CODE

```
#define BLYNK_TEMPLATE_ID "TMPL6fPEXdHt2"
#define BLYNK_TEMPLATE_NAME "Smart Room"
#define BLYNK_AUTH_TOKEN "GWPZDgIENCWprIU_yL32-
bPvkvKysbJj"

#include <WiFi.h>
#include <BlynkSimpleEsp32.h>
#include <DHT.h>

// Pin definitions
const int stepperPins[4] = {13, 12, 14, 27}; //
Motor 1: IN1, IN2, IN3, IN4
const int sensor1Pin = 2;  // Outside sensor (entry
first)
const int sensor2Pin = 4;  // Inside sensor (entry
second)
const int lightPin = 15;   // Light control pin
#define DHT_PIN 5      // DHT11 data pin
#define FAN_IN1 19     // L298N IN3
#define FAN_IN2 21     // L298N IN4
#define FAN_ENA 18     // L298N ENA (connected to
ENB)

const int cycles = 1200; // Number of cycles per
button press

// Wi-Fi credentials
char ssid[] = "OnePlus 12";
char pass[] = "xxxxxxxx"; // This is not the actual
password.

// Blynk virtual pins
#define LEFT_PIN V5   // Press left to rotate right
side (counterclockwise)
```

```
#define RIGHT_PIN V6  // Press right to rotate left
side (clockwise)
#define LIGHT_PIN V1  // Light control
#define FAN_PIN V2    // Fan ON/OFF
#define FAN_SPEED_PIN V3 // Fan speed (0-255) -
manual control
#define AUTO_MODE_PIN V4 // Auto fan mode toggle
#define COUNT_PIN V7   // People count display
#define TEMP_PIN V8    // *NEW: Temperature display*

// DHT11 setup
DHT dht(DHT_PIN, DHT11);

// Occupancy variables
volatile int peopleCount = 0;
enum State { IDLE, ENTRY_SENSOR1, ENTRY_SENSOR2,
EXIT_SENSOR2, EXIT_SENSOR1 };
volatile State currentState = IDLE;
volatile unsigned long lastTriggerTime = 0;
const unsigned long sequenceWindow = 2000; //
Increased to 2 seconds for multiple entries
const int debounceDelay = 10;              // 10ms
debounce
volatile bool sensor1Triggered = false;
volatile bool sensor2Triggered = false;
volatile int fanSpeed = 0; // Manual fan speed
control (0-255)
volatile bool autoFanMode = true; // Default to auto
mode

// OPTIMIZATION: Timing variables for reduced CPU
and Blynk usage
unsigned long lastTempRead = 0;
unsigned long lastSensorPrint = 0;
unsigned long lastBlynkUpdate = 0;
const unsigned long tempReadInterval = 3000;    //
Read temperature every 3 seconds
const unsigned long sensorPrintInterval = 5000; //
Print sensor status every 5 seconds
```

```
const unsigned long blynkUpdateInterval = 5000; //
Update Blynk every 5 seconds maximum
float lastTemperature = 0.0; // Store last valid
temperature
float lastBlynkTemperature = -999.0; // Store last
temperature sent to Blynk
const float tempChangeThreshold = 0.5; // Only
update Blynk if temperature changes by 0.5°C or more

// OPTIMIZATION: Batch Blynk updates to reduce
messages
bool needsBlynkUpdate = false;
bool tempNeedsUpdate = false;
bool fanNeedsUpdate = false;
bool lightNeedsUpdate = false;
bool countNeedsUpdate = false;

// Stepper variables
volatile bool leftAllowed = true;  // Initially
allow left
volatile bool rightAllowed = false; // Initially
disable right
volatile int stepIndex = 0;
volatile unsigned long previousStepTime = 0;
const int stepDelay = 1; // 1ms step delay
volatile bool isStepping = false;
volatile int stepsRemaining = 0;

// Step sequences
const int clockwiseSeq[8][4] = {
  {1, 0, 0, 0},
  {1, 1, 0, 0},
  {0, 1, 0, 0},
