2025 FALL IOT102
WiFi-Controlled Smart Traffic Lights

1st Bui Tan Hung, 2nd Phan Thanh Dat, 3rd Luong Hoang An,
4th Mai Ngoc Bao, and Duc Ngoc Minh Dang
FPT University, Ho Chi Minh Campus, Vietnam
{staboyvn12, phanthanhdat123, anlh47, maingocbao365}@fpt.edu.vn, and ducdnm2@fe.edu.vn

Abstract

This paper presents a four-approach (NS/WE) smart traffic-light controller with two operating modes—Auto and Manual—plus a time-of-day Day/Night profile that automatically switches to Night Mode from 23:00 to 06:00. Responsibilities are partitioned across two microcontrollers: an ESP32 acts as the master to drive all vehicle signal heads and four 7-segment countdowns (via shift registers such as 74HC595D), while an Arduino handles the pedestrian subsystem (IR sensors, pedestrian LED heads, a push button, and an audible buzzer). In Auto mode, a finite-state machine enforces safety constraints (minimum green/red, intergreen/all-red) and admits pedestrian requests through a debounced, anti-glitch handshake with the Arduino; validated requests are inserted between vehicle phases with protected clearances, preserving non-conflicting movements and countdown integrity. In Manual mode, operators can advance/hold phases or force/clear a pedestrian phase, with all safety timers still honored. The Day/Night scheduler (driven by the ESP32's real-time clock) automatically switches to Night Flash Mode between 23:00 and 06:00, during which all approaches flash yellow in unison, following standard late-night caution operation. This mode conserves power, reduces wear on components, and maintains safety when traffic volume is low. Bench prototypes indicate reliable IR detection, stable countdown operation, and smooth automatic transition between Auto, Manual, and Night Flash modes.

Index Terms

Traffic signal control, IoT, ESP32, Arduino, Pedestrian crossing, Night flash mode, Blynk, UART, 7-segment display

I. Introduction

The Wi-Fi–controlled traffic light for junction and pedestrian safety is an IoT-based solution designed to improve traffic management at a four-way intersection (North–South and West–East). With rapid urbanization and increasing traffic density, many intersections still rely on fixed-timer signal controllers that cannot respond to actual traffic or pedestrian conditions. This often leads to wasted green time, congestion, and unsafe crossings for pedestrians.

In many developed countries—especially in Western nations and several advanced Asian countries such as Japan and South Korea—pedestrian push-button systems have been widely adopted and integrated into smart traffic control frameworks. These systems allow pedestrians to request a safe crossing phase on demand, improving both traffic flow efficiency and pedestrian safety. However, in Vietnam, such pedestrian button systems are still rarely implemented or only exist at a few major intersections in large cities. Most traffic lights operate on fixed timing cycles without direct pedestrian interaction, leading to either long waiting periods or unsafe spontaneous crossings.

Recognizing this gap, the main objective of this project is to design and develop a smart traffic light system that supports two operating modes—Auto and Manual—and that can insert a pedestrian phase on demand using a push button and IR sensors. The system also applies a Day/Night schedule, automatically switching to a Night Flash Mode (flashing yellow for all directions) from 23:00 to 06:00 to match real nighttime operation and improve energy efficiency.

The proposed system consists of three key parts. First, an ESP32 traffic light controller drives all vehicle signal heads for the NS/WE approaches and updates four 7-segment countdown displays (through shift registers). Second, an Arduino pedestrian unit reads the push button and IR sensors, controls the pedestrian LED indicators (Walk/Don't Walk), and activates a buzzer for audible warnings. Third, a simple Wi-Fi web interface hosted on the ESP32 provides an easy-to-use platform for authorized personnel to monitor the system and manually control signal phases (e.g., advance to next phase or trigger a pedestrian cycle) when necessary.

In operation, when a pedestrian presses the button, the Arduino waits for a brief confirmation period (2 seconds). After this period, it validates the request by checking the IR sensor. If the pedestrian is still present (IR sensor is LOW), the Arduino then sends a crossing request signal (`PEDESTRIAN_REQ`) to the ESP32. If the IR sensor does not detect a pedestrian after the wait, the request is considered invalid and is never sent, preventing false triggers.

II. Main proposal

The development of intelligent transportation systems has become an urgent necessity in modern urban environments, where traffic congestion and pedestrian safety remain persistent challenges. In Vietnam, most traffic lights still operate on fixed timing cycles that do not adapt to real-time road conditions or pedestrian demand. This limitation not only reduces traffic efficiency but also increases the risk of unsafe pedestrian crossings.

The topicality of this research lies in applying IoT technologies—specifically, Wi-Fi–based microcontroller communication—to enhance the flexibility and interactivity of traditional traffic signal systems. By introducing a pedestrian push-button interface combined with infrared sensing, the system allows pedestrians to safely request crossing phases on demand, similar to the systems commonly found in developed countries such as Japan or South Korea.

Scientifically, the project demonstrates the integration of embedded systems, real-time communication, and hardware–software co-design in a practical urban scenario. From a practical perspective, it provides a low-cost and scalable model for smart intersections that can be deployed in Vietnamese cities, supporting the country's ongoing transition toward intelligent transportation and smart-city infrastructure.

A. System models and block diagram

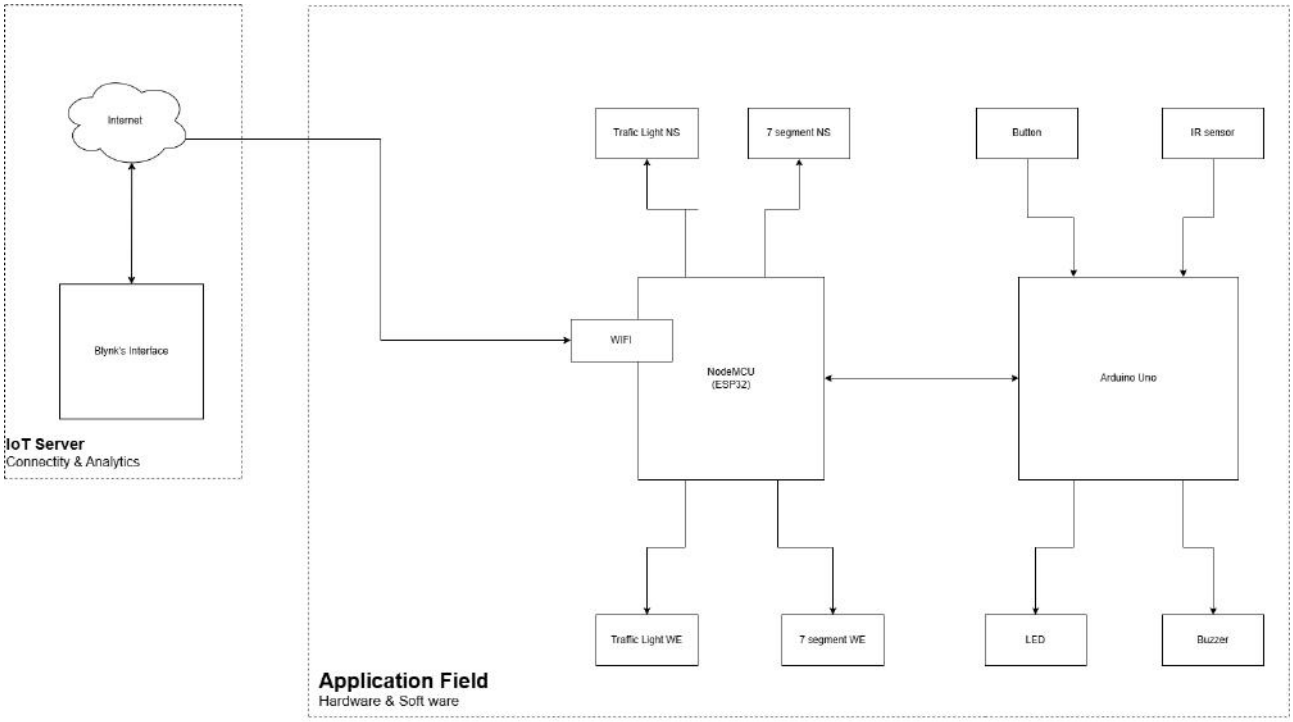

Fig. 1: Block diagram of the developed system.

