Names: Cosmin Narovici, Tommy Kang

Emails: cnaro2@uic.edu, dkang51@uic.edu, tommyk8477@gmail.com

NetID: cnaro2, dkang51

Project Name: The Wake Up-Inator

**Project Abstract:**

This project aims to create an "optimal" alarm clock that, in a set system, will use multiple Arduino units to manage external factors to cue a "wake up" in an optimal scenario. The resulting goal will strive to use REM cycles, room temperature, and light levels to provide a holistic wake-up experience that aims to remove the "morning grogginess and sluggishness." The system contains two interconnected Arduinos: one responsible for tracking time and temperature and phone signal inputs, the other managing hardware such as fans, heating, lighting, blinds, and the alarm, thus innovatively enhancing daily routines.

**Project Description:**

Sleep inertia, according to the CDC National Institute for Occupational Safety and Health, is defined as "a temporary disorientation and decline in performance and/or mood after awakening from sleep" (NIOSH Online). This can result in "slower reaction time, poorer short-term memory, and slower speed of thinking, reasoning, remembering, and learning" (NIOSH Online). The project is designed to be a responsive and personalized alarm clock that will serve as an "optimal" alarm clock by both accounting for, and adjusting several environmental variables. These include sunlight, temperature, and REM cycle timings. By creating a warmer room, paired with natural lighting from the sun, we can encourage and enhance the waking process while minimizing the lethargy, or sleep inertia, from waking up. Combining these sensory cues with an

alarm together creates an ideal scenario to start the day and reduce said sleep inertia, along with the timing of REM cycles (1.5 hour time periods).

The system's functionality, as a result, relies on multiple Arduinos, splitting the 2 up with dedicated tasks. One Arduino is needed to track input data, whether or not the alarm is "strict" or flexible," watching for temperature, time slept, alarm time (a set wake up time) and , while another handles outputs such as opening blinds, sounding the alarm, and adjusting the temperature in the room.

In using this multi-faceted approach, we can uniquely and efficiently handle multiple variables, while also automating the process in one set system. In doing so, we effectively solve a technical challenge while also meaningfully improve the user's quality of life.

**Final Project Design and Monitoring Arduinos:**

This project uses multiple Arduinos, each tasked with distinct workloads to make the wake up optimally efficient. By giving one Arduino a set of core responsibilities and responding exclusively to environmental cues, the Arduinos as a result are able to delegate tasks and operate cohesively.

Monitoring Arduino: The first Arduino represents the monitoring of the input data and tracking of user preferences. The data collected by this unit is time, temperature, sleep cycle, and the user's alarm choice. One of the key components is to enable a user to select between two modes: strict wake up which would set the alarm to the exact alarm time and flexible wake up which would set the alarm to a time as per REM cycles, which is subsequently calculated by this Arduino as well. Consequently, its task is to also take in environmental information through sensors and adjust temperatures accordingly based on the calculated foresighted optimal wake up time. It then sends signals to the second Arduino when the conditions are suitable.

Task Execution Arduino: The Monitoring Arduino simply sends a signal about which conditions have been met, and the second Arduino handles physical outputs. Sounding its alarm, opening the blinds, and gradually turning the heater on if the temperature requires it are all included. Data is sent by the Monitoring Arduino that then triggers a sequence of each action. There are timed, precisely layered physical actions to facilitate a budding experience of being woken up starting with the alarm sound and then followed by light and warming.

By stating these parts as specific roles, the Arduinos can communicate and operate in a system that works seamlessly with a number of wake up cues without the reliance or restriction of a singular Arduino. Additionally, the modular setup allows each function to be improved individually, making for easier troubleshooting as well as future expansion.

**Plan for Use and Communication Between the Multiple Arduinos:**

The alarm system's coordinated operation relies on Wi-Fi communication between the Monitoring Arduino and the Task Execution Arduino. This wireless setup enables seamless data exchange and task synchronization, ensuring each component responds precisely to environmental changes and user-defined settings.

