Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Самарский государственный технический университет»
(ФГБОУ ВО «СамГТУ»)

Институт «Автоматики и инженерных технологий»

Реализация конечного автомата (автомобильный светофор) на контроллере двумя методами

Лабораторная работа №2, отчёт

Выполнил студенты

3 курса, 3-ИАИТ-110 группы

Санталов Александр
Шмарин Илья
Ляпин Павел

Самара, 2025 г.

Цель работы – разработать и отладить конечный автомат (КДА) на платформе Arduino, моделирующий управление светофором автомобильного перекрёстка. Реализовать функционал двумя способами.

Задание 1:
Разработать и отладить конечный автомат (КДА) на платформе Arduino, моделирующий управление светофором автомобильного перекрёстка с использованием switch-case по текущему состоянию.

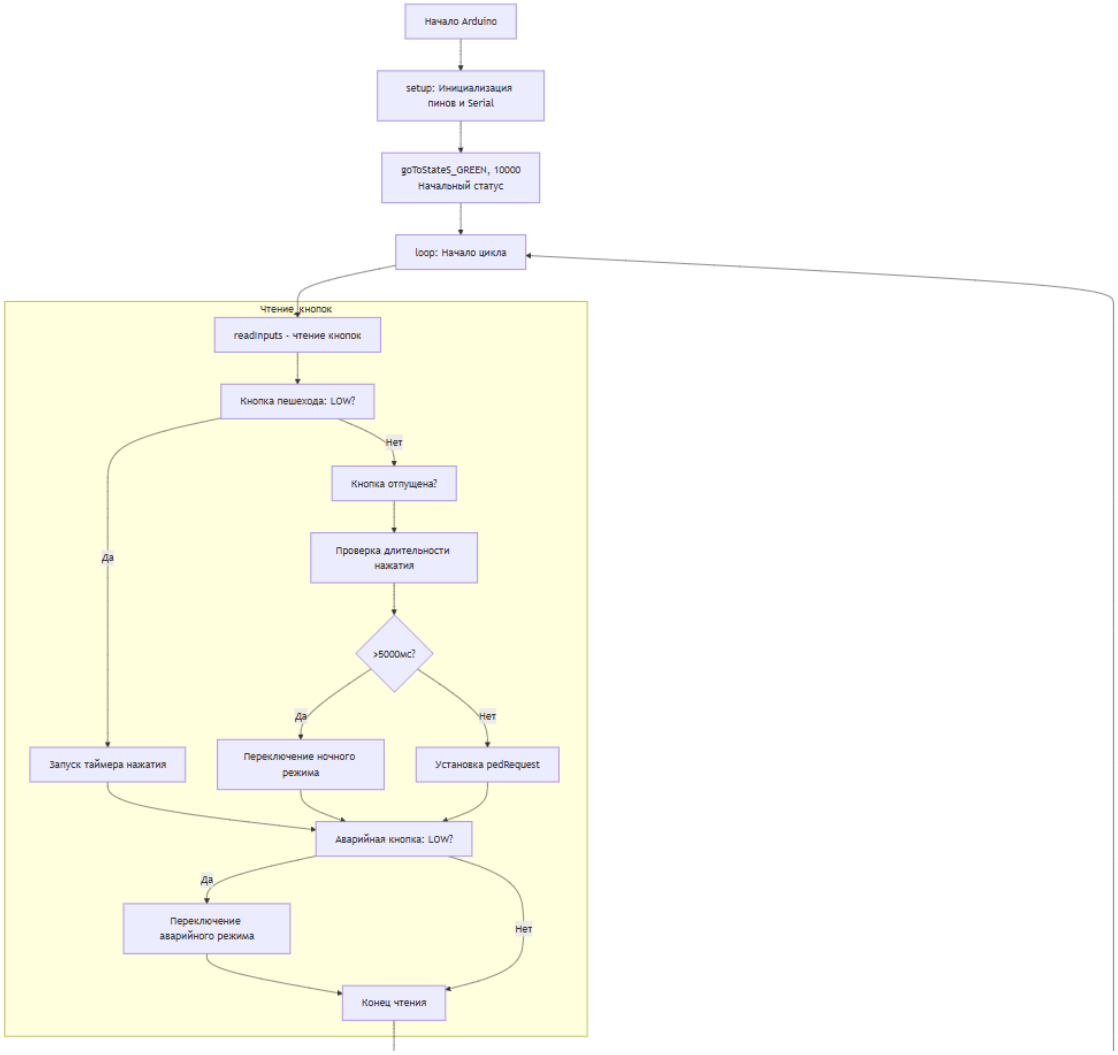
```
enum State { S_GREEN, S_YELLOW, S_RED, S_WARNING };
State currentState = S_GREEN;
unsigned long stateStart = 0;
unsigned long duration = 0;

bool pedRequest = false;
bool emergency = false;
bool night_mode_toggle = false;

// Пины
const int BUTTON_PIN = 2;
const int EMERGENCY_PIN = 3;
const int GREEN_PIN = 11;
const int YELLOW_PIN = 12;
const int RED_PIN = 13;

// Время удержания кнопки для ночного режима
uint32_t timerPressStart = 0;
uint32_t timerPressEnd = 0;
const int NIGHT_TOGGLE_HOLD_TIME = 5000;

// Для мигающего режима
static unsigned long lastBlinkTime = 0;
static bool yellowState = false;

// Дребезг
const unsigned long DEBOUNCE_DELAY = 50;
unsigned long lastButtonPressTime = 0;
unsigned long lastEmergencyPressTime = 0;
bool emergencyButtonPressed = false;


// ---- Логирование ----
String stateToString(State s) {
  switch (s) {
    case S_GREEN: return "ЗЕЛЁНЫЙ";
    case S_YELLOW: return "ЖЁЛТЫЙ";
    case S_RED: return "КРАСНЫЙ";
    case S_WARNING: return "ВНИМАНИЕ";
  }
  return "НЕИЗВЕСТНО";
}