B. Components and peripheral devices

The developed four-approach (NS/WE) smart traffic-light system integrates a set of electronic devices and modules—e.g., an ESP32 as the master controller, an Arduino UNO as the pedestrian unit, traffic-signal LED heads (R/Y/G), four 7-segment countdown displays driven via 74HC595D shift registers, a single active buzzer for the pedestrian phase, a push button for crossing requests, an IR sensor for pedestrian presence detection, a 3.3 V $\leftrightarrow$ 5 V level shifter, and appropriate drivers/resistors—to accomplish its intended functions. The ESP32 executes the finite-state machine, updates the countdown displays, and applies the Day/Night schedule (23:00–06:00), while the Arduino manages pedestrian input, validates IR presence, and coordinates buzzer activation via a REQ/ACK handshake with the ESP32.

TABLE I: System's components and peripheral devices

Components / Devices	**ID / Remarks**
ESP32 DevKit V1	Main controller (3.3 V logic), manages Wi-Fi connection and vehicle traffic phases
Arduino Uno R3	Secondary controller (5 V), handles pedestrian logic, IR sensor, and button input
74HC595D Shift Registers	8 units – 8-bit serial-in / parallel-out SPI output expanders for 7-segment driving
7-segment LED Displays	4 units – 5011BS **common anode** type, two-digit countdown timers for each approach
Traffic Light LED Modules	4 units – 5 V R/Y/G signal modules (North–South / West–East)
IR Sensor Module	1 unit – 5 V reflective IR module, detects pedestrian presence
Pedestrian Push Button	1 unit – N.O. momentary tactile button for crossing request
Pedestrian Signal LED	1 unit – 3 V LED (Walk / Don't Walk) indicator controlled by Arduino
Logic Level Converter	1 unit – MH bidirectional 3.3 V $\leftrightarrow$ 5 V converter (ESP32–Arduino–LED interface)
Jumper Wires	Various M–M and M–F types for inter-module signal and power connections
Breadboard / PCB	Used for prototyping and circuit testing

TABLE II: Interfacing Between ESP32 and Its Components (Pin-to-Pin)

ESP32 (DevKit V1)	Arduino UNO	7-Segment Displays	Traffic LED Heads (R/Y/G)	Logic Level Converter	Blynk / Wi-Fi
GND	GND (5 V side)	GND	GND (common return)	LV GND ↔ HV GND	Built-in Wi-Fi
3V3	—	—	—	LV (3.3 V) logic side	—
VIN (5 V)	5 V (VCC)	VCC → (74HC595D)	5 V input for modules	HV (5 V) converter input	—
GPIO 15	—	LATCH_A (74HC595D)	—	—	—
GPIO 2	—	CLOCK_A (74HC595D)	—	—	—
GPIO 4	—	DATA_A (74HC595D)	—	—	—
GPIO 5	—	LATCH_B (74HC595D)	—	—	—
GPIO 18	—	CLOCK_B (74HC595D)	—	—	—
GPIO 19	—	DATA_B (74HC595D)	—	—	—
GPIO 26	—	—	NS RED LED	—	—
GPIO 25	—	—	NS YELLOW LED	—	—
GPIO 33	—	—	NS GREEN LED	—	—
GPIO 13	—	—	WE RED LED	—	—
GPIO 12	—	—	WE YELLOW LED	—	—
GPIO 14	—	—	WE GREEN LED	—	—
GPIO 16 (RX2)	TX (Pin 3 via 5→3.3 converter)	—	—	LV RX ↔ HV TX	—
GPIO 17 (TX2)	RX (Pin 2 via 3.3→5 converter)	—	—	LV TX ↔ HV RX	—

TABLE III: Interfacing between Arduino UNO and its components (pin-to-pin)

Arduino UNO	Push Button	IR Sensor	Buzzer	Pedestrian LED	ESP32 (UART2)
GND	GND	GND	GND	GND	GND ↔ ESP32 GND
5 V (VCC)	VCC	VCC	+5 V	+5 V	5 V ↔ ESP32 VIN
Pin 10	Signal (input pull-up)	—	—	—	—
Pin 11	—	Signal (LOW when detected)	—	—	—
Pin 12	—	—	IN	—	—
Pin 13	—	—	—	LED (output)	—
Pin 2 (RX)	—	—	—	—	ESP32 TX (GPIO 17 via converter)
Pin 3 (TX)	—	—	—	—	ESP32 RX (GPIO 16 via converter)

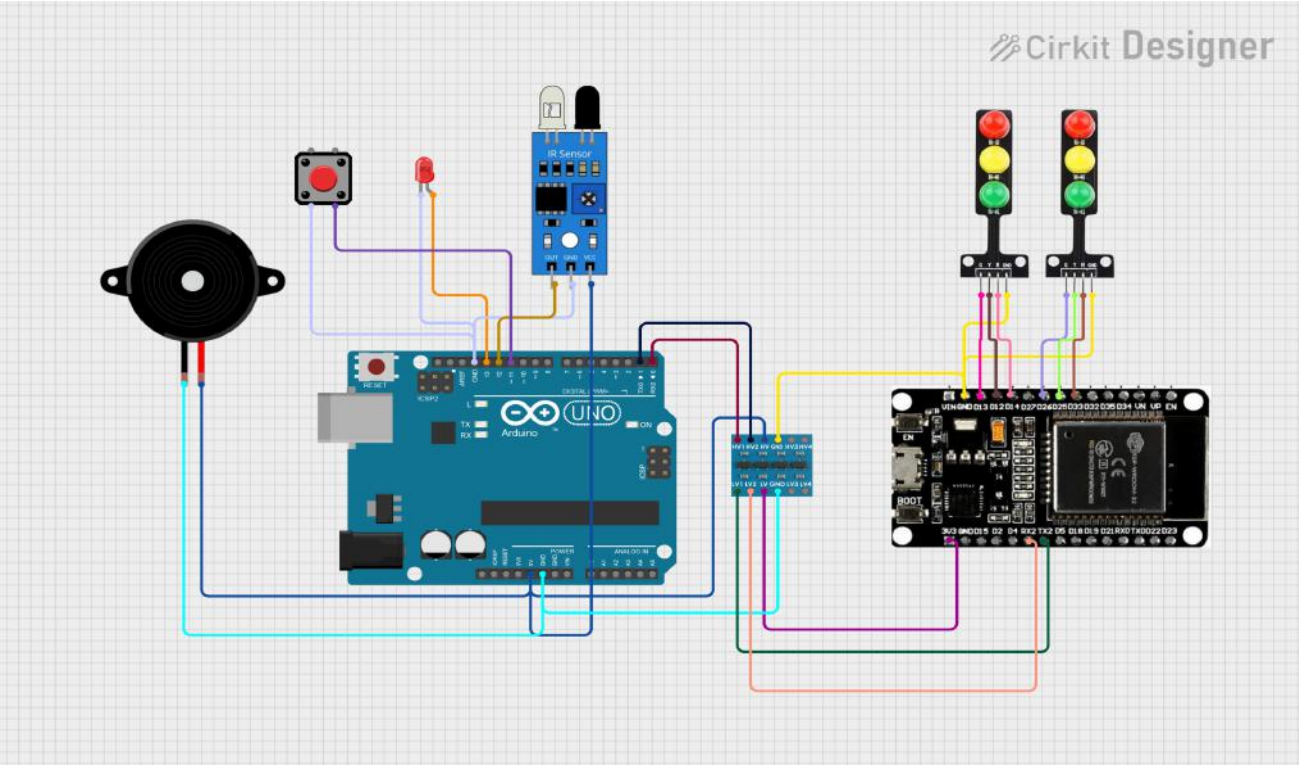

Fig. 2: Circuit connection between Arduino Uno, ESP32, IR sensor, level shifter, and 5 V traffic light modules (for the two lanes: North–South or West–East).