- Wi-Fi Communication Protocol:

The two Arduinos communicate wirelessly through Wi-Fi modules, allowing data packets to be transmitted without a direct connection. Using Wi-Fi adds flexibility to the system setup and provides a reliable, high-speed communication pathway for coordinating tasks across devices. The Monitoring Arduino initiates communication by sending data packets containing specific trigger instructions, such as “wake-up time reached,” “temperature low,” or “open blinds.” Each message is tagged to ensure actions on the Task Execution Arduino occur in the correct order. This wireless communication eliminates the need for a physical connection, increasing flexibility for future expansions.

- Data Flow:

The Monitoring Arduino continuously tracks key variables like time, temperature, and sleep cycle progression. This information is compared against the user's preset wake-up criteria.

When conditions are met (e.g., reaching an optimal REM wake-up time or detecting a low room temperature), the Monitoring Arduino transmits a Wi-Fi signal to the Task Execution Arduino. This signal initiates specific waking actions—starting with the alarm, followed by opening the blinds, and, if necessary, turning on the heater to raise room temperature. The wireless connection allows the system to activate each waking cue smoothly and without physical dependencies, providing flexibility in placement and minimizing wiring complexity.

Error Handling:

Both Arduinos use an acknowledgment protocol to confirm successful message delivery. Upon receiving each trigger instruction, the Task Execution Arduino sends a confirmation back to the Monitoring Arduino. If a signal isn't acknowledged, the Monitoring Arduino automatically resends the message to ensure task execution.

As a failsafe, the system has a default setting to trigger the alarm if communication is lost or errors persist, ensuring the user still wakes up even in the event of connectivity issues.

This wireless communication setup enhances the user's wake-up experience by enabling precise coordination across all system components. The Wi-Fi connection between the Monitoring and Task Execution Arduinos ensures flexibility, smooth operation, and reliable data transmission, making this alarm system responsive to both user preferences and environmental factors.

**Arduino Communication and Project Design:**

This project will use multiple Arduinos as they will each have tasks delegated to them and then use the information they store/collect to trigger functions of the other. The tasks would be broken up into one Arduino being in charge of data collection from the environment it is placed in and the other for actually doing the physical tasks required based on the data that is received from the data collecting Arduino. In other words, we can have one Arduino set to read and manage 6 pieces of data. These inputs and information processed specifically are: whether or not the alarm is “strict” or flexible,” time slept, time passed, REM cycles achieved, and temperatures read. The other Arduino would then be in charge of the actual tasks of opening the blinds, sounding the alarm, and adjusting temperatures at both night and day. Thus, the second Arduino will need to to be paired and set up with its corresponding mechanical or electrical switches regulating heat and the fan, servo/stepper motors for the blinds and a physical alarm to sound off when the appropriate time has been achieved. These are then set relative to the room in a non-intrusive fashion to ensure ease of use, eliminating hindrances with the hardware components, and kept aesthetically pleasing for consumer usage.

As aforementioned, the input and data tracked are: whether or not the alarm is “strict” or flexible,” a time trigger that starts the alarm, a tracker for when a phone is turned off (to start the timer indicating the person is now sleeping), a tracker for how long the phone has been off, a sensor for room temp, and the output is the heater that raises the room temp. In regards to time slept, we can have the Arduino read a connection signal from a phone, through an external app, and when reading that the signal has “been lost” (phone turns off) we then can indicate that the user has started to sleep. We then calculate actual time slept based on the 1.5 hour REM cycles

and have these and indicators on when to start/stop/actuate our hardware. We also use temperature sensors to detect if the room is not cold enough to sleep, and if the room is not hot enough to ensure that the body is primed to wake up. Cleveland Clinic again indicates that anything around 70 degrees is too hot for sleep, and ideally “60-67 degrees Fahrenheit” is ideal (Cleveland Clinic Online). So the Arduino will communicate with the other Arduino to regulate these temps: 67 degrees when sleeping (via fan or online connected AC) and a gradual increase to 72 degrees starting from 30 minutes prior to waking up.

The Arduino in charge of the physical tasks would be in charge of actually raising the blinds utilizing a servo or stepper motor motor that would either tug on the string of the blinds to pull them up or by a mechanism to twist the shades open to let in more light based on the time of the other Arduino. Additionally, it would be in charge of sounding a very loud speaker which would play an alarm sound based on the time information received from the other Arduino such as the set time to wake up. Finally, the task of turning on the heater and fan will also be delegated to the physical Arduino which will be decided by the temperature reading of the other Arduino holding the data/constantly reading in the new information to be used.

