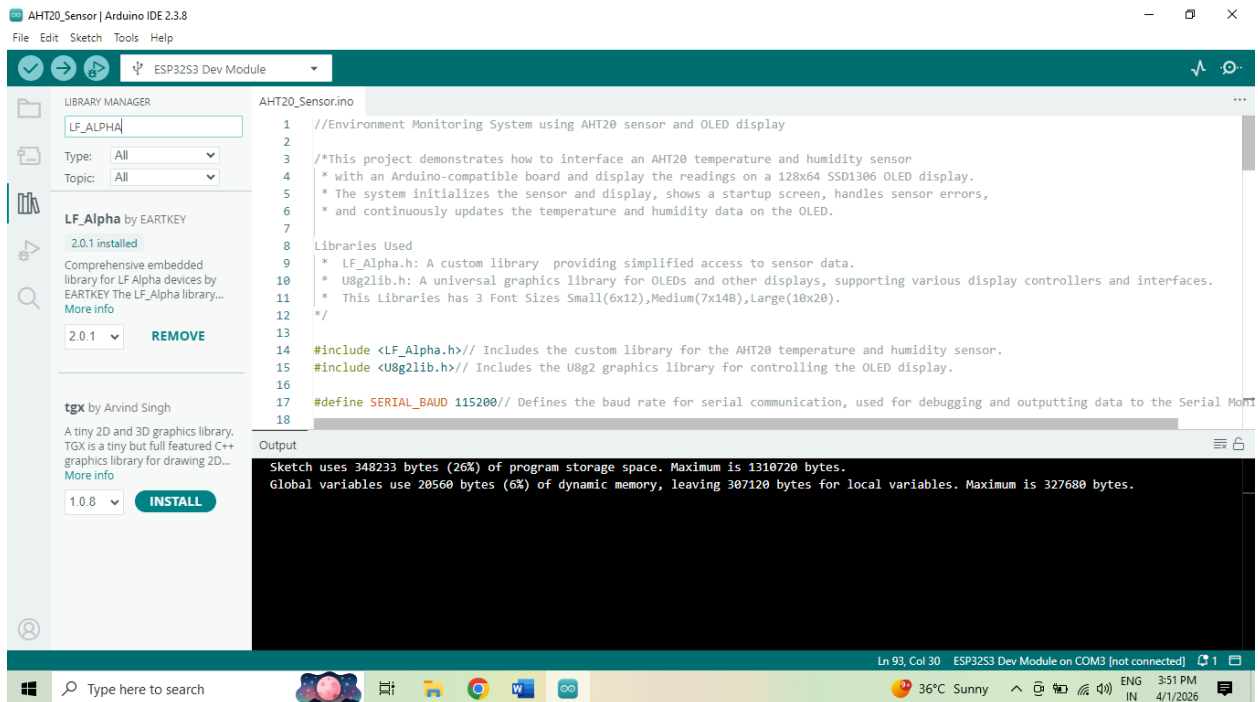


DESKTOP CLOCK

STEPS INVOLVED:

1. Install LF_ALPHA Library



- Install the latest version of LF_Alpha library.
- Select the board as ESP32S3 Dev Module.
- Select the appropriate port.

2. Complete Source Code

Platform: EARTKEY LF Alpha Development Kit (ESP32-S3 Mini-1)

Features: Real-time Clock, Pomodoro Timer, Audio-Visual Alerts

Library: LF_Alpha.h

```
#include <LF_Alpha.h>
```

```
#define SERIAL_MONITOR 115200
```

```
// ----- OBJECTS -----
```

```
// Instantiate the Real-Time Clock (RTC) object for timekeeping.
```

```

LF_Alpha_RTC RTC;
// Instantiate the OLED display object for visual output.
LF_Alpha_OLED OLED;
// Instantiate the Buttons object to handle user input from physical buttons.
LF_Alpha_Buttons buttons;
// Instantiate the Buzzer object for audible alerts.
LF_Alpha_Buzzer Buzzer;
// Instantiate the RGB LEDs object for visual status indicators.
LF_Alpha_rgbLEDs rgbLEDs;

// Flag to control the blinking state, typically used for visual indicators.
bool blinkState = true;
// Stores the last time a blink event occurred, used for timing intervals.
unsigned long lastBlinkTime = 0;
// Defines the current operating mode of the device:
// 0 = Clock mode (displays current time)
// 1 = Pomodoro mode (manages work/break sessions)
// Other modes (2, 3, 4) are used for settings adjustments within Pomodoro.
uint8_t mode = 0;

// ----- FLAGS (NEW) -----
// Flags to ensure mode-specific messages are printed only once when entering a mode.
bool clockMsgPrinted = false;
bool pomodoroMsgPrinted = false;
bool settingsMsgPrinted = false;

// ----- POMODORO -----
// Default work duration for Pomodoro, in seconds (25 minutes).
uint16_t workTime = 25 * 60;
// Default break duration for Pomodoro, in seconds (5 minutes).
uint16_t breakTime = 5 * 60;
// Remaining time in the current work or break session, in seconds.
uint16_t remainingTime = 25 * 60;

// Temporary variables for setting work and break times (in minutes) via user interface.
uint8_t tempWork = 25;
uint8_t tempBreak = 5;
// Flag indicating if the Pomodoro timer is currently running.
bool isRunning = false;
// Flag indicating if the current Pomodoro session is a work period (true) or a break period (false).
bool isWorkMode = true;