C. Software programming

Listing 1: ESP32 Four Ways Traffic Lights Unit Program

```
/*************** BLYNK CONFIG ***************/
#define BLYNK_TEMPLATE_ID TMPL6GlIX1T5U
#define BLYNK_TEMPLATE_NAME Traffic Light
#define BLYNK_AUTH_TOKEN 1FGL5bJGiIjhzNnrVCMEGccnRADRQ9y2
#define UART_RX_PIN 16 // RX2 (nhan tu Arduino TX qua level shifter 5V->3.3V)
#define UART_TX_PIN 17 // TX2 (gui sang Arduino RX qua level shifter 3.3V->5V)

HardwareSerial PedSerial(2); // dung UART2

#include <WiFi.h>
#include <WiFiClient.h>
#include <BlynkSimpleEsp32.h>
#include time.h

/*************** USER WIFI ***************/
char ssid[] = Hung 5G;
char pass[] = 0947532540;

// set up real time UTC + 7
const long gmtOffset_sec = 7 * 3600;
const int daylightOffset_sec = 0;

/*************** PIN CONFIG ***************/
#define A_RED_PIN 26
#define A_YELLOW_PIN 25
#define A_GREEN_PIN 33

#define B_RED_PIN 13
#define B_YELLOW_PIN 12
#define B_GREEN_PIN 14

#define LATCH_A_PIN 15
#define CLOCK_A_PIN 2
#define DATA_A_PIN 4

#define LATCH_B_PIN 5
#define CLOCK_B_PIN 18
#define DATA_B_PIN 19

/*************** 7-SEG MAP ***************/
const uint8_t segCode[10] = {
  0b11000000,
  0b11111001,
  0b10100100,
  0b10110000,
  0b10011001,
  0b10010010,
  0b10000010,
  0b11111000,
  0b10000000,
  0b10010000
};

const uint8_t segOff = 0xFF;

/*************** KHAI BAO TRANG THAI / BIEN DIEU KHIEN ***************/
int greenNS = 26;
int yellowNS = 4;
int greenWE = 26;
int yellowWE = 4;

int redNS = greenWE + yellowWE;
int redWE = greenNS + yellowNS;

boolean pedNS = false;
boolean pedB = false;

int pedestrianStatus = 11;

enum Phase { GREEN,
           YELLOW,
           RED };
Phase stateA = GREEN;
Phase stateB = RED;

unsigned long tA = 0;
unsigned long tB = 0;
```

```
int secA = greenNS;
int secB = redWE;

bool manual = false;
bool flashMode = false;

int reqNS = 0;
int reqWE = 0;

enum ControlSource { SRC_A,
                     SRC_B };
ControlSource lastControl = SRC_A;

/*************** KHAI BAO HAM TRUOC (PROTOTYPE) ***************/
void reportLaneStatusToBlynk(int laneAState);
void manualModeProcess();
void updateA_auto();
void updateB_auto();
void clearSegment(char lane);
void showNumber(char lane, int s);
void setA(int r, int y, int g);
void setB(int r, int y, int g);

/*************** LOW-LEVEL SEGMENT CONTROL ***************/
void sendSeg(int latch, int clock, int data, uint8_t v) {
  digitalWrite(latch, LOW);
  shiftOut(data, clock, MSBFIRST, v);
  digitalWrite(latch, HIGH);
}

void showNumber(char lane, int s) {
  if (s < 0) s = 0;
  if (s > 99) s = 99;
  int tens = s / 10;
  int ones = s % 10;

  if (lane == 'A') {
    sendSeg(LATCH_A_PIN, CLOCK_A_PIN, DATA_A_PIN, segCode[ones]);
    delay(2);
    sendSeg(LATCH_A_PIN, CLOCK_A_PIN, DATA_A_PIN, segCode[tens]);
  } else {
    sendSeg(LATCH_B_PIN, CLOCK_B_PIN, DATA_B_PIN, segCode[ones]);
    delay(2);
    sendSeg(LATCH_B_PIN, CLOCK_B_PIN, DATA_B_PIN, segCode[tens]);
  }
}

void clearSegment(char lane) {
  if (lane == 'A') {
    sendSeg(LATCH_A_PIN, CLOCK_A_PIN, DATA_A_PIN, segOff);
    sendSeg(LATCH_A_PIN, CLOCK_A_PIN, DATA_A_PIN, segOff);
  } else {
    sendSeg(LATCH_B_PIN, CLOCK_B_PIN, DATA_B_PIN, segOff);
    sendSeg(LATCH_B_PIN, CLOCK_B_PIN, DATA_B_PIN, segOff);
  }
}

/*************** LED CONTROL ***************/
void setA(int r, int y, int g) {
  digitalWrite(A_RED_PIN, r);
  digitalWrite(A_YELLOW_PIN, y);
  digitalWrite(A_GREEN_PIN, g);
}
void setB(int r, int y, int g) {
  digitalWrite(B_RED_PIN, r);
  digitalWrite(B_YELLOW_PIN, y);
  digitalWrite(B_GREEN_PIN, g);
}

/*************** GUI TRANG THAI LEN BLYNK ***************/
void reportLaneStatusToBlynk(int laneAState) {
  if (laneAState != -1) {
    Blynk.virtualWrite(V0, (laneAState == 0) ? 1 : 0);
    Blynk.virtualWrite(V1, (laneAState == 1) ? 1 : 0);
  }
}

/*************** AUTO MODE UPDATE ***************/
void updateA_auto() {
```

```
  if (millis() - tA >= 1000) {
    tA = millis();
    secA--;
    showNumber('A', secA);

    if (secA <= 0) {
      if (stateA == GREEN) {
        stateA = YELLOW;
        secA = yellowNS;
      } else if (stateA == YELLOW) {
        stateA = RED;
        secA = redNS;
      } else {
        stateA = GREEN;
        secA = greenNS;
      }
      if (stateA == RED && pedNS) {
        pedestrianStatus = 10;
      }
      if (pedNS && stateA == GREEN) {
        pedNS = false;
        pedestrianStatus = 11;
      }
    }

    if (stateA == GREEN) {
      setA(0, 0, 1);
    } else if (stateA == YELLOW) {
      setA(0, 1, 0);
    } else {
      setA(1, 0, 0);
    }
  }
}

void updateB_auto() {
  if (millis() - tB >= 1000) {
    tB = millis();
    secB--;
    showNumber('B', secB);

    if (secB <= 0) {
      if (stateB == GREEN) {
        stateB = YELLOW;
        secB = yellowWE;
      } else if (stateB == YELLOW) {
        stateB = RED;
        secB = redWE;
      } else {
        stateB = GREEN;
        secB = greenWE;
      }
    }

    if (stateB == GREEN) {
      setB(0, 0, 1);
    } else if (stateB == YELLOW) {
      setB(0, 1, 0);
    } else {
      setB(1, 0, 0);
    }
  }
}

/*************** MANUAL MODE ***************/
void manualModeProcess() {
  clearSegment('A');
  clearSegment('B');

  if (flashMode) {
    setA(0, 1, 0);
    setB(0, 1, 0);
    reportLaneStatusToBlynk(-1);
    delay(1000);
    setA(0, 0, 0);
    setB(0, 0, 0);
    delay(1000);
    return;
  }
```

```
236
237   if (lastControl == SRC_A) {
238     if (reqNS == 0) {
239       if (reqWE == 0) {
240         setA(0, 1, 0);
241         delay(2000);
242         reqWE = 1;
243       }
244       setA(1, 0, 0);
245       setB(0, 0, 1);
246       reportLaneStatusToBlynk(0);
247     } else if (reqNS == 1) {
248       if (reqWE == 1) {
249         setB(0, 1, 0);
250         delay(2000);
251         reqWE = 0;
252       }
253       setB(1, 0, 0);
254       setA(0, 0, 1);
255       reportLaneStatusToBlynk(1);
256     }
257   } else {
258     if (reqWE == 0) {
259       if (reqNS == 1) {
260         setB(0, 1, 0);
261         delay(2000);
262         reqNS = 0;
263       }
264       setB(1, 0, 0);
265       setA(0, 0, 1);
266       reportLaneStatusToBlynk(1);
267     } else if (reqWE == 1) {
268       if (reqNS == 1) {
269         setA(0, 1, 0);
270         delay(2000);
271         reqNS = 0;
272       }
273       setA(1, 0, 0);
274       setB(0, 0, 1);
275       reportLaneStatusToBlynk(0);
276     }
277   }
278 }
279
280 /*************** BLYNK CALLBACKS (8 BUTTONS) ***************/
281 BLYNK_WRITE(V0) {
282   if (!manual || flashMode) {
283     Blynk.virtualWrite(V0, 0);
284     return;
285   }
286   reqNS = 0;
287   reqWE = 0;
288   lastControl = SRC_A;
289 }
290
291 BLYNK_WRITE(V1) {
292   if (!manual || flashMode) {
293     Blynk.virtualWrite(V1, 0);
294     return;
295   }
296   reqNS = 1;
297   reqWE = 1;
298   lastControl = SRC_A;
299 }
300
301 BLYNK_WRITE(V2) {
302   flashMode = param.asInt();
303   if (!manual && flashMode) {
304     manual = true;
305     Blynk.virtualWrite(V3, 1);
306   }
307   if (flashMode) {
308     Blynk.virtualWrite(V0, 0);
309     Blynk.virtualWrite(V1, 0);
310   }
311 }
312
313 BLYNK_WRITE(V3) {
314   manual = param.asInt();
```

```
  if (!manual) {
    flashMode = false;
    Blynk.virtualWrite(V2, 0);
    Blynk.virtualWrite(V0, 0);
    Blynk.virtualWrite(V1, 0);
  } else {
    flashMode = false;
    Blynk.virtualWrite(V2, 0);
    reqNS = 1;
    reqWE = 0;
    secA = greenNS;
    secB = redWE;
    stateA = GREEN;
    stateB = RED;
    lastControl = SRC_A;
  }
}

/*************** SETUP ***************/
void setup() {
  pinMode(A_RED_PIN, OUTPUT);
  pinMode(A_YELLOW_PIN, OUTPUT);
  pinMode(A_GREEN_PIN, OUTPUT);
  pinMode(B_RED_PIN, OUTPUT);
  pinMode(B_YELLOW_PIN, OUTPUT);
  pinMode(B_GREEN_PIN, OUTPUT);
  pinMode(LATCH_A_PIN, OUTPUT);
  pinMode(CLOCK_A_PIN, OUTPUT);
  pinMode(DATA_A_PIN, OUTPUT);
  pinMode(LATCH_B_PIN, OUTPUT);
  pinMode(CLOCK_B_PIN, OUTPUT);
  pinMode(DATA_B_PIN, OUTPUT);
  Serial.begin(115200);
  delay(1000);
  Serial.println(ESP32 STARTED);
  PedSerial.begin(9600, SERIAL_8N1, UART_RX_PIN, UART_TX_PIN);
  setA(0, 0, 1);
  setB(1, 0, 0);
  showNumber('A', secA);
  showNumber('B', secB);
  reportLaneStatusToBlynk(1);
  WiFi.begin(ssid, pass);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
  }
  configTime(gmtOffset_sec, daylightOffset_sec, pool.ntp.org, time.nist.gov);
  struct tm timeinfo;
  while (!getLocalTime(&timeinfo)) {
    delay(1000);
  }
  Blynk.begin(BLYNK_AUTH_TOKEN, ssid, pass);
}

/*************** LOOP ***************/
void loop() {
  Blynk.run();
  struct tm timeinfo;
  if (!getLocalTime(&timeinfo)) {
    delay(1000);
    return;
  }
  PedSerial.write(pedestrianStatus);
  if (PedSerial.available()) {
    byte data = PedSerial.read();
    if (data == 11) {
      pedNS = false;
      pedestrianStatus = 11;
    }
    if (data == 10) {
      if (!manual && !flashMode) {
        if (stateA == RED) {
          pedestrianStatus = 10;
        }
        pedNS = true;
      }
    }
  }

  int hourNow = timeinfo.tm_hour;
```

```
394
395   if (manual) {
396     if (flashMode) {
397       clearSegment('A');
398       clearSegment('B');
399       setA(0, 1, 0);
400       setB(0, 1, 0);
401       reportLaneStatusToBlynk(-1);
402       delay(1000);
403       setA(0, 0, 0);
404       setB(0, 0, 0);
405       delay(500);
406     } else {
407       manualModeProcess();
408     }
409   } else {
410     if (hourNow >= 23 || hourNow < 6) {
411       clearSegment('A');
412       clearSegment('B');
413       setA(0, 1, 0);
414       setB(0, 1, 0);
415       reportLaneStatusToBlynk(-1);
416       delay(1000);
417       setA(0, 0, 0);
418       setB(0, 0, 0);
419       delay(1000);
420     } else {
421       reportLaneStatusToBlynk(-1);
422       updateA_auto();
423       updateB_auto();
424     }
425   }
426 }
```