```

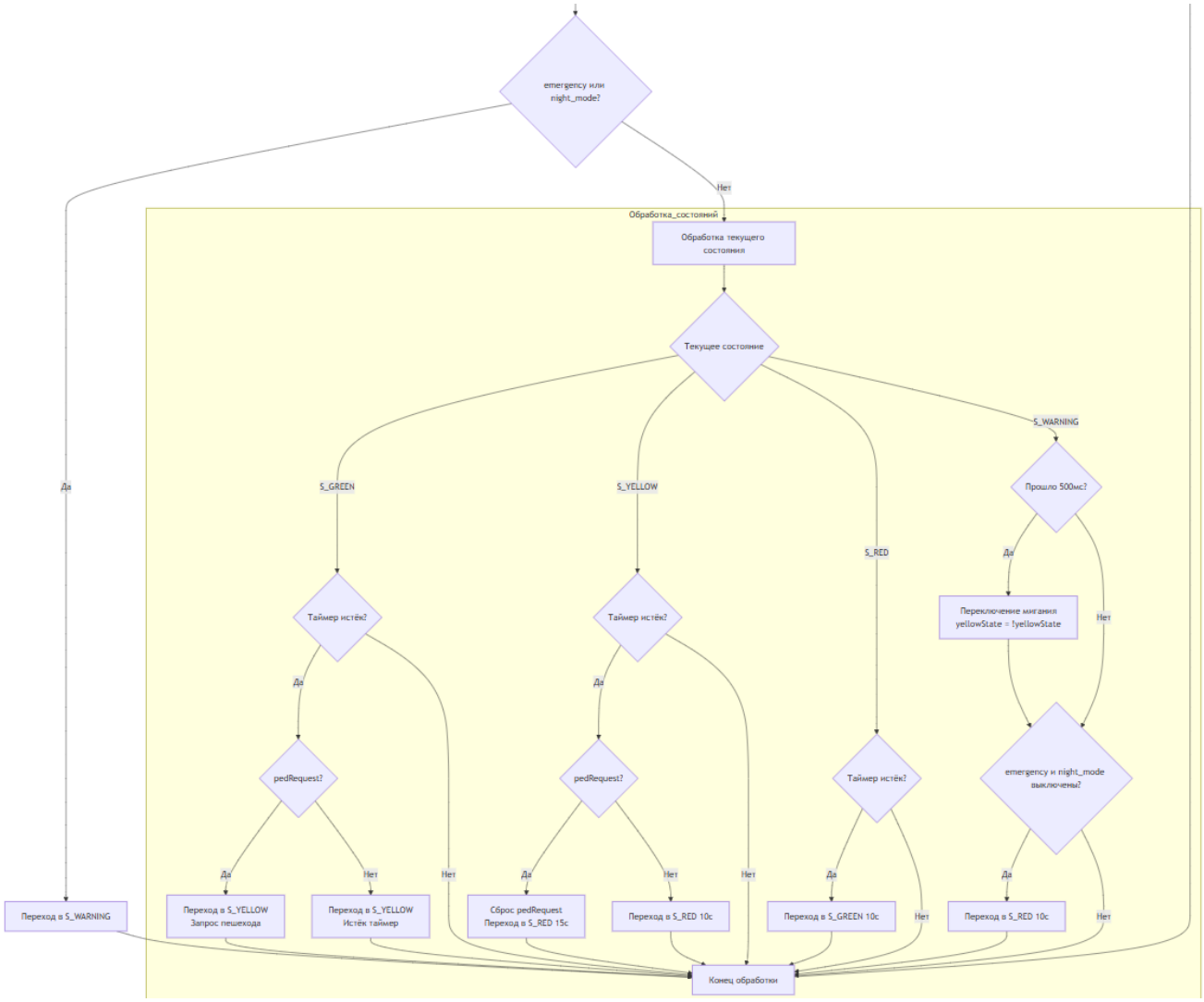
```
44
45 // ---- Управление светодиодами ----
46 void setOutputsForState(State s) {
47   switch (s) {
48     case S_GREEN:
49       digitalWrite(RED_PIN, LOW);
50       digitalWrite(YELLOW_PIN, LOW);
51       digitalWrite(GREEN_PIN, HIGH);
52       break;
53
54     case S_YELLOW:
55       digitalWrite(RED_PIN, LOW);
56       digitalWrite(YELLOW_PIN, HIGH);
57       digitalWrite(GREEN_PIN, LOW);
58       break;
59
60     case S_RED:
61       digitalWrite(RED_PIN, HIGH);
62       digitalWrite(YELLOW_PIN, LOW);
63       digitalWrite(GREEN_PIN, LOW);
64       break;
65
66     case S_WARNING:
67       digitalWrite(RED_PIN, LOW);
68       digitalWrite(YELLOW_PIN, yellowState ? HIGH : LOW);
69       digitalWrite(GREEN_PIN, LOW);
70       break;
71   }
72 }
73
74
75 // ---- Переход между состояниями ----
76 void goToState(State newState, unsigned long dur, const char* reason) {
77   unsigned long now = millis();
78   Serial.print("[");
79   Serial.print(now);
80   Serial.print("мс] ");
81   Serial.print(stateToString(currentState));
82   Serial.print(" -> ");
83   Serial.print(stateToString(newState));
84   Serial.print(" | Причина: ");
85   Serial.println(reason);
86
87   currentState = newState;
88   stateStart = now;
89   duration = dur;
90   setOutputsForState(newState);
91 }
92
93
94 // ---- Чтение кнопок ----
95 void readInputs() {
96   unsigned long now = millis();
```

```
97
98    // --- Кнопка пешехода ---
99    int buttonState = digitalRead(BUTTON_PIN);
100   if (buttonState == LOW && timerPressStart == 0 && (now -
          lastButtonPressTime > DEBOUNCE_DELAY)) {
101     timerPressStart = now;
102     lastButtonPressTime = now;
103   }
104   else if (buttonState == HIGH && timerPressStart != 0 && (now -
          lastButtonPressTime > DEBOUNCE_DELAY)) {
105     timerPressEnd = now;
106     lastButtonPressTime = now;
107
108     unsigned long pressDuration = timerPressEnd - timerPressStart;
109
110     if (pressDuration >= NIGHT_TOGGLE_HOLD_TIME) {
111       night_mode_toggle = !night_mode_toggle;
112       Serial.print("[");
113       Serial.print(now);
114       Serial.println("мс] Ночной режим переключен");
115     }
116     else {
117       pedRequest = true;
118       Serial.print("[");
119       Serial.print(now);
120       Serial.println("мс] Нажата кнопка пешехода");
121     }
122
123     timerPressEnd = 0;
124     timerPressStart = 0;
125   }
126
127   // --- Кнопка аварийного режима ---