**Original work:**

The way that our project attempts to incorporate original work is that it is more than just a system for tracking sleep and an alarm clock. Instead, it aims to build off of those two basic aspects through interacting with the physical environment which the person will be sleeping in, and tracking sleep based on REM cycles instead of just a set time. The aim is to be able to create an environment that is more suited to waking up easier and without sleep inertia and improving

the quality of sleep just by tracking the sleep through the REM cycles of a person. This would lead to improved recovery and an easier time waking up as an important aspect of sleep would not be interrupted.

**Timeline:**

Nov. 3-9:

- Begin Code

Nov. 10-16:

- Finish Code end of week, begin assembly

Nov. 17-23:

- Polish designs and code, get ready for presentation
- Design Presentation 11/22/2024

Nov. 24-30:

- Finish assembly of model and wire and run code

Dec. 1-6:

- Final testings
- Demonstration 12/06/2024

**List of Materials:**

-Fan

-Heater

-2 Wifi compatible Arduinos

-Wires

- 2 Motors (one Servo or Stepper one small 6v)

-Speaker/buzzer

-Lights (LED)

-Transistor

**References:**

Sleep Inertia:

https://valleysleepcenter.com/12-facts-about-sleep-inertia/#:~:text=Sleep%20inertia%20is%20the%20result,end%20of%20a%20sleep%20stage.

CDC Sleep Inertia:

https://www.cdc.gov/niosh/work-hour-training-for-nurses/longhours/mod7/03.html#:~:text=Sleep%20inertia%20is%20a%20temporary,reasoning%2C%20remembering%2C%20and%20learning.

REM Cycles:

https://my.clevelandclinic.org/health/body/12148-sleep-basics

Ideal Sleep Temp:

https://health.clevelandclinic.org/what-is-the-ideal-sleeping-temperature-for-my-bedroom

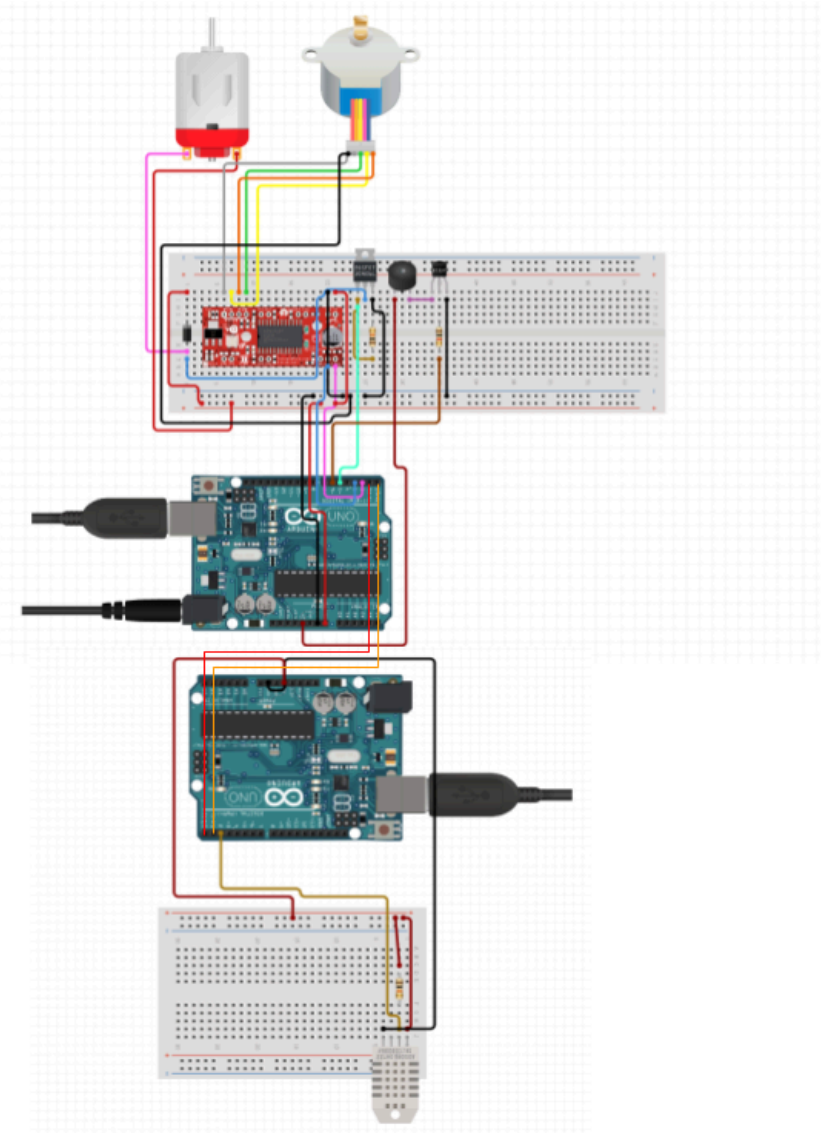
Effects of Heat:

https://pmc.ncbi.nlm.nih.gov/articles/PMC3427038/

Phone to Arduino Connection:

https://blog.arduino.cc/2023/03/08/get-more-out-of-your-smartphone-with-arduino/

**Diagram:**



**How to build:**

For the monitoring arduino, all that must be done is connecting up the humidity and temperature sensor to an input on the Arduino so that the data can be received and utilized for temperature regulation by the other Arduino. The output/hardware Arduino is more complex as there are 2 motors that need to be connected. The stepper motor will be connected to a stepper motor driver module to make things easier and that will then be connected to the Arduino. Additionally to the motor, a fan must be connected through using either wifi or connecting a second small 6V motor with a fan head on it. If taking the small motor approach, hooking it up to power and the Arduino is enough as it will be controlled based on the temperature information. Finally, some source of sound generation must be used like a buzzer which will be hooked up to the Arduino using a NPN transistor for proper power, ground, and connectivity to the Arduino.

**How to utilize:**

The way this project is meant to be utilized is by wifi connectivity to a phone to be able to control the functions that are available. When connected by phone, the user will be able to set if they want a strict alarm or a flexible alarm. This will mean that either the alarm will sound exactly at the given time while flexible means that it will push your alarm to the closest REM cycle that is going to be completed around the time you would want to wake up at. The stepper motor must also be hooked up to the string that would pull up the blinds so that when the alarm sounds, it is able to automatically pull the blinds up to let light into the room. Alternatively, it can be tied to just opening the blinds as well. Finally, setting the time that you would want to be woken up at. This concludes how to use the wakeup-inator.