Listing 2: Arduino Pedestrian Unit Program

```
1  #include <SoftwareSerial.h>
2  SoftwareSerial esp(2, 3); // RX, TX
3
4  #define BUTTON_PIN 10
5  #define IR_PIN 11
6  #define BUZZER_PIN 12
7  #define LED_PIN 13
8
9  unsigned long pressTime = 0;
10 bool waitingConfirm = false;
11 bool ledOn = false;
12 bool buzzerPlayed = false;
13
14 void setup() {
15   Serial.begin(115200);
16   esp.begin(9600);
17
18   pinMode(BUTTON_PIN, INPUT_PULLUP);
19   pinMode(IR_PIN, INPUT);
20   pinMode(BUZZER_PIN, OUTPUT);
21   pinMode(LED_PIN, OUTPUT);
22
23   digitalWrite(BUZZER_PIN, LOW);
24   digitalWrite(LED_PIN, LOW);
25
26   Serial.println(UNO READY);
27 }
28
29 void beepConfirm() {
30   for(int i = 0; i < 2; i++){
31     digitalWrite(BUZZER_PIN, HIGH);
32     delay(150);
33     digitalWrite(BUZZER_PIN, LOW);
34     delay(100);
35   }
36 }
37
38 void buzzerFiveTimes(){
39   for (int i = 0; i < 5; i++){
40     digitalWrite(BUZZER_PIN, HIGH);
41     delay(500);
42     digitalWrite(BUZZER_PIN, LOW);
43     delay(500);
```

```
  }
  buzzerPlayed = true;
}

void loop() {
  if (!waitingConfirm && digitalRead(BUTTON_PIN) == LOW) {
    Serial.println(Button pressed! Checking IR...);
    pressTime = millis();
    waitingConfirm = true;
    esp.write(10);
  }

  if (waitingConfirm && millis() - pressTime > 2000) {
    if (digitalRead(IR_PIN) == LOW) {
      Serial.println(Pedestrian confirmed, sending request...);
    } else {
      Serial.println(No pedestrian detected, cancel request.);
      esp.write(11);
    }
    waitingConfirm = false;
  }

  if (esp.available()) {
    byte msg = esp.read();
    Serial.print(ESP32: );
    Serial.println(msg);
    if (msg == 10 && !buzzerPlayed) {
      digitalWrite(LED_PIN, HIGH);
      if (!buzzerPlayed) {
        buzzerFiveTimes();
      }
    } else if (msg == 11) {
      digitalWrite(LED_PIN, LOW);
      buzzerPlayed = false;
    }
  }
}
```