128   int emergencyState = digitalRead(EMERGENCY_PIN);
129
130   if (emergencyState == LOW && !emergencyButtonPressed && (now -
          lastEmergencyPressTime > DEBOUNCE_DELAY)) {
131     emergencyButtonPressed = true;
132     lastEmergencyPressTime = now;
133     emergency = !emergency;
134
135     Serial.print("[");
136     Serial.print(now);
137     Serial.print("мс] Аварийный режим ");
138     Serial.println(emergency ? "ВКЛ" : "ВЫКЛ");
139   }
140   else if (emergencyState == HIGH && emergencyButtonPressed && (now -
          lastEmergencyPressTime > DEBOUNCE_DELAY)) {
141     emergencyButtonPressed = false;
142     lastEmergencyPressTime = now;
143   }
144 }
145
```

```
146
147 void setup() {
148   pinMode(RED_PIN, OUTPUT);
149   pinMode(YELLOW_PIN, OUTPUT);
150   pinMode(GREEN_PIN, OUTPUT);
151   pinMode(BUTTON_PIN, INPUT_PULLUP);
152   pinMode(EMERGENCY_PIN, INPUT_PULLUP);
153   Serial.begin(9600);
154   goToState(S_GREEN, 10000, "Начальный статус");
155 }
156
157
158 void loop() {
159   readInputs();
160
161   // --- Режим предупреждения (ночь или авария) ---
162   if (emergency || night_mode_toggle) {
163     if (currentState != S_WARNING)
164       goToState(S_WARNING, 500, emergency ? "Аварийный режим ВКЛ" : "Аварий
          ный режим ВЫКЛ");
165   }
166
167   switch (currentState) {
168     case S_GREEN:
169       if (millis() - stateStart >= duration) {
170         if (pedRequest) {
171           goToState(S_YELLOW, 3000, "Запрос пешехода");
172         } else {
173           goToState(S_YELLOW, 3000, "Истёк таймер");
174         }
175       }
176       break;
177
178     case S_YELLOW:
179       if (millis() - stateStart >= duration) {
180         if (pedRequest) {
181           pedRequest = false;
182           goToState(S_RED, 15000, "Запрос пешехода (удлинённый красный)");
183         } else {
184           goToState(S_RED, 10000, "Истёк таймер");
185         }
186       }
187       break;
188
189     case S_RED:
190       if (millis() - stateStart >= duration) {
191         goToState(S_GREEN, 10000, "Истёк таймер");
192       }
193       break;
194
195     case S_WARNING:
196       if (millis() - lastBlinkTime >= 500) {
197         yellowState = !yellowState;
```

```
198           setOutputsForState(currentState);
199           lastBlinkTime = millis();
200         }
201
202         if (!emergency && !night_mode_toggle)
203           goToState(S_RED, 10000, "Аварийный/Ночной режим ВЫКЛ");
204         break;
205     }
206 }
```





