() {

  DateTime currentTime = rtc.now();

  int year   = currentTime.year();

  int month  = currentTime.month();

  int day    = currentTime.day();

  int hour   = currentTime.hour();

  int minute = currentTime.minute();

  int selectedField = 0;

  lcd.clear();

  lcd.setCursor(0, 0);

  lcd.print("Set Time:");

  while (digitalRead(BUTTON\_PIN\_2) == HIGH &&

         digitalRead(BUTTON\_PIN\_3) == HIGH &&

         digitalRead(BUTTON\_PIN\_1) == HIGH)

  {

    // Display date/time

    lcd.setCursor(0, 1);

    lcd.print("                "); // clear line

    lcd.setCursor(0, 1);

    lcd.printf("%04d/%02d/%02d %02d:%02d",

               year, month, day, hour, minute);

    // Indicate current field

    switch (selectedField) {

      case 0: lcd.setCursor(0, 1);  lcd.print(">>"); break; // Year

      case 1: lcd.setCursor(5, 1);  lcd.print(">>"); break; // Month

      case 2: lcd.setCursor(8, 1);  lcd.print(">>"); break; // Day

      case 3: lcd.setCursor(11, 1); lcd.print(">>"); break; // Hour

      case 4: lcd.setCursor(14, 1); lcd.print(">>"); break; // Minute

    }

    // Joystick

    int vertical   = analogRead(JOY\_VERT\_PIN);

    int horizontal = analogRead(JOY\_HORZ\_PIN);

    // Increment or decrement the selected field

    if (vertical == 4095) {

      if (selectedField == 0) year++;

      else if (selectedField == 1) month = (month % 12) + 1;

      else if (selectedField == 2) day   = (day % 31) + 1;

      else if (selectedField == 3) hour  = (hour + 1) % 24;

      else if (selectedField == 4) minute= (minute + 1) % 60;

    }

    else if (vertical == 0) {

      if (selectedField == 0) year--;

      else if (selectedField == 1) month  = (month == 1 ? 12 : month - 1);

      else if (selectedField == 2) day    = (day == 1 ? 31 : day - 1);

      else if (selectedField == 3) hour   = (hour - 1 + 24) % 24;

      else if (selectedField == 4) minute = (minute - 1 + 60) % 60;

    }

    // Move between fields

    if (horizontal == 4095) { // left

      selectedField = (selectedField - 1 + 5) % 5;

    } else if (horizontal == 0) { // right

      selectedField = (selectedField + 1) % 5;

    }

    vTaskDelay(500 / portTICK\_PERIOD\_MS);

  }

  // After editing

  if (digitalRead(BUTTON\_PIN\_2) == LOW) {

    // Save

    rtc.adjust(DateTime(year, month, day, hour, minute, 0));

    lcd.clear();

    lcd.setCursor(0, 0);

    lcd.print("Time Updated!");

    vTaskDelay(2000 / portTICK\_PERIOD\_MS);

    lcd.clear();

    mode = 0;

  }

  else if (digitalRead(BUTTON\_PIN\_3) == LOW) {

    // Cancel

    lcd.clear();

    lcd.setCursor(0, 0);

    lcd.print("Edit Cancelled");

    vTaskDelay(2000 / portTICK\_PERIOD\_MS);

    lcd.clear();

    mode = 0;

  }

}

void Scheduling() {

  // blocking loop until Button 2 or 3 is pressed

  while (digitalRead(BUTTON\_PIN\_2) == HIGH &&

         digitalRead(BUTTON\_PIN\_3) == HIGH)

  {

    int yValue = analogRead(JOY\_VERT\_PIN);

    int xValue = analogRead(JOY\_HORZ\_PIN);

    handleJoystick(xValue, yValue);

    if (digitalRead(DRUG\_BUTTON) == LOW) {

      changeDrugIndex();

      vTaskDelay(500 / portTICK\_PERIOD\_MS);

    }

    // Update the LCD

    updateLCD2();

    vTaskDelay(200 / portTICK\_PERIOD\_MS);

  }

  // If Button 2 is pressed, save

  if (digitalRead(BUTTON\_PIN\_2) == LOW) {

    saveDrugDetails();

  }

  // Exit scheduling

  mode = 0;

}

Normal code with blurtooth app:  
#include <Wire.h>

#include <LiquidCrystal\_I2C.h>

#include <RTClib.h>

#include <ESP32Servo.h>

#include <BluetoothSerial.h>

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 \*                Bluetooth Setup