D. Programming Flowchart

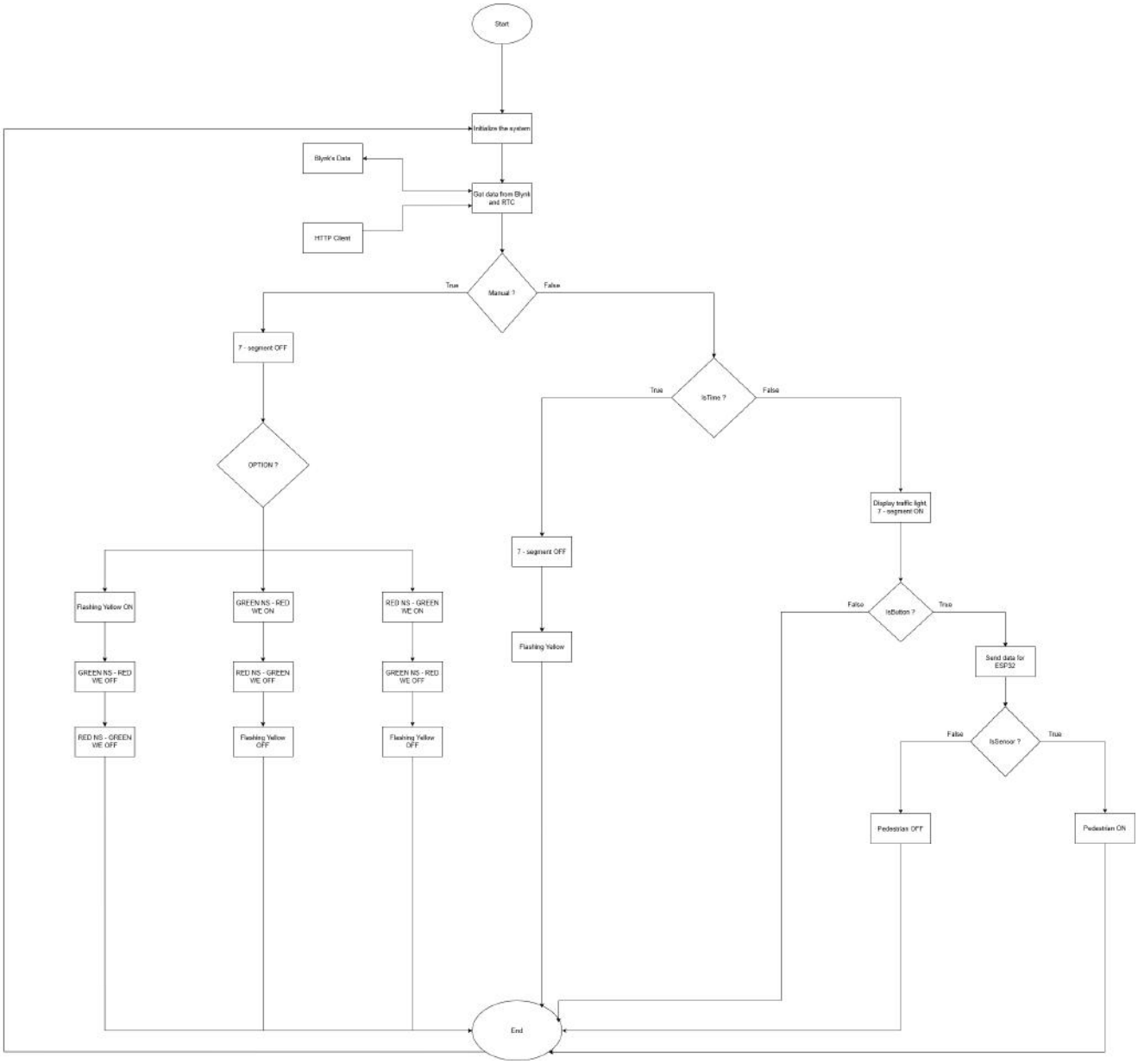

Fig. 3: Programming flowchart of the developed system.

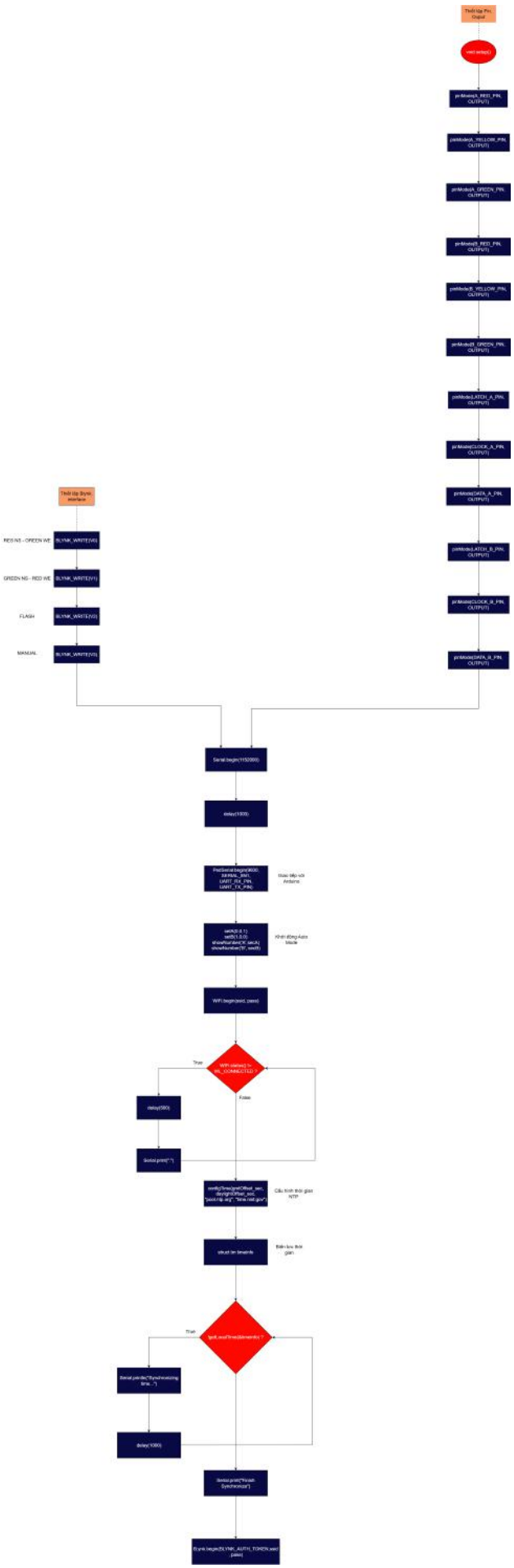

Fig. 4: Programming flowchart of the ESP32 system (setup part).