Задание 2:

Разработать и отладить конечный автомат (КДА) на платформе Arduino, моделирующий управление светофором автомобильного перекрёстка с использованием двумерной таблицы (состояние × событие), где каждая ячейка содержит обработчик для комбинации состояние/событие.

```
enum State { S_GREEN, S_YELLOW, S_RED, S_WARNING, NUM_STATES };
enum Event { E_NONE, E_TIMER, E_PED, E_EMERGENCY_ON, E_EMERGENCY_OFF,
   E_NIGHT_ON, E_NIGHT_OFF, NUM_EVENTS };

State currentState = S_GREEN;
unsigned long stateStart = 0;
unsigned long stateDur = 0;

// Переменные для кнопок
bool pedRequest = false;
bool emergency = false;
bool night_mode_toggle = false;

const int BUTTON_PIN = 2;
const int EMERGENCY_PIN = 3;
const int GREEN_PIN = 11;
```

```
const int YELLOW_PIN = 12;
const int RED_PIN = 13;

// Антидребезг
const uint32_t BTN_DEB = 50;
uint32_t buttonTmr = 0;
uint32_t emergencyTmr = 0;
uint32_t timerPressStart = 0;
const int NIGHT_TOGGLE_HOLD_TIME = 5000;

// Таймер мигания
unsigned long blinkStart = 0;
const unsigned long BLINK_INTERVAL = 500;

typedef void (*Handler)(void);
Handler fsmTable[NUM_STATES][NUM_EVENTS];

// --- Логирование переходов ---
void logTransition(State oldState, State newState, const char* reason) {
  Serial.print("[");
  Serial.print(millis());
  Serial.print("мс] ");

  // Слева- текущее состояние
  switch (oldState) {
    case S_GREEN: Serial.print("ЗЕЛЁНЫЙ"); break;
    case S_YELLOW: Serial.print("ЖЁЛТЫЙ"); break;
    case S_RED: Serial.print("КРАСНЫЙ"); break;
    case S_WARNING: Serial.print("ВНИМАНИЕ"); break;
  }

  Serial.print(" -> ");

  // Справа- новое состояние
  switch (newState) {
    case S_GREEN: Serial.print("ЗЕЛЁНЫЙ"); break;
    case S_YELLOW: Serial.print("ЖЁЛТЫЙ"); break;
    case S_RED: Serial.print("КРАСНЫЙ"); break;
    case S_WARNING: Serial.print("ВНИМАНИЕ"); break;
  }

  Serial.print(" | Причина: ");
  Serial.println(reason);
}

// --- Управление светодиодами ---
void setOutputsForState(State s) {
  switch (s) {
    case S_GREEN:
      digitalWrite(RED_PIN, LOW);
      digitalWrite(YELLOW_PIN, LOW);
      digitalWrite(GREEN_PIN, HIGH);
      break;
```

```
    case S_YELLOW:
      digitalWrite(RED_PIN, LOW);
      digitalWrite(YELLOW_PIN, HIGH);
      digitalWrite(GREEN_PIN, LOW);
      break;
    case S_RED:
      digitalWrite(RED_PIN, HIGH);
      digitalWrite(YELLOW_PIN, LOW);
      digitalWrite(GREEN_PIN, LOW);
      break;
    case S_WARNING:
      // стартуем с выключенного жёлтого- мигание будет управляться в loop()
      digitalWrite(RED_PIN, LOW);