```

```

// Stores the last time the timer was updated, used for accurate time decrementing.
unsigned long lastUpdate = 0;

// Total duration of the entire Pomodoro session, in seconds (e.g., 1 hour).
uint32_t totalSession = 60 * 60;
// Remaining time for the entire Pomodoro session, in seconds.
uint32_t totalRemaining = totalSession;

// Temporary variable for setting total session duration (in hours) via user interface.
uint8_t tempSession = 1;

// Variables to store the previous state of the buttons, used for edge detection (press/release).
bool lastOk = HIGH;
bool lastUp = HIGH;
bool lastDown = HIGH;
bool lastBack = HIGH;

// ----- SETUP FUNCTION -----
// Initializes serial communication, OLED, and RTC module.
void setup() {
  // Initialize serial communication at 115200 baud rate for debugging.
  Serial.begin(SERIAL_MONITOR);
  Buzzer.begin();
  rgbLEDs.begin();
  // Initialize the OLED display.
  OLED.begin();
  // Display the welcome screen.
  showWelcome();

  // Inform user about RTC initialization.
  Serial.println("Initializing RTC...");
  // Attempt to initialize the RTC module.
  if (!RTC.begin()) {
    // If RTC initialization fails, display an error and halt execution.
    showError("RTC Init Failed");
    while (1)
      ;
  }
  // Confirm successful RTC initialization.
  Serial.println("RTC Initialized Successfully");
}

```

```

Serial.println("RTC Running Correctly");
Serial.println("System Ready ");

// Set initial remaining time for Pomodoro to work time.
remainingTime = workTime;
}

// ----- MAIN LOOP -----
// The main program loop, continuously handles button input, updates timer, and displays content.
void loop() {
  // Continuously check and handle button presses.
  handleButtons();

  // Update Pomodoro timer if it's running and one second has passed.
  if (isRunning && millis() - lastUpdate >= 1000) {
    lastUpdate = millis();

    // TOTAL SESSION CHECK FIRST
    if (totalRemaining <= 0) {
      isRunning = false;
      Serial.println("\n[POMODORO] SESSION FINISHED");

      playAlert(3); //session
      return;
    }
    totalRemaining--;
    // STOP if session ended
    if (!isRunning) return;

    // WORK / BREAK TIMER
    if (remainingTime <= 0) {
      if (isWorkMode) {
        playAlert(1);
      } else {
        playAlert(2);
      }
      isWorkMode = !isWorkMode;
      remainingTime = isWorkMode ? workTime : breakTime;
    }
    remainingTime--;
  }
}

```

```

// Display content based on the current operating mode.
if (mode == 0) showClock();
else if (mode == 1) showPomodoro();
else showSettings(); // For modes 2, 3, 4 (settings).
delay(100); // Small delay to prevent busy-waiting and reduce CPU usage.
}

// ----- WELCOME SCREEN -----
// Displays a welcome message on the serial monitor and OLED display upon system startup.
void showWelcome() {
    // Output welcome message to the serial console.
    Serial.println("\n===== SYSTEM STARTED =====");
    Serial.println("Displaying Welcome Screen...");
    // Clear the OLED display buffer.
    OLED.clearDisplay();
    // Set text color to white.
    OLED.setTextColor(SSD1306_WHITE);
    // Set text size to 2 (larger).
    OLED.setTextSize(2);
    // Set cursor position and print "DESKTOP".
    OLED.setCursor(24, 0);
    OLED.println("DESKTOP");
    // Set cursor position and print "CLOCK".
    OLED.setCursor(34, 30);
    OLED.println("CLOCK");
    // Push the display buffer to the OLED screen.
    OLED.display();
    // Pause for 5 seconds to show the welcome screen.
    delay(5000);
}

// ----- ERROR DISPLAY -----
// Displays an error message on the serial monitor and OLED display.
void showError(String msg) {
    // Output error message to the serial console.
    Serial.println("ERROR: " + msg);
    // Clear the OLED display buffer.
    OLED.clearDisplay();
    // Set text color to white.
    OLED.setTextColor(SSD1306_WHITE);
    // Set text size to 1 (smaller).

```

```

OLED.setTextSize(1);
// Set cursor position and print "ERROR:".
OLED.setCursor(10, 25);
OLED.println("ERROR:");
// Set cursor position and print the specific error message.
OLED.setCursor(10, 40);
OLED.println(msg);
// Push the display buffer to the OLED screen.
OLED.display();
}

// ----- CLOCK DISPLAY -----
// Displays the current time and date in digital clock format on the OLED.
void showClock() {
// Check if the clock mode message has not been printed yet.
if (!clockMsgPrinted) {
// Print mode entry messages to the serial console.
Serial.println("\n[MODE] DIGITAL CLOCK DISPLAYED");
Serial.println("→ Press OK button to switch to Pomodoro mode");
// Set flags to prevent re-printing messages for this mode and reset others.
clockMsgPrinted = true;
pomodoroMsgPrinted = false;
settingsMsgPrinted = false;
}
// Read current time and date from the RTC module.
String time = RTC.readTime();
String date = RTC.readDate();
// Extract hour and minute strings from the time string.
String hour = time.substring(0, 2);
String minute = time.substring(3, 5);
// Convert hour string to integer for 12-hour format conversion.
uint8_t h = hour.toInt();
String ampm = "AM";
// Determine AM/PM and convert to 12-hour format.
if (h >= 12) {
ampm = "PM";
if (h > 12) h -= 12;
}
// Handle midnight (00:xx becomes 12:xx AM).
if (h == 0) h = 12;
// Format hour string, adding leading zero if necessary.