 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

BluetoothSerial SerialBT;  // Create a Bluetooth Serial object

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 \*                Pin Definitions

 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// Joystick

#define JOY\_VERT\_PIN  34  // Vertical (Y-axis) - Analog pin

#define JOY\_HORZ\_PIN  35  // Horizontal (X-axis) - Analog pin

#define drugbutton    32  // Joystick pushbutton

// LCD (I2C)

#define LCD\_SDA\_PIN   21  // I2C Data pin (SDA)

#define LCD\_SCL\_PIN   22  // I2C Clock pin (SCL)

#define LCD\_ADDRESS   0x27

LiquidCrystal\_I2C lcd(LCD\_ADDRESS, 16, 2);

// RTC (I2C)

#define RTC\_SDA\_PIN   21  // I2C Data pin (SDA) - same as LCD in this setup

#define RTC\_SCL\_PIN   22  // I2C Clock pin (SCL) - same as LCD

#define RTC\_ADDRESS   0x68

RTC\_DS3231 rtc;            // Real-time clock object (DS3231)

// Servos

Servo servos[3];

const int servoPins[3] = {18, 5, 17};   // Which pins the 3 servos are attached to

bool servoActionExecuted[3] = {false, false, false};

// Pushbuttons

#define BUTTON\_PIN\_1  2  // Pushbutton 1

#define BUTTON\_PIN\_2  0  // Pushbutton 2

#define BUTTON\_PIN\_3  4  // Pushbutton 3

// LEDs for each servo

#define LED1\_PIN  25

#define LED2\_PIN  26

#define LED3\_PIN  27

const int ledPins[3] = {LED1\_PIN, LED2\_PIN, LED3\_PIN};

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 \*            Drug Scheduling Data

 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// Arrays to store drug information

String drugNames[3]        = {"Drug1", "Drug2", "Drug3"};

int frequencyChoices[3];    // 1=Daily, 2=Weekly, 3=Monthly

int pillCounts[3];

int drugDays[3];            // If Weekly or Monthly, which day

int drugHours[3];

int drugMinutes[3];

// Temporary local storage for fields (used in scheduling mode)

int tempFrequency  = 1;

int tempDay        = 0;

int tempHour       = 0;

int tempMinute     = 0;

int tempPillCount  = 1;

// Current selection in scheduling mode

int currentIndex   = 0;  // which drug (0..2)

int currentField   = 0;  // which field: Frequency, Day, Hour, Minute, Pills

String fieldNames[] = {"Freq", "Day", "Hour", "Min", "Pills"};

// Debounce

unsigned long lastButtonPress = 0;

// Mode variable (0=normal, 1=time-setting, 2=scheduling)

int mode = 0;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 \*                 Forward Declarations

 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void displayRTCDateTime();

void updateRTC();

void updateDrugDetails();

void dispenseDrug(int index);

void setTimeUsingJoystick();

void Scheduling();

void handleJoystick(int xValue, int yValue);

void changeDrugIndex();

void loadTempValues();

void saveDrugDetails();

void updateLCD2();

void incrementField();

void decrementField();

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 \*                  RTC Display

 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void displayRTCDateTime() {

  DateTime now = rtc.now();

  lcd.setCursor(0, 0);

  lcd.printf("%02d-%02d-%04d", now.day(), now.month(), now.year());

  lcd.setCursor(0, 1);

  lcd.printf("%02d:%02d:%02d", now.hour(), now.minute(), now.second());

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 \*              Updating the RTC via Bluetooth

 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void updateRTC() {

  lcd.clear();

  lcd.setCursor(0, 0);

  lcd.print("Edit Mode...");

  Serial.println("Enter new date/time via Bluetooth in format: yyyy,MM,dd,hh,mm,ss");

  Serial.println("Example: 2024,07,06,12,30,45");

  // Wait for input from Bluetooth

  while (!SerialBT.available()) {

    delay(100);

  }

  String input = SerialBT.readString();

  input.trim();

  // Parse the input

  int firstComma  = input.indexOf(',');

  int secondComma = input.indexOf(',', firstComma + 1);

  int thirdComma  = input.indexOf(',', secondComma + 1);

  int fourthComma = input.indexOf(',', thirdComma + 1);

  int fifthComma  = input.indexOf(',', fourthComma + 1);

  if (firstComma == -1 || secondComma == -1 || thirdComma == -1

      || fourthComma == -1 || fifthComma == -1) {

    Serial.println("Invalid input format.");

    return;