      digitalWrite(YELLOW_PIN, LOW);
      digitalWrite(GREEN_PIN, LOW);
      break;
  }
}

// --- Обработчики событий ---
void onGreenTimer() {
  State oldState = currentState;
  currentState = S_YELLOW;
  stateDur = 3000;
  stateStart = millis();
  setOutputsForState(currentState);
  logTransition(oldState, currentState, "Timer expired");
}

void onGreenPed() {
  State oldState = currentState;
  currentState = S_YELLOW;
  stateDur = 3000;
  stateStart = millis();
  setOutputsForState(currentState);
  logTransition(oldState, currentState, "Pedestrian request");
}

void onYellowTimer() {
  State oldState = currentState;
  currentState = S_RED;
  stateDur = pedRequest ? 15000 : 10000;
  stateStart = millis();
  setOutputsForState(currentState);
  if (pedRequest) {
    pedRequest = false;
    logTransition(oldState, currentState, "Истёк таймер + запрос пешехода");
  } else {
    logTransition(oldState, currentState, "Истёк таймер");
  }
}
```

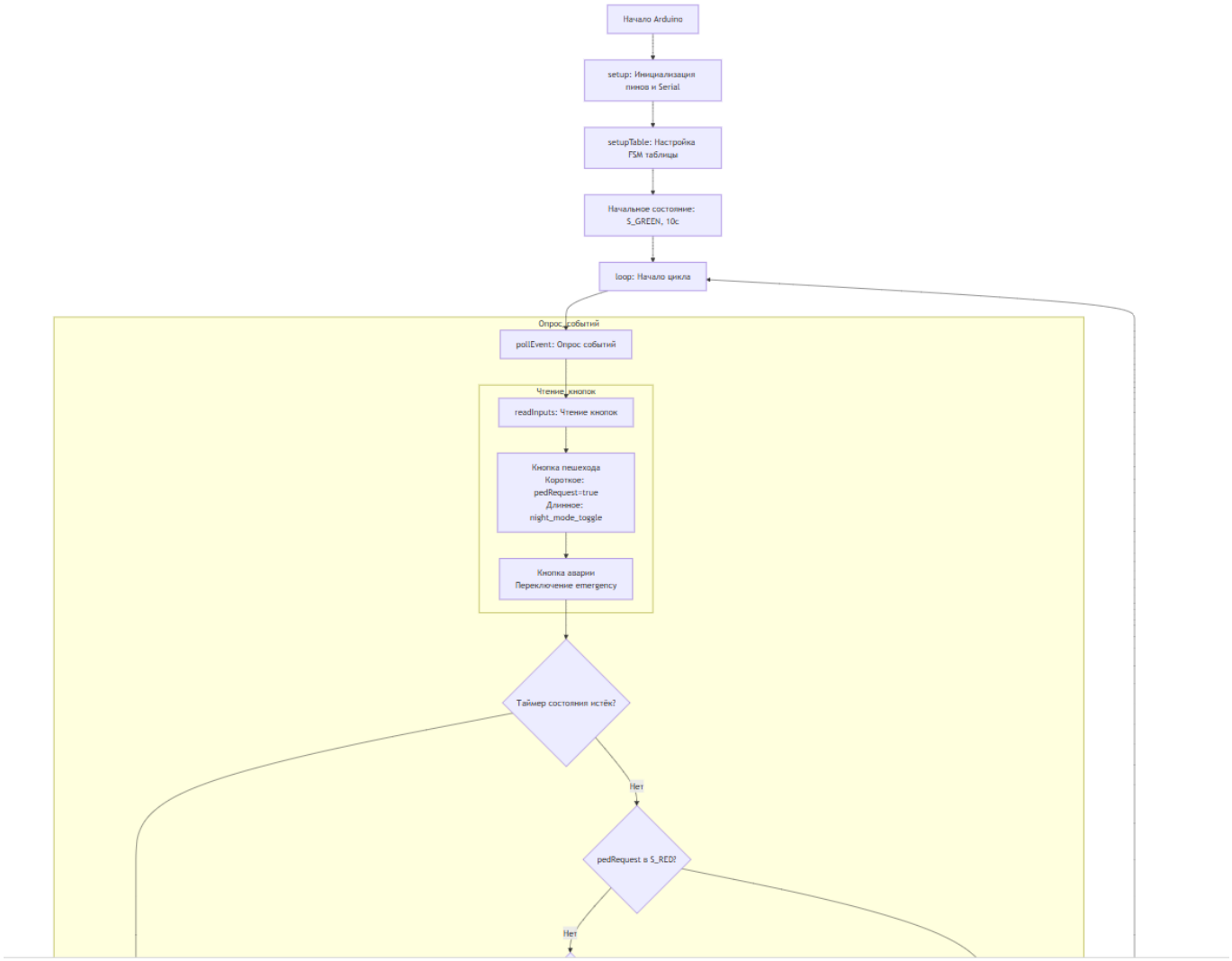
```
120
121 void onRedTimer() {
122   State oldState = currentState;
123   currentState = S_GREEN;
124   stateDur = 10000;
125   stateStart = millis();
126   setOutputsForState(currentState);
127   logTransition(oldState, currentState, "Истёк таймер");
128 }
129
130 void onAnyEmergencyOn() {
131   State oldState = currentState;
132   currentState = S_WARNING;
133   stateDur = 0xFFFFFFFFUL;
134   stateStart = millis();
135   blinkStart = millis();
136   setOutputsForState(currentState);
137   logTransition(oldState, currentState, "Аварийный режим ВКЛ");
138 }
139
140 void onEmergencyOff() {
141   State oldState = currentState;
142   currentState = S_RED;
143   stateDur = 10000;
144   stateStart = millis();
145   blinkStart = 0;
146   setOutputsForState(currentState);
147   logTransition(oldState, currentState, "Аварийный режим ВЫКЛ");
148 }
149
150 void onNightToggle() {
151   State oldState = currentState;
152   currentState = S_WARNING;
153   stateDur = 0xFFFFFFFFUL;
154   stateStart = millis();
155   blinkStart = millis();
156   setOutputsForState(currentState);
157   logTransition(oldState, currentState, "Ночной режим ВКЛ");
158 }
159
160 void onNightToggleOff() {
161   State oldState = currentState;
162   currentState = S_RED;
163   stateDur = 10000;
164   stateStart = millis();
165   blinkStart = 0;
166   setOutputsForState(currentState);
167   logTransition(oldState, currentState, "Ночной режим ВЫКЛ");
168 }
169
170 // --- Таблица переходов ---
171 void setupTable() {
172   for (int s=0; s<NUM_STATES; ++s)
```