```

```

String hourStr = (h < 10) ? "0" + String(h) : String(h);
// Combine formatted hour and minute.
String hourMin = hourStr + ":" + minute;

// Extract day, month, and year strings from the date string.
String d = date.substring(0, 2);
String m = date.substring(3, 5);
String y = date.substring(8);
// Clear the OLED display buffer.
OLED.clearDisplay();
// Set text size and cursor for the "DIGITAL CLOCK" header.
OLED.setTextSize(1);
OLED.setCursor(25, 5);
OLED.print("DIGITAL CLOCK");
// Implement colon blinking for the clock display.
if (millis() - lastBlinkTime >= 1000) {
  lastBlinkTime = millis();
  blinkState = !blinkState;
}
// If blinkState is false, replace the colon with a space to make it blink.
if (!blinkState) {
  hourMin.setCharAt(hourMin.indexOf(":"), ' ');
}
// Define parameters for drawing a rounded rectangle around the time.
uint8_t rx = 0, ry = 17, rw = 127, rh = 26, r = 6;
uint8_t gap = 3;
// Draw horizontal lines of the rounded rectangle.
for (uint8_t i = rx + r; i < rx + rw - r; i += gap) {
  OLED.drawPixel(i, ry, SSD1306_WHITE);
  OLED.drawPixel(i, ry + rh, SSD1306_WHITE);
}
// Draw vertical lines of the rounded rectangle.
for (uint8_t i = ry + r; i < ry + rh - r; i += gap) {
  OLED.drawPixel(rx, i, SSD1306_WHITE);
  OLED.drawPixel(rx + rw, i, SSD1306_WHITE);
}
// Draw rounded corners using trigonometric calculations.
for (uint8_t i = 0; i <= 90; i += 15) {
  float rad = i * PI / 180.0;
  int8_t dx = (int8_t)(r * cos(rad) + 0.5);
  int8_t dy = (int8_t)(r * sin(rad) + 0.5);
}

```

```

// Draw pixels for all four corners.
OLED.drawPixel(rx + r - dx, ry + r - dy, SSD1306_WHITE);
OLED.drawPixel(rx + rw - r + dx, ry + r - dy, SSD1306_WHITE);
OLED.drawPixel(rx + r - dx, ry + rh - r + dy, SSD1306_WHITE);
OLED.drawPixel(rx + rw - r + dx, ry + rh - r + dy, SSD1306_WHITE);
}
// Set text size and cursor for displaying the time (hour and minute).
OLED.setTextSize(2);
OLED.setCursor(17, 23);
OLED.print(hourMin);
// Set text size and cursor for displaying AM/PM indicator.
OLED.setTextSize(2);
OLED.setCursor(90, 23);
OLED.print(ampm);
// Set text size and cursor for displaying the date.
OLED.setTextSize(2);
OLED.setCursor(17, 50);
OLED.print(d + "/" + m + "/" + y);
// Push the display buffer to the OLED screen.
OLED.display();
}

// ----- POMODORO DISPLAY -----
// Displays the Pomodoro timer interface, showing remaining time and progress.
void showPomodoro() {
// Check if the Pomodoro mode message has not been printed yet.
if (!pomodoroMsgPrinted) {
// Print mode entry messages and instructions to the serial console.
Serial.println("\n[MODE] POMODORO DISPLAYED");
Serial.println("→ Press DOWN button to enter settings");
Serial.println("→ Press UP button to Pause/Resume timer");
Serial.println("→ Press BACK button to return to Digital Clock");

// Set flags to prevent re-printing messages for this mode and reset others.
pomodoroMsgPrinted = true;
clockMsgPrinted = false;
}
// Clear the OLED display buffer.
OLED.clearDisplay();
// Check if the Pomodoro session is completed.
if (!isRunning && totalRemaining == 0) {

```

```

Serial.println("\n[POMODORO] SESSION COMPLETED");
// Display "SESSION COMPLETED" message on the OLED.
OLED.setCursor(24, 20);
OLED.setTextSize(2);
OLED.setTextColor(SSD1306_WHITE);
OLED.println("SESSION");
OLED.setCursor(12, 45);
OLED.println("COMPLETED");
// Push the display buffer to the OLED screen.
OLED.display();
return; // Exit the function as session is complete.
}
// Calculate minutes and seconds from remaining time.
uint8_t minutes = remainingTime / 60;
uint8_t seconds = remainingTime % 60;
// Display "WORK TIME" or "BREAK TIME" based on current mode.
if (isWorkMode) {
    OLED.setTextSize(2);
    OLED.setCursor(13, 0);
    OLED.print("WORK TIME");
} else {
    OLED.setTextSize(2);
    OLED.setCursor(6, 0);
    OLED.print("BREAK TIME");
}
// Display the remaining time (MM:SS format).
OLED.setTextSize(2);
OLED.setCursor(30, 30);
if (minutes < 10) OLED.print("0");
OLED.print(minutes);
OLED.print(":");
if (seconds < 10) OLED.print("0");
OLED.print(seconds);
// Calculate progress for the progress bar.
uint16_t total = isWorkMode ? workTime : breakTime;
float progress = (float)(total - remainingTime) / total;
// Define progress bar dimensions.
uint8_t lineWidth = 100;
uint8_t filled = progress * lineWidth;
uint8_t startX = 14;
uint8_t y = 58;