  {0, 1, 1, 0},
  {0, 0, 1, 0},
  {0, 0, 1, 1},
  {0, 0, 0, 1},
  {1, 0, 0, 1}
};
const int counterclockwiseSeq[8][4] = {
```

```
  {1, 0, 0, 1},
  {0, 0, 0, 1},
  {0, 0, 1, 1},
  {0, 0, 1, 0},
  {0, 1, 1, 0},
  {0, 1, 0, 0},
  {1, 1, 0, 0},
  {1, 0, 0, 0}
};

// Task handles
TaskHandle_t stepperTaskHandle = NULL;
TaskHandle_t occupancyTaskHandle = NULL;

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

  // Stepper setup
  for (int i = 0; i < 4; i++) {
    pinMode(stepperPins[i], OUTPUT);
    digitalWrite(stepperPins[i], LOW);
  }

  // Occupancy setup
  pinMode(sensor1Pin, INPUT);
  pinMode(sensor2Pin, INPUT);
  pinMode(lightPin, OUTPUT);
  pinMode(FAN_IN1, OUTPUT);
  pinMode(FAN_IN2, OUTPUT);
  pinMode(FAN_ENA, OUTPUT); // Controls both ENA and
ENB if wired together
  digitalWrite(lightPin, LOW);
  stopFan();
  dht.begin();
  delay(2000); // Sensor stabilization

  // Create tasks
  xTaskCreate(stepperTask, "StepperTask", 2048,
NULL, 1, &stepperTaskHandle); // Core 0
```

```
  xTaskCreate(occupancyTask, "OccupancyTask", 4096,
NULL, 1, &occupancyTaskHandle); // Core 1
  delay(500); // Allow tasks to start

  // Connect to Blynk on the main core
  Blynk.begin(BLYNK_AUTH_TOKEN, ssid, pass);
  Serial.println("Combined System Started");

  // *NEW: Initial Blynk sync*
  syncInitialValues();
}

void loop() {
  Blynk.run(); // Handle Blynk on the main core

  // OPTIMIZATION: Batch update Blynk to reduce
messages
  unsigned long currentTime = millis();
  if (needsBlynkUpdate && (currentTime -
lastBlynkUpdate >= blynkUpdateInterval)) {
    performBatchBlynkUpdate();
    lastBlynkUpdate = currentTime;
    needsBlynkUpdate = false;
  }
}

// NEW: Sync initial values to Blynk
void syncInitialValues() {
  Blynk.virtualWrite(LIGHT_PIN,
digitalRead(lightPin));
  Blynk.virtualWrite(FAN_PIN, (fanSpeed > 0) ? HIGH
: LOW);
  Blynk.virtualWrite(FAN_SPEED_PIN, fanSpeed);
  Blynk.virtualWrite(COUNT_PIN, peopleCount);
  Blynk.virtualWrite(AUTO_MODE_PIN, autoFanMode);
  if (lastTemperature > 0) {
    Blynk.virtualWrite(TEMP_PIN, lastTemperature);
    lastBlynkTemperature = lastTemperature;
  }
  Serial.println("Initial values synced to Blynk");
```

```
}

// NEW: Batch Blynk updates to reduce server load
void performBatchBlynkUpdate() {
  if (tempNeedsUpdate) {
    Blynk.virtualWrite(TEMP_PIN, lastTemperature);
    lastBlynkTemperature = lastTemperature;
    tempNeedsUpdate = false;
    Serial.print("Temperature updated to Blynk: ");
Serial.println(lastTemperature);
  }

  if (fanNeedsUpdate) {
    Blynk.virtualWrite(FAN_PIN, (fanSpeed > 0) ?
HIGH : LOW);
    Blynk.virtualWrite(FAN_SPEED_PIN, fanSpeed);
    fanNeedsUpdate = false;
  }

  if (lightNeedsUpdate) {
    Blynk.virtualWrite(LIGHT_PIN,
digitalRead(lightPin));
    lightNeedsUpdate = false;
  }

  if (countNeedsUpdate) {
    Blynk.virtualWrite(COUNT_PIN, peopleCount);
    countNeedsUpdate = false;
  }
}

// Stepper task (unchanged)
void stepperTask(void *pvParameters) {
  for (;;) {
    if (isStepping) {
      unsigned long currentTime = millis();
      if (currentTime - previousStepTime >=
stepDelay) {
        if (stepsRemaining > 0) {
```

```
          const int* seq = leftAllowed ?