  }

  long newDate[6];

  newDate[0] = input.substring(0, firstComma).toInt();               // Year

  newDate[1] = input.substring(firstComma + 1, secondComma).toInt(); // Month

  newDate[2] = input.substring(secondComma + 1, thirdComma).toInt(); // Day

  newDate[3] = input.substring(thirdComma + 1, fourthComma).toInt(); // Hour

  newDate[4] = input.substring(fourthComma + 1, fifthComma).toInt(); // Minute

  newDate[5] = input.substring(fifthComma + 1).toInt();              // Second

  // Update the RTC

  rtc.adjust(DateTime(newDate[0], newDate[1], newDate[2],

                      newDate[3], newDate[4], newDate[5]));

  Serial.println("RTC Updated!");

  lcd.clear();

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 \*      Updating Drug Details via Bluetooth

 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void updateDrugDetails() {

  // Clear any old input

  while (SerialBT.available()) {

    SerialBT.read();

  }

  Serial.println("Enter details in format: d<index>,frequency,hour,minute[,day],pillCount");

  Serial.println("Example: d1,1,8,30,5");

  // Wait for input from Bluetooth

  while (!SerialBT.available()) {

    delay(100);

  }

  String input = SerialBT.readString();

  input.trim();

  // Split by commas

  int firstComma  = input.indexOf(',');

  int secondComma = input.indexOf(',', firstComma + 1);

  int thirdComma  = input.indexOf(',', secondComma + 1);

  int fourthComma = input.indexOf(',', thirdComma + 1);

  int fifthComma  = input.indexOf(',', fourthComma + 1);

  if (firstComma == -1 || secondComma == -1 || thirdComma == -1 || fourthComma == -1) {

    Serial.println("Invalid input format.");

    return;

  }

  // Extract drug index (d1 => index=0)

  int userIndex = input.substring(1, firstComma).toInt() - 1;

  if (userIndex < 0 || userIndex >= 3) {

    Serial.println("Invalid drug index.");

    return;

  }

  // Fill frequency, hour, minute

  frequencyChoices[userIndex] = input.substring(firstComma + 1, secondComma).toInt();

  drugHours[userIndex]       = input.substring(secondComma + 1, thirdComma).toInt();

  drugMinutes[userIndex]     = input.substring(thirdComma + 1, fourthComma).toInt();

  // If Weekly (2) or Monthly (3), parse day

  if (frequencyChoices[userIndex] == 2 || frequencyChoices[userIndex] == 3) {

    if (fourthComma == -1 || fifthComma == -1) {

      Serial.println("Invalid input format for weekly/monthly.");

      return;

    }

    drugDays[userIndex]   = input.substring(fourthComma + 1, fifthComma).toInt();

    pillCounts[userIndex] = input.substring(fifthComma + 1).toInt();

  }

  // If Daily (1), day is not used

  else if (frequencyChoices[userIndex] == 1) {

    if (fourthComma == -1) {

      Serial.println("Invalid input format for daily.");

      return;

    }

    drugDays[userIndex]   = 0; // Not used for daily

    pillCounts[userIndex] = input.substring(fourthComma + 1).toInt();

  }

  else {

    Serial.println("Invalid frequency choice.");

    return;

  }

  // Validate pill count range

  if (pillCounts[userIndex] <= 0 || pillCounts[userIndex] > 10) {

    Serial.println("Invalid pill count. Please enter a value between 1 and 10.");

    return;

  }

  // Print the updated details

  Serial.print("Updated Drug Details for index ");

  Serial.print(userIndex + 1);

  Serial.print(": Frequency = ");

  Serial.print(frequencyChoices[userIndex]);

  Serial.print(", Hour = ");

  Serial.print(drugHours[userIndex]);

  Serial.print(", Minute = ");

  Serial.print(drugMinutes[userIndex]);

  Serial.print(", Day = ");

  Serial.print(drugDays[userIndex]);

  Serial.print(", Pill Count = ");

  Serial.println(pillCounts[userIndex]);

  Serial.println("Drug details updated.");

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 \*               Dispense Drug Function

 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void dispenseDrug(int index) {

  if (index < 0 || index >= 3) {

    Serial.println("Invalid drug index.");

    return;