**Code for input arduino:**

```
#include <ESP8266WiFi.h>
```

```
#include <ESP8266WebServer.h>
#include <Servo.h>

// Network credentials
const char* WIFI_SSID = "YourSSID";
const char* WIFI_PASSWORD = "YourPassword";

// Pin definitions
const int SERVO_PIN = D1;         // Servo for blinds control
const int BUZZER_PIN = D2;        // Buzzer for alarm
const int HEATER_PIN = D3;        // Relay control for heater
const int FAN_PIN = D4;           // Relay control for fan
const int STATUS_LED_PIN = D5;    // LED to indicate active connection

// Servo configuration
const int BLINDS_CLOSED_POS = 0;
const int BLINDS_OPEN_POS = 180;
const int SERVO_STEP = 5;         // Degrees per movement for smooth
operation

// Temperature thresholds (in Fahrenheit)
const float SLEEP_TEMP_MAX = 67.0;
const float SLEEP_TEMP_MIN = 60.0;
const float WAKE_TEMP_TARGET = 72.0;

// Buzzer configuration
const int BUZZER_FREQUENCY = 2000;  // Hz
```

```
const int ALARM_DURATION = 500;      // ms
const int ALARM_PAUSE = 500;         // ms

// State tracking
struct SystemState {
  bool blindsOpen;
  bool alarmActive;
  bool heaterOn;
  bool fanOn;
  float targetTemp;
  unsigned long lastBuzzerToggle;
} state;

Servo blindsServo;
ESP8266WebServer server(80);

void setup() {
  Serial.begin(115200);

  // Initialize pins
  pinMode(BUZZER_PIN, OUTPUT);
  pinMode(HEATER_PIN, OUTPUT);
  pinMode(FAN_PIN, OUTPUT);
  pinMode(STATUS_LED_PIN, OUTPUT);

  // Initialize servo
  blindsServo.attach(SERVO_PIN);
```

```
  blindsServo.write(BLINDS_CLOSED_POS);
  state.blindsOpen = false;

  // Initialize all systems to off
  digitalWrite(HEATER_PIN, LOW);
  digitalWrite(FAN_PIN, LOW);
  digitalWrite(BUZZER_PIN, LOW);
  state.heaterOn = false;
  state.fanOn = false;
  state.alarmActive = false;
  state.lastBuzzerToggle = 0;

  // Connect to WiFi
  WiFi.begin(WIFI_SSID, WIFI_PASSWORD);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    digitalWrite(STATUS_LED_PIN, !digitalRead(STATUS_LED_PIN));
  }
  digitalWrite(STATUS_LED_PIN, HIGH);

  // Setup web server endpoints
  server.on("/command", HTTP_POST, handleCommand);
  server.begin();

  Serial.println("Hardware Arduino Ready");
}
```

```
void loop() {
  server.handleClient();

  // Handle active alarm if needed
  if (state.alarmActive) {
    handleAlarm();
  }

  // Manage temperature control
  manageTemperature();
}

void handleCommand() {
  if (server.hasArg("plain")) {
    String message = server.arg("plain");
    int separatorIndex = message.indexOf('|');

    if (separatorIndex != -1) {
      int messageType = message.substring(0, separatorIndex).toInt();
      String payload = message.substring(separatorIndex + 1);

      processCommand(messageType, payload);
    }
  }
  server.send(200, "text/plain", "Command received");
}
```

```
void processCommand(int messageType, String payload) {
  switch (messageType) {
    case 0: // TEMP_ADJUST
      state.targetTemp = payload.toFloat();
      Serial.println("Temperature target updated to: " +
String(state.targetTemp));
      break;

    case 1: // WAKE_TRIGGER
      startWakeSequence();
      break;

    case 2: // BLINDS_CONTROL
      if (payload == "OPEN") {
        openBlinds();
      } else {
        closeBlinds();
      }
      break;
  }
}

void startWakeSequence() {
  state.alarmActive = true;
  state.targetTemp = WAKE_TEMP_TARGET;
  openBlinds();
  Serial.println("Wake sequence initiated");
```

```
}

void handleAlarm() {
  unsigned long currentMillis = millis();

  if (currentMillis - state.lastBuzzerToggle >= (state.alarmActive ?
ALARM_DURATION : ALARM_PAUSE)) {
    state.lastBuzzerToggle = currentMillis;

    if (digitalRead(BUZZER_PIN) == LOW) {
      tone(BUZZER_PIN, BUZZER_FREQUENCY);
    } else {
      noTone(BUZZER_PIN);
    }
  }
}

void openBlinds() {
  if (!state.blindsOpen) {
    // Gradually open blinds for smooth operation
    for (int pos = BLINDS_CLOSED_POS; pos <= BLINDS_OPEN_POS; pos +=
SERVO_STEP) {
      blindsServo.write(pos);
      delay(50);  // Small delay for smooth movement
    }
    state.blindsOpen = true;
    Serial.println("Blinds opened");
```

```
  }
}

void closeBlinds() {
  if (state.blindsOpen) {
    // Gradually close blinds for smooth operation
    for (int pos = BLINDS_OPEN_POS; pos >= BLINDS_CLOSED_POS; pos -=
SERVO_STEP) {
      blindsServo.write(pos);
      delay(50);  // Small delay for smooth movement
    }
    state.blindsOpen = false;
    Serial.println("Blinds closed");
  }
}

void manageTemperature() {
  if (state.targetTemp > 0) {  // Only manage if we have a target
    if (state.targetTemp > SLEEP_TEMP_MAX) {
      // Need to heat
      if (!state.heaterOn) {
        digitalWrite(HEATER_PIN, HIGH);
        digitalWrite(FAN_PIN, LOW);
        state.heaterOn = true;
        state.fanOn = false;
        Serial.println("Heater activated");
      }
```

```
    } else if (state.targetTemp < SLEEP_TEMP_MIN) {
      // Need to cool
      if (!state.fanOn) {
        digitalWrite(FAN_PIN, HIGH);
        digitalWrite(HEATER_PIN, LOW);
        state.fanOn = true;
        state.heaterOn = false;
        Serial.println("Fan activated");
      }
    } else {
      // Temperature in acceptable range
      digitalWrite(HEATER_PIN, LOW);
      digitalWrite(FAN_PIN, LOW);
      state.heaterOn = false;
      state.fanOn = false;
    }
  }
}
```