```

```

// Draw the outline of the progress bar.
for (uint8_t i = 0; i < lineWidth; i += 4) {
    OLED.drawPixel(startX + i, y, SSD1306_WHITE);
}
// Fill the progress bar based on elapsed time.
for (uint8_t i = 0; i < filled; i++) {
    OLED.drawPixel(startX + i, y, SSD1306_WHITE);
}
// Push the display buffer to the OLED screen.
OLED.display();
}

// ----- SETTINGS DISPLAY -----
// Displays the settings interface for adjusting Pomodoro work, break, and total session times.
void showSettings() {
    // Check if settings messages have not been printed yet for the current setting mode.
    if (!settingsMsgPrinted) {
        // Print specific instructions based on the current setting mode.
        if (mode == 2) {
            Serial.println("\nSET WORK TIME");
            Serial.println("Press UP to increase");
            Serial.println("Press DOWN to decrease");
            Serial.println("Press OK after setting");
        } else if (mode == 3) {
            Serial.println("\nSET BREAK TIME");
            Serial.println("Press UP to increase");
            Serial.println("Press DOWN to decrease");
            Serial.println("Press OK after setting");
        } else if (mode == 4) {
            Serial.println("\nSET TOTAL SESSION");
            Serial.println("Press UP to increase");
            Serial.println("Press DOWN to decrease");
            Serial.println("Press OK after setting");
        }
        // Set flag to prevent re-printing messages for this setting mode.
        settingsMsgPrinted = true;
    }
    // Clear the OLED display buffer.
    OLED.clearDisplay();
    // Display the appropriate setting option and its current value.
    if (mode == 2) {

```

```

OLED.setTextSize(1);
OLED.setCursor(26, 4);
OLED.print("SET WORK TIME");
OLED.setTextSize(2);
OLED.setCursor(30, 30);
OLED.print(tempWork);
OLED.print(" min");
} else if (mode == 3) {
  OLED.setTextSize(1);
  OLED.setCursor(24, 4);
  OLED.print("SET BREAK TIME");
  OLED.setTextSize(2);
  OLED.setCursor(32, 32);
  OLED.print(tempBreak);
  OLED.print(" min");
} else if (mode == 4) {
  OLED.setTextSize(1);
  OLED.setCursor(14, 4);
  OLED.print("SET TOTAL SESSION");
  OLED.setTextSize(2);
  OLED.setCursor(36, 32);
  OLED.print(tempSession);
  OLED.print(" hr");
}
// Push the display buffer to the OLED screen.
OLED.display();
}

// ----- BUTTON HANDLER -----
// Manages button presses and triggers corresponding actions based on the current mode.
void handleButtons() {
  // Read the current state of each button.
  bool ok = buttons.read(okayButton);
  bool back = buttons.read(backButton);
  bool up = buttons.read(upButton);
  bool down = buttons.read(downButton);
  // Handle OK button press (falling edge detection).
  if (lastOk == HIGH && ok == LOW) {
    Serial.println("\n[BUTTON] OK PRESSED");
    // Actions based on current mode when OK is pressed.
    if (mode == 0) {

```

```

// From Clock mode, switch to Pomodoro mode.
Serial.println("→ Switching to Pomodoro");
mode = 1;
} else if (mode == 1) {
// In Pomodoro mode, restart the timer.
Serial.println("→ Restarting Pomodoro");
isWorkMode = true;
remainingTime = workTime;
totalRemaining = totalSession;
isRunning = true;
lastUpdate = millis();
} else if (mode == 2) {
// In Work Time setting, move to Break Time setting.
Serial.println("→ Work Time Set. Now set BREAK TIME");
mode = 3;
settingsMsgPrinted = false; // Reset flag to show new setting message.
} else if (mode == 3) {
// In Break Time setting, move to Total Session setting.
Serial.println("→ Break Time Set. Now set TOTAL SESSION");
mode = 4;
settingsMsgPrinted = false; // Reset flag to show new setting message.
} else if (mode == 4) {
// In Total Session setting, save all settings and start Pomodoro.
Serial.println("\n===== SETTINGS SAVED =====");
Serial.print("Work: ");
Serial.println(tempWork);
Serial.print("Break: ");
Serial.println(tempBreak);
Serial.print("Session: ");
Serial.println(tempSession);
// Apply temporary settings to actual Pomodoro variables.
workTime = tempWork * 60;
breakTime = tempBreak * 60;
totalSession = (uint32_t)tempSession * 3600; // Cast tempSession to uint32_t before multiplication
totalRemaining = totalSession;
isWorkMode = true;
remainingTime = workTime;
isRunning = true;
lastUpdate = millis();
Serial.println("→ Pomodoro Started");
mode = 1; // Return to Pomodoro display mode.

```

```

}

// Wait for the OK button to be released to prevent multiple triggers.
while (buttons.read(okayButton) == LOW)
    ; // WAIT RELEASE
}
// Update the last state of the OK button.
lastOk = ok;
// Handle UP button press (falling edge detection).
if (lastUp == HIGH && up == LOW) {
    Serial.println("[BUTTON] UP PRESSED");
    // Actions based on current mode when UP is pressed.
    if (mode == 1) isRunning = !isRunning; // Toggle Pomodoro timer (pause/resume).
    else if (mode == 2) tempWork++; // Increase work time in settings.
    else if (mode == 3) tempBreak++; // Increase break time in settings.
    else if (mode == 4) tempSession++; // Increase total session in settings.
    delay(200); // Debounce delay.
}
// Update the last state of the UP button.
lastUp = up;
// Handle DOWN button press (falling edge detection).
if (lastDown == HIGH && down == LOW) {
    Serial.println("[BUTTON] DOWN PRESSED");
    // Actions based on current mode when DOWN is pressed.
    if (mode == 1) {
        // From Pomodoro mode, enter Work Time setting mode.
        mode = 2;
        // Initialize temporary setting variables with current values.
        tempWork = workTime / 60;
        tempBreak = breakTime / 60;
        tempSession = totalSession / 3600;
        settingsMsgPrinted = false; // Reset flag to show new setting message.
    } else {
        // In setting modes, decrease the respective time, ensuring a minimum of 1.
        if (mode == 2) tempWork = max((uint8_t)1, (uint8_t)(tempWork - 1));
        else if (mode == 3) tempBreak = max((uint8_t)1, (uint8_t)(tempBreak - 1));
        else if (mode == 4) tempSession = max((uint8_t)1, (uint8_t)(tempSession - 1));
    }
    delay(200); // Debounce delay.
}
// Update the last state of the DOWN button.