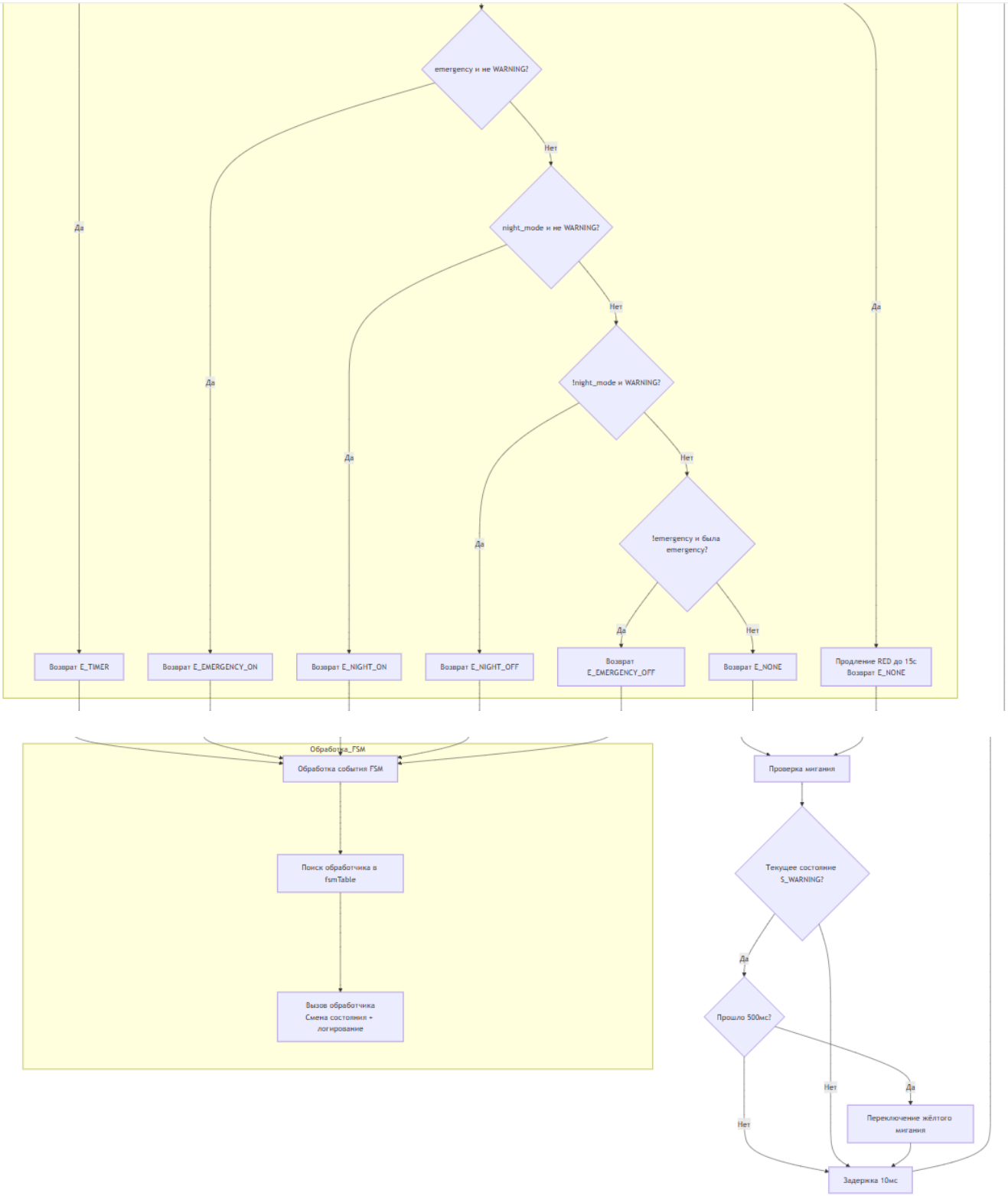
```
    for (int e=0; e<NUM_EVENTS; ++e)
      fsmTable[s][e] = NULL;

  fsmTable[S_GREEN][E_TIMER] = onGreenTimer;
  fsmTable[S_YELLOW][E_TIMER] = onYellowTimer;
  fsmTable[S_RED][E_TIMER] = onRedTimer;

  for (int s=0; s<NUM_STATES; ++s) {
    fsmTable[s][E_EMERGENCY_ON] = onAnyEmergencyOn;
    fsmTable[s][E_EMERGENCY_OFF] = onEmergencyOff;
  }

  // Ночной режим
  for (int s=0; s<NUM_STATES; ++s)
    fsmTable[s][E_NIGHT_ON] = onNightToggle;

  fsmTable[S_WARNING][E_NIGHT_OFF] = onNightToggleOff;
}

// --- Чтение кнопок ---
void readInputs() {
  // Кнопка пешехода / ночной режим
  if (millis() - buttonTmr >= BTN_DEB) {
    buttonTmr = millis();

    static bool buttonPState = false;
    bool buttonState = !digitalRead(BUTTON_PIN);

    if (buttonPState != buttonState) {
      buttonPState = buttonState;

      if (buttonState) {
        if (timerPressStart == 0)
          timerPressStart = millis();
      } else {
        if (timerPressStart != 0) {
          uint32_t holdTime = millis() - timerPressStart;
          if (holdTime >= NIGHT_TOGGLE_HOLD_TIME) {
            night_mode_toggle = !night_mode_toggle;
            Serial.print("[");
            Serial.print(millis());
            Serial.print("мс] Ночной режим: ");
            Serial.println(night_mode_toggle ? "ВКЛ" : "ВЫКЛ");
          } else {
            // короткое нажатие: устанавливаем флаг pedRequest, не переключ
               аем напрямую
            pedRequest = true;
            Serial.print("[");
            Serial.print(millis());
            Serial.println("мс] Кнопка пешехода нажата");
          }
          timerPressStart = 0;
        }
```

```
225       }
226     }
227   }
228
229   // Кнопка аварийного режима
230   if (millis() - emergencyTmr >= BTN_DEB) {
231     emergencyTmr = millis();
232
233     static bool emergencyPState = false;
234     bool emergencyState = !digitalRead(EMERGENCY_PIN);
235
236     if (emergencyPState != emergencyState) {
237       emergencyPState = emergencyState;
238
239       if (emergencyState) {
240         // переключаем флаг аварии
241         emergency = !emergency;
242         Serial.print("[");
243         Serial.print(millis());
244         Serial.print("мс] Авария: ");
245         Serial.println(emergency ? "ВКЛ" : "ВЫКЛ");
246       }
247     }
248   }
249 }
250
251 // --- Определение события ---
252 Event pollEvent() {
253   bool prevEmergency = emergency; // состояние аварии до опроса входов
254   readInputs();                   // может поменять emergency
255
256   if (currentState != S_WARNING && millis() - stateStart >= stateDur) {
257     return E_TIMER;
258   }
259
260   // Если короткое нажатие пришло во время RED - продлеваем RED один раз (и
        сбрасываем флаг)
261   if (pedRequest && currentState == S_RED) {
262     pedRequest = false;
263     stateStart = millis();
264     stateDur = 15000; // удлиняем RED до 15s с момента нажатия
265     Serial.print("[");
266     Serial.print(millis());
267     Serial.println("мс] Запрос пешехода: увеличиваем КРАСНЫЙ до 15000мс");
268     return E_NONE; // обработали, без изменения состояния
269   }
270
271
272   if (pedRequest && currentState == S_GREEN) {
273   }
274
275   if (emergency && currentState != S_WARNING) {
276     return E_EMERGENCY_ON;
```

```
277   }
278
279   // Сначала - проверяем ночной режим (вход/выход)
280   if (night_mode_toggle && currentState != S_WARNING) {
281     return E_NIGHT_ON;
282   }
283
284   if (!night_mode_toggle && currentState == S_WARNING && !emergency) {
285     return E_NIGHT_OFF;
286   }
287
288   // Если авария была включена и только что выключена-  это E_EMERGENCY_OFF
289   if (!emergency && prevEmergency && currentState == S_WARNING) {
290     return E_EMERGENCY_OFF;
291   }
292
293   return E_NONE;
294 }
295
296 // --- Setup ---
297 void setup() {
298   pinMode(RED_PIN, OUTPUT);
299   pinMode(YELLOW_PIN, OUTPUT);
300   pinMode(GREEN_PIN, OUTPUT);
301   pinMode(BUTTON_PIN, INPUT_PULLUP);
302   pinMode(EMERGENCY_PIN, INPUT_PULLUP);
303
304   Serial.begin(9600);
305   setupTable();
306
307   currentState = S_GREEN;
308   stateDur = 10000;
309   stateStart = millis();
310   setOutputsForState(currentState);
311
312   Serial.println("=== СВЕТОФОР ВКЛЮЧЕН ===");
313   Serial.println("[0ms] Стартовое состояние: ЗЕЛЁНЫЙ");
314 }
315
316 // --- Loop ---
317 void loop() {
318   Event ev = pollEvent();
319   if (ev != E_NONE) {
320     Handler h = fsmTable[currentState][ev];
321     if (h) h();
322   }
323
324   // Мигание в ночном/аварийном режиме
325   if (currentState == S_WARNING) {
326     if (millis() - blinkStart >= BLINK_INTERVAL) {
327       digitalWrite(YELLOW_PIN, !digitalRead(YELLOW_PIN));
328       blinkStart = millis();
329     }
```

```
330     }
331
332     delay(10);
333 }
```