**Code for output arduino:**

```
#include <ESP8266WiFi.h>
#include <ESP8266WebServer.h>
#include <TimeLib.h>
#include <Wire.h>
#include <DHT.h>

// Network credentials
```

```
const char* WIFI_SSID = "YourSSID";
const char* WIFI_PASSWORD = "YourPassword";

// Task execution Arduino IP and port
const char* TASK_ARDUINO_IP = "192.168.1.101";
const int TASK_ARDUINO_PORT = 80;

// Pin definitions
const int TEMP_SENSOR_PIN = D3;
const int MODE_SWITCH_PIN = D2;  // Switch for strict/flexible mode
const int LED_INDICATOR_PIN = D4; // LED to indicate active monitoring

// DHT sensor setup
DHT dht(TEMP_SENSOR_PIN, DHT22);

// Temperature thresholds (in Fahrenheit)
const float SLEEP_TEMP_MAX = 67.0;
const float SLEEP_TEMP_MIN = 60.0;
const float WAKE_TEMP_TARGET = 72.0;

// Time constants (in milliseconds)
const unsigned long REM_CYCLE_DURATION = 5400000;  // 90 minutes
const unsigned long BUFFER_TIME = 1200000;         // 20 minutes
const unsigned long TEMP_CHECK_INTERVAL = 60000;   // 1 minute
const unsigned long WAKE_TEMP_PREP_TIME = 1800000; // 30 minutes
const unsigned long MESSAGE_TIMEOUT = 5000;        // 5 seconds
const unsigned long RETRY_DELAY = 1000;            // 1 second
```

```
// Message types
enum MessageType {
  TEMP_ADJUST,
  WAKE_TRIGGER,
  BLINDS_CONTROL,
  ERROR_STATE
};

// Sleep tracking structure
struct SleepSession {
  bool isActive;
  bool isStrictMode;
  time_t startTime;
  time_t targetWakeTime;
  time_t calculatedWakeTime;
  unsigned long phoneOffDuration;
  float currentTemp;
  bool tempControlActive;
} currentSession;

WiFiClient client;
ESP8266WebServer server(80);

// Function declarations
time_t calculateOptimalWakeUpTime(time_t startTime, time_t targetTime,
bool isStrict);
```

```
void adjustTemperature(float currentTemp);
void sendMessage(MessageType type, String payload);
void checkPhoneState();
void monitorTemperature();
void handleWakeUpSequence();

void setup() {
  Serial.begin(115200);

  // Initialize pins
  pinMode(MODE_SWITCH_PIN, INPUT_PULLUP);
  pinMode(LED_INDICATOR_PIN, OUTPUT);

  // Initialize DHT sensor
  dht.begin();

  // Connect to WiFi
  WiFi.begin(WIFI_SSID, WIFI_PASSWORD);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    digitalWrite(LED_INDICATOR_PIN, !digitalRead(LED_INDICATOR_PIN));
  }
  digitalWrite(LED_INDICATOR_PIN, HIGH);

  // Initialize sleep session
  currentSession.isActive = false;
  currentSession.tempControlActive = false;
```

```
  Serial.println("READY");
}

void loop() {
  server.handleClient();

  // Read current mode setting
  currentSession.isStrictMode = digitalRead(MODE_SWITCH_PIN) == HIGH;

  // Check for phone state updates
  checkPhoneState();

  // Monitor and control temperature
  monitorTemperature();

  // Handle wake-up sequence if needed
  if (currentSession.isActive) {
    handleWakeUpSequence();
  }
}

time_t calculateOptimalWakeUpTime(time_t startTime, time_t targetTime,
bool isStrict) {
  time_t totalSleepTime = targetTime - startTime - 600; // Subtract 10
minutes
```

```
  unsigned long maxREMCycles = totalSleepTime / (REM_CYCLE_DURATION /
1000);

  if (isStrict) {
    return startTime + (maxREMCycles * (REM_CYCLE_DURATION / 1000)) +
1800; // Add 30 minutes
  } else {