  }

  lcd.clear();

  lcd.setCursor(0, 0);

  lcd.print("Dispensing drug:");

  lcd.setCursor(0, 1);

  lcd.print(drugNames[index]);

  delay(2000);

  // Turn on the LED for this servo

  digitalWrite(ledPins[index], HIGH);

  // Rotate servo for pill count

  for (int i = 0; i < pillCounts[index]; i++) {

    servos[index].write(0);

    delay(500);

    servos[index].write(90);

    delay(500);

  }

  // Turn off the LED

  digitalWrite(ledPins[index], LOW);

  servoActionExecuted[index] = true; // Mark action as executed

  lcd.clear();

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 \*        Setting Time via Joystick/Buttons

 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void setTimeUsingJoystick() {

  // Retrieve the current RTC time

  DateTime currentTime = rtc.now();

  // Temporary variables for editing

  int year   = currentTime.year();

  int month  = currentTime.month();

  int day    = currentTime.day();

  int hour   = currentTime.hour();

  int minute = currentTime.minute();

  int selectedField = 0;  // 0=Year,1=Month,2=Day,3=Hour,4=Minute

  lcd.clear();

  lcd.setCursor(0, 0);

  lcd.print("Set Time:");

  // Loop until user presses save or cancel

  while (digitalRead(BUTTON\_PIN\_2) == HIGH &&

         digitalRead(BUTTON\_PIN\_3) == HIGH &&

         digitalRead(BUTTON\_PIN\_1) == HIGH)

  {

    // Clear display line 1

    lcd.setCursor(0, 1);

    lcd.print("                ");

    // Display date/time

    lcd.setCursor(0, 1);

    lcd.printf("%04d/%02d/%02d %02d:%02d", year, month, day, hour, minute);

    // Mark current field with ">>"

    switch (selectedField) {

      case 0: lcd.setCursor(0, 1);  lcd.print(">>"); break; // Year

      case 1: lcd.setCursor(5, 1);  lcd.print(">>"); break; // Month

      case 2: lcd.setCursor(8, 1);  lcd.print(">>"); break; // Day

      case 3: lcd.setCursor(11, 1); lcd.print(">>"); break; // Hour

      case 4: lcd.setCursor(14, 1); lcd.print(">>"); break; // Minute

    }

    // Read joystick

    int vertical   = analogRead(JOY\_VERT\_PIN);

    int horizontal = analogRead(JOY\_HORZ\_PIN);

    // Increment / decrement selected field

    if (vertical == 4095) {

      // Increase

      if (selectedField == 0) year++;

      else if (selectedField == 1) month = (month % 12) + 1;

      else if (selectedField == 2) day   = (day % 31) + 1;

      else if (selectedField == 3) hour  = (hour + 1) % 24;

      else if (selectedField == 4) minute= (minute + 1) % 60;

    }

    else if (vertical == 0) {

      // Decrease

      if (selectedField == 0) year--;

      else if (selectedField == 1) month = (month == 1 ? 12 : month - 1);

      else if (selectedField == 2) day   = (day == 1 ? 31 : day - 1);

      else if (selectedField == 3) hour  = (hour - 1 + 24) % 24;

      else if (selectedField == 4) minute= (minute - 1 + 60) % 60;

    }

    // Navigate fields with horizontal movement

    if (horizontal == 4095) { // left

      selectedField = (selectedField - 1 + 5) % 5;

    }

    else if (horizontal == 0) { // right

      selectedField = (selectedField + 1) % 5;

    }

    delay(500);

  }

  // After user breaks the loop:

  if (digitalRead(BUTTON\_PIN\_2) == LOW) {

    // Save pressed

    rtc.adjust(DateTime(year, month, day, hour, minute, 0));

    lcd.clear();

    lcd.setCursor(0, 0);

    lcd.print("Time Updated!");

    delay(2000);

    lcd.clear();

    mode = 0;

  }

  else if (digitalRead(BUTTON\_PIN\_3) == LOW) {

    // Cancel pressed

    lcd.clear();

    lcd.setCursor(0, 0);

    lcd.print("Edit Cancelled");

    delay(2000);

    lcd.clear();

    mode = 0;

  }

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 \*                    setup()

 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void setup() {

  // Initialize Serial for debug

  Serial.begin(115200);

  // Initialize Bluetooth Serial

  SerialBT.begin("ESP32\_PillDispenser");

  Serial.println("Bluetooth serial started.");