counterclockwiseSeq[stepIndex] :
clockwiseSeq[stepIndex];
          for (int i = 0; i < 4; i++) {
            digitalWrite(stepperPins[i], seq[i]);
          }
          stepIndex = (stepIndex + 1) % 8;
          stepsRemaining--;
          previousStepTime = currentTime;
        } else {
          isStepping = false;
          delay(10); // Stabilize
        }
      }
    }
    delay(1); // Yield
  }
  vTaskDelete(NULL);
}

// OPTIMIZED: Occupancy task with reduced Blynk
updates
void occupancyTask(void *pvParameters) {
  for (;;) {
    int sensor1 = digitalRead(sensor1Pin);
    int sensor2 = digitalRead(sensor2Pin);
    unsigned long currentTime = millis();

    // Print sensor status less frequently
    if (currentTime - lastSensorPrint >=
sensorPrintInterval) {
      Serial.print("Sensor 1: ");
Serial.print(sensor1);
      Serial.print(", Sensor 2: ");
Serial.println(sensor2);
      Serial.print("People Count: ");
Serial.println(peopleCount);
      lastSensorPrint = currentTime;
    }
```

```
    switch (currentState) {
      case IDLE:
        if (sensor1 == LOW && !sensor1Triggered) {
          delay(debounceDelay);
          if (digitalRead(sensor1Pin) == LOW) {
            sensor1Triggered = true;
            lastTriggerTime = currentTime;
            currentState = ENTRY_SENSOR1;
            Serial.println("Sensor 1 triggered
(entry start), state: ENTRY_SENSOR1");
          }
        } else if (sensor2 == LOW &&
!sensor2Triggered) {
          delay(debounceDelay);
          if (digitalRead(sensor2Pin) == LOW) {
            sensor2Triggered = true;
            lastTriggerTime = currentTime;
            currentState = EXIT_SENSOR2;
            Serial.println("Sensor 2 triggered (exit
start), state: EXIT_SENSOR2");
          }
        }
        break;

      case ENTRY_SENSOR1:
        if (sensor2 == LOW && (currentTime -
lastTriggerTime <= sequenceWindow)) {
          delay(debounceDelay);
          if (digitalRead(sensor2Pin) == LOW) {
            peopleCount++;
            updateLightsAndFan();
            currentState = IDLE;
            sensor1Triggered = false;
            Serial.print("Person entered. Count: ");
Serial.println(peopleCount);
          }
        } else if (currentTime - lastTriggerTime >
sequenceWindow) {
          currentState = IDLE;
          sensor1Triggered = false;
```

```
          Serial.println("Entry sequence timed out,
state: IDLE");
        }
        break;

      case EXIT_SENSOR2:
        if (sensor1 == LOW && (currentTime -
lastTriggerTime <= sequenceWindow)) {
          delay(debounceDelay);
          if (digitalRead(sensor1Pin) == LOW) {
            if (peopleCount > 0) peopleCount--;
            updateLightsAndFan();
            currentState = IDLE;
            sensor2Triggered = false;
            Serial.print("Person exited. Count: ");
Serial.println(peopleCount);
          }
        } else if (currentTime - lastTriggerTime >
sequenceWindow) {
          currentState = IDLE;
          sensor2Triggered = false;
          Serial.println("Exit sequence timed out,
state: IDLE");
        }
        break;
    }

    // OPTIMIZED: Temperature reading with smart
Blynk updates
    if (currentTime - lastTempRead >=
tempReadInterval) {
      float temp = dht.readTemperature();
      if (!isnan(temp)) {
        lastTemperature = temp;
        Serial.print("Temperature: ");
Serial.print(temp); Serial.println(" °C");

        // OPTIMIZATION: Only update Blynk if
temperature changed significantly
```

```
        if (abs(temp - lastBlynkTemperature) >=
tempChangeThreshold) {
          tempNeedsUpdate = true;
          needsBlynkUpdate = true;
        }

        if (peopleCount >= 1 && autoFanMode) { //
Auto mode logic
          if (temp >= 33) {
            setFanSpeed(255);
            Serial.println("Temperature >= 33°C, fan
set to 255");
          } else if (temp >= 30) {
            setFanSpeed(180);
            Serial.println("Temperature >= 30°C, fan
set to 180");
          } else if (temp > 25) {
            setFanSpeed(128);
            Serial.println("Temperature > 25°C, fan
set to 128");
          } else {
            stopFan();