    unsigned long option1Cycles = maxREMCycles;
    unsigned long option2Cycles = maxREMCycles + 1;

    time_t option1Time = startTime + (option1Cycles * (REM_CYCLE_DURATION
/ 1000)) + (BUFFER_TIME / 1000);
    time_t option2Time = startTime + (option2Cycles * (REM_CYCLE_DURATION
/ 1000));

    unsigned long diff1 = abs(totalSleepTime - (option1Cycles *
(REM_CYCLE_DURATION / 1000) + (BUFFER_TIME / 1000)));
    unsigned long diff2 = abs(totalSleepTime - (option2Cycles *
(REM_CYCLE_DURATION / 1000)));

    return (diff1 <= diff2) ? option1Time : option2Time;
  }
}

void checkPhoneState() {
  if (Serial.available() > 0) {
    char phoneState = Serial.read();
```

```
    if (phoneState == '0' && !currentSession.isActive) {
      // Phone just turned off - start sleep session
      currentSession.isActive = true;
      currentSession.startTime = now();
      currentSession.calculatedWakeTime = calculateOptimalWakeUpTime(
        currentSession.startTime,
        currentSession.targetWakeTime,
        currentSession.isStrictMode
      );
      Serial.println("Sleep session started");
    }
    else if (phoneState == '1' && currentSession.isActive) {
      // Phone turned back on - end sleep session
      currentSession.isActive = false;
      Serial.println("Sleep session ended");
    }
  }
}

void monitorTemperature() {
  static unsigned long lastTempCheck = 0;

  if (millis() - lastTempCheck >= TEMP_CHECK_INTERVAL) {
    float temperature = dht.readTemperature(true); // Read in Fahrenheit

    if (!isnan(temperature)) {
```

```
      currentSession.currentTemp = temperature;

      if (currentSession.isActive) {
        // Check if we're approaching wake-up time
        time_t timeToWake = currentSession.calculatedWakeTime - now();

        if (timeToWake <= WAKE_TEMP_PREP_TIME / 1000) {
          // Start warming up the room
          if (temperature < WAKE_TEMP_TARGET) {
            sendMessage(TEMP_ADJUST, String(WAKE_TEMP_TARGET));
          }
        } else {
          // Maintain sleep temperature
          if (temperature > SLEEP_TEMP_MAX) {
            sendMessage(TEMP_ADJUST, String(SLEEP_TEMP_MAX));
          } else if (temperature < SLEEP_TEMP_MIN) {
            sendMessage(TEMP_ADJUST, String(SLEEP_TEMP_MIN));
          }
        }
      }
    }

    lastTempCheck = millis();
  }
}

void handleWakeUpSequence() {
```

```
  if (currentSession.isActive && now() >=
currentSession.calculatedWakeTime) {
    // Trigger wake-up sequence
    sendMessage(WAKE_TRIGGER, "START");
    delay(5000); // Wait 5 seconds
    sendMessage(BLINDS_CONTROL, "OPEN");

    // End sleep session
    currentSession.isActive = false;
    Serial.println("Wake-up sequence triggered");
  }
}

void sendMessage(MessageType type, String payload) {
  if (client.connect(TASK_ARDUINO_IP, TASK_ARDUINO_PORT)) {
    String message = String(type) + "|" + payload;
    client.println("POST /command HTTP/1.1");
    client.println("Host: " + String(TASK_ARDUINO_IP));
    client.println("Content-Type: text/plain");
    client.println("Content-Length: " + String(message.length()));
    client.println();
    client.println(message);

    if (!client.connected()) {
      client.stop();
    }
  }
```

```
}
```