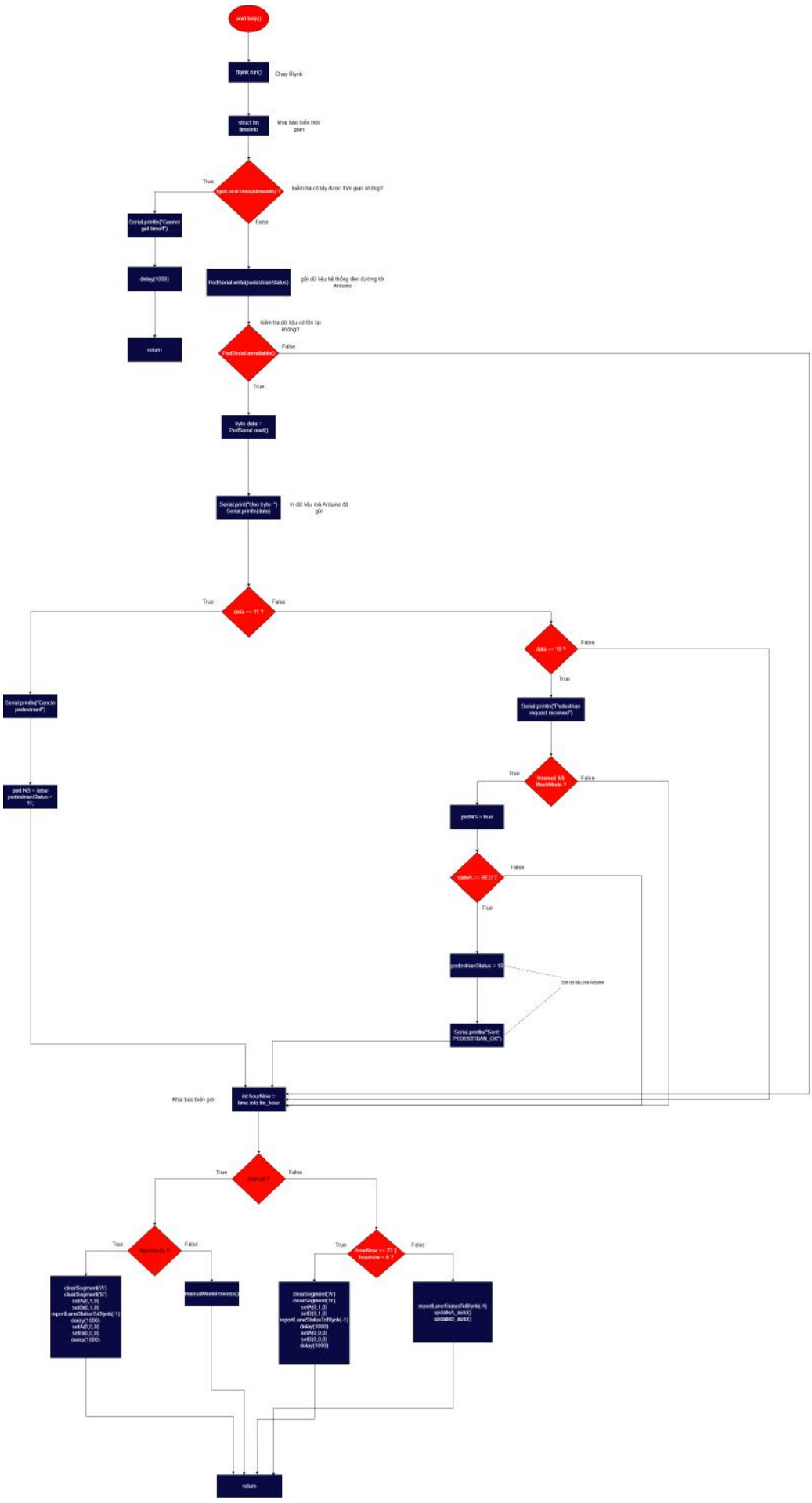

Fig. 5: Programming flowchart of the ESP32 system (loop part).

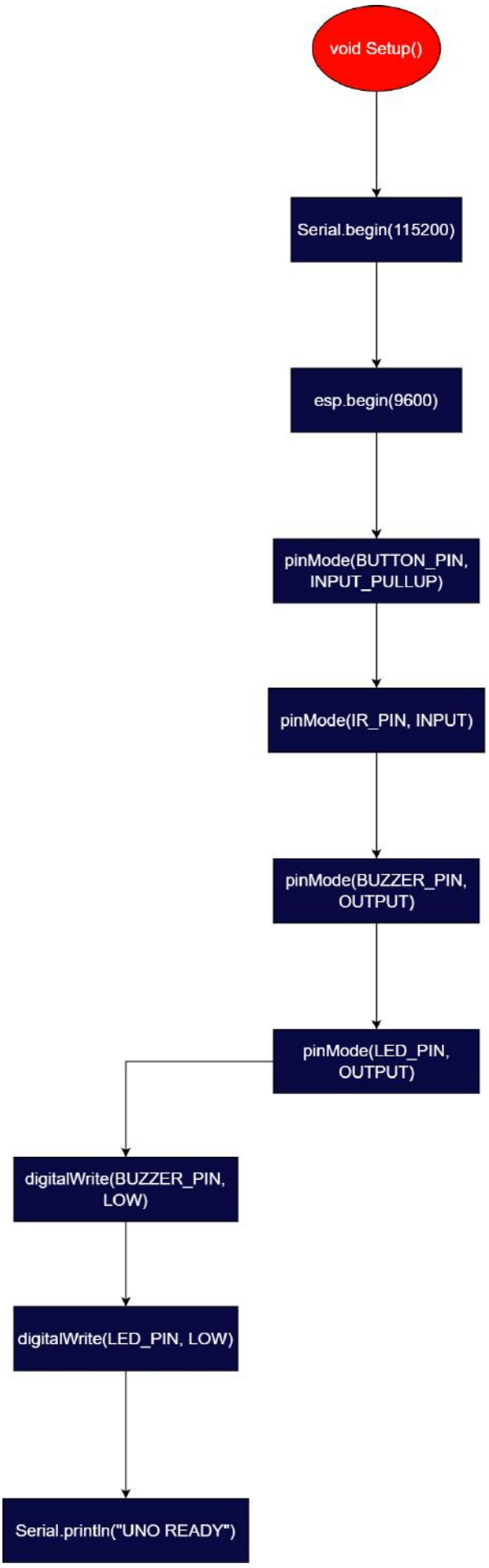

Fig. 6: Programming flowchart of the Arduino Uno system (setup part).

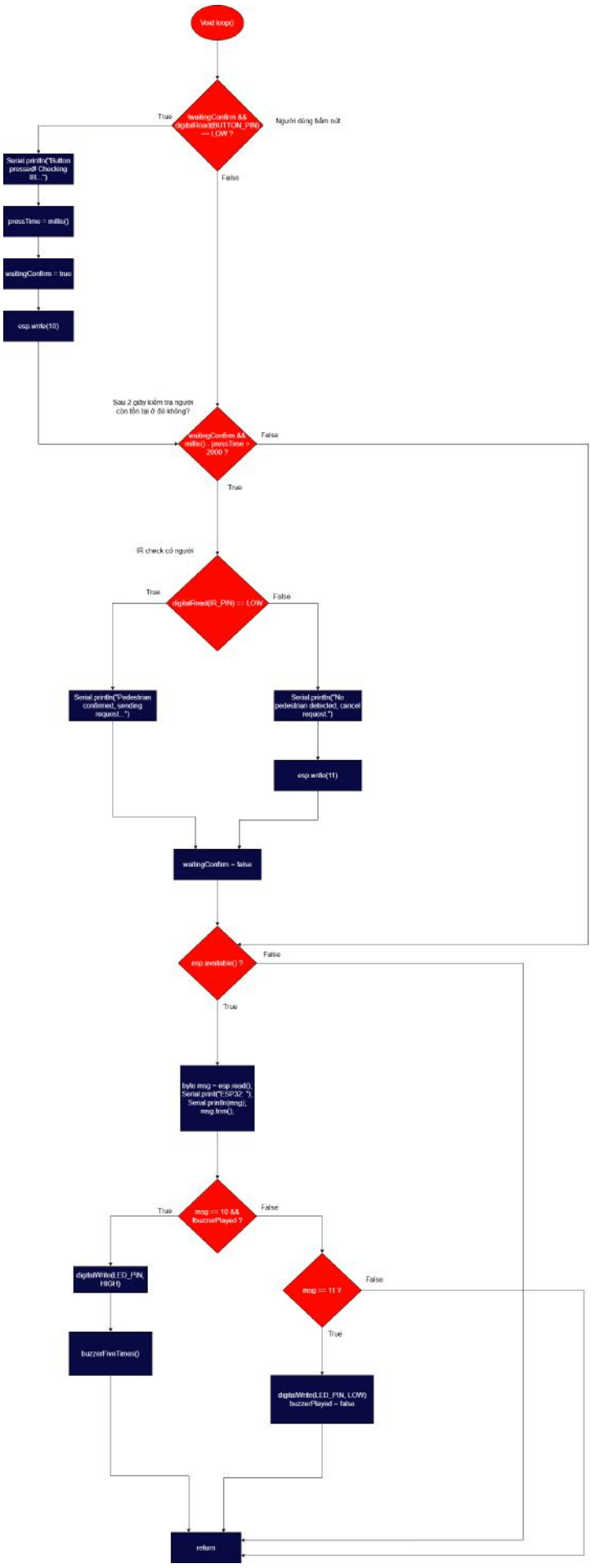

Fig. 7: Programming flowchart of the Arduino Uno system (loop part).

III. Results and Discussion

A. Prototype Implementation

The prototype of the Wi-Fi controlled traffic light system was implemented using an ESP32 microcontroller as the main controller and an Arduino Uno for pedestrian signal control. The ESP32 manages two traffic directions and communicates with the Arduino through UART. Each direction includes a set of red, yellow, and green LEDs and two 7-segment displays driven by 74HC595D shift registers to show countdown timers.