```

```

lastDown = down;
// Handle BACK button press (falling edge detection).
if (lastBack == HIGH && back == LOW) {
  Serial.println("[BUTTON] BACK PRESSED");
  // Actions based on current mode when BACK is pressed.
  if (mode == 1) mode = 0; // From Pomodoro mode, switch back to Clock mode.
  delay(200); // Debounce delay.
}
// Update the last state of the BACK button.
lastBack = back;
}

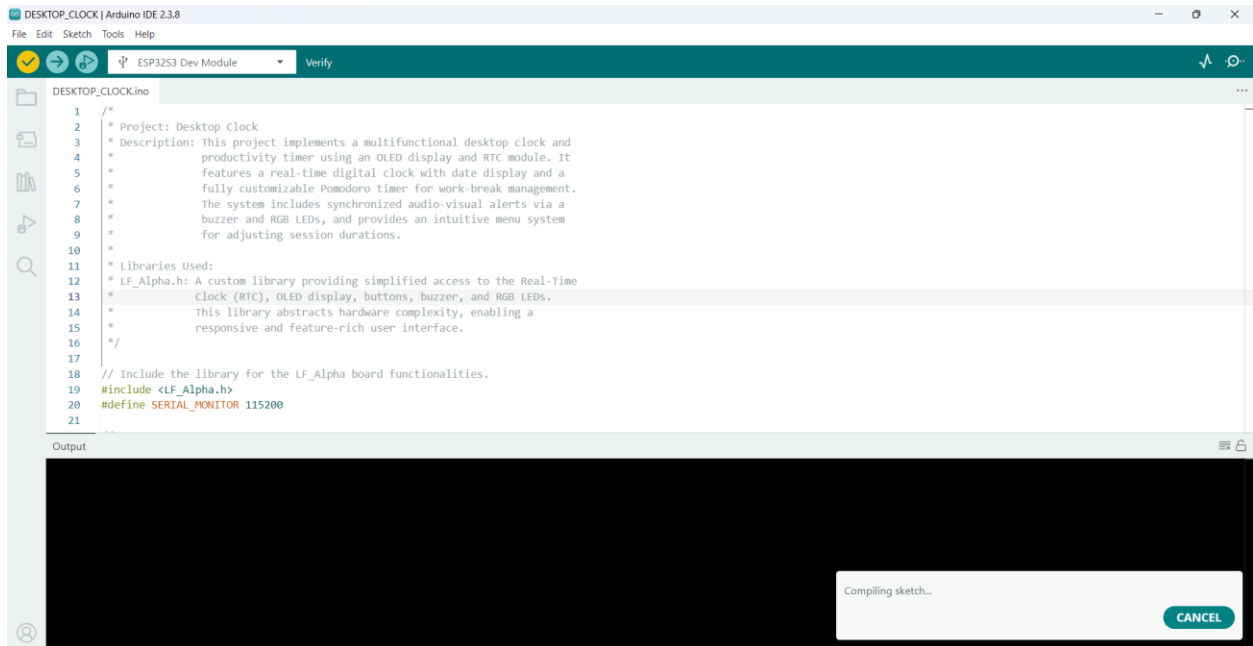
// ----- SYNC BUZZER + LED -----
void playAlert(uint8_t type) {
  /*
  type:
  1 → Work Done
  2 → Break Done
  3 → Session Complete
  */
  // WORK DONE → FAST RED + FAST BEEP
  if (type == 1) {
    Serial.println("[ALERT] Work Done");
    for (uint8_t i = 0; i < 3; i++) {
      // ON
      for (uint8_t j = 0; j < 5; j++) {
        rgbLEDs.write(j, 255, 0, 0);
      }
      rgbLEDs.show();
      Buzzer.write(HIGH);
      delay(150);
      // OFF
      rgbLEDs.clear();
      rgbLEDs.show();

      Buzzer.write(LOW);
      delay(150);
    }
  }
  // BREAK DONE → SLOW GREEN + SLOW BEEP
  else if (type == 2) {