Вывод:

В ходе практической работы были реализованы две версии программы светофора на Arduino.

1. Switch-case версия:

Использует простую структуру ‘switch-case‘ для переходов между состояниями. Реализованы таймеры, ночной и аварийный режимы, пешеходная кнопка и логи-

рование. Подход понятен и удобен для простых систем, но слабо масштабируется при росте числа состояний.

2. FSM (табличная) версия:
Логика построена на конечном автомате с таблицей переходов. Такой подход модульнее, проще в расширении и сопровождается тем же логированием.

Обе версии выполняют одинаковые функции, но подход с FSM более гибкий и профессиональный, тогда как switch-case проще для начального изучения принципов автоматов.

Вопросы к зачету по лабораторной работе

1. **Какова основная разница между классическим FSM (switch-case) и table-driven FSM?**

   - switch-case: состояния и переходы прописаны императивно в коде. Понятно и быстро для небольших автоматов, но при росте состояний/событий код разрастается и становится трудно расширяемым.
   - table-driven FSM: есть таблица (матрица) state × event → handler/nextState, иногда с указанием длительности. Более декларативно, масштабируемо и легче тестируется/расширяется (новые переходы — правка таблицы, не логики).

2. **Почему в реальном времени следует избегать длительных delay() в loop()?**

   `delay() блокирует выполнение всего кода на указанное время:`
   `- Пропускаются события (нажатия кнопок, прерывания)`
   `- Не обрабатываются изменения состояний`
   `- Система становится неотзывчивой`
   `- Невозможна параллельная обработка нескольких задач`

3. **Объясните принцип работы debounce и почему он нужен для кнопки.**

   Механизм отсеивает «шум» механического переключателя: при физическом замыкании контактов контакт шатается миллисекунды и даёт несколько скачков сигнала. Debounce — это либо аппаратная RC-цепочка, либо ПО: при изменении состояния ждём DEBOUNCE_DELAY (например 20–50 ms) и подтверждаем состояние только если оно стабильно. Это нужно, чтобы одна

реальная нажатие не превратилось в множество ложных событий.

4. **Как реализован пешеходный запрос (pedestrian request) в обоих вариантах?**
   switch-case версия: краткое нажатие устанавливает флаг pedRequest. При переходе YELLOW -> RED код проверяет pedRequest и, если установлен, устанавливает увеличенную длительность RED (например 15000 ms) и сбрасывает флаг.

   table-driven версия (ранние варианты): логика похожая — флаг ставится в readInputs(), событие E_PED могло бы генерироваться и вызывать переходы, но в итоге реализовали вариант, где E_PED не всегда генерируется немедленно: флаг учитывается при переходах таблицы (на YELLOW→RED).

5. **Почему мы используем millis() вместо таймеров delay()? Назовите преимущества.**

   Преимущества millis():
   1. Неблокирующее выполнение
   2. Возможность обработки multiple events
   3. Точное управление временем
   4. Возможность прерывания операций
   5. Лучшая отзывчивость системы
   6. Поддержка сложных временных логик

6. **Что происходит, если при активном EMERGENCY входе поступает пешеходный запрос?**
   При активном emergency система переходит в S_WARNING (мигающий режим). В наших реализациях pedRequest сохраняется, но не выполняется пока emergency/night_mode не выключены. То есть запрос запоминается, но не обслуживается немедленно. Это корректно с точки зрения безопасности — при аварии пешеходный цикл обычно игнорируется.

7. **Как реализован ночной режим и чем он отличается от аварийного?**
   Реализация: удержание пешеходной кнопки 5000 ms переключает флаг night_mode_toggle, что переводит систему в S_WARNING (мигающий жёлтый). Выход — повторное удержание (или отключение флага).

Отличие: семантически оба дают мигающий жёлтый, но аварийный (emergency) — включается отдельной кнопкой и может иметь приоритет над ночным; ночной — пользовательский/ручной режим для тихого времени. В коде они объединены по поведению (оба — S_WARNING), но логика включения/выключения и приоритеты — разные.

8. **Как должна вести себя система при заполненной очереди событий (в таблице)?**

   Мы не использовали явную очередь; использовали флаги и приоритетную проверку в pollEvent(). Если очередь есть и переполняется — следует: отбрасывать старые/наименее важные события, использовать приоритеты (EMERGENCY > NIGHT > TIMER > PED), или ограничивать длину очереди с политикой drop-oldest или drop-newest. Для realtime — лучше иметь приоритетную очередь и не накапливать события, требующие немедленного действия.

9. **Какие состояния у вас есть — и какие события их переводят в S_-WARNING?**

   Состояния: S_GREEN, S_YELLOW, S_RED, S_WARNING.
   В S_WARNING переводят события: EMERGENCY_ON (нажатие аварийной кнопки) и NIGHT_ON (удержание кнопки для ночного режима). Реализации объединяют эти события в проверку if (emergency || night_mode_toggle).

10. **Какая точность времени ожидается и от чего она зависит?**

    Ожидание миллисекундной точности, но:
    точность зависит от millis() — погрешность ± 1 ms в норме, но millis() имеет джиттер при прерываниях/delay;
    использование delay() и Serial.print в loop увеличивает джиттер;

11. **Покажите, где и как вы предотвращаете гонки (race conditions) при доступе к флагам (pedRequest, emergency).**
    В нашем однопоточном Arduino-скетче гонки в классическом смысле редки, потому что есть один поток исполнения (loop). Возможные проблемы — асинхронный доступ из прерываний; мы не используем прерывания для кнопок, поэтому флаги обновляются только в readInputs() и читаются в основном цикле — это безопасно.