The system supports three operation modes: automatic, manual, and flashing. In automatic mode, traffic lights change according to pre-set timing cycles. In manual mode, users can control light phases via the Blynk mobile application. In flashing mode, all lights blink yellow for caution. The ESP32 connects to Wi-Fi and synchronizes real-time data with the Blynk cloud platform, while the Arduino operates pedestrian lights based on signals received from the ESP32.

Figure 3 shows the main control flow of the system.

1) Setting Up IoT Server (Blynk Cloud): To enable IoT connectivity, the ESP32 is integrated with the Blynk Cloud platform. A template named *Traffic Light* was created in the Blynk Console (device type: ESP32, Wi-Fi), then a device instance was provisioned to obtain the authentication token. The firmware declares the Blynk identifiers as follows:

Listing 3: Blynk template identifiers for the ESP32 firmware

```
#define BLYNK_TEMPLATE_ID TMPL6GlIX1T5U
#define BLYNK_TEMPLATE_NAME Traffic Light
#define BLYNK_AUTH_TOKEN 1FGL5bJGiIjhzNnrVCMEGccnRADRQ9y2
```

After declaring the identifiers, the ESP32 connects to Wi-Fi and the Blynk Cloud using the standard client libraries:

Listing 4: Minimal Blynk connection code on ESP32

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

const char* ssid = YOUR_WIFI_SSID;
const char* pass = YOUR_WIFI_PASSWORD;

void setup() {
  Blynk.begin(BLYNK_AUTH_TOKEN, ssid, pass);
}

void loop() {
  Blynk.run();
}
```

In the *Web/Mobile Dashboard*, virtual pins are mapped to control actions, e.g., `V0` = next phase, `V1` = force pedestrian phase, `V2` = flashing mode toggle, and `V3` = status indicator. This enables manual override during experiments while the automatic scheduler remains active.

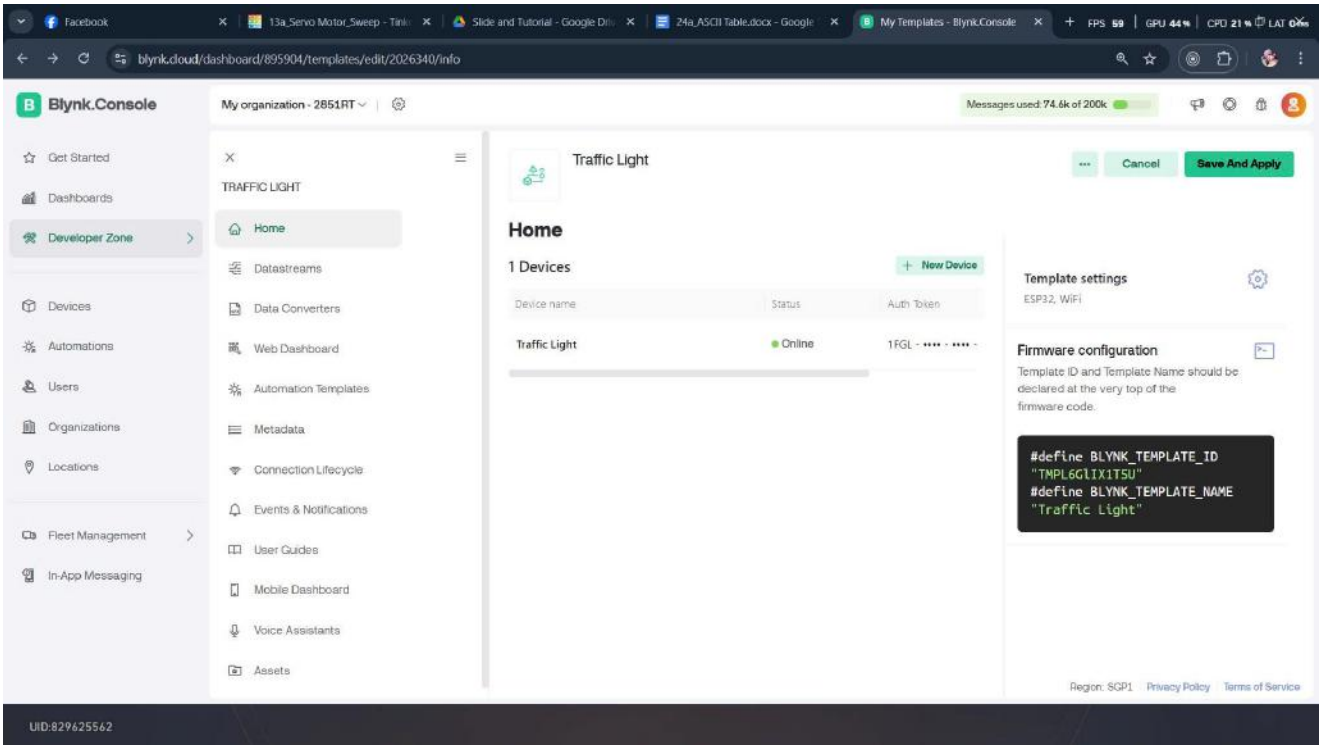

Fig. 8: Blynk Console *template* for the project (*Traffic Light*): device online and firmware identifiers (Template ID, Template Name, Auth Token).

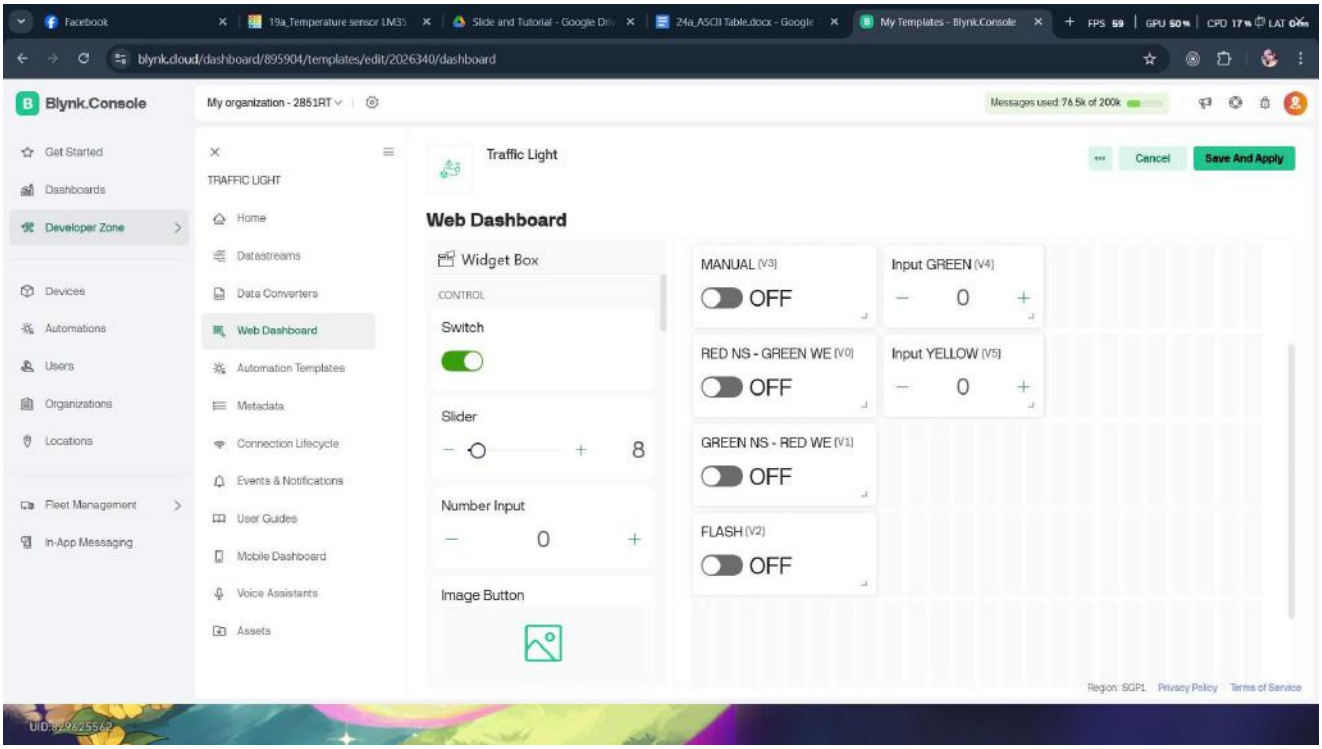

Fig. 9: Blynk *Web Dashboard* layout

Security note: for public submissions, rotate the Blynk `AUTH_TOKEN` or keep it in a private header (e.g., `secrets.h`) excluded from version control.

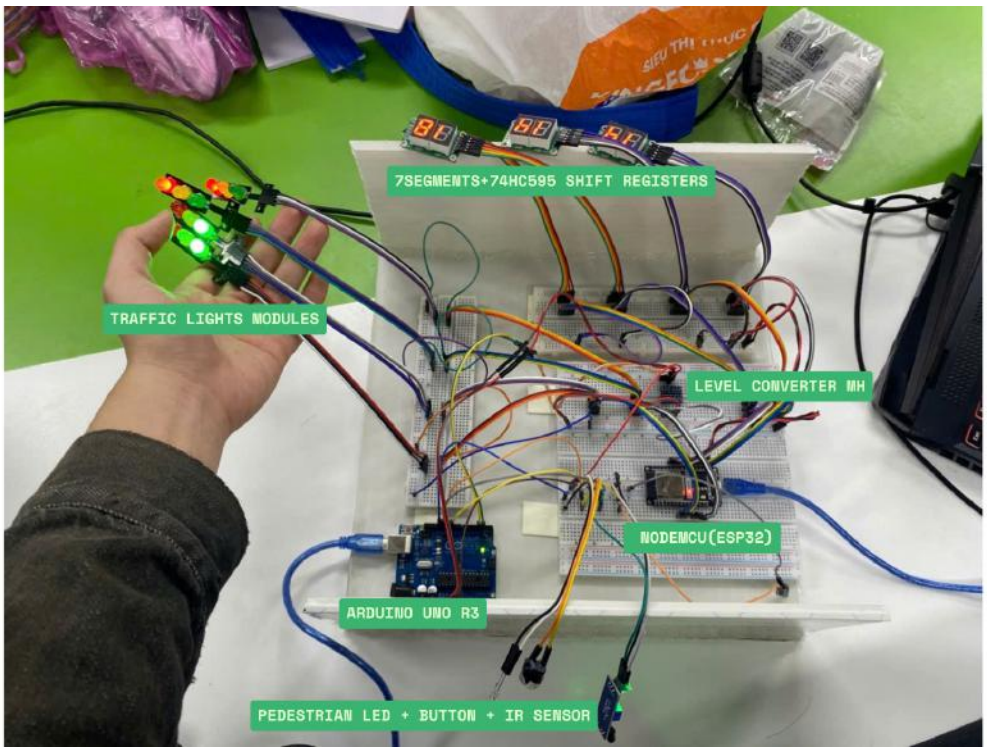

Fig. 10: Practical *Circuit design*: experimental setup with ESP32, Arduino, 7-segment displays, traffic light modules, IR sensors, and a pedestrian interface integrated via UART and Wi-Fi.

Fig. 11: UART handshake using byte protocol (10 = request, 11 = cancel/clear).

Arduino: Sending Request