```

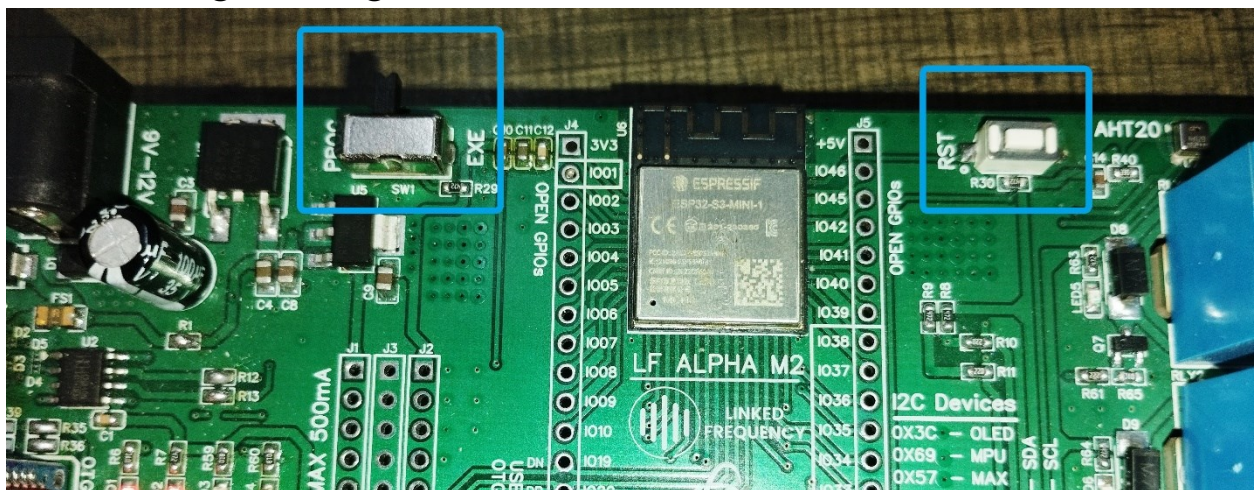
```
Serial.println("[ALERT] Break Done");
for (uint8_t i = 0; i < 2; i++) {
  // ON
  for (uint8_t j = 0; j < 5; j++) {
    rgbLEDs.write(j, 0, 255, 0);
  }
  rgbLEDs.show();
  Buzzer.write(HIGH);
  delay(400);
  // OFF
  rgbLEDs.clear();
  rgbLEDs.show();
  Buzzer.write(LOW);
  delay(400);
}
}
// SESSION COMPLETE → RAINBOW + LONG BEEP
else if (type == 3) {
  Serial.println("[ALERT] Session Complete");
  Buzzer.write(HIGH);
  for (uint8_t i = 0; i < 10; i++) {
    rgbLEDs.rainbow();
    rgbLEDs.show();
    delay(150);
  }
  Buzzer.write(LOW);
  rgbLEDs.clear();
  rgbLEDs.show();
}
}
```

3. Compile the Code

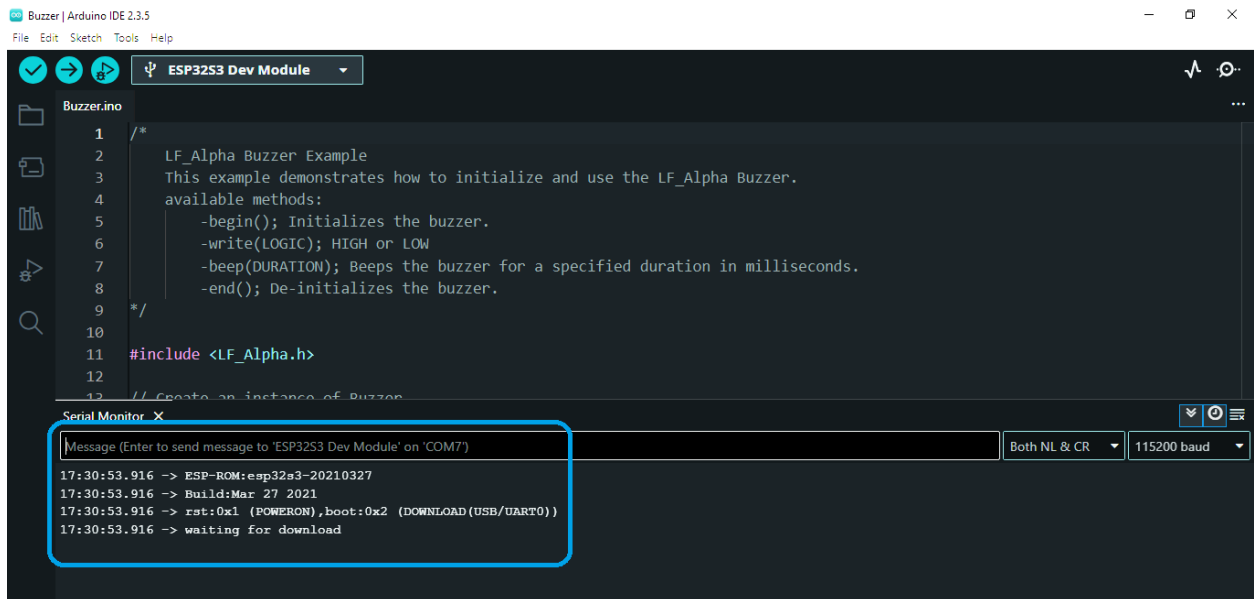


- The written program is compiled using the Arduino IDE to check for syntax errors and logical issues before uploading.
- During compilation, the IDE ensures that all required libraries (such as OLED libraries) are correctly installed and linked.
- Any compilation errors or warnings are analyzed and resolved, such as missing libraries, incorrect function usage, or hardware mismatches.