  // Initialize Servos

  for (int i = 0; i < 3; i++) {

    servos[i].attach(servoPins[i]);

    servos[i].write(90);  // initial position

  }

  Serial.println("Servos initialized.");

  // Initialize I2C (LCD, RTC)

  Wire.begin(LCD\_SDA\_PIN, LCD\_SCL\_PIN);

  // Initialize LCD

  lcd.init();

  lcd.backlight();

  Serial.println("LCD initialized.");

  // Initialize RTC

  if (!rtc.begin()) {

    Serial.println("RTC initialization failed!");

    lcd.print("RTC Error");

    while (1);

  }

  Serial.println("RTC initialized.");

  // Initialize joystick pins

  pinMode(JOY\_VERT\_PIN, INPUT);

  pinMode(JOY\_HORZ\_PIN, INPUT);

  pinMode(drugbutton,   INPUT\_PULLUP);

  Serial.println("Joystick pins set.");

  // Initialize pushbutton pins

  pinMode(BUTTON\_PIN\_1, INPUT\_PULLUP);

  pinMode(BUTTON\_PIN\_2, INPUT\_PULLUP);

  pinMode(BUTTON\_PIN\_3, INPUT\_PULLUP);

  Serial.println("Pushbutton pins set.");

  // Initialize LED pins

  for (int i = 0; i < 3; i++) {

    pinMode(ledPins[i], OUTPUT);

    digitalWrite(ledPins[i], LOW);  // start off

  }

  Serial.println("LED pins set.");

  // Initialize drug data arrays

  for (int i = 0; i < 3; i++) {

    frequencyChoices[i] = 0;

    pillCounts[i]       = 0;

    drugDays[i]         = 0;

    drugHours[i]        = 0;

    drugMinutes[i]      = 0;

  }

  Serial.println("Setup completed, all components initialized.");

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 \*                      loop()

 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void loop() {

  // Check Bluetooth for RTC or Drug detail updates

  if (SerialBT.available()) {

    char input = SerialBT.read();

    if (input == 'u') updateRTC();         // 'u' triggers RTC update

    if (input == 'd') updateDrugDetails(); // 'd' triggers drug detail update

  }

  // Check for button press to cycle through modes

  if (digitalRead(BUTTON\_PIN\_1) == LOW) {

    delay(200); // Debounce

    mode = (mode + 1) % 3;  // 0->1->2->0

  }

  // MODE LOGIC

  if (mode == 0) {

    // Normal (dispensing) mode

    lcd.setCursor(0, 0);

    displayRTCDateTime();

    DateTime rtcTime = rtc.now();

    // Check each drug schedule

    for (int i = 0; i < 3; i++) {

      bool executeServo = false;

      if (frequencyChoices[i] == 1) {

        // Daily

        if (rtcTime.hour() == drugHours[i] && rtcTime.minute() == drugMinutes[i]) {

          executeServo = true;

        }

      }

      else if (frequencyChoices[i] == 2) {

        // Weekly

        if (rtcTime.dayOfTheWeek() == drugDays[i] &&

            rtcTime.hour()         == drugHours[i] &&

            rtcTime.minute()       == drugMinutes[i]) {

          executeServo = true;

        }

      }

      else if (frequencyChoices[i] == 3) {

        // Monthly

        if (rtcTime.day()    == drugDays[i] &&

            rtcTime.hour()   == drugHours[i] &&

            rtcTime.minute() == drugMinutes[i]) {

          executeServo = true;

        }

      }

      if (executeServo && !servoActionExecuted[i]) {

        dispenseDrug(i);

      }

    }

    // Reset servoActionExecuted flags at the start of each hour

    if (rtcTime.minute() == 0 && rtcTime.second() == 0) {

      for (int i = 0; i < 3; i++) {

        servoActionExecuted[i] = false;

      }

    }

  }

  else if (mode == 1) {

    // Time-setting mode

    lcd.clear();

    lcd.setCursor(0, 0);

    lcd.print("Entering Edit");

    lcd.setCursor(0, 1);

    lcd.print("Time Mode...");

    lcd.clear();

    setTimeUsingJoystick();

    delay(1000);

  }

  else if (mode == 2) {

    // Scheduling mode

    lcd.clear();

    lcd.setCursor(0, 0);

    lcd.print("Scheduling Mode");

    lcd.clear();

    Scheduling();

    delay(1000);

  }

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 \*             Scheduling Mode Logic

