CODE

```
#define BLYNK_TEMPLATE_ID "TMPL6fPEXdHt2"
#define BLYNK_TEMPLATE_NAME "LED ON OFF"
#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 = 500; // Number of cycles per button press
```

```
// Wi-Fi credentials
char ssid[] = "OnePlus 12";
char pass[] = "XXXXXX";  //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)
#define AUTO_MODE_PIN V4 // Auto fan mode toggle

// 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 = 1000; // 1
second window
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

// 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");
}

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

// Stepper task
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);
}

// Occupancy task
void occupancyTask(void *pvParameters) {
  for (;;) {
    int sensor1 = digitalRead(sensor1Pin);
    int sensor2 = digitalRead(sensor2Pin);
    unsigned long currentTime = millis();
  if (currentTime % 1000 == 0) {
   Serial.print("Sensor 1: ");
Serial.print(sensor1);
   Serial.print(", Sensor 2: ");
Serial.println(sensor2);
  }

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

    float temp = dht.readTemperature();
    if (!isnan(temp)) {
      Serial.print("Temperature: ");
Serial.print(temp); Serial.println(" °C");
      if (peopleCount > 0 && autoFanMode && fanSpeed
== 0) { // Auto only if no manual speed and auto mode
enabled
        if (temp > 30) setFanSpeed(255); // Max speed
        else if (temp > 25) setFanSpeed(128); // Half
speed
        else setFanSpeed(0); // Off
      }
    } else {
      Serial.println("Failed to read DHT11");
      stopFan();
    }

    delay(10); // Yield
  }
  vTaskDelete(NULL);
}
```

```
// Update lights and fan
void updateLightsAndFan() {
  if (peopleCount > 0) {
    digitalWrite(lightPin, HIGH);
    if (autoFanMode && fanSpeed == 0) setFanSpeed(128); // Default half speed if auto and no manual
  } else {
    digitalWrite(lightPin, LOW);
    stopFan();
  }
  Blynk.virtualWrite(LIGHT_PIN, digitalRead(lightPin));
  Blynk.virtualWrite(FAN_PIN, (fanSpeed > 0) ? HIGH : LOW);
  Blynk.virtualWrite(FAN_SPEED_PIN, fanSpeed);
}

// Fan control
void setFanSpeed(int speed) {
  digitalWrite(FAN_IN1, HIGH);
  digitalWrite(FAN_IN2, LOW);
  analogWrite(FAN_ENA, speed);
  fanSpeed = speed; // Update global fanSpeed for sync
  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;
  Serial.println("Fan stopped");
}

// Blynk handlers
BLYNK_WRITE(LEFT_PIN) {
  int value = param.asInt();
  if (value == 1 && leftAllowed) {
    Serial.println("Rotating to right side
(counterclockwise) for " + String(cycles) + "
cycles");
    isStepping = true;
    stepsRemaining = cycles * 8; // 8 steps per cycle
    stepIndex = 0;
    leftAllowed = false;
    rightAllowed = true;
  }
}

BLYNK_WRITE(RIGHT_PIN) {
  int value = param.asInt();
  if (value == 1 && rightAllowed) {
    Serial.println("Rotating to left side (clockwise)
for " + String(cycles) + " cycles");
    isStepping = true;
    stepsRemaining = cycles * 8; // 8 steps per cycle
    stepIndex = 0;
    rightAllowed = false;
    leftAllowed = true;
  }
}

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)");
  }
  Blynk.virtualWrite(LIGHT_PIN,
digitalRead(lightPin));
}

BLYNK_WRITE(FAN_PIN) {
  int value = param.asInt();
  if (value == HIGH) {
    fanSpeed = 128; // Default half speed on manual
ON
    setFanSpeed(fanSpeed);
    Serial.println("Fan ON (Manual, Half Speed)");
  } else if (value == LOW) {
    fanSpeed = 0;
    stopFan();
    Serial.println("Fan OFF (Manual)");
  }
  Blynk.virtualWrite(FAN_PIN, (fanSpeed > 0) ? HIGH : LOW);
}

BLYNK_WRITE(FAN_SPEED_PIN) {
  int value = param.asInt();
  if (value >= 0 && value <= 255) {
    fanSpeed = value; // Manual speed takes priority
    setFanSpeed(fanSpeed);
    Serial.print("Fan Speed set to: "); Serial.println(fanSpeed);
  }
  Blynk.virtualWrite(FAN_SPEED_PIN, fanSpeed);
}

BLYNK_WRITE(AUTO_MODE_PIN) {
  autoFanMode = param.asInt();
  Serial.print("Auto Fan Mode: "); Serial.println(autoFanMode ? "Enabled" : "Disabled");
}
```

We used the Grok AI tool for assistance

https://grok.com/