4. Enter Programming Mode

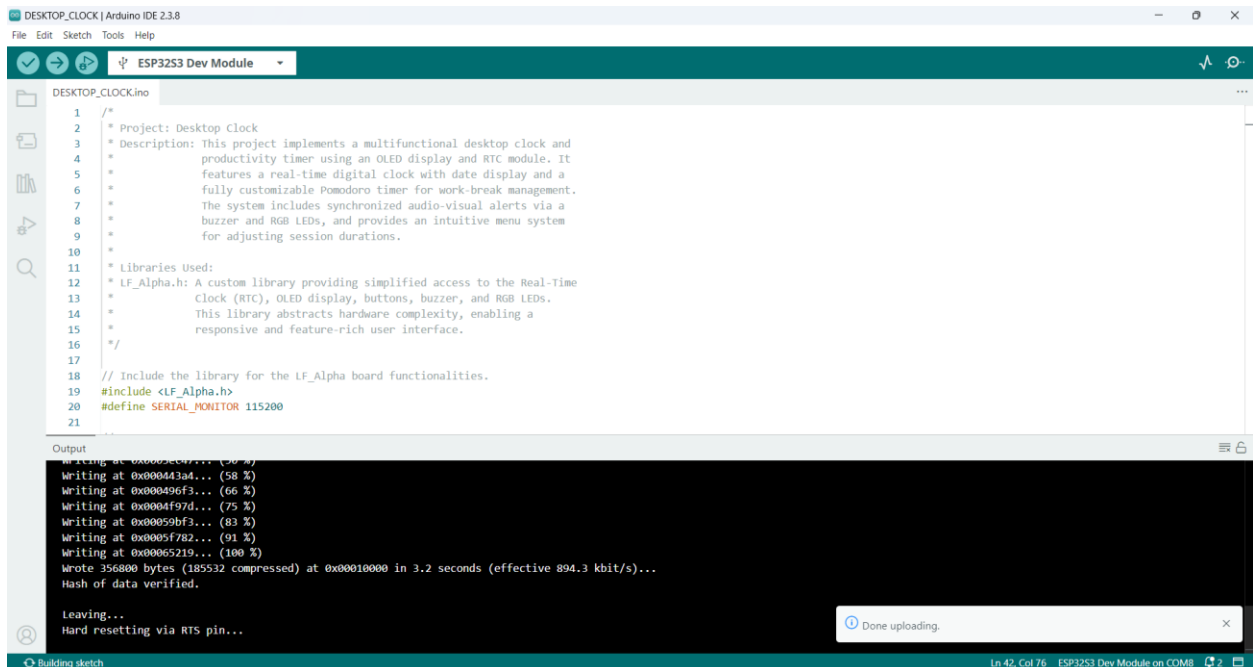


- Slide the mode switch to the **PROG** position, then press the **RESET** button. Observe the Serial Monitor for the "**waiting for download**" message.



- Select the correct baud rate (115200).

5. Upload the code.



- The code has been uploaded successfully.

6. Enter Execution Mode

- After the code is uploaded successfully, switch the mode slider to **EXE** (Execution) position and press the **RESET** button once again. Your code will now start running on the board.

7. Welcome Screen of OLED Display



- This is the welcome screen of the project “DESKTOP CLOCK”.

8. Output



- The output shows the Digital clock with high accuracy timekeeping station that reads data from the DS3231 RTC and displays current date and time on the SSD1306 OLED Screen.
- Operating Modes:
 1. 0 = Clock mode (displays current time)
 2. 1 = Pomodoro Clock (manage work/break sessions)

3. Other modes (2,3,4) are used for settings adjustment with Pomodoro.

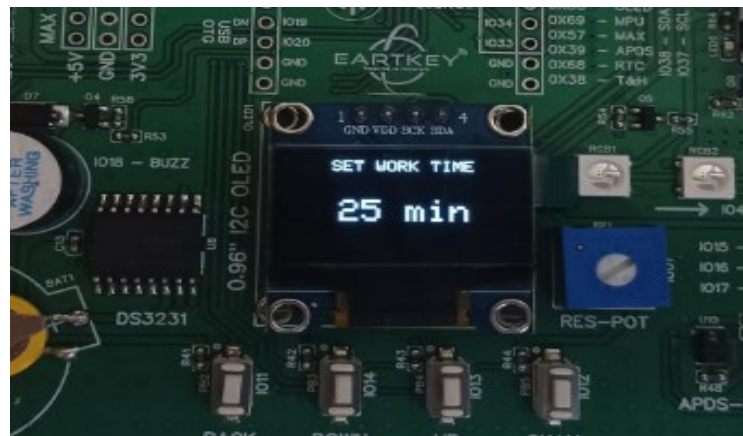
- Press OK Button to switch to Pomodoro Clock.

8.1 Pomodoro Clock



- A time management method using a timer to break work into 25-minute focused on intervals (pomodoros) followed by 5-minute breaks.
- Press DOWN Button to Set work time, break time and session time.

8.2 Settings



- Press OK Button after setting work time. UP Button for increment and DOWN Button for decrement the timer.



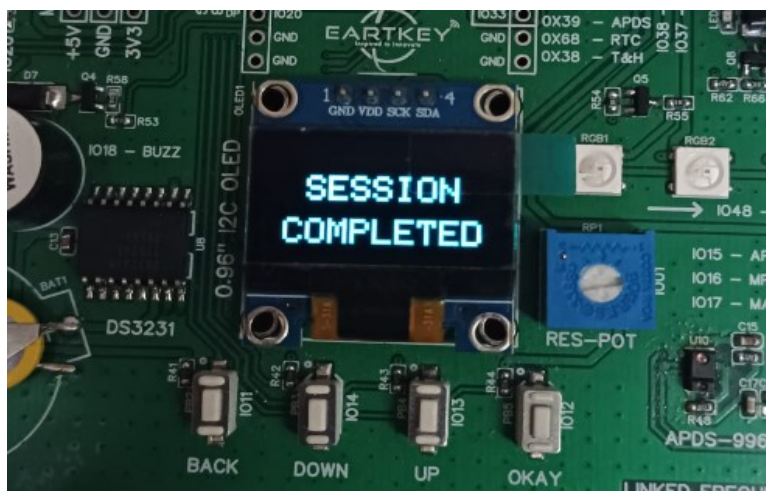
- Press OK Button after setting break time. UP Button for increment and DOWN Button for decrement the timer.