```
if (!waitingConfirm && digitalRead(BUTTON_PIN)
      == LOW) {
  Serial.println(Button pressed! Checking IR...
      );
  pressTime = millis();
  waitingConfirm = true;
  esp.write(10); // PEDESTRIAN_REQ
}

if (waitingConfirm && millis() - pressTime >
    2000) {
  if (digitalRead(IR_PIN) == LOW) {
    Serial.println(Pedestrian confirmed, keep
        request.);
  } else {
    Serial.println(No pedestrian detected,
        cancel request.);
    esp.write(11); // PEDESTRIAN_CANCEL
  }
  waitingConfirm = false;
}
```

ESP32: Receiving/Replying

```
if (PedSerial.available()) {
  byte data = PedSerial.read();
  Serial.print(UNO byte: );
  Serial.println(data);

  if (data == 11) {
    pedNS = false;
    pedestrianStatus = 11;
  }
  if (data == 10) {
    if (!manual && !flashMode) {
      if (stateA == RED)
        pedestrianStatus = 10;
      pedNS = true;
    }
  }
}
PedSerial.write(pedestrianStatus);
```

Listing 5: Night Flash (yellow) logic in AUTO mode

```
  } else {
    // AUTO MODE
    if (hourNow >= 23 || hourNow < 6) {
      // Dem: chop vang tu dong
      clearSegment('A');
      clearSegment('B');
      setA(0, 1, 0); // A vang
      setB(0, 1, 0); // B vang
      reportLaneStatusToBlynk(-1);
      delay(1000);
      setA(0, 0, 0);
      setB(0, 0, 0);
      delay(1000);
    } else {
```

```
      // Ban ngay: chay auto
      reportLaneStatusToBlynk(-1);
      updateA_auto();
      updateB_auto();
    }
  }
}
```

Fig. 12: Night Flash Mode: both approaches flash yellow synchronously, countdowns disabled.

B. Experimental Results

At this stage, the ESP32-based controller operates the vehicle phases and countdowns reliably in **Automatic** mode; mode switching and countdown updates remained stable across repeated day/night cycles.

The **pedestrian subsystem** has been **successfully integrated**: the push button triggers a crossing request, the IR presence check validates the request, and the ESP32–Arduino UART link exchanges `REQ`/`ACK` messages consistently. When a pedestrian phase is admitted, the system inserts it between vehicle phases with a short intergreen. In normal cycling, the transition from **GREEN** to **RED** includes an intentional **single yellow blink** (one-beat *intergreen*) to signal clearance rather than switching abruptly.

In **Manual (Wi-Fi) mode**, when the Wi-Fi link is strong, end-to-end command latency is **negligible (practically 0 s)** and phase changes occur immediately. Under weak Wi-Fi (low RSSI), commands are sometimes delayed by about **1–2 s**, and a few can be dropped, leaving the system in the previous phase until the next successful command. These behaviors reflect transport conditions rather than firmware logic; adding connection watchdogs and command acknowledgments mitigated the impact but cannot fully mask poor connectivity.

C. Discussion

The implemented system successfully demonstrates a robust ESP32-based traffic light controller. The UART communication between the ESP32 and Arduino was stable, enabling the successful integration of the pedestrian subsystem. The system correctly handles button requests, IR validation, and cross-communication. Furthermore, the RTC

synchronization via NTP functioned reliably, enabling automatic switching between Day (normal FSM) and Night Flash (all-yellow blinking) modes as designed.

IV. Conclusion

In this project, an ESP32-based Wi-Fi traffic light control system was successfully designed and implemented. The system operates automatically and fully integrates a pedestrian control subsystem via a reliable UART link with an Arduino. The integration of RTC synchronization night mode further enhances the system's autonomy, enabling scheduled Day/Night profile switching. All planned features, including automatic, manual, and pedestrian-actuated phases, function as expected.

V. Author's Contribution

All members of the group contributed collaboratively to the completion of this project. The hardware design and circuit implementation were mainly developed by the technical members responsible for assembling and testing the ESP32–Arduino system. The software development, including traffic light control logic and Wi-Fi connectivity, was handled by the programming team. Documentation, report writing, and presentation preparation were carried out collectively by all members to ensure the quality and completeness of the project.

TABLE IV: Author's contribution

#	Student ID	Student Name	Tasks	Contribution
1	SE196636	Bui Tan Hung	Block diagram, flowchart, PPTX, presentation	25%
2	SE190501	Mai Ngoc Bao	Design model, complete project	20%
3	SE182374	Phan Thanh Dat	Program Arduino and ESP32 coordination logic	30%
4	SE193347	Luong Hoang An	Write report, sub program Arduino and ESP32	25%
Total				100%