12. **Объясните, как восстановление после EMERGENCY реализовано — вернётся ли система в то же место цикла? Какие компромиссы есть?**
    В реализации: при выключении emergency система переходит в S_RED с фиксированной длительностью (например 10000 ms). То есть не возвращается в точную позицию цикла, где была ранее — это упрощение и безопасный компромисс: после аварии даётся предсказуемый красный интервал, чтобы восстановить поток. Компромисс: потеря точной последовательности и возможные задержки для машин/пешеходов, но выигрываем в предсказуемости и простоте.

13. **Как улучшить таблицу так, чтобы handlers были декларативными (handler + nextState + duration) и почему это полезно?**
    Сделать структуру:

```
struct Transition { Handler handler; State next; unsigned long duration
   ; };
Transition table[NUM_STATES][NUM_EVENTS];
```

    Тогда таблица не только содержит функцию, но и объявляет конечное состояние и длительность. Польза: переходы становятся декларативными — легко читать/генерировать/тестировать таблицу, добавлять логирование (общий код делает goToState(next,duration,reason)), уменьшить дублирование в обработчиках.

14. **Как расширить реализацию до двух направлений с контролем конфликтов (mutual exclusion)? Опишите логику.**

    Чтобы добавить два направления движения, нужно ввести два независимых светофора, но сделать так, чтобы они не могли гореть зелёным одновременно.

    Для этого можно:

    - Добавить отдельные состояния для каждой стороны, например: `S_NS_GREEN`, `S_NS_YELLOW`, `S_EW_GREEN`, `S_EW_YELLOW`.
    - В таблице переходов (или в конструкции `switch-case`) при переключении одного направления на жёлтый/красный — разрешать другому становиться зелёным.
    - Добавить флаг занятости (например, `activeDirection`), чтобы система знала, какое направление сейчас активно, и не включала второе, пока первое не завершит свой цикл.

- При поступлении пешеходного запроса или аварийного сигнала оба направления переходят в безопасное состояние — все красные.

Таким образом, система будет поочерёдно обслуживать направления, не создавая конфликтов между потоками движения.

15. **Если очередь событий переполняется, какие стратегии обработки событий вы предложите?**
- Удалить старые неактуальные события.
- Сохраняем старые, отклоняем лишние.
- Оставляем только высокоприоритетные (EMERGENCY) и отбрасываем пешеходные/таймерные.
- Объединять похожие события (несколько PED → 1 PED).
Выбор зависит от требований безопасности и важности событий.

16. **Предложите тесты для проверки корректности обработки long-press vs short-press.**

- Короткое нажатие (<5 с) → флаг `pedRequest`.
- Долгое нажатие (5 с) → переключение `night_mode`.
- Граничный случай: ровно 5000 мс.
- Быстрые повторные клики — проверка debounce.

17. **Какие метрики/логирование вы бы добавили для production-мониторинга светофора?**

- События переходов (`state transitions`).
- Кол-во запросов пешехода, активаций аварии и ночного режима.
- Время в каждом состоянии.
- Ошибки и задержки реакции.

18. **Как вы бы реализовали адаптивное время зелёного на основе потока машин (датчик/ультразвук)?**

Считать поток (маш/мин) с датчика(ов). Реализовать правило: green = base + k * vehicle_count, с ограничением min/max. Можно применять скользящее среднее для сглаживания и пороги для предотвращения частых изменений. Обновлять время перед началом следующего зелёного цикла.

19. **Какие проблемы может дать использование Serial.print в каждой итерации loop() и как уменьшить нагрузку?**
Serial.print медленный — блокирует выполнение, увеличивает джиттер и нарушает тайминги.
Решения: логировать только при переходах (мы так и делаем), группировать-/буферизовать сообщения, снизить скорость отправки, отправлять метрики раз в N секунд.

20. **Как бы вы добавили unit-тесты для FSM (симуляция времени и событий) на хост-машине?**
Абстрагировать время и ввод: заменить millis() и digitalRead() на интерфейсы, чтобы тесты могли симулировать время и события.
Реализовать FSM-ядро как библиотеку (чистые функции), и в тестах симулировать последовательности событий и проверять состояния/логи.

21. **Светофор застрял в одном состоянии — какие шаги для поиска причины?**

- Посмотреть Serial-логи — есть ли переходы/исключения?
- Проверить флаги (emergency, night_mode_toggle, pedRequest).
- Убедиться, что stateStart/duration корректны.
- Проверить, не держит ли аппаратная кнопка состояние (провода/подключение).

22. **Нажатие кнопки не регистрируется — как локализовать проблему (аппаратура/софт)?**
Аппарат: мультиметром проверить кнопку и проводку, проверить подтяжку к VCC/GND, контакт оконцов, правильность подключения (INPUT_-PULLUP).
Софт: вывести digitalRead() в Serial в loop, добавить LED индикацию состояния кнопки, временно убрать debounce, проверить логи. Если аппарат смотрит правильно (событие видно на digitalRead) — проблема в коде; иначе — в проводке/кнопке.

23. **Мигающий жёлтый неравномерный (дрожит) — какие возможные причины?**

- Частые вызовы `Serial.print()`.
- Падения питания.

- Ошибки логики переключения (`digitalRead()` от старого состояния).

24. **При включении EMERGENCY система не возвращается в норму — что проверить первым делом?**

    - Флаг emergency действительно сменился на false? (Serial лог).
    - Не остался ли night_mode_toggle включённым (он тоже вызывает WARNING).
    - Не застрял ли код в ветке, где goToState(S_WARNING, ...) переинициализируется постоянно.
    - Проверить логи, значения флагов, и последовательность событий.

25. **Как измерить реальные интервалы времени и сравнить с ожидаемыми (инструменты/метод)?**

    - Через `millis()` и разности таймстампов в логах.
    - Внешне — осциллографом или логическим анализатором.
    - Для долгих тестов — запись в SD/файл и сравнение с ожидаемыми значениями.