- Press OK Button after setting session time. UP Button for increment and DOWN Button for decrement the timer.

8.3 AUDIO-VISUAL ALERTS

Event	RGB LED	BUZZER
Work Session Complete	Fast Red Flash	Rapid Beeps
Break Session Complete	Slow Green Pulse	Steady Beeps
Total Session Finished	Rainbow Animation	Continuous Tone

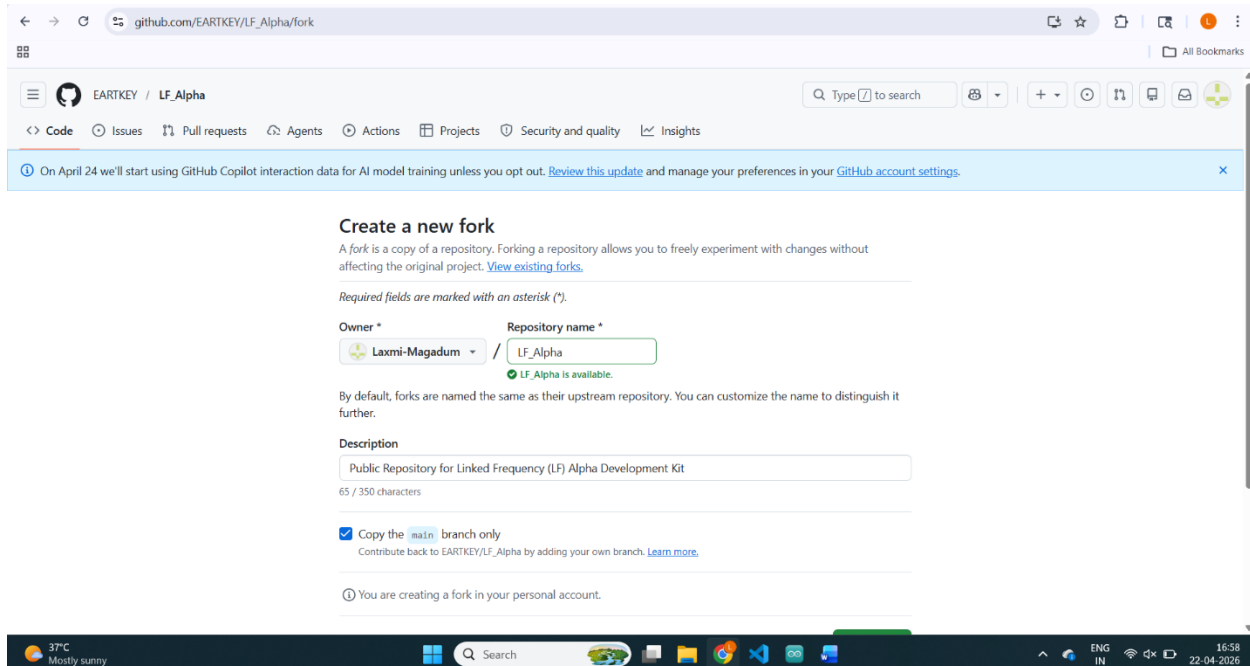


9. Github Workflow

- This section documents the complete Git and GitHub workflow used to fork the EARTKEY LF_Alpha repository, clone it locally, create a dedicated feature branch, and commit the project code. Following this workflow keeps the project history clean and makes it straightforward to open a pull request against the upstream repository if desired.

9.1 Fork the Repository

- Forking creates a personal copy of the upstream repository under your own GitHub account. All changes are made to your fork; the original EARTKEY repository is never modified directly.
1. Open a browser and navigate to github.com/EARTKEY/LF_Alpha.
 2. Click the Fork button in the upper-right corner of the page.
 3. On the fork dialog, confirm your GitHub account as the destination owner and click Create fork.



9.2 Clone the Fork Locally

- Cloning downloads the forked repository to the local machine and sets the remote origin to your fork. This is where all development work takes place.

```
# Cloning.
```

```
git clone https://github.com/EARTKEY/LF\_Alpha
```

9.3 Create a Branch

- Working on a dedicated branch keeps the main branch stable and makes the purpose of the change immediately visible from the branch name. The branch is created from the latest state of main.

```
# Create and switch to the feature branch
```

```
git checkout -b Desktop-clock
```

9.4 Add the Project Files

- Adds the files to git repository.

```
# Adding the files to git.
```

```
git add .
```

9.5 Commit

- Stage the new files and create a descriptive commit message. A well-written commit message explains what was added and why, not merely what files changed.

```
# Saves the files to git.
```

```
git commit -m "Desktop Clock".
```

9.6 Push the Branch to GitHub

- The git push command transfers the committed project files from the local system to a remote repository such as GitHub.

```
# Push the branch to git.
```

```
git push origin Desktop-clock
```

9.7 Open a Pull Request

1. Navigate to github.com/<your-username>/LF_Alpha in a browser.
2. GitHub will display a Compare & pull request banner for the recently pushed branch. Click it.
3. Set the base repository to EARTKEY/LF_Alpha and the base branch to main.
4. Write a concise title and description, then click Create pull request.