 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void Scheduling() {

  // Wait until user presses BUTTON\_PIN\_2 or BUTTON\_PIN\_3 to exit

  while (digitalRead(BUTTON\_PIN\_2) == HIGH &&

         digitalRead(BUTTON\_PIN\_3) == HIGH)

  {

    int yValue = analogRead(JOY\_VERT\_PIN);

    int xValue = analogRead(JOY\_HORZ\_PIN);

    // Joystick to navigate fields

    handleJoystick(xValue, yValue);

    // If joystick button is pressed, move to next drug index

    if (digitalRead(drugbutton) == LOW) {

      changeDrugIndex();

      delay(500);

    }

    // Update LCD display

    updateLCD2();

    delay(200);

  }

  // If Button 2 is pressed, save

  if (digitalRead(BUTTON\_PIN\_2) == LOW) {

    saveDrugDetails();

  }

  // Return to mode 0

  mode = 0;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 \*        Joystick & Field Navigation Helpers

 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void changeDrugIndex() {

  // Cycle through drug 0..2

  currentIndex = (currentIndex + 1) % 3;

  loadTempValues();

  Serial.print("Switched to Drug ");

  Serial.println(currentIndex + 1);

}

void handleJoystick(int xValue, int yValue) {

  // Horizontal: change field

  if (xValue == 4095) {

    // Move to previous field

    currentField = (currentField - 1 + 5) % 5;

  }

  else if (xValue == 0) {

    // Move to next field

    currentField = (currentField + 1) % 5;

  }

  // Vertical: increment/decrement

  if (yValue == 0) {

    // Decrement

    decrementField();

  }

  else if (yValue == 4095) {

    // Increment

    incrementField();

  }

}

void incrementField() {

  switch (currentField) {

    case 0: tempFrequency = constrain(tempFrequency + 1, 1, 3); break;

    case 1: tempDay       = constrain(tempDay + 1,       0, 31); break;

    case 2: tempHour      = constrain(tempHour + 1,      0, 23); break;

    case 3: tempMinute    = constrain(tempMinute + 1,    0, 59); break;

    case 4: tempPillCount = constrain(tempPillCount + 1, 1, 10); break;

  }

}

void decrementField() {

  switch (currentField) {

    case 0: tempFrequency = constrain(tempFrequency - 1, 1, 3);  break;

    case 1: tempDay       = constrain(tempDay - 1,       0, 31); break;

    case 2: tempHour      = constrain(tempHour - 1,      0, 23); break;

    case 3: tempMinute    = constrain(tempMinute - 1,    0, 59); break;

    case 4: tempPillCount = constrain(tempPillCount - 1, 1, 10); break;

  }

}

void saveDrugDetails() {

  // Save local values into global arrays

  frequencyChoices[currentIndex] = tempFrequency;

  drugDays[currentIndex]        = tempDay;

  drugHours[currentIndex]       = tempHour;

  drugMinutes[currentIndex]     = tempMinute;

  pillCounts[currentIndex]      = tempPillCount;

  Serial.print("Saved => Drug ");

  Serial.print(currentIndex + 1);

  Serial.print(" | Freq: "); Serial.print(tempFrequency);

  Serial.print(", Day: ");    Serial.print(tempDay);

  Serial.print(", Hour: ");   Serial.print(tempHour);

  Serial.print(", Min: ");    Serial.print(tempMinute);

  Serial.print(", Pills: ");  Serial.println(tempPillCount);

}

void loadTempValues() {

  // Load global array values into local variables

  tempFrequency = frequencyChoices[currentIndex];

  tempDay       = drugDays[currentIndex];

  tempHour      = drugHours[currentIndex];

  tempMinute    = drugMinutes[currentIndex];

  tempPillCount = pillCounts[currentIndex];

}

void updateLCD2() {

  // Clear LCD and display current field

  lcd.clear();

  lcd.setCursor(0, 0);

  lcd.print("Drug ");

  lcd.print(currentIndex + 1);

  lcd.setCursor(0, 1);

  lcd.print(fieldNames[currentField]);

  lcd.print(": ");

  switch (currentField) {

    case 0: lcd.print(tempFrequency); break;

    case 1: lcd.print(tempDay);       break;

    case 2: lcd.print(tempHour);      break;

    case 3: lcd.print(tempMinute);    break;

    case 4: lcd.print(tempPillCount); break;

  }

}