            Serial.println("Temperature <= 25°C, fan
stopped");
          }
        }
      } else {
        Serial.println("Failed to read DHT11 - using
last valid reading");
      }
      lastTempRead = currentTime;
    }

    delay(50);
  }
  vTaskDelete(NULL);
}

// OPTIMIZATION: Update lights and fan with batched
Blynk updates
```

```
void updateLightsAndFan() {
  if (peopleCount > 0) {
    digitalWrite(lightPin, HIGH);
  } else {
    digitalWrite(lightPin, LOW);
    stopFan(); // Turn off fan and light when no one
is present
  }

  // OPTIMIZATION: Mark for batched update instead
of immediate update
  lightNeedsUpdate = true;
  fanNeedsUpdate = true;
  countNeedsUpdate = true;
  needsBlynkUpdate = true;
}

// Fan control
void setFanSpeed(int speed) {
  digitalWrite(FAN_IN1, HIGH);
  digitalWrite(FAN_IN2, LOW);
  analogWrite(FAN_ENA, speed);
  fanSpeed = speed;

  // Optimization: Mark for batched update
  fanNeedsUpdate = true;
  needsBlynkUpdate = true;

  Serial.print("Fan speed set to: ");
Serial.println(speed);
}

void stopFan() {
  digitalWrite(FAN_IN1, LOW);
  digitalWrite(FAN_IN2, LOW);
  analogWrite(FAN_ENA, 0);
  fanSpeed = 0;

  // Optimization: Mark for batched update
  fanNeedsUpdate = true;
```

```
  needsBlynkUpdate = true;

  Serial.println("Fan stopped");
}

// OPTIMIZED: Blynk handlers with reduced updates
BLYNK_WRITE(LEFT_PIN) {
  int value = param.asInt();
  if (value == 1 && leftAllowed && !isStepping) { //
Check !isStepping to prevent interruption
    Serial.println("Rotating to right side
(counterclockwise) for " + String(cycles) + "
cycles");
    isStepping = true;
    stepsRemaining = cycles * 8;
    stepIndex = 0;
    leftAllowed = false;
    rightAllowed = true;
  } else if (value == 1 && isStepping) {
    Serial.println("Open operation in progress -
ignoring close request");
  }
}

BLYNK_WRITE(RIGHT_PIN) {
  int value = param.asInt();
  if (value == 1 && rightAllowed && !isStepping) {
// Check !isStepping to prevent interruption
    Serial.println("Rotating to left side
(clockwise) for " + String(cycles) + " cycles");
    isStepping = true;
    stepsRemaining = cycles * 8;
    stepIndex = 0;
    rightAllowed = false;
    leftAllowed = true;
  } else if (value == 1 && isStepping) {
    Serial.println("Close operation in progress -
ignoring open request");
  }
}
```

```
BLYNK_WRITE(LIGHT_PIN) {
  int value = param.asInt();
  if (value == HIGH) {
    digitalWrite(lightPin, HIGH);
    Serial.println("Lights ON (Manual)");
  } else if (value == LOW) {
    digitalWrite(lightPin, LOW);
    Serial.println("Lights OFF (Manual)");
  }
  // OPTIMIZATION: Immediate update for manual
control
  Blynk.virtualWrite(LIGHT_PIN,
digitalRead(lightPin));
}

BLYNK_WRITE(FAN_PIN) {
  int value = param.asInt();
  if (value == HIGH && !autoFanMode) {
    fanSpeed = 128;
    setFanSpeed(fanSpeed);
    Serial.println("Fan ON (Manual, Half Speed)");
  } else if (value == LOW && !autoFanMode) {
    fanSpeed = 0;
    stopFan();
    Serial.println("Fan OFF (Manual)");
  }
  // OPTIMIZATION: Immediate update for manual
control
  Blynk.virtualWrite(FAN_PIN, (fanSpeed > 0) ? HIGH
: LOW);
}

BLYNK_WRITE(FAN_SPEED_PIN) {
  int value = param.asInt();
  if (value >= 0 && value <= 255 && !autoFanMode) {
    fanSpeed = value;
    setFanSpeed(fanSpeed);
    Serial.print("Fan Speed set to: ");
Serial.println(fanSpeed);
```

```
  }
  // OPTIMIZATION: Immediate update for manual
control
  Blynk.virtualWrite(FAN_SPEED_PIN, fanSpeed);
}

BLYNK_WRITE(AUTO_MODE_PIN) {
  autoFanMode = param.asInt();
  Serial.print("Auto Fan Mode: ");
Serial.println(autoFanMode ? "Enabled" :
"Disabled");
  if (autoFanMode) {
    // Use last valid temperature reading
    if (lastTemperature > 0 && peopleCount >= 1) {
      if (lastTemperature >= 33) setFanSpeed(255);
      else if (lastTemperature >= 30)
setFanSpeed(180);
      else if (lastTemperature > 25)
setFanSpeed(128);
      else stopFan();
    } else {
      stopFan();
    }
  } else {
    // OPTIMIZATION: Immediate sync for mode change
    Blynk.virtualWrite(FAN_SPEED_PIN, fanSpeed);
  }
}
```

We used the Grok AI tool for assistance

https://grok.com/